UNIVERSITY OF VIENNA

Research Internship

Report #4
Experimental Results

Amir Sabzi
+989373107525
97.amirsabzi@gmail.com
April 2021

Table of Contents

List of Figures

1 Evaluation of flow reconfiguration with 2 hosts

1.1 Overview

As I explained in the previous report, when two hosts in the network communicate implicitly based on the RTT of ping packets, the timing behavior of the controller and delays of the links will be extremely important.
In the section 1.2 of the last report, I performed a timing analysis of VM-migration, and here I want to provide a quick review of that section. The main idea is that the receiver will discriminate 0 and 1 based on the RTT of ping packet. Thus, we should have an accurate decision threshold at the receiver side to decide based on that. In order to find this threshold, we need to know the probability distribution of RTT for ping packets after migration and the distribution of them before migration. Then, we can minimize the following error function to find an appropriate threshold:

$$P_e(T) = P(R = 1|S = 0) + P(R = 0|S = 1)$$

where $P(R = r|S = s)$ is the distribution of the RTT of ping packets at the receiver side when the sender sends s. It is clear that the error of the transmission would be defined as a function of the Threshold (T). Based on the experimental results that I demonstrated in the previous report, both

$P(R = r|S = 0)$ and $P(R = r|S = 1)$ will have Gaussian distribution. In the more complicated scenario, we can use the *kernel density estimation* method to estimate the probability density function of the $P(R = r|S = s)$. I demonstrated the mathematical procedure needed to calculate the Threshold (T) in the previous report. I embedded this procedure in the receiver script as a part of the calibration phase which I will describe in the next subsection.

1.2 Calibration Phase

The timing behavior of the networks depends on too many variables including delay of the links, load of the controller, the processing power of hosts, and the distance (geographical or hop-count distance) between the sender and the receiver in the network. Many of these conditions can change when we have a transmission. Consequently, to tune the threshold we should implement a calibration phase. In the calibration phase sender host will transmit a predefined bit string, and the receiver will record the RTT of the ping packets in each round of transmission and use them as samples to calculate statistics of the covert channel. Similar to many electronic devices, I used a square wave (i.e. a long string of consecutive 1 and 0s) as you can see in the Figure 1 to calibrate the receiver. The receiver will consider Round Trip Time of each packet as the amplitude of the wave at each interval. I used a string of 512 ones and zeros in the calibration phase. Then using the normal distribution fitted to the samples of zeros and ones, we can calculate the threshold.

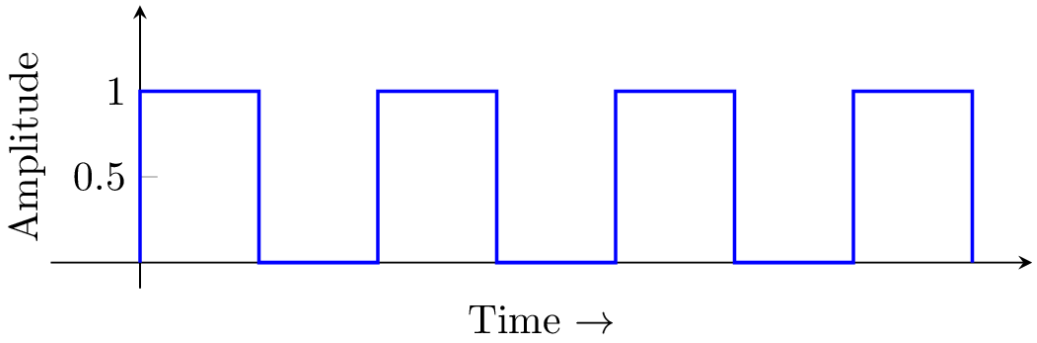


Figure 1: Sample square wave

1.3 Effect of the controller load

As I mentioned in the previous section, there are many factors that can affect the timing behavior of the network. One these factors is instantaneous load on the controller. We expect that when we have heavy load on the controller, the queues become full, and the flow reconfiguration will take more time compared the situation in which there is no load on the controller. To examine this, I defined four different scenarios:

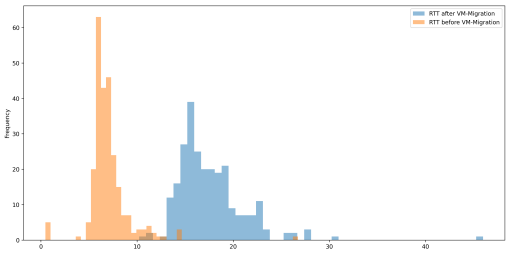
- **No Load**: in this scenario only receiver and sender will transmit data and there is no other load on the controller.
- **Light Load**: in this scenario we have 20 switches connected to controller and each of them will generate a packet-in message and send it to controller at the random intervals with 500ms average.
- **Medium Load**: it is exactly similar to the previous scenario but the

random intervals have an average about 200ms.

