RTP compatible: Two Models of Video Streaming over VANETs

by

Zhifei Fang

Thesis submitted to the
Faculty of Graduate and Postdoctoral Studies
In partial fulfillment of the requirements
For the M.A.Sc. degree in
Computer science

School of Electrical Engineering and Computer Science
Faculty of Engineering
University of Ottawa

Abstract

Because Vehicular Ad Hoc Networks (VANETs) often have a high packet loss rate, the formerly used protocol for video streaming, Real-time Transport Protocol (RTP), is no longer suitable for this specific environment. Previous conducted research has offered many new protocols to solve this problem; however, most of them cannot make full use of the existing Internet video streaming resources like RTP servers.

Our work proposes two models to solve this compatibility issue. The first model is called the converter model. Based on this model, we first modify RTP using Erasure Coding (EC) technique in order to adapt it to the high packet loss rate of VANETs. This newly developed protocol is called EC-RTP. And, we then developed two converters. The first converter stands on the boundary between the Internet and VANETs. It receives the RTP packets which sent from Internet. And then it translates them to the EC-RTP packets. These packets are transported over the VANETs. The second converter receives these EC-RTP packets, translates them back to the RTP packets. It then sends them to the RTP player, so that the RTP player can play these packets. To make EC-RTP can carry more kinds of video streams other than RTP, we proposed a second model. The second model is called the redundancy tunnel. Based on this model, we let the protocol between the two converters carry RTP protocol as its payload. We use the same technique as we have used to modify RTP. At last, we did some experiments with Android tablets. The experiment results show our solution can use the same player to play the same video resources as RTP does. However, unlike RTP, it can reduce packet loss rate.

Acknowledgements

I would like to thank everyone who made this possible.

My deepest gratitude goes first and foremost to Professor Azzedine Boukerche, my supervisor, for his constant encouragement and guidance. He has walked me through all the stages of the writing of this thesis. Without his consistent and illuminating instruction, his energy, and of course the funding he gives to me, this thesis could not have reached its present form.

Secondly, I would like to express my heartfelt gratitude to Dr.Cristiano Rezende, who led me into the world of video streaming. I would like to also express my gratitude to Dr.Abdelhamid Mammeri, who helped me to finish my experiments and edit this thesis.

Thirdly,I feel grateful to my lab colleagues. We often argued about topics during lunch. The more the truth is debated, the clearer it becomes.

Last my thanks will go to my beloved family for their loving consideration and great confidence they have had in me all through these years. I also owe my sincere gratitude to my friends and my fellow classmates who gave me their help and time; they listened to me and helping me work out my problems during the difficult course of this thesis.

Dedication

This is dedicated to the one I love.

My girlfriend Jingwen Feng, she made coffee for me when I was tired.

Table of Contents

List of Tables **ix**

List of Figures **x**

1 Introduction **1**

2 Related work **3**

2.1 Redundancy 4

2.1.1 Forward Error Correction (FEC) 5

2.1.2 Erasure Coding 5

2.1.3 Network Coding 7

2.1.4 Selection of different coding technologies 8

2.2 Interleaving 8

2.3 Key frame retransmission 10

2.4 Dynamic Rate Adjustment 11

2.5 Error concealment 11

2.6 Lower layer technology 12

2.7 The Next Generation codec 13

3 Problem formulation **14**

3.1 VANETs basic Topology 14

3.2 Several Scenarios for video streaming 15

3.2.1 The user in a car plays the video from the Internet 15

3.2.2 The user on the internet plays a real-time video captured from the vehicle's camera 15

3.2.3 Car to Car video sharing 17

3.3 Packets loss 17

3.4 Models 18

3.4.1 Internet boundary 19

3.4.2 VANETs boundary 19

3.4.3 Converter model 19

3.4.4 Generic Tunnel 21

3.4.5 Redundancy Tunnel model 21

3.4.6 Models Comparison 23

4 Protocols **24**

4.1 RTP 24

4.1.1 Overview 24

4.1.2 RTP Payload Format for H.264 Video 25

4.2 REDUNDANCY 27

4.2.1 Erasure Coding Model 27

4.3 EC-RTP 28

4.3.1 Prototype 28

4.3.2 Header design 30
4.3.3 EC-RTP server 34
4.3.4 Performance influenced by redundancy packets 34
4.4 EC-RTP to RTP Converter 34
4.4.1 Design 34
4.5 Generic Routing Encapsulation (GRE) 35
4.6 EC Generic Routing Encapsulation (EC-GRE) 36

5 Implementation **37**

5.1 Overview 37
5.2 Topology 39
5.3 RTSP server 40
5.3.1 Why RTSP is used 40
5.3.2 RTSP protocol 40
5.3.3 MP4 file 46
5.3.4 SDP protocol 48
5.3.5 RTCP protocol 51
5.3.6 Architecture of RTSP server 51
5.4 RTSP player 53
5.5 Converter 53
5.5.1 Architecture of Converter 53
5.5.2 Encoding 54
5.5.3 An EC-RTP/RTP packets sort algorithm 55
5.5.4 Redundancy rate adjustment 61
5.6 Simulator 63

6 Experiment **64**

6.0.1 Simulator Test 64

6.0.2 Running Car Test 64

6.0.3 Delay Test 67

6.0.4 CPU consumption Test 68

6.0.5 Video quality Test 68

7 Conclusion **73**

References **74**

List of Tables

2.1 Methods of Packet loss compensation 4

3.1 Advantages and disadvantages of different models 23

5.1 SDP session description (* is optional) 50

5.2 5 types of RTCP packet 52

5.3 Advantages and disadvantages of using bitmap sorting elements 56

6.1 A series of testing maximum Wi-Fi communication distance results 66

List of Figures

2.1 An example of FEC 5

2.2 An example of EC 6

2.3 An example of NC 8

2.4 An example of interleaving 9

2.5 An example of frame retransmission 10

2.6 An example of frame Interpolation 12

3.1 Basic VANETs topology[36] 15

3.2 The user in a car playing the video from the Internet 16

3.3 A user on the internet playing a real-time video captured from the vehicle's camera 16

3.4 A car to car video sharing scenario 17

3.5 Internet boundary 19

3.6 VANETs boundary 20

3.7 The topology of converter model 20

3.8 The topology of our converter model solution 21

3.9 Generic Tunnel working principle 22

3.10 Redundancy tunnel working principle 23

4.1 RTP header 25
4.2 H.264 video map RTP packets 26
4.3 Our erasure coding model 28
4.4 The strategy of generating redundancy packets 29
4.5 An example of Index field in different modes 31
4.6 Packets will not have same length 32
4.7 Make packets to have same size 33
4.8 EC-RTP header 33
4.9 EC-RTP to RTP Converter 35
4.10 GRE header 35

5.1 Google's Nexus 10 38
5.2 Topology 39
5.3 MP4 file structure 47
5.4 AVCDecoderConfigurationRecord structure 49
5.5 A SDP example 51
5.6 Architecture of RTSP server 53
5.7 Architecture of Converter 54
5.8 The state of the bitmap after initialization 56
5.9 The sate of the bitmap after handling the first element 57
5.10 The final state of the bitmap after traversing all elements 57
5.11 Basic data structure: one bitmap and one array 58
5.12 Advanced data structure: two bitmaps and two arrays 59
5.13 Redundancy Adjustment 62

6.1 Result of testing simulator in different packet loss rate 65

6.2 5 meter mark on the runway 66

6.3 The other Wi-Fi signal on the stadium 67

6.4 A map of the route on which ran the tests were run on 68

6.5 Results of testing a player in a car moving away from the server 69

6.6 Results of testing delay in different packet loss rates 70

6.7 Results of testing CPU consumption in different packet loss rates 70

6.8 Results for testing video quality in a 10% packet loss rate 72

Chapter 1

Introduction

Vehicular Ad Hoc Networks (VANETs) often have a high packet loss rate. The traditional Real-time Transport Protocol (RTP) has no features to handle packet loss. Actually, if there is a high packet loss rate, the receiver cannot even play the video stream. However, the RTP is widely used on the traditional IP network; there are a lot of solutions that use RTP as their transport protocol to provide video streaming services. Nowadays, many researchers offer many new protocols to solve the packet loss issue; however, most of them are lack the capacity to communicate with the RTP protocol. Therefore, we cannot deploy RTP directly on VANETs, making VANETs even more expensive.

In our work, we aim to modify the RTP so that it can be used in VANETs environment. In addition, our solution can also communicate with the RTP server at the same time. To counter packet loss, redundancy is one of the several possible strategies. According to [1], we decided to use the Ensure Coding (EC) technique to develop our new version of RTP, which will be called the Ensure Coding Real-time Transport Protocol (EC-RTP). To let the EC-RTP easily communicate with RTP, we designed a converter which can convert EC-RTP to RTP. In addition, to make our converter can support more protocols other than RTP, we designed EC Generic Routing Encapsulation (EC-GRE).

The contributions in this work are divided into mainly 4 parts. We will first conduct

a study on VANET topology and on some cases of its use. And, we will then formulate our problem based on the study. Additionally, we provide 2 models to solve this problem after formulating it. Our second contribution is to design an EC model, based on this, we designed our EC-RTP and EC-GRE. Our third contribution is to build a simulation based on the Android system. It can support EC-GRE, EC-RTP, RTP and the converter which can convert EC-RTP to RTP. Our fourth contribution is to conduct experiments based on this simulation to verify the effectiveness of the EC-RTP and EC-GRE. Compared with RTP, we verified if the redundancy improves the quality of video streaming.

Chapter 2 will show a survey on video streaming packet recovery techniques. Chapter 3 will outline the basic VANET topology, based on the topology we will find out where the packet loss happens, determine the boundaries of the VANET protocol and the Internet protocol, and propose 2 models that can easily increase VANET compatibility with Internet protocols. Chapter 4 shows the design of the converter which can convert EC-RTP to RTP. Chapter 5 shows a demonstration based on the Android system. This uses the model we talked about in Chapter 3 and protocols we talked about in Chapter 4. Chapter 6 describes some experiments. Compared with the original RTP, our experiment can verify if our EC-RTP can provide a better video quality. Our conclusions and future work are presented in Chapter 7.

Chapter 2

Related work

VANETs are very lossy networks, this is because their topology changes very frequently. Therefore, we need to find an effective way to decrease the packet loss rate. In this chapter, we will introduce some techniques people use to reduce packet loss rates. Therefore, we can adapt a way for our solution.

People initially designed the IP network to transmit files that did not involve some real-time media streaming. Since the transmitted files do not have any real-time requirements, the IP network allowed for delay and packets were lost. What is more, because VANETs are lossy networks, the high packet loss rate may cause poor video quality. Transmission Control Protocol (TCP) was designed to solve this packet loss problem; however, because TCP waits for the feedback of every packet, for real-time video streaming, TCP may cause an unacceptable delay. That is the reason why people decide to choose User Datagram Protocol (UDP) for real-time video streaming. However, UDP is not a reliable protocol. There is no guarantee of reliability or of ordering packets; they may arrive out of order, be duplicated, or not arrive at all. When the network jams, the UDP may also drop some packets. Nowadays, people have specifically performed research to solve this packet loss problem.

Packet loss compensation technology can be classified into two generic groups: com-

pensation based on the sender and compensation based on the receiver. The technology from the sender includes redundancy, interleaving, key frame retransmission and Dynamic Rate Adjustment. The technology from the receiver includes multiple error concealment methods. Besides those, there are still some other techniques for video streaming.

Sender	**Receiver**
• Redundancy	• Error concealment
• Interleaving	
• Key frame retransmission	
• Dynamic Rate Adjustment	

Table 2.1: Methods of Packet loss compensation

2.1 Redundancy

Redundancy is currently the most promising technology used to improve the quality of video streaming. The idea is that it does not only send out the main information, but also some redundancy information. When some packets are lost, this redundant information can recover these packets. The advantage to this technology is that we no longer care about what kind of media we are now transmitting. The disadvantage, which is also very obvious, is that sending out the redundancy information can increase bandwidth cost. And, the process of generating redundancy information can cause transmission delay.

2.1.1 Forward Error Correction (FEC)

IEEE standard RFC 5109 [2] provides a FEC method for protecting against packet loss over packet-switched networks. This is a method designed for RTP. However, not all the players can support this method. In addition, this method is not very flexible. It treats all packets with the same priority. And It doesn't have enough ability to handle high packets loss environments. Figure 2.1 shows an example of FEC.

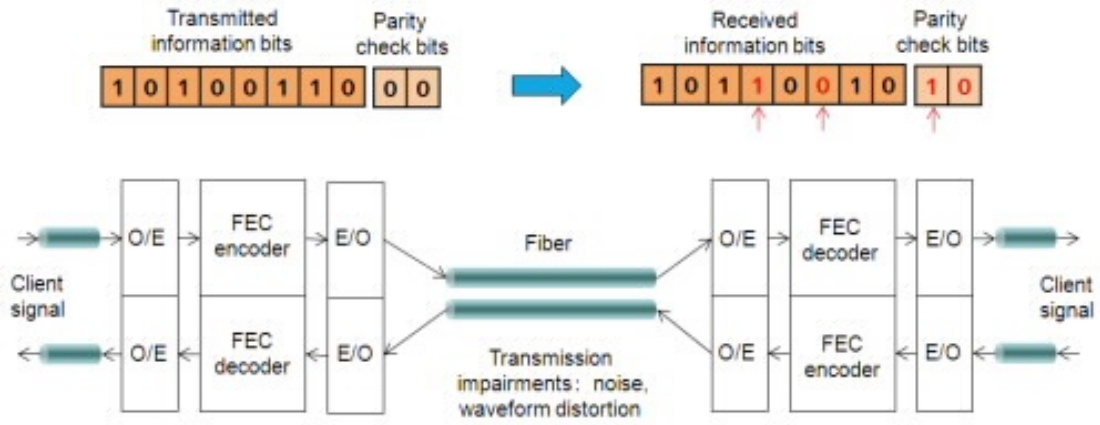

Figure 2.1: An example of FEC

Pasin et al. [3] suggested a packet level FEC and interleaving algorithm. Their experiment shows that this algorithm decrease the packet loss rate. It shows that a feedback mechanism may be a good choice for the decreasing of redundancy bandwidth consummation.

2.1.2 Erasure Coding

Erasure coding (EC) is one of the FEC method. The idea behind EC is that the loss of a partial transmitted data does not influence the ability of receivers to obtain the original content. In EC, original packets are encoded into a larger amount of packets at the sender's side; and, intermediary nodes may decode received packets for their own benefit (i.e. in

the case of video dissemination) relaying the exactly same received packets. Figure 2.2 shows an example of EC.

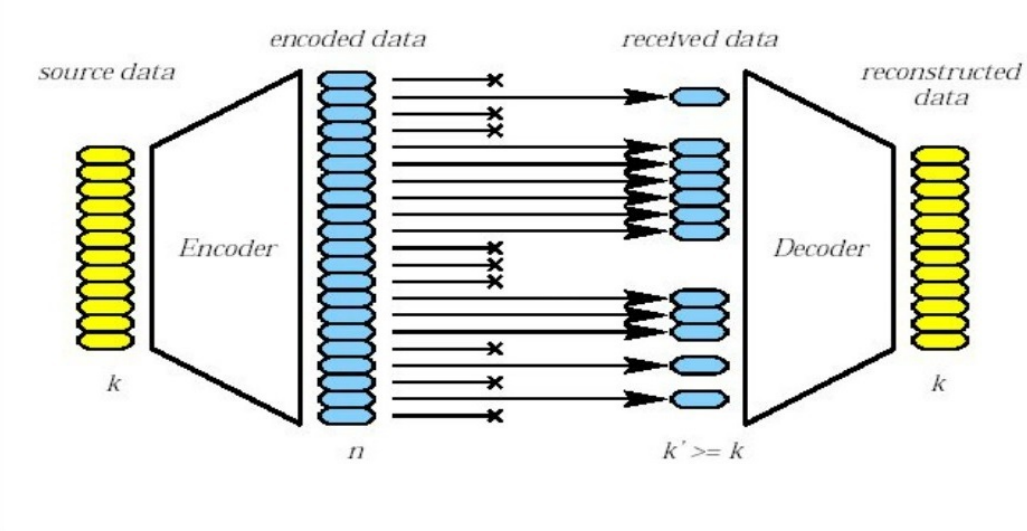

Figure 2.2: An example of EC

Although EC was first designed for storage system, it is widely used in the open source software, Cloud system [4], [5], [6], [7] and [8]. And now a lot of research shows it can eliminate problematic packet loss [9] effects. There are different kinds of EC techniques. However, few of these can using in the real-time video streaming, because the storage system does not have the real-time demands.

Among these EC techniques, XOR-based-coding is one worthy of consideration [10] [11].There are a lot of XOR based coding techniques, for example, Krohn et al. [12] outlines an EC technique for efficient multicast transfers. In [13] and [14], Luby introduced a rateless erasure coding technique that are very efficient as the data length grows.

Random Linear Coding [15] (RLC) is another technique which was created to optimize the distribution of information in multicast scenarios. It can, however, be adapted to add redundancy in a network in order to handle packet loss.

Rezende et al. [9] observed that XOR-based Coding achieves higher delivery ratios than RLC with similar amounts of additional redundancy.

Sardari et al., [16] and [17] suggest using EC to disseminate multimedia content. Their work takes into consideration the scenario in which roadside infrastructure is available to assist in the dissemination process.

2.1.3 Network Coding

Network Coding (NC) [18] in VANETs is aimed specifically at making more efficient use of the shared medium by requesting that intermediary nodes transmit newly encoded packets. Through this manner, transmissions observed by neighbouring nodes can result in the gathering of the original transmitted content, instead of the simple gathering duplicates. In NC, intermediary nodes wait for the reception of enough packets before decoding a block, and re-encoding the original content into newly encoded packets which are then transmitted further. NC techniques have to offer great diversity in terms of the newly encoded packets by intermediary nodes; and, NC techniques need to make sure that such packets are received by the relevant receiver that require the original information.

Figure 2.3 shows a classical example of NC.

The analysis by Lee et al. [19] of implementation issues of concerning NC in VANETs. Their results show that NC parameters must be carefully configured by taking resource constraints into account. This paper reveals that NC will consume a lot of resources in the network.

Lee et al. [20] proposed a NC based file swarming protocol that targets VANETs. Their simulation shows that their protocol is efficiency and effectiveness.

Ahmed and Kanhere [21] purposed a novel network coding based co-operative content distribution scheme called VANETCODE. The randomization introduced by this coding scheme makes distribution efficient.

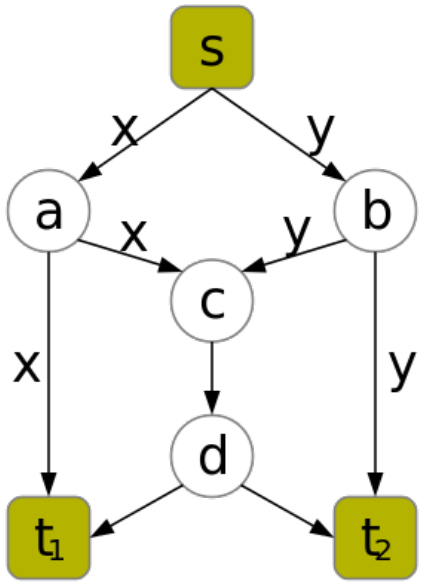

Figure 2.3: An example of NC

2.1.4 Selection of different coding technologies

Zhang et al. [22] proposed a solution that combines both EC and NC using RLC for lossy network. In their work, the source node and the intermediary nodes send out a number of encoded packets which is greater than the size of a block.They suggest that the redundant information should be used to handle packet loss while NC to improve throughput efficiency.

