Distributed Computer Systems Lab
http://disco.informatik.uni-kl.de

Technische Universität
Kaiserslautern

Performance Evaluation of Distributed Systems

Khojema Vora & Subin Joseph

Agenda

1. Introduction
2. Lossless Compression Algorithms
3. System Specifications
4. Maximum Compression Ratio
5. Chunk Size Vs. Compression Ratio & Profiling
6. Experimental Setup
7. Stream Compression System Design
8. Performance Evaluation
9. Conclusion

Introduction

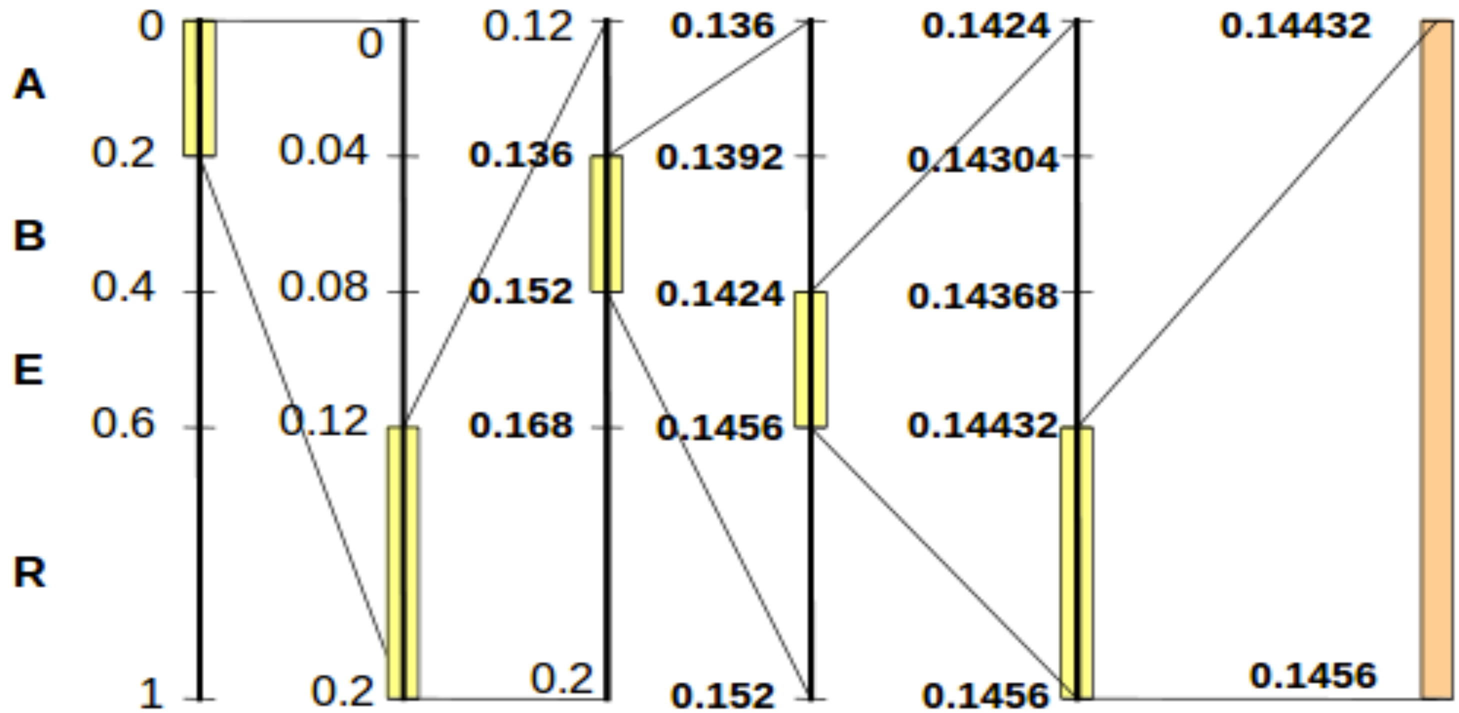
- **Main Goal:** To reduce the network load without loss of packets
- **Solution**: Lossless Compression
  - Two kinds of lossless compression
  - **Block Compression**: All the input data is compressed or decompressed by one call to the compression or decompression function.
  - **Stream Compression**: Compression or decompression function is called repeatedly to compress or decompress data from a source buffer to a destination buffer
  - **Here implementation is based on Stream compression**
- **Challenges:**
  - Air traffic data should be delivered in real-time.
  - Compression method is limited by CPU processing time
- Find a reasonable trade-off between compression ratio and CPU processing time.

