Dedicate to my parents, 
Huazhi Sun and Xirong Ma, 
for their love and support. 
In memory of my late grandfather, 
Zhenpu Ma (1927 - 2016).
ACKNOWLEDGMENTS

In my research, I have received assistance from many people. First, I would like to thank my advisor, Dr. Dapeng Oliver Wu for his support, advice, guidance, and good wishes. Dr. Wu has had a profound influence not only as my graduate advisor in UF, but also on my life. His availability at all times including weekends, dedication towards work, professional integrity, and pursuit of perfection helped me become a better individual. Dr. Wu has made it his responsibility to make sure that I, as well as all of his other students, have had the financial support we need to accomplish our goals. I am grateful to him for the freedom and flexibility he gave me throughout my Ph.D. study.

My gratitude goes to the committee members (in alphabetical order), Dr. Ahmed Helmy, Dr. Janise McNair, and Dr. John M. Shea, for their invaluable comments, productive suggestions, and the time for reading the draft of my dissertation.

My special thanks go to Dr. Huazi Zhang, who worked with me on many critical problems in my most difficult days, and contributed several distinctive insights to my research. I sincerely thank Dr. Bo Yan of Fudan University, who was my advisor in the college. It was him who led me on the way to research and encouraged me to pursue the Ph.D. degree.

During the course of my research, I have submitted several papers to peer reviewed conferences and journals. The anonymous reviewers have provided valuable insights, pointers to literature, and criticisms that I have used to make my research stronger.

I would like to thank my parents for their continuous support on my research path. They are the source of my strength.

My thoughts go to my late grandfather, Zhenpu Ma (1927 - 2016), who passed away one day before my dissertation defense. He was a learned automobile engineer and a skillful craftsman. He practiced the spirits of engineer in his whole life, and made the society a better place. His spirits of wisdom, curiosity, honesty and dedication edify me as a child, and inspire me to become an engineering Ph.D.. I am sad that he could not see me graduate, but his memory will be with me always.
TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACKNOWLEDGMENTS</td>
<td>4</td>
</tr>
<tr>
<td>LIST OF TABLES</td>
<td>9</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>10</td>
</tr>
<tr>
<td>ABSTRACT</td>
<td>12</td>
</tr>
<tr>
<td>CHAPTER</td>
<td></td>
</tr>
<tr>
<td>1 INTRODUCTION</td>
<td>14</td>
</tr>
<tr>
<td>1.1 Background</td>
<td></td>
</tr>
<tr>
<td>1.1.1 Fountain Codes</td>
<td>15</td>
</tr>
<tr>
<td>1.1.2 Network Coding</td>
<td>16</td>
</tr>
<tr>
<td>1.2 Challenges and Proposed Solutions</td>
<td>17</td>
</tr>
<tr>
<td>1.2.1 Challenges</td>
<td></td>
</tr>
<tr>
<td>1.2.2 Solutions from Channel Coding</td>
<td>18</td>
</tr>
<tr>
<td>1.2.3 Solutions from Network Coding</td>
<td>18</td>
</tr>
<tr>
<td>1.2.4 Solutions from Source Coding</td>
<td>18</td>
</tr>
<tr>
<td>1.3 Organization</td>
<td>19</td>
</tr>
<tr>
<td>2 CHANNEL CODING STRATEGY: DELAY-AWARE FOUNTAIN CODES</td>
<td>20</td>
</tr>
<tr>
<td>2.1 Delay-aware Sliding Window Fountain Codes</td>
<td>21</td>
</tr>
<tr>
<td>2.1.1 Time-based Window Size vs. Packet-based Window Size</td>
<td>23</td>
</tr>
<tr>
<td>2.1.2 Sliding Window vs. Block Coding</td>
<td>25</td>
</tr>
<tr>
<td>2.2 Optimal Window-wise Sampling Strategy</td>
<td>27</td>
</tr>
<tr>
<td>2.2.1 Per-frame Optimization Scheme</td>
<td>30</td>
</tr>
<tr>
<td>2.2.1.1 Problem description</td>
<td>32</td>
</tr>
<tr>
<td>2.2.1.2 Solution</td>
<td>32</td>
</tr>
<tr>
<td>2.2.1.3 Computational complexity</td>
<td>33</td>
</tr>
<tr>
<td>2.2.2 Slope-only Optimization Scheme</td>
<td>33</td>
</tr>
<tr>
<td>2.2.2.1 Problem description</td>
<td>38</td>
</tr>
<tr>
<td>2.2.2.2 Solution</td>
<td>39</td>
</tr>
<tr>
<td>2.2.2.3 Computational complexity</td>
<td>39</td>
</tr>
<tr>
<td>2.3 Implementation Details</td>
<td>40</td>
</tr>
<tr>
<td>2.3.1 Evaluation Criteria</td>
<td>40</td>
</tr>
<tr>
<td>2.3.2 Degree Distribution</td>
<td>40</td>
</tr>
<tr>
<td>2.3.3 Warm-up/Cool-down Period</td>
<td>42</td>
</tr>
<tr>
<td>2.4 System Design</td>
<td>42</td>
</tr>
<tr>
<td>2.4.1 Packet Structure</td>
<td>42</td>
</tr>
<tr>
<td>2.4.2 DAF Encoder</td>
<td>43</td>
</tr>
<tr>
<td>2.4.3 DAF Decoder</td>
<td>45</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>2.4.4 General Framework of Fountain Code Systems</td>
<td>45</td>
</tr>
<tr>
<td>2.5 Real-time Algorithm for DAF</td>
<td>46</td>
</tr>
<tr>
<td>2.5.1 Model Predictive Control</td>
<td>47</td>
</tr>
<tr>
<td>2.5.2 MPC-based Optimal Control for DAF Codes: the Offline Version</td>
<td>47</td>
</tr>
<tr>
<td>2.5.2.1 Horizon</td>
<td>48</td>
</tr>
<tr>
<td>2.5.2.2 Objective function</td>
<td>49</td>
</tr>
<tr>
<td>2.5.2.3 Process model</td>
<td>49</td>
</tr>
<tr>
<td>2.5.2.4 Computational complexity</td>
<td>51</td>
</tr>
<tr>
<td>2.5.3 Online Algorithm with Bit Rate Prediction</td>
<td>51</td>
</tr>
<tr>
<td>2.5.4 Optimization Results and Evaluations</td>
<td>53</td>
</tr>
<tr>
<td>2.5.4.1 ASP results comparison</td>
<td>53</td>
</tr>
<tr>
<td>2.5.4.2 Measuring the quality of ASP</td>
<td>54</td>
</tr>
<tr>
<td>2.6 Unequal Error Protection for DAF</td>
<td>56</td>
</tr>
<tr>
<td>2.6.1 Importance Profile</td>
<td>58</td>
</tr>
<tr>
<td>2.6.2 Applying UEP to Optimization Objective</td>
<td>58</td>
</tr>
<tr>
<td>2.6.3 Work with MPC-based DAF</td>
<td>59</td>
</tr>
<tr>
<td>2.6.4 Optimization Results</td>
<td>60</td>
</tr>
<tr>
<td>2.7 Simulation Experiments and Performance Evaluations</td>
<td>61</td>
</tr>
<tr>
<td>2.7.1 Simulator Setup</td>
<td>61</td>
</tr>
<tr>
<td>2.7.2 Schemes for Comparison</td>
<td>62</td>
</tr>
<tr>
<td>2.7.3 Performance Evaluation for DAF and DAF-L</td>
<td>64</td>
</tr>
<tr>
<td>2.7.3.1 Evaluate the effect of delay and code rate</td>
<td>64</td>
</tr>
<tr>
<td>2.7.3.2 Evaluate the effect of packet loss rate</td>
<td>68</td>
</tr>
<tr>
<td>2.7.3.3 Evaluate the effect of step size</td>
<td>70</td>
</tr>
<tr>
<td>2.7.4 Performance Evaluation for MPC-based DAF</td>
<td>71</td>
</tr>
<tr>
<td>2.7.4.1 Evaluate the effect of horizon length</td>
<td>71</td>
</tr>
<tr>
<td>2.7.4.2 Compare with the other schemes</td>
<td>73</td>
</tr>
<tr>
<td>2.7.5 Performance Evaluation for UEP-based DAF</td>
<td>74</td>
</tr>
<tr>
<td>2.8 Summary of Channel Coding Strategy</td>
<td>75</td>
</tr>
<tr>
<td>3 NETWORK CODING STRATEGIES: FUN AND MIMO FUN CODES</td>
<td>78</td>
</tr>
<tr>
<td>3.1 FUN Coding Description</td>
<td>78</td>
</tr>
<tr>
<td>3.1.1 FUN Overview</td>
<td>78</td>
</tr>
<tr>
<td>3.1.2 FUN-1</td>
<td>79</td>
</tr>
<tr>
<td>3.1.2.1 Precoding of FUN-1</td>
<td>80</td>
</tr>
<tr>
<td>3.1.2.2 Outer code of FUN-1</td>
<td>80</td>
</tr>
<tr>
<td>3.1.2.3 Inner code of FUN-1</td>
<td>80</td>
</tr>
<tr>
<td>3.1.2.4 XOR coding of FUN-1</td>
<td>81</td>
</tr>
<tr>
<td>3.1.2.5 Decoding of FUN-1</td>
<td>82</td>
</tr>
<tr>
<td>3.1.3 FUN-2</td>
<td>82</td>
</tr>
<tr>
<td>3.1.3.1 Inner code of FUN-2</td>
<td>82</td>
</tr>
<tr>
<td>3.1.3.2 Decoding of FUN-2</td>
<td>83</td>
</tr>
<tr>
<td>3.2 MIMO FUN Description</td>
<td>84</td>
</tr>
<tr>
<td>3.2.1 An Intuitive Example of MIMO FUN</td>
<td>84</td>
</tr>
</tbody>
</table>
3.2.2 Algorithm of MIMO FUN .................................................. 86
3.2.3 Implementation Details .................................................. 87
  3.2.3.1 Ad-hoc route selection ............................................. 87
  3.2.3.2 Global optimization ................................................. 88
  3.2.3.3 Exploiting structure of RLNC to improve XOR type schemes . 88
  3.2.3.4 Relay performs RLNC recoding and local feedback ............ 88
  3.2.3.5 Partial decoding in relay nodes .................................. 88
3.2.4 Example 1: Four-wheel Topology .................................... 89
3.2.5 Example 2: Dumbbell Topology ...................................... 89
3.3 Comparison Schemes and Numerical Results .............................. 91
  3.3.1 Naive Broadcast Scheme w/ ACK and Retransmission ............. 91
  3.3.2 COPE-like Algorithm ................................................ 93
  3.3.3 CORE Scheme ....................................................... 93
  3.3.4 MIMO FUN ........................................................ 94
  3.3.5 Simulation Results .................................................. 94
3.4 Experimental Results .................................................... 95
3.5 Summary of Network Coding Strategy .................................. 97

4 SOURCE CODING STRATEGY: RATE CONTROL FOR CLOUD GAMING VIDEO 98
4.1 Cloud Gaming ................................................................. 98
4.2 Related Work ............................................................... 100
4.3 ROI and Key Frame Patterns for Gaming Video ......................... 101
  4.3.1 ROI Extraction ....................................................... 102
    4.3.1.1 Downscaling ................................................ 105
    4.3.1.2 Normalization ............................................... 105
    4.3.1.3 Boundary .................................................... 106
    4.3.1.4 Mean filtering ............................................. 106
  4.3.2 Key Frame Detection ............................................... 107
    4.3.2.1 Intra frame position allocation ............................. 107
    4.3.2.2 Control of frame rate .................................... 108
4.4 Rate Control for H.264/AVC Based Video ................................ 108
  4.4.1 Preference Adjustment ............................................. 108
  4.4.2 Relative Importance Ratio ...................................... 109
  4.4.3 Frame-level Bit Allocation .................................... 109
  4.4.4 MB-level Bit Allocation ........................................ 110
4.5 QoE-based Evaluation and Simulation Results .......................... 111
  4.5.1 QoE-based Quality Evaluation ................................... 112
  4.5.2 Simulation Results ................................................ 113
  4.5.3 Computational Complexity Comparisons ........................... 116
4.6 Summary of Source Coding Strategy .................................. 117

5 CONCLUSIONS ................................................................. 119

REFERENCES ................................................................. 121
## LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-1</td>
<td>Definitions of the notations for the variables related to fountain codes.</td>
<td>22</td>
</tr>
<tr>
<td>2-2</td>
<td>Definitions of the notations for the variables related to video coding.</td>
<td>23</td>
</tr>
<tr>
<td>2-3</td>
<td>Decoding ratio comparisons between different schemes with variant settings.</td>
<td>65</td>
</tr>
<tr>
<td>2-4</td>
<td>IDR comparisons between different schemes for variant sequences under PLR of 5%, 10% and 15%.</td>
<td>69</td>
</tr>
<tr>
<td>2-5</td>
<td>IDR comparisons between different SWFC schemes with variant Δt. PLR = 10%.</td>
<td>71</td>
</tr>
<tr>
<td>2-6</td>
<td>IDR comparisons of DAF-based schemes using different horizon lengths.</td>
<td>72</td>
</tr>
<tr>
<td>2-7</td>
<td>IDR comparisons of proposed schemes and existing video streaming algorithms.</td>
<td>75</td>
</tr>
<tr>
<td>2-8</td>
<td>Decoding ratio and PSNR comparisons.</td>
<td>76</td>
</tr>
<tr>
<td>3-1</td>
<td>Performance comparison between different topologies using MIMO FUN.</td>
<td>96</td>
</tr>
<tr>
<td>3-2</td>
<td>Performance comparison between different schemes.</td>
<td>97</td>
</tr>
<tr>
<td>4-1</td>
<td>Pros and cons of cloud gaming.</td>
<td>100</td>
</tr>
<tr>
<td>4-2</td>
<td>Results comparisons between JVT-G012 Li et al. (2003b), Liu et al. (2008) (with and without our proposed ROI), Shen et al. (2013) with our proposed ROI and proposed algorithm for 4CIF (704 × 576 pixels) video sequences.</td>
<td>114</td>
</tr>
<tr>
<td>4-3</td>
<td>Computational complexity comparisons between JVT-G012 Li et al. (2003b), Shen et al. (2013) and proposed algorithm for 4CIF (704 × 576 pixels) video sequences.</td>
<td>117</td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>------------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>2-1</td>
<td>Comparison of coding structure between block coding scheme, and sliding window scheme.</td>
<td>26</td>
</tr>
<tr>
<td>2-2</td>
<td>Compare block vs. sliding window.</td>
<td>27</td>
</tr>
<tr>
<td>2-3</td>
<td>The resulting ASP using different SWFC schemes for video sequence <em>foreman</em>.</td>
<td>28</td>
</tr>
<tr>
<td>2-4</td>
<td>The optimization result of sampling distributions for each window of <em>foreman</em> using per-frame optimization scheme.</td>
<td>33</td>
</tr>
<tr>
<td>2-5</td>
<td>The sampling distributions when slope factor $a = 1$, $a = 0$, and $a = -1$.</td>
<td>35</td>
</tr>
<tr>
<td>2-6</td>
<td>The distribution functions when slope factor $a = 1$, $a = 0$, and $a = -1$.</td>
<td>36</td>
</tr>
<tr>
<td>2-7</td>
<td>An example of sampling distribution for each frame within a window.</td>
<td>37</td>
</tr>
<tr>
<td>2-8</td>
<td>The optimization result of sampling distributions for each window of <em>foreman</em> using slope-only optimization scheme.</td>
<td>39</td>
</tr>
<tr>
<td>2-9</td>
<td>The definition of warm-up/cool-down periods.</td>
<td>42</td>
</tr>
<tr>
<td>2-10</td>
<td>The packet structure and the header structure of a DAF packet.</td>
<td>43</td>
</tr>
<tr>
<td>2-11</td>
<td>The flowchart of the DAF encoder.</td>
<td>44</td>
</tr>
<tr>
<td>2-12</td>
<td>The flowchart of the DAF decoder.</td>
<td>45</td>
</tr>
<tr>
<td>2-13</td>
<td>A flowchart of MPC-based delay-aware fountain codes.</td>
<td>48</td>
</tr>
<tr>
<td>2-14</td>
<td>Bit rate for <em>foreman</em> and the optimized sampling distributions for each window using different optimization schemes.</td>
<td>53</td>
</tr>
<tr>
<td>2-15</td>
<td>Resulting ASP using different DAF-based schemes for <em>foreman</em>.</td>
<td>54</td>
</tr>
<tr>
<td>2-16</td>
<td>The coverage ratios of the resulting ASP obtained by different schemes for <em>foreman</em>.</td>
<td>55</td>
</tr>
<tr>
<td>2-17</td>
<td>The bit rate of <em>foreman</em> and the importance profile when GOP size is 40.</td>
<td>60</td>
</tr>
<tr>
<td>2-18</td>
<td>The result of sampling distributions for each window of <em>foreman</em> using DAF and proposed scheme.</td>
<td>61</td>
</tr>
<tr>
<td>2-19</td>
<td>Relations of IDR vs. code rate of CIF sequence <em>foreman</em> when $T_{Delay} = 0.5$, 1, 1.5, and 1.83 seconds. Five sliding window schemes are compared. $PLR = 10%$. $\Delta t = 1$.</td>
<td>66</td>
</tr>
<tr>
<td>2-20</td>
<td>Relations of IDR vs. delay of CIF sequence <em>foreman</em> when $C = 1.0$, 0.9, 0.85, and 0.75. Five sliding window schemes are compared. $PLR = 10%$. $\Delta t = 1$.</td>
<td>67</td>
</tr>
</tbody>
</table>
2-21 Comparison of the decoding ratios of CIF sequence *foreman*. Five delay-aware fountain code schemes are compared. There are two dimensions of variables, $T_{\text{Delay}}$ and $C$. $\text{PLR} = 10\%$. $\Delta t = 1$. ................................................. 68

2-22 Comparisons of IDR curves for *foreman* when fixing delay $T_{\text{Delay}} = 1.2$ s and code rate $C = 0.77$. Various values of $H$ are chosen for MPC-O and MPC-M. ........... 72

2-23 Resulting IDR surfaces of online window-based fountain codes schemes for *foreman*. 74

3-1 Three-wheel topology. ................................................................. 85

3-2 Example topologies. ................................................................. 87

3-3 Decoding coefficients when dof is full. ......................................... 90

3-4 Generic wheel topology. ............................................................. 91

3-5 Simulation vs. theoretical results. ............................................... 95

4-1 Cloud gaming overview. .................................................................. 99

4-2 Examples of the ROI of different types of games. The red masks represent ROI. ... 103

4-3 The flowchart of an RC-aware cloud gaming system demonstrating the relationship between the modules. ................................................................. 104

4-4 Example of ROI detection (blue areas) for an RPG game. The brighter area indicates it is more important. ................................................................. 104

4-5 Example of ROI detection (blue areas) for an RAC game. The brighter area indicates it is more important. ................................................................. 105

4-6 Examples of a sudden content change. Two consecutive frames are shown but there is a big change in the content. ......................................................... 107

4-7 47th frame from the sequence “RAC 02”. Compressed at 800 kbps. Comparison between JVT-G012, Liu et al. (2008) and proposed method. .......................... 115

4-8 429th frame from the sequence “FPS 01”. Compressed at 300 kbps. This is a key frame. Comparison between JVT-G012 and proposed method. ....................... 116
With the prevalence of smart mobile devices, the traffic load within the Internet has shifted away from non-multimedia data to multimedia traffics, particularly, the video content. The explosive demand of on-line video from smart mobile devices poses unprecedented challenges to delivering high quality of experience (QoE) over wireless networks. Streaming high-definition video with low delay is difficult mainly due to (i) the stochastic nature of wireless channels, (ii) the fluctuating videos bit rate and (iii) the limited bandwidth of wireless networks.

Apparently, in order to effectively solve the aforementioned problems, it is inadequate to focus on any single aspect, since it needs joint optimization of video coding, networking and communication techniques. In my Ph.D. study, I put my effort into improving QoE of video communication over wireless networks towards the following three aspects:

First, in order to deal with the packet losses in wireless network, I propose a novel delay-aware fountain coding (DAF) technique that integrates channel coding and video coding. In Sun et al. (2016), I reveal that the fluctuation of video bit rate can also be exploited to further improve fountain codes for wireless video streaming.

Second, because the broadcast nature of the wireless network video distribution technologies enables scalable and bandwidth-efficient media delivery to a larger number of users via a common physical channel, network coding will also bring significant improvement of throughput of wireless broadcast network. Joint FoUntain coding and Network coding
(FUN) Huang et al. (2014a); Zhang et al. (2016) and MIMO FUN are proposed to boost information spreading over multi-hop and mesh lossy networks. The novelty of our FUN approach lies in combining the best features of fountain coding, intra-session network coding, and cross-next-hop network coding.

Last but not least, in order to stabilize the bit rate generated by video codec, rate control techniques of video codec should be utilized. My previous work Sun and Wu (2015) makes use of the unique characteristics of human visual system (HVS) of video game players to assist gaming video rate control.
CHAPTER 1
INTRODUCTION

1.1 Background

The internet has witnessed the explosive growth of video traffic over the recent decade, especially after the prevalence of smart mobile devices and surveillance cameras. At the same time, internet nowadays relies more and more on wireless networks. The most ambitious on-going projects include Google’s Project Loon and Facebook Aquila, which are both to try to beam the Internet down from the skies wirelessly, making it ubiquitous around the globe.

Some of the most popular mobile video applications provide the pre-recorded video streams such as YouTube and Netflix, and others provide live video communications such as Skype and FaceTime. It is believed that in the near future, an increasing number of individual users will become video content provider, by sharing videos on social networks, sending video/audio instant messages, holding video conferences, streaming home surveillance data, etc. As a result, a huge amount of multimedia contents will be generated and consumed.

Another important area of interest would be Internet of Things (IoT). Because IoT devices with video-based applications provide the high accuracy and convenience that no simple sensor in the past can provide, innovative applications of real-time video communication over wireless networks could touch people’s lives in profound and different ways. For example, it can enable the first responders to evaluate the severity of an emergency situation before they arrive on the scene through the real-time video provided by the caller’s smart phone. The municipal governments can also monitor the road traffic on real-time through the cameras mounted on the drones that fly over wherever the traffic jams.

On the other hand, the prevalent smart mobile devices make these contents more accessible to people than ever. Thanks to the evolution of communication technologies, wireless networks, such as 3G/4G, LTE, WiFi, etc., are generally available in our daily lives. However, despite the progress, the stochastic nature of wireless channels still persist: its vulnerability to channel noise, inter-user interference and low data rate under mobility. The
problems easily deteriorate in video-dominant applications where the requirement on channel quality is the highest. As a result, how to stream videos with low delay, stable data rate and high quality raises formidable challenges in communication society.

1.1.1 Fountain Codes

By far, the most commonly used multimedia streaming protocols are based on UDP, due to its simplicity. However, since UDP does not provide loss recovery functionality, its performance is not satisfactory in highly lossy wireless channels. In order to deal with packet losses, TCP is used in several video streaming protocols, such as TCP-based real-time transport protocol (RTP) Perkins (2003). TCP uses error detecting code to determine whether a packet is damaged during transmission, and uses automatic repeat request (ARQ) protocol to recover the damaged packets. Although TCP can recover the lost packet, its control mechanisms are designed for wired network. In wireless networks, its disadvantages are obvious: the packets required by acknowledgments (ACK) and retransmissions take extra bandwidth of the channels; it is not suitable for broadcast/multicast; the fluctuating bit rate of video data may cause burst loss. It should be pointed out that, although rate control techniques in video coding are widely adopted to stabilize output bit rate, they may inevitably lead to video quality degradation and fluctuation He et al. (2005).

Fortunately, the computational capability of the mobile devices has improved dramatically, allowing the implementation of more sophisticated encoding/decoding and signal processing techniques in exchange for higher performance. As a result, forward error correction (FEC) erasure codes are extensively revisited in multimedia transmission. One important class of FEC codes are fountain codes Byers et al. (2002), such as Luby transform (LT) code Luby (2002) and Raptor code Shokrollahi (2006). Fountain codes are ideal for wireless video streaming for the following reasons.

1. Retransmission-free property: Because fountain codes belong to FEC, it will reconstruct the original data using the redundancy sent by the sender, without demanding ACK or retransmission.
2. **Efficient broadcast/multicast**: In wireless broadcasting/multicasting, the recipients may receive different subsets of the transmitted packets due to channel diversity. The benefit of fountain codes is that different receivers can decode from different subsets of received packets as long as the number of received packets exceed a threshold.

3. **Ratelessness**: Fountain codes are rateless codes, so the source data can be encoded at any rate depending on the channel condition. This feature allows the actual transmission rate to automatically approach channel capacity without the use of channel estimation.

### 1.1.2 Network Coding

Although fountain codes could effectively recover the lost packets, they are not necessarily optimal due to the accumulate packet losses over multiple hops. Take an $L$-hop network with per-hop packet loss ratio $\epsilon$ for example, the end-to-end throughput is upper bounded by $(1 - \epsilon)^L$, which may drop to zero as the number of hops $L \rightarrow \infty$. Network coding Ahlswede et al. (2000); Li et al. (2003a); Koetter and Médard (2003); Ho et al. (2003) overcomes the aforementioned drawback through introducing redundancy at relay nodes. Specifically, a relay node performs random linear network coding (RLNC) by combining and recoding the packets it has received. RLNC can achieve an end-to-end throughput of $1 - \epsilon$ for the same $L$-hop lossy network Yang and Yeung (2014).

In spite of the throughput gain, RLNC suffers from high computational complexity and excessive coefficient overhead Yang et al. (2014). To reduce complexity, the packets of a file is partitioned into non-overlapping or overlapping subsets (or segments Wang and Li (2007), generations Chou et al. (2003); Li et al. (2011), blocks Park et al. (2006), batches Chachulski et al. (2007), trunks Heidarzadeh and Banihashemi (2010); Tang et al. (2012); Yang and Tang (2014)), and coding is restricted within each subset. Alternatively, a cross-next-hop network coding architecture called COPE Katti et al. (2006) was proposed to recover the combined packets at next-hop relay nodes, but not end nodes, which also leads to significant complexity reduction.

