End-to-End Detection of Compression of Traffic Flows by Intermediaries

Ashwini Badgujar (abadgujar@dons.usfca.edu)
Dharti Madeka (dpmadeka@dons.usfca.edu)
Kedar Khetia (kmkhetia@dons.usfca.edu)

Introduction

This document details the implementation of technique to detect the compression of traffic flows by intermediaries. The technique uses difference between the arrival time of the first and the last packet in the train of packets. It does not require synchronized clocks at the sender and receiver. We have validated our approach by sending two sets of 6000 UDP packets back-to-back (called packet train) and then calculating the difference between arrival time of the first and the last packet. Wherein if the difference in arrival time of the first and last packets of the two trains is more than a fixed threshold (100 ms), the application reports Compression detected otherwise the application reports No compression detected. Our findings demonstrate an accurate detection of compression applied to traffic flows by intermediaries.

NS-3 Classes Modified in the actual design

We have modified following classes in the ns-3 architecture for the following reasons:

1. **UdpClient**
   (a) **m_setEntrophy**
   - m_setEntrophy is a private variable with boolean type which is set to generate high and low entropy of data.

   (b) **createLowEntropy**
   - createLowEntropy is a public function used to randomly generate low entropy data (data with 0s).

   (c) **createHighEntropy**
   - createHighEntropy is a public function used to randomly generate high entropy data (data with random 0s and 1s).

   (d) **SetEntropyValue**
   - SetEntropyValue is a public function used to set the low and high entropy data as per the given entropyValue which is a boolean passed as a parameter.

2. **PointToPointNetDevice**
   (a) **m_protocol**
   - m_protocol is a public variable which sets protocol number for the compression and decompression.

   (b) **EnableCompression**
   - EnableCompression is a public function added to PointToPointNetDevice class to enable compression between intermediaries.

   (c) **EnableDeCompression**
   - EnableDeCompression, a public function which is newly added to PointToPointNetDevice class enables decompression between intermediaries .

   (d) **Compress**
   - Compress, a private function is added to the PointToPointNetDevice class which compresses the packet given to it as parameter and returns compressed packet.

   (e) **UnCompress**
   - DeCompress, a private function is added to the PointToPointNetDevice class which decompresses the packet given to it as parameter and returns decompressed packet.

   (f) **compressionEnabled**
   - compressionEnabled, a private boolean variable is used to enable the compression.

   (g) **deCompressionEnabled**
   - deCompressionEnabled is a private boolean variable used to enable the decompression.

3. **UDPServer**
   (a) **getTimeDiff**
   - getTimeDiff is a public method that gives time difference between first packet and last packet.

   (b) **diff**

- diff is a public time variable which stores the difference between the arrival time between the first and the last packet.

(c) **firstPacket**

- firstPacket is a private time variable which stores the arrival time of the first packet.

(d) **lastPacket**

- lastPacket is a private time variable which stores the arrival time of the last packet.

Issues, Challenges and Limitations

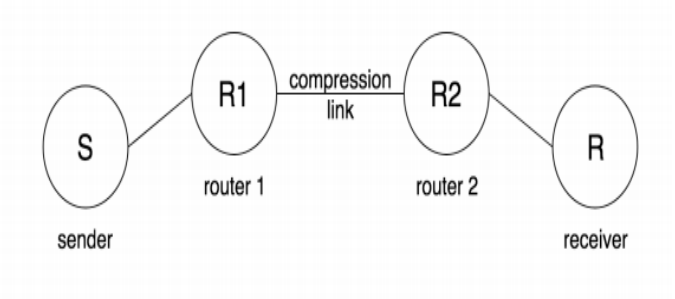
1. **Config Files**
   - ns-3 has its own way of defining and handling config files.
   - Understanding handling of config file in ns-3 is a bit tricky and thus took time.
2. **zlib configuration**
   - There are various ways of configuring zlib.
   - One of them is modifying wscripts in order to configure zlib.
   - Another way is using command `sudo apt install zlib`
   - Figuring out which way works (since one way didn't way for all the team team members) for the installation was time cosuming.
3. **ns-3 environment**
   - ns-3 environment setup was tiresome since some of the test cases failed for the first installation and thus installation was done for number of times in order to get a successful setup.