Lossless Compression Algorithms

- **Arithmetic coding**
- **The Idea:** Encodes data (the data string) by creating a code string which represents a fractional value on the number line between 0 and 1.
- Create an interval [low, high) for each symbol, based on cumulative probabilities.
- Scale the remaining intervals:
  - New Low = Current Low + *Sumn-1(p)*(H–L)*
  - New High = Current High + Sum*n(p)**(H –L)
- Consider the string **ARBER**

Symbol	A	B	E	R
Low	0	0.2	0.4	0.6
High	0.2	0.4	0.6	1

Arithmetic Coding



- First take symbol A and subdivide the interval A(0-0.2) in the same as we subdivided the original chunk.
- The final interval for the input string ARBER is [0.14432, 0.1456).

Arithmetic Coding

- For the sample interval, one may choose point 0.14432,which in binary is: 0 0 1 0 0 1 0 0 1 1 1 1 0 0 1 0 0 0 1 0 0 1 1 1 1 1 0 1 0 0 0 0 0 0 1 0 1 0 0 0 1 0 1 0 0 0 0 1 1 1 1 (51Bits)
- Decoding is straightforward: Start with the last interval and divide intervals proportionally to symbol probabilities. Proceed until and **END-OF-STREAM** control sequence is reached

LZW Algorithm (Lempel-Ziv-Welch)

- **The Idea:** Rely on reoccurring patterns to save data space and a dictionary based Algorithm
- LZW works by encoding strings. Some strings are replaced by a single code word (assume code word is fixed- 12 bits)
- For 8 bit characters, first 256 (or less) entries in table are reserved for the characters
- Rest of table (257-4096) represent strings
- Consider this string : thisisthe
- By default uncompressed ASCII is equal to

01110100 01101000 01101001 01110011 01101001 01110011 01110100 01101000 01100101 or 116 104 105 115 105 115 116 104 101

LZW Algorithm (Lempel-Ziv-Welch)

Current	Next	Output	Add to dictionary
t116	h104	t116	th256
h104	i105	h104	hi257
i105	s115	i105	is258
s115	i105	s115	si259
i105	s115	is is in dictionary	Check if “ist” is
is258	t116	is258	ist260
t116	h104	th is in dictionary	Check if “the” is
th256	e101	th256	the260
e101		e101	

- Output will be: 116 104 105 115 258 256 101
- The encoded string is 7bytes compared to 9 bytes in uncompressed string