Rezende et al. [1] outlines how to use different redundancy methods suitable for VANETs. Based on this paper, in our work, we try to apply EC as our coding technology, because our work is focus on end-to-end scenario [23].

2.2 Interleaving

When a lot of packets are suddenly lost, the redundancy packets may not be enough to recover all the lost packets. Interleaving is one of the choices to counter this situation. The main idea behind interleaving is that original data is divided into small units. These units

are smaller than a packet. Before sending, these units are reordered so that every packet has data from different frames. At the receiver side, the unit are then put in the original order. When a packet is lost, since the units inside the packet come from different frames, the respective unit is lost rather than lost a whole frame.

Actually, interleaving cannot recover the lost packets. At the same time, the reordering process consumes a lot of time. This may cause an unacceptable delay for real-time video streaming. However, it can help to reduce the impact of packet loss. Figure 2.4 shows the process of interleaving.

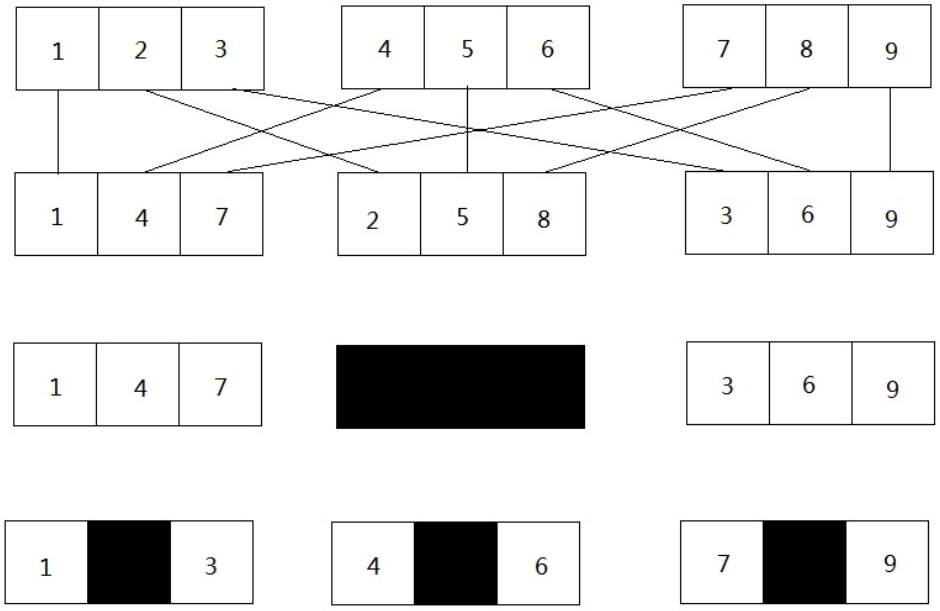

Figure 2.4: An example of interleaving

Lambert et al. [24] proposed to test the new Flexible Macroblock Ordering(FMO) in the H.264/AVC digital video compression standard. FMO is a kind of Interleaving. It splits a frame into some slices. The use of FMO alone maybe not enough, so Qadri et al. in [25] and [26], and Razzaq and Mehaoua [27], tried to combine the this method with different technology such as we have mentioned above FEC, and the one we will introduce later Error concealment.

2.3 Key frame retransmission

According to the H.264/AVC standard, a video can be divided into different types of frames. Among these frames, some play the key role. These are the key frames, and other frames only record the changes of the key frame. Therefore, the qualities of the key frames are the important factor for video quality. In other words, we need to just retransmit these key frames, instead of retransmitting all the frames. Therefore, bandwidth will be saved. However, when we meet a network jam, retransmission often fails. Figure 2.5 shows an example of frame retransmission.

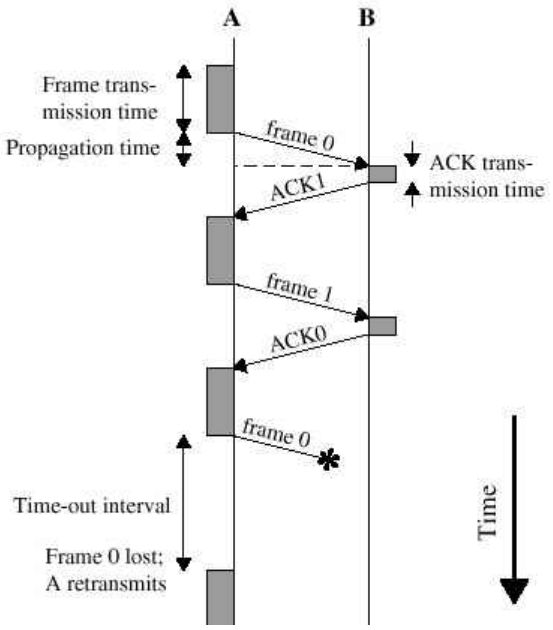

Figure 2.5: An example of frame retransmission

The IEEE standard RFC 3366[28] provides a frame retransmission technique called Automatic Repeat Request (ARQ). In the ARQ, when the side receiving data does so correctly, it must provide feedback to the side sending data. Otherwise, the side sending data stops doing this. ARQ can make the transmission reliable. However, the retransmission delay makes ARQ does not adapt to real-time video system.

2.4 Dynamic Rate Adjustment

The main idea of this technique is to change the bit rate of a video stream dynamically. This is to let the video bit rate adapt to the network bandwidth. For example, if the network is jammed, because a high bit rate video has a bigger size than a low bit rate video, we will not have enough bandwidth to send out a high bit rate video. At the same time, the receiver side often has a bad video playing experience. A proper way to avoid this bad experience is to decrease the video bit rate.

Xing and Cai [29] described a technique using dynamic rate adjustment on VANETs. And Krasic et al. [30] described a technique using priority data dropping. In this paper, the authors also showed how to express adaptation policies; and the moreover showed how to deal with priority-mapping.

2.5 Error concealment

Error concealment is a technique used by the sender side. Here are some types of classical error concealments.

1. Insertion

   The side where reception takes place reorders the received packets; this is because these packets may not in the right order. After this process, there may be some empty positions, due to packets in the position were lost during the transmission. To conceal these errors, a simple method is to insert zeros into these positions. Or, we can copy the packets that were in place before the more recently positioned lost packets to fill the now empty positions.

2. Interpolation

   For Interpolation, after the reordering process, we use the packet nearest to the lost packet to compute a packet similar to that lost packet. We then use this packet to

replace the lost packet. We may thus reduce the bad effect that caused by packets lost. Figure 2.6 shows an example of frame Interpolation. We find that 1a is generated by frame 1 and 2.

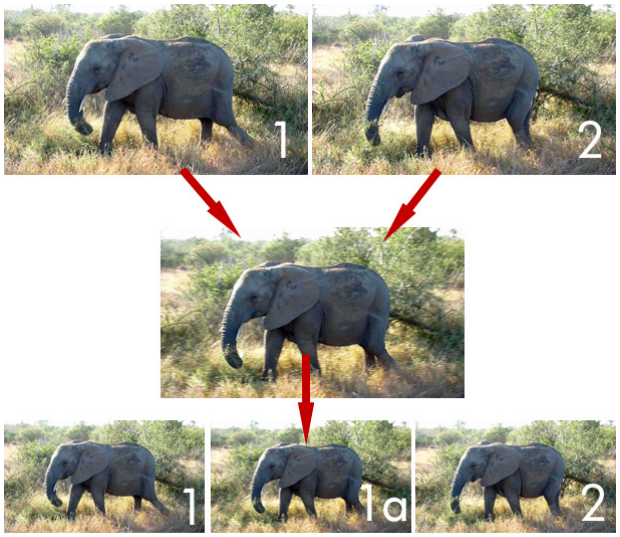

Figure 2.6: An example of frame Interpolation

The shortcoming of error concealment is that it is passive. It starts to work, only when the packets are lost. So it is often combined with other techniques. Ho et al. [31] intended to enhance the video streaming quality on the receiver end by recovering lost packets with error resilience. In addition, they chose more stable routing paths to make inter-vehicle data transmission more reliable.

2.6 Lower layer technology

Packet loss often occurs when VANETs topology changes. Assume we have 3 vehicles A, B and C. and that they are running on a road. B is in the middle of A and C, now A is leaving the communication range of B; and C is about to enter that range. B wants to send packets out, it first connects to A. However, when A leaves the communication range, the new connection between B and C has not simultaneously occurred; thus, the packets

sent out from B will be lost. Reducing the time needed to establish a new connection is one way to solve this problem.

IEEE 802.11p [32] is a new protocol designed for VANETs. Eichler [33] evaluated its performance. The results show that 802.11p shall provides a multi-channel dedicated short-range communication solution with high performance for multiple application types. However, 802.11p is too new too use, only a few devices support it for now.

Asefi et al. provided an adaptive Medium Access Control (MAC) retransmission that limits selection scheme just like IEEE 802.11p. This technique can quickly establish connections among nodes.

2.7 The Next Generation codec

The newly emerging video coding standard called High Efficiency Video Coding (HEVC), also known as H.265, will soon be released. If this new coding standard has the ability to solve the packet loss issue, we can use this new coding standard as our standard.

Piol et al. [35] evaluate this standard and the result, however, showed that even this next generation coding standard is still very sensitive to packet loss.

Chapter 3

Problem formulation

Over thie course of this chapter, we firstly study several possible video streaming scenarios to determine the compatibility issue. And, we then develop our two models to solve the compatibility issue.

3.1 VANETs basic Topology

According to [36], VANETs have essentially two basic parts: one part is the vehicle to vehicle network, the other one is the vehicle to roadside infrastructure network. VANETs enable vehicles to communicate with other vehicles which are out of sight or even out of radio transmission range. They also enable vehicles to communicate with roadside infrastructure. In addition, existing roadside infrastructure lets the VANETs connect to the Internet.

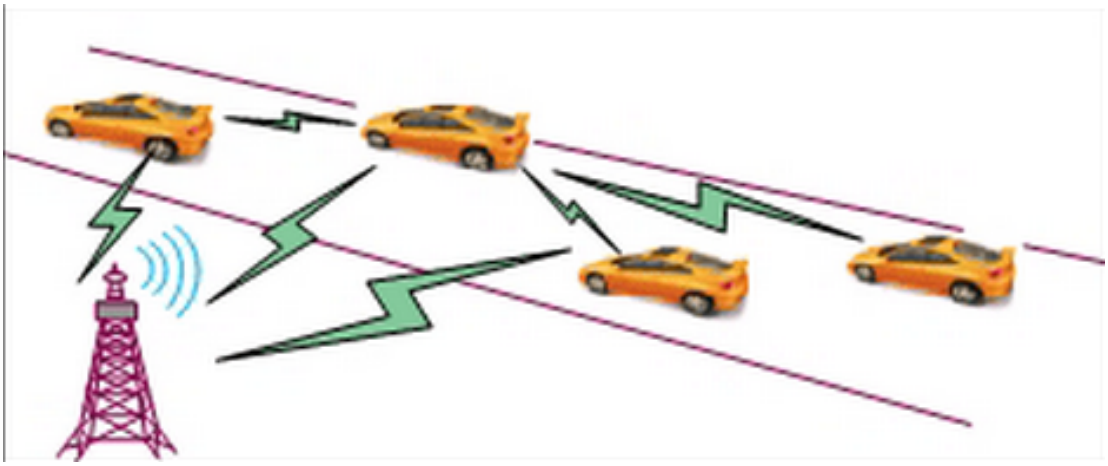

Figure 3.1: Basic VANETs topology[36]

3.2 Several Scenarios for video streaming

3.2.1 The user in a car plays the video from the Internet

In this scenario, as Figure 3.2 shows, there is a car in the VANETs; inside the car, there is a small Local Area Network (LAN), one laptop, and one cell phone connected to a vehicle-mounted wireless router. The user of the laptop wants to watch a video from the Internet; his request will be first sent to the vehicle-mounted wireless router. It will then forward the request to the VANETs, the VANETs will communicate with Internet through the roadside infrastructure. And, the Internet will retrieve the video from the video streaming server.

3.2.2 The user on the internet plays a real-time video captured from the vehicle's camera

In this scenario, as Figure 3.2 shows, the user of the PC wants to see the video captured from the camera on Car 1. The stream will flow exactly as it did in Section 3.2.1 but in the opposite direction.

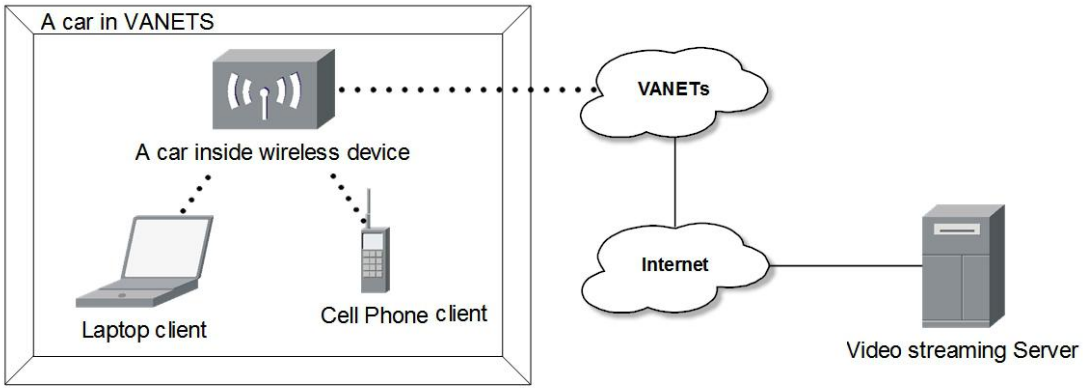

Figure 3.2: The user in a car playing the video from the Internet

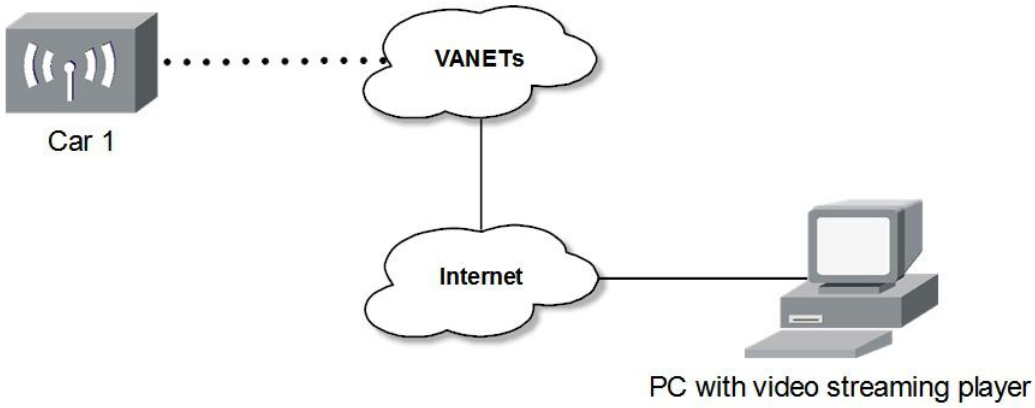

Figure 3.3: A user on the internet playing a real-time video captured from the vehicle's camera

3.2.3 Car to Car video sharing

In this scenario, as Figure 3.4 shows, cell phone 1 and cell phone 2 want to share a video with each other. Because both phones are in two different cars, these cars are connected to each other through VANETs. In each car, there is a wireless router, this router is connected to each cell phone. The router in the vehicle plays a gateway role for the VANETs.

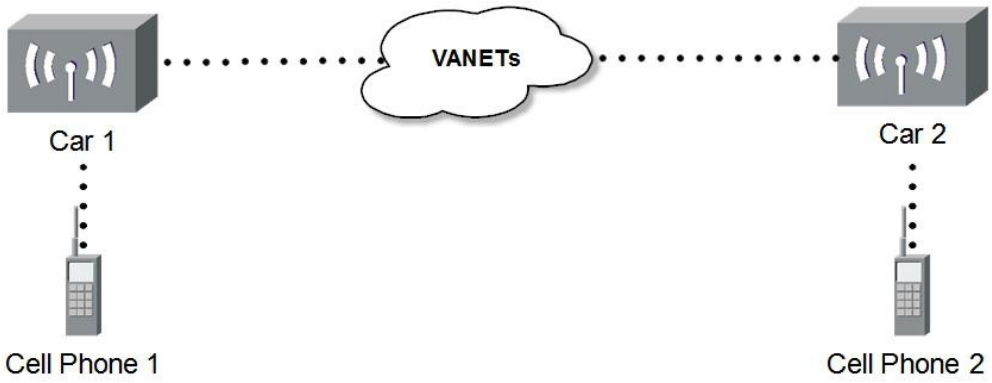

Figure 3.4: A car to car video sharing scenario

3.3 Packets loss

From the 3 scenarios described, the main area where packet loss in the topology will occur in the VANETs. When the Internet protocols want to go through VANETs, the risk of packet loss will increase dramatically. The main reason for this is that: the Ad-hoc connection is easily interfered with by the environment, while the topology in the VANETs changes frequently. For example, in Section 3.2.1 a car connects to the roadside infrastructure, and when the car runs at a high speed, it will switch to a different roadside infrastructures. This switching will cause a lot of packet loss.

3.4 Models

From the cases discussed above, we can draw the following conclusion: when Internet protocols go through the VANETs, they may meet a high packet loss rate. When this situation happens, a lot of these protocols cannot work properly. This is especially the case for real-time video streaming protocols. Therefore, the user will obtain an unsuccessful experience. At last, our problem can be describe in the text below.

Assume we have 2 different networks, one is a traditional network like Internet, the other one is a new network like VANETs. They are both packet-switched networks. There are, however, some differences among the features like packet loss rate, delay, sending rate, etc. In the traditional network, there are some protocols used widely. However, there are no similar protocols in the new network. Due to these different features, if we take these widely used protocols directly from the traditional network and implement them on the new network, the user experience may be bad. Research that considers this issue mainly focuses on designing new protocols which can achieve same goal as these widely used protocols in the new network. However, because these newly designed protocols often have a different working principle, they cannot communicate with protocols running on the traditional networks; this causes resources to be wasted.

We have two goals to achieve. The first one is to improve the user experience of video streaming by applying various techniques. At the same time, we want to keep use the same player and server as people do who used in Internet. In this work, we use the RTP player and server. According to Chapter 2, we decided to use EC as our technique to reduce packet loss rate. Since the VANETs are the networks with a high packet loss rate, we need to first determine the VANETs and Internet boundary.

3.4.1 Internet boundary

As we discussed above, when data is required the Internet, it needs to first go through the roadside infrastructure. Therefore, it is easy to determine the boundary of the Internet. Figure 3.5 shows this kind of boundary.

