EFFICIENT BUFFER MANAGEMENT AND CODING TECHNIQUES FOR VIDEO STREAMING IN OVERLAY NETWORKS

by

MASOOD AHMED USMANI 2010103045
ARPITA RAVEENDRAN 2010103033
RAMESH KALIDAS 2010103590
GURUPRASAD SRIDHARAN 2010103567

A project report submitted to the

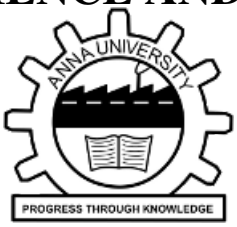
FACULTY OF INFORMATION AND COMMUNICATION ENGINEERING

in partial fulfillment of the requirements for the award of the degree of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

ANNA UNIVERSITY, CHENNAI – 25

APRIL 2014

BONAFIDE CERTIFICATE

Certified that this project report titled **EFFICIENT BUFFER MANAGEMENT AND CODING TECHNIQUES FOR VIDEO STREAMING IN OVERLAY NETWORKS** is the *bonafide* work of **MASOOD AHMED USMANI (2010103045)**, **ARPITA RAVEENDRAN (2010103033)** and **RAMESH KALIDAS (2010103590)** and **GURUPRASAD SRIDHARAN (2010103567)** who carried out the project work under my supervision, for the fulfillment of the requirements for the award of the degree of Bachelor of Engineering in Computer Science and Engineering. Certified further that to the best of my knowledge, the work reported herein does not form part of any other thesis or dissertation on the basis of which a degree or an award was conferred on an earlier occasion on these or any other candidates.

Place: Chennai
Date:

Dr. C. Chellappan
Dean and Professor
Department of Computer Science and Engineering
Anna University, Chennai – 25

COUNTERSIGNED

Head of the Department,
Department of Computer Science and Engineering,
Anna University Chennai,
Chennai – 600025

ACKNOWLEDGEMENT

We express our deep gratitude to our guide, **Dr. C. Chellappan** for guiding us through every phase of the project. We appreciate his thoroughness, tolerance and ability to share his knowledge with us. We thank him for being easily approachable and quite thoughtful. Apart from adding his own input, he has encouraged us to think on our own and give form to our thoughts. Without his immense support through every step of the way, we could never have made it to this extent. We would also like to extend our thanks to **Mr. T. Ruso** for helping us throughout by all means and for his unstinting support. We express our thanks to the panel of reviewers **Dr. Arul Siromoney**, **Dr. Rajeswari Sridhar** and **Dr. Geetha Palanisamy** for their valuable suggestions and critical reviews throughout the course of our project. We thank our parents, family, and friends for bearing with us throughout the course of our project and for the opportunity they provided us in undergoing this course in such a prestigious institution.

MASOOD AHMED USMANI

ARPITA RAVEENDRAN

RAMESH KALIDAS

GURUPRASAD SRIDHARAN

ABSTRACT

IP multicast is one of the best techniques for video streaming in the Internet. It faces issues with respect to address allocation, routing, authorisation, group management, security, Quality of Service (QoS) and scalability. Local ISPs don't enable IP multicast services because of the cost incurred in using multicast-enabled routers.

To solve these issues some of the IP layer functionality have been shifted into Application Layer thus leading to Application Layer Multicast (ALM) protocols. However, ALM protocols face issues related to synchronous data delivery, scalability, link stress, link stretch and node failures.

Protocols such as NICE, CoolStreaming/DONet, Bin-cast and mTreebone are known to tackle these issues. We propose the Smooth Streaming protocol, which comprises efficient buffer management and coding techniques to address the aforementioned issues in ALM.

Our approach is novel because none of the ALM protocols in existence focus on management of buffer contents.

ABSTRACT

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

ALM Application Layer Multicast

IGMP Internet Group Management Protocol

FF Finite Field

GF Galois Field

NC Network Coding

SF Store and Forward

IPA Infinitesimal Perturbation Analysis

PIM Protocol Independent Multicast

NICE Network Information and Control Exchange

MPEG Moving Pictures Experts Group

CHAPTER 1

INTRODUCTION

1.1 PROBLEM STATEMENT

An overlay network is a network with logical or virtual links that is built on top of an IP network. The physical network has only a limited number of IP addresses available. Moreover, multicasting in IP requires installation of special software on the routers and it faces issues with respect to address allocation, routing, authorisation, group management, security, Quality of Service(QoS) and scalability. To solve these issues, some of the IP layer functionality have been shifted into the Application Layer. Therefore, we consider overlay networks with virtual addresses so as to enable multicasting in the application layer, thus leading to Application Layer Multicast (ALM) protocols. However, Multimedia applications like multiplayer online gaming, e-learning, multi-party video conference, tele-medicine, et cetera over an overlay network face challenges with respect to delay, throughput and robustness.

1.2 OBJECTIVES

Our smooth streaming protocol aims to address the aforementioned issues related to **delay, throughput, robustness, average time to send** and **reliability**.

1.3 METHODOLOGY

Our protocol which comprises of efficient buffer management and coding techniques involves the following methodologies:

- Network Coding at the source and some selected intermediate nodes.
- Buffer management using techniques such as IPA and Stochastic approximation.
- Routing using a technique called 'treebone' which is a tree-mesh overlay structure.

1.4 DEFINITIONS

- OVERLAY NETWORK:

  An overlay network is a computer network which is built on the top of another network. Nodes in the overlay can be thought of as being connected by virtual or logical links, each of which corresponds to a path, perhaps through many physical links, in the underlying network. For example, distributed systems such as cloud computing, peer-to-peer networks, and client-server applications are overlay networks because their nodes run on top of the Internet.

- ALM:

  High bandwidth multi-source multicast among widely distributed nodes is a critical capability for a wide range of applications including audio and video conferencing, multi-party games and content distribution. Throughout the last decade, a number of research projects have explored the use of multicast as an efficient and scalable mechanism to support such group communication applications. Multicast decouples the size of the receiver set from the amount of state kept at any single node and potentially avoids redundant communication in the network. The limited deployment of IP Multicast, a best effort network layer multicast protocol, has led to considerable interest in alternate ap-

proaches that are implemented at the application layer, using only end-systems. In an overlay or end-system multicast approach participating peers organise themselves into an overlay topology for data delivery. Each edge in this topology corresponds to a unicast path between two end-systems or peers in the underlying Internet. All multicast-related functionality is implemented at the peers instead of at routers, and the goal of the multicast protocol is to construct and maintain an efficient overlay for data transmission.

- FINITE FIELD:

  In algebra, a finite field or Galois field is a field that contains a finite number of elements, called its size. Finite fields only exist for which the size is a prime power p_k (where p is a prime number and k is a positive integer). For each prime power, there is a finite field with this size, and all fields of a given size are isomorphic. The characteristic of a field of size p_k is p (this means that adding p copies of any element always results in zero). In a finite field of size q, the polynomial $T_q - T$ is the product of q different linear factors, and, therefore, has all the elements of the field as roots. The multiplicative group of the non-zero elements of a finite field is a cyclic group, and thus, the non-zero elements can be expressed as the powers of a single (but generally not unique) element called a primitive element.

- LINEAR NETWORK CODING:

  Linear network coding is a technique which can be used to improve a network's throughput, efficiency and scalability, as well as resilience to attacks and eavesdropping. Instead of simply relaying the packets of information they receive, the nodes of a network take several packets and combine them together for transmission. This can be used to attain the maximum possible information flow

in a network. It has been proven that linear coding is enough to achieve the upper bound in multicast problems with one or more sources.

- RANDOM LINEAR NETWORK CODING:

  Random network coding is a simple yet powerful encoding scheme, which in broadcast transmission schemes allows close to optimal throughput using a decentralised algorithm. Nodes transmit random linear combinations of the packets they receive, with coefficients chosen from a Galois field. If the field size is sufficiently large, the probability that the receiver(s) will obtain linearly independent combinations (and therefore obtain innovative information) approaches one.