System Specifications

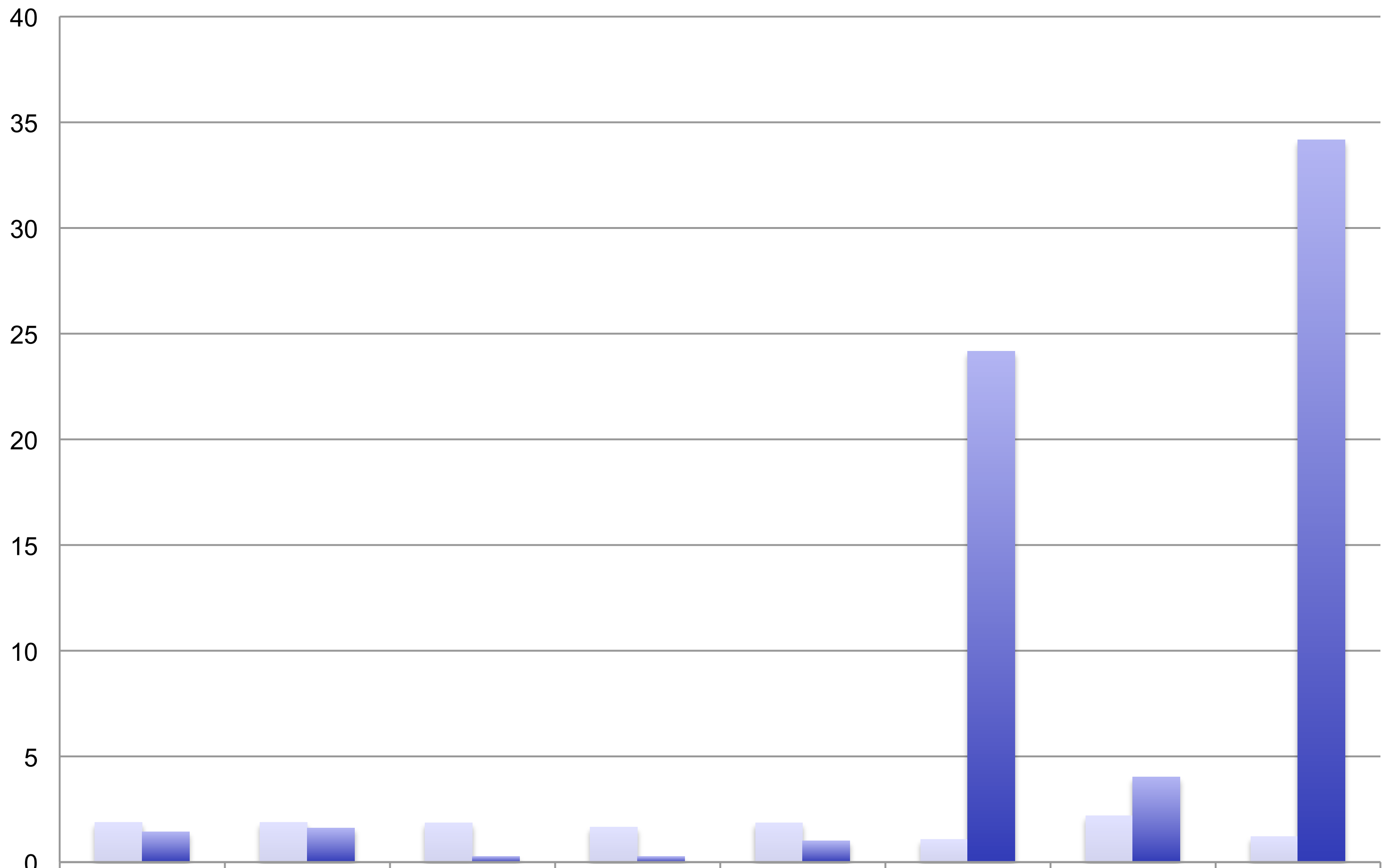
- We used mainly two systems to implement, test and evaluate each given task (Hardware Specifications)
  - **Linux System**:
  - Memory : 3.8 GiB
  - Processor: Intel CoreTM i5-4210U CPU @ 1.70GHz
  - OS Type : 64-bit

  - **Mac OSX 10.11.3**:
  - Memory : 4 GB 1333 MHz DDR3
  - Processor: Intel CoreTM i5 2.3 GHz
  - OS Type : 64-bit
- Software Specifications:
  - Python Language(Version 3)
  - PyCharm Integrated Development Environment(IDE).

Maximum Compression Ratio

- **Goal**: To analyze 5 different compression algorithms and evaluations of their results
- We have considered following compression algorithms;
  - Burrows-Wheeler
  - Zlib (Deflate)
  - Gzip (Deflate)
  - Huffman Coding
  - LZMA
  - LZW
- We evaluated different parameters such as, Compression ratio, Time recorded for compression, Original file size and Compressed file size
- Original file size: ~300 MB

Maximum Compression Ratio



	Burrows wheeler- level 1	Burrows wheeler- level 9	Deflate	gzip-Level 1	gzip-Level 9	Huffman Coding	LZMA	LZW
Compression Ratio	1.871581886	1.876403624	1.861655914	1.658454663	1.862131738	1.069734991	2.193335882	1.199236801
Compression Time (Min)	1.4368333	1.618	0.28	0.2681	1.0008333	24.16	4.03	34.16

