PROACTIVE AD HOC DEVICES FOR RELAYING REAL-TIME VIDEO PACKETS

by

Tien Pham Van

Dissertation submitted to the Faculty of
Computer Science, Electrical Engineering and Mathematics,
the University of Paderborn
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy
2007

Advisory Committee:
Prof. Dr. rer. nat. Franz Josef Rammig, Chair/Advisor
Prof. Dr. math. Friedhelm Meyer auf der Heide, Co-Advisor
Prof. Dr.-Ing. Ulrich Rückert, Co-Advisor

ABSTRACT

Title of dissertation: PROACTIVE AD HOC DEVICES FOR RELAYING REAL-TIME VIDEO PACKETS

Tien Pham Van, MSc.

Dissertation directed by: Professor Franz J. Rammig
Faculty of Computer Science,
Electrical Engineering and Mathematics

Being a set of mobile computing devices connected over wireless links, an ad hoc wireless network is characterized by dynamic changes that affect the communication. Real-time video communication over wireless multi-hop ad hoc networks remains challenging, as video traffic is bursty and highly time-sensitive while network resources are limited and time-varying. This dissertation introduces a novel architecture for intermediate ad hoc nodes in which they are not just passive forwarders, but are aware of video packet semantics and hence actively get involved in the communication. Each node is proposed to reserve a small memory space for transiently caching video packets, so that it can responsively process ARQ requests on behalf of the sender. This will obviously shorten the length of retransmission round, and therefore enhance communication reliability and reduce power consumption as a whole.

As retaining packets, the node is also able to select the most appropriate packets to relay first when channel conditions are unfavorable, aiming at minimizing the playback distortion. In particular, we propose a novel discarding mechanism in which useless packets can be detected and destroyed to save energy and bandwidth. Both theoretical analysis and experimental results collected from a real-life testbed of heterogeneous platforms support our proposed framework, with respect to feasibility and efficiency.

Acknowledgments

This dissertation has been completed as a result of my work in Research Group "Design of Distributed Embedded Systems", Department of Computer Science, Electrical Engineering and Mathematics, University of Paderborn, under the direction of Professor Franz Josef Rammig. Without his encouragement, suggestion, and guidance, the study would never been accomplished. We have gone through countless meetings and discussions that were indispensable in constructing the trajectory of the study. In addition, professionally valuable comments from him on the occasions of "AG Workshops" and other seminars did impulse the development of my work. I am profoundly indebted to Professor Rammig for his tireless supports in all aspects.

Acting as my second and third Supervisors, Professor Friedhelm Meyer auf der Heide and Professor Ulrich Rückert have given me numerous instructive recommendations for which I am deeply grateful to them. I believe that their comments through "Doktorandenkolloquiums" held by the International Graduate School of Dynamic Intelligent Systems did make the mainstream of the study more systematic.

During my time in such a creative and dynamic scientific team as Research Group "Design of Distributed Embedded Systems", I have also received extensive supports from the colleagues. In particular, I would like to express my sincere thanks to Marcelo Götz and Florian Dittmann for their helpful comments, to Gunnar Steinert for sharing his experience concerning Linux kernel, to Johannes for his assistance on the abstract written in the German language, to Vera Kühne for her management work, and to Bodo Blume for his technical assistance.

It goes without saying that I was fortunate to be admitted to the International Graduate School of Dynamic Intelligent Systems, an innovative interdisciplinary institution of the University of Paderborn. For its generous supports, I am deeply thankful to the school, especially the "Graduate School Team".

Last but not least, my love and gratitude at heart go to my wife Bui for whatever she has done and experienced. It was too much for her to suffer during my study away from home.

Dedication

To my wife, Thanh Bui Thi.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

ACK	Acknowledgment
AODV	Ad hoc On-Demand Distance Vector Routing
ARQ	Automatic Repeat-reQuest
B	Bidirectionally-predictive-coded (video frame)
BL	Base Layer
CoDiO	Congestion-Distortion Optimised
CTS	Clear To Send
EL	Enhanced Layer
FEC	Forward Error Correction
GoP	Group of Picture
HDTV	High Definition TV
I	Intra coded (video frame)
IP	Internet Protocol
IPTV	Internet Protocol TV
LC	Layered Coding
MAC	Medium Access Control
MCP	Motion Compensated Prediction
MDMC	Multiple Description Motion Compensation
MPEG	Motion Picture Expert Group
MPT	Multi Path Transport
NACK	Negative Acknowledgment
OLSR	Optimized Link State Routing
P	Predictive-coded (video frame)
PSNR	Peak Signal to Noise Ratio
QoS	Quality of Service
RaDiO	Rate-Distortion Optimised
RED	Random Early Detection
RtME	Real-time Media Engine
RTP	Real Time Protocol
RTS	Request To Send
TCP	Transmission Control Protocol
UDP	User Datagram Protocol

Chapter 1
INTRODUCTION

Communicating solely over wireless channels, ad hoc networks are characterized by restricted and fluctuant resources. Real-time video communication over such networks can be expected in various scenarios, such as disaster rescues, inter-vehicle information exchanges, and military operations. To realize the applications, we need to address numerous problems arising from the nature of wireless multihop connections as well as the dependence on battery power. In this chapter, we first take a glance at real-time video communication over ad hoc networks and its issues. Then, what this study aims at is clarified, considering the gap between QoS (Quality of Service) requirements and the features of ad hoc wireless networks. The chapter ends up with a short summary of the rest of this dissertation.

1.1 Motivation

Real-time video communication has been accounting for major part of traffic flowing over the Internet [1]. Advances in network infrastructure and computer technology nowadays allow people to visually communicate worldwide. Distributed video applications can be found in a wide range, from video-on-demand to live video conference, from video phone to large broadcasting systems, e.g. IPTV [2]. While the applications have been prevailing over the Internet, they still draw attention of the research community. As users are more and more demanding, the main issue is how to efficiently transmit a large volume of time-sensitive data given that packets are discarded from time to time due to lacking of network resources. In reality, transmission of video packets faces numerous issues as it requires stringent QoS specifications on bandwidth and delay. Though various compression techniques have been introduced and developed, networks cannot fully accommodate traffic generated by distributed streaming applications [3][4][5][6][7]. Naturally, video traffic is bursty and bandwidth-greedy, whereas most of multimedia systems rely on connectionless transport protocols, such as RTP/UDP [1][8][9]. As a result, they tend to overwhelm other adaptive traffic classes, such as HTTP. During congestion periods,

discarding packets is therefore inevitable.

Loss or corruption of packets does induce additional distortion to the playback video at the receiver node. While error concealment techniques [10][11] help harmonize the impact, their effect is not unlimited. Loss of some important data segments, such as synchronization data, may seriously disrupt a long sequence of frames [12][13]. Additionally, to maintain an acceptable playback quality, excessive communication delay is not tolerated. For instance, in real-time video, end-to-end delay should not be higher than 500 milliseconds [1][14]. This makes long-distance multihop communications difficult and hinders retransmission of lost packets.