- TREEBONE:

  Recently, application-layer overlay networks have been suggested as a promising solution for live video streaming over the Internet. To organise a multicast overlay, a natural structure is a tree, which, however, is known to be vulnerable to end-hosts dynamics. Data-driven approaches address this problem by employing a mesh structure, which enables data exchanges among multiple neighbours, and thus, greatly improves the overlay resilience. It unfortunately suffers from an efficiency-delay trade-off, because data have to be pulled from mesh neighbours by using extra notifications periodically. The idea is to identify a set of stable nodes to construct a tree-based backbone, called treebone, with most of the data being pushed over this backbone. These stable nodes, together with others, are further organised through an auxiliary mesh overlay, which facilitates the treebone to accommodate node dynamics and fully exploit the available bandwidth between overlay nodes. This hybrid design, referred to as mTreebone, brings

a series of unique and critical design challenges. In particular, the identification of stable nodes and seamless data delivery using both push and pull methods.

1.5 SCOPE OF THE PROJECT

Almost all Multimedia applications involve video streaming over an overlay network. Network coding is a general technique, independent of the underlying technology. Our buffer management algorithm can be implemented in any network which has an unpredictable traffic scene. Thus, our protocol can be used in applications like multiplayer online gaming, e-learning, multi-party video conference, tele-medicine, et cetera where Smooth Streaming will handle the sending of video packets in a more efficient way than other ALM protocols.

1.6 ORGANISATION OF THESIS

The outline of thesis follows:

Chapter 2: Survey of the previous works that have taken place in domains we've explored to develop our system.

Chapter 3: Description of our Systems Functional and Non-functional requirements.

Chapter 4: The design principles of our system (i.e) An outline of how our protocol works.

Chapter 5: The complete System's development with a detailed description of each Module along with the Algorithms implemented.

Chapter 6: Enumeration of the obtained results and related discussion.

Chapter 7: A write up about the conclusions we drew from our project besides exploring possible future work.

CHAPTER 2

RELATED WORK

2.1 MULTICAST ROUTING

- NICE: Banerjee et al. [3] discuss about the NICE protocol (Internet Co-operative Environment) which discusses a hierarchical tree based protocol for multicast routing. They construct this hierarchical structure of overlay nodes using metrics related to delay and bandwidth for low latency distribution. Using NICE, it is possible to control the variations in the hierarchical multicast trees with low control overhead.

- PIM: [7] and [1] describe two Protocol Independent Multicast protocols which are used for distribution of data through a multicast network. PIM does not have its own routing mechanism. It uses the routing information provided by some other routing protocols.

- IGMP: [5] describes the Internet Group Management Protocol which is used by hosts/routers (IPv4 systems) to report their IP multicast group memberships to any neighbouring multicast routers. IGMP is also used for other IP multicast management functions using message types other than those used for group membership reporting functions and messages. It is a vital part of IP multicast.

- mTreebone: Wang et al. [14] discuss the mTreebone protocol which combines the advantages of both tree and mesh overlays. The network performance depends upon a stable subset of nodes known as the backbone. Treebone is a tree layout of the backbone nodes in the network. The non-stable nodes appear in the outskirts of the treebone as leaves. The stable nodes along with other nodes are organised using an auxiliary mesh overlay which provides resilience to handle peer dynamics and exploit available bandwidth. They provide algorithms for construction and control of the treebone structure.

- CoolStreaming/DONet: Zhang et al. [16] describe a push/pull mechanism for data exchange in overlay networks. They use random network coding with random gossiping for disseminating video blocks in a peer-to-peer live streaming network. Every node periodically exchanges data availability information with a randomly chosen set of neighbours and retrieves unavailable from one or more neighbours or supplies data to neighbours. They provide experimental results which prove the robustness and scalability of the ALM protocol.

- BinCast: Besharati et al. [4] describe a binning technique to cluster nearby receivers based on distance from a constant number of landmarks. Landmarks are the distribution centres in the overlay network. Then a k-ary tree constructed over the clusters with the most stable nodes in each cluster as the head. The heads decide where it is necessary to split/merge clusters when a node joins/leaves. They use a metric based on RTT to

measure the weighted distance from landmarks.

2.2 NETWORK CODING

- Ho and Lun [9] proposed a distributed random linear network coding approach for multi-source multicast networks and showed the advantages of randomised network coding over routing in such scenarios. They provided the success probability of such a coding scheme in arbitrary networks and generalised the method to work with correlated sources.

- Shokrollahi [12] introduced Raptor codes, an extension of Luby codes with linear time encoding and decoding and provided complexity analysis and derived the decoder error probability as a function of overhead.

- Thomos and Frossard [13] present a system for collaborative video streaming in wired overlay networks proposing a scheme that builds on both raptor codes and network coding in order to improve the system throughput and the video quality at clients.

2.3 BUFFER MANAGEMENT

- Markou et al. [11] present a technique for dynamic control of buffer size in wireless networks. The parameter considered is loss and workload costs which are constantly monitored. The buffer threshold is then set according to the cost at various times. They propose the use of IPA (Infinitesimal Perturbation Analysis) algorithm for cost monitoring and Stochastic approximation algorithm

for setting the threshold value accordingly. The simulation results indicated that this technique exhibits good convergence properties.

CHAPTER 3

REQUIREMENTS ANALYSIS

3.1 FUNCTIONAL REQUIREMENTS

3.1.1 Network Creation

- The system should be able to read input from text files.
- It should generate a graph out of the input.

3.1.2 Node creation

- It should create a thread for each node.
- It should perform initialisation based on the type of the node and start the thread.

3.1.3 Network coding

- It should form a coded packet from a list of uncoded/coded packets.
- It should be able to check the linear independence of packets for decoding.
- It should be able to decode a set of linearly independent packets.

3.1.4 Treebone creation

- It should create a n-ary tree and add the source node to the tree as root.
- It should insert random nodes into the tree.

3.1.5 Node promotion

- Each node should check itself against the age threshold at regular time intervals.
- If a node has an age greater than the threshold, it should promote itself to the tree from the mesh.
- The system should reassign the type of the promoted node.

3.1.6 Node addition

- When a node arrives, it should be added to the mesh.
- Random edges should be formed between the new node and an existing set of nodes.
- A new thread should be set up for the same node.

3.1.7 Node deletion

- When a mesh/tree node leaves, it should be deleted lazily and connectivity should be established between its neighbours.
- The thread corresponding to the leaving node must be killed.

3.1.8 Buffer management

- The system should be able to monitor packet losses and update busy periods and cumulative time accordingly.
- At the end of observation period, system should compute the cost gradient and make threshold update decisions.

3.2 NON-FUNCTIONAL REQUIREMENTS

3.2.1 Scalability

The system should be able to handle a large number of nodes in the network. It should display scalability with respect to both time and memory. Scalability shouldn't be affected by peer dynamics.

3.2.2 Reliability

The system should be able to transmit the entire video within a bounded time to all the clients. There should not be any loss of data/quality. It should maintain a consistent view of the network at all times.

3.2.3 Robustness

The system should be able to maintain mesh and tree connectivity even when nodes leave. It shouldn't be affected due to lossy links in the network.

CHAPTER 4

SYSTEM DESIGN

4.1 ARCHITECTURE DIAGRAM

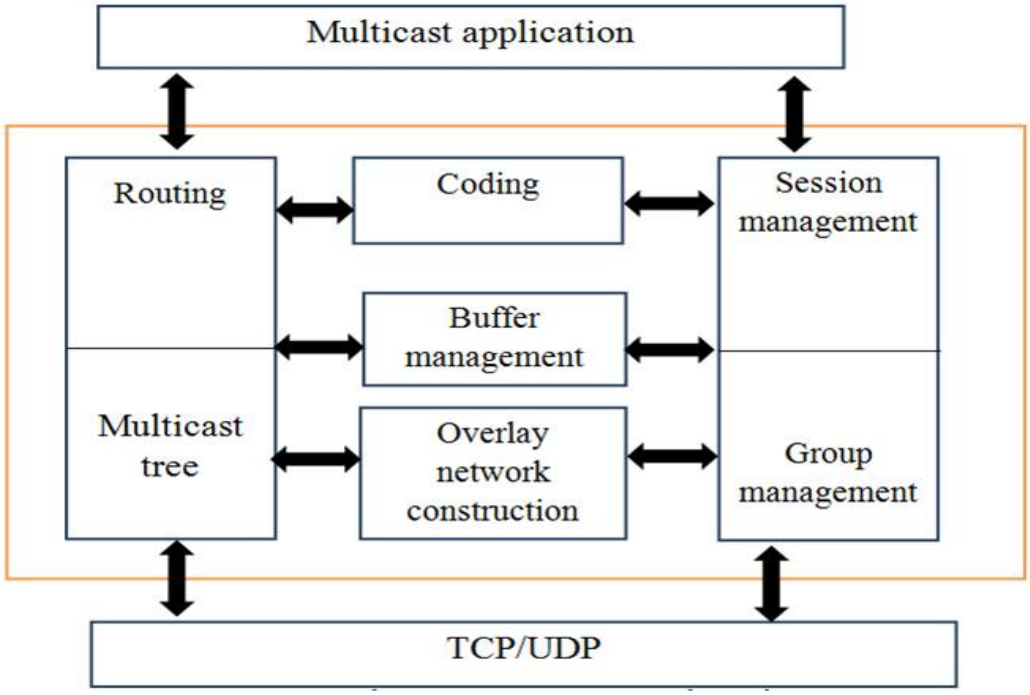


Figure 4.1 Architecture Diagram

Figure 4.1 is the architecture diagram which describes the different operations that our protocol performs in relevant layers of the network stack.