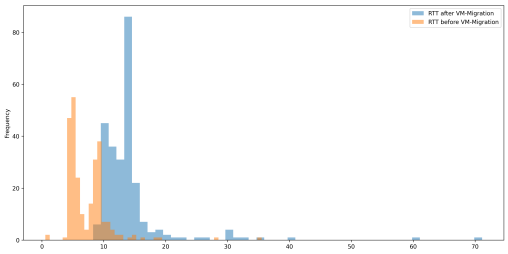
- **Heavy Load**: the same scenario with the random intervals with 50ms average.

I generated these load using ofcprobe tool. I should clarify that the effect of the load on the communication and covert channel parameters depends on the processing power of the control. In the Figure 2 you can see histogram of RTT of the packets before and after migration (i.e. delays that represent zero and one respectively). As you can see there, the more load on the controller, the more intervention between samples we will have. Therefore, the error will decrease when we try to decide based on a single threshold. I should mention that the throughput in this setup is about **7 bit per second**.

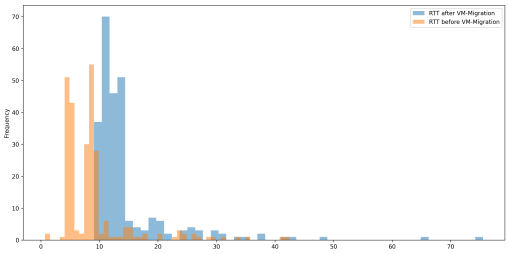
After calculating the threshold, we can evaluate the channel error based on that threshold. I named it as *expected error* because it can give us a sense of the channel overall error. I repeated the transmission 62 times for each scenario, and in the Figure 3 you can see the box plot of calculated threshold and estimated error for four different scenarios. As you can see in the Figure 3, the error will increase with the load almost linearly. On the other hand, threshold does not follow the same pattern. The error for the medium load would be some value between 0.18 to 0.33 which is not acceptable at all. Thus, we need to think about new ideas to increase the accuracy of the channel.



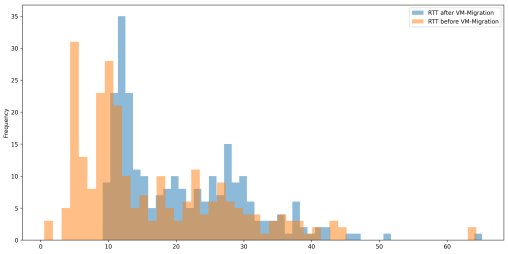
(a) RTT of Packets with no load



(b) RTT of Packets with light load

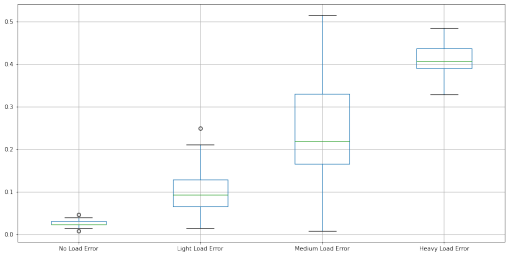


(c) RTT of Packets with medium load

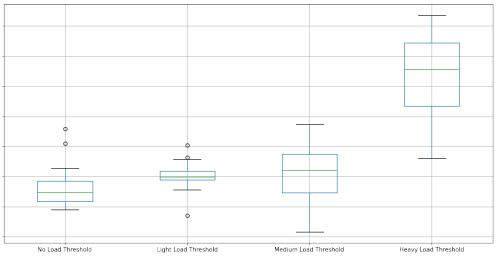


(d) RTT of Packets with heavy load

Figure 2: Histogram of packets round-trip time in different scenarios



(a) Estimated error



(b) Calculated threshold

Figure 3: Box plot of threshold and error for different scenarios

1.4 Edge-Sensitive Modulation and Neural Networks

In this section, I will try to apply some ideas that can decrease the channel error. As I mentioned previously, we need a threshold to decide at the receiver side. When the controller and switches start their service under heavy, after a while their queues will become full of request messages and corresponding responses. Thus, the overall delay for both cases (zeros and ones) will increase, and consequently, the calculated threshold can not discriminate ones and zeros anymore.