Recently, several joint fountain coding and network coding schemes have been proposed to strike a good balance between throughput and complexity. In this sense, both fixed Silva et al. (2009) and tunable sparsity levels Feizi et al. (2012, 2014) yield satisfying results.
Meanwhile, several approaches Maymounkov et al. (2006); Lun et al. (2008); Mahdaviani et al. (2012); Yang and Yeung (2014); Huang et al. (2014b) employed two-layered joint coding to achieve the same goal. Specifically, the source node uses erasure codes as the outer codes to encode the native packets, and each relay node further recodes these coded packets using \textit{intra-session network coding} (where coding is restricted to one network flow) as the inner code. In Maymounkov et al. (2006); Lun et al. (2008); Mahdaviani et al. (2012), the outer codes are block code, random linear erasure code and a fixed-rate version of the Raptor code, respectively. In Batched Sparse (BATS) codes Yang and Yeung (2014), a rateless outer code is employed through a matrix generalization of a fountain code. From our perspective, BATS is the best joint coding scheme so far due to its (i) efficient belief propagation (BP) decoding, and (ii) robustness against packet loss.

1.2 Challenges and Proposed Solutions

1.2.1 Challenges

In my Ph.D. study, I aim to solve three important problems faced by multimedia communication over networks.

The first one is to deal with the fluctuation of bit rate generated by video codec. It is proven that if a user want to consume videos with stable visual quality, the generated bit rates must be fluctuating. However, it is not a desired property because it may cause burst losses for both wired and wireless networks.

The second one is to deal with the lost data that dropped by network. Especially for wireless networks, packet loss is a very common phenomenon because of its stochastic nature. As a result, how to recover the lost packet is crucial to improve video quality.

The third one is to make use of the broadcasting nature of wireless channels. Because most wireless networks use broadcast channel to transmit data, this is a positive property that could potentially improve the throughput if used correctly.

Apparently, in order to effectively solve the aforementioned problems, it is inadequate to focus on any single aspect, since it needs joint optimization of video coding, networking and
communication techniques. I put my effort into improving QoE of video communication over lossy networks towards the following three aspects.

1.2.2 Solutions from Channel Coding

In order to deal with the packet losses in wireless network and to accommodate fluctuating video bit rates into stable channel data rate, I propose a novel delay-aware fountain code scheme for video streaming, that deeply integrates channel coding and video coding. The proposed schemes take advantages of all the benefits of fountain codes, and address the issues faced by delay-aware applications at the same time. The basic idea that gives fountain codes the delay-awareness is to segment the file into many overlapping data windows and transmit them sequentially, as proposed in Bogino et al. (2007), which is called sliding window fountain codes (SWFC). The basic structure of the proposed scheme is based on SWFC, but our novelty lies in that we do not treat the sliding windows as homogeneous, which, according to existing methods, have fixed length and uniform sampling distribution. According to a novel performance metric, i.e., in-time decoding ratio, which better reflects the real video watching experience, our methods significantly improve the performance of fountain codes in the wireless video streaming setting.

1.2.3 Solutions from Network Coding

Because the broadcast nature of the wireless network video distribution technologies enables scalable and bandwidth-efficient media delivery to a larger number of users via a common physical channel, network coding will also bring significant improvement of throughput of wireless broadcast network. Joint FoUntain coding and Network coding (FUN) Huang et al. (2014a); Zhang et al. (2016) is proposed to boost information spreading over multi-hop lossy networks. The novelty of our FUN approach lies in combining the best features of fountain coding, intra-session network coding, and cross-next-hop network coding.

1.2.4 Solutions from Source Coding

In order to stabilize the bit rate generated by video codec, rate control techniques of video codec should be utilized. My work Sun and Wu (2015) makes use of the unique characteristics
of human visual system (HVS) of video game players to assist gaming video rate control. Discussions about the characteristics of game players’ HVS are conducted. Then, some schemes of extracting region of interest and key frames from gaming videos are raised. Based on that, a low-complexity Macro-block level rate control scheme is proposed based on region of interest and scene-change detection. Since the proposed work is the first one to solve this problem, it shows great potential for the development of the cloud gaming industry.

1.3 Organization

Chapter 2 presents a novel delay-aware fountain code (DAF) scheme for video streaming, which deeply integrates channel coding and video coding.

Chapter 3 presents FUN codes and MIMO FUN, a wireless meshed networks protocol designed to optimize throughput of wireless multi-hop and mesh networks.

Chapter 4 proposes some novel strategies for rate control of cloud gaming video based on Region of Interests (ROI).

Chapter 5 concludes the dissertation by summarizing the contributions.
CHAPTER 2  
CHANNEL CODING STRATEGY: DELAY-AWARE FOUNTAIN CODES

As introduced in Chapter 1, fountain codes are very suitable for data transmission over lossy networks. However, the traditional fountain codes are initially designed for achieving the complete decoding of the entire original file. The time delay of the transmission depends on the size of the whole file and the network condition. That means if a video is transmitted using traditional fountain codes, users cannot watch it until the whole video file is successfully decoded.

However, a lot of video streaming applications are delay-aware and loss-tolerant, which means (i) the time interval between video being generated and being played can not exceed a certain threshold; and (ii) partial decoding is tolerable, albeit higher decoding ratio is more desirable. Besides, because of the delay awareness, the decoding of every packet has an expiration time, implying that the code to be implemented cannot be truly “rateless”. But we still need the good features from fountain codes, e.g., robustness to packet loss and adaptability to channel fluctuation.

It is also noteworthy that different video streaming applications have different levels of delay tolerance. Some applications are very delay-sensitive, such as real-time video chat, cloud gaming, etc., all of which are characterized by bi-directional communication. Some uni-directional applications, on the other hand, have a loose tolerance on end-to-end delay, such as TV broadcasting, Internet live streams. The others, such as the streaming of pre-recorded content, are the least sensitive to delay.

In this chapter, we propose a novel channel coding strategy, called Delay-Aware Fountain code (DAF) for video streaming, that deeply integrates channel coding and video coding. The proposed schemes take advantages of all the benefits of fountain codes, and address the issues faced by delay-aware applications at the same time.
2.1 Delay-aware Sliding Window Fountain Codes

In order to introduce delay awareness into fountain codes, the most intuitive solution is to partition the video file into fixed-length data blocks, separately encode them, and transmit sequentially. Hence, a user can play the currently decoded blocks without having to wait until the entire file is downloaded. We call this method as the block coding scheme.

From the perspective of video transmission, a smaller block size is preferred, because the bigger block size leads to larger latency. From the perspective of the fountain codes, however, the block size needs to be as big as possible to maintain a smaller coding overhead Liva et al. (2010). The fundamental trade-off between video watching experience and coding performance is crucial for the design of delay-aware fountain codes.

Our work focuses on a deep integration of fountain codes and video coding, hence the concepts in both fountain codes and video coding will be constantly referred. For the sake of convenience, we define two sets of variables as in Table 2-1 and 2-2. Table 2-1 defines the variables related to fountain codes, and Table 2-2 defines the properties related to video coding.

Two basic parameters of fountain codes are packet size $P$ and data rate $R$, whose definitions are given in Table 2-1. In order to use fountain codes, the size of packet payload should be the same. However, the output of a video codec for each frame does not necessarily have the same length. Therefore, the data generated from each frame must be divided into several $P$-byte data blocks. If the last block is insufficient to fill $P$ bytes, we add some paddings to make it a $P$-byte packet.

Some basic parameters of video coding are: frame rate $F$, video length $T$, number of packets consisting each frame $s(t)$, the maximum tolerable latency $T_{\text{Delay}}$, and the size of group of pictures (GOP) in the video codec $N_{\text{GOP}}$, whose definitions are given in Table 2-2. For notation simplicity, all the concepts relating to “time” in this article are actually in the unit of “number of frames”. Nevertheless, they can be easily converted to actual time units using frame rate $F$. We can then get the total number of native packets $k$ as $k = \sum_{t=1}^{T} s(t)$. 

21
Table 2-1. Definitions of the notations for the variables related to fountain codes.

<table>
<thead>
<tr>
<th>Notation</th>
<th>Unit</th>
<th>Definition</th>
<th>Relation to other variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P$</td>
<td>byte</td>
<td>Packet size. The number of bytes in the payload of each coded packet.</td>
<td></td>
</tr>
<tr>
<td>$R$</td>
<td>byte/second</td>
<td>Data rate. The data rate that is used to send the coded packets.</td>
<td>$R = \frac{k F \cdot P}{(T - W) \cdot C}$</td>
</tr>
<tr>
<td>$C$</td>
<td>N/A</td>
<td>Code rate. The overall code rate for encoding.</td>
<td>$C = \frac{k}{N}$</td>
</tr>
<tr>
<td>$PLR$</td>
<td>N/A</td>
<td>Packet loss rate. The end-to-end packet loss rate.</td>
<td></td>
</tr>
<tr>
<td>$\Delta t$</td>
<td>frame</td>
<td>Step size. The number of frames the window shifts each time it slides forward.</td>
<td>Should be an integral multiple of $N_{GOP}$</td>
</tr>
<tr>
<td>$W$</td>
<td>frame</td>
<td>Window size. The number of frames in a sliding window.</td>
<td>Should be an integral multiple of $\Delta t$ and $W \leq T_{Delay} - \Delta t$</td>
</tr>
<tr>
<td>$w_W(t)$</td>
<td>packet</td>
<td>Window size in number of packets. The number of native packets in the sliding window starting from $t$th frame.</td>
<td>$w_W(t) = \sum_{i=t}^{t+W-1} s(i)$</td>
</tr>
<tr>
<td>$k$</td>
<td>packet</td>
<td>Total number of native packets.</td>
<td>$k = \sum_{t=1}^{T} s(t)$</td>
</tr>
<tr>
<td>$N$</td>
<td>packet</td>
<td>Total number of coded packets to be sent.</td>
<td>$N = \lfloor N_{window} \times N_W \rfloor = \lfloor \frac{R \cdot (T - W)}{F \cdot P} \rfloor$</td>
</tr>
<tr>
<td>$N_W$</td>
<td>packet</td>
<td>Number of coded packets to be sent within each sliding window.</td>
<td>$N_W = \frac{R \cdot \Delta t}{F \cdot P}$</td>
</tr>
<tr>
<td>$N_{window}$</td>
<td>window</td>
<td>Number of windows.</td>
<td>$N_{window} = \frac{T - W}{\Delta t}$</td>
</tr>
</tbody>
</table>

The definitions of the other variables will be introduced when they are used later in this article.

Some variables are related to others, and the relationships are shown in the last column of the tables.

Existing delay-aware fountain code schemes, such as proposed in Bogino et al. (2007); Cataldi et al. (2010); Ahmad et al. (2011); Sejdinovic et al. (2009); Vukobratovic et al. (2009), mainly focus on designing them from the aspects of fountain codes: such as the structure of windows (block coding, sliding window, expanding window, etc.), and the unequal error protection (UEP). However, most of them did not deeply consider their integration with video coding. In the following two sections, we will discuss two key novel designs of the proposed scheme, as opposed to those existing designs.
Table 2-2. Definitions of the notations for the variables related to video coding.

<table>
<thead>
<tr>
<th>Notation</th>
<th>Unit</th>
<th>Definition</th>
<th>Relation to other variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F$</td>
<td>frame/second</td>
<td>Frame rate. The number of frames to be played per second.</td>
<td></td>
</tr>
<tr>
<td>$T_{\text{Delay}}$</td>
<td>frame</td>
<td>Tolerable delay. The maximum end-to-end delay the user can tolerate.</td>
<td></td>
</tr>
<tr>
<td>$s(\text{frmno})$</td>
<td>packet</td>
<td>Number of native packets in the $\text{frmno}^{\text{th}}$ frame.</td>
<td></td>
</tr>
<tr>
<td>$N_{\text{GOP}}$</td>
<td>frame</td>
<td>GOP size. The number of frames in a group of pictures (GOP).</td>
<td></td>
</tr>
<tr>
<td>$T$</td>
<td>frame</td>
<td>Video length. Total number of frames in the video sequence.</td>
<td>Should be an integral multiple of $\Delta t$</td>
</tr>
<tr>
<td>$\text{pkt}(t_0, k)$</td>
<td>packet</td>
<td>The number of packets in the first $k$ frames in the window starting from the $t_0^{\text{th}}$ frame of the video.</td>
<td>$\text{pkt}(t_0, k) = \sum_{i=t_0}^{t_0+k-1} s(i)$</td>
</tr>
<tr>
<td>$\text{pktno(\text{frmno})}$</td>
<td>packet no.</td>
<td>Starting packet sequence number of the $\text{frmno}^{\text{th}}$ frame in the video.</td>
<td>$\text{pktno(\text{frmno})} = \sum_{i=1}^{\text{frmno}-1} s(i) + 1$</td>
</tr>
<tr>
<td>$\text{frmno(\text{pktno})}$</td>
<td>frame no.</td>
<td>Frame sequence number from which the $\text{pktno}^{\text{th}}$ original packet belongs to. This function is a mapping from packet sequence number to frame sequence number.</td>
<td>$\text{frmno(\text{pktno})} = \min \left{ f</td>
</tr>
</tbody>
</table>

2.1.1 Time-based Window Size vs. Packet-based Window Size

The first distinct aspect about our proposed scheme is that the size of windows are based on time (or interchangeably speaking, based on the number of frames).

Although the specific methods may vary, a lot of existing work, such as Bogino et al. (2007); Cataldi et al. (2010); Ahmad et al. (2011); Sejdinovic et al. (2009); Vukobratovic et al. (2009), designed the delay-aware fountain codes based on the following core idea: group the video data into windows (either overlapping or non-overlapping), and send the windows
one by one within each period of time. In the aforementioned work, the coding parameters, such as the size of the windows, the speed of window movement, and the total length of the data, are constant numbers based on the number of packets. Inherently, the authors considered the number of packets as an abstraction of the number of frames. This assumption is understandable because the video packets are arranged in chronological order, and delivering the packets sequentially is equivalent to delivering the frames sequentially.

However, the packet-based schemes ignored an important characteristic of video data: the data rate of the video is nonuniform over time. It should be noted that there are different amounts of bits for each frame. Even if rate control is used, as we explained in Chapter 1, there are still bit rate fluctuations between frames. This fact makes packet-based windows different from time-based windows. If we divide the video streaming data into blocks with equal number of packets, and transmit them using fountain codes, we will observe the following phenomenons:

1. *Improper partition of frames and GOPs:* Because a frame may contain various number of packets, it is highly probable for the packets from one frame, or one GOP, to fall into two blocks. The data of a partial frame or a partial GOP will not be properly decoded by a video codec. As a result, using packet-based windows may cause video playback error, even if all native packets are correctly decoded. On the other hand, partitioning the windows according to number of frames will ensure one frame to be grouped in a same window. When further conditions are applied, we can also ensure the packets of one GOP to be grouped in a same window.

2. *Uncontrollable delay:* Another problem of packet-based windows is that we do not know the resulted time delay. Because there are different amounts of frames in each block, the time of delay varies from time to time. As a result, packet-based window cannot be used in real-time or delay-aware systems, where the time delay is a very crucial factor. Even if we have to use packet-based window in delay-aware systems, where the largest delay can be tolerated is $T_{Delay}$, we need to know the number of packets in each frame of the video before-hand, and select the fewest number of packets in any $T_{Delay}$-frame period as the packet-based window size. Otherwise, if the window size is larger than that, the delay must exceed $T_{Delay}$ at some point of time. In that case, the tolerable delay $T_{Delay}$ is underused for most of the time, and it contradicts to the designing principle of making the best use of delay. Contrarily, if the size of time-based window is set to be $T_{Delay}$, it will make the best use of the delay at all time.
3. **Unstable data rate**: One of the assumptions of existing works is to transmit the coded packets using a fixed code rate, so that the encoder generates the same amount of coded packets within any packet-based window. However, because of the nonuniformity of the data rate of video, there are different amounts of time to transmit each data block. So, if the code rate is a constant, the data rate will be different in different periods. However, because our goal is to output stabilized data rate, this is not acceptable. On the other hand, if the encoder generates the same amount of coded packets within any time-based window, the data rate will be a constant. Admittedly, fixing the data rate will also make the code rate unstable, if no other mechanisms are used. Nevertheless, in section 2.2, we will introduce a method to achieve the stability of code rate while using fixed data rate.

As a result, the proposed scheme uses number of frames as the unit of most coding parameters. As defined in Table 2-1 and 2-2, $W$, $\Delta t$, and $T$ are all in the unit of frames, and they are constant parameters for a certain application. On the contrary, the packet numbers in each window $w_W(t)$ are varied.

Although in we only explain the advantages of time-based scheme over packet-based scheme using the example of block coding this section, they still hold when using sliding window scheme, which will be introduced in the next section.

### 2.1.2 Sliding Window vs. Block Coding

The concept of SWFC was first proposed in Bogino et al. (2007) for LT codes. Then, the similar idea expanded to Raptor codes and unequal error protection (UEP) was applied in Cataldi et al. (2010). The main idea of the sliding window schemes is to virtually extend the block size, so as enhance the performance of the fountain codes by reducing the overhead. As shown in Fig. 2-1A, the block coding scheme has a relatively small block size, and the coded packets for each block are only linked to the source packets in a small window. But in Fig. 2-1B, the overlap between sliding windows makes decoded packets in one window to help the decoding of other windows. In that sense, the size of the window is virtually extended.

In Sejdinovic et al. (2009); Vukobratovic et al. (2009), the authors proposed an expanding window fountain code scheme, which is a variant of sliding window. Instead of using the overlapping fixed-size windows, the packets in each window must be a subset of the next window. Although this scheme is also delay-aware, it is not suitable for video streaming. Because the packets in the front are more likely to be chosen into the coded packets than the
ones in the back, its primary design objective was to achieve UEP, rather than to deliver the video data with equal probabilities.

In our design, we use SWFC scheme. The movement of the sliding windows is controlled by \( \Delta t \). It means the step size of the window when it moves to the next window. For simplicity, we assume \( W \) and \( T \) to be integral multiples of \( \Delta t \). In order to avoid dividing the frames from one GOP into different windows, \( \Delta t \) should be an integral multiple of \( N_{GOP} \), the number of frames in one GOP. Hence, the number of windows needed to be sent, \( N_{\text{window}} \), could be obtained by \( N_{\text{window}} = \frac{T}{\Delta t} - \frac{W}{\Delta t} \).

On the other hand, Ahmad et al. (2011) also adopted the idea of “virtually expand the block size”, but achieved it by using block coding scheme. The virtual block size is expanded by duplicating all symbols in each block. However, after a closer examination, we find the actual block size of block coding schemes is even smaller than the size of SWFC schemes when the tolerable delay is fixed. As shown in Fig. 2-2A, because the encoder can only start to encode the next block’s packets when all the packets in the next block are available, the block size of block coding is at most half of the tolerable delay. This fact makes the performance of block coding schemes, such as Ahmad et al. (2011), even poorer. To extend it to more general cases, if the step size of sliding window is \( \Delta t \), the encoder can start to encode the next window when next \( \Delta t \) packets are available, so the size of sliding window cannot exceed \( T_{\text{Delay}} - \Delta t \), as shown in Table 2-1. If we deem block coding as a special case of sliding window when
A Block coding. B Sliding window when $\Delta t = 1$. C Meanings of the colors in the blocks.

Figure 2-2. Compare block vs. sliding window.

$\Delta t = W$, it can easily draw the aforementioned conclusion. We also know that the biggest window size is obtained when $\Delta t = 1$, as shown in Fig. 2-2B.

A very important derived parameter is $N_W$, the number of coded packets to be sent within each sliding window, which is a link between fountain codes and video coding. It derives from data rate $R$, video frame rate $F$, packet size $P$ and step size $\Delta t$. As shown in Table 2-1, $N_W = \frac{R \Delta t}{F \cdot P}$. Then, the total number of coded packets to be sent, $N$, can be defined based on $N_W$: $N = \lfloor N_{\text{window}} \times N_W \rfloor = \lfloor \frac{R \cdot (T - W)}{F \cdot P} \rfloor$. The overall code rate $C$ is then defined using $C = \frac{k}{N}$.

Reversely, given the other basic parameters, data rate $R$ can also be expressed using code rate $C$, as $R = \frac{k \cdot F \cdot P}{(T - W) \cdot C}$. However, because we assume the data rate to be a constant and the definition of $k$ requires the knowledge of whole video, we still consider $R$ as the basic parameter.

2.2 Optimal Window-wise Sampling Strategy

For an LT encoder, each coded packet is generated using the following two steps: 1. Randomly choose the degree $d_n$ of the packet from a degree distribution $r(d)$; 2. Choose $d_n$ input packets at random from the original packets uniformly, and a coded packet is obtained as the exclusive-or (XOR) of those $d_n$ packets. The designing of the degree distribution $r(d)$ is a key part of the design. The design must take into account the balance between low computational complexity and high coverage fraction: some high-degree encoded packets...
should be generated so that no native packet will be left disconnected (or unchosen); on the other hand, a sufficient proportion of low-degree packets should also be included to facilitate the low-complexity belief propagation (BP) decoding.

The optimization of degree distribution for LT code has been well studied in some work, e.g. Luby (2002); Hyytiä et al. (2006, 2007). It should be noted that, the optimization procedures are all built on a common prerequisite – the sampling distribution should be uniform. It is easy to know that, among all the sampling distributions, the highest efficiency of fountain codes is achieved using uniform distribution. That is because if the sampling distribution of the native packets is nonuniform, some packets will have lower probability to be sampled than others. Therefore, the mathematical expectation of the number of the coded packets needed for the low-probability packets to be sampled will increase. In that case, in order to decode all the native packets, nonuniform sampling needs more coded packets than uniform sampling does, thus lowering the overall coding efficiency.

Figure 2-3. The resulting ASP using different SWFC schemes for video sequence foreman.

However, with the time-based sliding window, even if every window’s sampling distribution is uniform, the overall sampling distribution may still be nonuniform. The reason is that the number of packets might be different for different frames. For example, as shown in Fig. 2-3A, the bit rate is not constant for the CIF video sequence foreman coded with coding structure
IPP. The fountain codes encoder generates a same amount of coded packets within a same length of time period, and the time-based windows may contain different number of packets. As a result, if the windows are uniformly sampled and non-overlapped (block coding), the probability of each packet being sampled inside the windows with more packets will be lower than the ones with less packets. In Fig. 2-3B, the black line shows the sampling probabilities of each packet in every frame when the video is segmented into 20-frame blocks. It is easy to understand that the probability is inversely proportional to the number of bits in that video block.

The similar phenomenon is observed when using overlapping sliding windows. However, the relation between bit rate and sampling probability is more complicated than the block coding case. In SWFC scheme, instead of being related to only one window, the sampling probability of a frame is related to all the windows that covers it, as shown in (2-1).

\[
P(t) = \sum_{\omega \in \text{all windows cover frame } t} p_{\omega}^{pkt}(t) \tag{2-1}
\]

where \( p_{\omega}^{pkt}(t) \) denotes the average sampling probability of each packet in frame \( t \) within the window \( \omega \). So, \( P(t) \) denotes the total probability of every packet in frame \( t \) accumulated through all the sliding windows covering that frame, called accumulated sampling probability, or ASP in the rest of this article. Here, because we assume that the bits within each frame have equal importance, it is assumed that all the packets in one frame have a same sampling probability, which leads to (2-2).

\[
p_{\omega}^{pkt}(t) = \frac{1}{s(t)} p_{\omega}^{frm}(t) \tag{2-2}
\]

where \( p_{\omega}^{frm}(t) \) denotes the total probability of the packets in frame \( t \) to be sampled, within the window \( \omega \). \( s(t) \) is the number of packets in frame \( t \) as defined in Table 2-2.

Using a uniform-distribution sliding window (window size \( W = 20 \), step size \( \Delta t = 5 \)) to slide through the video sequence Foreman, we can obtain the ASP as shown as the red line in
Fig. 2-3B. The ASPs shown here are normalized, so the average value of all the probabilities in one scheme is normalized to 1. Because the ASP forms non-uniform distribution, we know its coding efficiency is low for fountain code. In terms of the probability of decoding, the packets with low ASP will have lower decoding probability than the ones with high ASP. Because the complex frames (with high motion, high picture complexity, etc.) will generate more bits in video coding, the packets in high-complexity frames will suffer from more packet losses than others. In terms of video quality, that will lead to frame freezes and unstable quality.

Fortunately, the overlapping property of the sliding window provides a way to stabilize the ASP: the sampling probabilities within each window can be assigned unequally to achieve the overall uniformity of the ASP. Although selecting the best sampling distribution for each window is an optimization problem, we can still intuitively understand it as follows: if the vicinity of one window has relatively low bit rate and it will get higher in the future, in order to make the overall sampling distribution as homogeneous as possible, we do not want to “waste” the sampling opportunities on the imminent frames, which are already sampled in previous windows for too many times; instead, the encoder should sample more from the future side of the window, such that it could balance the low sampling probability of the upcoming high bit rate frames.

In the following sections, we will introduce some solutions to this problem. To give a glimpse of what can optimal window-wise sampling strategies do, the blue and green lines in Fig. 2-3B show the resulting ASPs using different optimization strategies. We can see that they are significantly more stable than the non-optimal schemes, which are in red and black.

2.2.1 Per-frame Optimization Scheme

In order to optimize the sampling distributions for all windows, we must know the video length $T$, window size $W$ and the number of packets in each frame $s(t)$ (or its vector form $s = [s(1) s(2) \cdots s(T)]$). We define $p_{tt}^{frm}(i)$ to denote the probability of sampling the packets in the $i$th frame of the window starting from the $t$th frame. As in (2–2), the sampling probability for each packet in that frame within the window, $p_{tt}^{pkt}(i)$, is defined as (2–3).
\[
p_{t}^{pkt}(i) = \frac{1}{s(t+i-1)}p_{t}^{frm}(i)
\]

(2-3)

As in (2-1), the ASP for the \(t^{th}\) frame is defined as (2-4).

\[
P(t) = \sum_{t_0=t-W+1}^{t} p_{t_0}^{pkt}(t-t_0+1)
\]

(2-4)

\[
= \frac{1}{s(t)} \sum_{t_0=t-W+1}^{t} p_{t_0}^{frm}(t-t_0+1)
\]

For simplicity, this accumulation process does not consider the step sizes of \(\Delta t\) other than 1. Because both the video length \(T\) and window size \(W\) are defined to be integral multiples of \(\Delta t\), if \(\Delta t > 1\), all the parameters can be down-sampled by a factor of \(\Delta t\). For example, the new \(T' = \frac{1}{\Delta t} T\), \(W' = \frac{1}{\Delta t} W\), \(s'(t) = \sum_{i=(t-1)\Delta t+1}^{t\Delta t} s(i)\), and \(\Delta t' = 1\), so (2-3) and (2-4) still hold.