4.2 DESIGN PRINCIPLES

We have three stages at which various operations take place: source, intermediate nodes, and destination.

- Source:

  - We take some MPEG4 encoded video which is first split into generations of packets where each generation consists of a certain no. of blocks.
  - The source performs network coding on the video packets.
- Intermediate nodes:
  - At these nodes, the packets get queued in the input buffer. According to the network traffic, we modify the buffer threshold and manage the buffers dynamically.
  - If the node is a store-and-forward node, it simply sends the packets forward.
  - If the node is a network coding node, re-encoding is performed on the received packets.
  - Routing is performed using the treebone method, wherein we construct a tree-mesh structure similar to the one in [14], where they combine both tree and mesh overlays.
- Destinations:
  - At the client side, the packets are decoded using the information sent along with the packets.
  - The decoded packets are played back to the client.

CHAPTER 5

SYSTEM DEVELOPMENT

5.1 DEVELOPMENT OF VARIOUS MODULES

5.1.1 Network Coding scheme

We propose to use a variant of random linear network coding which works in a decentralised manner suggested by [8].

The description of the algorithm is as follows,
At the source node, we have K message packets $w_1, w_2, \ldots, w_K$, which are vectors of length λ over some finite field F_q. (If the packet length is b bits, then we take $\lambda = \lceil b/\log(2q) \rceil$.) The message packets are initially present in the memory of the source node. The coding operation performed by every intermediate node is the same. The node stores the received packets into its memory. Packets are formed for injection as random linear combinations of its memory contents at the time of packet injection on an outgoing link. The coefficients of the combination are drawn uniformly from F_q. Since all coding is linear, we can write any packet u in the network as a linear combination of $w_1, w_2, \ldots, w_K$, namely, $u = \sum_{k=1}^{K} \gamma_k w_k$, where γ is the global encoding vector of u, and it must be sent along with u, as side information in its header. The overhead this incurs is negligible if packet size is sufficiently large.Nodes are assumed to have unlimited memory. The scheme can be modified so that received packets are stored into memory

only if their global encoding vectors are linearly-independent of those already stored.This modification ensures that nodes never need to store more than K packets.

Algorithm Initialisation:

- The source node stores the message packets $w_1, w_2, \ldots, w_K$, in its memory.

Operation:

- When a packet is received by a node, the node stores the packet in its memory.
- When a packet injection occurs on an outgoing link of a node, the node forms the packet from a random linear combination of the packets in its memory. Suppose the node has L packets $u_1, u_2, \ldots, u_L$ in its memory. Then the packet formed is
  $u_0 := \sum_{l=1}^{L} \alpha_l u_l$
  where α_l is chosen according to a uniform distribution over the elements of F_q. The packet's global encoding vector λ, which satisfies
  $u_0 = \sum_{k=1}^{K} \gamma_k w_k$, is placed in its header

Decoding:

- Each sink node performs Gaussian elimination on the set of global encoding vectors from the packets in its memory. If it is able to find an inverse, it applies the inverse to the packets to obtain $w_1, w_2, \ldots, w_K$; otherwise, a decoding error occurs.

5.1.2 Buffer Management

In networks which have heavy and unpredictable traffic, we need to have mechanisms for effective resource management. We

should design systems which make optimal use of the already present infrastructure. Our major concern in this video streaming project is efficient management of the buffers at intermediate nodes. Before we go into the type of queueing algorithm we are to use, the basic control parameter (i.e.) the buffer threshold has to be set. We need a system by which we can modify the buffer threshold and set an optimal threshold value at every time instant dynamically as video packets arrive and as traffic conditions change. In order to modify buffer threshold, some performance measure should be continuously monitored. We consider the cost here. Initially, we use the Infinitesimal Perturbation Analysis algorithm to obtain sensitivity estimates of the various costs with respect to buffer threshold. We then make use of these estimates in the Stochastic Approximation Algorithm to set the optimal buffer threshold value. Some variables referred:

θ - Buffer threshold

$\alpha_n(t)$ - rate of arrival of packets which terminate at that node

$\alpha_f(t)$ - rate of arrival of forwarded packets from neighbouring nodes

$\beta(t)$ - message transmission rate

$\eta(t;\theta)$ - rate at which queued packets are forwarded to neighbouring nodes

$\sigma(t;\theta)$ - rate of timeout of queued packets

$x(t;\theta)$ - size of buffer's content at time t

$\delta(t;\theta)$ - system's outflow rate

$\gamma(t;\theta)$ - rate that packets are dropped when $x(t;\theta) \geq \theta$, where θ is the control parameter

As already mentioned, we control θ in such a way that a certain

performance measure (here, cost) is minimised.

$J_T = Q_T(\theta) + RC.L_T(\theta)$ Where: J_T is the total cost that should be minimised. $Q_T(\theta)$ is the cost due to packet workload/packet delays. $L_T(\theta)$ is the cost due to packet loss. RC is the rejection cost of each packet. T is the length of our observation interval.

J_T represents a trade-off between providing low delay and low packet loss probability. (i.e.) for small θ, the buffer overflow probability is high but workload or packet delay is small. As θ increases, packet loss probability drops, but workload increases. As a result, we have to do constrained optimisation and minimise the workload such that the packet overflow probability is below some threshold. We obtain sensitivity estimates $\frac{dL(\theta)}{d\theta}$ and $\frac{dQ(\theta)}{d\theta}$ through IPA algorithm.

Algorithm 1: Initial Perturbation Algorithm

Initialise a counter $L = 0$ and a cumulative timer $W = 0$
$\tau \leftarrow 0$
if *an overflow event is observed at time t and* $\tau = 0$ **then**
    $\tau \leftarrow t$
end
if *a busy period ends at time t and* $\tau > 0$ **then**
    $L \leftarrow L + 1$
    $W \leftarrow W + (t - \tau)$
    $\tau \leftarrow 0$
end
if $t \geq T$ *and* $\tau > 0$ **then**
    $L \leftarrow L + 1$
    $W \leftarrow W + (t - \tau)$
end

The final values of L and W, divided by the length of the observation interval T, provide the IPA derivative estimates $\frac{dL(\theta)}{d\theta}$ and $\frac{dQ(\theta)}{d\theta}$ respectively, by which we calculate the estimator of $\frac{dJ_T(\theta)}{d\theta}$ (i.e.) $\frac{dJ_T(\theta)}{d\theta} = \frac{W}{T} + RC.\frac{L}{T}$ Subsequently these estimates are used in the Stochastic Approximation Algorithm.

Algorithm 2: Stochastic Approximation Algorithm

```
1 Set an initial, arbitrary value θ_0 for the buffer size.
  for each step n do
2     Calculate the estimate dJ_T/dθ
      if dJ_T/dθ = 0 then
3         θ_{n+1} ← θ_n − 1
4     else
5         θ_{n+1} ← θ_n − s.dJ_T/dθ
6     end
7 end
```

If θ_0 is set to a very high value, it is possible that no packets are lost in the interval [0,T] and thus $\frac{dJ_T}{d\theta}$ will evaluate to 0 (according to the IPA Algorithm). However, this does not necessarily mean an optimal buffer size (due to excessive workload), therefore, when $\frac{dJ_T}{d\theta}$ evaluates to 0, we reduce θ by 1.