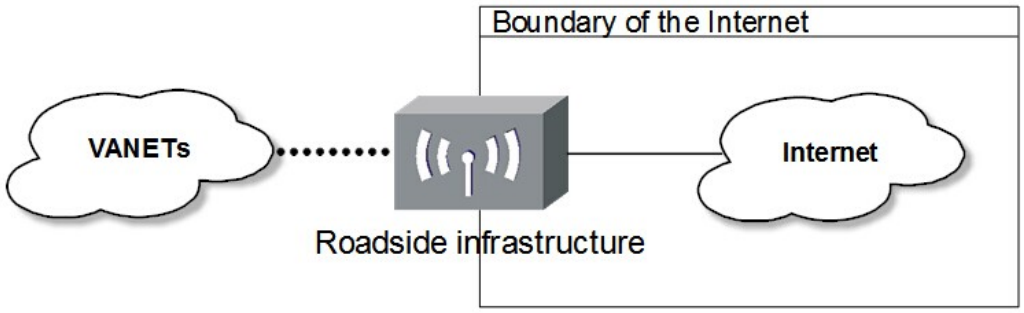

Figure 3.5: Internet boundary

3.4.2 VANETs boundary

Other than the boundary of the roadside infrastructure, inside a vehicle, the LAN is different with VANETs. The vehicle-mounted wireless router is in a state of relative rest with its clients. There is no Doppler effect. The clients do not need to roam. Thus packet loss rate is lower than it is in VANETs. Therefore, there is another boundary for VANETs. Figure 3.6 shows these two boundaries.

3.4.3 Converter model

Figure 3.7 illustrates the topology of this model. In this figure, we added a converter between the traditional network (A) and the new network (B); this converter takes charge of translating protocol adapted A into protocol adapted B. To reduce the delay caused by the translation process, and the load of the converter, the protocol adapted A and protocol

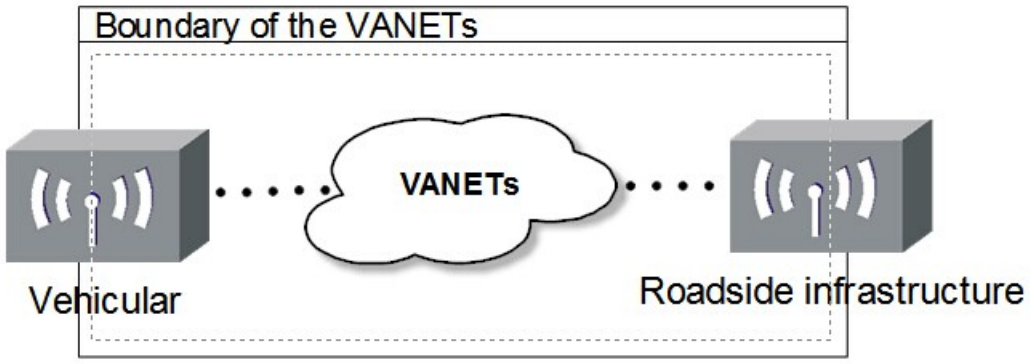

Figure 3.6: VANETs boundary

adapted B should take the form of the same one protocol but as different versions. The protocol adapted B should be one version modified from the original protocol running in A. In addition, this modification should not affect the working principle of the original protocol.

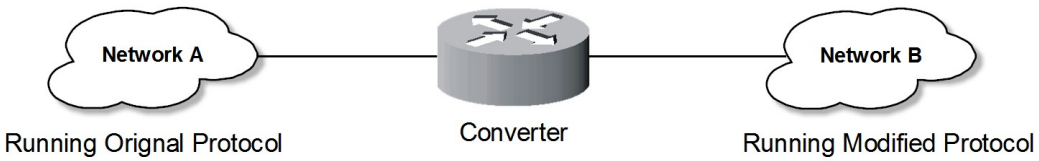

Figure 3.7: The topology of converter model

In our work, A represents the internet and B represents the VANETs. The protocol adapted A is RTP. Therefore, we need to design a modified version of RTP. We call this modified version EC-RTP. In addition, because we want to use the RTP server and player at the same time, we need to add another converter to translate EC-RTP back to RTP. Figure 3.8 shows the topology of our solution.

The benefit of a modified protocol which works on the Internet is that we can add the some additional features to the protocol. However, the process of modifying a specific

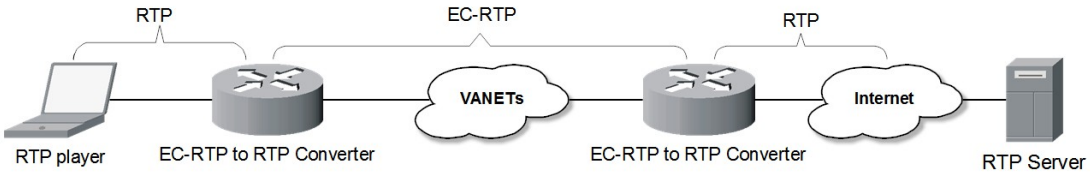

Figure 3.8: The topology of our converter model solution

protocol and developing a converter are still a lot of work. Though we achieved our goals, the cost of the VANETs may not be reduced. This solution may be more suitable for those networks that have certain requirements. Therefore, we start to started a new model. This model will not be impacted by care about what kind of server and player we are now using. It will build a tunnel between the two converters.

3.4.4 Generic Tunnel

The Generic Tunnel packet has its own header, it enables all the of a packet of given protocol to move through VANETs behind its header. It will act like a lower level protocol. Just add its own header before the one it will carried. The figure 3.9 shows us the position of the header of the Generic Tunnel. In fact, by using this tunnel, we can treat the Generic Tunnel as a new protocol. We can now use any techniques to make this tunnel have strong ability to deal with high packets loss rates.

At the same time, different protocols may have different requirements. Some real-time protocols request a low packet loss, while some request low packet delay. One simple generic tunnel may not be able to afford these specific requirements at the same time.

3.4.5 Redundancy Tunnel model

The Redundancy Tunnel is one kind of Generic Tunnel. This tunnel can help a protocol, designed for the Internet, gain the ability to resist packet loss when it needs to go through

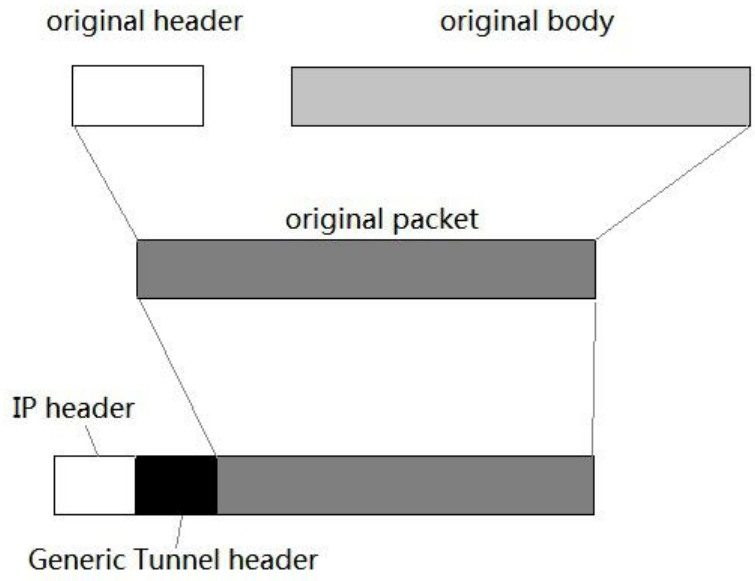

Figure 3.9: Generic Tunnel working principle

the VANETs. Figure 3.10 demonstrates how this kind of tunnel works. Assume that we need to send some packets from Network A through the VANETs to the Network B. The protocol we used was designed for the Internet, for example User Datagram Protocol (UDP). When the packet reaches the Redundancy Tunnel Gateway (RTG) A, the RTG will generate the redundancy packet; it will re-encapsulate the destination of the IP address to the RTG B and some additional information which will be needed when the other side of tunnel RTG B tries to recover the lost packets. After this, these packets will be sent through the VANETs. When the RTG B receives the packets, it will try its best to recover as many packets as it can. RTG B perform decapsulation to get the original protocol header. it will drop the duplicate packets, and resend the original protocol packets. The whole process for the protocol we use will be transparent; in other words, the protocol we use here will not be affected.

Besides, using this tunnel, we do not need the redundancy packets to travel from the very beginning to the end; these redundancy packets will only affect the load of VANETs. This will decrease the effect of speed of these redundancy packets.

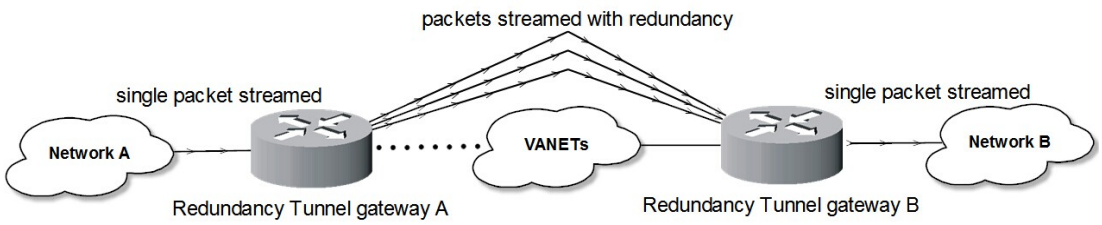

Figure 3.10: Redundancy tunnel working principle

3.4.6 Models Comparison

The Table 3.1 shown below, outlines the advantages and disadvantages of these two models.

	Advantages	**Disadvantages**
Converter model	Can incorporate different features based on the needs of different protocols	The converter cannot be reused for other protocols. There cannot be a standard converter for each protocol
Generic Tunnel	Can be Simply configured once. Can be use for many protocols	Cannot adjust some feature for the special needs of its payload protocol.

Table 3.1: Advantages and disadvantages of different models

Chapter 4

Protocols

4.1 RTP

4.1.1 Overview

RTP was developed by the Audio-Video Transport Working Group of the Internet Engineering Task Force (IETF). It was first published in 1996 as RFC 1889, and was superseded by RFC 3550 [37] in 2003. RTP provides end-to-end network transport functions suitable for applications transmitting real-time data, such as audio, video or simulation data, over multicast or unicast network services. RTP does not, however, address resource reservation. RTP is based on the UDP, thus when packet loss occurs, we cannot ask the resource to re-transfer the lost packet. This does not guarantee quality-of-service for real-time services.

RTP packets are composed of a fixed-length header and variable body length. Figure 4.1 shows the header of RTP. The header does not need to be changed in order to let the RTP communicate from end to end. The header carries a lot of useful information, such as defining the type of packets, time stamp, etc.

The body of RTP can carry different format streams with different methods. There are a lot of standards made by IEEE, to simplify matters, we will use H.264 as our payload

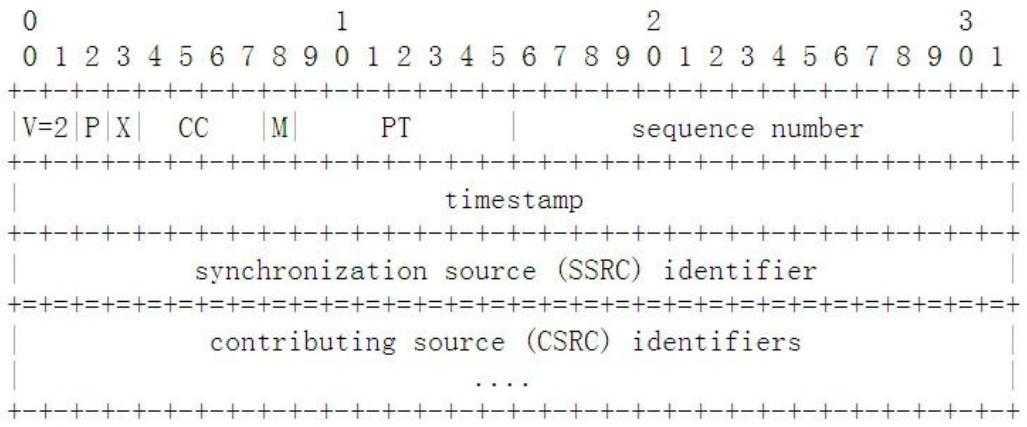

Figure 4.1: RTP header

format.

RTP itself does not have any mechanisms to control packet loss. And, according to the H.264 codec is packet loss sensitive. Thus real-time data packet loss, for a H.264 sequence of real-time video data is very detrimental.

4.1.2 RTP Payload Format for H.264 Video

H.264 is currently one of the most commonly used formats for the recording, compression and distribution of video content. The final draft on the first version of the standard was completed in May 2003.

H.264 is now widely used in a lot of systems. For example, Apple has officially adopted H.264 as the format for Quick Time [38]. It is also one of the formats chosen to be supported by both high definition DVD standards. It is also destined to become the future standard format for Blu-ray. H.264 is also used in camcorders and Blu-ray recorders.

The RTP payload format allows for packetization of one or more Network Abstraction Layer Units (NALUs). These NALUs are produced by an H.264 video encoder. The payload format has a wide applicability. This is because it supports applications from

simple low bit rate conversational usage, to Internet video streaming by way of interleaved transmission. In other words, the RTP payload format also supports high bit rate video-on-demand [39] applications.

The Figure 4.2 shows how the H.264 video is mapped into RTP packets. Because every frame in the video does not have the same size, and NALUs can contain one frame or a part of a frame, NALUs also are not the same size. Some NALUs are bigger than Max Transfer Unit(MTU). And Some NALUs is smaller than it. Because each RTP packet should be smaller than the MTU, in the case of the bigger one, we cannot use it as payload of a RTP packet directly. So, we need to use more than one packet to carry this bigger NALU. And, for the smaller ones, we can use one RTP packet to carry several NALUs.

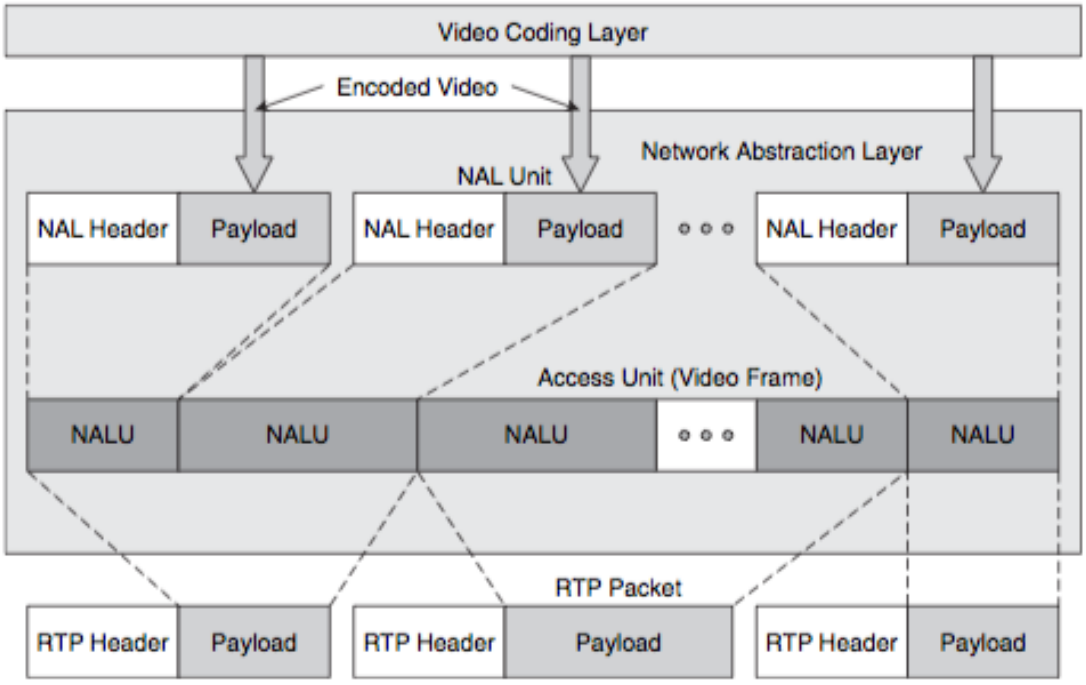

Figure 4.2: H.264 video map RTP packets

4.2 REDUNDANCY

In Chapter 2, we decide to use the EC technique as our transmitting strategy. The following section will introduce our EC model. This model is based on XOR.

4.2.1 Erasure Coding Model

Figure 4.3 shows our EC model.

At first, we divide the data being sent into blocks. These blocks have the same size. And each block will be divided into 2 parts of isometric size. We call the first part of the data A, and the second part of the data B. We call the redundancy part C.

According to the character of XOR, it is easy to find these formulas below. We can generate C by using Formula 4.1.

$$C = A \oplus B \tag{4.1}$$

$$A = C \oplus B \tag{4.2}$$

$$B = C \oplus A \tag{4.3}$$

According to Formula 4.2 and 4.3, if we have any of the two from the selection of A,B and C, we can recover the missing part easily. This model can decrease the packet loss rate. At the same time, because the formulas have a low computing complexity, this model has, therefore, a fast encoding and decoding speed.

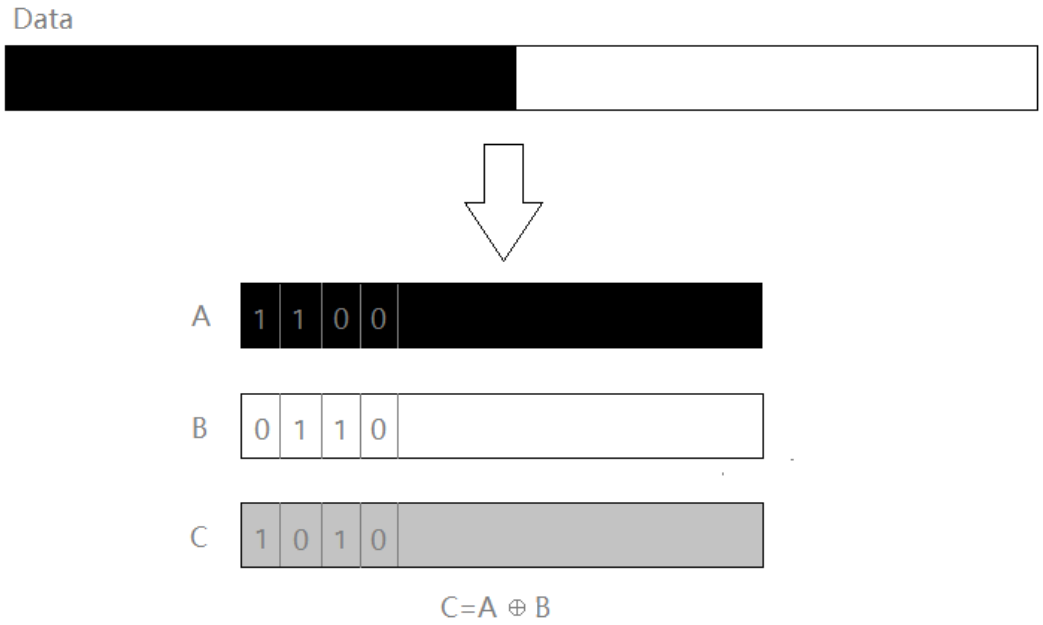

Figure 4.3: Our erasure coding model

4.3 EC-RTP

4.3.1 Prototype

First, we divide data into packets like the normal RTP does. We then group the packets as blocks; each block will have the same even number of packets. We call this number S. Now the question is how can we generate C in a proper way. We define the process of generating C packets as encoding process, and we define the process of recovering a packet as the decoding process. There are a lot of ways to generate C. We do this as Figure 4.4 shows. After applying our EC model, for one block, the first half of the block becomes A. The other half of the block becomes B. We then can have three new formulas. A_i means i^{th} packet in A. B_j means j^{th} packet in B. C_{ij} means C packet generated by A_i and B_j. All the operations will be computed bit by bit.

$$C_{ij} = A_i \oplus B_j \tag{4.4}$$

$$A_i = C_{ij} \oplus B_j \tag{4.5}$$

$$B_j = C_{ij} \oplus A_i \tag{4.6}$$

Now we can generate C packets by using Formula 4.4. A greater amount of C means low packet loss rate. However, this means more bandwidth consumption at the same time. So, we let the users define how many C packets they want. For example, the first $S/2$ C are generated by $A_1 \oplus B_1 and A_2 \oplus B_2 ... A_{S/2} \oplus B_{S/2}$; the second $S/2$ C are generated by $A_1 \oplus B_2 and A_2 \oplus B_3 ... A_{S/2} \oplus B_1$; the third $S/2$ C are generated by $A_1 \oplus B_3$, $A_2 \oplus B_4 ... A_{S/2-1} \oplus B_1 and A_{S/2} \oplus B_2$.