Let \(x_{t,i} = p_{t}^{frm}(i)\) and make a matrix from them. We get the parameter matrix \(A\) as in (2-5).

\[
A =
\begin{bmatrix}
    x_{1,1} & x_{1,2} & \cdots & x_{1,W} \\
    x_{2,1} & x_{2,2} & \cdots & x_{2,W} \\
    \vdots & \vdots & \ddots & \vdots \\
    x_{T-W+1,1} & x_{T-W+1,2} & \cdots & x_{T-W+1,W}
\end{bmatrix}
\]

(2-5)

The number of rows is \(W\) because each window has \(W\) sampling probabilities. The number of columns is \(T - W + 1\) because there are \((T - W + 1)\) windows in total (again, \(\Delta t\) is assumed to be 1). Because every row in the matrix represents the probability distribution within a window, the elements in \(A\) must satisfy the constraints of (2-6).

\[
\sum_{w=1}^{W} x_{t,w} = 1, \quad \forall t
\]

(2-6)

\[
x_{t,w} \geq 0, \quad \forall t, w
\]
With this notation, (2–4) can be rewritten into a parameterized form as (2–7).

\[ P_A(t) = \frac{1}{s(t)} \sum_{t_0=t-W+1}^{t} x_{t_0, t-t_0+1} \]  

(2–7)

The objective is to find the optimal parameter matrix \( A \), which minimizing the fluctuation of the ASPs, \( P_A(t) \). Because in this problem, the parameters to be optimized are the sampling probabilities for each frame of every window, we call this method the *per-frame* optimization scheme.

### 2.2.1.1 Problem description

Given the total number of frames \( T \), the window size \( W \), and number of packets in each frame \( s(t) \), we want to find a set of parameters as in (2–5), for which the mean square error of the sampling probabilities of all packets attains its minimum value. The optimization problem is defined in (2–8).

\[
\arg\min_A \sum_{t=W}^{T-W+1} \left( P_A(t) - \bar{P}_A \right)^2 \\
\text{s.t.} \quad \sum_{w=1}^{W} x_{t,w} = 1, \quad \forall t \\
x_{t,w} \geq 0, \quad \forall t, w
\]

(2–8)

where \( \bar{P}_A = \frac{1}{T-2W+2} \sum_{t=W}^{T-W+1} P_A(t) \).

It should be noted that the range of frames we want to stabilize is from \( W \) to \( W - T + 1 \). Because the frames in that range are all covered by exactly \( W \) sliding windows, they are deemed as stable frames. On the other hand, the frames before \( W \) or after \( W - T + 1 \) are covered by less than \( W \) sliding windows, so they are considered to be warm-up/cool-down frames, and not be counted as the targets of the optimization.

### 2.2.1.2 Solution

If the conditions of \( x_{t,w} \geq 0 \) are ignored, this optimization problem can be solved using Lagrange multiplier. Otherwise, it can be solved by Karush-Kuhn-Tucker (KKT) conditions.
Figure 2-4. The optimization result of sampling distributions for each window of *foreman* using per-frame optimization scheme.

An example of the optimization result is shown in Fig. 2-4. It is the optimization result of sampling distributions for each window of CIF sequence *Foreman* using per-frame optimization scheme. Window size $W = 20$ and step size $\Delta t = 5$. Because there are too many windows to be clearly shown in one figure, only a fraction of the windows is presented here. The probabilities are normalized. The trend of the bit rate, which is represented by dashed green line, is also plotted in the figure, in order to indicate the relationship between bit rate and optimization results. The blue line in Fig. 2-3B shows the resulting ASP using this per-frame optimization strategy.

2.2.1.3 Computational complexity

Because there are $\frac{W(W+\Delta t)}{\Delta t^2}$ variables to optimize and $\frac{T-W}{\Delta t} + 1$ conditions for Lagrange multiplier (if using KKT conditions, there are $\frac{(W+W+\Delta t)(T-W+\Delta t)}{\Delta t^2}$ conditions), the optimization process yields the system of equations with $\frac{(W+\Delta t)(T-W+\Delta t)}{\Delta t^2}$ equations (or $\frac{2(W+\Delta t)(T-W+\Delta t)}{\Delta t^2}$ equations in KKT conditions). Assuming that $T \gg W \gg \Delta t$, if we omit constant factors and lower order terms, the solution of both KKT conditions and Lagrange multiplier involves the generation of a parameter matrix of $\frac{T-W}{\Delta t^2} \times \frac{T-W}{\Delta t^2}$ and the computation of its inverse matrix. As a result, the computational complexity is $O\left(\left(\frac{T-W}{\Delta t^2}\right)^3\right)$.

2.2.2 Slope-only Optimization Scheme

Although optimizing the sampling distribution for each frame within every window yields the most optimal solution in terms of minimizing the fluctuation of sampling probabilities
between frames, per-frame optimization is unrealistic in practical designs. First of all, there are too many parameters to be optimized. The computational complexity is \( O\left(\frac{TW}{\Delta t^2}\right)^3 \), which is too high for large \( T \) or \( W \). Secondly, in order to reconstruct the coded packets on the decoder side, the encoder must tell the receiver what sampling distribution is used in each window, by explicitly including every frame’s sampling probability in the packet header. That will introduce a large overhead in the packet header. Since bigger packets are more vulnerable to channel noise, including too much information in headers will increase packet loss rate in wireless networks. As a result, a more concise description for the sampling distributions is needed for the practical designs, so they can be obtained with lower computational complexity, and be transmitted in the headers with less bits.

We introduce a slope-only description for the sampling distributions. It requires only one parameter – slope factor, denoted as \( a \), to control the shape of the distribution. However, it should be noted that using less bits will inevitably lose the precision of describing the sampling distributions. Therefore, compared to the optimal performance that can achieved by using per-frame description, slope-only description may result in suboptimal performance.

The slope factor is a real number, and it ranges from \(-1\) to \(1\). The distribution functions are defined to be linear functions, and the slope factor only controls the slopes of them. The distribution functions are packet-based. By contrast, the distribution functions in per-frame description are frame-based. We do not want any of the packet’s probability to be 0, because in that case, the effective window size will shrink. As a result, we define that when the slope factor \( a = 1 \), the distribution function of the packets starts from 0 and increases linearly, forming a forward triangular distribution, as shown in the top of Fig. 2-5; when the slope factor \( a = 0 \), the distribution function is a uniform distribution, as shown in the second line of Fig. 2-5; when the slope factor \( a = -1 \), the distribution function is the reverse of that in \( a = 1 \), or a backward triangular distribution, as shown in the third line of Fig. 2-5. Therefore, the distribution functions of all the slope factor values in the middle are continuously defined.
Figure 2-5. The sampling distributions when slope factor $a = 1$, $a = 0$, and $a = -1$.

As defined in table 2-1, for time $t$ in video sequence, the number of packets in the window is $w_W(t)$. For each window, a linear distribution function can be defined over the interval $[0, w_W(t)]$. Because the integration of the function in $[0, w_W(t)]$ must be 1, we can get the definitions of the lines for different slope factors $a$. When slope factor $a = 1$, it passes points $(0, 0)$ and $(w_W(t), \frac{2}{w_W(t)})$, as the red line shown in Fig. 2-6; when slope factor $a = -1$, it passes points $(0, \frac{2}{w_W(t)})$ and $(w_W(t), 0)$, as the green line shown in Fig. 2-6. The lines for all slope factors will always pass the point $(\frac{1}{2}w_W(t), \frac{1}{w_W(t)})$. As a result, the distribution function given any $a$ and $t$ is (2-9).

$$y = \frac{2a}{w_W^2(t)}x + \frac{1-a}{w_W(t)}, \quad x \in [0, w_W(t)]$$  \hspace{1cm} (2-9)

As stated in section 2.2.1, the sampling probabilities of the packets within a same frame should be the same. As a result, the probabilities of sampling one frame should be grouped together, and the actual sampling probability of each packet should be the average value of all packets in its frame. As the example shown in Fig. 2-7, there are four frames, each of which contains 3, 4, 2 and 2 packets respectively. The actual sampling probability for each
packet is the average value of the packets in the interval of its frame. Given slope factor $a$, the probability of sampling the $i^{th}$ frame in the window starting from the $t^{th}$ frame, denoted as $p_{t,a}^{frm}(i)$, is then defined as (2–10).

\[
p_{t,a}^{frm}(i) = \int_{p_{t,i-1}}^{pkt(t,i)} \left( 2a \frac{w_{W}^{2}(t)}{w_{W}(t)^{2}} x + \frac{1 - a}{w_{W}(t)} \right) dx
\]

\[
= \left( \frac{2a}{w_{W}^{2}(t)} \left( pkt(t, i) - \frac{s(t + i - 1)}{2} \right) + \frac{1 - a}{w_{W}(t)} \right) \times s(t + i - 1),
\]

\[
i = 1, 2, ..., W
\]

where $pkt(t, i)$ is defined in table 2–2. The second equality holds because the distribution function is a linear function, and the average value is taken at the middle point of each interval.

As in (2–2), given slope factor $a$, the probability of each packet to be sampled in the $i^{th}$ frame within the $t^{th}$ window, denoted as $p_{t,a}^{pkt}(i)$, is the average value of the distribution function in the interval $[pkt(t, i - 1), pkt(t, i)]$, which is defined in (2–11).
Figure 2-7. An example of sampling distribution for each frame within a window.

\[
p_{\text{pkt}}^{p}(i) = \frac{p_{\text{frm}}(i)}{s(t + i - 1)}
\]

\[
= \frac{2a}{w_{W}(t)} \left( \frac{\text{pkt}(t, i) - \frac{s(t + i - 1)}{2}}{2} \right) + 1 - a \frac{1}{w_{W}(t)},
\]

\[
i = 1, 2, ..., W
\]

We can rewrite (2–11) into a more concise version as in (2–12).

\[
p_{\text{pkt}}^{a}(i) = d_1(t, i) \cdot a + d_2(t), \quad i = 1, 2, ..., W
\]

where,

\[
d_1(t, i) = 2 \cdot \frac{\text{pkt}(t, i) - \frac{s(t + i - 1)}{2}}{w_{W}(t)} - \frac{1}{w_{W}(t)}
\]

\[
d_2(t) = \frac{1}{w_{W}(t)}
\]

We can see in (2–13) that \(d_1\) is only relevant to \(s, W, t\) and \(i\), and \(d_2\) is only relevant to \(s, W\) and \(t\). Because \(T, W\) and \(s\) are constant values for a certain video, they can be deemed as parameters for the functions, but not arguments.

As in (2–1), the ASP for each packet in frame \(t\), denoted as \(P_a(t)\) is defined in (2–14).
\[ P_a(t) = \sum_{t_0 = t-W+1}^{t} p_{b_0, a_{t_0}}^{p_{kt}} (t - t_0 + 1) \]  

(2-14)

\[ a = \begin{bmatrix} a_1 & a_2 & \cdots & a_{T-W+1} \end{bmatrix} \]  

(2-15)

where \( a \) denotes the set of slope factors for all windows in the video sequence (from frame 1 to frame \((T - W + 1)\)) as in (2-15). Again, for simplicity, this accumulation process does not consider the step sizes of \( \Delta t \) other than 1.

Similarly, we can rewrite (2-14) for clearer notations as in (2-16).

\[ P_a(t) = d_3(t) \cdot a + d_4(t) \]  

(2-16)

where \( \cdot \) denotes the dot product of the two vectors of \((T - W + 1)\) elements, and

\[
\begin{align*}
d_3(t) &= \begin{bmatrix} d_3(t, 1) & d_3(t, 2) & \cdots & d_3(t, T - W + 1) \end{bmatrix} \\
d_3(t, i) &= \begin{cases} d_1(i, t - i + 1) = \frac{2 \cdot p_{kt}(t - i + 1) - s(t)}{w_{w}(t)} - \frac{1}{w_{w}(t)}, & \text{if } i \in [t - W + 1, t]; \\ 0, & \text{otherwise}. \end{cases} \\
d_4(t) &= \sum_{t_0 = t-W+1}^{t} d_2(t_0) = \sum_{t_0 = t-W+1}^{t} \frac{1}{w_{w}(t_0)} 
\end{align*}
\]  

(2-17)

From (2-17) we can see that \( d_3 \) and \( d_4 \) are only relevant to \( s, W \) and \( t \), but irrelevant to \( a \).

With the equations defined above, we can describe the optimization problem as follows.

**2.2.2.1 Problem description**

Given the total number of frames \( T \), the window size \( W \), and number of packets in each frame \( s(t) \), we want to find a set of slope factors as in (2-15), for which the mean square
error of the sampling probabilities of all packets attains its minimum value. The optimization problem is defined in (2–18).

$$\arg \min_a \sum_{t=W}^{T-W+1} (P_a(t) - \overline{P}_a)^2$$

$$s.t. \quad -1 \leq a_t \leq 1, \quad \forall t$$

where $\overline{P}_a = \frac{1}{T-2W+2} \sum_{t=W}^{T-W+1} P_a(t)$. The range of frames we want to stabilize is also from $W$ to $W - T + 1$, for the same reason as stated in section 2.2.1.

### 2.2.2.2 Solution

As in the per-frame scheme, it can be solved by KKT conditions. The result of each window’s sampling distribution using slope-only optimization is shown in Fig. 2-8 with the same settings as in Fig. 2-4. The green line in Fig. 2-3B shows the resulting ASP using this slope-only optimization strategy. We can observe that, in terms of stability of ASP, slope-only scheme yields worse result than per-frame scheme.

### 2.2.2.3 Computational complexity

Because there are $\frac{T-W}{\Delta t} + 1$ variables to optimize and $2 \times (\frac{T-W}{\Delta t} + 1)$ conditions for KKT conditions, the optimization process yields the system of equations with $3 \times (\frac{T-W}{\Delta t} + 1)$ equations. Assuming that $T \gg W \gg \Delta t$, if we omit constant factors and lower order terms, the solution involves the generation of a parameter matrix of $\frac{T}{\Delta t} \times \frac{T}{\Delta t}$ and the computation.
of its inverse matrix. As a result, the computational complexity is \( O\left(\frac{T}{\Delta t}\right)^3 \). Compared to that of per-frame scheme, the computational complexity of slope-only scheme is lowered by the factor of \((W/\Delta t)^3\).

### 2.3 Implementation Details

In this section, we will discuss some detailed aspects in the implementation of proposed method.

#### 2.3.1 Evaluation Criteria

In our work, we use the packet decoding ratio to evaluate the performance of the schemes, since higher packet decoding ratio implies higher visual quality of video.

It is worth noting that the evaluation criteria of SWFC used in delay-aware multimedia streaming is different from the file transfer applications, and it is commonly overlooked by existing SWFC schemes. In delay-aware applications, if a packet is decoded after its playback time, it has to be counted as a packet loss for the video decoder, since the player does not rewind the video.

As a result, we introduce the criteria of in-time decoding ratio (IDR), which only counts a decoded packet as “in-time” decoded when it is within the current window. Comparatively, file decoding ratio (FDR) means the percentage of total decoded packets after the complete coding session finishes. For SWFC schemes, there is always \( FDR \geq IDR \); for block coding, \( FDR = IDR \).

#### 2.3.2 Degree Distribution

As mentioned in section 2.2, the degree distribution plays a crucial role in the performance of LT code, and it has been studied in several works, such as Luby (2002); Hyytiä et al. (2006, 2007). The basic principle of optimal degree distribution is based on Robust Soliton distribution (RSD) proposed in Luby (2002). It derives from the combination of the elements of mass function of the Ideal Soliton distribution (ISD) and an extra set of values. The definition of ISD is given in (2–19).
\[ \rho(i) = \begin{cases} \frac{1}{k}, & \text{for } i = 1 \\ \frac{1}{(i-1)}, & \text{for } i = 2, \ldots, k \end{cases} \]  
(2-19)

where \( \rho(\cdot) \) stands for ISD. \( k \) denotes the data length. Then, a set of supplement values are added to ISD, and it is standardized so that the values add up to 1. The definition of RSD is given in (2-20).

\[
\beta = \sum_{i=1}^{k} \rho(i) + \tau(i),
\]

\[
r(i) = \left( \rho(i) + \tau(i) \right) / \beta, \quad \text{for } i = 1, \ldots, k.
\]

where \( r(\cdot) \) denotes the RSD. \( \tau(\cdot) \) is the set of supplement values as described in Luby (2002) as (2-21).

\[
\tau(i) = \begin{cases} \frac{R}{i k}, & \text{for } i = 1, \ldots, \lfloor k/R \rfloor - 1 \\ \frac{R \cdot \ln(R/\delta)}{k}, & \text{for } i = \lfloor k/R \rfloor \\ 0, & \text{for } i = \lfloor k/R \rfloor + 1, \ldots, k \end{cases}
\]

(2-21)

where \( R = c \cdot \ln(k/\delta) \sqrt{k} \). It has two parameters \( \delta \) and \( c \), which should derive from an optimization process, such as Cataldi et al. (2006), in order to gain the optimal performance. However, it is too expensive to compute on-line and is also out of our current scope. Thus, we decided to fix them so RSD can get good average performances in our implementations.

However, having the adjusted evaluation criteria in delay-aware applications, the value of data length \( k \) needs to be changed in RSD. In (2-19), we must ensure that, within any window, there is at least one coded packet having the degree of 1, so the in-time BP decoding could be triggered within the delay. As a result, \( k \) should be set to \( w_W \), which is the packet number within a window. Because we also add the supplement values of \( \tau(\cdot) \) to increase
its robustness, the performance is not overly sensitive to the data length. As a result, for simplicity, we set $k$ for each round of SWFC as the average value of $w_W$ of that sequence.

### 2.3.3 Warm-up/Cool-down Period

If the window always contains $W$ frames, and it slides from the $1^{st}$ frame to the $T^{th}$ frame at the speed of $\Delta t = 1$, it is easy to realize that all the frames will be covered in $W$ windows, except for the first $W - 1$ frames and the last $W - 1$ frames. Namely, the first and the last $i^{th}$ frame will be covered in $i$ windows when $i \leq W - 1$. We call those two periods as *warm-up* and *cool-down* periods (W/CP), as illustrated in Fig. 2-9, since they are undersampled and yield unstable decoding ratio.

In our implementation, before the actual SWFC begins, both encoder and decoder will obtain the length of W/CP. The encoder will fill these two periods with padding characters, and the decoder will do the same and automatically mark those packets as decoded. Then, the SWFC is performed. The detailed procedures of encoder and decoder will be introduced in section 2.4.2 and 2.4.3. Also, for the sake of fairness, the pseudo-decoded padding packets in W/CP should not be counted as being decoded in evaluation, since they do not contain any useful information.

### 2.4 System Design

In this section, we design a practical system for the delay-aware fountain code scheme. From the acronyms of the scheme, we name our protocol as DAF.

#### 2.4.1 Packet Structure

Because of the reasons we stated in section 2.2.2, we design DAF protocol based on the slope-only description and optimization scheme.
A The APP layer structure of a DAF packet.  

B The header structure of a DAF packet.

Figure 2-10. The packet structure and the header structure of a DAF packet.

The structures of a DAF packet and its header are shown in Fig. 2-10. As shown in Fig. 2-10A, the APP layer data of DAF packet consists of two parts: header and payload. The payload is coded, and its length is given in the header. Fig. 2-10B shows the structure of the header. The total size of header is 15 bytes. It includes the starting packet position of the window \( \text{StartP} \), the size of current window in the unit of number of packets \( \text{WSize} \), the slope factor used in current window \( \text{SlopeF} \), packet ID \( \text{PacketID} \) and packet size \( P \).

It should be noted that, although the window size is fixed in the unit of frame number, the number of packets in it is an unknown variable to the receiver, so \( \text{WSize} \) should be explicitly included in the header. The data length of \( \text{SlopeF} \) determines the precision of the slope factors used in generating sampling distribution. In our protocol design, we use 4 bytes as the length of it, which stores a real number as the float type in C++. \( \text{PacketID} \) starts from 1, and will be increased by 1 every time a coded packet is sent. It serves the similar purpose as in fountain code, which is the random seed for generating degrees and sampling packets.

2.4.2 DAF Encoder

The system design of DAF encoder is shown as a flowchart in Fig. 2-11. Beforehand, the coding parameters, degree distributions and W/CP are already obtained by the encoder, as stated in section 2.3. The system takes two sets of input: the parameters assigned by user (e.g. \( T_{\text{Delay}}, \Delta t, R, C \)) and the video source.

The video source feeds the system with streamed video data, and it is first processed by the video preprocessing module, as shown in the dotted box. This module has two functionalities. First, it analyzes the video, and gets the basic information such as \( F, s \),
Figure 2-11. The flowchart of the DAF encoder.

$N_{GOP}$, etc. It also optimizes the slope factors $a$ using the slope-only optimization scheme introduced in section 2.2.2. Second, it segments the data from each frame (or GOP) in to several $P$-byte packets, and pads the insufficient packets to $P$ bytes. It puts the segmented video packets in the packet buffer.

The middle row of the flowchart describes the encoding algorithm of DAF system. After the procedure is triggered by the timer, the scheduler will determine whether to move the window to the next position, according to the parameters and the current status. If not, StartP, WSize and SlopeF remain the same as last sent packet; if the window slides, let $StartP = StartP + \Delta t$, $WSize = w_w(StartP)$, and $SlopeF = a(StartP)$. In both cases, let $PacketID = PacketID + 1$.

In the next step, a degree $d$ is chosen according to the degree distribution, like that in LT codes. Then, $d$ packets are sampled from the packet buffer in the range confined by $StartP$ and $WSize$. Different from the uniform distribution in ordinary LT codes, each window’s packet-wise sampling distribution is generated by $P(2)$, given $SlopeF$. The bit-wise XOR of these $d$ original packets is obtained as the payload of current coded packet.

At last, the parameters and the payload are assembled as an APP layer packet, according to the structure shown in Fig. 2-10. The packet will be sent using UDP. Last but not least, the program will set the timer to trigger the procedure again according to the frequency of sending packets, which is determined by parameters such as $F,P,R,C$, etc.
2.4.3 DAF Decoder

The system design of DAF decoder is shown as a flowchart in Fig. 2-12. Also, the coding parameters, degree distributions and W/CP are already obtained by the decoder. The procedure starts when a coded packet is received.

The decoding procedure is basically the reverse of encoding procedure. Having StartP, WSize, SlopeF and PacketID, the degree $d$, the sampling distribution and the composition of the coded packet can be reconstructed. They are fed into a belief propagation (BP) decoder, which tries to decode the original packets. The decoded packets are stored in the packet buffer. Bear in mind that, because of the application is delay-aware, we only count the in-time decoded packets as successfully received ones as stated in section 2.3.1. However, the expired decoded packets are kept in decoder’s buffer, because they may help the decoding of future packets.

The video playback module takes the packets from the packet buffer as the time goes. First, the packets are re-assembled into frames (or GOPs). If a packet is not decoded yet when it is taken, it is considered as a packet loss. If that happens, the image processing techniques such as error concealment could be performed to fix it before playing it. However, because error concealment is not in the scope of this work, the lost part here will be simply left blank and let it freeze in the video player.

2.4.4 General Framework of Fountain Code Systems

It is worth mentioning that the framework of DAF system is a generalization of many existing schemes based on fountain codes. Different fountain code schemes can be easily
implemented by changing settings and modules in DAF system, but the protocol does not need to be changed.

For example, if a user does not need the sampling distribution optimization due to limited computational power, $\text{SlopeF}$ can be set to 0, which means uniform distribution, to have a low complexity version of DAF (DAF-L); the original fountain code can be viewed as a special case when $W = T$, and let the timer continually send the coded packets until an ACK is received; the block coding schemes can also be viewed as special case when $\Delta t = W$; furthermore, the sliding window schemes with packet-based window size, like Bogino et al. (2007), is a special case with fixed $W\text{Size}$; finally, expanding window Sejdinovic et al. (2009); Vukobratovic et al. (2009) can be viewed as another special case if we modify the scheduler of the encoder, by fixing $\text{StartP}$.

As a result, the proposed system enjoys the flexibility to meet different requirements.

\section*{2.5 Real-time Algorithm for DAF}

Although DAF and DAF-L can improve the quality of video streaming over lossy wireless network, there are three major drawbacks:

- The decoding performance of DAF-L is relatively low. As we will see in Section 2.7, although the performance of DAF-L is higher than other delay-aware schemes, there is still a notable gap between DAF and DAF-L (10\% maximum in decoding ratio). That is because DAF-L does not use the window-wise sampling distribution optimization as in DAF. In exchange, DAF-L is a low-complexity and online algorithm.

- DAF suffers from high computational complexity. The global optimization function of DAF brings the complexity of $O(T^3)$. Since the computational scale grows cubically with the video length $T$, it is not a practical algorithm for long videos.

- DAF is an offline scheme. Because the bit rate information of all frames is required to perform the optimization, the whole video file must be available before encoded by fountain codes. Therefore, it cannot be used on real-time video streaming applications.

In order to extend the application of DAF codes to real-time video streaming, we are looking for a method to close the gap between DAF-L and DAF: an online algorithm with a lower computational complexity than DAF, but higher performance than DAF-L. We found that Model Predictive Control (MPC) provides a locally optimal solution to any
objective-function-minimizing process control problem, which gives us an inspiration for
the solution to our problems.

2.5.1 Model Predictive Control

The term Model Predictive Control (MPC) does not designate a specific control strategy,
but a wide range of methodologies which make an explicit use of a process model to obtain
the control signal by minimizing an objective function Camacho and Alba (2013). The
methodology of all the MPC-based controllers is characterized by the following steps:

1. At each time τ, the process model is explicitly used to get the output at a future time
   horizon \( H \). These predictions \( y_\tau(\tau + k) \)
   \(^1\) for \( k = 1...H \) depend on the known values
   (history inputs and outputs) up to time \( \tau \) and the future control signals, which are those
   to be optimized.