Modification:

Since Algorithm 2 only computes positive cost, we made some improvements to it. Our implementation involves taking the difference of gradients and then scaling it so that the values are in the

range [-1,1]. Let the scaled difference of gradient value be called α. The maximum threshold is then updated as $\theta_{n+1} = \theta_n + s.\alpha$

5.1.3 Routing

We construct a tree-mesh structure similar to the one in [14], where they combine both tree and mesh overlays. It is known that tree structures are not resilient to the volatility of nodes in the overlay network. Also, data-driven approaches using mesh structures face the problem of control overhead when new nodes join the network. It leads to increased network traffic due to proliferation of control messages. It has been shown in [14] that the best approach is to combine both tree and mesh overlays. The tree structure consists of nodes which are stable. Stability is proportional to the time spent in the network. Messages are propagated from the source through the Treebone and it eventually reaches the mesh overlay connecting the non-stable nodes at the leaves of the Treebone. We use a collaborative push/pull delivery system, in which packets are pushed over the treebone sequentially, and any missed blocks are pulled from the mesh overlay. This is depicted in Figure 5.1.

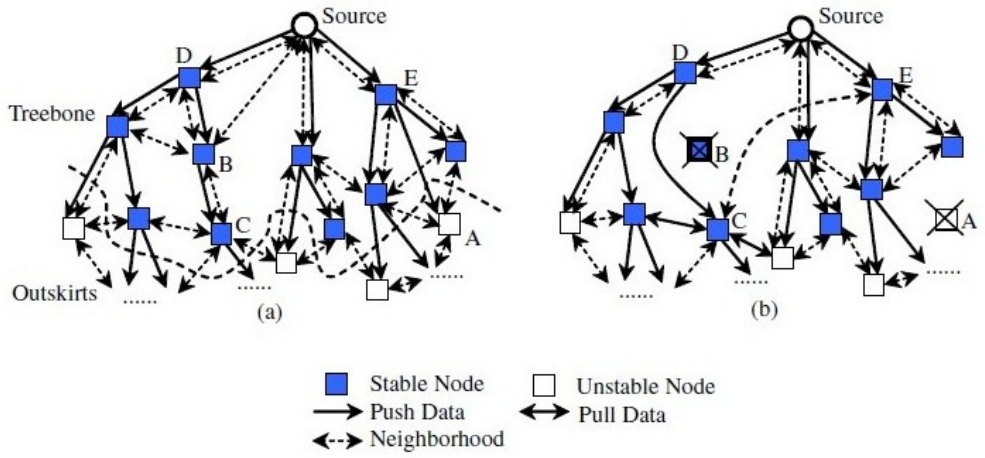


Figure 5.1 mTreebone Framework (a) A hybrid overlay (b) Handling node dynamics

When a node joins the network, it is added to the auxiliary mesh network, as one of the leaves of the treebone. A node may leave the network gracefully or abruptly fail. Failure can be detected by lack of control messages in the mesh or persistent losses in the treebone. In any case, mesh neighbours need to re-establish neighbour relationships with known local nodes, and treebone children need to relocate their parents. If a node leaves the treebone, while the treebone is being repaired, the mesh takes over data delivery temporarily. We update the Treebone periodically by including non-treebone nodes whose age exceeds the threshold. A node is included in the Treebone if its Estimated Service Time(EST) is greater than the age threshold T(t).

We use the following formula for EST derived in [14],

$$EST(t) = (L-t)\left(\frac{k\left(\frac{1}{x_t^{k-1}}-1\right)}{(k-1)\left(\frac{1}{x_t^{k}}-1\right)} - x_t\right)$$

$$T(t) = 0.3(L-t)$$

L - Session length

$L-t$ - Residual session length at time t

$T(t)$ - Age threshold at time t

t - Node arrival time

k - Shape parameter of Pareto distribution

x_t - Normalised age threshold $\left(\frac{T(t)}{L-t}\right)$

We need to maximise EST(t) with respect to x_t. We propose to extend their method by dynamically estimate k using observed node durations. We order the nodes in the treebone based on the score of each node which is calculated as the weighted sum of age, bandwidth, CPU speed. The function used to calculate the weight varies between the levels of the tree. This weight is used to decide whether to promote the node to the treebone or not.
We use Hash Table and adjacency list data structures for maintaining the treebone control structure and edge information for facilitating improved flexibility and scalability.

5.2 MODULE DIAGRAM

Figure 5.2 and Figure 5.3 depict our module diagram split into three stages (ie) functionality in three regions: Source node, Intermediate nodes and Client nodes.

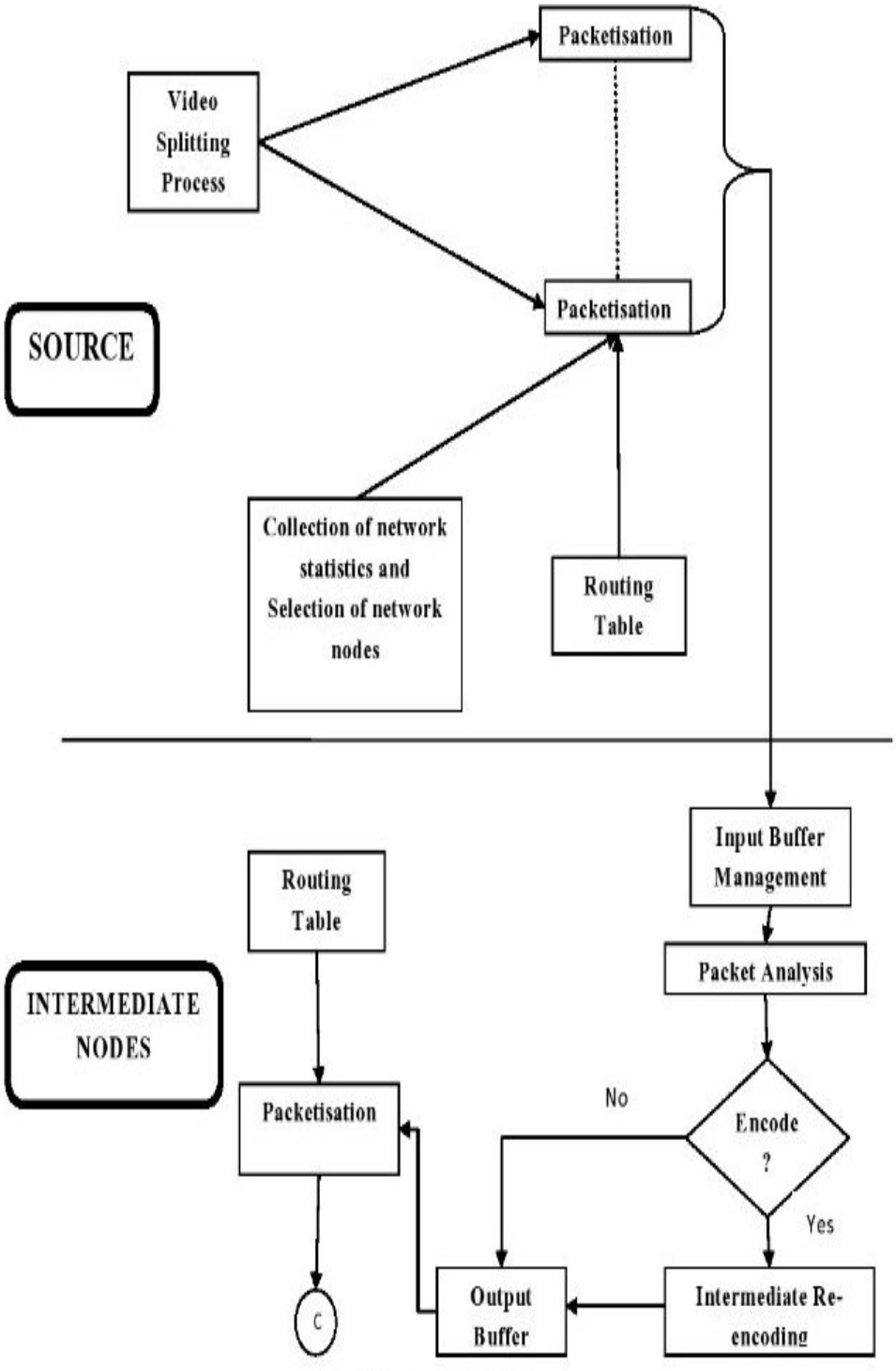


Figure 5.2 Module Diagram

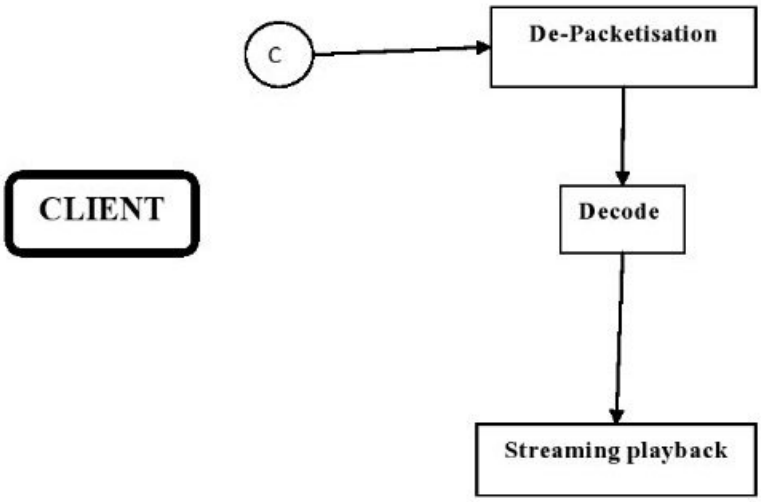


Figure 5.3 Module Diagram (contd.)