Figure 1.1: Conventional transmission protocol (UDP/IP/IEEE802.11b) gives an unacceptable quality. Sample pictures are taken from MPEG4 CIF sequence *Akiyo* that was transmitted over four IEEE802.11b hops.

An ad hoc network is a collection of autonomous computing devices that communicate with each other over wireless links, such as IEEE 802.11b interface [15]. The key feature of this type of networking is the absence of any permanent infrastructure; a network is able to operate immediately after deployment and is highly flexible. Perspective video communication over such networks can be expected in various scenarios, both in civil and in military activities. However, hard communication conditions due to node movements, interferences, and environmental changes do intensify challenges against delivery of video packets. If hosts employed conventional networking protocols, the perception quality would be unacceptable. Figure 1.1 shows an example video received after traversing a four hop connection.

While video applications generate a huge data volume that is time-sensitive, network resources are not only limited but also unstable due to the nature of wireless and mobile communication. In IP-based infrastructure networks, packet loss occurs mainly in congestion periods during which routers run out of buffer space [4][7]. On the other hand, there are numerous causes of loss in wireless ad hoc networks, e.g. collisions, erroneous channels, and interferences [16][17]. Obstacles in such conditions are highly unpredictable and are hard to deal with, especially when packets traverse multiple hops. Consequently, conventional networking models would not work efficiently if applied into ad hoc wireless communication systems.

As ad hoc networks are mostly deployed for field applications, design of any communication protocol must carefully take into account constraints on energy, memory, and computing resource. This is especially true in the case of streamed video, given that video sources usually generate huge amount of data in a rather long time [1][18]. Practically, compression techniques exploit temporal redundancy, loss of some important packets makes their dependent ones meaningless (see Chapter 2). Transmission of those packets is not only useless to the decoding process, but also induces waste of energy and bandwidth [19][20]. In a resource-limited network, this also potentially leads to rejection of later important packets, and hence magnifies distortion of the received video.

Investigating video-specialized studies on higher layers of the communication protocol stack, we have learnt the following:

- First, all the proposals so far have been focused on ending hosts only, let-

ting intermediate nodes[1] be simple passive forwarders. Proposed strategies, such as retransmission [8][21], cross-layer design [22][23][24], rate adaption [25][26], etc, are essentially sender- and/or receiver-driven. Available results have mainly been reported from simulations in which error rate and delay are largely assumed to comply with conventional distribution laws. Note however that highly dynamic factors in reality might make things considerably different from what they should be [27]. According to [28], given the same network scenario, well-known network simulators: NS2 [29], OPNET Modeler [30], and GloMoSim [31]) output amazingly discrepant results.

- Secondly, while concentrating on perception quality of video, they did not address the issue of fairness among nodes involved in communication. Note that both senders and receivers consume considerable amount of computational resource and energy for video coding and playback [32][33].

- Thirdly, the matter of energy efficiency has never been taken into account in parallel with relieving distortion and impact of packet loss. Known algorithms are solely to reinforce the transmission policy at the sender and/or the feedback strategy at the receiver; power consumption of forwarders is basically not considered [8][17][21][23][25][34]. As said before, detecting and dropping useless packets are worth-doing as we consider saving energy. Intermediate ad hoc nodes in any known proposals are however unable to do this; every incoming packet is forwarded at best-effort, even it has become stale.

To fill the gaps above, it is both demanding and worth to looking into intermediate nodes, making them proactive in the process of relaying video packets. Our study has followed this philosophy, taking the nature of video compression techniques into account.

1.2 Focused issues

Designing a complete model for transmission of real-time video over wireless ad hoc devices is a large problem. It covers a great number of issues, from channel

[1]Throughout this dissertation, phrases "intermediate node", "relaying node", "forwarding node", "forwarder", and sometimes "ad hoc router" can be used interchangeably.

allocation to route selection, from scheduling packets to source coding control, etc. It goes without saying that, from any point of view, the most problematic obstacle is that video traffic is heavy and time-sensitive while effective bandwidth is narrow and changing. Toward realizing the applications, this study concentrates on the packet forwarding strategy at intermediate nodes, rather than ending hosts. What we have attempted to solve includes the following:

- ***Minimization of distortion***. As modern video coding techniques take advantage of correlation among video frames to reduce the data volume, encoded frames are not independent of each other. Packets do not equally contribute to the playback at the receiver. Some packets are more important than the others in the sense that loss of the former voids the latter and hence induces significant distortion. For instance, in MPEG compression standards [12], frames are grouped into units of *group of picture* (GoP). Corruption of the unique intra-coded frame of a GoP means that all its subsequent frames are un-decodable, even they arrived correctly at the receiver. We tried to minimize distortion by introducing a smart packet selection tactic. Namely, once congestion occurs, intermediate nodes are given chances to select the most appropriate packets to forward, based on their importance and remaining *time-to-live*. The selection algorithm has a low complexity compared to other known algorithms proposed for the sending node, e.g. CoDiO [17][23][24]. This is noticeable since reinforcing complicated algorithms in every intermediate node will create considerable delay and cumulative power consumption.

- ***Reduction of energy consumption***. In ad hoc wireless communication, collision may accidentally occur at a receiver node because two transmitting hosts are not mutually visible. This phenomenon is called *hidden terminal*. In another case, called *exposed node*, a node may be unnecessarily prevented from sending packets due to a neighboring transmitter. To overcome these problems, CTS/RTS handshake can be used in the link layer. This mechanism requires a plenty number of messages to exchange upon each video packet [35]. The number of messages is even larger if the acknowledgment mechanism [36][37] in the link layer is employed. In a wireless multi-hop network, once a node send a packet, it may be received at more than one node. Note that both *transmit* and *receive* do consume energy [37][38]. Thus, removing useless

transmits is worth-doing. We also save energy in retransmission of lost packets by reducing the number of traversed hops thanks to the proactiveness of relaying devices.

- ***Lightening of load imposed on the sender***. We know that video processing at the sender is heavy in term of computation and energy consumption [32][33]. In this study, we tried to distribute communication tasks over intermediate nodes. They share the responsibility of retransmission and packet selection as well as enhancing channel adaptability with the sender.

1.3 Contributions

Toward development of real-time video applications over ad hoc wireless network, our study aims at designing a novel architecture for forwarding video packets. Intermediate nodes are able to identify semantics of video packets and hence treat them appropriately, taking both distortion and power consumption into account. This section summarizes contributions of this study in brief.

1.3.1 Proactive node architecture

For the first time, we propose to make relaying devices proactive and video-cooperative. Namely, they are actively getting involved in processing video packets, rather than passively forwarding every incoming packet as conventional routers. Specifically, proactive nodes know which packets should be given higher priority if the communication path cannot accommodate all the traffic; they can also detect and block packets that are known to be useless. To realize the proactiveness, we implemented a middleware called *Real-time Media Engine* that maintains transiently cached packets, selects most appropriate packets to forward, and possibly retransmits lost packets on behalf of the sender.