2. The control sequence \( u_\tau(\tau + k), k = 0...H - 1 \) is calculated by minimizing the objective
   function. The minimization function usually takes the form of a quadratic function of the
   error between the prediction and the target value.

3. The first action \( u_\tau(\tau) \) from the calculated control sequence is sent to the process whilst
   the other control actions are rejected. Then, the next sampling instant \( y(\tau + 1) \) will
   be known, and step 1 will be repeated with this new value, and the next control action
   \( u_{\tau+1}(\tau + 1) \) is calculated, which in principle will be different to \( u_\tau(\tau + 1) \) because of
   the new information available.

The algorithm is iteratively executed each time a new instant is sampled. All various MPC
algorithms only differ with each other in the horizon length, predictive model and the objective
function to be minimized. Note that it is in general not a globally optimal algorithm, since the
control decisions are made only based on the history values and the prediction of a finite future
horizon.

2.5.2 MPC-based Optimal Control for DAF Codes: the Offline Version

We propose a novel scheme that combines MPC and DAF codes to impose a finite horizon
to the optimization process of DAF codes, in order to overcome its disadvantages described
at the beginning of this section. We call the proposed scheme as DAF-M, where M stands for

\(^1\) This notation indicates the value of the variable at the time \( \tau + k \) is calculated at time \( \tau \).
MPC. A general flowchart of DAF-M is shown in Fig. 2-13. The text on the top of each block describes the procedure in MPC. The text in the parentheses below explains its function in DAF-M. We will go into the details later.

In this part, we focus on the localization of the optimization problem, so we temporarily assume that we know the full bit rate information of a video sequence. As a result, DAF-M is an offline algorithm. We will discuss the prediction of bit rate and other practical problems for online algorithm later in this section.

2.5.2.1 Horizon

Denote $\tau$ as the index of current time (also the index of current sliding window). Since the window size is $W$, the frame that was newly added to the encoding buffer is the $((\tau + W - 1))^{th}$ one.

In order to determine what slope to use in the $\tau^{th}$ window, we may calculate the best set of slopes over a future horizon based on both the known ASP in the past and the predicted ASP in the future, then take the first slope as the one used in encoding the $\tau^{th}$ window.

We denote the length of the horizon as $H$, so the range of windows involved in our slope optimization is $[\tau, \tau + H - 1]$.

We hope to clarify the difference between window size $W$ and horizon length $H$ here. The two concepts are confusable, since both of them are spans of frames, and with both values increasing, the performance and computational complexity of the controller tend to be higher.
In fact, they are independent parameters that serve different purposes. $W$ is typically chosen in accordance with the longest tolerable end-to-end delay. It is specified by the user, so it is an input value that does not change in a communication session. On the other hand, $H$ is a parameter to balance the computational complexity and desired performance. It is chosen by the application according to the computing power, network condition, quality demand, etc. The extreme case would be $H$ equals to 1, with which only one slope is calculated, the algorithm reduces to a greedy algorithm; if $H$ equals to $T - \tau + 1$, thus all the frames in the future are considered in order to obtain each slope, it becomes the same as the globally optimal algorithm of DAF.

2.5.2.2 Objective function

Our target is similar to the global optimization problem in (2-18), which is minimizing the variance of ASP. With a finite horizon imposed, the objective function is limited to a local range. Because the slopes in the horizon $[\tau, \tau + H - 1]$ can affect the ASP of the frames in $[\tau, \tau + H + W - 2]$, the objective function is the variance over that range as in (2-22).

$$J = \sum_{t=\tau}^{\tau+H+W-2} \left( P_{s,a_{t}^{H}}(t) - \overline{P_{s,a_{t}^{H}}} \right)^2,$$

where $a_{t}^{H}$ denotes the $H$-length slope vector for the windows starting from $\tau$. $P$ denotes the vector of ASP in the range of $[\tau, \tau + H + W - 2]$, including both past and predictive values. The calculation of $P$ will be discussed in the following part. $\overline{P_{s,a_{t}^{H}}}$ is the average value over $P$.

2.5.2.3 Process model

The process model plays a decisive role in a MPC controller. The chosen model must be capable of capturing the process dynamics so as to precisely predict the future outputs. Different from the problems in most industrial control scenarios, in DAF codes, if the bit rate vector $s$ is given, the mapping between control sequences (slope vector) and the predicted output (future ASP) is deterministic:
ASP (s, a) = \left[ ASP_{s,a}(1) \ ASP_{s,a}(2) \cdots ASP_{s,a}(l) \right],

(2-23)

where s is a bit rate vector, and a is a slope vector. \( ASP_{s,a}(t) \) is defined in (2-16). Vector length l is equal to the length of a. In the calculation of ASP, s needs to contain \( W - 1 \) more elements than a.

Although ASP model is deterministic if we know all the bit rates s and slope parameters a in the future, the process model is no longer deterministic when a finite horizon is imposed.

For the current window \( \tau \), the ASP in the range of objective function in (2-22) consists of three parts, all of which are \( (H + W - 1) \)-length vectors representing a part of ASP within the range of \([\tau, \tau + H + W - 2]\). Some of the components only have a part of non-zero elements in that range, which will be pointed out later.

- **\( P_{\text{init}} \):** the initial ASP that already sampled by previous windows.
  Non-zero range: \([\tau, \tau + W - 2]\).
  \[
  P_{\text{init}} = ASP \left( s_{\tau-W+1}^{2W-2}, a_{\tau-W+1}^{W-1} \right),
  \]
  where the index notation in \( s_i^l \) and \( a_i^l \) means the elements in these two vectors are the \( t \)th to \((t + l - 1)\)th elements from vectors s and a, respectively. Because \( P_{\text{init}} \) is computed based on known s and previous selections of a, it has no variable parameter.

- **\( \tilde{P} \):** the range where ASP are affected by the currently optimized slopes within the horizon.
  Non-zero range: \([\tau, \tau + H + W - 2]\).
  \[
  \tilde{P}_{s,a_H} = ASP \left( s_{\tau-H}^{H+W-1}, a_{\tau-H}^{H} \right),
  \]
  where \( a_{\tau-H}^{H} \) is the \( H \)-length slope vector to be optimized as in (2-22).

- **\( \hat{P} \):** the range where ASP will be affected by the slopes out of the horizon range in the future.
  Non-zero range: \([\tau + H, \tau + H + W - 2]\).
  \[
  \hat{P}_{s} = ASP \left( s_{\tau+H}^{2W-2}, a_{\tau+H}^{W-1} \right),
  \]
  where \( \hat{a} \) denotes the prediction of future slopes. Since the optimal choices of the slopes in this part will also be affected by the slopes in the further future, in order to limit the length of horizon, we shall make a prediction without full knowledge about further frames. This is the part that may bring error. Fortunately, because the selection of
slopes of frames from further away will have less impact on current optimal slope choice, a simple and safe prediction is enough. In this work, we simply use an all-zero vector as \( \mathbf{a} = 0 \), assuming all the windows in that range will use uniform distributions. In that case, we can eliminate the variable parameter \( \mathbf{a} \). The future bit rates up to time \( \tau + H + 2W - 3 \) are still needed as input parameters.

The result is a combination of the three components:

\[
P_{s,a'} = P_{\text{init}} + \tilde{P}_{s,a'} + \hat{P}_s.
\]  

(2–27)

We plug (2–23) - (2–27) into the objective function (2–22) and solve the optimization function (2–28).

\[
\arg\min_{a'} J = \sum_{t=\tau}^{\tau+H+W-2} (P_{s,a'}(t) - \overline{P_{s,a'}})^2,
\]

(2–28)

\[
s.t. \quad -1 \leq a_t \leq 1, \quad \forall t.
\]

After solving the optimal \( a_H^t \), only the first element \( a_t(\tau) \) is chosen to encode the \( \tau \)th window.

2.5.2.4 Computational complexity

For each window \( \tau \), the optimal slope (2–28) can be solved by Karush-Kuhn-Tucker (KKT) conditions. Because there are \( H \) variables to optimize and \( 2H \) conditions for KKT conditions, the optimization process yields a system of equations with \( 3H \) equations. If we omit constant factors, the solution involves the generation of a parameter matrix of \( H \times H \) and the computation of its inverse matrix. The algorithm will be executed iteratively for each window for \( T \) times. As a result, the total computational complexity is \( O(T \times H^3) \). Since \( H \ll T \) and \( H \) does not increase with video length, it can be considered as a linear complexity algorithm for time \( T \). Comparatively, DAF-M has a much lower complexity than \( O(T^3) \) of DAF.

2.5.3 Online Algorithm with Bit Rate Prediction

From (2–25) and (2–26), we know that in order to optimize the slopes in horizon \( H \), we must know the bit rates in \( H + 2W - 3 \) frames ahead of current time \( \tau \), or \( H + W - 2 \) frames
ahead of the last known (a.k.a. the \((\tau + W - 1)^{th}\)) frame. However, in live-stream videos, we do not know the future bit rates beforehand. As a result, in order to make DAF-M an online algorithm, we need to make a prediction on future bit rates, and the length of prediction is \(H + W - 2\).

Admittedly, a more accurate prediction of video bit rate will help MPC optimization to obtain better control sequences. Therefore, using advanced algorithms, such as Abdennour (2005); Doulamis et al. (2003); Bhattacharya et al. (2003); Lee et al. (2007b); Sun and Yan (2011a) may improve the coding result. A survey of video frame size forecasting algorithms can be found in Haddad et al. (2013). Nevertheless, since video bit rate is a stochastic process, we can never expect a perfect prediction. Since our work does not focus on bit rate prediction algorithms, a simple linear formula \((2-29)\) is adapted to predict the future bit rates. A similar formula is used in Adas (1998) and yields good results.

\[
\hat{s}_{t+1} = (1 - \alpha) \hat{s}_t + \alpha s_t, \tag{2-29}
\]

where \(s_t\) denotes the bit rate of the frame newly added to the DAF encoder, and \(\hat{s}_{t+1}\) denotes the prediction for the future frames. \(\alpha \in (0, 1]\) controls the weight of new frame in the prediction of bit rate. We fix it to 0.75 so our algorithm can get good average performances in our implementations.

Since \((2-29)\) is not accurate enough to perform the frame-by-frame prediction, only a level of future bit rate is obtained. We replace all the elements of \(s\) in \((2-27)\) which exceed the last known frame with the predicted bit rate \(\hat{s}_{t+1}\). We call the online MPC-based DAF codes algorithm as DAF-O.

When it comes to computational complexity, in our work, the prediction of bit rate in DAF-O is only \(O(1)\), so the online algorithm does not increase the total computational complexity in DAF-M.
Figure 2-14. Bit rate for *foreman* and the optimized sampling distributions for each window using different optimization schemes.

2.5.4 Optimization Results and Evaluations

In this part, we show some examples of optimization results obtained by different schemes, and analyze the resulting ASP.

2.5.4.1 ASP results comparison

Fig. 2-14A shows the bit rates for CIF video sequence *foreman* coded with H.264/AVC standard. Fig. 2-14b-d present the optimization results using DAF-O, DAF-M and DAF respectively, with $W = 30$ and $H = 10$. In Fig. 2-14b-d, each slope line represents a sampling distribution within a window. Since $\Delta t = 1$, there should be a window for every $[t, t + W - 1]$ span. Because there are too many windows to be clearly shown in one figure, only a fraction of the windows is presented here.

We can observe that (i) the first $W - 1$ slopes in DAF-M (Fig. 2-14C) and DAF-O (Fig. 2-14B) are uniform distributions. That is because there are not enough past ASP to calculate $P_{\text{init}}$ in that range (see (2–24)), so the slopes in that range are set to 0 as a warm-up period; (ii) the latter part of DAF-M in is very similar to that in the globally optimal result of DAF in Fig. 2-14D, whilst DAF-O is not, because of its lack of information about the future bit rates.
Figure 2-15. Resulting ASP using different DAF-based schemes for *foreman*.

Fig. 2-15 shows the resulting ASP using the optimized slopes from Fig. 2-14B, 2-14C and 2-14D (including DAF-L, which uses all uniform distributions). The numbers following scheme names in the legend are corresponding variations of ASP, and there is $\text{DAF} < \text{DAF-M} < \text{DAF-O} < \text{DAF-L}$.  

### 2.5.4.2 Measuring the quality of ASP

Although we use variance of ASP as the objective function in (2-22), it is not an effective metric to measure the quality of the resulting ASP in fountain codes. For example, if obtained sampling distributions generate very low ASP, even if the variance is low, they are not desirable. Besides, the performance of obtained sampling distribution may vary under different code rates. Therefore, we introduce a new metric to measure the quality of ASP, called *ASP coverage ratio*, denoted as $\rho$ as in (2-30).

$$\rho_p (c) = \frac{\sum_{t=1}^{T} \left[ P_t \geq \frac{1}{c} \right]}{T}, \quad (2-30)$$

where $\left[ \cdot \right]$ is an Iverson bracket notation, which is defined to be 1 if the condition in square brackets is satisfied, and 0 otherwise. $\rho (c)$ gives the ratio of frames whose ASP surpasses $1/c$. It is a monotonically increasing function, and its output values are in the range of $[0, 1]$: if $c < 1/P_{\text{max}}$, the output value is 0; if $c \geq 1/P_{\text{min}}$, the output value is 1 (where $P_{\text{max}}$ and $P_{\text{min}}$ denote the maximum and minimum values in $P$ respectively).
Figure 2-16. The coverage ratios of the resulting ASP obtained by different schemes for *foreman*.

It is a good metric to measure the quality of ASP in this case for the following reasons. ASP reflects the average accumulated sampling probability for a frame, so, under the same code rate, greater ASP of a frame indicates higher probability for it to be sampled than others. It also means, in order to be sampled into at least one coded packet, a frame with higher ASP expects lower code rate, and ASP is inversely proportional to expected code rate. Therefore, if the ASP of a frame is higher than the reciprocal of code rate $\frac{1}{c}$, that frame is likely to be sampled in coded packets under code rate $c$, thus it has the chance to be decoded on the receiver side. As a result, $\rho(c)$ represents the ratio of frames that could be decoded under code rate $c$.

Fig. 2-16 shows the coverage ratio curves of ASP from Fig. 2-15 and block coding. $c$ is normalized to the range of $[0, 1]$. The result justifies the gains brought by the proposed DAF codes: when $c$ is greater than 60% of $\frac{1}{P_{\text{min}}}$, the ASP coverage ratios are over 60% and there is $\text{DAF} \geq \text{DAF-M} \geq \text{DAF-O} \geq \text{DAF-L}$. That means in terms of the ratio of possible decoded frames within that code rate range, DAF outperforms DAF-M, DAF-M outperforms DAF-O, and DAF-O outperforms DAF-L.

Conversely, the relationships of $\rho$ are inversed when $c < 50\%$. However, since in those scenarios the code rates are too low that the decoding ratios are below 50\% and the video can hardly be viewed for all schemes, they are not the cases users concern about. In other words,
the proposed DAF codes sacrifice the performance when the code rate is extremely low, for better performance when the code rate is in a moderately high range.

It should be noted that, this metric does not fully represent the actual decoding ratio in video communication. It only considers the sampling probability, but (i) it does not imply when the frames are decoded, so it does not represent in-time decoding ratio (IDR); (ii) it does not consider the factors of fountain codes, such as degree distribution, codeword length, etc. That is the reason why the curve of block coding does not look significantly worse than DAF-L in Fig. 2-16, while the IDR results of block coding is far inferior to any other schemes, as we will see in Section 2.7.

2.6 Unequal Error Protection for DAF

It should be noted that, by far, DAF only considered equal error protection (EEP). As a result, DAF assumes that all the data in the video has the same importance, thus every packet should have the equal chance to be decoded. Actually, DAF is heavily based on this assumption: the objective function of (2-18), which is the major optimization process in DAF, is to minimize the variance of sampling possibility.

In video applications, the importance of data may be different. Especially when the channel bandwidth is insufficient, decoding all packets with uniform probability sometimes induces distortion. For example, the different levels of importance could be reflected according to the following criteria.

1. **Picture types, such as I-frames, P-frames and B-frames.**
   An I-frame is typically the first frame in a GOP, and all other P-frames describes the picture based on its relationship with the previous frame. Similarly, B-frames are encoded based on the neighboring P-frames. The error in I-frame is more likely to be propagate to the whole GOP than P-frames and B-frames. As a result, the importance of I-frames is higher than that of P-frames, and the importance of P-frames are higher than that of B-frames.

2. **Data types, such as header information, motion vector and residue of predictive macro block (MB).**
   Without header information, the whole MB cannot be decoded. Without motion vectors, the residue is useless. As a result, the importance of headers is higher than that of motion vector, and the importance of motion vector are higher than that of residue.
3. **The position of a frame in a GOP.**
   In a GOP, the frames in the front will propagate the error further, due to the predictive nature of video coding. As a result, the importance of frames in the front of a GOP is higher than the frames in the back.

4. **Layers in scalable video coding (SVC).**
   Some video coding standards have the SVC extension to support scalable video coding. The video can be encoded into a base layer (BL) and multiple enhancement layers (EL). Decoding of each layer is based on the previous layer, and the overall video quality is gradually improve by using more upper layers. Apparently, the importance of BL is always higher than that of EL.

5. **Picture content.**
   The data within a picture also has different importance because of the content. Since human visual system (HVS) is more sensitive to the region of interest (ROI), the error in ROI is more noticeable. As a result, the importance of data that representing ROI is higher than other data.

   Such disparity of importance between bits raises a need for codes with unequal error protection (UEP). UEP codes were first studied in Masnick and Wolf (1967). Since then, there has been many UEP schemes proposed to work with LDPC codes, e.g. Rahnavard and Fekri (2004); Pishro-Nik et al. (2005). Because of the advantages of fountain codes as we introduced earlier, designing fountain codes with UEP property is also of great interest. The performance of UEP-based fountain codes was theoretically analyzed in Rahnavard et al. (2007); Arslan et al. (2012). The authors pointed out that UEP property does not only protect the important bits, but also lowers the bit error rates in LT codes and Raptor codes within certain ranges of overhead ratio. The potential of UEP in video communication is also studied in some work, e.g. Sejdinovic et al. (2009); Vukobratovic et al. (2009); Ahmad et al. (2011).

   In Sejdinovic et al. (2009); Vukobratovic et al. (2009), the UEP is achieved by using the coding windows that containing each other, so the bits in the innermost window will have the highest protection. Ahmad et al. (2011) divides the video data into several partitions, and virtually replicates each partition by different factors in order to achieve different protection levels.

   It should be noted that, all the aforementioned schemes require the data to be partitioned into several importance levels. More often, only two levels are used in their experiments: most important bits (MIB) and least important bits (LIB). Besides, the information about
importance levels should be either coordinated between encoder and decoder beforehand, or explicitly transmitted in the packet headers. Either way will introduce additional coding overhead.

In this section, we propose an enhanced algorithm for DAF that can provide UEP.

2.6.1 Importance Profile

In order to describe the levels of importance, the importance profile, $I(t)$, is introduced. There should be one value assigned to each frame, and the value should reflect the relative ratio of importance between frames. Except for that, we do not impose any other restrictions on importance profile.

The core problem is how to make use of the importance profile. The most commonly used method in existing UEP-based FEC codes is to transform the importance values into some coefficients, and multiply them to the sampling probability. Then, instead of sampling all the packets with uniform distribution, the packets are sampled according to the adjusted sampling probabilities.

Because the importance value reflects the sampling probability and the sampling frequency is determined by code rate, only the relative ratios of importance profile are important. For example, if the user wants the more important bits (MIB) to be sampled with twice the probability of less important bits (LIB), the importance value of packets containing MIB (denoted as $I(MIB)$) and LIB (denoted as $I(LIB)$) could be 2 and 1, respectively. It should be the same if we set $I(MIB) = 300$ and $I(LIB) = 150$.

Because of the scope of this article, this work does not discuss how to determine the importance values for the video data. Therefore, we consider the importance profile as an input from the upper layer, and it is independent of the video codec.

2.6.2 Applying UEP to Optimization Objective

In this section, we will introduce the method to integrate UEP into DAF. In other existing work, such as Masnick and Wolf (1967); Ahmad et al. (2011); Rahnavard et al. (2007), the sender needs to explicitly transmit the importance profile of the containing data to the receiver,
in order to synchronize the encoder and decoder. By containing the importance profile in the headers of packets, the method enlarges the packet size and increases the possibility of packet loss.

In this work, because the sampling probability is already non-uniform by design (a sloped distribution defined by a parameter $\alpha$), we can make use of it to achieve UEP. In order to sample the packets according to importance profile, we need to change the objective of optimization problem defined in (2–18) into the function in (2–31).

$$\arg\min_a \sum_{t=W}^{T-W+1} (ASP_{s,a}(t) - ASP_{s,a}^{TAR}(t))^2,$$

where $ASP_{s,a}^{TAR}(t)$ represents the target ASP for every frame, which could be computed by (2–32).

$$ASP_{s,a}^{TAR}(t) = \overline{ASP}_{s,a} \times I(t) / \overline{I},$$

where $\overline{ASP}_{s,a}$ and $\overline{I}$ represent the average values of $ASP$ and importance profile, respectively, over the current sliding window.

The optimization is to minimize the quadratic distance between the actual values and the targets. Because the relative ratio of importance $I(t) / \overline{I}$ is considered in the target ASP level, it achieves the goal of modifying the sampling probability according to the importance profile within the framework of DAF. The change only needs to be made on the encoder side, and the decoder can decode the packets the same way as DAF.

### 2.6.3 Work with MPC-based DAF

It is also possible to integrate UEP property with the MPC-based DAF. (2–28) can be modified the same way as in (2–31).

$$\arg\min_{a^{\mu}} \sum_{t=\tau}^{\tau+H+W-2} \left( P_{s,a^{\mu}}(t) - P_{s,a^{\mu}}^{TAR}(t) \right)^2,$$

where $P_{s,a^{\mu}}^{TAR}(t)$ represents the target packet probability for every frame, which could be computed by (2–33).
where $P_{s,a}^{TAR}(t)$ represents the target ASP for every frame within a window, which could be computed by (2–34).

$$P_{s,a}^{TAR}(t) = P_{s,a}^{init} \times I(t) / \bar{I}. \tag{2–34}$$

However, the computation of $P_{s,a}^{init}$ should be modified according to the change of the optimization objective. From (2–27) we learn that $P_{s,a}^{init}$ consists of three components. While $P_{init}$ and $\tilde{P}_{s,a}^{init}$ are the deterministic parts, $\hat{P}_s$ represents the predicted ASP level in the future. In this case, the all-zero slope vector ($\hat{a} = 0$) assumption in the original MPC-DAF work might not accurately predict the future ASP level, because it assumes all the windows in that range will use uniform distributions. Instead, we multiply the original $\hat{P}_s$ by the coefficient of $I(t) / \bar{I}$, and get (2–35).

$$P_{s,a}^{init} = P_{init} + \tilde{P}_{s,a}^{init} + \hat{P}_s \times I(t) / \bar{I}. \tag{2–35}$$

### 2.6.4 Optimization Results

In this section, we give a concrete example to illustrate the optimization results of proposed scheme.

We use the CIF video sequence *foreman* as an example. Fig. 2-17A shows the bit rate of this sequence encoded with H.264/AVC. Because we encode the sequence with GOP size of 40 frames, we are using a importance profile that makes the first frame in a GOP 1.2 times
more likely to be sampled than the last frame in a GOP, and all frames in between are linearly continuous. Fig. 2-17B shows the importance profile.

By applying the optimization described in (2–31), we get a slope factor for every window. Using the slope vector, the resulting ASP is shown in Fig. 2-18. The figure also shows the ASP of plain sliding LT code (with uniform sampling distribution within every window) and original DAF. We can observe that compared to the result of DAF, UEP-DAF allocate higher sampling probability to the frames in the front of a GOP than in the back.

2.7 Simulation Experiments and Performance Evaluations

In this section, we conduct simulation experiments of all the proposed schemes. We compare them to some state-of-the-art delay-aware coding schemes, in order to evaluate their performances in different scenarios.

2.7.1 Simulator Setup

We conduct the simulation experiments on Common Open Research Emulator (CORE) Ahrenholz (2010) providing virtualization on application (APP), transport (UDP or TCP) and network (IP) layer controlled by a graphical user interface, and Extendable Mobile Ad-hoc Network Emulator (EMANE) U.S. Naval Research Laboratory, Networks and Communication Systems Branch (2014) for link (MAC) and physical (PHY) layer simulation, with the use of virtual machines of Oracle® VM VirtualBox.
We use CORE to emulate the topology of the virtual network. Two VMs are connected to the virtual network as source (or encoder/sender) node running the client application, and destination (or decoder/receiver) node running the server application. A video is streamed from client to server using different schemes.

EMANE is used for emulation of IEEE 802.11b on PHY and MAC layer of each wireless node. Because of the forward error correction (FEC) nature of fountain code, we do not need the reliable transmission provided by MAC layer, so the retransmission mechanism of 802.11b is disabled for the fountain-code-based schemes. For the simplicity of performance evaluation, we also disable the adaptive rate selection mechanism of 802.11b, and only allow the 11 Mbps data rate to be used. Ad-hoc On-Demand Distance Vector (AODV) protocol is used for routing.

The setup of all the experiments as follows. There are two nodes in the network: a source node and a destination node. The communication path from the source to the destination has one hop. Both nodes are immobile during an experiment.

2.7.2 Schemes for Comparison

We implement nine schemes for comparisons. They are all implemented using the general framework of DAF with different settings and headers, as we mentioned in section 2.4.4. The schemes are abbreviated as follows:

1. **DAF**: the proposed full optimization version of delay-aware fountain code protocol as introduced in section 2.4.

2. **DAF-L**: it is the low complexity version of DAF scheme. It is DAF without using the optimized window-wise sampling distribution. The values in the SlopeF fields are simply set to 0.

3. **DAF-M**: the MPC-based DAF codes as proposed in Section 2.5.2.