Implementation Details

- The implementation is as follows:
- We take a UDP packet.
- The m_protocol value which is taken as input in the config file is matched with the protocol number in the UDP packet header.
- If the protocol number matches, we preprocess the packet.
- For this project, we use only one IP protocol entered with protocol number 0x0021.
- Pre-processing takes the matched packet and replaces the original protocol number with the LZS protocol number. It then appends original protocol to the original data.
- After pre-processing, the data from the packet is compressed and put in a new packet.
- This new packet is then sent to the sender.

Simulation Validation and Verification

- Using ns-3, we created a 4-node topology where nodes S and R are the end-hosts running the network application.
- Nodes R1 and R2 are the intermediate routers where the link between them is compression-enabled.
- The topology is as follows :

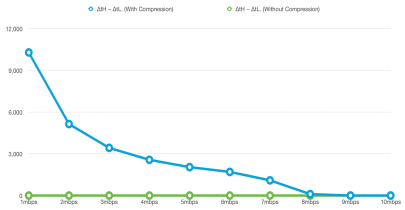


- Two sets of 6000 UDP packets (called packet train) are sent back-to-back from sender to receiver.
- We calculate the difference between arrival time of the first and the last packet.
- If the difference in arrival time of the first and last packets of the two trains is more than a fixed threshold (100 ms), the application reports Compression detected otherwise the application reports No compression detected.
- Following is the screenshot of the results that we achieved :

This is a graph of data rate and difference between high and low entropy of the system.

ΔtH – ΔtL

DATA RATE	DIFFERENCE(WITH COMPRESSION)	LOW ENTROPY	HIGH ENTROPY	DIFFERENCE (WITHOUT COMPRESSION)	LOW ENTROPY	HIGH ENTROPY
1mbps	10,296	6,922	17,218	0	17,069	17,069
2mbps	5,144	6,922	12,066	0	11,996	11,996
3mbps	3,430	6,922	10,352	0	10,305	10,305
4mbps	2,570	6,922	9,492	0	9,460	9,460
5mbps	2,045	6,922	8,967	0	8,953	8,953
6mbps	1,700	6,922	8,622	0	8,614	8,614
7mbps	1,094	6,922	8,016	0	7,911	7,911
8mbps	102	6,922	7,024	0	6,922	6,922
9mbps	0	6,922	6,922	0	6,922	6,922
10mbps	0	6,922	6,922	0	6,922	6,922



- In the above figure, the blue line shows the difference between the arrival time of the first and last packet when the compression is enabled and green line defines the difference between the arrival time between the first and the last packet.
- When the compression is disabled, there is no difference in the arrival time between the packets, which can be seen through green line.
- When the compression is enabled(blue line), the difference in the arrival time between the packets gets reduced as the data rate between the links increases.

API Usage

- User must know the details and usage of config file. Config file is in text format which provides the m_protocol value.
- User must use following command for the execution of the application : `./waf --command-template="%s --ns3::ConfigStore::Filename=config.txt --ns3::ConfigStore::Mode=Save -ns3::ConfigStore::FileFormat=RawText --capacity=8Mbps" --run CompresionLink`

Alternative Design

- Following is the alternative design that we propose :
- Rather than removing data from old packet and creating a new packet with compressed data, the compressed data can be put into the same packet.
- Therefore, the preprocessing of removing the data from old packet and putting compressed data into new packet will be omitted and same old packet with new compressed data will be sent to the sender.

References

1. library used for compression and decompression
   - zlib
2. research paper for reference in implementation of Compression and Decompression
   - End-to-End Detection of Compression of Traffic Flows by Intermediaries