5.3 IMPLEMENTATION SPECIFICATIONS

5.3.1 Hardware

- Atleast 30 MB of free hard disk space
- Intel Core i3 @ 2.33 GHz or better
- Atleast 512 MB of available RAM
- 3.2 MP web camera or better

5.3.2 Software

- Java Applet
- Eclipse Kepler 4.3.2

5.4 SIMULATION SETUP

Table 5.1 Simulation setup

Parameters	Values
Observation Period(T)	100 ms
Payload Length	1 KB
Segment Threshold	10 KB
Step Size	30
Pareto parameter (k)	0.85
Segment duration (L)	1.5
No. of Children (Treebone)	4

Table 5.1 is a list of various parameters, along with the values, we use in our code.

CHAPTER 6

RESULTS AND DISCUSSION

The simulation of the system proves that the system works correctly. Overall, if the video received by the client through the overlay network is consistent with the video sent from the source, the simulation is said to work correctly. During the execution of the program, metrics such as time taken and other parameters along with relevant details were displayed in the console and written into log files.

The protocol was tested with different scenarios: a) Without buffer management and b) With buffer management and as suggested, our smooth streaming protocol, an ALM protocol with buffer management techniques showed improved results when compared to older systems without buffer management. Another scenario was with a) treebone and with b) random gossiping and use of our suggested treebone structure showed significant improvements with random gossiping.

6.1 DEFINITIONS

- AVERAGE DECODING TIME : The time taken by each client to decode the encoded packets, averaged over all the clients.

- AVERAGE MAXOUTS : Maxouts can be defined as the

drops encountered when the buffer threshold for a SF node is exceeded. We averaged the maxout values over all the SF nodes.

– SEGMENT SIZE : Segment size can be defined as the size of a generation of packets.

– DROP PROBABILITY : Drop probability can be defined as the probability of occurrence of packet loss. We monitored the decoding time by including a random drop probability and tested the efficiency of our system with lossy networks.

6.2 PLOTTING AVERAGE DECODING TIME AGAINST AVERAGE MAXOUTS

The **average of decoding time** for all the segments is taken along **x-axis** and the **average of the maxouts** for all the segments is taken along the **y-axis**.

As we can see in Figure 6.2, a system with a buffer management module has almost similar decoding time as one without it, but the number of maxouts is far lower. We experimented with different values of step size for the Stochastic Approximation Algorithm and plotted their average decoding time vs average number of maxouts. We could see that choosing step size as 30 gives the minimum number of maxouts.

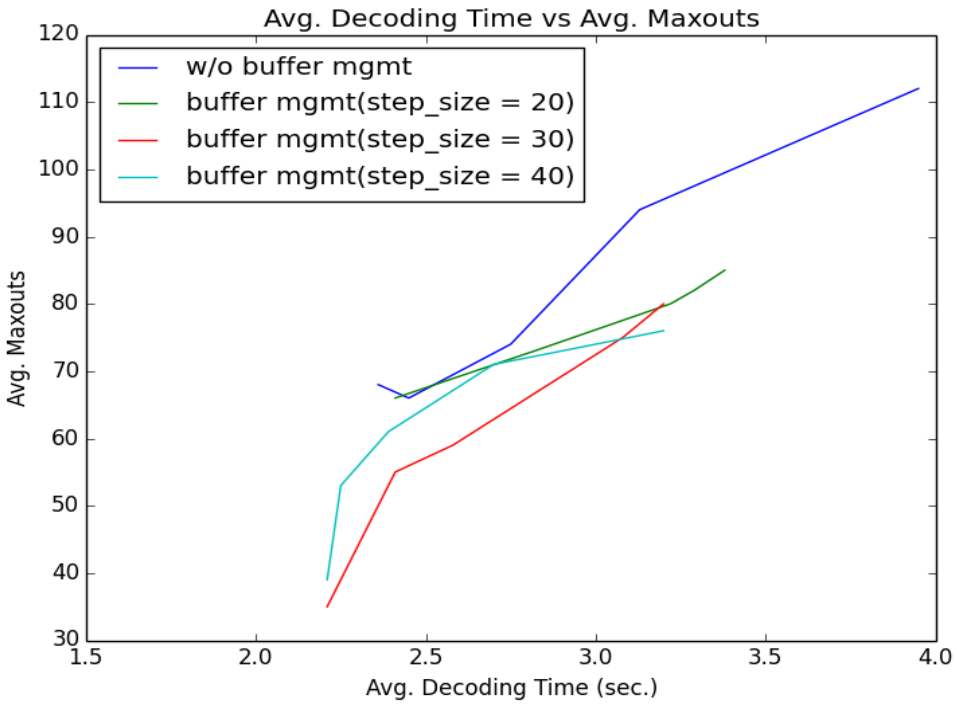


Figure 6.1 Plot of Decoding Time vs Maxouts

6.3 MEAN TIME AND DELAY

6.3.1 With respect to segment size

The **segment threshold of each generation** is taken along the **x-axis** and **average decoding time** for all segments is taken along the **y-axis**.

As could be seen from Figure 6.3.1, the relationship between Segment Threshold and Average decoding time is complicated. Segment Threshold is defined as the size threshold based on which the video file to be transmitted is split into blocks. All packets in a specific block or generation are coded together (intra-session coding). If we increase the segment threshold, the number of packets per segment increases and there are fewer segments in the video (assuming that the packet size is fixed). Segment Threshold has a

perfect positive correlation with decoding time per segment. They have a causal relationship with each other. When the segment threshold increases, the clients have to wait longer to receive the necessary packets to begin decoding. Thus the decoding time per segment increases with increase in segment threshold. But the overall decoding time for all segments has a different relationship with segment threshold. Choosing the optimal segment threshold value that globally minimises the time taken to decode all the video segments is an optimisation problem. The decoding time per segment depends on the generation size and loss probability. Choosing the right segment threshold is critical for decoding the transmitted packets since we use random linear network coding. Greater the number of packets in a generation, lesser the probability of decoding failure. Further, decoding failure and generation size are non-linearly related [8]. Thus, choosing the segment threshold is a trade-off between decodability and decoding time.