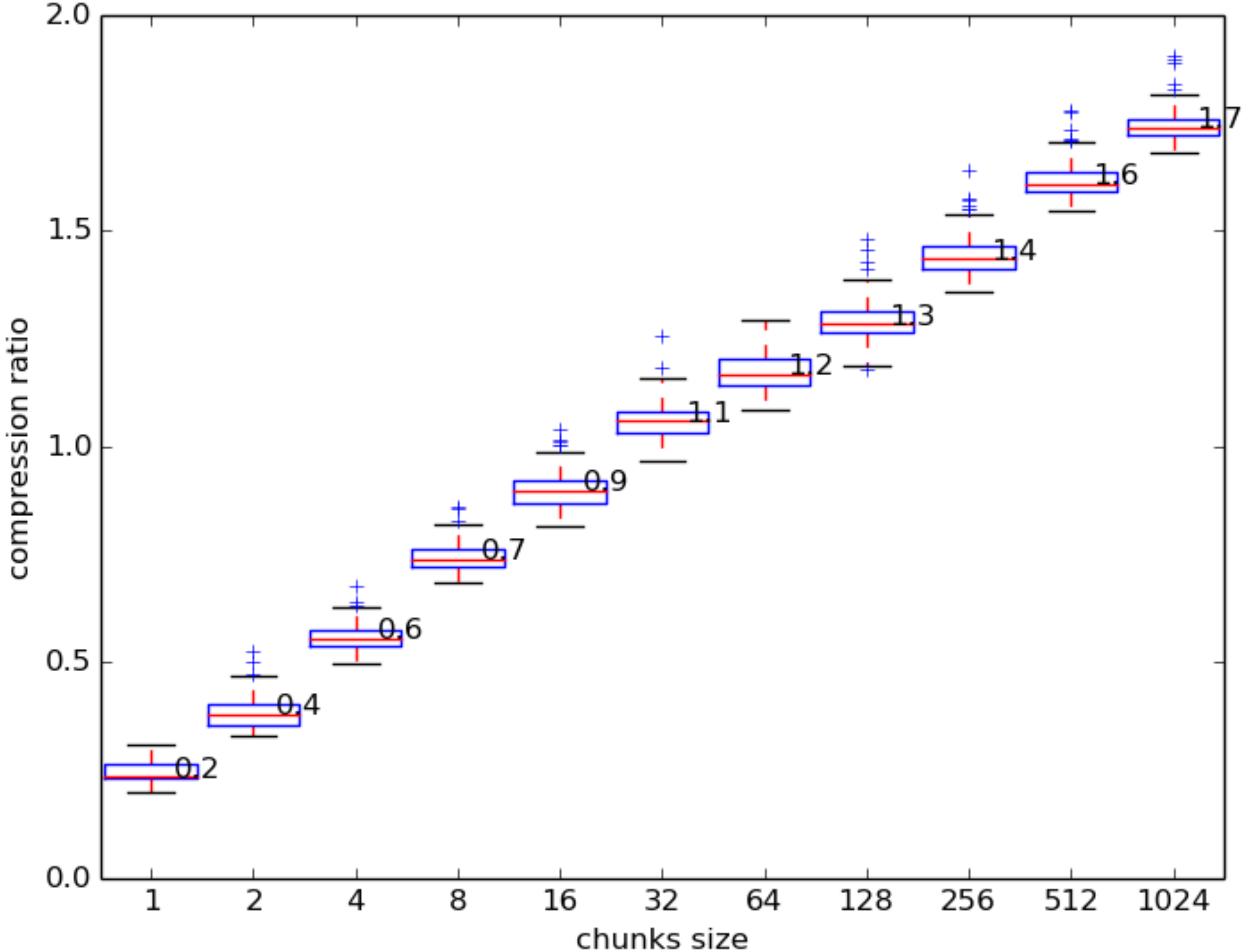
Maximum Compression Ratio

- Higher compression ratio has given by LZMA at average time of ~4 minutes
- Huffman coding has provided lower compression ratio in approximate ~25 minutes
- LZW took approximate ~35 minute for compression
- Deflate algorithm (Zlib) took ~28 seconds for compression