4. **DAF-O**: the online version of MPC-based DAF codes as proposed in Section 2.5.3.

5. **UEP-DAF**: the UEP version of DAF codes as proposed in Section 2.6. The importance profile used in this scheme is the same as we introduced in Section 2.6.4, which protects the frames in the front of every GOP.
6. **S-LT**: this scheme is sliding window LT code from Bogino et al. (2007). This SWFC scheme has a fixed number of packets in each window. As we pointed out in section 2.1.1, in order to keep the window from exceeding the allowed time delay, the window size should not be larger than the fewest number of packets in any $T_{Delay}$-frame period. As a result, the $WSize$ fields are set to that number in all packets. Also, the values in the $SlopeF$ fields are set to 0.

7. **Block**: this scheme is block coding. The video file is segmented into the blocks according to the time delay, and they are sent one by one using fountain code. Because of the reason we pointed out in section 2.1.2, the block size is set to the half of the allowed $T_{Delay}$. The step size $\Delta t$ is set to the block size on the encoder. $SlopeF$ is also set to 0.

8. **Expand**: this is the expanding window scheme of Sejdinovic et al. (2009). The beginning position of the windows is fixed, and the newly generated frames are added into the window. All the $StartP$ field is set to 1.

9. **TCP**: this scheme uses TCP protocol to stream video. In order to add delay awareness, the video file is also segmented into the blocks like in “Block” scheme, but they are sent using TCP. For the sake of fairness, the maximum data rate is limited to the same amount as required by the SWFC schemes. Also, the concept of in-time decoding is applied – the packets are deemed as lost if they can not be delivered within allowed time delay. The reliable transmission and congestion control mechanisms of TCP are activated.

All the eight fountain-code-based schemes use the following parameter setting: the packet size $P = 1024$ bytes; for degree distribution, let $\delta = 0.02, c = 0.4$ (as defined in (Luby, 2002, Definition 11)). For TCP scheme, the maximum data rate is limited to the same amount as required by the fountain-code-based schemes for the sake of fairness.

Eight benchmark CIF test sequences are used for our evaluation, which include *foreman, coastguard, mobile, akiyo, bus, news, football and stefan*, provided by Seeling and Reisslein (2012). They are coded into H.264/AVC format using *x264 x264 team* (2016), encapsulated into ISO MP4 files using MP4Box GPAC project (2016), and streamified by *mp4trace* tool from EvalVid tool-set Klaue et al. (2003). The coding structure is IPPP, which contains only one I-frame (the first frame) and no B-frame, and all the rest are P-frames. For the sake of clarity, all the delays shown in the experiments are in the unit of seconds. Because the frame rate for all sequences are 30 frames per second, it is easy to convert the unit between seconds and number of frames.
We conduct 20 experiments for each setting with different random seeds, and take the median value of them as the performance measure. Two results are shown for each set of experiment: in-time decoding ratio (IDR) and file decoding ratio (FDR).

2.7.3 Performance Evaluation for DAF and DAF-L

We conduct the experiments for the evaluation of the following three aspects on the performance: 1) delay and code rate; 2) packet loss rate; 3) step size.

2.7.3.1 Evaluate the effect of delay and code rate

In this case, the distance between the two nodes is carefully set so that the packets with 1024-byte payload will have 10% packet loss rate (PLR). Let $\Delta t = 1$.

Denote by $C$ and $T_{Delay}$ the code rate and the tolerable delay, respectively. It should be noted that, although data rate $R$ is the basic parameter as stated in section 2.1, code rate $C$ is a fair comparative metric among different video sequences. As a result, we use $C$ instead of $R$ in this section.

Fig. 2-19 shows the relations of IDR vs. $C$ of CIF sequence foreman for different $T_{Delay}$. The results of four delays are shown: 0.5, 1, 1.5 and 1.83 seconds. Fig. 2-20 shows the relations of IDR vs. $T_{Delay}$ of CIF sequence foreman for different $C$. The results of four code rates are shown: 1.0, 0.9, 0.85, and 0.75. Only partial results of “block” scheme are shown, because its values are too small to be maintained in the same scale as others. There are two dimensions of variables, $T_{Delay}$ and $C$, so the results of each scheme form a surface. Fig. 2-21 shows five surfaces of the schemes.

The numerical comparative results between different schemes with variant delays and code rates of sequences foreman and coastguard are shown in Table 2-3.

From the results above, we have the following observations:

- Among all, DAF has the highest decoding ratio. As shown in Fig. 2-21, almost the entire surface of DAF is above the other schemes. The performance of DAF-L is lower than DAF, but higher than others. DAF outperforms DAF-L because the overall sampling distribution of DAF is more homogeneous. Proposed schemes improve the decoding ratio when coding resource is insufficient or tolerable delay is small.
Table 2-3. Decoding ratio comparisons between different schemes with variant settings. 
$PLR = 10\%$.

<table>
<thead>
<tr>
<th>Sequence</th>
<th>Code Rate</th>
<th>Data Rate (kbps)</th>
<th>Delay (s)</th>
<th>Scheme</th>
<th>IDR</th>
<th>FDR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DAF</td>
<td>93.99%</td>
<td>98.81%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DAF-L</td>
<td>74.42%</td>
<td>98.29%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>S-LT</td>
<td>66.34%</td>
<td>82.78%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Block</td>
<td>29.32%</td>
<td>29.32%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Expand</td>
<td>56.14%</td>
<td>82.67%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>TCP</td>
<td>68.21%</td>
<td>68.21%</td>
</tr>
<tr>
<td></td>
<td>0.74</td>
<td>3166</td>
<td>0.8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>foreman</td>
<td>0.79</td>
<td>3097</td>
<td>1.2</td>
<td>DAF</td>
<td>95.20%</td>
<td>98.72%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DAF-L</td>
<td>81.35%</td>
<td>97.94%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>S-LT</td>
<td>73.74%</td>
<td>97.26%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Block</td>
<td>27.68%</td>
<td>27.68%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Expand</td>
<td>59.21%</td>
<td>83.64%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>TCP</td>
<td>62.25%</td>
<td>62.25%</td>
</tr>
<tr>
<td></td>
<td>0.90</td>
<td>2872</td>
<td>1.7</td>
<td>DAF</td>
<td>93.73%</td>
<td>98.22%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DAF-L</td>
<td>84.20%</td>
<td>96.90%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>S-LT</td>
<td>81.71%</td>
<td>96.88%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Block</td>
<td>25.14%</td>
<td>25.14%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Expand</td>
<td>61.84%</td>
<td>83.18%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>TCP</td>
<td>57.23%</td>
<td>57.23%</td>
</tr>
<tr>
<td></td>
<td>0.72</td>
<td>2757</td>
<td>0.5</td>
<td>DAF</td>
<td>94.32%</td>
<td>99.11%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DAF-L</td>
<td>69.07%</td>
<td>99.00%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>S-LT</td>
<td>63.60%</td>
<td>86.78%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Block</td>
<td>25.81%</td>
<td>25.81%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Expand</td>
<td>44.30%</td>
<td>72.50%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>TCP</td>
<td>45.10%</td>
<td>45.10%</td>
</tr>
<tr>
<td>coastguard</td>
<td>0.79</td>
<td>2611</td>
<td>0.8</td>
<td>DAF</td>
<td>91.16%</td>
<td>98.86%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DAF-L</td>
<td>75.78%</td>
<td>98.54%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>S-LT</td>
<td>38.57%</td>
<td>98.23%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Block</td>
<td>16.83%</td>
<td>16.83%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Expand</td>
<td>46.90%</td>
<td>73.39%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>TCP</td>
<td>38.01%</td>
<td>38.01%</td>
</tr>
<tr>
<td></td>
<td>0.86</td>
<td>2496</td>
<td>1.2</td>
<td>DAF</td>
<td>91.54%</td>
<td>98.52%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DAF-L</td>
<td>83.72%</td>
<td>97.91%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>S-LT</td>
<td>34.98%</td>
<td>97.86%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Block</td>
<td>23.09%</td>
<td>23.09%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Expand</td>
<td>49.45%</td>
<td>73.94%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>TCP</td>
<td>32.83%</td>
<td>32.83%</td>
</tr>
</tbody>
</table>
The performance of S-LT is lower than two proposed schemes, but higher than others. DAF and DAF-L outperform S-LT because the their window size is bigger.

If $C$ is low enough or $T_{\text{Delay}}$ is large enough, the decoding ratios of all three SWFC schemes converge to 100%. Correspondingly, if $C$ is too high or $T_{\text{Delay}}$ is too small, their performances are equally bad, or DAF may be even worse than the DAF-L scheme. That is because when data rate is extremely limited, DAF makes all the frames unlikely to be decoded at the same time, while in DAF-L scheme, some frames with very low bit rate will be decoded. However, since in those scenarios the video decoding ratios are below 50%, which is too low to be properly viewed, they are not the cases we concern about most.

The decoding ratio of all the schemes is an increasing function of $T_{\text{Delay}}$, and also a decreasing function of $C$. That means larger delay and lower code rate lead to higher overall performance, which meets our expectation. Also, Table 2-3 shows that in order to obtain the decoding ratio at a certain level, we need to balance $T_{\text{Delay}}$ and $C$.

Expanding window scheme Sejdinovic et al. (2009) performs poorer than the three SWFC codes, but better than block coding. That is because: on the one hand, the decoding

Figure 2-19. Relations of IDR vs. code rate of CIF sequence *foreman* when $T_{\text{Delay}} = 0.5, 1, 1.5, \text{and } 1.83$ seconds. Five sliding window schemes are compared. $PLR = 10\%$. $\Delta t = 1$.
Figure 2-20. Relations of IDR vs. delay of CIF sequence foreman when $C = 1.0$, 0.9, 0.85, and 0.75. Five sliding window schemes are compared. PLR = 10%. $\Delta t = 1$.

- Probability of the newly added frames within the $T_{\text{Delay}}$ is low, and it gets lower as more frames are added into the window; on the other hand, it has the largest block length among all the fountain-code-based schemes, so the high coding gain compensates the former drawbacks.

- TCP’s performance is similar to expanding window scheme. The reason is that TCP is not suitable for wireless scenarios where PLR is high Holland and Vaidya (2002). The slow start, congestion avoidance phases and congestion control mechanisms lower its performance.

- Block scheme performs the poorest among all schemes. Since the blocks are too small ($T_{\text{Delay}}/2$) and non-overlapping, the coding overhead is very large Liva et al. (2010).

- The above observations are true for both IDR and FDR. There is always $FDR \geq IDR$, as we pointed out in section 2.3.1. For TCP and Block schemes, there is $FDR = IDR$, because the frames prior to current window will never be decoded in the future.

- Although decoding ratios of DAF are high ($90\% - 99\%$) compared to other schemes, it hardly reaches 100%, due to the limitations of LT code Shokrollahi (2006).
Figure 2-21. Comparison of the decoding ratios of CIF sequence *foreman*. Five delay-aware fountain code schemes are compared. There are two dimensions of variables, $T_{\text{Delay}}$ and $C$. $\text{PLR} = 10\%$. $\Delta t = 1$.

2.7.3.2 Evaluate the effect of packet loss rate

In this case, each configuration of an experiment is conducted three times with the distance between two nodes carefully chosen, so that the end-to-end $\text{PLR}$ of the 1024-byte payload packets will be 5%, 10% and 15%. Let $\Delta t = 1$.

The IDR results are compared in Table 2-4, where N/A means the corresponding decoding ratio is below 10% and unable to recover any consecutive frames, making the actual decoding ratio insignificant. We have the following observations:
Table 2-4. IDR comparisons between different schemes for variant sequences under PLR of 5%, 10% and 15%.

<table>
<thead>
<tr>
<th>Sequence</th>
<th>Code Rate</th>
<th>Delay</th>
<th>Scheme</th>
<th>IDR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>PLR =5%</td>
<td>PLR =10%</td>
</tr>
<tr>
<td>mobile</td>
<td>0.77</td>
<td>0.5</td>
<td>DAF</td>
<td>95.61%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DAF-L</td>
<td>93.22%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>S-LT</td>
<td>66.90%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Block</td>
<td>26.91%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Expand</td>
<td>43.04%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>TCP</td>
<td>82.24%</td>
</tr>
<tr>
<td>akiyo</td>
<td>0.83</td>
<td>1</td>
<td>DAF</td>
<td>94.85%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DAF-L</td>
<td>93.26%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>S-LT</td>
<td>80.68%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Block</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Expand</td>
<td>51.36%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>TCP</td>
<td>76.31%</td>
</tr>
<tr>
<td>bus</td>
<td>0.86</td>
<td>0.5</td>
<td>DAF</td>
<td>93.14%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DAF-L</td>
<td>91.62%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>S-LT</td>
<td>85.72%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Block</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Expand</td>
<td>49.50%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>TCP</td>
<td>87.20%</td>
</tr>
<tr>
<td>news</td>
<td>0.79</td>
<td>0.7</td>
<td>DAF</td>
<td>94.44%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DAF-L</td>
<td>93.71%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>S-LT</td>
<td>84.92%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Block</td>
<td>20.84%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Expand</td>
<td>48.48%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>TCP</td>
<td>68.23%</td>
</tr>
<tr>
<td>football</td>
<td>0.86</td>
<td>0.5</td>
<td>DAF</td>
<td>98.76%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DAF-L</td>
<td>96.90%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>S-LT</td>
<td>66.94%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Block</td>
<td>14.77%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Expand</td>
<td>64.15%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>TCP</td>
<td>84.13%</td>
</tr>
<tr>
<td>stefan</td>
<td>0.88</td>
<td>0.7</td>
<td>DAF</td>
<td>98.49%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DAF-L</td>
<td>98.49%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>S-LT</td>
<td>97.61%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Block</td>
<td>22.53%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Expand</td>
<td>75.25%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>TCP</td>
<td>82.27%</td>
</tr>
</tbody>
</table>
• The relationship of DAF, DAF-L and S-LT remains the same as in case 1 for different \( PLR \): DAF achieves the highest decoding ratio among all schemes; DAF-L scheme is the second best; S-LT performs the worst among the three. That shows the proposed schemes maintain their advantages over the state-of-the-art schemes in a wide range of network conditions.

• The performance of block coding scheme is still the lowest among all schemes.

• The decoding ratio of expanding window scheme Sejdinovic et al. (2009) is proportional to \((1 - PLR)\). The similar phenomenon can be observed from Fig. 2-19. Meanwhile, IDR decreases faster when \( PLR \) gets higher for other fountain-code-based schemes. That is because Sejdinovic et al. (2009) has bigger block size than others. The effect of block size upon the performance curve has been studied in Palanki and Yedidia (2004). As a result, the performance of Sejdinovic et al. (2009) gets better compared to other sliding-window-based schemes when \( PLR \) is high.

• TCP performs relatively well in the cases when \( PLR = 5\% \), but they are extremely inefficient when \( PLR = 15\% \). That is because its performance is very sensitive to packet losses. High loss rate will cause TCP to time out.

2.7.3.3 Evaluate the effect of step size

The setup of this set of experiments is the same as case 1. However, we change the value of \( \Delta t \) in both optimization and coding process, in order to tests its effect on performance. We only compare DAF, DAF-L and S-LT, because they are SWFC schemes and can be affected by \( \Delta t \).

The IDR results are shown in Table 2-5. We observe that IDR of \( \Delta t = 5 \) is lower than \( \Delta t = 1 \) by about 3 percents. There are some factors leading to this.

• Larger \( \Delta t \) gives less slope factors to be optimized in DAF. The objective function in (2–18) may be higher, so the resulting ASP may be more unstable.

• Larger \( \Delta t \) results in smaller window size, because \( W \leq T_{Delay} - \Delta t \) as stated in section 2.1.2.

• Larger \( \Delta t \) leads to smaller overlap between windows, which makes the virtual block size gets smaller.

As we mentioned in section 2.2, the computational complexity of DAF scheme is \( O\left(\frac{T}{\Delta t}^3\right) \), which is inversely proportional to the third power of \( \Delta t \). As a result, larger \( \Delta t \) leads to significantly lower computational complexity in optimization process. \( \Delta t \) can be
Table 2-5. IDR comparisons between different SWFC schemes with variant $\Delta t$. $PLR = 10\%$.

<table>
<thead>
<tr>
<th>Sequence</th>
<th>Code Rate</th>
<th>Delay (s)</th>
<th>Scheme</th>
<th>$\Delta t = 1$</th>
<th>$\Delta t = 5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>foreman</td>
<td>0.79</td>
<td>1.2</td>
<td>DAF</td>
<td>95.20%</td>
<td>92.23%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DAF-L</td>
<td>81.35%</td>
<td>79.54%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>S-LT</td>
<td>73.74%</td>
<td>70.61%</td>
</tr>
<tr>
<td>coastguard</td>
<td>0.79</td>
<td>0.8</td>
<td>DAF</td>
<td>93.16%</td>
<td>90.08%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DAF-L</td>
<td>75.78%</td>
<td>72.54%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>S-LT</td>
<td>38.57%</td>
<td>33.30%</td>
</tr>
</tbody>
</table>

viewed as a tuning knob for DAF to balance computational complexity and desired decoding performance. However, $\Delta t$ does not affect the computational complexity of DAF-L and S-LT, since they do not require optimization.

2.7.4 Performance Evaluation for MPC-based DAF

2.7.4.1 Evaluate the effect of horizon length

In this case, we compare the performance of four DAF-based schemes. For DAF-M and DAF-O, we set $H$ to 1, 5, 10, 15 and 20 to observe the influence of horizon length.

Fig. 2-22 shows the IDR curves of the DAF-based schemes for foreman. Because there are two dimensions of variables, we obtain the IDR curves vs. $C$ when fixing $T_{\text{Delay}} = 1.2$ s in Fig. 2-22A, and also obtain the IDR curves vs. $T_{\text{Delay}}$ when fixing $C = 0.77$ in Fig. 2-22B. Results of DAF-M and DAF-O are plotted with green dashed lines and red dotted lines respectfully. The different horizon lengths $H$ are indicated by the width of lines and the brightness of colors: the thicker and darker lines indicate larger $H$, and the thinner and lighter lines indicate smaller $H$.

The numerical results of them when fixing both delay $T_{\text{Delay}} = 1.2$ s and code rate $C = 0.8$ for foreman are shown in Table 2-6.

From the results above, we have the following observations:

- The decoding ratio of all the schemes is an increasing function of $T_{\text{Delay}}$, and also a decreasing function of $C$. That means larger delay and lower code rate lead to higher overall performance, which meets our expectation.

- Among the DAF-based schemes, the performance of DAF is the highest, while DAF-L is the lowest. The results of DAF-M and DAF-O are generally between those of DAF and
A Fix $T_{\text{Delay}} = 1.2$ s.

B Fix $C = 0.77$.

Figure 2-22. Comparisons of IDR curves for foreman when fixing delay $T_{\text{Delay}} = 1.2$ s and code rate $C = 0.77$. Various values of $H$ are chosen for MPC-O and MPC-M.

Table 2-6. IDR comparisons of DAF-based schemes using different horizon lengths.

<table>
<thead>
<tr>
<th>Scheme</th>
<th>$H$</th>
<th>IDR</th>
</tr>
</thead>
<tbody>
<tr>
<td>DAF</td>
<td>N/A</td>
<td>93.71%</td>
</tr>
<tr>
<td>DAF-L</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DAF-O</td>
<td>20</td>
<td>90.16%</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>88.45%</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>87.80%</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>86.15%</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>80.86%</td>
</tr>
<tr>
<td>DAF-M</td>
<td>20</td>
<td>85.29%</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>83.81%</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>87.03%</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>82.88%</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>68.00%</td>
</tr>
<tr>
<td>DAF-L</td>
<td>N/A</td>
<td>76.52%</td>
</tr>
</tbody>
</table>

DAF-L. The relationships of decoding ratios is consistent with the ASP coverage ratios as seen in Fig. 2-16.

- In general, DAF-M outperforms DAF-O under the same horizon length. That is because DAF-M can be deemed as DAF-O with prefect bit rate prediction. As a result, the gap between DAF and DAF-M is only caused by local bit rate knowledge, while the performance drop from DAF to DAF-O comes from both limited horizon length and inaccuracy of bit rate prediction.

- For DAF-M, the decoding ratio gets higher with the increasing horizon length. However, since the computational complexity increases cubically with $H$, an extra large $H$ is not affordable.
• For DAF-O, longer horizon length does not guarantee better performance. In the example of Table 2-6, the highest IDR of DAF-O is achieved when \( H = 10 \), but not \( H = 20 \). The reason is that in DAF-O, the long-term bit rate prediction will become inaccurate. Due to accumulated long-term prediction error, the performance of DAF-O will reduce when the prediction length increases.

• When \( H = 1 \), the performance of DAF-O is significantly lower than other \( H \) values. That is because it is deducted to a greedy algorithm, and the decision is only based on the history ASP.

2.7.4.2 Compare with the other schemes

In this case, we compare the proposed schemes with the existing video streaming algorithms. To strike a good balance between complexity and performance, we use \( H = 10 \) for MPC-O and MPC-M.

First, we want to show IDR comparisons of the online window-based fountain codes schemes: DAF-O, DAF-L, S-LT, expanding window and block coding. We choose all the combinations of \( T_{\text{Delay}} \in [0.6, 1.5] \) and \( C \in [0.6, 0.9] \) to conduct the experiments on foreman.

There are two dimensions of variables, so the results of each scheme form a surface of IDR. Fig. 2-23 shows five surfaces of the online schemes.

The numerical results obtained by more schemes for more sequences are shown in Table 2-7.

From the results above, we have the following observations:

• Among all the schemes, DAF has the highest performance, followed by DAF-M, both of which are offline algorithms. Considering the computational complexity of DAF-M is orders of magnitude lower than DAF, DAF-M is a more practical offline algorithm.

• Among all the online schemes, DAF-O has the highest performance, followed by DAF-L. As shown in Fig. 2-23, the surface of DAF-O is almost always higher than any other scheme.

• If \( C \) is too high or \( T_{\text{Delay}} \) is too small, the performance of DAF-L may be lower than DAF. The result accords with the ASP coverage ratio in Fig. 2-16 when \( c \) is small, but it is not a very typical scenario since all of them are too low to be properly watched.

• The performance of TCP is relatively low. The reason is that TCP is not suitable for wireless channels where packet loss rate is high Holland and Vaidya (2002). Its congestion control mechanism does not help the performance.
Figure 2-23. Resulting IDR surfaces of online window-based fountain codes schemes for **foreman**.

- Block coding scheme performs the lowest among all schemes, although its ASP coverage ratio is not very low compared to others. The reason is that the blocks are too small and non-overlapping, so the coding overhead is very large *Liva et al.* (2010).

### 2.7.5 Performance Evaluation for UEP-based DAF

We use the CIF sequence **foreman** and H.264/AVC to test the performance of UEP-based DAF. The importance profile used here is the same as we introduced in Section 2.6.4, which protects the frames in the front of every GOP.

In order to show the improvement made by UEP-based scheme, other than In-time decoding ratio (**IDR**) and file decoding ratio (**FDR**), average PSNR is also reported in the experimental results shown in Table 2-8.
We can observe that the proposed UEP-DAF scheme outperforms all the other existing ones, on both decoding ratio and PSNR. As for the comparison between DAF and proposed scheme, although the decoding ratio does not improve greatly, the improvement of PSNR is significant. That is because the UEP protects the frames in the front of every GOP, so the length of error propagation is reduced.

2.8 Summary of Channel Coding Strategy

This chapter proposed a novel channel coding strategy for transmitting videos over the lossy networks: Delay-Aware Fountain codes. They deeply integrate channel coding and video coding. This is the first work to exploit the fluctuation of bit rate in video data at the level of
Table 2-8. Decoding ratio and PSNR comparisons.

<table>
<thead>
<tr>
<th>Code Rate</th>
<th>Delay (s)</th>
<th>Scheme</th>
<th>IDR</th>
<th>FDR</th>
<th>PSNR (dB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.74</td>
<td>0.8</td>
<td>UEP-DAF</td>
<td>94.39%</td>
<td>98.92%</td>
<td>34.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DAF</td>
<td>93.99%</td>
<td>98.81%</td>
<td>28.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DAF-L</td>
<td>74.42%</td>
<td>98.29%</td>
<td>13.3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>S-LT</td>
<td>66.34%</td>
<td>82.78%</td>
<td>8.7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Block</td>
<td>29.32%</td>
<td>29.32%</td>
<td>3.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Expand</td>
<td>56.14%</td>
<td>82.67%</td>
<td>6.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TCP</td>
<td>68.21%</td>
<td>68.21%</td>
<td>7.6</td>
</tr>
<tr>
<td>0.79</td>
<td>1.2</td>
<td>UEP-DAF</td>
<td>95.88%</td>
<td>99.10%</td>
<td>35.4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DAF</td>
<td>95.20%</td>
<td>98.72%</td>
<td>29.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DAF-L</td>
<td>81.35%</td>
<td>97.94%</td>
<td>16.9</td>
</tr>
<tr>
<td></td>
<td></td>
<td>S-LT</td>
<td>73.74%</td>
<td>97.26%</td>
<td>12.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Block</td>
<td>27.68%</td>
<td>27.68%</td>
<td>3.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Expand</td>
<td>59.21%</td>
<td>83.64%</td>
<td>6.9</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TCP</td>
<td>62.25%</td>
<td>62.25%</td>
<td>7.1</td>
</tr>
<tr>
<td>0.90</td>
<td>1.7</td>
<td>UEP-DAF</td>
<td>94.13%</td>
<td>98.62%</td>
<td>34.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DAF</td>
<td>93.73%</td>
<td>98.22%</td>
<td>27.4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DAF-L</td>
<td>84.20%</td>
<td>96.90%</td>
<td>17.9</td>
</tr>
<tr>
<td></td>
<td></td>
<td>S-LT</td>
<td>81.71%</td>
<td>96.88%</td>
<td>17.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Block</td>
<td>25.14%</td>
<td>25.14%</td>
<td>3.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Expand</td>
<td>61.84%</td>
<td>83.18%</td>
<td>6.9</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TCP</td>
<td>57.23%</td>
<td>57.23%</td>
<td>6.1</td>
</tr>
</tbody>
</table>

channel coding, and to incorporate it towards the optimal design of video streaming-oriented fountain codes.