1.3.2 Packet selection algorithm

As video packets have different roles to the decoding process, selective transmission to minimize distortion has been proposed to ending nodes [13][23][39][40] and infrastructure-based *active networks* [7][41][42][43]. In ad hoc communication

devices, due to resource limitation and highly dynamic factors, all known selection strategies above cannot be applied. Distortion optimization algorithms such as RaDiO (Rate-Distortion Optimisation) [3][6][40] and CoDiO (Congestion-Distortion Optimisation) [17][23][24] are promising for reducing distortion, but they pose heavy computation load; if reinforced in all intermediate nodes they would excessively consume memory and computation resources and enlarge end-to-end delay. Furthermore, unpredictability of channel conditions potentially undermine the expected optimality.

Instead, we design a low-complexity algorithm for selecting packets. In this algorithm, packets are served according to their priority and their remaining time-to-live. The algorithm is flexible in the sense that interoperability of proactive devices and conventional ad hoc machines is guaranteed; its simplicity and variability make forwarding nodes quickly react to channel changes.

1.3.3 Joint discarding

Introduction of the proactive design creates the ability to detect stale packets. Within the framework, intermediate nodes possibly go dropping useless packets once an important packet is rejected. Particularly, in such a case, it also attempts to inform other upstream nodes so that they jointly discard its dependent packets. These behaviors obviously help save bandwidth and energy.

To avoid generating overhead, we do not force intermediate nodes to transmit a separate packet each time an important packet is dropped. Instead, notification information is conveyed over NACK (*negative acknowledgment*) messages that are created and transferred upstream upon occurrences of packet loss.

1.3.4 Implementation of a real-life testbed

Unlike the majority of reported studies on ad hoc networks, we verify our framework via a real-life testbed, rather than simulations. As stated in [27][28], assumption in simulations might not coincide to what happens in reality due to highly dynamic and unpredictable features of wireless ad hoc communications. The authors of [28] recommended that studies investigating higher layers like video streaming should implement testbeds to verify their proposals.

The testbed was deployed in the building of the Heinz Nixdorf Institute, where it suffered interference and noise from the WLAN Access Points, other ad hoc networks, and laptop computers of students. The deployed network was composed of up to 8 nodes, running under heterogeneous platforms.

1.4 Outlines of the thesis

The rest of this dissertation includes seven chapters. This section summarizes what will be presented in each of them. At first, Chapter 2 introduces video communication over ad hoc networks. It brings up an overview of distributed multimedia applications under networking scenarios. The chapter particularly highlights characteristics of video traffic that challenge the deployment of networking applications. Ad hoc networks emerge as a prominent solution for ubiquitous connectivity; multimedia applications over such networks will be prevailing in the next few years. Yet resource limitation in multihop wireless connections is a major obstacle against their development. The chapter insightfully analyzes the issues that the research community must address to realize this promising communication service. Additionally, the chapter also investigates state-of-the-art studies on video communication over ad hoc wireless networks. We extensively survey related works and point out the gaps in fulfilling the research needs regarding realization of the communication service. Main threads to go through include distortion-minimized scheduling, video source code control, error protection strategies, and multipath transport.

Toward filling the gaps above, Chapter 3 proposes a *proactive framework* targeting at ad hoc devices that relay video packets. After briefly showing what the framework means, we analyse the benefits of shortening communication paths, particularly while retransmitting lost packets. Concepts related to caching video packets and the discarding mechanism will also be defined in this chapter. In addition, we discuss both advantageous and open features and the applicable range of the framework from the network development perspective.

Next comes Chapter 4 that presents the design of a *middleware* that realizes the proactiveness for ad hoc forwarders. We specifically describe how video packets are processed thereat. In particular, the chapter shows how packets are transiently cached, freed up, queued, and forwarded. Furthermore, the mechanism of handling

retransmission requests will also be discussed. Video transmission essentially requires broad bandwidth since the traffic is heavy and bursty. As a result, relaying nodes frequently lack time slots for fresh packets and retransmitted ones. This chapter hence addresses the issue of how to select packets in the most appropriate manner so that distortion of the received video is minimized. In the chapter, we also concretely discuss a packet verification mechanism to rule out duplicate transmission as well as to enhance the per-hop based adaptability. Last but not least, the feasibility of the framework is evaluated regarding memory usage and computation load, which supposedly confirms its applicability into today's mobile computers and most of embedded devices.

As resources of ad hoc networks are restricted while video traffic is inherently heavy, avoidance of useless packet transmission is always encouraged. Chapter 5 presents a comprehensive discarding strategy in which relaying nodes can detect and drop meaningless packets. The remarkable feature of the design is that relaying nodes do remove useless packets, not only locally but also jointly. Indeed, thanks to notifications associated with retransmission requests, relaying nodes "tell" each other removals of important video frames so that they may drop useless dependent packets more thoroughly.

Adding the proactiveness to relaying nodes does bring up benefits with regards to quality of the received video and energy consumption, but it also poses extra processing load and memory usage. Chapter 6 specifically describes how the framework is implemented from the coding point of view, but not in a statement-by-statement fashion. We define and implement system functions respecting the node performance in relaying packets. The middleware is expected to fulfill the enhanced forwarding functions while does not induce excessive processing load.

Before concluding the dissertation in Chapter 8, we report experimental results in Chapter 7. Experiments are made with a well-known video sequence on a real-life testbed rather than via simulations. We extensively consider both the quality of video and the performance of relaying nodes. In addition to evaluation of computation load and memory usage, we also analyse the traffic load.

Chapter 2
VIDEO COMMUNICATION OVER AD HOC NETWORKS

Ad hoc wireless networking emerges as an eminent solution for ubiquitous computing mainly because of their flexibility in deployment. This feature makes the networks especially suitable to field applications, among which is distributed real-time video. The needs of developing this communication can be found in a variety of military and civil scenarios, as stated in Section 1.1. Once being deployed, a video ad hoc network can either assist the operators in their commission or comfort them and their work. A large number of examples of the former are in battlefield operations, disaster rescues, and vehicle-safety assurances, while the latter perspective may be expected in less critical cases such as entertaining car drivers.

Unfortunately, while mobility and wireless connections bring up a lot of advantages that promise to fill the gap between the needs in practice and the conventional networking technologies, they also create numerous challenges against the realization of any communication service, not to mention resource-greedy video. In reality, any wireless multihop network can offer only limited bandwidth whereas video traffic is naturally heavy. Movement of nodes, fading, etc... substantially affect the communication performance, making it very uncertain. Meanwhile, video sources usually offer a contiguous load that is intolerant of delay.

This chapter is expected to give an overview of ad hoc wireless communication under the prospect of deployment of real-time video applications. Features of video streaming and ad hoc networking that hinders the development of the applications will then be analysed. In this regard, we will insightfully look into the unfavorable characteristics that challenge against their realization. Finally, we investigate related studies on issues of communications over ad hoc wireless networks and point out the gap that our study attempts to fill up.