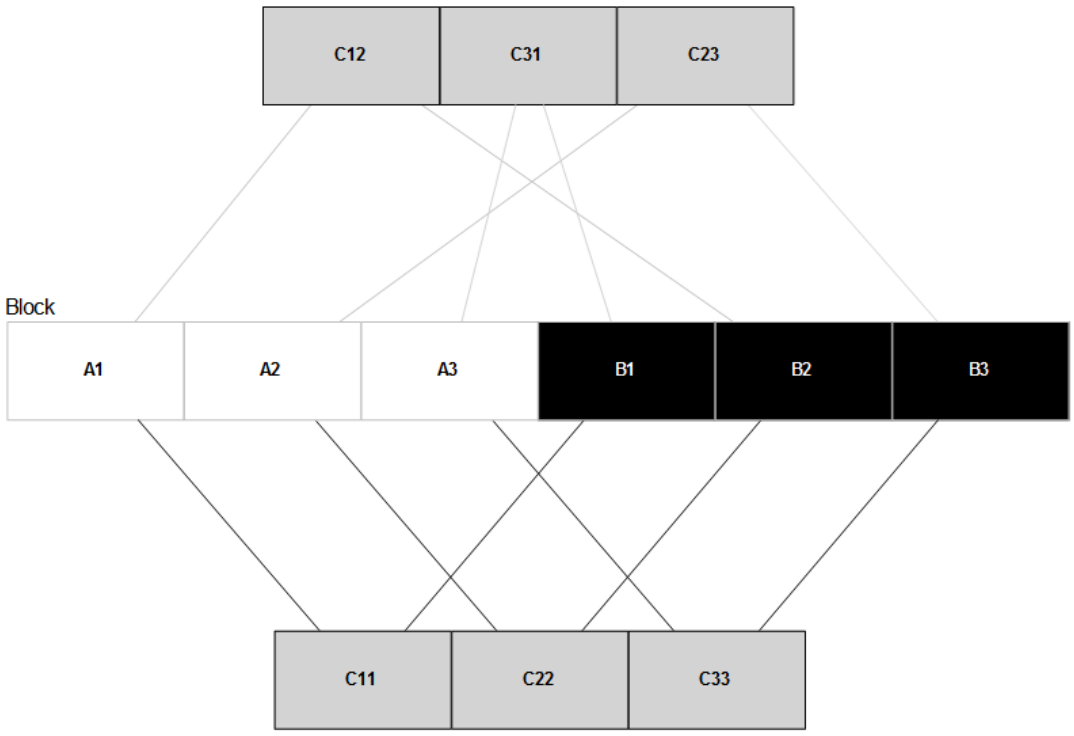

Figure 4.4: The strategy of generating redundancy packets

The reason why we use this model is that VANETs often loose packets in a consecutive

way. Imagine that we did not use the model we talked above, we use a new method called alternate grouping instead: we thus pick up the first packet of the block, assume it is the first packet in A. We pick up the second packet of the block and assume it is the first packet in B. We pick up the third packet of the block and assume it is the second packet in A. We continue to do this until all the packets in the block have their own group. Once the packet loss has occurred, assume we have lost a total of 3 consecutive packets at the beginning of the block. If we use the alternate grouping method described above, we will loose A_1, B_1 and A_2. Even when we have received C_1, we cannot recover A_1 or B_1. This is because we need two out of the following tree: A_1, B_1 and C_1. However, in our original model, although we loose 3 packets of A, which are A_1, A_2 and A_3, when we receive C_1, C_2 and C_3, we can easily recover all lost packet back by using Formula 4.5.

4.3.2 Header design

In order to add the redundancy support to RTP, based on the Erasure Coding model we have introduced above, we need to add some useful information into the header.

Since we have A packets, B packets and C packets, identifying a packet is very important. According to Formula 4.4, Formula 4.5 and Formula 4.6, i and j are also very important. For example, in the encoding process, we need i and j to generate C packets. In the decoding process, when a packet is lost, we can use i and j to find which packet we need in order to recover the lost packet. We call i and j index numbers. To record index numbers in a packet, we add a new field to the header. We call this field an index field.

As Figure 4.5 shows, we first used 2 or 1 byte to store the index field. To make the index field more flexible, we designed the 2 modes for the index field. They are called the normal mode and the compressed mode. They are designed for adapting to different situations. We use the first bit of the field as a flag bit. If the flag bit is set to 0, the index field is working in the normal mode. Otherwise, if the flag bit is set to 1, the index field is working in the compressed mode. In the normal mode, we use 2 bytes to store the

index field. The higher byte is used to store the index number for A, and the lower byte is used to store index number for B. If a packet is a C packet, the two bytes will store index number for the A and B packet which generate C. For example, we generate $C_1 1$ by using $C_1 1 = A_1 \oplus B_1$. For the A_1 packet, the index field in the A_1 packet header is set to 0000 0001 0000 0000. In the index field of A_1, the higher byte stores index number for A. Because the packet is an A packet, the lower byte is set to zero. For the B_1 packet, the index field is set to 0000 0000 0000 0001, the higher byte is set to zero, and the lower byte is used to store the index number. The first bit of the field is taken to record the mode and A packets and B packets should have same quantity, for this reason we can have $2^7 = 128$ A packets and B packets. Therefore, the maximum block size is $128 + 128 = 256$. However, in some situations, we do not need such a big block size. Therefore, we need to use the compressed mode. In the compressed mode, we use only one byte to store both index number of A and B. The 3 bits after the flag bit are used to store index number of A packets, the following 4 bits are used to store index number of B packets. Only the last 3 bits is used to store index number of B packets, because we need the A packets and B packets have same quantity. In the compressed mode, the protocol will support small blocks with a size of less than $2^3 + 2^3 = 16$. In Section 4.2.1, we discussed about the our EC

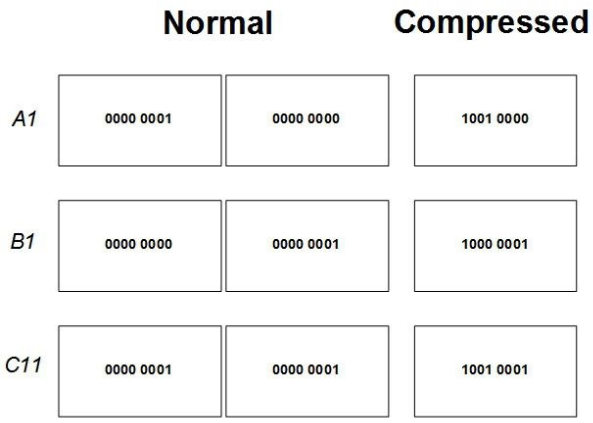

Figure 4.5: An example of Index field in different modes

model, performing XOR bit by bit requires that the packets are the same size. However,

in Section 4.1.2 we discuss that the H.264 payload RTP packets do not have the same packet size. The size of these packets depends on the original video frame size. Figure 4.6 describes this situation.

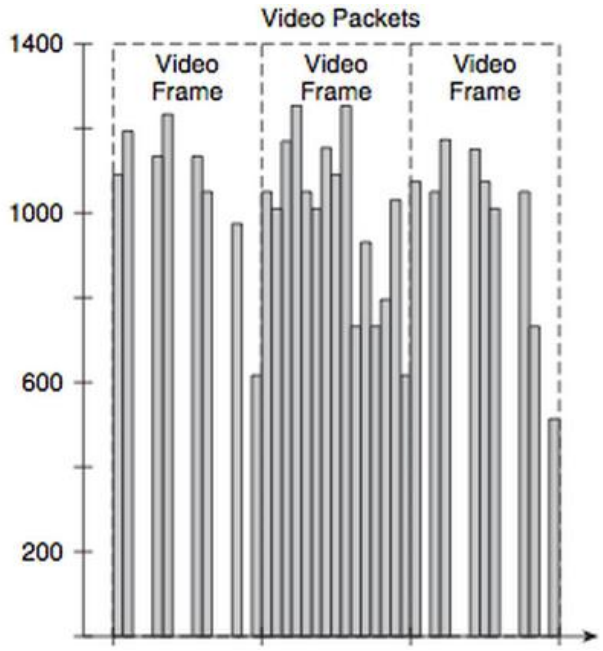

Figure 4.6: Packets will not have same length

According to [2], the larger NALUs can be split into several packets, while the small NALUs can be aggregated in one packet. With this feature, some packets will reach the maximum packet size, but some will not. This feature can help us decrease influence of the different sizes; however, this is not enough. What we need is for the packets to be exactly the same size. To achieve this goal, we first defined a maximum packet size. Choosing the MTU as the maximum packet size is an effective way. We then simply add 0 to the tail of a packet which is smaller than the maximum packet size. So that we can make the smaller packet reach the maximum packet size. Figure 4.7 shows how to make packets to have the same size.

After this process, there may be some useless zeros located at the end of a packet. Since these zeros are located at the end of a packet, we add a field to the header to record the

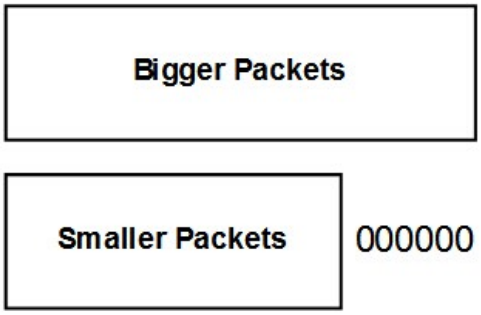

Figure 4.7: Make packets to have same size

original packet size. In the decoding process, we cut off the packet with according to its original size in this field. MTU is set ordinarily from around 1400 to 1500. Therefore, we used two bytes to store this field. At last we can support the packet size ranging from 0 to 65535.

Developed form all the previously discussed designs, the modified RTP header will be as it is shown in Figure 4.8.

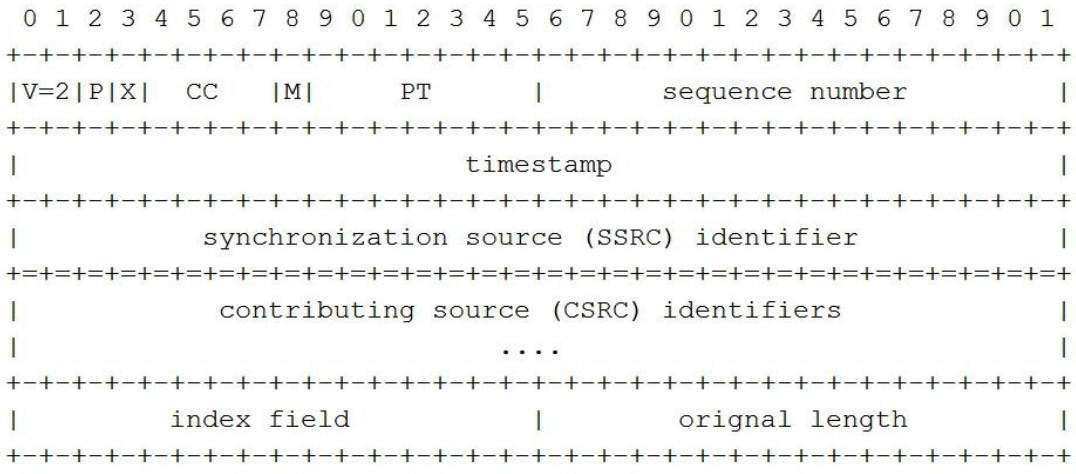

Figure 4.8: EC-RTP header

4.3.3 EC-RTP server

It is easy to build an EC-RTP server based on a RTP server. This is because of the similarity between the two protocols. The video encoding part will be the same as that of the RTP server. What we need to do is to generate the redundancy packets. Every packet header must then be modified and then sent.

4.3.4 Performance influenced by redundancy packets

A greater amount of redundancy packets means that more data will be required to be sent out. When network congestion happens, the bandwidth may not be enough to send out these packets. Therefore, redundant packets can impact the experience of playing a real-time video.

A proper way to solve this problem is to use the idea behind bit rate adjustment technique. We can change the number of redundant packets based on the current sending rate. A detailed design will be shown in Chapter 5.

4.4 EC-RTP to RTP Converter

4.4.1 Design

Figure 4.9 shows the basic topology of the EC-RTP to RTP converter. One converter is located on the boundary between the Internet and the VANETs. The other converter is located between VANETs and the RTP player. And both converters act as a pair. They know each other's IP address. And they will hence the following things: when one of the converters receives the RTP traffic, it will translate this traffic to EC-RTP traffic, and, the before-mentioned converter will then relay the traffics to the other converter. It will translate the packets back and send them to the right destination.

A detail design and implementation will be shown in Chapter 5.

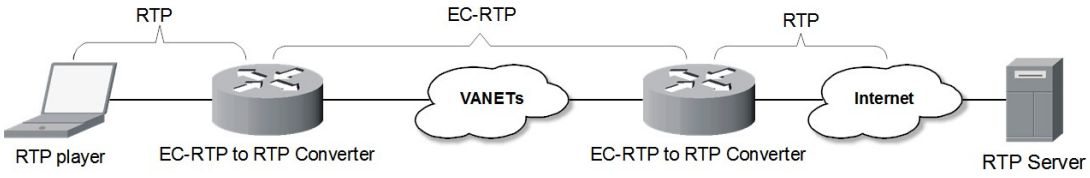

Figure 4.9: EC-RTP to RTP Converter

4.5 Generic Routing Encapsulation (GRE)

Generic Routing Encapsulation (GRE) is a tunneling protocol developed by Cisco Systems. It can encapsulate a wide variety of network layer protocols inside virtual point-to-point links over an Internet Protocol inter-network [40].

Figure 4.10 shows how GRE works. GRE carries other protocols as its payload. Therefore, GRE can carry any kind of packets without knowing what they really are.

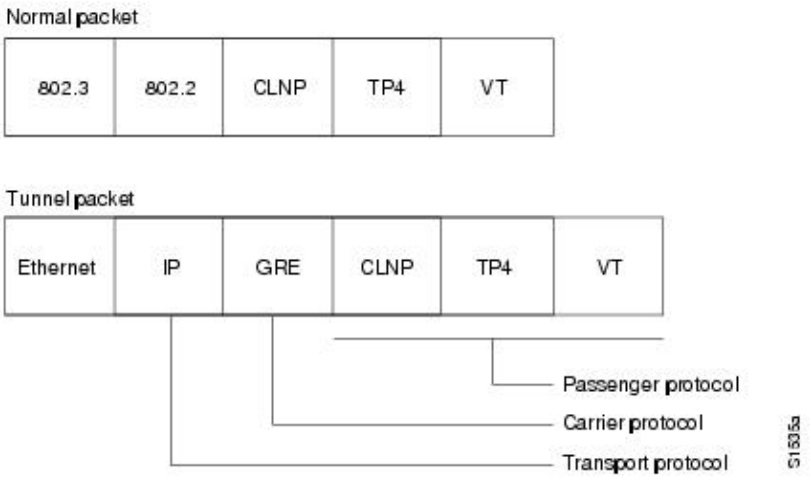

Figure 4.10: GRE header

4.6 EC Generic Routing Encapsulation (EC-GRE)

In Section 3.4.5, we designed a model for the redundancy tunnel, which can carry many different kinds of protocols. And in Section 4.5 we introduced the GRE, we can modified it for application on the a redundancy tunnel with same EC model we have introduced above, which we have called the EC-GRE.

The only difference between the EC-GRE and the EC-RTP is that EC-GRE carries RTP as its payload. Therefore, we can use the same modification used for the RTP to the GRE. We thus added the index field and the packet size field to the header of GRE. At last, we run the EC-GRE between the converters, now the converters become the tunnel gateways.

Chapter 5

Implementation

5.1 Overview

Most research about VANETs is currently based on the Network Simulator Version 2 (NS2) instead of on real devices. Though NS2 has its own advantages, the simulated results may differ from the results on achieved the real devices. After we have designed our protocols, we will implement them on the real devices.

Recently, Google and a number of automakers have been planning to bring Android to cars with the launch of a new group called the Open Automotive Alliance. The alliance consists of Google, GM, Honda, Audi, Hyundai, and chipmaker Nvidia. And it will focus on implementing the successful mobile operating system on in-car entertainment systems "in a way that is purpose-built for cars." The first cars with Android integration are planned to be launched by the end of 2014. Because the Android system is a potential system launched on the cars, we will focus on this system.

We will use Google's newest Android tablet Nexus 10 as our platform. Here is a summary of this device:

- CPU: Dual-core ARM Cortex-A15

- GPU: Mali-T604

- Memory: 2 GB RAM

- Wireless: Wi-Fi 802.11 b/g/n (MIMO+HT40)

- Android system: 4.2.1

- Cameras: 5 MP (main) ,1.9 MP (front)

We can use a pure Android device to run our simulation, because Nexus 10 is made by Google. In addition, the Wi-Fi connection of Nexus 10 is more stable.

We will use 3 Nexus 10 devices; and, each of them will play a different role in the system. The first one is the video streaming server. The second one is the video player. And the last one is a simulator. which can simulate packets loss. The video streaming server has 2 different modes. One provides the video which captured by the camera of the tablet, and the other mode is provides the video file which is inside the video server.

Figure 5.1: Google's Nexus 10

5.2 Topology

Figure 5.2 demonstrates the whole demonstration's topology. We used 3 tablets. One tablet is The Real-Time Streaming Protocol (RTSP) server, and another tablet is the RTSP player, both of these tablets are connected to the routing simulator of VANETs.

Since we want to show that our solution is compatible with Internet protocols, we will use the RTSP protocol. This will be introduced in the following section.

- RTSP server: this is the place where we provide the video streaming service.
- VANETs Routing simulator: the server and player will connect to this tablet, like a wireless access point. In addition, the tablet will here drop some packets randomly. We can thus simulate the situations that often occur in VANETs, such as transparent route changed and packet loss.
- RTSP player: this is the client side of the system. This tablet will contact the video streaming service and play the stream which the server sends back.