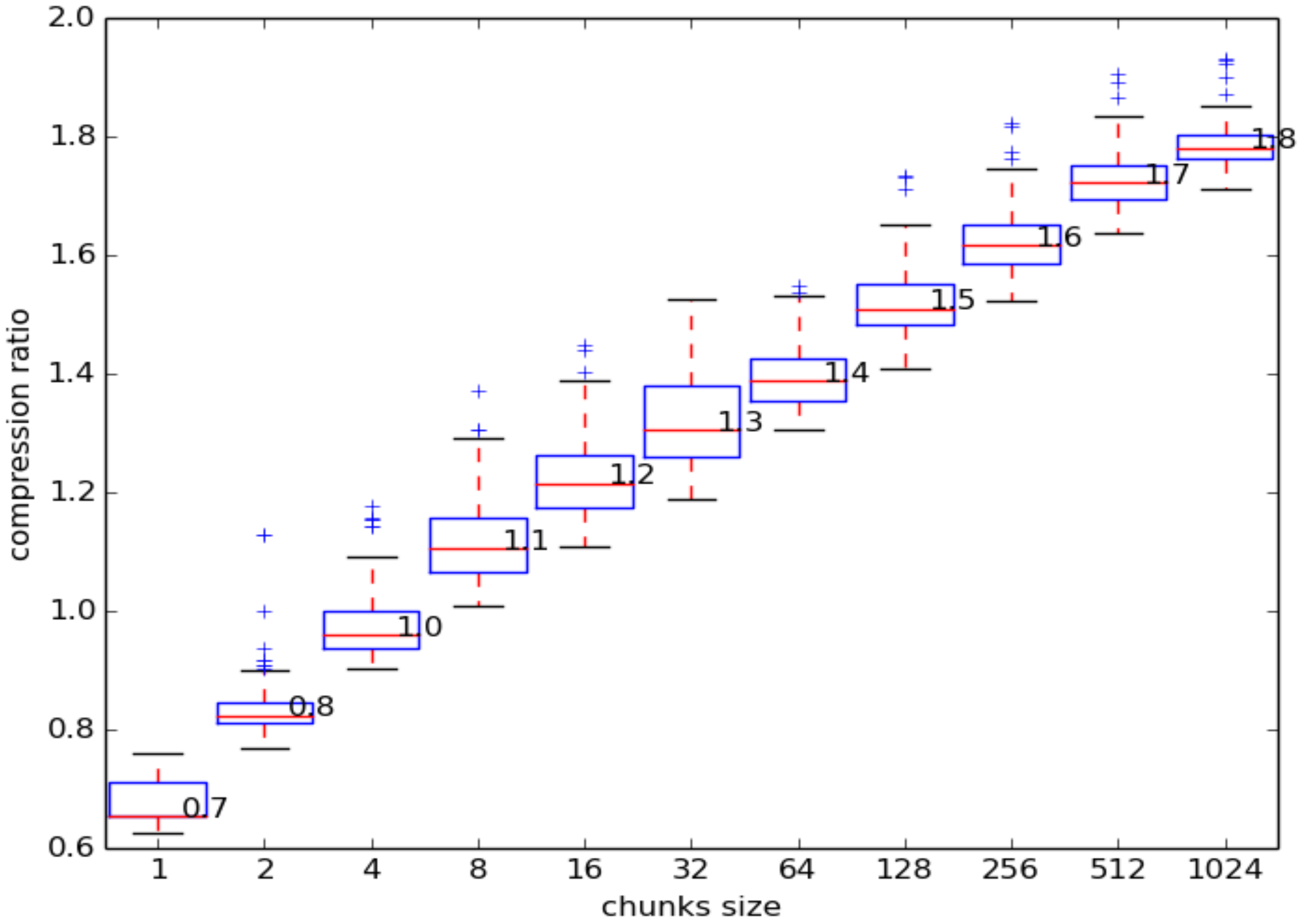
Chunk Size Vs. Compression Ratio and Profiling

- **Goal:** To evaluate compression ratio for different block sizes of given data
- We considered Zlib and Burrows-Wheeler compression algorithms
- The experiment was conducted on different group sizes of 1,2,4,8,16,64,….,1024 and repeat it with 100 different groups for each size
- Used cleaned data set by removing the ESC characters from each packet
- Dropped all packets except packet type '2' and '3'

Chunk Size Vs. Compression Ratio of Burrows Wheeler



Chunk Size Vs. Compression Ratio of Zlib



Chunk Size Vs. Compression Ratio

- Burrows Wheeler gives compression ratio
  - Chunk size 1 → 0.2
  - Chunk size 1024 → 1.7
- Zlib gives compression ratio
  - Chunk size 1 → 0.7
  - Chunk size 1024 → 1.8
- Zlib takes 1.2ms to compress chunk size of 1024
- Burrows Wheeler takes 6ms to compress chunk size of 1024
- **Insight:** As chunk size grows compression ratio also increases

Profiling LZW

- Analyzed the execution time characteristics of main functions in a LZW compression algorithm.
  - **Pack function**: Takes an iterator of integer codepoints as input and returns an iterator over bytes, containing the codepoint packed into varying lengths
  - **Encode_bytes**: Converts the given input to byte format
  - **Read_bytes**: Reading bytes
  - **Write_bytes**: Writing bytes
  - **Bits_to_bytes**: Which converts data's in bit format to byte format
  - **Int_to_bits**: Converts integer format to bits format

Profiling LZW

Sr. Nr.	Name of the Functions	Execution Time (Sec)
1	Pack	509.971
2	Encode_bytes	500.919
3	Read_bytes	73.034
4	Write_bytes	95.960
5	Bits_to_bytes	261.806
6	Int_to_bits	475.010