2.1 Overview

In spite of fast progress in networking technologies and video compression standards, streaming real-time video still draws attention from the research com-

munity. In the conventional Internet and heterogeneous networks, the main problem is how to maintain an acceptable perception quality while keeping consumption of resources moderate. Considering ending nodes, most of studies are focused on higher layers, trying to build a suitable transmission policy for video packets [13][23][39][40]. These studies basically propose to adjust the generated traffic so that the transmission performance is optimized, given a network with certain QoS specifications. On the other hand, research efforts on intermediate network nodes in general attempt to treat video packets in a different way so that their QoS requirements are satisfied at most. For instance in DiffServ [44], packets of interactive multimedia are given "fast track" when the router serves multiple service classes. This means that they suffer less latency than packets of other best-effort traffic, e.g. FTP, HTTP. Once congestion occurs, video packets are nevertheless subjected to dropping before the others. Another approach is to integrate a multimedia-specific forwarding functionality into active routers [4][7][42], where video packets are treated discriminatively according to their contribution to the decoding process. Namely, packets, whose loss causes serious degradation of the received video, will be given higher priority.

Nowadays, advanced wired access networks allow data speed up to hundred Mbps (such as ADSL, LAN), eliminating most of issues concerning bandwidth at the last hop. However, multimedia networking applications that span over the global Internet still face the problem of resource shortage, despite network infrastructures uninterruptedly expand. The main reason is that the number of users rapidly increases and that they are more and more demanding. At the same time, introduction of broadband services such as IPTV adds more challenges to network designers.

In ad hoc wireless networks, realizing real-time video applications faces even more problems, since communications are made under more unfavorable conditions. Because nodes are mobile, communication links are not robust with respect to bandwidth and error rate. Fading and other environmental impacts further destabilize the communication. Meanwhile, video traffic is inherently contiguous and heavy; packets are highly sensible to delay. These make the development of video communication a challenging task.

2.2 Characteristics of real-time video traffic

Distributed multimedia applications over the network mainly fall into two categories:

- Streaming of pre-encoded video. Video contents are created, encoded, and stored into a central server for later playing back. Network users wishing to access the contents first set up connections to the server and then ask for the desired archives. Video packets are delivered to the users in parallel with the playback. A typical service of this form is video-on-demand, which has been prevailing in the Internet.

- Real-time video. In this type of communication, video packets, after being encoded, are transmitted instantly to the receiver side. Only transient delay (usually less than a second [14]) is allowed; packets suffering excessive delay would be considered lost as the communication is in real-time. Consequently, requirements on QoS is even more stringent. Typical examples of these applications are broadcasting and interactive video (e.g. video conference). Our study has concentrated on this category of communication.

The major challenge in designing the network system is how to stream packets smoothly so that the playback does not suffer interrupt and loss of too many video frames. As previously stated, packet loss is unavoidable since the traffic is heavy and intolerant of delay whereas any practical network is not always available to serve any packet. In reality, the best-effort nature of the network does not guarantee that any packet correctly arrives at the destination. To relieve the effect of packet loss, we need to understand the nature of video packetization.

2.2.1 Data-dependency

Most of modern video compression techniques exploit both temporal and spatial redundancy to greatly reduce data volume. As a result, frames are not equally important with respect to the reconstruction process at receiver side [12]. Namely, in the encoding process at the sender, a dependent video frame is predicted from a previous frame, and then the prediction error is actually coded. This well-known technique is called *motion compensated prediction* (MCP), which is employed in

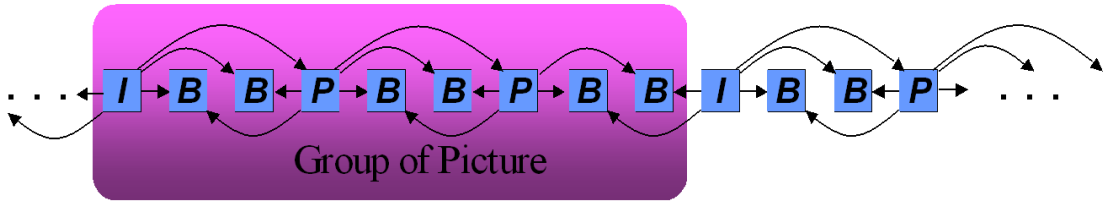

Figure 2.1: Frame dependency in MPEG-4. Arrow curves originated from a frame point to its dependent ones.

most of modern video compression standards, particularly in the well-known standards MPEG. Obviously, it helps decrease the data volume since visual contents of adjacent video frames are highly correlated [12]. However, MCP also creates *inter-frame dependency*. For example in MPEG4 - a very popular video coding standard series, each frame can be encoded according to one of three modes: intra-coded (I-frame), predictive-coded (P-frame), or bidirectionally-predictive-coded (B-frame). In essence, a B frame cannot be decoded without its adjacent P frames, likewise a P frame must refer to the previous one, and all subsequent frames of a GoP (Group of Picture) are considered useless if its I frame is lost, as illustrated in Figure 2.1. Occurrence of error or loss of a packet may result in error propagation to other frames, even they are correctly received [10][11][45]. As a result, video packets are substantially different in their contribution to the reconstruction of the received video.

When the encoded video is packetized for delivering over the network, the application layer may add additional header to each packet to indicate how significant the data are, before the packet is sent down to the underlying layers (e.g. RTP/UDP). In conventional networking devices (e.g. IP-based ones), however, network routers do not understand this information. As a packet arrives at a router, it simply finds the destination network address and then consults a routing table to decide which hop to forward. In other words, routers are blind to frame semantics of packets. Namely, they do not know that packets of I frames contribute more than the other with regard to the playback process. Should congestion occur at some node, incoming packets may be rejected in accordance with a probabilistic rule, for example, RED (Random Early Detection) algorithm [46][47]. In this algorithm and its modified versions [48][49], acceptance of arriving packets is basically controlled by observing the buffer level at the router, as shown in Figure 2.2. Packets may

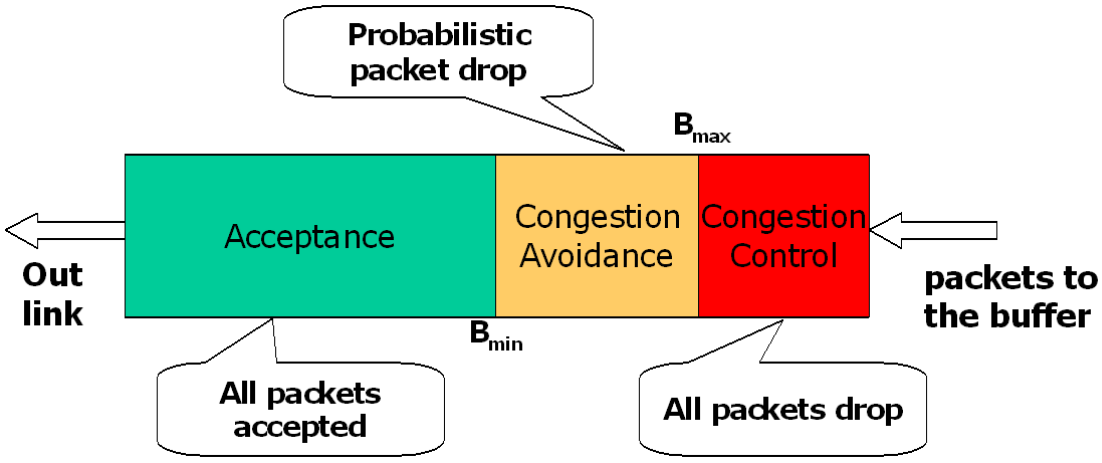

Figure 2.2: RED: discarding packets based on buffer levels. Three regions separated by B_{min} and B_{max} determine the admission of arriving packets.