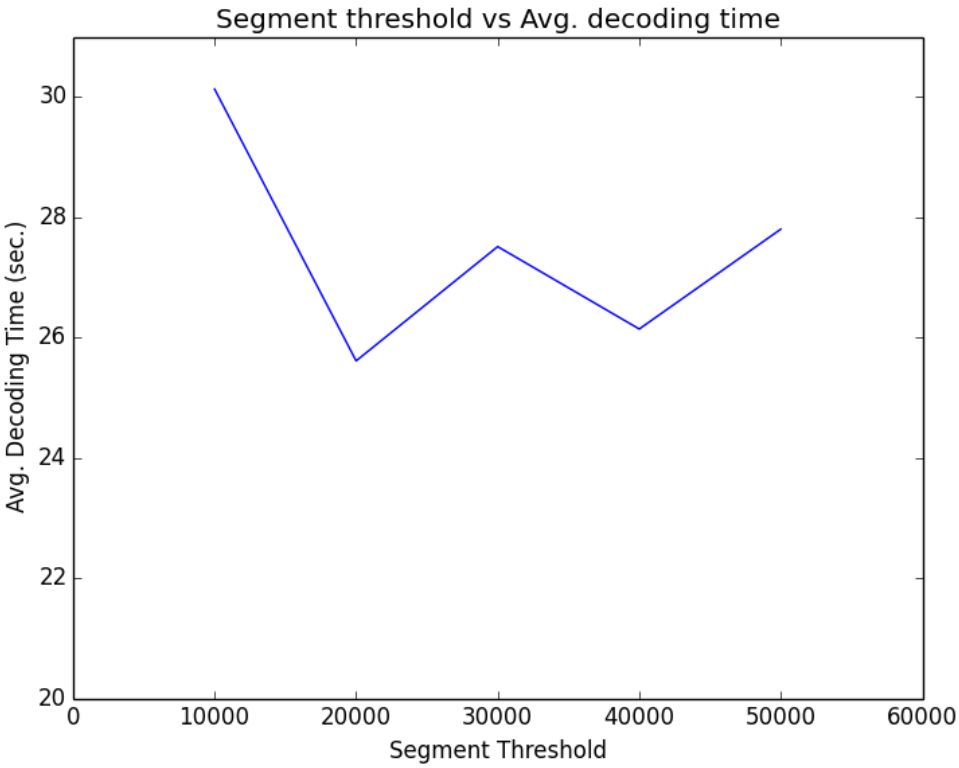


Figure 6.2 Plot of Segment Threshold vs Decoding Time

6.3.2 With respect to drop probability

Varying **values of drop probability** are taken along the **x-axis** and the **values of average decoding time** over several trials are taken along the **y-axis**.

Figure 6.3.2 shows the values of decoding time averaged over multiple trials plotted against drop probability in the network links (with segment threshold = 10000). The figure shows that the relationship between the drop probability and the decoding time is almost linear. We verified this by curve fitting. Thus, our method works well in the case of lossy networks too. Its performance degrades (i.e. end-to-end delay increases) linearly with increase in the drop probability. This robustness exhibited by the method

could be attributed to the use of network coding in the intermediate nodes.

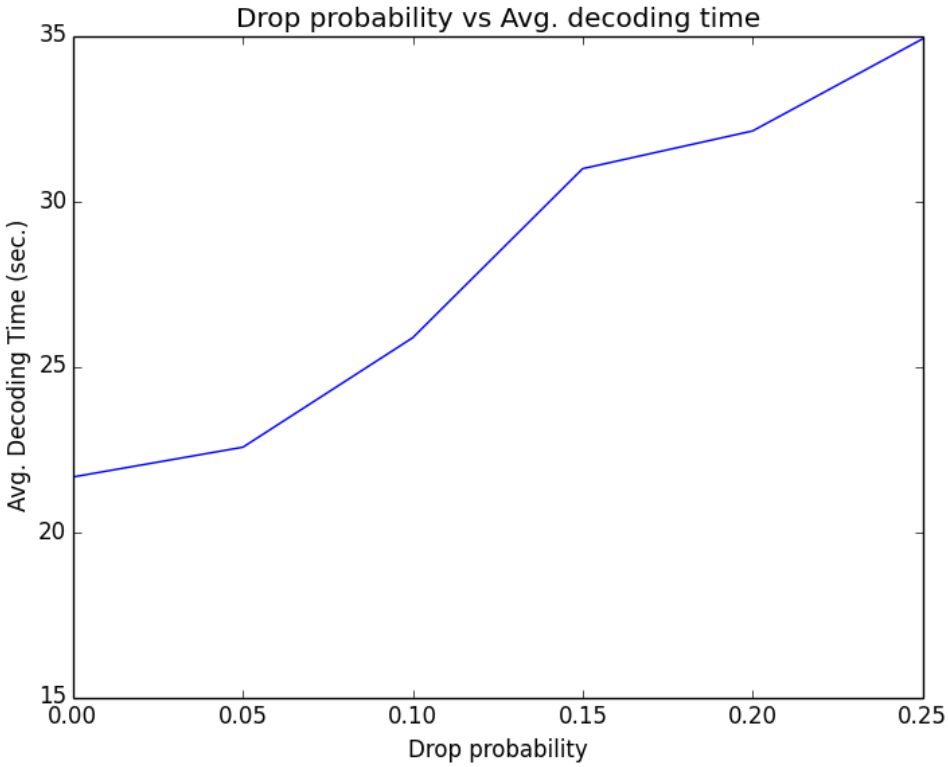


Figure 6.3 Plot of Drop Probability vs. Decoding Time

6.3.3 With respect to number of nodes

Table 6.1 Decoding times with different number of nodes

S.No	No. of nodes	Avg. Decoding Time
1	5	25.93
2	10	32.58
3	15	38.28
4	20	44.76
5	25	55.16
6	30	64.33

Here, the varying **values of number of nodes** are taken along the **x-axis** and the **average decoding time** is taken along the **y-axis**. Figure 6.3.3 shows the values of decoding time averaged over multiple trials plotted against the number of nodes in the network. From the graph, we can see that there is a linear relationship between the number of nodes and the decoding time. Thus the method scales well even for very large networks. It doesn't explode in complexity.

Table 6.1 shows the actual values of average decoding time for networks with different number of nodes.

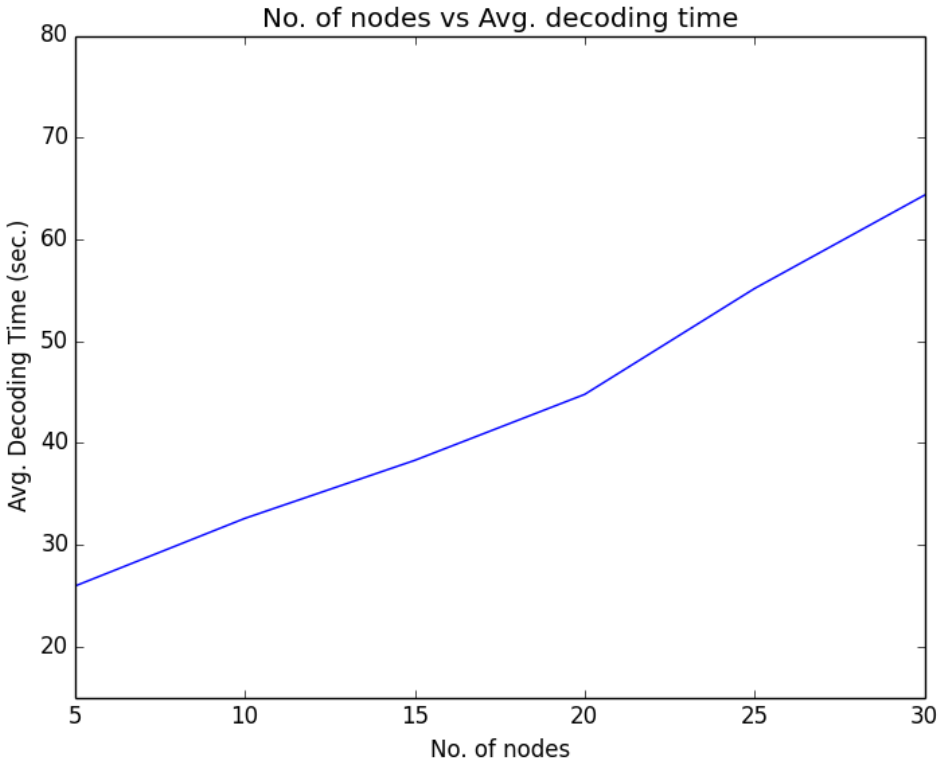


Figure 6.4 Plot of Number of nodes vs. Decoding Time

6.3.4 With treebone and random gossiping

Table 6.2 Decoding times with treebone and random gossiping in mesh

S.No	Size (in MB)	Run 1	Run 2	Run 3	Average
1	1.12	3.25s	3.11s	3.054s	3.138s
2	0.27	3.27s	3.06s	1.75s	2.69s
3	2.16	2.76s	3.21s	2.86s	2.94s
4	0.46	2.29s	3.25s	3.48s	3.01s

The use of treebone and random gossiping in our protocol helped reduce the time it takes for all clients to acquire the transmitted video (decoding time). Random gossiping is used in the mesh structure and speeds up transmission since each node forwards or sends coded packets to only a random subset of its neighbours. It shows good convergence for dense networks. We are able to achieve convergence even in the case of lossy networks (with p = 0.2 being the loss probability along a link). The treebone is constructed with respect to the source and it provides a stable path for pushing data. We obtained the results in Table 6.2 with a k-ary tree for the treebone (k = 4).