Result shows average time over multiple runs (5 Runs)

Experimental Setup

- The sensor data consists of
  - <esc> "2": 6 byte MLAT timestamp, 1 byte signal level, 7 byte Mode-S short frame
  - <esc> "3": 6 byte MLAT timestamp, 1 byte signal level, 14 byte Mode-S long frame
- Drop the packets that begin with unknown codes <esc> "?"
- **System Design Requirements**
  - Real-time Capability: Compression and Decompression should never exceed 1s
  - Low complexity of compression: Compression part should have low complexity
  - Low Communication Overhead: Performance of the system measures in Total transmit data including system/Control overhead of the compression system.
- The compression and decompression system should be able to work via Linux pipelines as follows.

```
netcat sensordata | compress | tee a.bin | decompress > b.bin
```

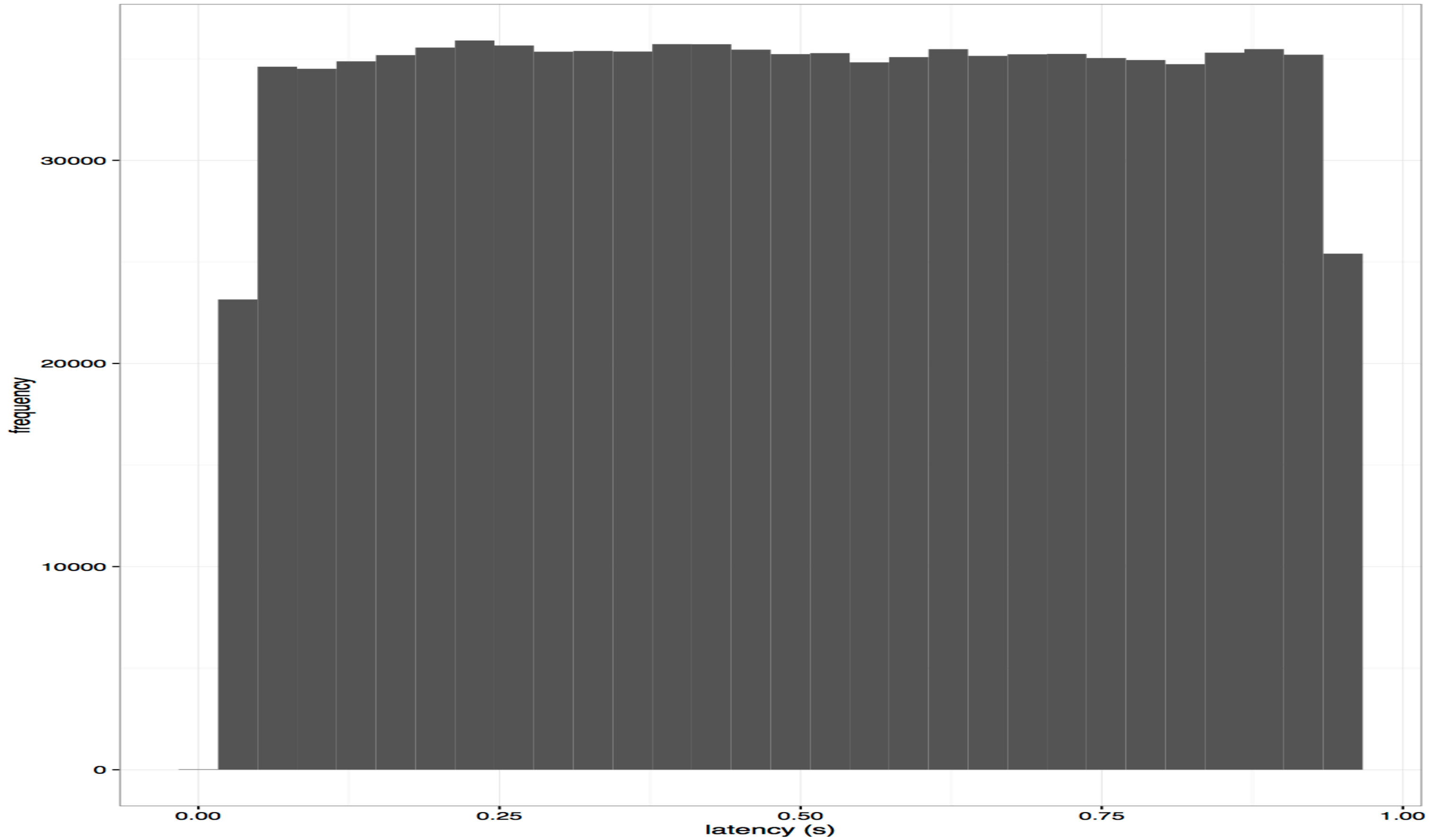