be partially or entirely rejected if the the *average queue length* [47] exceeds the predefined levels, regardless of how important they are. There are three regions of treating packets divided by two thresholds B_{min} and B_{max}: acceptance, congestion avoidance, and congestion control. In the first region, all packets are admitted as long as the average queue length is below B_{min}. Should the average queue length get higher, they are probabilistically discarded until the critical level B_{max}. Above this level, all incoming packets will be rejected. As being unaware of frame semantics, packets are dropped without taking care of frame semantics, degradation during congestion is therefore more significant.

Obviously, distortion during playback does not linearly depend on communication loss rate. To relieve distortion caused by packet loss, video frames should be treated in a discriminative fashion according to their contribution to the decoding process. Specifically, packets of I frames should have the highest priority, then come those of P frames, with regard to admission control. This strategy has been proposed to active network nodes [4][7][42] as previously stated in Section 2.1, where more functions can be flexibly added and programmed. In the meantime, existing programmable networking devices such as wireless base stations, proxies, etc, may also employ a semantics-oriented scheduling policy [4][50].

It goes without saying that inter-frame dependency adds more difficulty to any optimization aiming at enhancing the perception quality. Once being unable

to accommodate all the incoming packets, the node must selectively discard some of them; selection must be made so that no useless packet is chosen while keeping bandwidth fully utilized. Algorithms such as RaDiO [51] propose to choose packets/frames for transmitting so that the rate-distortion *lagrangian function* is minimized. These algorithms, however, basically have a complexity that is exponentially proportional to the number of packets under consideration, which is not bearable to such limited resource as ad hoc nodes. In fact, related studies consider using them at ending nodes only [17][23].

Under the impact of changing factors, quality of wireless channels fluctuates drastically over time, from good to bad and vice versa [27][52]. In multi-hop wireless connections, the propagation is even more random, making end-to-end conditions very unpredictable and time-varying. On the other hand, selection of a pattern of packets is made on the time base of group of picture [4][50]. Within the time of a group of picture, if one important packet that has been selected to forward is suddenly lost, the expected optimality may seriously be damaged.

2.2.2 Bandwidth greediness

Communication of raw video is unaffordable because of its huge data volume. Numerous compression standards have been developed for networking needs, of which MPEG4 and its subset H.264/AVC are the most remarkable. These codecs lower a video possibly down to less than a hundred Kbps [12][18] while allowing users to manipulate individual objects (see Table 2.1). Even data speed is drastically reduced, it is still unrealistic for the network to convey every packet successfully [1].

Unlike other data applications, during a session, multimedia traffic is generated continuously. Huge traffic volume usually encourages system designers to prefer connectionless transport protocols like UDP and its variants. In this manner, video packets are sent out without any flow controlling mechanism. This helps cut off communication overhead and potentially reduces end-to-end latency thanks to the simplicity of the protocols. Note that no retransmission is executed. However, the strategy also makes the traffic non-adaptive to the network.

Practically, multimedia applications accept loss of packets to some extent. As such, it is admissible that they rely on UDP-based transport protocols. The ap-

Table 2.1: Data speed of different video coding schemes. Most of the data can be found in [18]

Standard	**Data speed**	**Possible application**
MPEG2	1.5 to 15 Mbps	DVD systems, video on demand, IPTV
MPEG4	64 Kbps to 300 Mbps	graphic with object oriented interactivity and multimedia networking
H.263	20 to 500 Kbps	multimedia networking
H.264	20 Kbps~	mainly multimedia networking

plication of these protocols helps avoid excessive delay and overhead, but it also creates obstacles in making the traffic adaptable to the channel instability. Without appropriate control measures, a video tends to grab the available bandwidth; once the network lacks resource, important packets may be rejected while useless packets are served. This would result in severe distortion at playback. Consequently, various schemes for tailoring its traffic, including source coding control [21][22][25][26] and adaptive packet scheduling [53][54], have been studied. In conventional wired networks or single-hop wireless communications, video packets should be selectively discarded depending on their semantics if congestion occurs [14][55][56].

In ad hoc networks, channel capacity is substantially limited, making the issue more significant. According to [25][26], as the number of hops increases, the effective bandwidth quickly gets smaller. The need of introducing the above controls is therefore magnified.

2.2.3 Variable bit rate

Unlike channel capacity of conventional switching networks, video data speed is highly time-varying. This is mainly due to the application of compression to reduce bit rate. Frames with more motion of objects will create more data than static scenes. Rate variation is also caused by coding configuration. In channel-adaptive streaming, source coding is controlled to reduce unexpected loss during communication. As previously stated in Section 2.2.1, compression techniques exploit temporal redundancy, intra coded frames usually induce more data than predicted frames.

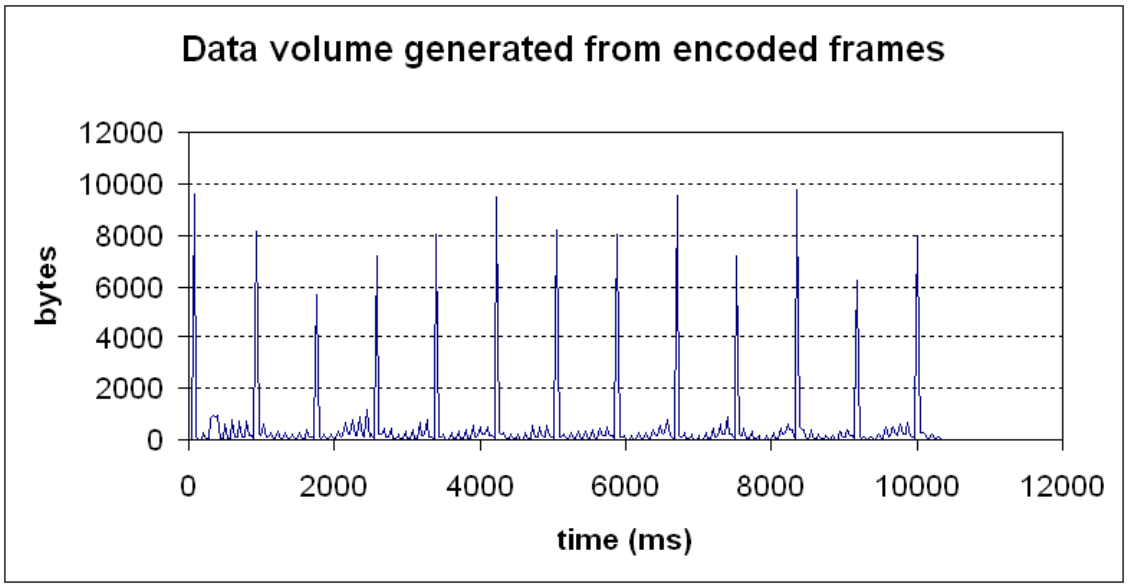

Figure 2.3: Traffic sample of *Akiyo* sequence. The size of I frames (ones that regularly emerge) is by far larger than that of frames of other types.