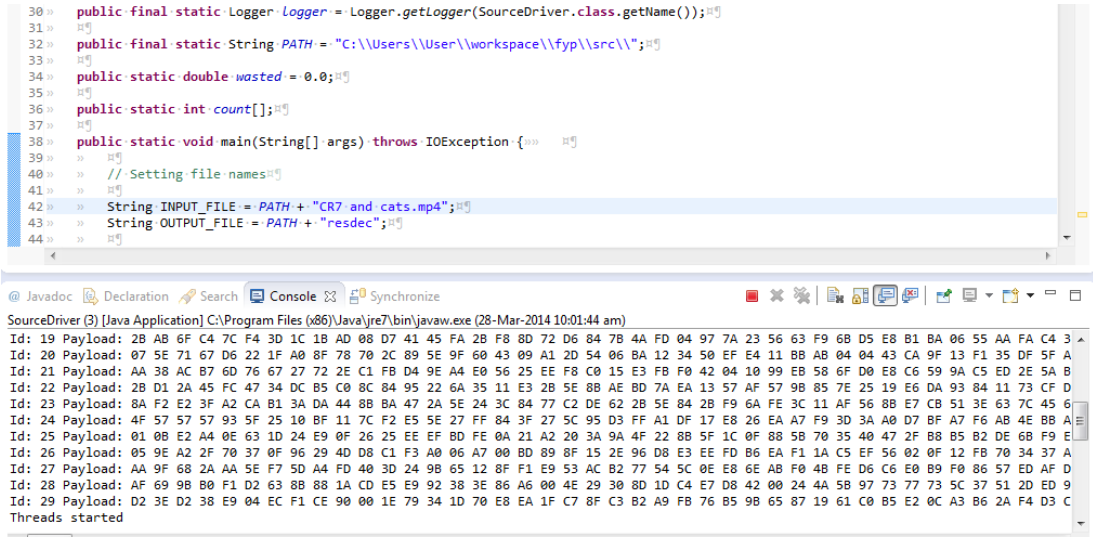


Figure 6.5 Screenshot of running program

Figure 6.3.4 is a screenshot of the running code along with the console.

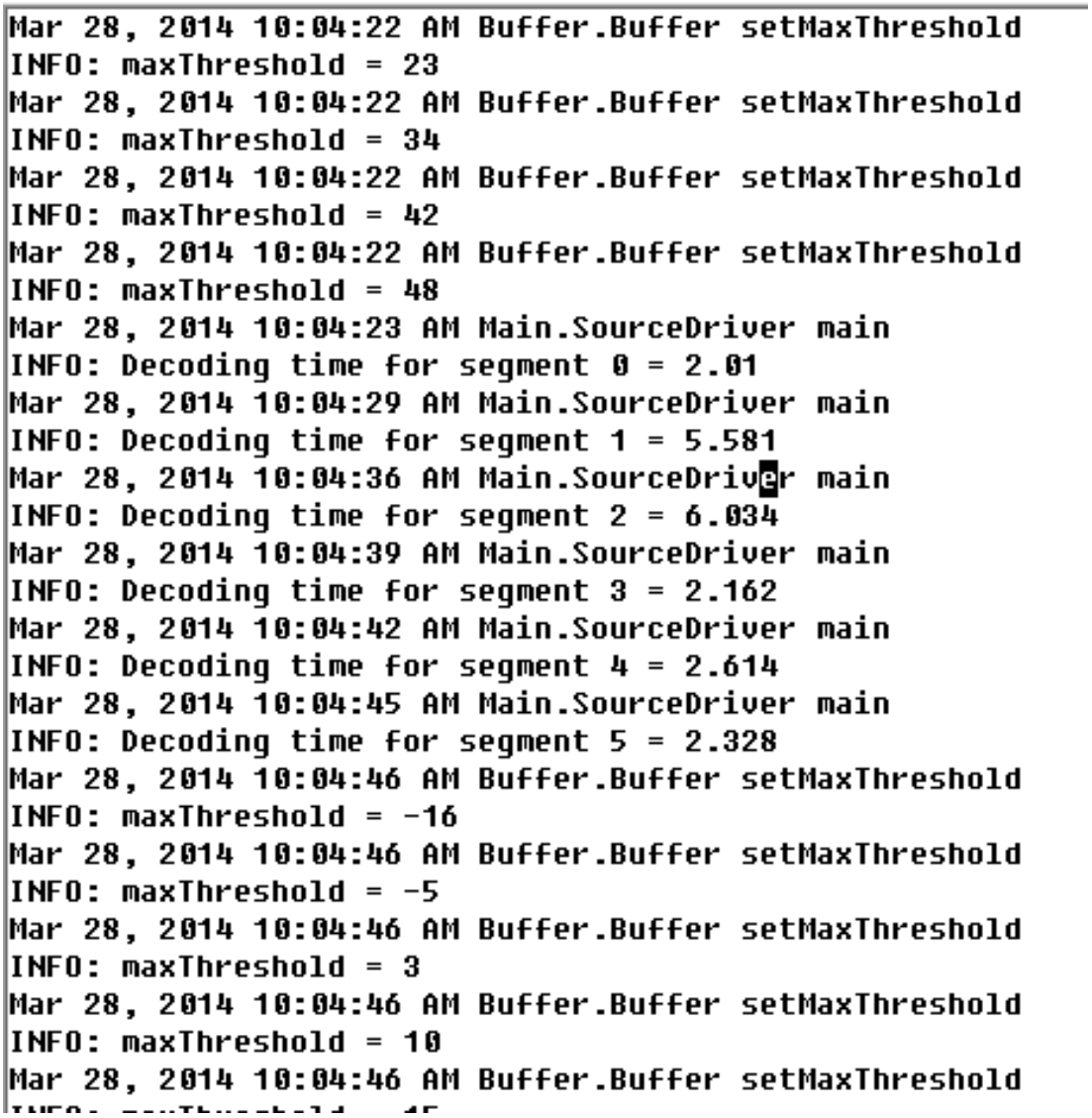


Figure 6.6 Screenshot of log file

Figure 6.3.4 is a screenshot of the log file that is used to dynamically record program-related metrics, like segment-wise decoding time and buffer management parameters, as the video gets transmitted.

CHAPTER 7

CONCLUSIONS

7.1 CONTRIBUTIONS

We have managed to demonstrate the robustness of our protocol through simulation. Our approach is different from previous approaches in that we make use of network coding in combination with a tree-mesh overlay. It could be considered as an extension to mTreebone. Use of network coding makes the approach much more suitable for video multicasting. Also, our method makes use of an algorithm to choose optimal buffer size for SF nodes. This leads to better memory utilisation and thus it can accommodate memory constrained nodes. We have also described methods to handle arrival and departure of peers. This makes our method applicable to multicast in heterogeneous peer-to-peer overlay networks. We also showed that our method performs reasonably well in lossy networks.

7.2 FUTURE WORK

Our future work would focus on extending the proposed method to handle multicasting with multiple sources. Currently, we haven't considered the problem of allocating network resources to a set of connections. We simply assume the availability of dedicated network resources. We would like to make our method more practical by considering the problem of optimal subgraph selection. Find-

ing minimum cost subgraph for multiple connections is NP Hard. But there are good sub-optimal approximations for coding multiple connections simultaneously. Also, we are thinking of using Raptor codes for further protection against erosion of data in packets for transmissions between SF nodes in lossy networks, thereby studying the effect of poor signal-to-noise ratio on our protocol. We would like to explore such techniques to improve our existing method.

APPENDIX A

NETWORK CODING AND ROUTING

A.1 NETWORK CODING

Network coding is about mixing/combining different packets within a multicast session/across sessions so as to improve throughput. The packets are mixed together by performing coding at intermediate nodes in the network. It is suitable for multi-hop wireless networks with multicast connections. It has been shown in [8] that network coding improves the throughput of wireline unicast connections as well. Network coding has been shown to provide improved throughput, security, complexity and robustness.