Stream Compression System Design

- **Sensor(Source):**
  - Ignore ESC character and packets of unknown type(<esc> '?')
  - Use buffer to store packets of different sizes
  - **Idea:** Continuously stores sequence of messages in a buffer for some amount of milliseconds and then compress the data
  - Send compressed data to Server(Destination)
- **Server(Destination)**
  - Receives sequence of compressed data
  - Decompress it with same algorithm

Stream Compression System Design

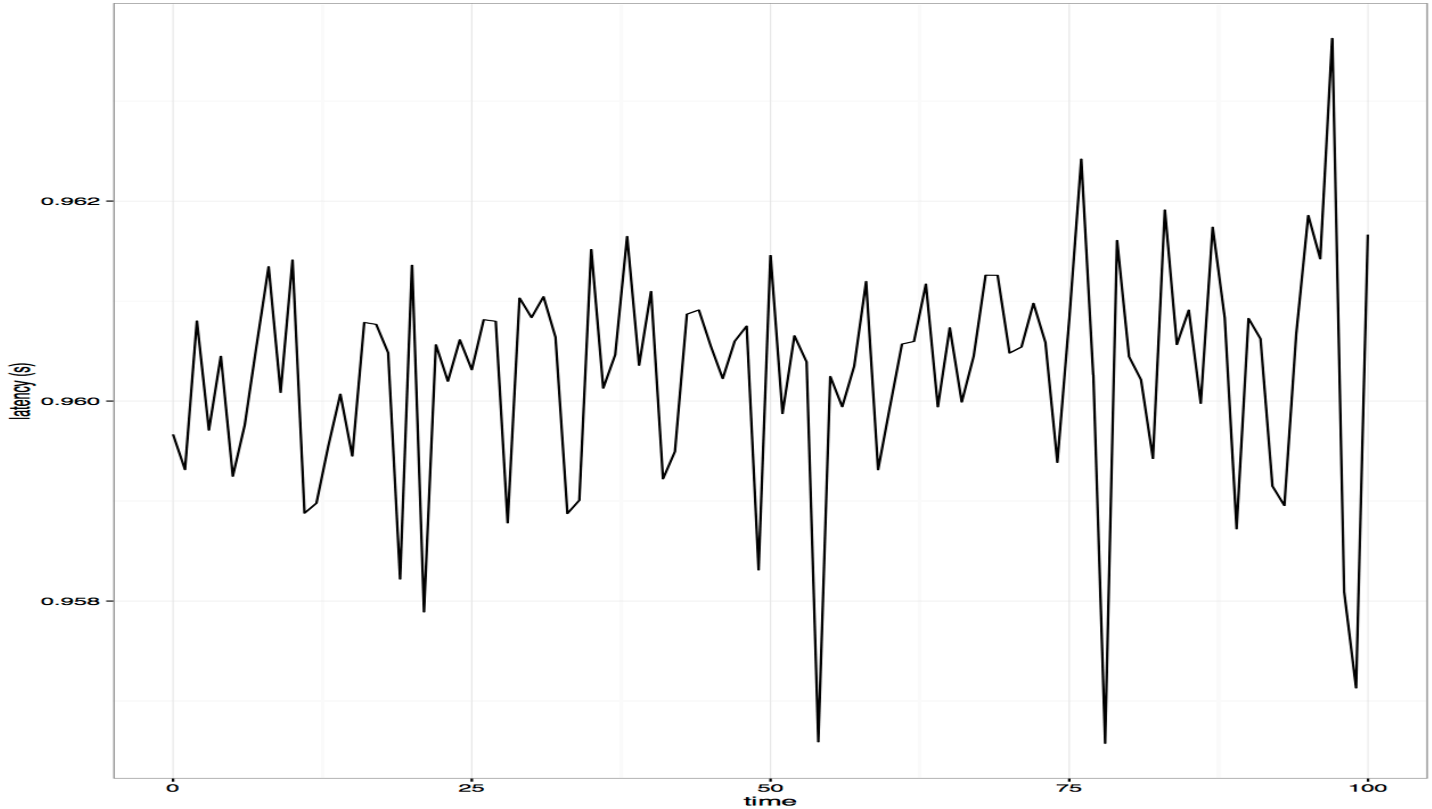
- **Choice of Compression Algorithm**
  - Use LZMA if compression ratio is critical, and you are willing to sacrifice speed to achieve it.
  - Use Zlib, if speed and compression are more or less equally important
  - Our choice was Zlib(refer slide 16)
- Planned to design a compression system that buffer for 900ms and compress and decompress the data in remaining 100ms