Figure 2.3 shows an example of the traffic pattern of MPEG4 encoded *Akiyo* video, where an I frame may be as large as several Kbytes while a B frame can be as small as some ten bytes.

To utilize bandwidth efficiently, an adaptive rate control is mandatory at the sender. This can be assisted by cooperations of the network and the receiver. In [17][23], the authors propose an adaptive packet scheduling in which the sender selects packets to send by considering end-to-end delay. Another way is to control video coding parameters so that the compressed rate is adjusted [25][26]. In this study, we also propose the use of sending buffer as a supplemental way to smooth the transmission of video packets.

2.2.4 Time-sensitiveness

Video applications, to some extent, accept loss rate of packets, while the requirement on delay is strict. Packets need to arrive at the receiver punctually or they would be considered lost. In multi-hop connections, communication delay should not be larger than a threshold, typically of 500 milliseconds [1][14]. If communication delay approaches to the threshold, delay variance (jitter) also affects the perception quality. To comfort the perception, jitter can somewhat be absorbed by

using buffers at the receiver [57].

Once congestion occurs, packets suffer more latency, and the receiver likely detects more lost packets. This would restrict the retransmission process as an effective remedy to loss. Specifically, because the network resources (e.g. bandwidth, buffer, etc) are not enough to convey the video traffic, more packets are dropped, raising the number of ARQ requests. Lack of the communication resources will in turn limit the ability to response to these requests. At the same time, the average round trip time gets increased, reducing the number of possible retransmission times for each particular packet. Consequently, retransmission in resource-limited wireless networks should carefully be designed to avoid the deadlock.

2.3 Ad hoc wireless networking

An ad hoc network is formed from a set of nodes communicating usually without assistance of any fixed coordinating infrastructure (e.g. base stations in wireless cellular networks, digital exchanges in public telephone networks). All nodes operate essentially under the same hierarchical level. They can be either homogeneous or heterogeneous in hardware and/or software configurations, and be mobile in most of cases. A node can be a portable device such as PDA, a laptop computer, or an embedded control unit attached to a vehicle, helicopter, etc. Being considered as the base for "anywhere" and "anytime" connectivity and applications, it is a hot topic of interdisciplinary research, from low power design to signal processing, from mobility modeling and management to deployment of real-time applications. As the communication environment is highly dynamic and unpredictable, designs targeting at conventional wired networks basically do not work efficiently. Furthermore, communication links in those networks are reliable, higher layer functionalities, such as routing protocols, can be studied independently of physical behaviors. On the other hand, unstable wireless connections desire considerations of cross-layer interactivity for resilience. Layers may share instant information about capacity, error, delay, energy, etc., to decide their most suitable behaviors [21][23][25].

Wherever the network is deployed and no matter how powerful the nodes are, it has common disadvantageous features. As communication links are solely wireless (e.g. 802.11b, Bluetooth), the network performance is essentially affected by

environmental factors, e.g. noise, fading, etc. Furthermore, node mobility and interference are also major obstacles to deployment of any service that desires reliability and/or instantness in communication.

2.4 Real-time video over ad hoc wireless networks

As wireless links are inherently unreliable, guaranteeing hard QoS specifications for video is unrealistic. For real-time traffic that accepts loss to some extent, the network should deliver packets in best-effort fashion. This section discusses obstacles in transmission of video packets over such resource-limited networks.

2.4.1 Capacity scantiness

In modern wired media such as optic fiber, 100BaseT cable, the link error rate is extremely low and effective bandwidth is predeterministic according to physical specifications and signal modulation methods employed. In contrast, capacity of wireless channels is tightly dependent on distance between nodes, fading, interference, etc. These factors adversely affect the signal-to-noise ratio (SNR) at the receiver node, which in turn curtails the actual bandwidth.

When the communication path consists of multiple hops, the issue is even more serious. As mentioned previously, the end-to-end capacity gets decreased if packets traverse more hops. In [25] for instance, as the number of hops (2Mbps 802.11b WLAN) increases from one to six, the throughput consistently decreases from around 1000Kbps down to 200Kbps. If the nodes are far away from each other, the effective bandwidth is even narrower since the propagated signal is more attenuated.

While end-to-end bandwidth is limited, a video source usually generates high data speed, regardless of compression. Compared to other popular applications such as HTTP, each session of real-time video is usually longer [1]. Consequently, rejection of video packets, particularly during bad periods, is inevitable. To our best knowledge, so far, no investigation on how to effectively control packet admission at ad hoc relaying nodes is reported. In this study, we fill the gap by proposing a low-complexity algorithm to selectively drop packets in case of congestion (see Chapter 4).

2.4.2 Energy constraint

In ad hoc wireless networks, power is a serious factor to consider as design of mobile devices must respect requirements on portability, weight, and size. Deployed for field applications, machines mostly run on battery power, how to save energy is therefore not a negligible question. Unfortunately, the issue of energy in relaying nodes has never been addressed specifically for video taking frame semantics into account, though studies target at generic communication services can be found in most of layers, particularly in the MAC protocol [38][58].

Exhaustion of battery in any node may cause a failure of an active route, or even a disconnection of the network. To insure the network connectivity, each node must transmit (or relay) the signal at a strong-enough level so that the adjacent nodes can receive packets correctly. On the other hand, the signal level should be also as low as possible to save energy for the node itself, and to reduce interference. As designed in the MAC protocol, multiple nodes may share the same channel, limiting the transmitting power helps raise the number of transmission simultaneously. The stronger the signal, the more likely collisions take place, which results in more retransmissions and extra energy consumed [59][60].

In any ad hoc node, power consumers are CPU, HDD, wireless network cards, etc. Consumption can be bound to either computation or communication tasks. High-complexity algorithms do not only occupy more computation resource, but also incur more energy. Additionally, video packets are time-sensitive, complicated algorithms reinforced at multiple intermediate nodes potentially do harm to the video. According to [37][38], both transmitting and receiving packets do consume much more power than idle states. Consequently, dropping useless packets as soon as possible is always worth-doing.

2.4.3 Instability

Wireless link performance tightly depends on the distance between the nodes. We know that SNR determines the error rate; when the nodes are too far away from each other, the error rate may be unacceptable. In reality, ad hoc nodes are not bound to fixed positions, the channel quality is consequently unstable and time-varying. Some application fields such as military battles may contain very

fast-moving vehicles (e.g. tanks, helicopters), the network topology accordingly changes frequently. Topology changes may also happen when new nodes are added or existing nodes are disconnected. Fast movement of nodes also forces the design of physical transceivers to consider the *Doppler* shift carefully [61].