To address this issue, we use can use edge-sensitive modulation. It is a well-known method to transmit information in the channel that has DC noise like what we have in our covert channel. Therefore, we can define a transmission from lesser delay to higher delay as one, and we define transmission from higher delay to lesser one as a zero. This method is very similar to the Manchester coding, and as I showed in the Figure 4, it will transmit a single bit in two cycle. Thus, it will reduce the overall throughput by a factor of 2 but it can decrease the channel transmission error significantly.

As I described, it is not the best way to classify the behavior of the network based on a single threshold because average of the delay can change during the communication. Thus, I created a feature space using the delays. In fact each sample can demonstrated as $< y, X >$, here we have $y \in \{0, 1\}$ which is the transmitted bit in a particular round and we will consider it as the label of the sample, and $X = \{x_1, x_2, x_3\}$ where x_1 is the RTT of previous ping

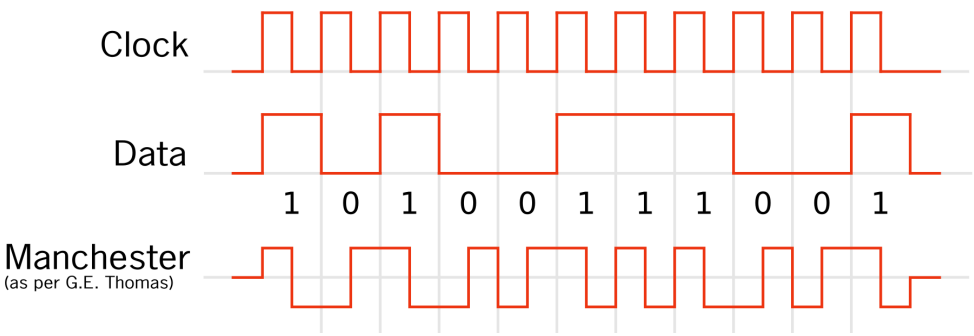


Figure 4: An example of Manchester encoding

packet, x_2 is the RTT of current ping packet, x_2 is the RTT of next ping packet. Thus, we will train our neural network in the calibration phase, and then using it, it is possible to receive bits by classifying the behavior of the network as 1 or 0. As I demonstrated in the Figure 5, I designed two fully connected hidden layer for my network. Each hidden layer can have many neurons, and based on trial and error I found a value more than 100 as an optimal number but here in the figure I chose 10 for the sake of resolution. To avoid overfit, I used dropout method with the probability of 0.1 for two hidden layers. I used *sigmoid* activation function for these layers, and the final layer is a softmax.

I used a bit string with length of 1024 to evaluate and compare these methods. In the table 1 you can see the results. As you can see in this table, the error is relatively smaller when we use edge-sensitive decoding and neural networks but it should be considered that Manchester coding uses to cycles to transmit a bit. Thus, the best method in term of the accuracy and throughput is PCM in encoding stage and then using neural networks to