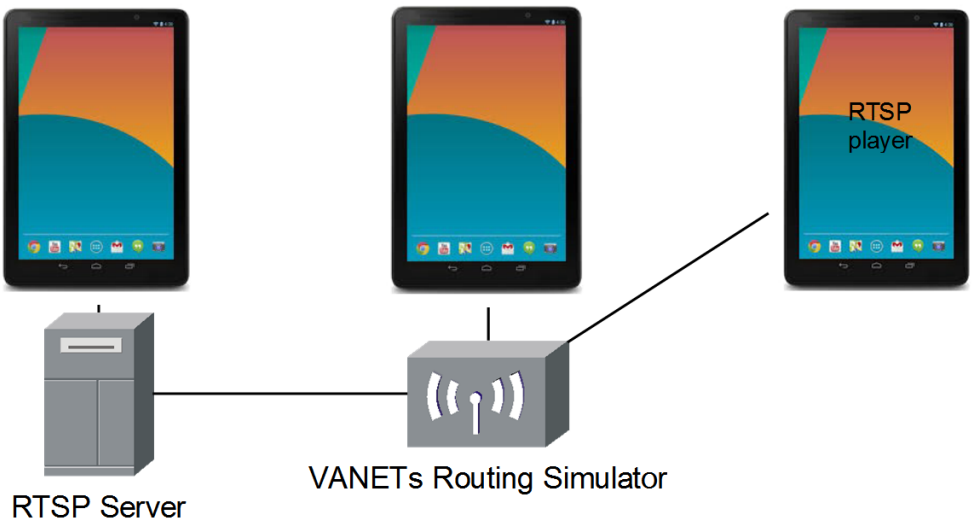

Figure 5.2: Topology

5.3 RTSP server

5.3.1 Why RTSP is used

RTSP is a very famous protocol. Here is a list of reasons why we have chosen it.

- RTSP is one of the IETF standards, the document for this is RFC 2336 [**?**]. The standard helps the protocol maintain consistent behaviour, no matter what company implements it.

- This protocol will use RTP as its transport layer protocol. We can easily convert it to our own EC-RTP.

5.3.2 RTSP protocol

For now, there are few RTSP servers based on the Android system. One aspect of our work is to implement a proper RTSP server on the Android system. Before we do this, we need to know how this protocol works.

RTSP overview

The Real-Time Streaming Protocol (RTSP), developed by RealNetworks and Netscape, is an application layer protocol; it establishes and controls either a single stream or several time-synchronized streams of continuous media such as audio and video.The protocol is intentionally similar in syntax and operation to HTTP/1.1. The RTSP server uses Uniform Resource Locator (URL) [41] to identify resources.

When a client requests a URL, the server will first build a session for the client. This is so that it can track every client. And then, the server use the Session Description Protocol (SDP), which will be introduced in the following section. This will be used to give the streaming information to the client; the server will then negotiate with the client concerning

the transport protocol (RTP) and the port number for each instance of streaming. After this process is done, the server will start sending video streaming to the client with the protocol they negotiated.

RTSP can be based on TCP or UDP. In our implementation, because of the high packet loss rate in VANETs, we used TCP to carry the packet of RTSP.

RTSP URL

RTSP URL is used to locate the resources on the server. We can give a URL to a single file, or we can give a URL to a device such like video camera, microphone, etc. The URL parser is one of the key parts of our implementation; we used the Application Programming Interface (API) of Android to implement this part [42].

RTSP URL grammar:

$RTSP\_URL = (\text{“}rtsp:\text{”}|\text{“}rtspu:\text{”}|\text{“}rtsps:\text{”})\text{“}//\text{”}host[\text{“}:\text{”}port][abs_path]$

Some examples:

$rtsp://192.168.20.136:5000/$

$rtsp://www.example.com/media.mp4$

RTSP methods

RTSP has 2 types of methods; one is the request method, and the other is the response method. Both have different formats.

Below are some useful methods we need to implement [43].

1. OPTION

   OPTION method is a request method; the client uses it to query how many kinds of methods are available on the server.

2. DESCRIBE

   a client use the DESCRIBE method to ask the server for information about the media data it requested. In addition, a client can use the Accept header to tell the server it can be understand the media data on the client side.

3. SETUP

   The SETUP method can let clients on the server side tell what kind of transport protocols and port numbers it is able to use to receive streaming data. Even in the case of a stream already built up, the client can also use the SETUP method to tell whether the server uses other protocol to transmit data. If server does not support it, it will respond with the error message: "45x Method not valid in this state".

4. PLAY

   A client uses the PLAY method to tell the server that it is ready to receive streamed data with the protocol and port number confirmed by the SETUP method. Therefore, if the client does not receive the acknowledgement of the server for the SETUP method, the client cannot send out the PLAY's request.

   The PLAY method can use a Range header to determine the time region of data it needs from the server side. This means the Range header will give the start and end position of a stream. Besides this, the PLAY request can be pipelined (queued). A server should handle the PLAY requests in the order of arrival.

   One PLAY request without a Range header is also valid. It represents a stream which will be played from the very beginning to the end. The stream may, however, be played from the very beginning to the time when it is paused. If a stream has been paused, the start point of the stream will be the point where it was paused.

5. TEARDOWN

   The client use the TEARDOWN method to tell the server that it does not need any more data. The server will close the session and stop the data transmission.

6. GET_PARAMETER

   The client uses this method as a heartbeat packet to determine if the server is still alive.

Here is an example of methods communication between the RTSP Server(S) and the RTSP Client(C). this process shows how a client connects to the server.

1. $S \leftarrow C$:

   OPTIONS rtsp://www.example.com/media.mp4 RTSP/1.0

   CSeq: 1

   Require: implicit-play

   Proxy-Require: gzipped-messages

2. $C \leftarrow S$:

   RTSP/1.0 200 OK

   CSeq: 1

   Public: DESCRIBE, SETUP, TEARDOWN, PLAY, PAUSE

3. $S \leftarrow C$:

   DESCRIBE rtsp://audio.example.com/twister/audio.en RTSP/1.0

   CSeq: 2

4. $C \leftarrow S$:

   RTSP/1.0 200 OK

   CSeq: 2

   v=0

   o=- 2890844526 2890842807 IN IP4 192.16.24.202

s=RTSP Session

m=audio 0 RTP/AVP 0

a=control:rtsp://audio.example.com/twister/audio.en

m=video 0 RTP/AVP 31

a=control:rtsp://video.example.com/twister/video

5. $S \leftarrow C$:

   SETUP rtsp://audio.example.com/twister/audio.en RTSP/1.0

   CSeq: 1

   Transport: RTP/AVP/UDP; unicast; client_port=3056-3057

6. $C \leftarrow S$:

   RTSP/1.0 200 OK

   CSeq: 1

   Session: 12345678

   Transport: RTP/AVP/UDP; unicast; client_port=3056-3057;server_port=5000-5001

7. $S \leftarrow C$:

   SETUP rtsp://video.example.com/twister/video RTSP/1.0

   CSeq: 1

   Transport: RTP/AVP/UDP; unicast; client_port=3058-3059

8. $C \leftarrow S$:

   RTSP/1.0 200 OK

   CSeq: 1

   Session: 23456789

   Transport: RTP/AVP/UDP; unicast; client_port=3058-3059; server_port=5002-5003

9. $S \leftarrow C$:

   PLAY rtsp://video.example.com/twister/video RTSP/1.0

   CSeq: 2

   Session: 23456789

   Range: smpte=0:10:00-

10. $C \leftarrow S$:

    RTSP/1.0 200 OK

    CSeq: 2

    Session: 23456789

    Range: smpte=0:10:00-0:20:00

    RTP-Info: url=rtsp://video.example.com/twister/video;seq=12312232;rtptime=78712811

11. $S \leftarrow C$:

    PLAY rtsp://audio.example.com/twister/audio.en RTSP/1.0

    CSeq: 2

    Session: 12345678

    Range: smpte=0:10:00-

12. $C \leftarrow S$:

    RTSP/1.0 200 OK

    CSeq: 2

    Session: 12345678

    Range: smpte=0:10:00-0:20:00

    RTP-Info: url=rtsp://audio.example.com/twister/audio.en; seq=876655; rtptime=1032181

Now that we have obtained our RTSP server, because the negotiation process is very important and short, we have decide to use TCP to transfer these RTSP methods. In addition, we need to use SDP protocol in the negotiation progress. It will be introduced in the following section. However, SDP is one protocol used for describe media. So that, before introducing SDP protocol, we first need to understand the media data we transmitting. It is the MP4 file.

5.3.3 MP4 file

We first analysed the MP4 file, based on the two modes we wanted to implement (file and real-time video), before we continued to introduce the SDP protocol. By gathering information about the MP4 file, we can let our RTSP protocol provide the SDP feedback.

ISO/IEC 14496-14:2003 determines that the MP4 file format is derived from ISO/IEC 14496-12 and ISO/IEC 15444-12; the ISO based media file format. This revises and completely replaces Clause 13 of ISO/IEC 14496-1, in which the file format was previously specified.

The MP4 file format defines the storage of the MPEG-4 content in files. It is a flexible format. It permits a wide variety of use, such as editing, display, interchange and streaming.

The basic unit of the MP4 file is Box. These Boxes can contain either data or metadata. MP4 standard permits us to use different data and metadata Boxes. Normally, putting the metadata before data, we can read more information before playing this video (audio). Figure 5.3 shows the structure of the MP4 file .

Box

First, the order of bytes in a Box is the network byte order known as Big-Endian. This simply means that, high bytes are located in the low end of memory storage. The Box is composed of by the header and body. The header will indicate the type and size of the Box. And, the body will have a different format and meaning based on the type.

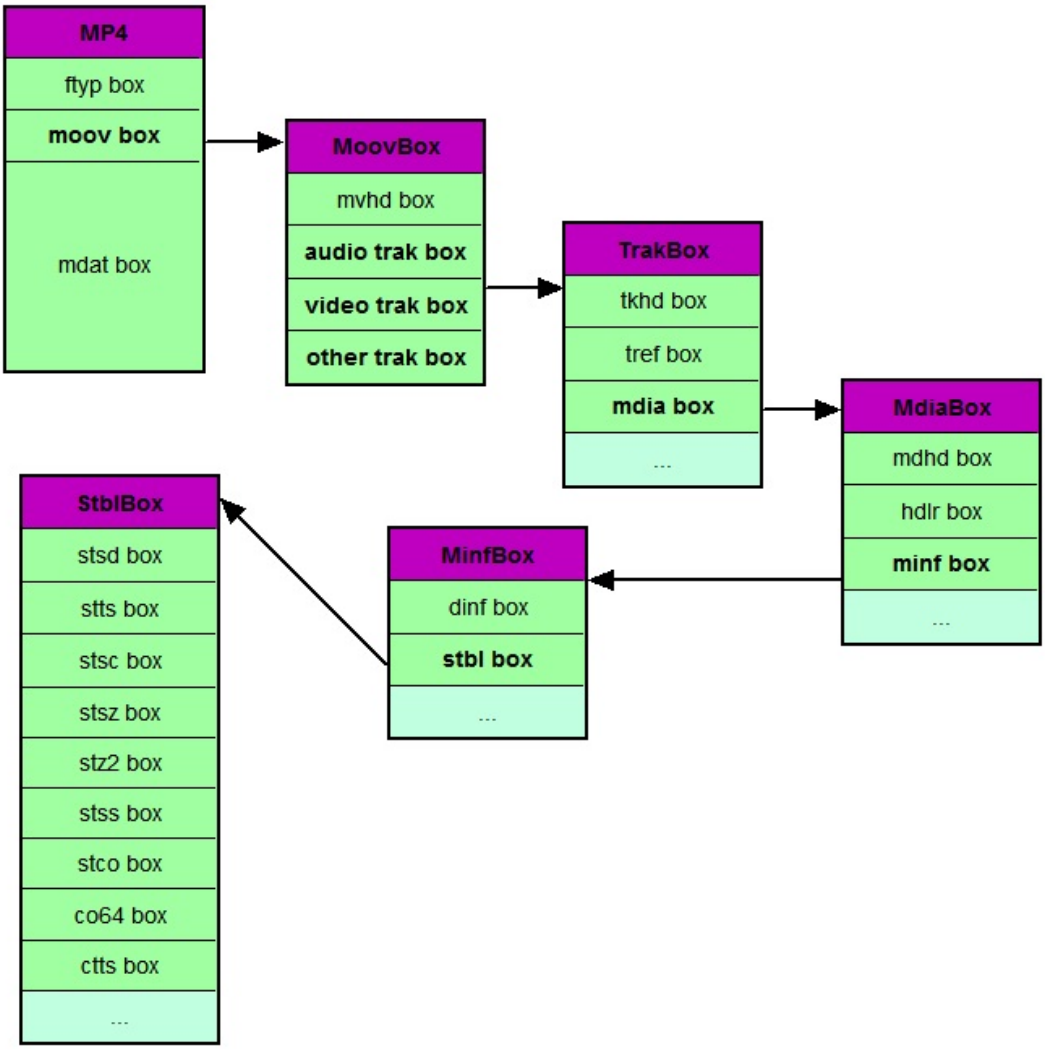

Figure 5.3: MP4 file structure

For our task, the biggest priority is to convey the following information about the MP4 file: width, height, fps, codec, etc. This information is located in the Moov Box. In the Moov Box, there are some Trak Boxes. These Boxes are used to describe the media information concerning the video and audio. So first, we need to examine the Trak Box one by one, and to determine whether it is a video track or not. And we need to then discover if the video is H.264 video or not. In the Trak Box, we have the Mdia Box, which is used to record the information about this media. In the Mdia Box, we can find the Minf Box. and in Minf we can find the Stdl, and in the Stdl, we can find Stsd. This is our final location. The H.264 video will have an Avc1 Box in the Stsd. The Avc1 is also named "AVCDecoderConfigurationRecord". We need SPS and PPS in the Avc1 box.

SPS and PPS

SPS is also named Sequence Parameter Set. It includes all necessary information about one sequence. PPS is also named Picture Parameter Set. It includes information about a picture such as the type and sequence number. Both sets are very important for the player to play the video successfully.

To obtain these two parameters, we need to read the Avc1 box. Figure 5.4 shows its structure, as well as the method needed to properly determine it.

5.3.4 SDP protocol

The Session Description Protocol (SDP) provides a standard representation of information to participants. This information includes the following: media details, transport addresses, and other session description metadata. It is purely a format for session description – it does not incorporate a transport protocol. We used SDP in RTSP to answer the method DESCRIBE of client.

Table 5.1 shows the main description of SDP. For us, the relevant information is the SPS and PPS. Figure 5.1 shows an example of how to describe a video stream. In the

unsigned int(8) configurationVersion = 1;	8bit
unsigned int(8) AVCProfileIndication;	8bit
unsigned int(8) profile_compatibility;	8bit
unsigned int(8) AVCLevelIndication;	8bit
bit(6) reserved = '111111'b;	6bit
unsigned int(2) lengthSizeMinusOne;	2bit
bit(3) reserved = '111'b;	3bit
unsigned int(5) numOfSequenceParameterSets;	5bit
for (i=0; i< numOfSequenceParameterSets; i++) {	
unsigned int(16) sequenceParameterSetLength ;	16bit
bit(8*sequenceParameterSetLength) sequenceParameterSetNALUnit;}	
unsigned int(8) numOfPictureParameterSets;	8bit
for (i=0; i< numOfPictureParameterSets; i++) {	
unsigned int(16) pictureParameterSetLength;	16bit
bit(8*pictureParameterSetLength) pictureParameterSetNALUnit;}}	

Figure 5.4: AVCDecoderConfigurationRecord structure

Parameter	Purpose
v =	Protocol version
o =	Session owner / Session ID
s =	Session name
*i =	Session information
*u =	Session URI
*e =	E-mail address
*p =	Phone number
*c =	Connection information
*b =	Bandwidth information
*z =	Time zone
*k =	Encryption key
*a =	Session attribute
t =	Session activity time
r =	Repeat times
m =	Media name and transport address

Table 5.1: SDP session description (* is optional)

sprop-parameter-sets parameter, we attached the SPS and PPS. For the file mode, we read the file, found the Avc1 Box, and read the SPS and PPS. If we are now in real-time video mode, we can record the video data captured from the camera on the server for one second. We then use an Android API, save these data to a temporary MP4 file. We then can get the parameter as we do in the file mode. And at last we delete that file.

```
m=video 0 RTP/AVP 96
b=AS:13953
a=rtpmap:96 H264/90000
a=mpeg4-esid:201
a=control:trackID=65736
a=fmtp:96 profile-level-id=428015; packetization-mode=1; sprop-parameter-sets=Z00AH41oBQBbkA==,aO4G8g==
a=framesize:96 1280-720
```

Figure 5.5: A SDP example

5.3.5 RTCP protocol

RTP is intended for transmitting audio, video and other real-time data transmissions. Compared the traditional transport level protocols which focus on reliability, RTP focus on real-time performance. It often works with RTCP. RTCP can help RTP manage the session, through flow control, congestion control, etc.

Table 5.2 shows 5 types of RTCP packets. We use RTCP to transport the packet loss rate information between the sides of server and client. This information can be used to adjust our redundancy packet numbers. Besides, we can use RTCP to transport the sending rate information, so that we can let the bit rate of our video adjust by the sending rate.

5.3.6 Architecture of RTSP server

We now have many protocols, so our next step is to implement them and make everything work together. We divided our RTSP server into four layers. The first layer is the UI layer. It takes charge of communicating with the user. And it can let the user set some

Type	Name	Purpose
200	SR	Sender Report
201	RR	Receiver Report
202	SDES	Source Description Items
203	BYE	Stop transmitting
204	APP	Used for a specific application

Table 5.2: 5 types of RTCP packet

parameters of the server. The second layer is the RTSP layer. It has some functions that include parsing RTSP method and RTSP URL, in addition to controlling RTSP sessions. These functions enable the server to negotiate with the client. And they can then set up the proper transport protocol between them. The third layer is the streaming abstract layer. It helps the system manage streams. When the server decide to send out a video stream with the negotiated protocol, the server chooses a proper packetizer of that protocol to packetize this stream. The last layer is the function layer. It have many useful classes of function, such as RTP packetizer, MP4 file parser, some high efficiency data structures, etc. Figure 5.6 shows all of them.

Basically, when the server receives a request, the URL parser will first try to locate the resource on the server. And then, the session controller will then help the client to build up a session. The server and the client will then communicate with each other using the RTSP methods. In the DESCRIBE method, session controller will use the MP4 file parser to generate the SDP information. In the SETUP method, the session controller will give a stream to the client session. This stream is set up with a proper transporting protocol. The packetizer of the chosen transporting protocol will start to generate packets.

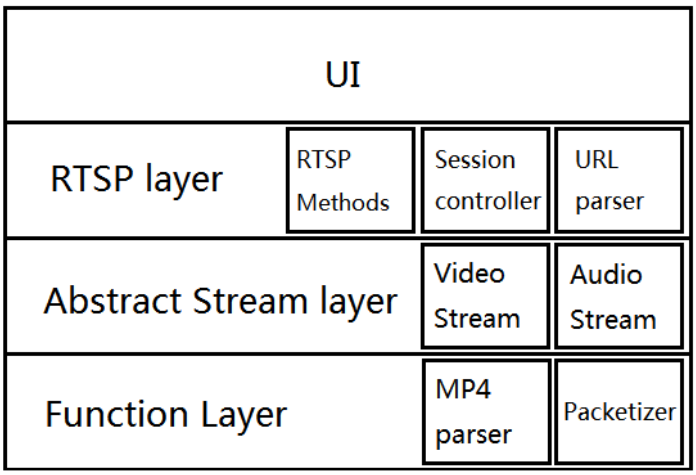

Figure 5.6: Architecture of RTSP server

5.4 RTSP player

Google's Android Software Development Kit (SDK) has a VideoView class which can display a video file. The VideoView class can load images from various sources (such as resources or content providers). It takes care of computing its measurements from the video so that it can be used in any layout manager. And it provides various display options such as scaling and tinting. And it can play the RTSP URL directly. [44] To validate our system compatibility, the player will not be changed. We will use the exact same one as Google's Android SDK relies upon. Actually, we used this client as a black box.