Besides common types of interference in wireless communication (adjacent channel interference and co-channel interference), an ad hoc network also suffer interference due to sharing medium among multiple nodes. The denser the network, the clearer the impact of the interference [62]. When multiple nodes sharing the same channel transmit at the same time without power control, the consequence will be significant if nodes are close to each other. The interference obviously reduces SNR and therefore decreases the effective bandwidth. As video flows are inherently bursty and heavy, the effect of interference is potentially amplified, intensifying error rate and delay to packets.

Another major obstacle coming from the transmission medium is fading, which is highly random and environment-dependent. *Fast fading* (caused by the interference of multiple versions of the same signal [63]), following either Rayleigh or Ricean distribution [64], may sometimes deteriorate the channel performance. Operating in areas with complicated terrain, ad hoc nodes also cope with *slow fading*, which is attributed to objects lying between the sender and the receiver, such as walls of building, vehicles, etc. The effect of this fading usually lasts multiple seconds and may interrupt the playback at the receiver.

The aforementioned factors, whether individually or jointly, result in fluctuation of communication reliability, delay, and bandwidth, both on per-hop and end-to-end bases. This adversely affects real-time traffic and desires a highly dynamic design for each forwarding node, not only the sender and the receiver. So far, related studies aiming at adaptability merely consider ending nodes, treating relaying nodes along the path as a "black" cluster. Should one or multiple links significantly degrade or completely fail, the routing protocol re-routes video traffic over another path, partially or completely different from the previous one. Such re-routing discontinues the communication in a considerable time [65], which does matter in real-time video. This issue will be further mentioned in Chapter 3.

2.4.4 Error-proneness

Communicating solely over wireless channels, ad hoc nodes suffer high loss rate compared to wireline transmission systems. Moreover, the error rate is time-varying and unpredictable due to changing factors stated previously. Practically, the video playback accepts erroneous and lost packets, especially when error concealment techniques is exploited. However, severe loss rate will dissatisfy the user. Note that, loss of important packets causes error propagation to many frames, or even disrupts the media synchronization [45].

As massive erroneousness characterizes wireless communication, various error-resilience techniques have been proposed, throughout the protocol stack from the bitstream level to the application layer. Most remarkable video-specific proposals relate to forward error correction (FEC) [21][25][66], retransmission of lost data units in the link level, or packet-level redundancy [8]. Introduction of FEC means that more data are added to packets, which helps the receiver node detect and correct errors. In the second strategy, retransmission resolves errors at link level, rather than end-to-end. Differently, the last technique proposes to transmit multiple instances of a packet to enhance the chance of success. Actually, these techniques have been seen more or less in conventional communication systems. They are tailored to lossy and dynamic channels that features mobile ad hoc networks. While these tactics supposedly combat with communication errors, they also induce more overhead and are possibly computationally expensive. Careful consideration should be taken into account since video traffic is inherently heavy, otherwise adverse consequences may be observed. The next section further analyses these schemes.

Under adverse impact of fading, node mobility, etc., video communication may suffer burst errors/loss which severely damage video frames continuously in a considerably long period. During our experiments, we observed from time to time that a large number of packets pertaining to some episodes of *Akiyo* were missing, which lasted several seconds (see Chapter 7). Should techniques of transmission diversity (e.g. frequency, time, polarization) be exploited, the effect can be relieved. Note however that in ad hoc wireless networks where a large number of nodes communicate, these resources are also scanty. Once traffic is distributed over multiple disjoint paths concurrently, the probability that burst errors occur in all the paths at the same time will be obviously lessened. Fortunately, most ad hoc routing protocols

inherently bring multiple routes to each pair of nodes [8][67].

2.4.5 Fairness for sender

Reported research works in the field of video over ad hoc networks largely rely on simulations to evaluate their proposals, letting alone the issue of heavy workload imposed on the sender. In reality, as this node takes over the encoding of the video, it consumes much computation resource and energy [25][32][33]. Particularly, complicated algorithms aiming at optimizing the transmission policy under network resource constraints do not only produce extra computation load but also desire more memory [68]. Unfortunately, all the proposals specifically to temporal-dependency video solely aim at this already-busy node.

Furthermore, techniques to combat end-to-end errors, such as video-specific FEC and retransmission as mentioned previously, are usually reinforced at the sender with or without the cooperation of the receiver. This obviously adds further work to the node, especially during bad communication periods.

2.5 Related work

As soon as ad hoc wireless networks emerged as a hot solution for ubiquitous computing, transmission of real-time video over such networks has been attracting much attention from the research community. This section makes a survey on related studies on the topic.

2.5.1 QoS-supporting in lower layers

Each class of traffic over the network has specific QoS requirements. Non-real-time applications, for example, HTTP, FTTP, and distributed database, usually require a hard assurance on data integrity but tolerate delay. These applications solely rely on reliable transport protocols (e.g. TCP) for delivering packets. To deal with packet loss and errors, the underlying transport protocol employs retransmission. On the other hand, real-time applications like video accept errors and loss to some extent, but are sensible to latency. As stated in various studies (e.g. [14][21]), allowable end-to-end delay is only a few hundred milliseconds. At conventional advanced routers, for instance in DifferServ, packets may be classified according to

their time-sensitiveness. Interactive multimedia packets should be scheduled for transmitting ahead of other non real-time ones. Another way is to use a signaling mechanism to reserve bandwidth before the communication session starts, an example of this approach is RSVP (Resource Reservation Protocol) [69].

Ad hoc networks are composed of autonomous computers that are "loosely" connected together over wireless connections that are inherently unreliable and fluctuated, firm assurance of QoS specifications is therefore unrealistic. Routing protocols, while determining routes for hosts, are subjected to various constraints, e.g. energy, can only provide routes that are good to some metrics but not the others. Note that requirements on QoS are different from application to application. For instance, in military operations, security and reliability of hosts are the prime parameters [16]. In sensor networks, energy conservation is the most important.

2.5.1.1 MAC protocols with resource reservation

In ad hoc communication, multiple nodes share a common broadcast radio channel with a limited spectrum. Protocols controlling access to the shared medium (i.e. MAC) need to insure fair share among communication nodes. They should also maximize bandwidth efficiency together with addressing specific issues such as *hidden* and *exposed* terminal problems [35][70]. Node mobility, interference, and fading usually adversely affect the communication performance, and make available resources unpredictable. It is generally difficult for MAC protocols to guarantee QoS requirements in an ad hoc wireless network, given that resources are so constrained.

On supporting real-time applications, including distributed multimedia, several protocols have been proposed [71][72]. In these protocols, after the bandwidth reservation phase, the nodes get exclusive access to the bandwidth [16]. Note however that, reservation does not mean that each node will be allocated a satisfactory bandwidth. Unfavorable factors associated with wireless mobile communications may alter the effective bandwidth that a specific node gains. For example, when the channel is erroneous, massive retransmission per hop-basis in the link level definitely lowers the actual throughput [27]. Furthermore, contention may occur during the reservation phase itself, making the bandwidth assignment indefinite. Lack of centralized coordinators adds further difficulty to the QoS provisioning task.