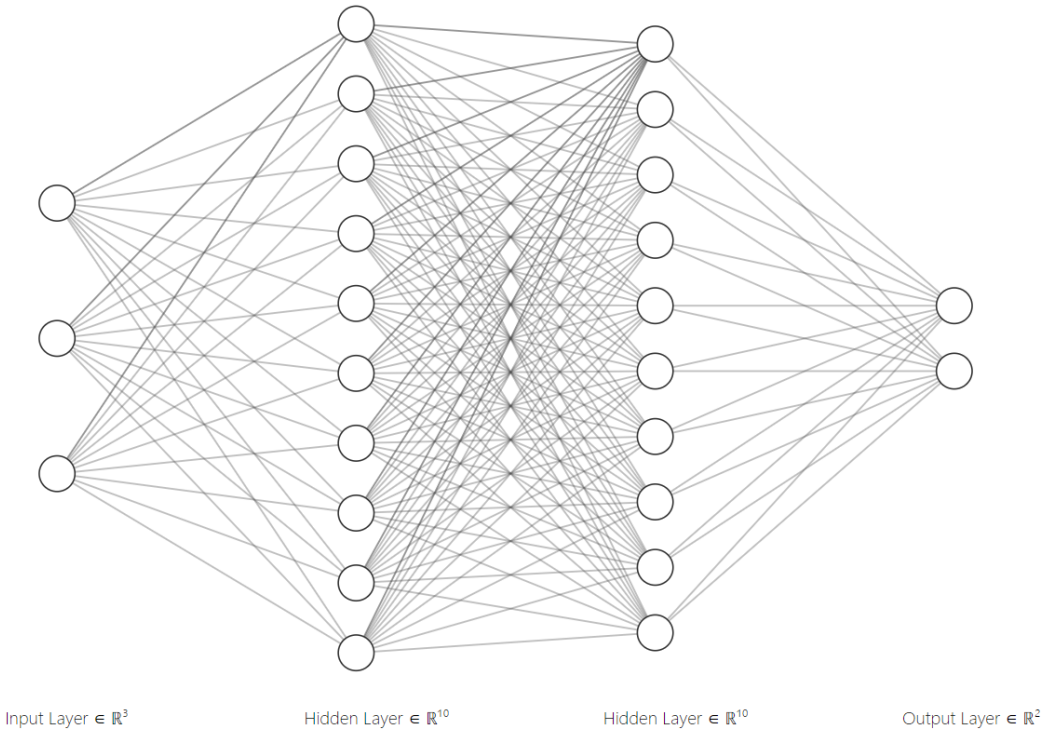


Figure 5: The neural network architecture

detect bits.

Table 1

Encoding Method	**Decoding Mechanism**	**Error rate (No Load)**	**Error rate (Light Load)**	**Error rate (Medium Load)**	**Error rate (Heavy Load)**
PCM	Threshold	0.029748	0.1289	0.18752	0.42745
Manchester	Edge-Sensitive	0.00457	0.04687	0.12180	0.27843
PCM	Neural Network	0.0114	0.03921	0.0890	0.2254

2 parallelized Flow reconfiguration

As I mentioned in the previous section, currently, the throughput of flow reconfiguration method with two host is about 7 bps, which is not dangerously high in real-world scenario. Therefore, it is extremely important to increase the throughput of the channel. The easiest way is to enhance the throughput is reducing the transmission round duration but it definitely will cause high error rate.

In today's data centers, many hosts can have multiple interfaces connected to the same switch. We can exploit this by running multiple threads on each host, each of them transmit a part of the main bit string. As a result, we can parallelize the communication and increase the throughput. As you can see in the figure 6, the receiver, *h1*, and the sender, *h3*, have 4 interfaces. Each of *h3* interfaces transmit data to its corresponding interface in *h1* using the flow reconfiguration with that particular Ethernet address.

I wrote the a few modular python scripts which only take number of interfaces I created desired topology and transmit data in parallel. I tested this codes in our test bed at the university of Vienna servers. In our setup, it was possible to add up to 11 interfaces to each host, and these interfaces started to send data covertly. Therefore, I could increase the throughput to about **70 bit per second** without a considerable effect on the error rate of the channel.

It is possible to use all the ideas presented in previous section to reduce the

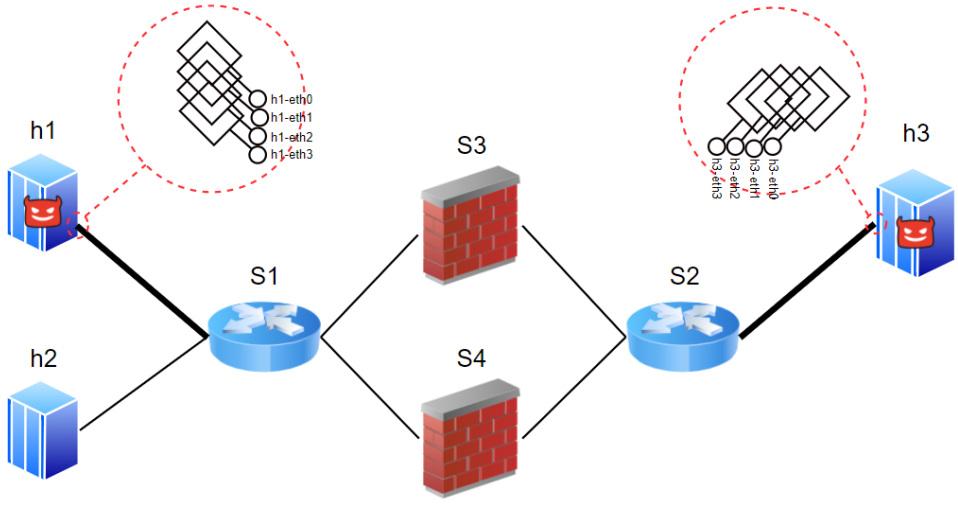


Figure 6: Network topology for the parallelization of flow reconfiguration

channel error rate when there is a load on the controller. Although, I have not performed some of those evaluations for parallelized flow reconfiguration so far, it can be considered for next stages of the project.