5.5 Converter

5.5.1 Architecture of Converter

The converter will have two roles. It will function both as the sender and as the receiver. Both of them can use RTCP to send some key information to each other like the packet loss rate. Each of them will have 3 threads per session. One is in charge of receiving packet,

sorting them into the right order. One is in charge of decoding/encoding the redundancy packets, and the other one is in charge of sending packets out. Each thread can work independently, and all threads have their own memory. Therefore, we can decrease the delay caused by sorting, decoding and encoding. Figure 5.7 shows the architecture of the converter.

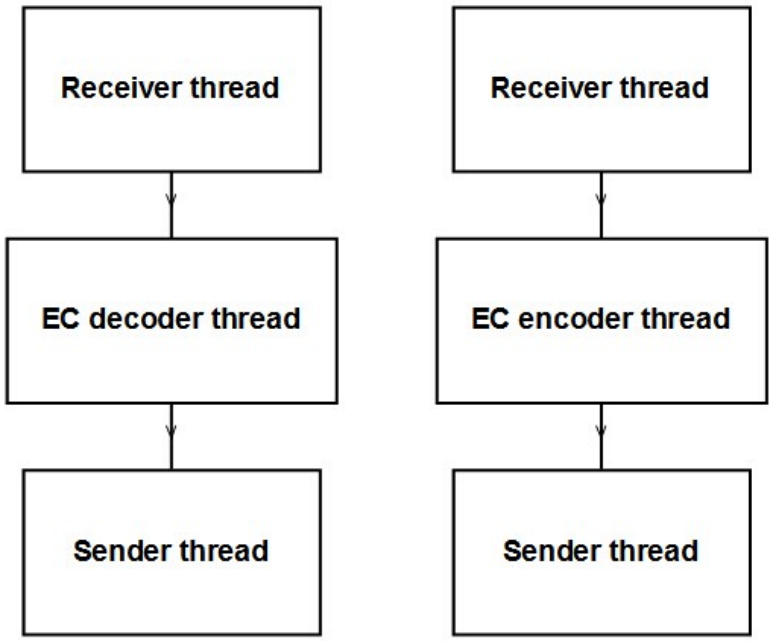

Figure 5.7: Architecture of Converter

5.5.2 Encoding

The strategy of encoding will follow what we have discussed in Chapter 4. However, in the implementation level, there is still one thing that needs to be considered. If the converter receives and stores the RTP packets until it has a full block. It will then generates the redundancy packets and send out all the packets. Because it needs to wait for the reception of all packets of a block, that will cause unwanted delay. In this case, we use a cache to

copy the receiving packets, and then forward these packets out. Once we have a whole block, we can generate the redundancy packets and send them out.

5.5.3 An EC-RTP/RTP packets sort algorithm

When receiving EC-RTP/RTP packets, one difficult problem lies in how to determine whether a packet has not yet reached its destination due to the network delay, or it just have been lost during the transport. Because the delay of each packet may be different, the packets may not reach their destination in the order they were sent in. Thus, an efficient algorithm for sorting the EC-RTP/RTP packets is required.

If we consider this problem, we can easily find that we cannot store all packets, and then sort them based on the sequence number field in the packet. The reason why we cannot do this is mainly because, first, we cannot save all the packets in the memory; and secondly, even if we use the quick sort algorithm to handle this, the time complexity will be $\mathcal{O}(n \lg n)$ which will cause an unacceptable delay. One method that will likely be successful is to sort the packets as soon as we have received them.

As we have mentioned above, there will still be problems determining if a packet is delayed or lost. According to the RTP, we do not have any retransmission mechanisms, so a sensible method is to set up a timer and a time-out time. When a packet exceeds the allotted time, we treat this packet as a lost packet.

A high efficiency data structure: Bitmap

According to [45], a bitmap uses a bit to mark the value of an element, and the key can be the element itself. The bitmap actually is just an array, which can be used to map to other elements. Table 5.3 shows the advantages and disadvantages of this data structure.

The following presents an example of using bitmap to sort elements:

Advantages	Disadvantages
• High operational efficiency, no need to compare and swap. • Occupy less memory, for example, if we have about 10000000 elements, we just need about: $10000000/8bit = 1250000Byte = 1.25MB$.	• All the elements should not be duplicated.

Table 5.3: Advantages and disadvantages of using bitmap sorting elements

Imagine that we have the following numbers: 2,6,4,3,1. We need to sort them in an ascending order. The following shows the process of using a Bitmap to sort them.

Because we only have numbers less than 7, we only need 8 bits (1 byte) to represent all the elements. Firstly, we initialize this byte with all zeros (shown in Figure 5.8). Now, each digit represents a number. 1 means there is an element in that position. 0 means there is no elements in that position. For example, the first digit represents 0, the second digit represents 1, the third digit represents 2, etc. Also, we can use the first digit to represents 1, but, here we used the first digit 0 represented in the example.

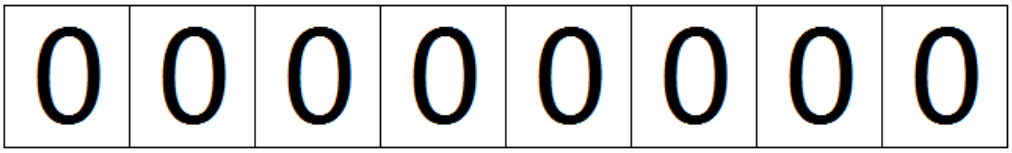

Figure 5.8: The state of the bitmap after initialization

In step two, we traversed all 5 elements. The first element is 2, we just changed the third bit from 0 to 1. Figure 5.9 shows the result.

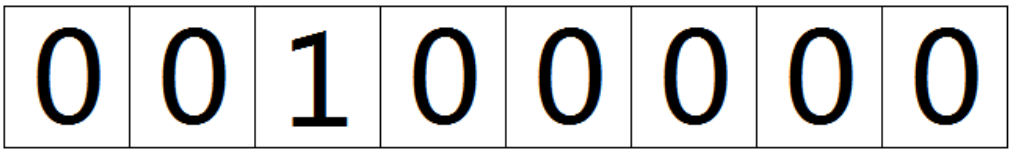

Figure 5.9: The sate of the bitmap after handling the first element

The following process will handle all the remaining elements and will repeat step two. The next element is 6; for this we change the seventh bit to 1. The element after 6 is 4, we change the fifth bit to 1. We will keep doing this, until we reach the last element. Figure 5.10 shows the final state of the bitmap.

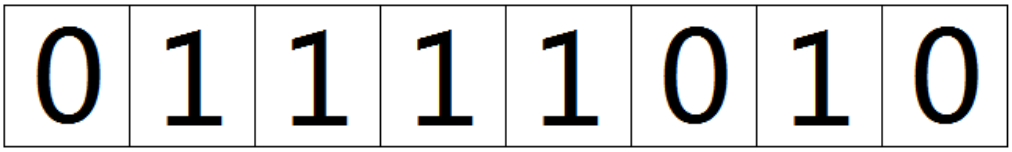

Figure 5.10: The final state of the bitmap after traversing all elements

When we have finished, we can traverse the bits in the bitmap. When we find a bit that has been set to 1, we output its position in the bitmap, so we will get the result: 1,2,3,4,6.

Bitmap can save space; and it also have a good speed for solving sorting problems. The time complexity will decrease to $\mathcal{O}(n)$. This is very suitable for the mobile device with limited CPU speed and limited memory. Additionally, all the packets for a session will have continuous sequence numbers. In other words, there will be no duplicate sequence number per session, only some missing numbers. We can therefore use a bitmap to sort these packets.

Designing a data structure to solve this problem

Now that we can use a bitmap to design a high speed cache, we can determine the right place of a packet by the $\mathcal{O}(1)$ time cost. First we can repeat the process as we described above. We use one bitmap and one array. The bitmap is used to map the index of the array to the element of array, just like Figure 5.11 shows.

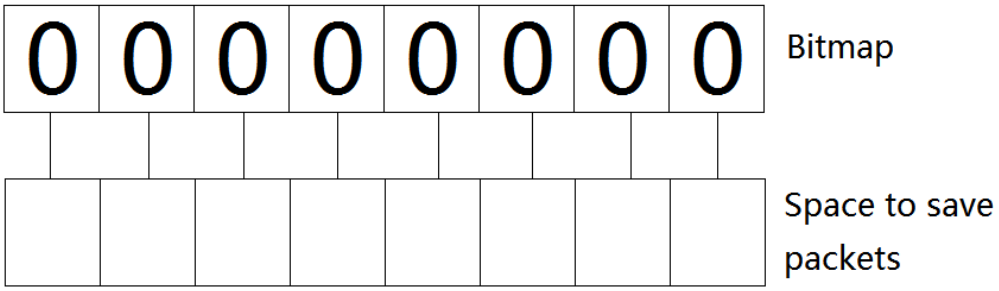

Figure 5.11: Basic data structure: one bitmap and one array

let us assume the sequence number of a packet is Seq, the index of array is i, and the cache size is S. We can get the relation between them with the following formula:

$$i = Seq \mod S \tag{5.1}$$

When we receive a packet, we read the sequence number of this packet. With this formula we can get the right memory space of the packet. And copy this packet into the right space. And then, we can set the relevant bit to 1.

However, we still have some problems. We are still concerned with When should we stop this process and send the packets out to the decoding thread. Just waiting for the packets to fill the cache may be one simpler way. However we need to solve the loss/delay problem first. This is because, if there is a packet lost, the cache cannot be filled up. Besides, we cannot store whole packets in the memory. We need to thus send some packets already in right order to the decoding thread. Therefore, we must determine when we

should send them out of the cache. If we do not send them out on time, the following packets may overwrite the previous packets. This is due to the Formula 5.1 we used.

It seems like only one bitmap does not satisfy our requirements; so, we added another bitmap to help us solve this problem.

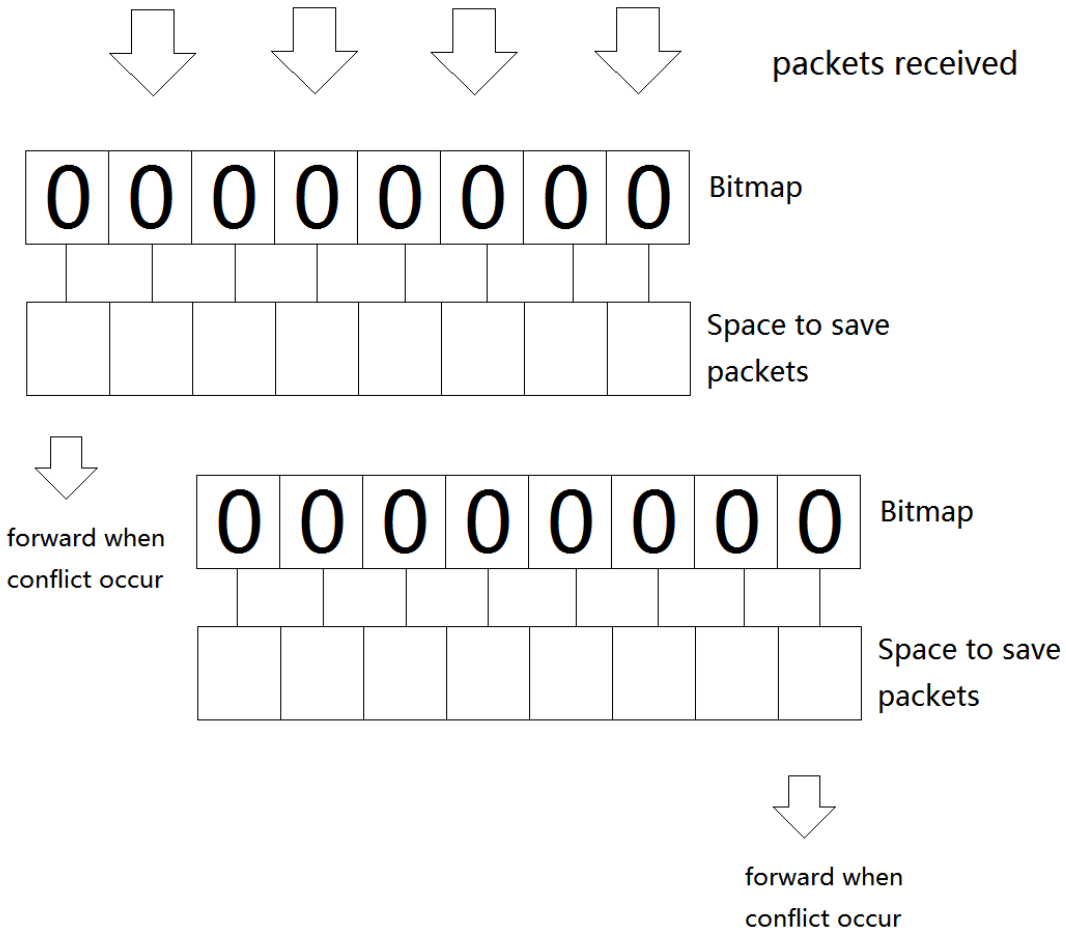

Figure 5.12: Advanced data structure: two bitmaps and two arrays

Now we have 2 bitmaps. When we receive one packet, we check the first bitmap; if there is already one packet, we check the second bitmap, if there is already one packet here, we forward all packets in the cache to the next process. At the same time, we record the quotient of Seq/S. This quotient become the $setFlag$ as Formula 5.2 shows. It will

give out the expected range of Seq of received packets. For example, if $setFlag = 1$, that means the received packets must in the range from $(setFlag - 2) * S$ (the packets in the second bitmap) to $setFlag * S$. If the Seq of a received packet is out of the expected range, we simply drop it.

$$setFlag = Seq/S \tag{5.2}$$

Algorithm Two Cache Switch packet sorting gives out the pseudo code of this sorting algorithm.

Two Cache Switch packet sorting

```
1  i ← packet.Seq mod S
2  nowSetFlag ← packet.Seq/S
3  if nowSetFlag == setFlag and isPacketCached[i] == 0
4     then cache[i] ← packet
5          isPacketCached[i] = 1
6          ▷ Save the packet in the cache and set the bitmap.
7  elseif nowSetFlag > setFlag
8     then
9          if sendSet == ture ▷ The second bitmap only need one bit
10            then cacheWaitToSend.sendOut()
11                 sendSet = false
12                 ▷ Send out the second cache.
13         if sendSet == false
14            then cacheWaitToSend[index] ← packet
15                 sendSet = ture
16         setFlag + +
17         renew(isPacketCached)
18         renew(cache)
19 elseif nowSetFlag == setFlag − 1
20    then swap(cache, cacheWaitTosend)
21         cache[index] = 1
```

5.5.4 Redundancy rate adjustment

When network congestion occurs, because the redundant packets consume more bandwidth, sending out more redundant packets will make things worse. In other words, we need to adjust the number of redundant packets based on the sending rate. If the sending rate is high enough, we can increase the number of redundant packets; if it is not, we need to

decrease the number of redundant packets. Figure 5.13 describes this process. Basically, the sender and receiver will use RTCP packets to inform each other of the packet loss rate.

First, we need to determine if there are any packets lost or not. If there is no packet loss, we do not need to send any redundant packets. If packet loss occurs, we also need check the video streams bit rate, if the bit rate is too low, this means there may have a network congestion. This is because RTCP will help our video stream become adjusted to the sending rate. In this case, we cannot send out more redundant packets. Besides, if there are no more lost packets, this does not means we will not have packet loss again later. So we decrease the number of redundant packets step by step, instead of stopping sending the redundancy packets out once for all.

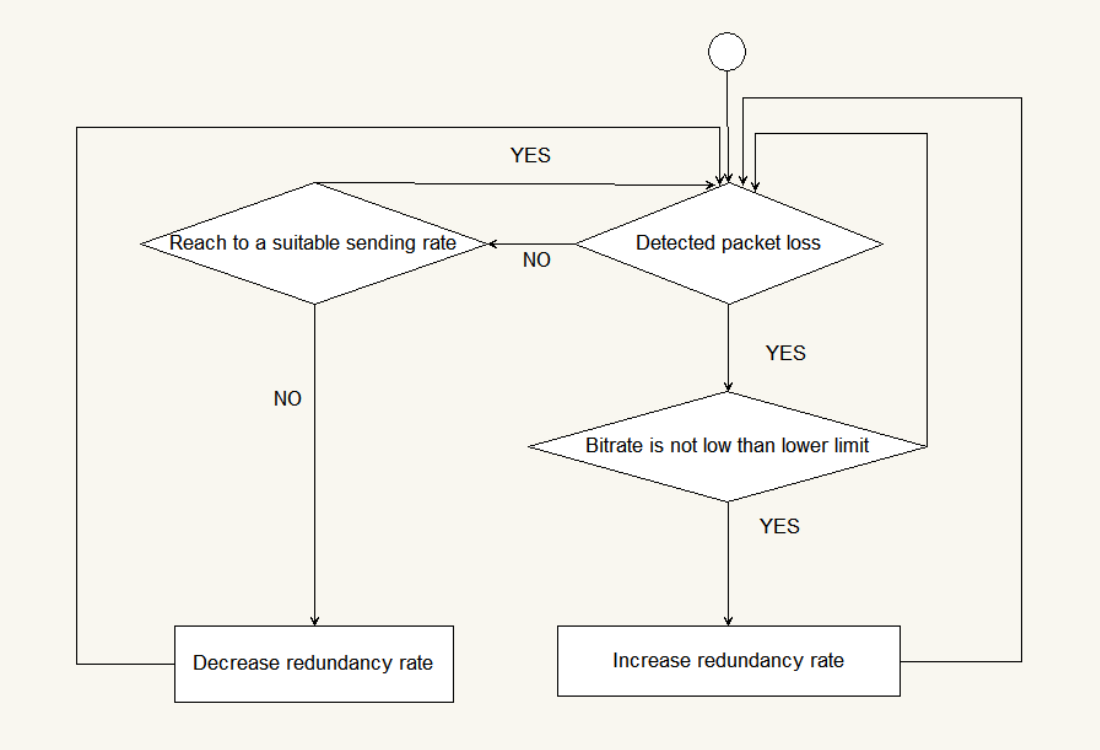

Figure 5.13: Redundancy Adjustment

5.6 Simulator

The main task of the simulator is to randomly drop packets. We need to make sure that each packets have an equal chance to be dropped. To achieve this goal, we first design a function which can generate the random value "drop" or not "drop" as input percentage. Algorithm Loose or Send gives out the pseudo code of this function.

Loose or Send

```
1  loss ← configuredLossPercentage
2  if loss == 0
3      then exit
4  rad = newboolean[10000]
5  for i ← 0 to 10000
6          do
7              if i < loss
8                  then rad[i] = false
9                  else
10                         rad[i] = true
11 index = radom.nextInt(9999)
12 return rad[index]
```

The main idea is to generate some values proportionally, for example, if we want a 10% loss, we generate 1000 false value and 9000 true value (false represents loss, true represents send) in an array. We use a random generator generate 0 9999 equitably. Thus the probability that the generator will choose true value will be 9000/10000 = 90% and the probability that the generator will choose false value will be 1000/10000 = 10%.

Use this function we can determine if a packet is lost or not.

Chapter 6

Experiment

6.0.1 Simulator Test

Before we begin the test component of our work, we must verify that the simulator is working perfectly. This test is built in the following way: we set up the packet loss rate at $10\%, 30\%, 40\%, 80\%$. The simulator finally gave us a stable estimated loss rate. At the same time, we have tested our 2 kinds of tunnels in each environment. Figure 6.1 shows the results; we have obtained our proper loss rate. Everything works fine.