Based on this idea, we developed three coding strategies: DAF, MPC-based DAF and UEP-based DAF. DAF is the fully optimized version. It achieves the highest performance, but suffers from high computational complexity. DAF-M is based on MPC, and its computational complexity is orders of magnitude lower than DAF. MPC-O is the online variant of DAF-M that can be used in live video streaming applications. UEP-based DAF uses a user-specified frame-level importance profile to perform UEP, and the profile does not rely on any specific video coding standard. Because it utilizes the packet header of DAF, the proposed scheme does not need additional coordination between encoder and decoder. The simulation results show that the decoding ratios of our schemes are 15% to 100% higher than the state-of-the-art
delay-aware schemes in a variety of settings. The PSNR of the videos transmitted by proposed schemes are also the highest among existing algorithms.
CHAPTER 3
NETWORK CODING STRATEGIES: FUN AND MIMO FUN CODES

As I stated in Chapter 1, in order to further boost information spreading over multi-hop lossy networks, we cannot solely rely on the channel coding which only utilizes the source and destination nodes. Joint FoUntain coding and Network coding (FUN) is proposed in Huang et al. (2014a) to combine the best features of fountain coding, intra-session network coding, and cross-next-hop network coding. The first section of this chapter will provide an overview of FUN codes.

However, the application of FUN code is limited to opposite-direction transmissions using a shared multi-hop route, so it is impractical for multiple-input and multiple-output (MIMO) transmissions in wireless mesh networks. To widen the applications of FUN code, MIMO FUN is proposed to maximize the MIMO throughput by applying inter-session linear network coding (RLNC) and making use of the shared nature of wireless channels. Based on proposed theory, we design a protocol achieving global throughput optimization by applying local coding schemes on terminal and relay nodes. As such, our MIMO FUN approach is capable of achieving unprecedented high throughput over lossy channels. At the end of this section will show that the proposed approaches achieve higher throughput than the existing network coding schemes for wireless networks.

3.1 FUN Coding Description

3.1.1 FUN Overview

We consider an L-hop network consisting of a pair of end nodes, say Node 1 and Node \( L + 1 \), and \( L - 1 \) relay nodes. Assume that there are two unicast flows between the two end nodes, i.e., a forward flow from Node 1 to Node \( L + 1 \) and a backward flow from Node \( L + 1 \) to Node 1. The hops are indexed from Hop 1 to Hop \( L \) with respect to the forward flow. We propose two coding schemes, i.e., FUN-1 and FUN-2:

- FUN-1 basically combines BATS coding Yang and Yeung (2014) with COPE Katti et al. (2006) for two flows. But FUN-1 is not a simple combination of BATS and COPE; when combining the packets from two flows, the XOR operation is not directly performed on
the received packets, but on the BATS recoded packets; therefore, during the recovery process, a relay node needs to know the transfer matrix of the next-hop node, in addition to its own packet, to recover the BATS coded packet.

- FUN-2 combines BATS coding with RLNC for two flows; each relay node needs to add a new encoding vector to the header of a re-coded packet; only the destination node performs decoding.

Under FUN-1, two sub-layers, i.e., Layer 2.1 and Layer 2.2, are inserted between Layer 2 (MAC) and Layer 3 (IP). Layer 2.1 is for cross-next-hop network coding, similar to the functionality of COPE Katti et al. (2006). Layer 2.2 is for BATS coding Yang and Yeung (2014). At a source node, Layer 2.2 uses a fountain code to encode all native packets from upper layers (similar to the outer code in a BATS code); there is no Layer 2.1 at a source node. At a relay node, Layer 2.1 is used for cross-next-hop network coding and Layer 2.2 is used for intra-session network coding (similar to the inner code in a BATS code); for Layer 2.2, the relay node runs a procedure called FUN-1-2.2-Proc, which performs RLNC within the same batch. At a destination node, Layer 2.2 decodes the coded packets received; there is no Layer 2.1 at a destination node.

Under FUN-2, only one sub-layer, i.e., Layer 2.2, is inserted between Layer 2 (MAC) and Layer 3 (IP). At a source node, Layer 2.2 uses a fountain code to encode all native packets from upper layers (similar to the outer code in a BATS code). At a relay node, if Layer 2.2 receives a packet with FUN-2 switch enabled, it will run a procedure called FUN-2-2.2-Proc for mixing packets from two flows; otherwise, it will run the procedure FUN-1-2.2-Proc, which does not mix packets from two different flows. Note that different from a BATS code, FUN-2-2.2-Proc performs re-coding of batches from two different flows. At a destination node, Layer 2.2 decodes the coded packets received.

Detailed descriptions of FUN-1 and FUN-2 are given below.

3.1.2 FUN-1

Assume a source (Node 1) wants to transmit a file consisting of $K$ native packets to a destination (Node $L + 1$) over $L$ hops. Each packet, denoted by a column vector in $\mathbb{F}_q^T$, has $T$ symbols in a finite field $\mathbb{F}_q$, where $q$ is the field size. The set of $K$ native packets is denoted by
the following matrix

\[ B = [b_1, b_2, \cdots, b_K], \quad (3-1) \]

where \( b_i \) is the \( i \)-th native packet. With an abuse of notation, when treating packets as elements of a set, we write \( b_i \in B \).

The precoding, outer code, inner code and XOR coding of FUN-1 are described in the following.

### 3.1.2.1 Precoding of FUN-1

At a source node, precoding is performed, similar to RaptorQ (RQ) code Luby et al. (2011). The precoding can be achieved by a traditional erasure code such as LDPC and Reed-Solomon code. The precoding of FUN-1 is performed at a source node at Layer 2.2. After precoding, the output packets are further encoded by the outer encoder of FUN-1.

### 3.1.2.2 Outer code of FUN-1

The outer code of FUN-1 is also performed at a source node at Layer 2.2, which is the same as the outer code of a BATS code. Specifically, a source node encodes the \( K \) native packets into a potentially unlimited number of batches, each containing \( M \) coded packets. The \( i \)-th batch \( X_i \) is generated from a subset \( B_i \subseteq B \) \((B \in \mathbb{F}_q^{T \times K})\) by the following operation

\[ X_i = B_iG_i, \quad (3-2) \]

where \( G_i \in \mathbb{F}_q^{d_i \times M} \) is called the generator matrix of the \( i \)-th batch; \( B_i \in \mathbb{F}_q^{T \times d_i} \); \( X_i \in \mathbb{F}_q^{T \times M} \). Matrix \( G_i \) is randomly generated, with all entries independently and identically chosen from \( \mathbb{F}_q \) according to a uniform distribution.

### 3.1.2.3 Inner code of FUN-1

A relay node, after receiving the packets within the same batch, encodes them into new packets by taking random linear combinations. Specifically, random linear network coding (RLNC) is performed at Layer 2.2 within the same batch. Denote by \( Y_{i,i} \) the set of packets in
the $i$-th batch that is correctly received by node $l$, the forward flow evolves as follows

$$
Y_{i,l+1} = \begin{cases} 
X_i E_{i,1}, & l = 1, \\
Y_{i,l} H_i E_{i,l}, & l > 1,
\end{cases}
$$

(3-3)

where $E_{i,l}$ is the erasure matrix of Hop $l$. Specifically, $E_{i,l}$ is an $M \times M$ diagonal matrix whose entry is one if the corresponding packet is correctly received by Node $l + 1$, and is zero otherwise. $H_i \in \mathbb{F}_q^{M \times M}$ is the recoding matrix of an RLNC for the $i$-th batch at Node $l$.

At the destination (Node $L + 1$), denote by $Y_i$ the $i$-th received batch of the forward flow, we have

$$
Y_i \triangleq Y_{i,L+1} = X_i E_{i,1} H_{i,2} E_{i,2} \cdots H_{i,L} E_{i,L} \\
\triangleq X_i H_i,
$$

(3-4)

where $H_i = E_{i,1} H_{i,2} E_{i,2} \cdots H_{i,L} E_{i,L} \in \mathbb{F}_q^{M \times M}$ is called the transfer matrix for the $i$-th batch, which is also added to the header of a corresponding coded packet as a global encoding vector.

Similarly, the inner code for the $j$-th batch of the backward flow is denoted as below

$$
\bar{Y}_j \triangleq \bar{Y}_{j,1} = \bar{X}_j H_{j,1} \bar{H}_{j,2} \bar{E}_{j,2} \bar{H}_{j,2} \bar{E}_{j,1} \triangleq \bar{X}_j \bar{H}_j.
$$

(3-5)

3.1.2.4 XOR coding of FUN-1

At a relay node, the XOR coding and decoding of FUN-1 are performed at Layer 2.1. It is similar to COPE Katti et al. (2006) but only combines the flows from the two neighboring nodes. At Node $l$, if the output queues of Layer 2.2 for the forward flow (from Node $l - 1$) and the backward flow (from Node $l + 1$) both have at least one batch of $M$ re-coded packets, packet-wise XOR operation is performed on both batches to generate $M$ XOR coded packets, i.e., $p_m = y_{i,m} \oplus \bar{y}_{j,m}, \forall m \in \{1, \cdots, M\}$, where $y_{i,m}$ is the $m$-th recoded packet of the $i$-th batch for the forward flow, $\bar{y}_{i,m}$ is the $m$-th recoded packet of the $j$-th batch for the backward flow, and $p_m$ is the $m$-th XOR coded packet. After the XOR operation, the FUN.XOR bit is enabled and the following information is put in the header of each XOR coded packet: 1)
packet ID $m$, 2) the MAC address of the next-hop node of Packet $y_{i,m}$, 3) batch ID $i$ of Packet $y_{i,m}$, 4) the MAC address of the next-hop node of packet $\tilde{y}_{j,m}$, 5) batch ID $j$ of packet $\tilde{y}_{j,m}$, 6) local encoding vectors of packets $y_{i,m}$ and $\tilde{y}_{j,m}$. Otherwise if only one flow has output from Layer 2.2, no operation is performed in Layer 2.1 and the FUN_XOR bit is disabled.

### 3.1.2.5 Decoding of FUN-1

At Layer 2.1, the XOR decoding is performed locally at relay nodes, in which a packet from the forward flow can be recovered by XORing the XOR coded packet with the corresponding packet from the backward flow, i.e., $y_{i,m} = p_m \oplus \tilde{y}_{j,m}, \forall m \in \{1, \cdots, M\}$. Similar operation is performed to recover a packet from the backward flow, i.e., $\tilde{y}_{j,m} = p_m \oplus y_{i,m}, \forall m \in \{1, \cdots, M\}$. At Layer 2.2, however, decoding is performed at the end nodes, i.e., Node 1 and Node $L + 1$, to recover the $K$ native packets. Similar to a raptor code, belief propagation (BP) is used to decode the outer code and inner code of FUN-1.

### 3.1.3 FUN-2

FUN-2 also consists of outer code, inner code, and precoding, which are described as below. The precoding and outer code of FUN-2 are the same as FUN-1. The differences of FUN-2 lie in the inner code and decoding parts. To limit the size of the encoding vector in the packet header, FUN-2 only allows the mixing of two batches from two flows once; i.e., if a packet is already a mixture of two packets from two flows, it will not be re-coded again at a relay node. Also, to alleviate computational burden at relay nodes, the mixed packets will not be recovered immediately but only to be decoded at the two end nodes, i.e., Node 1 and Node $L + 1$.

#### 3.1.3.1 Inner code of FUN-2

The inner code of FUN-2 is similar to the inner code of FUN-1 in the sense that both of them use RLNC. The difference is that, FUN-2 does not perform XOR coding to mix two flows as FUN-1 does, but embeds this function in the inner code of FUN-2. Besides, the mixing of packets from two flows is performed only once, instead of many times. The way of mixing is also slightly different, i.e., through RLNC rather than XOR coding.
Under FUN-2, if the two flows are mixed at Node $l$, the inner coding is the same as FUN-1 until the two flows meet at Node $l$. At Node $l$, the following re-coding is applied to two juxtaposed matrices of received packets:

\[
Z_{i\oplus j,l} = [\bar{Y}_{i,l}, \bar{Y}_{j,l}]H_{i\oplus j,l} = [Y_{i,l-1}H_{i,l-1}E_{i,l-1}, \bar{Y}_{j,l+1}H_{j,l+1}E_{j,l+1}]H_{i\oplus j,l},
\]  

(3-6)

where $Z_{i\oplus j,l} \in \mathbb{R}^{T \times M}$ contains the $M$ re-coded packets generated by Node $l$; $H_{i\oplus j,l} \triangleq [H_{i,l}, \bar{H}_{j,l}]^T \in \mathbb{R}^{2M \times M}$ is the transfer matrix of an RLNC for the $i$-th batch of the forward flow and the $j$-th batch of the backward flow at Node $l$. After inner-encoding, each column of the matrix $H_{i\oplus j,l}$ is added to the global encoding vector of the corresponding coded packets.

All $M$ re-coded packets in $Z_{i\oplus j,l}$ are broadcasted from Node $l$ to both Node $l + 1$ and Node $l - 1$ over the erasure channels of Hop $l - 1$ and Hop $l + 1$, respectively.

\[
Y_{i,l+1} = Z_{i\oplus j,l}E_{i,l},
\]

\[
\bar{Y}_{j,l-1} = Z_{i\oplus j,l}E_{j,l-1}.
\]

Beyond Node $l$, all relay nodes will continue to re-code in the same way as FUN-1. That is, the $i$-th batch of the forward flow and the $j$-th batch of the backward flow will be recoded according to (3-4) and (3-5), respectively.

### 3.1.3.2 Decoding of FUN-2

In the decoding process, the destination node of the forward flow is also a source node of the backward flow. So this destination node can use its known packets of the backward flow to decode the coded packets of the forward flow.
According to (3-4), (3-5) and (3-6), the destination (Node $L + 1$) receives the following batch in the forward flow

$$
Y_{i,j,L+1} = [X_iE_{i,1} \cdots H_{i,l-1}E_{i,l-1} \tilde{X}_jE_{j,L} \cdots \tilde{H}_{j,l+1}E_{j,l+1}]
$$

$$
\times [H_{i,l} \tilde{H}_{j,l}]^T E_{i,l}H_{i,l+1}E_{i,l} \cdots H_{i,L}E_{i,L}
$$

$$
= X_iH_i + \tilde{X}_jE_{j,L} \cdots \tilde{H}_{j,l+1}E_{j,l+1} \tilde{H}_{j,l} \times
$$

$$
E_{i,l}H_{i,l+1}E_{i,l} \cdots H_{i,L}E_{i,L}
$$

$$
\triangleq X_iH_i + \tilde{X}_j\tilde{H}_j,
$$

(3-7)

where $\tilde{X}_j$ are the packets injected to the backward flow which is known to the destination, and $\tilde{H}_j$ is contained in the global encoding vectors. Therefore, $Y_i$ can be recovered by subtracting the latter part

$$
Y_i = X_iH_i = Y_{i,j,L+1} - \tilde{X}_j\tilde{H}_j.
$$

The backward flow can be similarly processed to recover $\tilde{Y}_j$. The rest part of FUN-2 decoding is the same as FUN-1.

### 3.2 MIMO FUN Description

#### 3.2.1 An Intuitive Example of MIMO FUN

In order to give the reader a rough understanding of how MIMO FUN works, we provide a simple example to start with. As the example shown in Fig. 3-1, there are four nodes in the network. Three terminals ($A, B$ and $C$) are both transmitters and receivers, and each of the terminal wants to receive and content sent by the other two. However, because of the range of transmission, they cannot communicate to each other directly. As a result, in order to get the packets from the source, all the packets are needed to be forwarded by the relay node in the middle ($R$). Apparently, there are six individual flows in total – one for each terminal-to-terminal pair. If we treat each flow as an independent uni-cast connection, this process will take twelve transmissions if there is no packet dropped. If a standard multicast routing protocol is used, such as Internet Group Management Protocol (IGMP), and the
Figure 3-1. Three-wheel topology.

packets in $R$ are forwarded in broadcast mode, it needs at least six transmissions — two for each packet to deliver from its source to destinations.

However, if we allow the relay node to do recode on the received packets, it has the potential to reduce the number of transmissions. For example, $A, B$ and $C$ first send its packet (denoted as $P_A$, $P_B$ and $P_C$ correspondingly) to the relay node $R$, and $R$ broadcasts two coded packets that linearly combining all the received packets, i.e. $P_1 = \alpha_1 P_A + \beta_1 P_B + \gamma_1 P_C$ and $P_2 = \alpha_2 P_A + \beta_2 P_B + \gamma_2 P_C$. We assume the parameters are randomly chosen integers, and $P_1$ and $P_2$ are linearly independent. If all the terminals receive the two coded packets correctly, because all of them have their own original packets, each terminal can recover the other two original packets by solving linear equations. This process takes five transmissions instead of twelve for uni-cast or six for ideal multicast. The saved transmissions can be used in sending new data, thus improving bandwidth.

As we will introduce in the rest of this article, the improvement of the proposed scheme will be more significant when packet loss is introduced. Because MIMO FUN allows each node to overhear and store the packets send by neighbor nodes, and a receiver can decode the original packets once receive enough coded packets, there is no need to retransmit any particular packet. In lossy condition of previous example, the relay node only needs to keep sending new mixture of stored packets until all its receivers can decode the original packets.

The benefit is not limited to simple scenarios like the example above. To fully exploit the coding gain in more complex and general networks, it needs careful design of the protocol, which will be discussed in the rest of this chapter.
3.2.2 Algorithm of MIMO FUN

Our scheme is based on the following assumptions:

1. There are two kinds of nodes in the network, terminal nodes and relay nodes. All terminals are both senders are receivers. Each terminal node needs to receive all the packets sent by all other terminal nodes through relay nodes. We call it the all-to-all scenario. Denote $N_T$ and $N_R$ as the number of terminals and relay nodes respectively.

2. The topology and routing information of the whole network is known by all nodes. We assume that the routes where generated beforehand using standard routing algorithms, e.g., B.A.T.M.A.N Neumann et al. (2008).

3. Each terminal has a fixed size of data to send. The size of each packet is all the same as well. Assume each source has $k$ packets to send.

4. We allow all the nodes to store packets. They all have the ability to compute the rank (or degree of freedom, dof) of stored packets. Each of them also knows the rank and coefficients of packets buffered by neighboring nodes.

It should be pointed out that, with the all-to-all assumption, we can also achieve multicast with arbitrary selective destinations, by discarding the unwanted packets after receiving all on the receiver side.

Because each terminal wants to receive all the packets sent by other terminals, and the number of terminals is fixed during the transmission, the received data from different sources can be joined into a bigger chunk of data. If we give each of the terminal nodes an terminal ID from 1 to $N_T$, which are agreed by all the nodes, each terminal will have a same block of data to receive, and the number of packets in the integrated data is $N_T \times k$.

When we join the data from each source together, the problem can be viewed from a different direction: if each terminal node contains a part of the data, how can we spread and aggregate the partial data from each other with the help of relay nodes using the least number of transmissions.

We assume that the number of packets sent by each terminal is $k = 1$ in the rest of this article. The reason is because this work focuses on the inter-session RLNC scheme design, and $k = 1$ can simplify the analysis. As we will see, the cases of $k > 1$ can be easily derived from $k = 1$ case, because RLNC can also be applied on intra-session packets. Moreover, even if the
scheme is using $k = 1$ and there are multiple packets to send from the sources, it only needs to send one packet at a time and use MIMO FUN multiple times.

Here we provide two examples in order to illustrate MIMO FUN. In the following examples, we denotes the terminal nodes alphabetically as $A$, $B$, $C$, etc. Correspondingly, the original packets sent by those terminal nodes are denoted as $P_A$, $P_B$, $P_C$, etc. If each packet is a row vector, as we pointed out previously, the integrated data that every terminal should receive is $P_{\text{integ}} = [P_A, P_B, P_C, ...]^T$, where $[.]^T$ means the transpose operation, so there are $N_T$ rows of packets. When the original packets are recoded at relay nodes, the coded packets are denoted as $P_i = \alpha_i P_A + \beta_i P_B + \gamma_i P_C + ...$, which is the linear combination of the original packets. In the following topologies, we do not consider the overhear outside the direct communication links.

### 3.2.3 Implementation Details

Below we describe the key features of MIMO FUN.

#### 3.2.3.1 Ad-hoc route selection

In MIMO FUN each multicast session determines its route(s) independently of other active sessions. Classical multicast routing techniques may be used.
3.2.3.2 Global optimization

MIMO FUN identifies the type of each node in a global fashion. The relay node randomly combines packets of different flows within the region, and broadcast to forward the mixed packets to improve the dof of all neighboring nodes.

3.2.3.3 Exploiting structure of RLNC to improve XOR type schemes

Although the use of RLNC packets provides additional resiliency to packet losses across the network in general, we argue that these RLNC packets can also improve performance in inter-session coding regions, which typically depend on packet overhearing to be successful and are thus susceptible to losses. To do so, the nodes should not only overhear transmissions corresponding to other flows, but store them in order to expedite the recovery process of each generation of packets. To leverage the RLNC structure, the relay will (i) randomly combine only the data within the RLNC packets, and (ii) keep the coding coefficients of each flow untouched. This feature separates MIMO FUN from approaches that use RLNC and XOR coding.

3.2.3.4 Relay performs RLNC recoding and local feedback

The relay node randomly combines newly received RLNC packets of each flow. If no new packets of one or several flows are available, the relay generates a new RLNC packet by linearly combining the packets in its queue corresponding to that flow (recoding on a per flow basis). The resulting packet is mixed with packets of other flows. Furthermore, MIMO FUN allows the relay to signal to the nodes transmitting packets to that relay when it has received all degrees of freedom, i.e., it has enough coded packets to recover the original data. Although the relay does not decode, signaling to the node upstream allows that node to stop transmissions for that generation of packets.

3.2.3.5 Partial decoding in relay nodes

Since MIMO FUN mixes RLNC packets of different flows, this implies that some level of decoding or, rather a cancellation of the effect of one flow over the other, is needed. We show
that only a partial decoding is needed, if any, to recover linear combinations involving a single flow and that had contributions from other flow.

3.2.4 Example 1: Four-wheel Topology

This example uses a sample network topology as shown in Fig. 3-2A, which is called four-wheel topology. There are five nodes in the network. Four terminal nodes, and there is a relay node in the middle. As the assumption, each node wants to receive the content broadcasted by the other three. The communications only happens between the nodes with links in between as shown in the figure, and terminals cannot directly communicate to each other. As a result, all the packets need to be forwarded by the relay node, which will recode the received packets.

The proposed strategy of coding goes as follows:

1. All the terminals send the packets to $R$. If there are packet losses, terminal should retransmit the lost original packet. After all the packets are delivered, $R$ has 4 packets in the buffer: $[P_A, P_B, P_C, P_D]^T$.

2. $R$ examines the terminals’ dof. Considering the packet already known by each user, sending 3 linearly independent combinations of the 4 buffered packets is adequate for them to decode all the other packets. If all the packets are correctly delivered, sending more than 3 coded packets will not help its neighbors to increase rank. As a result, $R$ broadcasts 3 coded packets:

   \[ P_1 = \alpha_1 P_A + \beta_1 P_B + \gamma_1 P_C + \delta_1 P_D, \]

   \[ P_2 = \alpha_2 P_A + \beta_2 P_B + \gamma_2 P_C + \delta_2 P_D \]

   and

   \[ P_3 = \alpha_3 P_A + \beta_3 P_B + \gamma_3 P_C + \delta_3 P_D. \]

   If there are terminals do not receive enough coded packets to decode all the data, $R$ should continue to send more randomly combined coded packets, e.g. $P_4$, $P_5$, etc., until all the terminals can decode all the data. It should be pointed out that if a terminal has already get full dof, it does not need to overhear and store newly received packets anymore, since it does not increase the dof of stored packets.

3. If there is no packet loss, the coefficients received by each terminal is shown as Fig. 3-3, and all the terminals now have sufficient dof to decode all the data. If some packets are lost and the other coded packets are received, the coefficients in session of “broadcasted by $R$” will be different.