Performance Evaluation



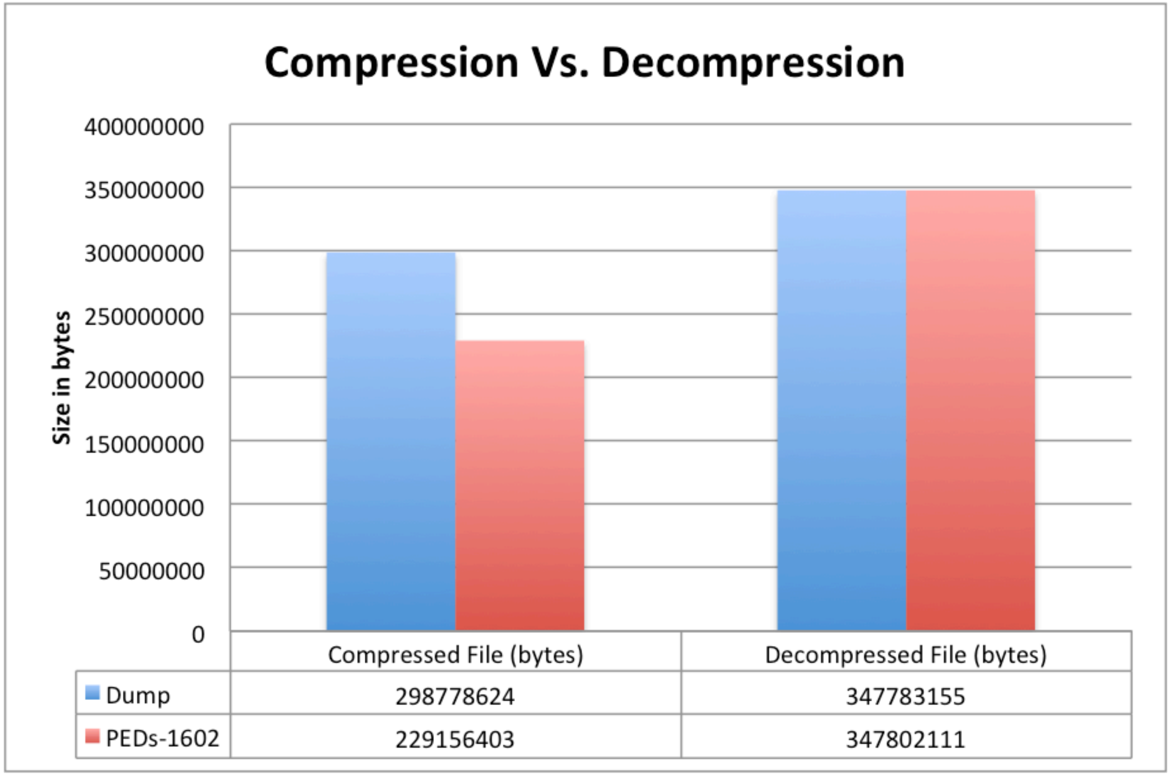
- Frequency Vs Latency of Sensor data
  - Evaluated 1 million packets of final result
  - Frequency distribution was linear over majority of time and latency did not exceed time limit of 1s

Performance Evaluation



- Maximum latency distribution of Sensor data
  - Considered the packets which show the maximum latency distribution
  - Noticed that the major portion of the latency was during buffering of the sensor data.
  - The maximum latency varies from 0.956 seconds to 0.963 seconds.

Performance Evaluation



- Comparison of compression ratios of group 1602 and 'Dump'
  - Dump is the output of the out-of-the-box Linux stream compression using the Lempel-Ziv algorithm
  - Implementation was tested over more than 24 hours
  - Recorded compression ratio is 1.52 for 1602 and 1.16 for 'Dump'

Conclusion

- Could reduce the network load significantly without any loss of packets by meeting the given system design requirements.
- Achieved high compression ratio of ~1.52 in sequence of data stream using deflate algorithm (Zlib).
- The maximum latency of around 0.963s recorded during our experiment.