6.0.2 Running Car Test

In this test, we put our system on a running car, to simulate a real scenario that may happen in the VANETs.

Before we started, we needed to know the maximum distance of the Wi-Fi communication between the 2 tablets used. This is because we do not know the power of the wireless device of our tablets. So we put the system in a stadium; with the help of the marks on the runway, we obtained the general communication distance.

Usually, we can find the 60 meter and 100 meter mark on the runway. Also, there are some marks on the runway like Figure 6.2 shows, which designate the 5 meters mark.

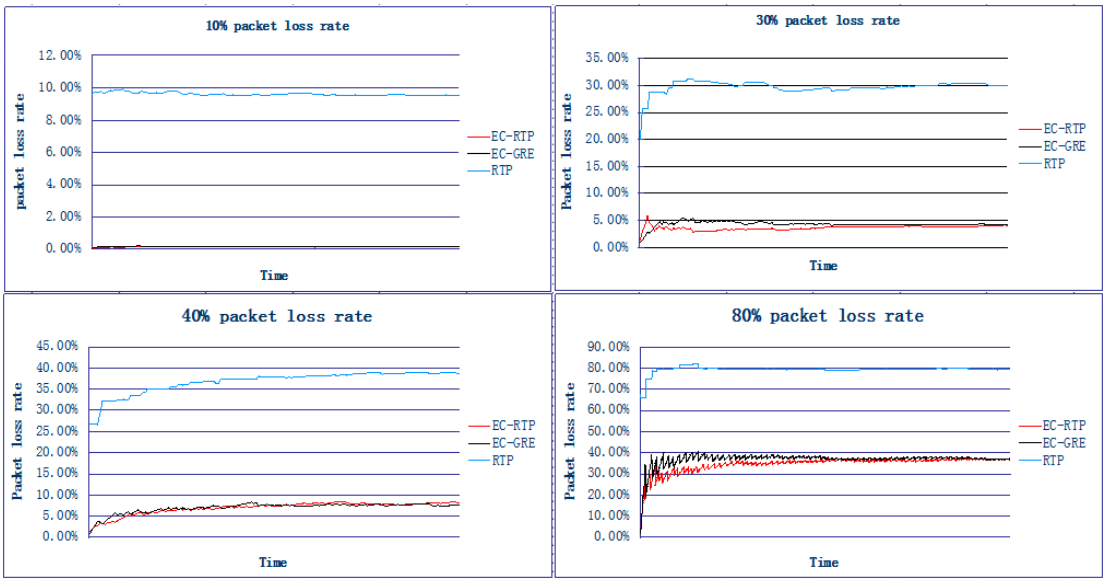

Figure 6.1: Result of testing simulator in different packet loss rate

Another important factor is signal disturbance. To ensure there will be no other Wi-Fi signal, we used software to measure the environment. The results of which are shown in Figure 6.3.

We have tried to deliver a single picture to determine the maximum distance between the two tablets. To reduce the test number, we used the binary search approach to let us quickly find the proper distance fast. The starting point is 100 meters, which is halfway between 0 and the theoretical maximum distance of the Wi-Fi (200 meters). If we can deliver the picture perfectly at this distance, the maximum distance will be between 100 meters and 200 meters. Otherwise, it means that the maximum is between 0 and 100 meters. Then we set the distance with the new middle value of the new range again, and we will repeat the test. Until we find the maximum distance.

Table 6.1 shows the result.

After we have finished the test, we know the maximum communicable distance is about 60 meters. Next, we need to find a location that can provide about 60 meters distance for our system. Figure 6.4 shows the test location. The green mark is the starting point and

No.	Distance	Result
1	100 m	Cannot find peers
2	50 m	Found peers, connected, transfer completed
3	75 m	Cannot find peers
4	60 m	Found peers, connected, transfer completed slowly
5	65 m	Found peers, cannot connect

Table 6.1: A series of testing maximum Wi-Fi communication distance results

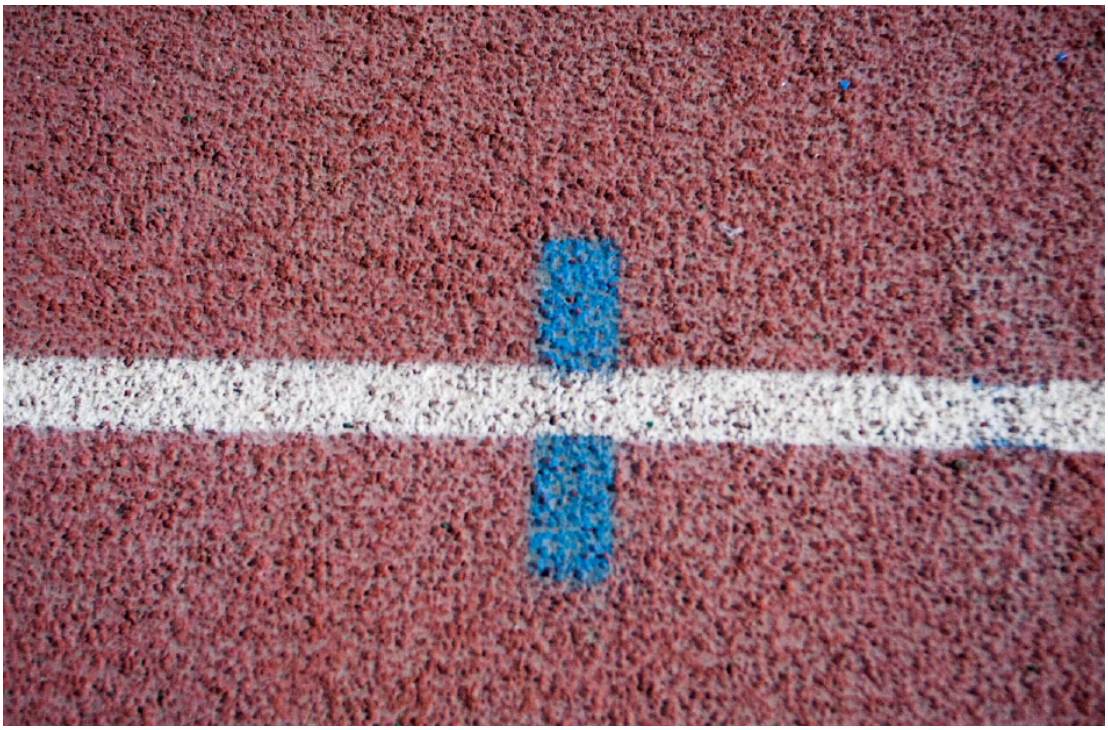

Figure 6.2: 5 meter mark on the runway

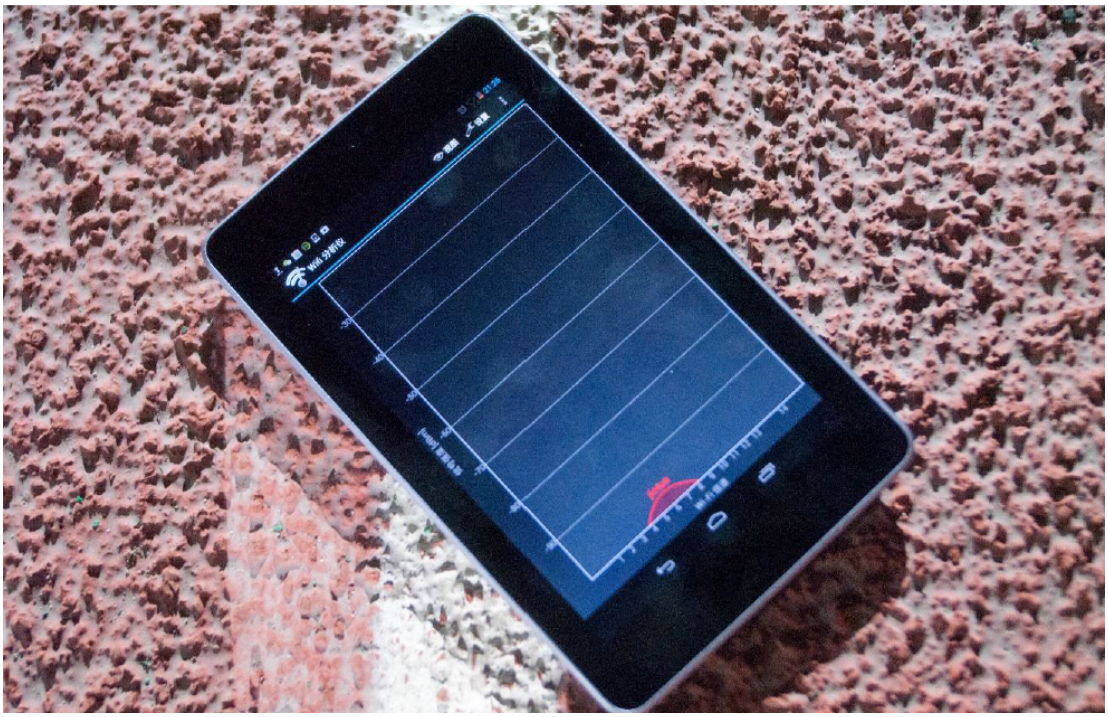

Figure 6.3: The other Wi-Fi signal on the stadium

the red mark is the end point.

The simulator and the RTSP server was set on the car, and the RTSP player was set on the roadside. We let the car go through the line we have set up. Figure 6.5 shows the final result.

6.0.3 Delay Test

In this test, we want to make sure that our simulator, converter, and tunnel will not cause a lot of delay. The RTP without the use of any features, will be our control group. We continue to use the same test as we have done in Section 6.0.1.

Figure 6.6 shows the final result; when there are a lot of packets lost, the solution we have applied may increase the delay by about 2 seconds. However, when the packet loss rate occurs in a reasonable region, the increment in the delay can be ignored.

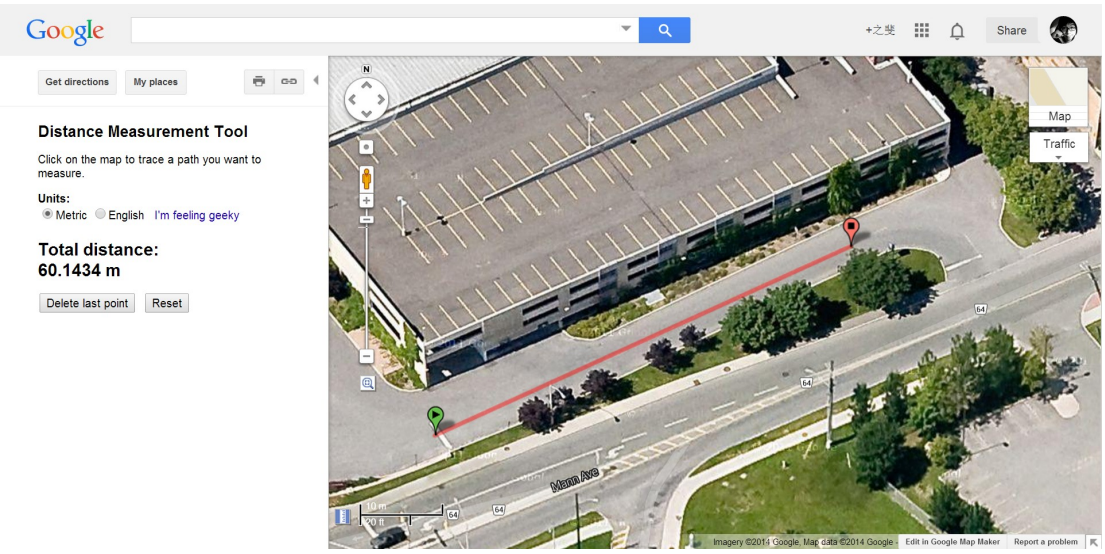

Figure 6.4: A map of the route on which ran the tests were run on

6.0.4 CPU consumption Test

In this test, we want to make sure that our simulator, converter, and tunnel will not consume a lot of resources from the tablet. We continue to use the same test as we done in section 6.0.1. We used software to record the CPU usage for 1 minute. And we compute the average and the maximum usage.

Figure 6.7 shows the final result. It tells us our algorithm is working fine. Even in case of the 80% loss, the maximum CPU usage does not increase more than 20%.

6.0.5 Video quality Test

In this experiment, we set the simulator to the 10% packet loss rate. We used the file mode so that we can compare the received file and the original file easily.

Peak signal-to-noise ratio

We used Peak signal-to-noise ratio (PSNR) as our video quality measurements. PSNR is the most commonly used measurement to measure the quality of reconstructed of lossy

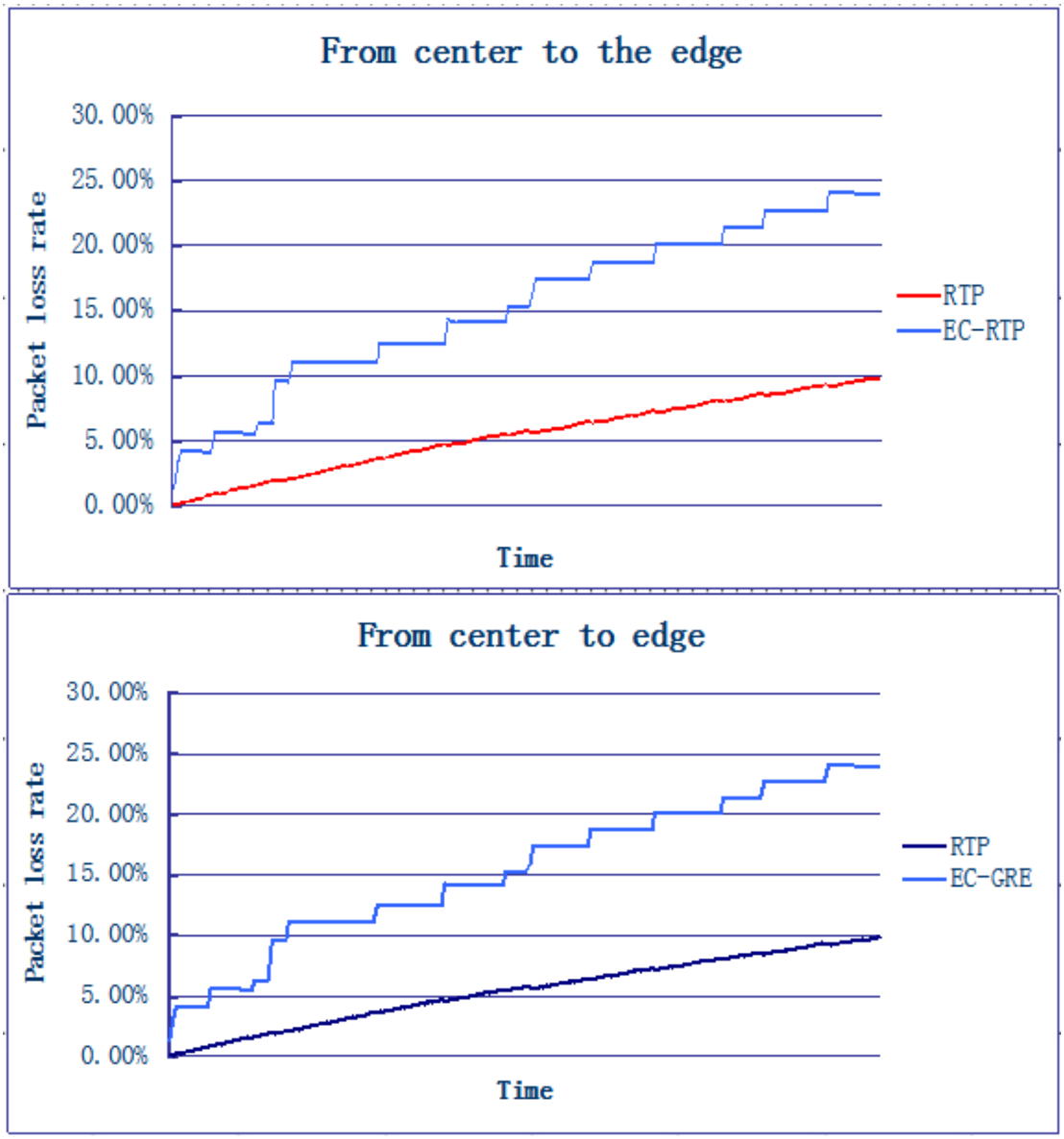

Figure 6.5: Results of testing a player in a car moving away from the server

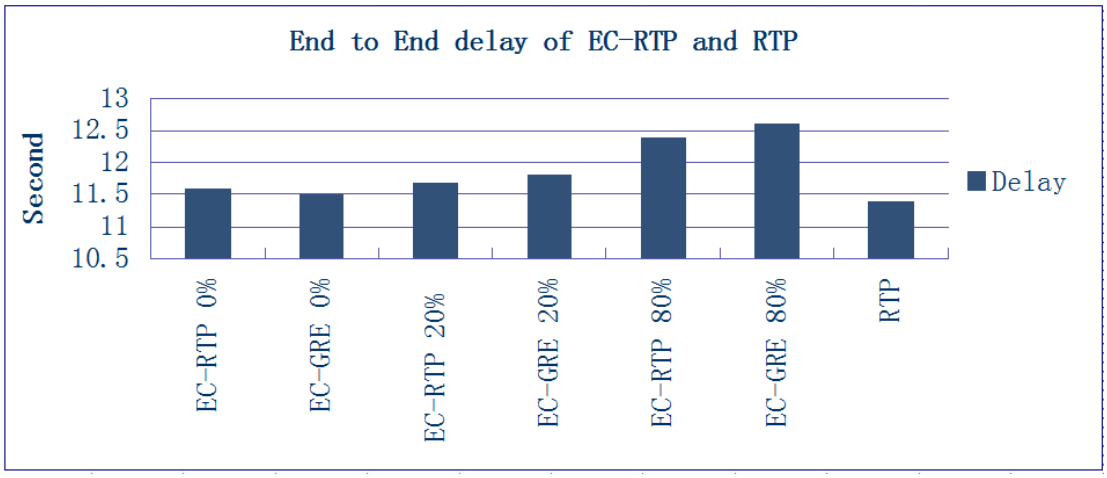

Figure 6.6: Results of testing delay in different packet loss rates

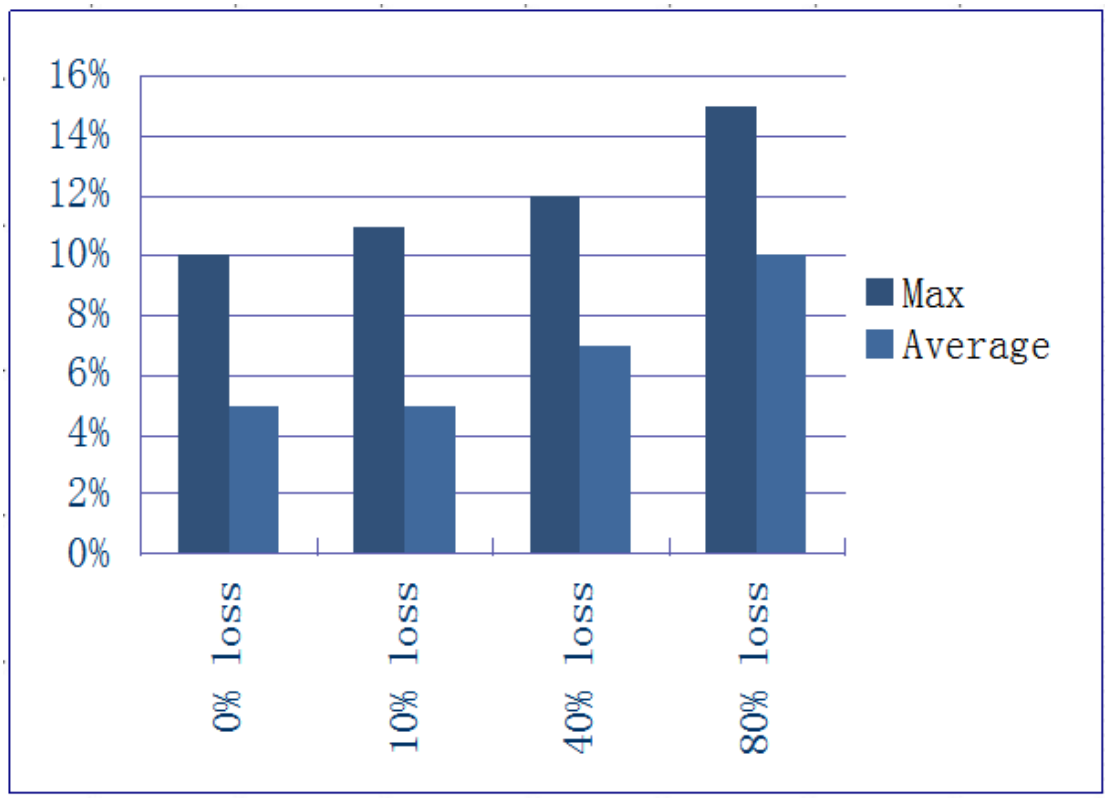

Figure 6.7: Results of testing CPU consumption in different packet loss rates

compression codecs.