3.2.5 Example 2: Dumbbell Topology

In this example, there are still four terminals, but there are three relay nodes in the middle: $R_1$, $R_2$ and $R_3$. The communications only happens between the nodes with links in between as shown in Fig. 3-2B. It is called dumbbell topology here.
A's Decoding Coefficients:

\[
\begin{array}{cccccc}
A & B & C & D & Y \\
\alpha_1 & \beta_1 & \gamma_1 & \delta_1 & P_A \\
\alpha_2 & \beta_2 & \gamma_2 & \delta_2 & P_1 \\
\alpha_3 & \beta_3 & \gamma_3 & \delta_3 & P_2 \\
\end{array}
\]

Known by A:

\[
\begin{array}{cccccc}
A & B & C & D & Y \\
0 & 1 & 0 & 0 & P_A \\
0 & 0 & 1 & 0 & P_1 \\
0 & 0 & 0 & 1 & P_2 \\
\end{array}
\]

Broadcasted by R:

\[
\begin{array}{cccccc}
A & B & C & D & Y \\
\alpha_1 & \beta_1 & \gamma_1 & \delta_1 & P_A \\
\alpha_2 & \beta_2 & \gamma_2 & \delta_2 & P_1 \\
\alpha_3 & \beta_3 & \gamma_3 & \delta_3 & P_2 \\
\end{array}
\]

B's Decoding Coefficients:

\[
\begin{array}{cccccc}
A & B & C & D & Y \\
\alpha_1 & \beta_1 & \gamma_1 & \delta_1 & P_B \\
\alpha_2 & \beta_2 & \gamma_2 & \delta_2 & P_1 \\
\alpha_3 & \beta_3 & \gamma_3 & \delta_3 & P_2 \\
\end{array}
\]

Known by B:

\[
\begin{array}{cccccc}
A & B & C & D & Y \\
0 & 1 & 0 & 0 & P_B \\
0 & 0 & 1 & 0 & P_1 \\
0 & 0 & 0 & 1 & P_2 \\
\end{array}
\]

Broadcasted by R:

\[
\begin{array}{cccccc}
A & B & C & D & Y \\
\alpha_1 & \beta_1 & \gamma_1 & \delta_1 & P_B \\
\alpha_2 & \beta_2 & \gamma_2 & \delta_2 & P_1 \\
\alpha_3 & \beta_3 & \gamma_3 & \delta_3 & P_2 \\
\end{array}
\]

C's Decoding Coefficients:

\[
\begin{array}{cccccc}
A & B & C & D & Y \\
\alpha_1 & \beta_1 & \gamma_1 & \delta_1 & P_C \\
\alpha_2 & \beta_2 & \gamma_2 & \delta_2 & P_1 \\
\alpha_3 & \beta_3 & \gamma_3 & \delta_3 & P_2 \\
\end{array}
\]

Known by C:

\[
\begin{array}{cccccc}
A & B & C & D & Y \\
0 & 0 & 1 & 0 & P_C \\
0 & 0 & 0 & 1 & P_1 \\
0 & 0 & 0 & 0 & P_2 \\
\end{array}
\]

Broadcasted by R:

\[
\begin{array}{cccccc}
A & B & C & D & Y \\
\alpha_1 & \beta_1 & \gamma_1 & \delta_1 & P_C \\
\alpha_2 & \beta_2 & \gamma_2 & \delta_2 & P_1 \\
\alpha_3 & \beta_3 & \gamma_3 & \delta_3 & P_2 \\
\end{array}
\]

D's Decoding Coefficients:

\[
\begin{array}{cccccc}
A & B & C & D & Y \\
\alpha_1 & \beta_1 & \gamma_1 & \delta_1 & P_D \\
\alpha_2 & \beta_2 & \gamma_2 & \delta_2 & P_1 \\
\alpha_3 & \beta_3 & \gamma_3 & \delta_3 & P_2 \\
\end{array}
\]

Known by D:

\[
\begin{array}{cccccc}
A & B & C & D & Y \\
0 & 0 & 0 & 1 & P_D \\
0 & 0 & 0 & 0 & P_1 \\
0 & 0 & 0 & 0 & P_2 \\
\end{array}
\]

Broadcasted by R:

\[
\begin{array}{cccccc}
A & B & C & D & Y \\
\alpha_1 & \beta_1 & \gamma_1 & \delta_1 & P_D \\
\alpha_2 & \beta_2 & \gamma_2 & \delta_2 & P_1 \\
\alpha_3 & \beta_3 & \gamma_3 & \delta_3 & P_2 \\
\end{array}
\]

Figure 3-3. Decoding coefficients when dof is full.

The proposed strategy of coding goes as follows:

1. Similar to the first step in four-wheel topology, all the terminals send the packets to its nearest relay nodes, and retransmit if packet is lost. So, \( R_1 \) has 2 packets in the buffer: \([P_A, P_D]\), and \( R_3 \) has 2 packets in the buffer: \([P_B, P_C]\). \( R_2 \) has nothing buffered right now.

2. The relay nodes examine the dof of terminals on each next hop. Let us focus on \( R_1 \) first: it is directly connected to \( A \) and \( B \), but \( C \) and \( D \) is only reachable through \( R_2 \). By detecting the buffered dof of neighbors, it finds out the random mixtures of buffered packets \( P_A \) and \( P_D \) can improve the ranks of node \( A \) and \( D \) by 1, and improve the rank of \( R_2 \) by 2. As a result, sending two coded packets will be a waste for node \( A \) and \( D \), only one coded packet is sent for now. The condition of \( R_3 \) is similar, only is broadcasted.

3. \( R_2 \) also overheard the coded packets broadcasted by \( R_1 \) and \( R_3 \).

4. For \( R_2 \), the mixture of \( P_1 \) and \( P_2 \) will increase the ranks for all the terminals. Without loss of generality, let the coded packet and broadcast it.

5. \( R_1 \) and \( R_3 \) received \( P_3 \).

6. With newly received \( P_3 \), both \( R_1 \) and \( R_3 \) could increase the ranks for all terminals by broadcasting the mixture of all buffered packets. For \( R_1 \), without loss of generality, let the coded packet, where is linearly independent. Similarly, \( R_3 \) broadcast the coded packet.

7. \( R_2 \) overheard the coded packets broadcasted by \( R_1 \) and \( R_3 \). Notice that \( R_2 \) now has the full dof of all packets, so it can generate the code of arbitrary mixture of \( A, B, C \) and \( D \).

8. \( R_2 \) broadcasts the mixture of all packets.
9. $R_1$ and $R_3$ received $P_3$. Notice that all the relay nodes now have the full dof of all packets.

10. $R_1$ and $R_3$ sends the linearly independent coded packet that increase the rank of all users.

11. Finally, the ranks of all the terminals become full.

    All the terminals now have sufficient dof to decode all the packets.

### 3.3 Comparison Schemes and Numerical Results

In this section, we will compute the mathematical expectation of number of packets needed to send in different schemes, and simulate the procedure in TDMA network to evaluate the performance of proposed scheme. Let us focus on the wheel topology for analysis. That is one relay node in the middle surrounded by $n$ terminals as shown in Fig. 3-4.

For simplicity of analysis, the loss rate between relay node and all the terminals are assumed to be the same ($p_e$) and it is bidirectional. The overhearing between nodes are ignored.

Our proposed scheme is compared to the following existing schemes:

1. Naive broadcast scheme (with ACK and retransmission)
2. COPE-like algorithm Katti et al. (2006)
3. CORE Krigslund et al. (2013)

#### 3.3.1 Naive Broadcast Scheme w/ ACK and Retransmission

This broadcasting scheme relies on MAC layer retransmissions to deliver data packets. There are two phases. i) The terminals send their packet to relay node. In order to guarantee
delivery, a node will retransmit its packet if the relay node does not receive it. ii) Relay node in the middle directly broadcasts any packet it received to all the terminals. However, to ensure delivery, the relay node will retransmit the packet until each of the terminal receives it at least one time.

The mathematical expectation of packets needed to send is the weighted summation of the probabilities: \[ E[X] = \sum_{i=1}^{\infty} i \times pr_i \], where \( i \) denotes the number of packets needed to send, and \( pr_i \) denotes the probability of successful delivery using exact \( i \) packets (called \( i \)th delivery in the rest of this article).

In the first phase, for each terminal, the probability of \( i \)th delivery is

\[ pr_i = (1 - p_e) p_e^{i-1} \tag{3-8} \]

So,

\[ E = \sum_{i=1}^{\infty} i (1 - p_e) p_e^{i-1} = 1/(1 - p_e) \tag{3-9} \]

Using inclusion-exclusion principle, the probability of “\( i \)th delivery when there \( n \) receivers, and each receiver must receive the packet at least once” is:

\[ pr_i^{(n)} = \sum_{k=1}^{n} (-1)^{k-1} \binom{n}{k} [(1 - p_e) p_e^{i-1}]^k (1 - p_e)^{n-k} \tag{3-10} \]

We can observe that \( pr_i^{(1)} = pr_i \).

As a result, the probability of \( i \)th delivery in the second phase is \( pr_i^{(n-1)} \), since each packet only needs to be received by \( n - 1 \) terminals (except for the owner of that packet). Since both terminals and relay node need to deliver \( n \) packets respectfully, the total expectation is

\[ E[X] = n \sum_{i=1}^{\infty} i \times pr_i^{(1)} + n \sum_{i=1}^{\infty} i \times pr_i^{(n-1)} \tag{3-11} \]
3.3.2 COPE-like Algorithm

We assume that the MAC protocol will retransmit data packets in the links from terminals to $R$ and will retransmit XORed packets from $R$ until receivers acknowledges reception. Terminal $#j, j \neq i$ must overhear the transmissions from $R$ in order to capture an XORed packet as in Hundebøll et al. (2012). We make this assumption to maintain compatibility to commercial systems because MAC layers in wireless networks, e.g., WiFi, provide Automatic Repeat reQuest (ARQ) mechanisms for unicast but not for broadcast transmissions. Incoming packets to the relay are stored in a queue corresponding to the corresponding flow.

This scheme also has two phases. The first phase is the same as previous one. In the second phase, relay node will broadcast the XOR combination of the packets from two sources, namely $P_1 \oplus P_2, P_2 \oplus P_3, \ldots, P_{n-1} \oplus P_n$. The delivery confirmation is performed in per-packet manner.

Since there are $n-1$ XORed packets needed to broadcast, the total expectation of packets using COPE-like algorithm is

$$E[X] = n \sum_{i=1}^{\infty} i \times pr_i^{(1)} + (n - 1) \sum_{i=1}^{\infty} i \times pr_i^{(n)}$$ (3-12)

3.3.3 CORE Scheme

The simplest CORE scheme. The relay performs intersession coding every time a coding opportunity is detected, i.e., a new RLNC packets are received from each source. In the absence of coding opportunities the relay falls back to forwarding received RLNC packets. Sources send RLNC packets to the destinations with no recoding at the relay. Packets are transmitted using unicast sessions, as in COPE-like, allowing retransmissions. When transmitting from the relay, the destination with the highest loss probability is chosen as receiver, and the other destination is forced to overhear. If link quality is the same, destination is chosen uniformly at random.
3.3.4 MIMO FUN

The propose scheme also has two phases. The first phase is the same as previous ones. In the second phase, relay node will broadcast the linear combinations of the packets from all sources. The ith coded packet is denoted as $C_i = \alpha_{i,1}P_1 + \alpha_{i,2}P_2 + \cdots + \alpha_{i,n}P_n$. Different from COPE-like scheme, the terminals will not confirm the coded packets one by one. Instead, each terminal will acknowledge to the relay node once it collects enough DoF to decode all the packets. When all the terminals acknowledge they have enough DoF, the relay node stops broadcasting.

Here, the probability of “ith delivery when there are n receivers, and each receiver needs to receive at least l coded packets” is

$$p_{i}^{(n,l)} = \sum_{k=1}^{n} (-1)^{k-1} \binom{n}{k} \left[ \binom{i-1}{l-1} (1-p_e)^l p_e^{i-l} \right]^k$$ (3-13)

We can observe that $p_{i}^{(n,1)} = p_{i}^{(n)}$.

As a result, the probability of ith delivery in the second phase is $p_{i}^{(n,n-1)}$, since each source needs to receive at least $n-1$ different coded packets to decode all the other packets. So, the total expectation is

$$E[X] = n \sum_{i=1}^{\infty} i \times p_{i}^{(1)} + \sum_{i=1}^{\infty} i \times p_{i}^{(n,n-1)}$$ (3-14)

3.3.5 Simulation Results

We conduct the simulation of wheel topology on MATLAB. The network is TDMA. The range of $n$ is [2, 6]. The range of packet loss rate is [0.0, 0.7]. We did not simulate the cases when $n > 6$, because the overhearing effect will be significant if the terminals are so dense, hence lowering the fidelity of the simulation.

The solid lines are simulated results, and the broken lines are theoretical results. We observe the follows from the results:

1. The theoretical results fit with the simulated results very well.
2. Less packets to send means higher performance. As a result, in general, the performance of MIMO FUN is greater than COPE-like algorithm, and COPE-like algorithm outperforms naive broadcasting.

3. When $n = 2$, the performances of COPE-like and MIMO FUN are practically the same. The difference of the two gets bigger when $n$ is bigger.

### 3.4 Experimental Results

We use CORE and EMANE as the emulators. We use 802.11b in MAC and PHY layer with CTS/RTS enabled. The transmission rate (of header and payload) is set to 1 Mbit/s. In all the scenarios, all terminals are both senders and receivers. The task is each terminal sending its message to all the other terminals.

Three topologies are used in our experiments. The wheel topology is one relay node in the middle surrounded by $n$ terminals as in Fig. 3-2.

Two variants of wheel topology are tested:
1. Wheel-3
2. Wheel-4
Table 3-1. Performance comparison between different topologies using MIMO FUN.

<table>
<thead>
<tr>
<th>Topology</th>
<th>No. of flows</th>
<th>Payload size (Byte)</th>
<th>Max hops</th>
<th>Avg. duration (s)</th>
<th>Throughput (Kbit/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wheel-3</td>
<td>6</td>
<td>1024</td>
<td>2</td>
<td>0.0696</td>
<td>689.47</td>
</tr>
<tr>
<td>Wheel-4</td>
<td>12</td>
<td>1024</td>
<td>2</td>
<td>0.158</td>
<td>607.82</td>
</tr>
<tr>
<td>Dumbbell</td>
<td>12</td>
<td>1024</td>
<td>4</td>
<td>0.268</td>
<td>358.22</td>
</tr>
</tbody>
</table>

The second type of topology is called dumbbell topology.

3. Dumbbell

It contains three tiers: A, B, C and D are terminals; R1 and R3 are edge routers; R2 is backbone router.

There is a uniform distance of 22 m between each node that linked in the figures. We set the transmission range to 25 m so that stations can communicate only with the two adjacent neighbors. The carrier sense range is set to 50 m, i.e., nodes can sense two-hop neighbors. We manually input the routing table into each node before starting the experiments.

The corresponding MIMO FUN application on each node is executed at the same time using a timer. The duration time is calculated based on the disparity between the time when first packet is sent by any node and when all nodes receive all the packets. Each experiment is run for 10 times, and the average number is recorded as the duration.

The resulting throughput considers 3 factors: payload size (m, in Byte), number of end-to-end flows (k) and duration time (t). So, \( \text{Throughput} = \frac{8 \times m \times k}{t} \)

We compare our scheme to unicast TCP. The topology and 802.11b set-up are the same. The packet size is set to 1024 Bytes. Each end-to-end flow is transmitted individually. In order to get the stable throughput result, we send 1 MB data in each flow, and discard the first 1 KB.

TCP scheme is implemented and tested in two ways. First, we test each flow separately, and calculate their average throughput. That result is equivalent to each terminal node transmitting sequentially. Second, we test each scenario with each terminal node transmitting
Table 3-2. Performance comparison between different schemes.

<table>
<thead>
<tr>
<th>Topology</th>
<th>Scheme</th>
<th>Throughput (Kbit/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wheel-3</td>
<td>MIMO FUN</td>
<td>689.47</td>
</tr>
<tr>
<td></td>
<td>TCP</td>
<td>382.33</td>
</tr>
<tr>
<td></td>
<td>TCP w/ contention</td>
<td>311.4</td>
</tr>
<tr>
<td>Wheel-4</td>
<td>MIMO FUN</td>
<td>607.82</td>
</tr>
<tr>
<td></td>
<td>TCP</td>
<td>382.33</td>
</tr>
<tr>
<td></td>
<td>TCP w/ contention</td>
<td>281.76</td>
</tr>
<tr>
<td>Dumbbell</td>
<td>MIMO FUN</td>
<td>358.22</td>
</tr>
<tr>
<td></td>
<td>TCP</td>
<td>192.12</td>
</tr>
<tr>
<td></td>
<td>TCP w/ contention</td>
<td>102.87</td>
</tr>
</tbody>
</table>

at the same time, which will lead to heavier channel contention. The final results are shown below.

The resulting throughput of TCP fits the conclusion of Hofmann et al. (2007).

### 3.5 Summary of Network Coding Strategy

This chapter is concerned with the network coding solution to the problem of information spreading over lossy communication channels. To address this problem, a joint FoUntain coding and Network coding (FUN) approach has been proposed. The novelty of our FUN approach lies in combining the best features of fountain coding, intra-session network coding, and cross-next-hop network coding. This chapter also proposed MIMO FUN, a wireless meshed networks protocol designed to optimize throughput of all-to-all transmission scenarios. MIMO FUN does inter-session RLNC coding on the packets from different flows at intermediate relay nodes. It reduces the number of transmissions by exploiting the broadcast nature of wireless channels, and enhances the reliability by allowing neighbors to overhear and store coded packets. The designed MIMO FUN protocol improves the coding efficiency by probing the received dof status through per-hop ACK, and maximizing the information contained in each transmission of coded packet, so that the dofs' of all its neighbors can benefit from it. Our numerical and simulation results show that MIMO FUN outperformed all other considered schemes for a wide range of conditions.
CHAPTER 4
SOURCE CODING STRATEGY: RATE CONTROL FOR CLOUD GAMING VIDEO

Cloud gaming, also called gaming on demand, is a new kind of service that provides real-time video game experience to the players over the Internet. Although cloud gaming services are getting more and more popular recently, its performance is highly limited by the network bandwidth and latency.

In this chapter, we propose some rate control strategies to address this problem, which is from a source coding perspective. This work makes use of the unique characteristics of human visual system (HVS) of video game players to assist gaming video rate control. Discussions about the characteristics of game players' HVS are conducted. Then, some schemes of extracting region of interest and key frames from gaming videos are raised. Based on that, a low-complexity Macro-block level rate control scheme is proposed based on region of interest and scene-change detection. Since the proposed work is the first one to solve this problem, it shows great potential for the development of the cloud gaming industry.

4.1 Cloud Gaming

Cloud gaming is a new popular Internet service that combines the concepts of cloud computing and on-line gaming. It provides the entire gaming experience to the gamers by using the resource of the remote computing servers. Because all the graphics computing and data processing are done in the remote data center, the user’s terminal device is nothing more than a controller (mouse, keyboard or game controller) plus a monitor. The player no longer needs to buy expensive and cutting-edge gaming hardware, like graphics card and big RAM, but still enjoy the latest game. The terminal devices only need broadband Internet connections and the ability to display High Definition (HD) video.

An overview flow chart is shown in Fig. 4-1. As explained in Shea et al. (2013), on the client side, the user controls the game just like on a local device, such as PC, TV or mobile devices. Every time the player performs an operation, such as pressing a key, moving the mouse, or using the controller, the cloud gaming system will send the controlling signals to
the remote game servers through Internet. On the server side, the remote servers receive the controlling signals and execute the game programs accordingly. Usually, it involves intensive graphics computation to generate high-quality pictures in real time. Then, the servers will stream the compressed gaming video back to users’ devices. All the gaming program execution, graphic computation and video compression are done on the remote game servers. There will be a continuous control stream from user to server and a continuous video stream from server to user all the time until the session is disconnected.

While it may reduce hardware costs for users, and increase the profit for developers and publishers by reducing the expenditures on retail chains, it also raises a lot of new challenges, especially for the service quality in terms of bandwidth and latency for the underlying network. Table 4-1 shows a list of Pros and Cons of cloud gaming compared with traditional gaming.

As shown in Table 4-1, a good gaming system must balance the high performance and good accessibility. Recently, cloud gaming becomes a very hot trend in game industry. Many cloud gaming platforms are getting popular, especially after the leading game console corporations all announced that they will integrate cloud gaming systems into their latest game consoles. However, although they provide very impressive gaming experience, it seems that the only bottleneck that hiders people from using them is the high bandwidth requirement. Thus, solving this problem may be a matter of life and death for this business.
Table 4-1. Pros and cons of cloud gaming.

<table>
<thead>
<tr>
<th>Advantages</th>
<th>Trad. Gaming</th>
<th>Cloud Gaming</th>
</tr>
</thead>
<tbody>
<tr>
<td>No download and install time</td>
<td>X</td>
<td>✓</td>
</tr>
<tr>
<td>No expensive hardware is needed</td>
<td>X</td>
<td>✓</td>
</tr>
<tr>
<td>Accessible from any platform</td>
<td>X</td>
<td>✓</td>
</tr>
<tr>
<td>Syncing saved game everywhere</td>
<td>X</td>
<td>✓</td>
</tr>
<tr>
<td>Easy to get free trial for gamer</td>
<td>X</td>
<td>✓</td>
</tr>
<tr>
<td>Easy to distribute for publisher</td>
<td>X</td>
<td>✓</td>
</tr>
<tr>
<td>Social features (spectators, brag clips, etc.)</td>
<td>X</td>
<td>✓</td>
</tr>
<tr>
<td>Allowing to play off-line</td>
<td>✓</td>
<td>X</td>
</tr>
<tr>
<td>No lag</td>
<td>✓</td>
<td>X</td>
</tr>
<tr>
<td>Supporting HD resolution</td>
<td>✓</td>
<td>X</td>
</tr>
</tbody>
</table>

When cloud gaming is based on a network of a relatively low quality condition, e.g. playing cloud games on a wireless mobile device, users still want the gaming experience to be good and smooth as well. In order to provide a decent and stable video streaming quality under a given network condition (or a limited bit rate), rate control (RC) of video coding must be performed. Compared to RC for ordinary videos, RC in clouding gaming have more restrictions, like sensitivity to latency, demand for high image quality in key frames. Although many challenges are faced, many unique characteristics of human visual system (HVS) of video game players can also be exploited to improve bandwidth efficiency.

4.2 Related Work

Since cloud gaming is a newest concept, its related research is not fully conducted yet. To the best of the authors’ knowledge, almost all related researches on this topic are conducted after 2010. By now, most of the papers are discussing the quality of experience (QoE) evaluation scheme, like Jarschel et al. (2011), and measurement of latency, like Chen et al. (2011). Hobfeld et al. (2012) pointed out several challenges related to cloud application’s QoE management. However, no one proposed the rate control scheme for cloud gaming, which is a crucial problem as well.

As stated in Shea et al. (2013), the major cloud gaming providers Gaikai and Onlive both use versions of the H.264/AVC encoder. Although there are a lot of rate control schemes are proposed for H.264 standard, like Liu et al. (2007, 2006); Yan and Sun (2012); Sun and
Yan (2011b); Lee et al. (2007a); Chen and Hsu (2011), most of them are for ordinary videos, but not specifically considering the features of video gamers’ HVS. The existing rate control methods have bad performance on cloud gaming video, essentially because they are designed for non-interactive applications, and not directly applicable to the interactive situations.

On the other hand, as the growing emphasis on QoE-based evaluation, more and more researchers are interested in ROI-based video coding. Accordingly, some work focuses on ROI-based video rate control for H.264, like Li et al. (2006); Sun et al. (2006); Liu et al. (2008); Lee and Yoo (2011); Kim et al. (2010). However, because the ROI extraction method for arbitrary videos is a very hard problem itself, ROI based rate control may not achieve its ideal performance. At the worst cases, the video viewer may get a worse experience because of the wrong ROI detection. For example, many of the ROI detection algorithms for the video rate control are designed for the conversational and head-and-shoulder types of video sequences, like Sun et al. (2006); Liu et al. (2008). In Liu et al. (2008), the ROI detection methods are mostly implemented as face detection and motion detection, which limit their scope of application. Furthermore, limited by the computational resources provided by the real-time encoder, the ROI detection is further simplified to a skin tone detection and the residue between two consecutive frames, which makes the detected ROI very inaccurate for non-conversational scenarios. Last but not least, the algorithms are designed for processing videos with QCIF resolution (176 × 144 pixels), not high resolution videos. There are also some papers using motion information as the basis of ROI and adjusting the coding scheme accordingly, such as Lee and Yoo (2011); Kim et al. (2010).

4.3 ROI and Key Frame Patterns for Gaming Video

Since people are more sensitive to the areas where they are interested in, it is reasonable to enhance the region of interest (ROI) while sacrificing the non-ROI regions when the overall coding and transmission resources are limited.

Compared with general videos, encoding cloud gaming videos has three convenient and exceptional features that can be exploited to enhance the performance. First, people are
far more concentrative on the ROI while playing games than watching ordinary videos, so the difference between ROI and non-ROI can be bigger and still does not lower the viewer’s experience. Second, the transmitted gaming video is originally generated by the game programs and graphics processors, so it is very convenient for the game developers to actively provide the side information to help the video encoder, such as foreground and background, depth of an object, scene-change position, etc. It is far more accurate and efficient than passively analyzing the ordinary video. Third, the gaming servers are equipped with powerful GPUs and DRAMs. The massive computational resources can supply the overheads brought by rate control for HD videos.

More generally speaking, these features can be further extended to all interactive VOD systems, which involve both users’ control signals and servers’ video feedback, such as remote desktop services, interactive remote surveillance camera, etc. Also, these applications all need to balance the video quality and the bandwidth occupation. When using these services, people also have very concentrated ROIs. For example, remote desktop users may be more interested in the content in the “top” windows than the “backgrounds”; and they may be more interested in the things around the cursor than others. For simplicity, this work will only concentrate on cloud gaming services, but other similar applications can also be adapted in the proposed framework.

4.3.1 ROI Extraction

As stated above, video gamers, while playing, pay more concentration on ROI than ordinary video viewers do. It is also important to realize that the patterns of ROI in gaming videos are more regular and more accessible than those in arbitrary videos.

For example, for first-person shooter games (FPS), the ROI is always on the middle of the screen, because players will always face to the direction they feel interested. Player is also interested in his/her weapon, status and mini map. An example is shown in Fig. 4-2(a), where the red mask represents ROI. For racing games (RAC), the track, player’s car, dashboard, ranking information and mini map will be ROI, but the background may be not so important
A first-person shooter game (FPS)  
B Racing game (RAC)

Figure 4-2. Examples of the ROI of different types of games. The red masks represent ROI.

during a tense racing game. Thus, the necessary streaming bits can be reduced by lowering
the quality of non-important parts. An example is shown in Fig. 4-2(b), where the red mask
represents ROI.

There are two ways to get ROI. First one is given by the game developer: because the
graphics are generated by the game program, the developers can specify the important regions
for video encoders. The flowchart of an RC-aware cloud gaming system demonstrating the
relationship between the modules is shown in Fig. 4-3. As the right part of the figure shows,
along with the game program, which takes input from user and provides graphic models for
graphics processor to render, there is a module to determine the importance value for each
object. For example, in the racing game scenario, the objects of player’s car, dashboard has
the highest importance value, the track has the lower importance value, and other background
objects have the lowest importance value. In our scheme, the importance value and ROI value
are interchangeable, because the more important an object is, the more interested one may
be in its region on the screen. After the object importance values are given to the graphics
processor, the ROI values for each pixel can be generated along with the original rendered
picture. Then, the RC module can make use of the generated ROI map to perform ROI-based
rate control.

Even if the accurate ROI provided by game developer is unavailable in a video game, there
is a second way to get ROI: extracted by the video encoder: because of the similar layout of
the same game genre, as stated above, it is not as hard as it for arbitrary videos. For the other
information like scene change and acceptable frame rate, which will be discussed later, can also be obtained through either of the two ways. In this work, the study focuses on performing rate control in the codec, not extracting ROI, so it is assumed that the ROI of every frame is already known.