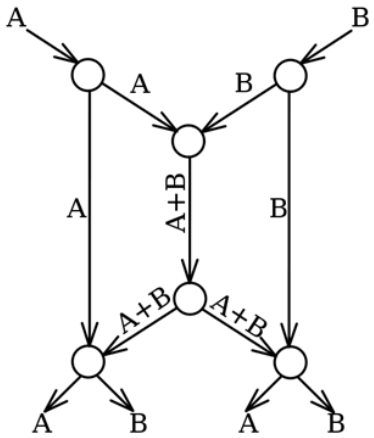


Figure A.1 Butterfly Diagram

Description of network coding from [2] : As shown in Figure A.1, two source nodes (at the top of the picture) have information A and B that must be transmitted to the two destination nodes (at

the bottom). Each edge can carry only a single value at a time.

If only routing were allowed, then the central link would be only able to carry A or B, but not both. Suppose we send A through the centre; then the left destination would receive A twice and not know B at all. Sending B poses a similar problem for the right destination. No routing mechanism can be used to send A and B simultaneously to both destinations.

A and B can be transmitted to both destinations simultaneously by sending the sum of the symbols through the center node. The left destination receives A and A + B, and can calculate B by subtracting the two values. Similarly, the right destination will receive B and A + B, and will also be able to determine both A and B.

It is evident from this example that throughput gains are obtained since with coded networks we perform fewer packet transmissions in multicast connections compared to non-coded networks. Performing optimal routing for multicast connections is a NP-Complete problem in routing networks which requires the construction of Steiner Trees. For coded networks, the problem of optimal routing is transformed into a linear optimization problem, which is computationally feasible. There are security benefits to using network coding since coded packets are not easy to decode if the eavesdropper doesn't have enough source packets. A possible security disadvantage is that a malicious node can perform packet injections and go undetected(pretend to send coded packets). Network coding provides robustness in packet loss/link failures.

A.2 RANDOM LINEAR NETWORK CODING

Nodes transmit random linear combinations of the packets they receive, with coefficients chosen from a Galois field. If the field size is sufficiently large, the probability that the sink(s) will obtain linearly independent packets approaches 1. It should however be noted that, although random network coding has excellent throughput performance, if a receiver obtains an insufficient number of packets, it is unlikely that it can recover any of the original packets. This can be addressed by sending additional random linear combinations until the receiver obtains the appropriate number of packets.

It has been shown that the technique of random linear network coding can be used to achieve maximum throughput according to max-flow min-cut theorem (theoretical upper bound). In [10], this random linear network coding is used alongside random gossiping to maximize information diffusion efficiency given limited bandwidth.

In larger networks, network coding faces the problem of redundancy if all the intermediate nodes are allowed to code. This in turn leads to poor bandwidth utilisation and increased latency at client side. The client will receive packets that are not innovative (whose coding coefficients may be obtained as a linear combination of coefficients of packets already received). This is even more the case with smaller generations of packets (all packets sent from the same source belong to a single generation). [6] showed that it is possible to achieve the same throughput without performing encoding/re-coding at every intermediate node. There is an optimal subset of nodes which when chosen for network coding achieves the minimum delay. The remaining nodes

don't perform any coding and they simply save-and-forward the packets.

A problem with random gossiping is that duplicates should not be sent to clients, and to allow for this, there needs to be regular exchanges of buffer availability maps, which adds significant communication overhead. To mitigate this, Wang et al [15] proposed R2, random push with random network coding, that divides the content of the media stream in a sliding window into generations. Hence the buffer availability maps need only reflect the generations available at a node, and not the individual blocks hence greatly reducing the overhead.

Networks in general face the problem of bit errors in transmission. This is truer in wireless networks rather than wired networks because of the higher probability of interference and noise that introduce errors.

A.3 RANDOM GOSSIPING

Peer-to-peer file distribution systems such as BitTorrent use random gossiping which distributes a subset of the blocks received at each node to a subset of its neighbors. Since peer-to-peer networks are highly dynamic, and peers can join and leave at will, available bandwidth and network topology are highly volatile. It has been shown in [10] that random gossiping is simple to implement, resilient to the level of volatility caused by peer arrivals and departures, and has been shown to have good performance from a theoretical perspective.

An example for random gossiping,

Suppose that we want to find the object that most closely matches some search pattern, within a network of unknown size, but where the computers are linked to one another and where each machine is running a small agent program that implements a gossip protocol.

To start the search, a user would ask the local agent to begin to gossip about the search string. (We're assuming that agents either start with a known list of peers, or retrieve this information from some kind of a shared Web site.)

Periodically, at some rate (let's say ten times per second, for simplicity), each agent picks some other agent at random, and gossips with it. Search strings known to A will now also be known to B, and vice versa. In the next "round" of gossip, A and B will pick additional random peers, maybe C and D. This round-by-round doubling phenomenon makes the protocol very robust, even if some messages get lost, or some of the selected peers are the same or already know about the search string.

On receipt of a search string for the first time, each agent checks its local machine for matching documents.

Agents also gossip about the best match, to date. Thus, if A gossips with B, after the interaction, A will know of the best matches known to B, and vice versa. Best matches will "spread" through the network.

REFERENCES

[1] A. Adams, J. Nicholas, and W. Siadak, "Protocol independent multicast - dense mode (pim-dm): Protocol specification (revised)", URL: http://www.ietf.org/rfc/rfc3973.txt, January.

[2] Rudolf Ahlswede, Ning Cai, Shuo-Yen Robert Li, and Raymond W. Yeung, "Network information flow", *IEEE Trans. on Inf. Theory*, vol. 46, num. 4, pp. 1204–1216, July 2000.

[3] S. Banerjee, B. Bhattacharjee, and C. Kommareddy, "Scalable application layer multicast", In *Proceedings of ACM SIGCOMM 2002*, pp. 205–217. ACM, August 2002.

[4] R. Besharati, M. Bag-Mohammadi, and M. A. Dezfouli, "A topology aware application layer multicast protocol", In *Proc. IEEE CCNC 2010*. IEEE, 2010.

[5] B. Cain, S. Deering, I. Kouvelas, W. Fenner, and A. Thyagarajan, "Internet group management protocol, version 3", URL: http://www.ietf.org/rfc/rfc3376.txt, October.

[6] N. Cleju, N. Thomos, and P. Frossard, "Selection of network coding nodes for minimal playback delay in streaming overlays", *IEEE Transactions on Multimedia*, vol. 13, num. 5, pp. 1103–1115, October 2011.

[7] B. Fenner, M. Handley, H. Holbrook, and I. Kouvelas, "Protocol independent multicast - sparse mode (pim-sm) : Protocol specification (revised)", URL: http://tools.ietf.org/rfc/rfc4601.txt, August.

[8] T. Ho and D. S. Lun, *Network Coding: An Introduction*, PhD thesis, Massachusetts Institute of Technology, 2008.

[9] T. Ho, M. Médard, R. Koetter, D. R. Karger, M. Effros, J. Shi, and D. Leong, "A random linear network coding approach to multicast", *IEEE Trans. Inf. Theory*, vol. 52, pp. 4413–4430, 2006.

[10] B. Liu and D. Niu, "Random network coding in peer-to-peer networks: From theory to practice", *Proceedings of the IEEE*, vol. 99, num. 3, pp. 513–523, March 2011.

[11] M. M. Markou and C. Panayiotou, "Dynamic control and optimization of buffer size in wireless networks", *IEEE Vehicular Technology Conference*, vol. 4, pp. 2162–2166, May 2005.

[12] A. Shokrollahi, "Raptor codes", *IEEE Trans. Inf. Theory*, vol. 52, num. 6, pp. 2551–2567, 2006.

[13] N. Thomos and P. Frossard, "Network coding of rateless video in streaming overlays", *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 20, num. 12, pp. 1834–1847, December 2010.

[14] F. Wang, Y. Xiong, and J. Liu, "mtreebone: A collaborative tree-mesh overlay network for multicast video streaming", *IEEE Trans. Parallel and Distributed Systems*, vol. 21, num. 3, pp. 379–392, March 2010.

[15] M. Wang and B. Li, "R2 : Random push with random network coding in live peer-to-peer streaming", *IEEE J. Sel. Areas Commun.*, vol. 25, num. 9, pp. 1655âĂŞ–1666, December 2007.

[16] X. Zhang, J. Liu, B. Li, and T. P. Yum, "Donet/coolstreaming: A data-driven overlay network for peer-to-peer live media streaming", In *Proc. IEEE INFOCOM 2005*, volume 3, pp. 2102–2111. IEEE, 2005.