PSNR is most easily defined via the mean squared error (MSE). Given a noise-free $m*n$ monochrome image I and its noisy approximation K, MSE is defined as 6.1.

$$MSE = \frac{1}{mn}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}[I(i,j) - K(i,j)]^2; \tag{6.1}$$

The PSNR can be defined as Formula 6.2, Formula 6.3 and Formula 6.4.

$$PSNR = 10\log_{10}(\frac{MAX_I^2}{MSE}) \tag{6.2}$$

$$PSNR = 10\log_{10}(\frac{MAX_I}{\sqrt{MSE}}) \tag{6.3}$$

$$PSNR = 20\log_{10}(MAX_I) - 10\log_{10}(MSE) \tag{6.4}$$

According to [46] and [47], acceptable values for wireless transmission quality loss are considered to be about 20 dB to 25 dB. In the absence of noise, the two images I and K are identical. And, thus the MSE is zero. In this case the PSNR is undefined.

Figure 6.8 shows the result that the video quality increases. We set our system to a 10% packet loss rate.

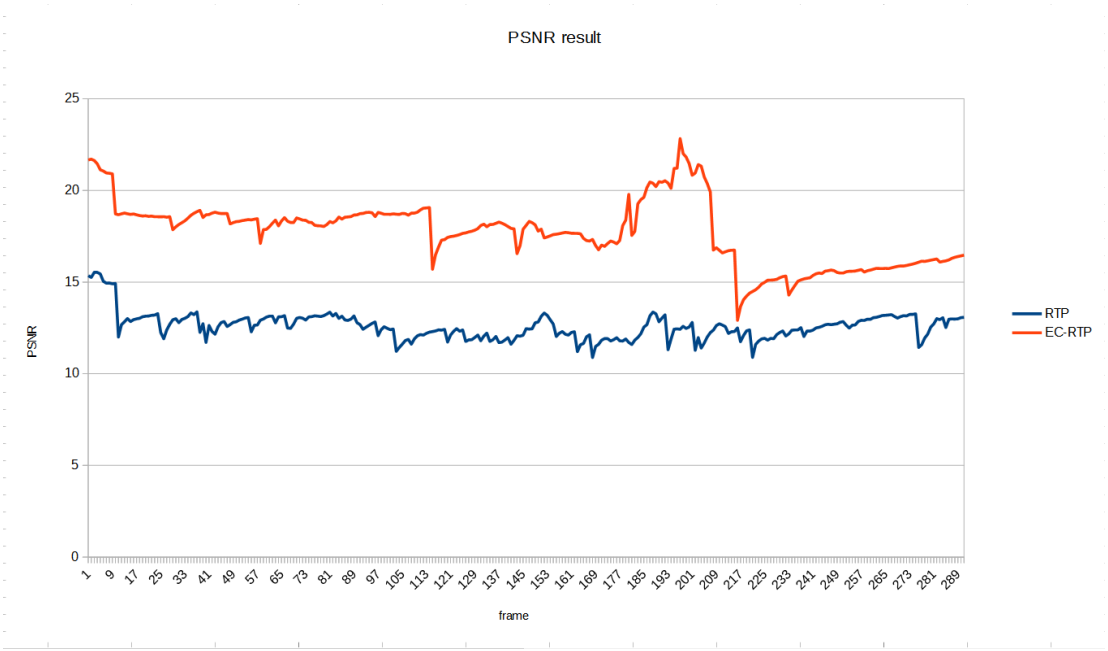

Figure 6.8: Results for testing video quality in a 10% packet loss rate

Chapter 7

Conclusion

In our work, we have implemented a RTSP server and a packet loss simulator on the Android system. This could be a good test platform for VANETs video streaming. We have also provide 2 models to solve the compatibility problem, and to make the software design for Internet can perform good result. We use EC technology to solve the high packet loss rate problem, and use the redundancy rate adjustment method to avoid the shortcomings of redundancy technique. At last we did some experiment with our system. The results shows that VANETs could be an extension of Internet, EC technology can provide our better video quality. We have additionally design a UDP sorting algorithm, which was a highly efficient method of sorting UDP packets. And it consume little resources.

Because our major work is to solve the compatibility issue, our video quality do not reach to a high level. However, if we combine other technique, we think we can get the better results. The importance of our work is that our solution is RTP compatible.

References

[1] Cristiano Rezende, Mohammed Almulla, Azzedine Boukerche, and Antonio A.F. Loureiro. The selective use of redundancy for video streaming over vehicular ad hoc networks. *ACM Transactions on Multimedia Computing, Communications and Applications*, July 2013.

[2] Adam Li. Rfc 5109. *RTP payload format for generic forward error correction*, 2007.

[3] Marco Pasin, Matteo Petracca, Paolo Bucciol, Antonio Servetti, and Juan Carlos De Martin. Error resilient real-time multimedia streaming over vehicular networks. *Vehicle Systems and Safety, Dallas, TX, USA*, 2009.

[4] James Lee Hafner. Weaver codes: Highly fault tolerant erasure codes for storage systems. In *FAST*, volume 5, pages 211–224, 2005.

[5] James S Plank, Jianqiang Luo, Catherine D Schuman, Lihao Xu, Zooko Wilcox-O'Hearn, et al. A performance evaluation and examination of open-source erasure coding libraries for storage. In *FAST*, volume 9, pages 253–265, 2009.

[6] Mingqiang Li, Jiwu Shu, and Weimin Zheng. Grid codes: Strip-based erasure codes with high fault tolerance for storage systems. *ACM Transactions on Storage (TOS)*, 4(4):15, 2009.

[7] Kevin M Greenan, Xiaozhou Li, and Jay J Wylie. Flat xor-based erasure codes in storage systems: Constructions, efficient recovery, and tradeoffs. In *Mass Storage*

Systems and Technologies (MSST), 2010 IEEE 26th Symposium on, pages 1–14. IEEE, 2010.

[8] James Lee Hafner. Hover erasure codes for disk arrays. In *Dependable Systems and Networks, 2006. DSN 2006. International Conference on*, pages 217–226. IEEE, 2006.

[9] Cristiano Rezende, Mohammed Almulla, and Azzedine Boukerche. The use of erasure coding for video streaming unicast over vehicular ad hoc networks. In *Local Computer Networks (LCN), 2013 IEEE 38th Conference on*, pages 715–718. IEEE, 2013.

[10] Jay J Wylie and Ram Swaminathan. Determining fault tolerance of xor-based erasure codes efficiently. In *Dependable Systems and Networks, 2007. DSN'07. 37th Annual IEEE/IFIP International Conference on*, pages 206–215. IEEE, 2007.

[11] Johannes Bloemer, Malik Kalfane, Richard Karp, Marek Karpinski, Michael Luby, and David Zuckerman. An xor-based erasure-resilient coding scheme. 1995.

[12] Maxwell N Krohn, Michael J Freedman, and David Mazieres. On-the-fly verification of rateless erasure codes for efficient content distribution. In *Security and Privacy, 2004. Proceedings. 2004 IEEE Symposium on*, pages 226–240. IEEE, 2004.

[13] Michael Luby. Lt codes. 2002.

[14] Richard Karp, Michael Luby, and Amin Shokrollahi. Finite length analysis of lt codes. In *Information Theory, 2004. ISIT 2004. Proceedings. International Symposium on*, page 39. IEEE, 2004.

[15] S-YR Li, Raymond W Yeung, and Ning Cai. Linear network coding. *Information Theory, IEEE Transactions on*, 49(2):371–381, 2003.

[16] Mohsen Sardari, Faramarz Hendessi, and Faramarz Fekri. Dmrc: dissemination of multimedia in vehicular networks using rateless codes. In *INFOCOM Workshops 2009, IEEE*, pages 1–6. IEEE, 2009.

[17] Mohsen Sardari, Faramarz Hendessi, and Faramarz Fekri. Ddrc: Data dissemination in vehicular networks using rateless codes. *Journal of Information Science & Engineering*, 26(3), 2010.

[18] Rudolf Ahlswede, Ning Cai, S-YR Li, and Raymond W Yeung. Network information flow. *Information Theory, IEEE Transactions on*, 46(4):1204–1216, 2000.

[19] Seung-Hoon Lee, Uichin Lee, Kang-Won Lee, and Mario Gerla. Content distribution in vanets using network coding: the effect of disk i/o and processing o/h. In *Sensor, Mesh and Ad Hoc Communications and Networks, 2008. SECON'08. 5th Annual IEEE Communications Society Conference on*, pages 117–125. IEEE, 2008.

[20] Uichin Lee, Joon-Sang Park, Joseph Yeh, Giovanni Pau, and Mario Gerla. Code torrent: content distribution using network coding in vanet. In *Proceedings of the 1st international workshop on Decentralized resource sharing in mobile computing and networking*, pages 1–5. ACM, 2006.

[21] Shabbir Ahmed and Salil S Kanhere. Vanetcode: network coding to enhance cooperative downloading in vehicular ad-hoc networks. In *Proceedings of the 2006 international conference on Wireless communications and mobile computing*, pages 527–532. ACM, 2006.

[22] Xingjun Zhang, Cuiping Jing, Feilong Tang, Scott Fowler, Huali Cui, and Xiaoshe Dong. Joint redundant and random network coding for robust video transmission over lossy networks. *Mobile Information Systems*, 8(3):213–230, 2012.

[23] Atsushi Fujimura, Soon Y Oh, and Mario Gerla. Network coding vs. erasure coding: Reliable multicast in ad hoc networks. In *Military Communications Conference, 2008. MILCOM 2008. IEEE*, pages 1–7. IEEE, 2008.

[24] Peter Lambert, Wesley De Neve, Yves Dhondt, and Rik Van de Walle. Flexible macroblock ordering in h. 264/avc. *Journal of Visual Communication and Image Representation*, 17(2):358–375, 2006.

[25] N Qadri, Muhammad Altaf, Martin Fleury, Mohammed Ghanbari, and Hanadi Sammak. Robust video streaming over an urban vanet. In *Wireless and Mobile Computing, Networking and Communications, 2009. WIMOB 2009. IEEE International Conference on*, pages 429–434. IEEE, 2009.

[26] N Qadri, Muhammad Altaf, Martin Fleury, and Mohammed Ghanbari. Robust video communication over an urban vanet. *Mobile Information Systems*, 6(3):259–280, 2010.

[27] Abdul Razzaq and Ahmed Mehaoua. Video transport over vanets: Multi-stream coding with multi-path and network coding. In *Local Computer Networks (LCN), 2010 IEEE 35th Conference on*, pages 32–39. IEEE, 2010.

[28] Gorry Fairhurst and Lloyd Wood. Advice to link designers on link automatic repeat request (arq). 2002.

[29] Min Xing and Lin Cai. Adaptive video streaming with inter-vehicle relay for highway vanet scenario. In *IEEE ICC 2012 Wireless Networks Symposium*, Dept of ECE, University of Victoria, BC, Canada, 2012.

[30] Charles Krasic, Jonathan Walpole, and Wu-chi Feng. Quality-adaptive media streaming by priority drop. In *Proceedings of the 13th international workshop on Network and operating systems support for digital audio and video*, pages 112–121. ACM, 2003.

[31] Hsuan-Fu Ho, Kuochen Wang, and Yi-Ling Hsieh. Resilient video streaming for urban vanets. In *Proc. 7th Workshop on Wireless Ad Hoc and Sensor Networks*, 2011.

[32] Daniel Jiang and Luca Delgrossi. Ieee 802.11 p: Towards an international standard for wireless access in vehicular environments. In *Vehicular Technology Conference, 2008. VTC Spring 2008. IEEE*, pages 2036–2040. IEEE, 2008.

[33] Stephan Eichler. Performance evaluation of the ieee 802.11 p wave communication standard. In *Vehicular Technology Conference, 2007. VTC-2007 Fall. 2007 IEEE 66th*, pages 2199–2203. IEEE, 2007.

[34] Mahdi Asefi, Jon W Mark, and Xuemin Shen. A mobility-aware and quality-driven retransmission limit adaptation scheme for video streaming over vanets. *Wireless Communications, IEEE Transactions on*, 11(5):1817–1827, 2012.

[35] Pablo Piol, A. Torres, O. Lopez, M. Martinez, and Manuel P. Malumbres. Evaluating hevc video delivery in vanet scenarios. In *Wireless Days*, pages 1–6. IEEE, 2013. URL `http://dblp.uni-trier.de/db/conf/wd/wd2013.html#PinolTLMM13`.

[36] Bo Yu and Chengzhong Xu. Vehicular ad-hoc networks: An information-centric perspective. *ZTE Communications*, 8(3), 2010.

[37] H Schulzrinne, S Casner, R Frederick, and V Jacobson. Rfc 3550. *RTP: a transport protocol for real-time applications*, 7, 2003.

[38] Apple. *What is Quicktime 7?* `http://www.apple.com/ca/quicktime/what-is/`.

[39] Y.-K. Wang, R. Even, Huawei Technologies, T. Kristensen, and R. Jesup. Rfc 6184. *RTP Payload Format for H.264 Video*, May 2011.

[40] D. Farinacci, T. Li, S. Hanks, D. Meyer, and P. Traina. Rfc 2784. *Generic Routing Encapsulation (GRE)*, April 2000.

[41] Tim Berners-Lee, Larry Masinter, and Mark McCahill. Rfc 1738: Uniform resource locator. *Internet Engineering Task Force*, 1994.

[42] Google. *URL class Manual*, . `http://developer.android.com/reference/java/net/URL.html`.

[43] H. Schulzrinne and R. Lanphier A. Rao. Rfc 2326. *Real Time Streaming Protocol (RTSP)*, April 1998.

[44] Google. *VideoView class Manual*, . `http://developer.android.com/reference/android/widget/VideoView.html`.

[45] Jon Bentley. *Programming Pearls*. Dorling Kindersley Pvt Ltd, 2006.

[46] Nikolaos Thomos, Nikolaos V Boulgouris, and Michael G Strintzis. Optimized transmission of jpeg2000 streams over wireless channels. *Image Processing, IEEE Transactions on*, 15(1):54–67, 2006.

[47] Xiangjun Li and Jianfei Cai. Robust transmission of jpeg2000 encoded images over packet loss channels. In *Multimedia and Expo, 2007 IEEE International Conference on*, pages 947–950, 2007.

[48] Dan Wing. Rfc 3605. *Symmetric RTP/RTP Control Protocol (RTCP)*, October 2007.

[49] G. Dommety. Rfc 2890. *Key and Sequence Number Extensions to GRE*, September 2000.

[50] Mark Handley, Colin Perkins, and Van Jacobson. Rfc 4566. *SDP: session description protocol*, July 2006.

[51] Hao Jiang, Siyue Chen, Yang Yang, Zhizhong Jie, Henry Leung, Jun Xu, and Lin Wang. Estimation of packet loss rate at wireless link of vanet–rple. In *Wireless Communications Networking and Mobile Computing (WiCOM), 2010 6th International Conference on*, pages 1–5. IEEE, 2010.

[52] Jari Korhonen and Ye Wang. Effect of packet size on loss rate and delay in wireless links. In *Wireless Communications and Networking Conference, 2005 IEEE*, volume 3, pages 1608–1613. IEEE, 2005.

[53] Yung-Cheng Chu and Nen-Fu Huang. Delivering of live video streaming for vehicular communication using peer-to-peer approach. In *2007 Mobile Networking for Vehicular Environments*, pages 1–6. IEEE, 2007.

[54] Zhe Wang and Mahbub Hassan. Blind xor: low-overhead loss recovery for vehicular safety communications. *Vehicular Technology, IEEE Transactions on*, 61(1):35–45, 2012.

[55] Fei Xie, Kien A Hua, Wenjing Wang, and Yao Hua Ho. Performance study of live video streaming over highway vehicular ad hoc networks. In *Vehicular Technology Conference, 2007. VTC-2007 Fall. 2007 IEEE 66th*, pages 2121–2125. IEEE, 2007.

[56] Quan Huynh-Thu and Mohammed Ghanbari. The accuracy of psnr in predicting video quality for different video scenes and frame rates. *Telecommunication Systems*, 49(1): 35–48, 2012.

[57] Alain Hore and Djemel Ziou. Image quality metrics: Psnr vs. ssim. In *Pattern Recognition (ICPR), 2010 20th International Conference on*, pages 2366–2369. IEEE, 2010.

[58] Zhou Wang, Alan C Bovik, and Ligang Lu. Why is image quality assessment so difficult? In *Acoustics, Speech, and Signal Processing (ICASSP), 2002 IEEE International Conference on*, volume 4, pages IV–3313. IEEE, 2002.

[59] Quan Huynh-Thu and Mohammed Ghanbari. Scope of validity of psnr in image/video quality assessment. *Electronics letters*, 44(13):800–801, 2008.

[60] Cisco. *Implementing Tunnels*. `http://www.cisco.com/c/en/us/td/docs/ios/12_4/interface/configuration/guide/inb_tun.html`.

[61] Netpbm. *pnmpsnr User Manual*, March 2001. `http://netpbm.sourceforge.net/doc/pnmpsnr.html`.

[62] Qiben Yan, Ming Li, Zhenyu Yang, Wenjing Lou, and Hongqiang Zhai. Throughput analysis of cooperative mobile content distribution in vehicular network using symbol level network coding. *Selected Areas in Communications, IEEE Journal on*, 30(2): 484–492, 2012.

[63] Kayhan Zrar Ghafoor, Kamalrulnizam Abu Bakar, Zaitul Marlizawati Zainuddin, Chih-Heng Ke, and Alberto J Gonzalez. Reliable video geocasting over vehicular ad hoc networks. *Adhoc & Sensor Wireless Networks*, 15, 2012.