In the simulation, the ROIs are extracted by manual appointment and object tracking. Fig. 4-4 and 4-5 show two examples of ROI detection and extraction. In these figures, (a) shows the original frame. (b) and (c) show the ROI mask over the frame. The blue areas represent ROI, and the less transparent, the more important the area is. Take Fig. 4-4 as an example, the inner circle of the screen is colored as 45% blue, outer circle as 15% blue, and all other ROI areas as 30% blue. It was observed from the experiments that the ROI of RAC games
generally changes faster than other game genres. In the contrast, for the game genres like FPS and Role-playing game (RPG), the ROI layout changes very little or even keeps still.

Because the lowest level of rate control is Macro-block (MB) level, the ROI information is translated to the ROI importance at the MB level. An MB typically consists of $16 \times 16$ samples, so it is the default MB size in our work. In our scheme, the bigger ROI value means it is more important and it needs more protections during video transmission.

The procedure for extracting MB-level ROI weighted values from detected ROI mask is given in the following four steps.

4.3.1.1 Downscaling

The first step is to downscale the ROI to MB-level, as (4–1).

$$ROI_{MB}[i] = \frac{1}{16 \times 16} \sum_{j=r_i}^{r_i+15} \sum_{k=c_i}^{c_i+15} ROI_{frame}[j, k]$$

where $i$ is the MB index. $r_i$ and $c_i$ are row and column position of the top left corner of the $i^{th}$ MB in the frame (assuming the origin of the whole frame is on the top left corner).

$ROI_{frame}$ denotes the ROI map of the whole frame, and $ROI_{frame}[j, k]$ denotes the ROI value of the pixel on $j^{th}$ row and $k^{th}$ column of the image. $ROI_{MB}$ is the ROI value for each MB, which is an average ROI value of all the pixels in this MB.

4.3.1.2 Normalization

The second step is to normalize values as (4–2).
\[ \lambda[i] = \frac{ROI_{MB}[i]}{\overline{ROI}_{MB}} - 1 \]  

(4-2)

where \(i\) is the MB index. \(\lambda[i]\) is the MB-level importance weight. \(\overline{ROI}_{MB}\) is the average \(ROI_{MB}\) of all the MBs in the current frame.

4.3.1.3 Boundary

Then, the \(\lambda[i]\) is further bounded by a lower and upper boundary. They eliminate too high or too low ROI values, as (4-3).

\[ \lambda[i] = \max\left\{ \lambda_L, \min\{\lambda_U, \lambda[i]\} \right\} \]  

(4-3)

where \(i\) is the MB index. \(\lambda_L\) and \(\lambda_U\) represent the lower and upper boundary of \(\lambda\) respectively. Although the boundaries are the parameters that can be tuned by users, they are set to 0.0 and 1.0 as the default values here. According to (4-2), \(\lambda[i]\) is centered on zero, where the importance weights of less important MBs are below 0, and those of more important ones are above 0. By trimming the values into the interval of \([0.0, 1.0]\) in (4-3), we only filter out the MBs with high importance values, and all the less important MBs are given to a same importance value, 0. We design the parameters like this because our scheme does not encourage the encoder to perform rate control by sacrificing the image quality of less important region too much. In other words, proposed scheme enhances the quality of the most important regions, but treats the other regions impartially.

4.3.1.4 Mean filtering

Because the \(\lambda\) value of the spatial neighbouring MBs may varies largely, if the visual quality of the MBs is allocated according to \(\lambda\), there will be a sharp difference between two neighbouring blocks. This will induce block artifacts at the edge between ROI and non-ROI blocks. To eliminate the artifacts, the \(\lambda\) value should change smoothly. As a result, a \(3 \times 3\) mean filter is applied to all the \(\lambda\) values for each MB to smoothen the initial values.

The final extracted ROI weighted values are shown in (d) of Fig. 4-4 and 4-5. The value range is from 0 to 1. The brighter area indicates larger value.
4.3.2 Key Frame Detection

There are different types of scenes in the gaming videos. For example, there are periods when the player is controlling the game, and periods when he/she is not in control (known as cut scenes); there are scenes that player is focusing on moving graphic areas on the screen, and scenes that he/she is reading still text information, etc. As a result, for different types of video sequences, different coding strategies should be applied to them.

For video coding, the parameters can be obtained and adjusted include QP, frame rate, and the type of frames, such as intra frame, inter frame or skip frame. Those parameters can be controlled for encoding different video scenes, in order to enhance the visual quality, lower its bandwidth demand and shorten latency.

The different video scenes are started and ended with certain frames. Those frames are the key frames. Just like ROI detection, the detection of key frames and the determination of according parameters can be either given by game program, or obtained by other algorithms.

4.3.2.1 Intra frame position allocation

![A and B frames with sudden content change](image)

Figure 4-6. Examples of a sudden content change. Two consecutive frames are shown but there is a big change in the content.

Generally, the gamer is very sensitive to the low quality when the content of a scene changes. For example, when the gamer takes out a map in a FPS game, he/she wants the map to appear immediately, without seconds of blurry interval. As Fig. 4-6 shows, the two consecutive frames are largely different, because the player wants to see the score board. It is very commonly seen in the FPS games. Unfortunately, the blurry happens sometimes in today's
cloud gaming services, and the gaming experience is affected. If a predicted frame (P-frame) is used in scene-change frame, the quality will be very low, because the residual of two frames is not capable of describing very large scene changing. It may take several frames to get back to normal quality.

Proposed scheme is to allocate an intra frame (I-frame) to every scene-change position. With this improvement, gamers will see the clear new scene quicker.

4.3.2.2 Control of frame rate

For frame rate, the encoder can allocate lower frame rates to the cut scenes than to the regular game process, because when players are not controlling the game, they do not care much about the latency.

4.4 Rate Control for H.264/AVC Based Video

JVT-G012 Li et al. (2003b) is the rate control algorithm recommended by H.264/AVC standard. In JVT-G012, the target bit for each frame is first determined according to the given bandwidth, and the target bit for each MB is determined according to its predicted mean absolute deviation (MAD). The MAD is predicted by a linear model. With the target bit rate, the last step is to determine the QP for each MB using a quadratic rate quantization (R-Q) model. The allocation of bits between MBs is based on the MAD and remaining bit rate. It does not consider the ROI and HVS.

A novel MB-level bit allocation scheme is proposed based on the basic framework of JVT-G012. The strategy is to adjust the target bit rate for each MB according its importance. It contains the following three steps.

4.4.1 Preference Adjustment

First, before encoding each frame, the ROI weighted value for each MB, $\lambda$, needs to be computed. Then, a weight for each MB is computed according to $\lambda$ based on user’s specification as (4–4).

$$w[i] = \lambda[i] \times (Bias_{ROI} - 1) + 1$$ (4–4)
where $i$ is the MB index. $Bias_{ROI}$ is a value specified by user, which is equal or greater than 1, indicating the bias towards ROI compared with non-ROI. The value represents the how many times the target bits we want to allocate to a MB with highest importance weight, compared to a non-important MB. In our scheme, $Bias_{ROI}$ is set to 3.0. That means it will allocate 3 times of the bits for the highly important MBs than the bits allocated to non-important MBs. If $Bias_{ROI} = 1$, there is no preference towards ROI, and the algorithm is the same as JVT-G012.

4.4.2 Relative Importance Ratio

The relative ratio of current MB’s $w$ to the average $w$ of remaining MBs is computed as (4–5).

$$w_{Ratio}[i] = \frac{w[i] \times (N_{MB} - i + 1)}{\sum_{k=i}^{N} w[k]}$$

where $i$ is the MB index. $N_{MB}$ is the total number of MBs in one frame.

4.4.3 Frame-level Bit Allocation

The objective of this stage is to determine the target bit for each P-frame before it is encoded. The target bit is determined by considering both the relative complexity in (4–6) and the buffer occupancy in (4–7).

$$T_{complexity}[j] = \frac{B_r[j]}{N_r}$$

where $B_r[j]$ is the number of remaining bits after $(j - 1)^{th}$ frames in current GOP, and $N_r$ is the number of remaining P-frames in current GOP.

$$T_{buffer}[j] = \frac{R_c}{f} + \gamma \times \left( T_{bl}[j] - C_{bl}[j] \right)$$

where $R_c$ is the bitrate of current channel, and $f$ is the frame rate. $T_{bl}$ and $C_{bl}$ are the target buffer level and current buffer level, whose definitions are given by JVT-G012. $\gamma$ is a
constant of 0.5. Then, the target bit for $j^{th}$ frame in a GOP is a combination of $T_{complexity}$ and $T_{buffer}$ as (4–8).

$$T_{frame}[j] = \beta \times T_{complexity}[j] + (1 - \beta) \times T_{buffer}[j] \quad (4-8)$$

where $\beta$ is a constant of 0.9 in our proposed scheme. Further adjustment is made to conform with the hypothetical reference decoder (HRD) requirement, which is similar as that in Li et al. (2003b).

$$T_{frame}[j] = \min\left\{U[j], \max\{T_{frame}[j], Z[j]\}\right\} \quad (4-9)$$

where $Z[j]$ and $U[j]$ are computed by:

$$Z[j] = \begin{cases} \frac{R_c}{f} & j = 1 \\ Z[j-1] + \frac{R_c}{f} - b[j-1] & \text{other} \end{cases} \quad (4-10)$$

$$U[j] = \begin{cases} 0.8 \times \frac{R_c}{f} \times \varpi & j = 1 \\ U[j-1] + \left(\frac{R_c}{f} - b[j-1]\right) \times \varpi & \text{other} \end{cases} \quad (4-11)$$

where $b[j-1]$ is the actual generated bits in the $(j-1)^{th}$ frame. $\varpi$ is a constant with typical value of 0.9.

### 4.4.4 MB-level Bit Allocation

After getting the frame-level target bit $T_{frame}$ for each frame, they are unequally allocated to each MB in the frame, according to its importance weight ratio $w_{Ratio}$ computed in (4–5). The target bit of one MB’s texture bits is computed as (4–12).

$$T_{text}[i] = \frac{T_{frame}}{N_{MB}} \times w_{Ratio}[i] \quad (4-12)$$

where $i$ is the MB index. $N_{MB}$ is the total number of MBs in one frame. After getting the target bit $T_{text}$, the QP of each MB can be computed by using the R-Q model proposed by JVT-G012.
To sum up, from (4–3) we know that \( \lambda[i] \) is bounded by 0 and 1, so, according to (4–4), \( w[i] \) is bounded by 1 and 3. In (4–5), the relative weight ratio \( (w_{\text{Ratio}}[i]) \) is normalized by the sum of total weight \( w[i] \), and in (4–12), the target texture bits is adjusted based on the equally allocated target texture bits within a frame. As a result, the adjusted target bits between MBs will not change suddenly, so the image quality smoothness is implicitly achieved within a frame.

It is worth mentioning that, the reason our scheme is based on H.264/AVC is that, according to our research, all the mainstream clouding gaming providers are based on H.264/AVC. To the best of our knowledge, until now, there is no cloud gaming provider that uses High Efficiency Video Coding (HEVC) or other as video compression standard. As a result, we did not design RC scheme for HEVC in our work.

However, our proposed scheme is a relatively universal framework of applying rate control to cloud gaming: the core idea behind it is obtaining ROI information by interaction with game program and assigning the weighted target bits accordingly. Proposed scheme is not designed for a specific standard, and it can be easily extended to HEVC. In the standard of HEVC, the first rate control scheme adopts the algorithm proposed in JCTVC-H0213 Choi et al. (2012), which is a transplant of H.264/AVC’s JVT-G012. It proposed a pixel-wise unified rate-quantization (URQ) model working on the multi-level regardless of block sizes, which is a low computational complexity algorithm to handle the chicken-and-egg dilemma of rate control problem. Just like JVT-G012 in H.264/AVC, JCTVC-H0213 also tends to divide the bits equally for all MBs. As a result, the main bit allocation scheme described in (4–12) could also be used in HEVC, which makes adjustments after texture bits for current MB is computed by JCTVC-H0213.

4.5 QoE-based Evaluation and Simulation Results

After proposing the strategy, the evaluation and comparison of this strategy need to be conducted. Many of the current cloud gaming papers are about QoE evaluation, which give us the tools to assess the performance of proposed scheme both objectively and subjectively.
4.5.1 QoE-based Quality Evaluation

Because the purpose of ROI-based rate control algorithm is to enhance the ROI while sacrificing the non-ROI regions when the overall coding and transmission resources are limited, traditional quality assessment methods like Peak signal-to-noise ratio (PSNR) or Mean squared error (MSE) are not suitable for evaluating the performance of ROI-based. The definitions are in (4–13) and (4–14).

\[
PSNR = 10 \cdot \log_{10} \left( \frac{\text{MAX}_I^2}{\text{MSE}} \right) \\
= 20 \cdot \log_{10} \left( \frac{\text{MAX}_I}{\sqrt{\text{MSE}}} \right) \\
= 20 \cdot \log_{10} (\text{MAX}_I) - 10 \cdot \log_{10} (\text{MSE}) \quad (4\text{–}13)
\]

where \(\text{MAX}_I\) is the maximum possible pixel value of the image. When the pixels are represented using 8 bits per sample, this is 255. \(\text{MSE}\) represents the mean squared error as (4–14).

\[
\text{MSE} = \frac{1}{mn} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} [I(i, j) - K(i, j)]^2 \quad (4\text{–}14)
\]

where \(I\) and \(K\) represent a noise-free \(m \times n\) monochrome image and its noisy approximation respectively.

PSNR and MSE are all to measure the quality of overall image, without considering the difference of area importance. In order to consider the ROI and QoE into the evaluation, the Integrated Mean Squared Error (IMSE) is introduced as (4–15).

\[
\text{IMSE} = \frac{1}{mn} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} w(i, j) \times [I(i, j) - K(i, j)]^2 \quad (4\text{–}15)
\]

where \(w(i, j)\) is the weight of the pixel \((i, j)\). The definition of \(w\) is the same as (4–4). The evaluation method makes the pixels in ROI have more weight than other pixels, so the result can simulate the subjective visual experience of human.
4.5.2 Simulation Results

The proposed rate control method is incorporated into the H.264/AVC reference software JM12.2 codec in order to evaluate its performance. The simulation was conducted with four 4CIF resolution (704 × 576 pixels) test sequences, named as “RAC 01” (a 900-frame racing game footage.), “RAC 02” (a 450-frame racing game footage.), “FPS 01” (a 900-frame first-person shooter game footage.), and “RPG 01” (a 900-frame role-playing game footage.). In addition, in order to test the performance of handling scene change, “FPS 01” is recorded with a lot of scene-change points, including the scenes of death-cam and viewing scoreboard. They are recognized as “key frames” in proposed method.

In the simulation, each sequence is coded at 30 fps by a single GOP with structure of IPPP. The reference frame is set to 1 and the search window is set to 16 pixels in baseline profile. “RAC 01” and “FPS 01” are encoded with 300 kbps, and “RAC 02” and “RPG 01” are encoded with 800 kbps to test the performance under different channel bandwidth. Rate control algorithms JVT-G012 Li et al. (2003b), Liu et al. (2008) and Shen et al. (2013) were selected as the references for comparison with proposed method.

The settings of the different methods are generally the same, except for that we test Liu et al. (2008) with two different ROIs: one set of results are generated by the encoder using the ROI proposed by Liu et al. (2008); and the other set of results are generated by Liu et al. (2008)’s RC algorithm, but using the same ROI as in our scheme. Because the way of extracting ROI and key frame in gaming video is one of the contributions of our work, we want to show the image quality effects brought by the choice of ROI in this set of comparisons. The results of Shen et al. (2013) are all generated by its own RC algorithm, but using the same ROI as our proposed scheme, for the fairer comparison.

Among all the test sequences, the 47th frame of “RAC 02” sequence is presented in Fig. 4-7, it is compressed at 800 kbps, where (a) is the original frame; (b) is its ROI, (c) to (e) are the QP values of JVT-G012, Liu et al. (2008) and proposed method, respectively, where brighter block means smaller QP (higher quality) is assigned to this MB; (f) to (h) are the MB
Table 4-2. Results comparisons between JVT-G012 Li et al. (2003b), Liu et al. (2008) (with and without our proposed ROI), Shen et al. (2013) with our proposed ROI and proposed algorithm for 4CIF (704 × 576 pixels) video sequences.

<table>
<thead>
<tr>
<th>Sequence Name</th>
<th>Frame No.</th>
<th>Bitrate (kbps)</th>
<th>Method</th>
<th>MSE in ROI</th>
<th>IMSE</th>
<th>Gain</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAC 01</td>
<td>900</td>
<td>300</td>
<td>JVT-G012</td>
<td>145.23</td>
<td>144.89</td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Liu et al. (2008) w/o our ROI</td>
<td>150.53</td>
<td>149.40</td>
<td>-4.50</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Liu et al. (2008) w/ our ROI</td>
<td>143.26</td>
<td>144.22</td>
<td>0.67</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Shen et al. (2013) w/ our ROI</td>
<td>143.02</td>
<td>143.85</td>
<td>1.04</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Proposed</td>
<td>142.93</td>
<td>143.71</td>
<td>1.18</td>
</tr>
<tr>
<td>FPS 01</td>
<td>900</td>
<td>300</td>
<td>JVT-G012</td>
<td>119.87</td>
<td>119.45</td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Liu et al. (2008) w/o our ROI</td>
<td>122.26</td>
<td>121.54</td>
<td>-2.09</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Liu et al. (2008) w/ our ROI</td>
<td>118.52</td>
<td>119.02</td>
<td>-0.42</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Shen et al. (2013) w/ our ROI</td>
<td>118.21</td>
<td>118.82</td>
<td>0.62</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Proposed</td>
<td>118.27</td>
<td>118.85</td>
<td>0.60</td>
</tr>
<tr>
<td>RAC 02</td>
<td>450</td>
<td>800</td>
<td>JVT-G012</td>
<td>83.98</td>
<td>82.72</td>
<td>-6.68</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Liu et al. (2008) w/o our ROI</td>
<td>83.98</td>
<td>82.72</td>
<td>-6.68</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Liu et al. (2008) w/ our ROI</td>
<td>73.89</td>
<td>74.54</td>
<td>1.50</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Shen et al. (2013) w/ our ROI</td>
<td>74.01</td>
<td>74.72</td>
<td>1.32</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Proposed</td>
<td>73.53</td>
<td>74.07</td>
<td>1.97</td>
</tr>
<tr>
<td>RPG 01</td>
<td>900</td>
<td>800</td>
<td>JVT-G012</td>
<td>10.23</td>
<td>9.41</td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Liu et al. (2008) w/o our ROI</td>
<td>10.11</td>
<td>9.72</td>
<td>-0.30</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Liu et al. (2008) w/ our ROI</td>
<td>9.02</td>
<td>9.22</td>
<td>0.20</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Shen et al. (2013) w/ our ROI</td>
<td>9.10</td>
<td>9.18</td>
<td>0.23</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Proposed</td>
<td>9.01</td>
<td>9.22</td>
<td>0.20</td>
</tr>
</tbody>
</table>

types of JVT-G012, Liu et al. (2008) and proposed method, respectively, where yellow dots indicate skip MB, blue dots indicate inter MB, and red dots indicate intra MB; (i) and (j) are the encoded frames and 3 amplified blocks using JVT-G012 and proposed method. From Fig. 4-7, it can be observed that proposed method outperforms JVT-G012 on the details of ROI parts. The block types of the three methods are about the same, but the QPs are different. The QP of proposed method fits the ROI better.

Fig. 4-8 shows the 429th frame from the sequence “FPS 01”, compressed at 300 kbps. It is a key frame, so proposed method assigned an intra frame here in order to display the clear scoreboard faster. (a) is the frame encoded by JVT-G012, and (b) is the frame encoded by proposed method. It can be observed that proposed result is clearer than JVT-G012’s.
More detailed numerical experimental results are reported in Table 4-2, where "MSE in ROI" denotes the MSE in ROI regions (where $w[i]$ of an MB is larger than 1). "IMSE" denotes the average IMSE for each sequence of all its frames. "Gain" denotes the reduction of IMSE compared to JVT-G012’s result. It can be observed from this table that, for all
the sequences, the method of Liu et al. (2008) with its own ROI fails to gain the quality in the experiments, which reflects its limitation. Since Liu et al. (2008) is designed for the conversational and head-and-shoulder types of video sequences, the ROI detection methods are mostly implemented as face detection and motion detection, which will not work in our test sequences. The results show that in most cases, the proposed scheme’s IMSE outperforms the other schemes. The average IMSE gain over JVT-G012 is 0.99. For some cases, however, IMSE of proposed scheme is a little bit higher than Shen et al. (2013). Nevertheless, we believe our results are still acceptable, since the computational complexity of our scheme is significantly lower than Shen et al. (2013), which will be shown in the next section.

It can be also observed that, for JVT-G012 and Liu et al. (2008) without our ROI, the MSE in the ROI region is higher than the total IMSE; while in the other three cases, the MSE in the ROI region is lower than the total IMSE. That means the image quality in the ROI are enhanced by the latter three schemes, and the image quality in non-ROI regions are sacrificed.

4.5.3 Computational Complexity Comparisons

From our perspective, the computational complexity of the RC algorithm is a very important aspect of our scheme. Because the real-time must be guaranteed in transmission of cloud gaming video, the round trip delay between user’s operation and the corresponding response on the screen must be minimized. So, this application is very sensitive to the
computational complexity. As a result, one of the main objectives of our work is designing a rate control scheme with a low-complexity algorithm.

We compare the computational complexities between proposed scheme and Shen et al. (2013), by showing the average coding time for each frame. The testing sequences are the same as the ones in Table 4-2. The comparison results of the average coding time per frame are shown in Table 4-3, where $T_{JVT-G012}$ represents the benchmark coding time taken by JVT-G012, and $T_{processed}$ represents the coding time of the encoder adopting the ROI-RC schemes. The “Percentage” column represents the change of coding time in percentage. From the table we can learn that the average coding time per frame of proposed RC scheme only increases about 1%, while the average coding time of Shen et al. (2013) increases about 3%. The reason is that Shen et al. (2013) needs to analyze the motion property and the texture characteristic of the underlying video to determine the visual attention regions, which is more complex than proposed scheme, thus the coding time delay may increase.

### 4.6 Summary of Source Coding Strategy

In this chapter, some strategies for rate control of cloud gaming video are presented. Firstly, the discussions about the characteristics of game players’ HVS and the possible ways to make use of them are conducted. These discussions can be extended into a broader range of applications, including all interactive VOD systems. And a new bit allocation scheme is proposed on MB layer based on ROI. In addition, an objective QoE based quality evaluation
method is also proposed. Experimental results show that the proposed method outperforms other rate control algorithms for cloud gaming video.
CHAPTER 5
CONCLUSIONS

This dissertation aims to solve three important problems faced by multimedia communication over networks. The first one is to deal with the fluctuation of bit rate generated by video codec. The second one is to deal with the lost data that dropped by network. The third one is to make use of the broadcasting nature of wireless channels. Accordingly, I provided solutions to those problems from three perspectives: channel coding, network coding and source coding.

Chapter 2 proposed a novel channel coding strategy for transmitting videos over the lossy networks: Delay-Aware Fountain codes. They deeply integrate channel coding and video coding. This is the first work to exploit the fluctuation of bit rate in video data at the level of channel coding, and to incorporate it towards the optimal design of video streaming-oriented fountain codes.

Based on this idea, we developed three coding strategies: DAF, MPC-based DAF and UEP-based DAF. DAF is the fully optimized version. It achieves the highest performance, but suffers from high computational complexity. DAF-M is based on MPC, and its computational complexity is orders of magnitude lower than DAF. MPC-O is the online variant of DAF-M that can be used in live video streaming applications. UEP-based DAF uses a user-specified frame-level importance profile to perform UEP, and the profile does not rely on any specific video coding standard. Because it utilizes the packet header of DAF, the proposed scheme does not need additional coordination between encoder and decoder. The simulation results show that the decoding ratios of our schemes are 15% to 100% higher than the state-of-the-art delay-aware schemes in a variety of settings. The PSNR of the videos transmitted by proposed schemes are also the highest among existing algorithms. This work hopes to be a first step toward further understanding this important coding opportunity.

Chapter 3 is concerned with the network coding solution to the problem of information spreading over lossy communication channels. To address this problem, a joint FoUntain coding and Network coding (FUN) approach has been proposed. The novelty of our FUN
approach lies in combining the best features of fountain coding, intra-session network coding, and cross-next-hop network coding. This chapter also proposed MIMO FUN, a wireless meshed networks protocol designed to optimize throughput of all-to-all transmission scenarios. MIMO FUN does inter-session RLNC coding on the packets from different flows at intermediate relay nodes. It reduces the number of transmissions by exploiting the broadcast nature of wireless channels, and enhances the reliability by allowing neighbors to overhear and store coded packets. The designed MIMO FUN protocol improves the coding efficiency by probing the received dof status through per-hop ACK, and maximizing the information contained in each transmission of coded packet, so that the dofs’ of all its neighbors can benefit from it. Our numerical and simulation results show that MIMO FUN outperformed all other considered schemes for a wide range of conditions.

In Chapter 4, some strategies for rate control of cloud gaming video are presented. Firstly, the discussions about the characteristics of game players’ HVS and the possible ways to make use of them are conducted. These discussions can be extended into a broader range of applications, including all interactive VOD systems. And a new bit allocation scheme is proposed on MB layer based on ROI. In addition, an objective QoE based quality evaluation method is also proposed. Experimental results show that the proposed method outperforms other rate control algorithms for cloud gaming video. We propose a prototype of the RC-aware cloud gaming system. The system can produce real-time ROI information for video encoder to perform proposed rate control scheme.
REFERENCES


URL http://gpac.wp.mines-telecom.fr/mp4box/


URL http://www.rfc-editor.org/rfc/rfc6330.txt


BIOGRAPHICAL SKETCH

Kairan Sun received his B.S. degree in computer science from Fudan University, Shanghai, China in 2012. He received his M.S. and Ph.D. degrees in electrical and computer engineering from University of Florida, Gainesville, FL, USA in 2016. He is a student member of IEEE SPS.

He was a research assistant in Multimedia Communications and Networking Lab supervised by Prof. Dapeng Oliver Wu. He joined Facebook Inc. as research scientist in 2017. His research interests include coding theory, digital image and video processing and machine learning.