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

Technical Manual

Ashwini Badgujar, Dharti Madeka, Kedar Khetia

March 18, 2019

## 1 Introduction

As per the research paper, we present a probing technique to detect the compression of traffic flows by intermediaries. Our technique is non-intrusive and robust to cross traffic. It is entirely end-to-end, requiring neither changes to nor information from intermediate nodes. We present two different but similar approaches based on how cooperative the end-hosts are. Our proposed technique only uses packet inter-arrival times for detection. It does not require synchronized clocks at the sender and receiver. Simulations and Internet experiments were used to evaluate our approach. Our findings demonstrate an accurate detection of compression applied to traffic flows by intermediaries.

ns-3 has been developed to provide an open, extensible network simulation platform, for networking research and education. In brief, ns-3 provides models of how packet data networks work and perform, and provides a simulation engine for users to conduct simulation experiments. Some of the reasons to use ns-3 include to perform studies that are more difficult or not possible to perform with real systems, to study system behavior in a highly controlled, reproducible environment, and to learn about how networks work.

## 2 Using ns-3 for End-to-End Detection of Compression of Traffic Flows by Intermediaries

Following classes are modified in the ns-3 framework :

```
UdpClient
```

UdpClient is used to send udp packets to the server. We use variables like PacketSize, MaxPackets to set the properties of the UDP packet.

We define a new variable setEntropy to set the boolean for setting high and low entropy. We create low entropy and high entropy data in this class.

A function to state how the entropy variable will be set and work in defined in this class.

The header on which this class is dependent is udp client header which provides all client side functionality for a UDP client.

```
UdpServer
```

We provide all the server side functionalities in this class. The time calculation for the arrival of first and last packet is recorded here and the difference between them is calculated in this class.

This class gives us the overall idea of how the udp server in our application works.

Udp Server reciprocates to the requests and creates the output of time difference in this class.

```
Point to Point Net Device
```

We add the functionalities for command line argument and configuration file in this class. The variable used as capacity for the data rate for the routers is declared and defined in this class.

## 3 Main Class

We have a main class (Compression Link) which is used to build the topology. Following are the functionalities provided by the main class:

### 3.1 Creating Nodes

We create 4 nodes (2 routers, 1 sender and a receiver) in this class. We create links in the nodes and we convert them to point to point net devices.

### 3.2 Topology

We create the topology , start get all the parameters and inputs (configuration file and command line arguments) in this class.

The simulator starts and stops in this class. Since this class becomes the main class to start and stop the application.

## 4 ns3 in implementation

We have used the architecture provided by the ns3 framework which gives the idea of simulation and how it works in the real world networking.

```
Technical Aspects
```

## 5 Libraries

```
zlib
```

zlib is designed to be a free, general-purpose, legally unencumbered – that is, not covered by any patents – lossless data-compression library for use on virtually any computer hardware and operating system. The zlib data format is itself portable across platforms. Unlike the LZW compression method used in Unix compress(1) and in the GIF image format, the compression method currently used in zlib essentially never expands the data. (LZW can double or triple the file size in extreme cases.) zlib's memory footprint is also independent of the input data and can be reduced, if necessary, at some cost in compression. A more precise, technical discussion of both points is available on another page.

## 6 Implementation

### 6.1 Compression Link

We have implemented functionality that takes a PPP packet, and first checks the protocol number in the header. We have a configuration file that specifies packet types to compress. If we determine that the checked packet type matches one in the configuration file, we pre-process then compress it. For the purposes of this project, our configuration file has one protocol entered, IP, protocol number 0x0021. Pre-processing should first take the matched packet then replace the original protocol number with the LZS protocol number, 0x4021. It should then take the original protocol number and append it to the original data, and compress that whole bitstring and replace it with the original data section in the original packet, as illustrated in Figure 1. Decompressor, at the other side of the compression link, should then reverse all the pre-processing steps performed at the compressor, to retrieve the original incoming packet before pushing it to the next interface.

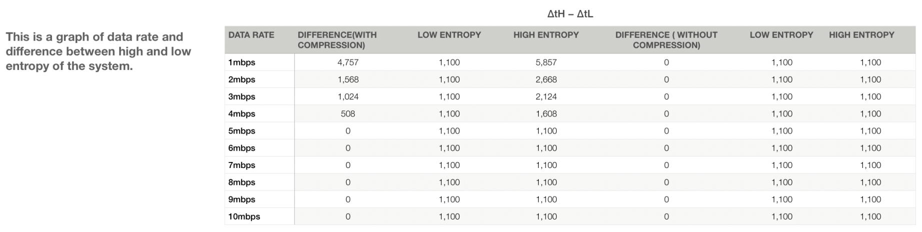
### 6.2 Compression Detection Application

Our network application is a client/server application where the sender sends two sets of 6000 UDP packets back-to-back (called packet train), and the receiver records the arrival time between the first and last packet in the train. The first packet train consists of all packets of size 1100 bytes in payload, filled with all 0's, while the second packet train contains random sequence of bits. You can generate random sequence of bits using random. If the difference in arrival time between the first and last packets of the two trains is more than a fixed threshold = 100 ms, the application reports Compression detected!, whereas when the time difference is less than  there was probably no compression link on the path and the application should display No compression was detected.

## 6.3 Simulation Verification and Validation

We created a 4-node topology in ns-3 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. Our simulations are built using ns-3.

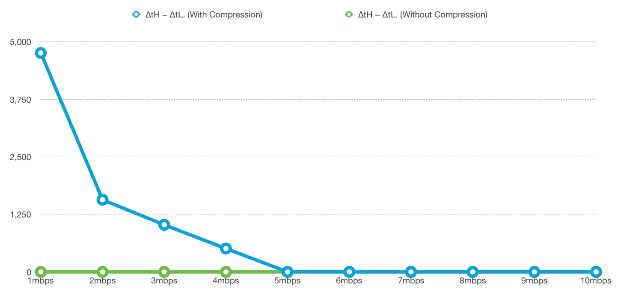
## 7 Testing



**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 | 4,757 | 1,100 | 5,857 | 0 | 1,100 | 1,100 |
| 2mbps | 1,568 | 1,100 | 2,668 | 0 | 1,100 | 1,100 |
| 3mbps | 1,024 | 1,100 | 2,124 | 0 | 1,100 | 1,100 |
| 4mbps | 508 | 1,100 | 1,608 | 0 | 1,100 | 1,100 |
| 5mbps | 0 | 1,100 | 1,100 | 0 | 1,100 | 1,100 |
| 6mbps | 0 | 1,100 | 1,100 | 0 | 1,100 | 1,100 |
| 7mbps | 0 | 1,100 | 1,100 | 0 | 1,100 | 1,100 |
| 8mbps | 0 | 1,100 | 1,100 | 0 | 1,100 | 1,100 |
| 9mbps | 0 | 1,100 | 1,100 | 0 | 1,100 | 1,100 |
| 10mbps | 0 | 1,100 | 1,100 | 0 | 1,100 | 1,100 |


ΔtH – ΔtL. (With Compression)
ΔtH – ΔtL. (Without Compression)
1mbps
2mbps
3mbps
4mbps
5mbps
6mbps
7mbps
8mbps
9mbps
10mbps