2.5.1.2 QoS routing

From the network point of view, the goal of routing protocols is to find out the route(s) to forward packets. Guaranteeing hard QoS specifications is basically unrealistic in ad hoc multihop networks, since wireless links are highly unstable and unreliably. Furthermore, available state information is inherently imprecise [73]. Instead, routing protocols can only search for temporarily good paths with respect to some aspects, e.g. reliability, delay, bandwidth, etc. Routes recommended by different routing protocols are not the same, since they consider QoS parameters differently [74]. In case of distributed multimedia applications, the key parameters are bandwidth and delay. QoS routing protocols supporting video transmission should accordingly provide routes with largest end-to-end bandwidth and least delay.

In practice, ad hoc routing protocols are classified into proactive, reactive, and hybrid [16]. As inter-arrival between two consecutive packets is rather short while they are time-sensitive, ad hoc nodes that relay video packet should prefer proactive routing protocols to maintain routes. This is to insure that packets do not have to wait for route discovery when they arrive at a node.

2.5.2 Distortion minimized schedule

On minimizing distortion, several scheduling schemes at the sender have been proposed. For instance, the authors of [17][23] introduced CoDiO (Congestion Distortion Optimization), which is a variant of RaDiO. This cross-layer approach attempts to prescribe a transmission schedule so that a *lagrangian function* of distortion and end-to-end delay is minimized. Namely:

$$minimize : D + \lambda\Delta \tag{2.1}$$

$$subject\ to : R < b_{max} \tag{2.2}$$

where D denotes the distortion of the received video stream; Δ is the end-to-end delay which serves as the congestion metric; λ is a factor; R is the rate of the video stream that is limited to the channel rate b_{max}. Distortion D is formulated as a sum of encoding loss and transmission loss. The former is modeled according to the rate-distortion relation of the video. In essence, this term is independent of the network status. On the other hand, the latter is assumed to perfectly match a

well-known distribution, and is dependent on the network delay and the available bandwidth. Link delay, according to [17], is also assumed to comply with the exponential distributions. While simplifying the model, these assumptions may raise a question on accuracy, since behaviors of multihop wireless connections are very unpredictable [27]. Estimation of end-to-end delay Δ is made by considering only the bottleneck point, ignoring the queuing time in other nodes. The path capacity is predicted based on a TCP-friendly rate control mechanism, which allows to collect statistics from observing acknowledgments.

With the above assumptions, simulation results shows a clear improvement with respect to PSNR (Peak Signal to Noise Ratio) [23]. So far, no experimental result on a real-life ad hoc network testbed realizing this approach has ever been reported. In wireless multihop ad hoc connections, as behavior of each hop is affected by numerous dynamic factors such as fading and movement of nodes, end-to-end metrics are notoriously uncertain [16]. Unplanned dropping of an important packet due to some instant channel degradation may be harmful to the optimality that the prescribed schedule expects, since transmission of the dependent packets is no longer necessary. Though not explicitly analysed, the CoDiO algorithm has a complexity comparable to RaDiO, which was proved to be heavy [3][4]. Furthermore, determination of distortion resulting from frame substitution is also rather complicated, since it is related to the estimation of the delay distribution function [24]. Consequently, while this scheduling approach promises a high performance on perception quality if applied to the ending hosts, it is not efficient to introduce to multiple relaying nodes. Augmentation of high-complexity algorithms in forwarding nodes will add considerable delay and power consumption. This should be avoided since real-time video packets are time-sensitive [7].

2.5.3 Source coding control

As ad hoc network communication conditions are time-varying, video traffic flowing over the network should be tailored to the channel conditions so that resources are used up. The issue of rate adaptation has been extensively studied [21][25][53].

2.5.3.1 Cross layer feedback

This approach basically relies on a cross-layer design where the video source coding exploits information from the routing layer to control the rate. In [25][26], the authors proposed a feedback mechanism in which the application layer adjusts the video rate in accordance with the path status. On implementation, the feedback information from the underlying protocol (e.g. AODV or OLSR) is simply to indicate the hop count of the route that conveys video packets. Based on this information, the sender tunes the quantization parameter (QP) on the source coding to adapt the traffic rate to the path. Obviously, predicting bandwidth simply from the hop count may not be accurate, since channel performance is also affected by other factors such as distance between nodes. Inaccuracy may result in underutilization of channel resource or bursts of packet dropping.

In those studies, error combating techniques were combined with the rate control so that effect of loss of important packets is lightened. For instance in [25], experiments on RTP/UDP/IP streaming were deployed, testing the efficiency of FEC and packet redundancy strategies. The measured statistics show that when the hop-count increases, redundant transmission of P frames outperforms FEC. Additionally, with this cross-layer control, not only packet loss rate is decreased, but it also remains unaffected by change of hop-count, as reported in [26].

2.5.3.2 Multistream coding

As presented in [21][34], a video flow can be encoded into multiple substreams. These substreams are then transmitted over multiple paths that the routing protocol brings up. The authors examined two ways to decompose the video flow: a layered coding (LC) with selective ARQ scheme and multiple description motion compensation coding scheme (MDMC). The former technique naturally generates multiple streams according to packet semantics, whereas the latter employs multiple description coding (MDC) to create equally important substreams.

In the LC coding, video packets are typically divided into base layer (BL) and enhancement layer (EL). Basically, BL contains the crucial part of video frames (e.g. I, P frames) while EL is composed of less important packets (e.g. of B frames). Successfully receiving packets of BL layer guarantees a minimum playback quality

that is improved if the receiver additionally gets packets over EL. Such a decomposition gives scalability with respect to variable bandwidth. In multipath transport (MPT) mode, those streams are separately transmitted to different paths. To decode packets of EL successfully, BL stream must be correctly received.

Differently, MDMC generates multiple substreams that are relatively independent. Loss in one substream does not affect the others. Perception quality of the received video is improved in accordance with the number of correctly received substreams. To insure the independence, the sender must insert sufficient overhead to each substream so that it is decodable without referring to any other substream. This obviously reduces the overall coding efficiency [8]. Additionally, the applicability is bound to the coding method. In [34], the authors propose to use *genetic algorithms* (GA) for solving complex cross-layer optimisation problems of routing. Simulation results show that the GA solutions do improve PSNR compared to trajectory and network-centric methods.

2.5.4 Error protection for video packets

As wireless channels are lossy, several error protection schemes applied specifically to video applications have been studied. Basically, they fall into two categories: redundant transmission and forward error protection (FEC). While these techniques lighten the effect of errors, they also introduce considerable overhead. Thus, the tradeoff between the ability of loss recovery and channel efficiency should be carefully considered; otherwise, its effect might be opposite to what we expect [8].

Conventional coding schemes that support FEC have been extensively studied. Nevertheless, most of them aim at lower layers, dealing with universal traffic, rather than being video-specific. Well-known error protection techniques include convolutional code [75] and CRC (Cyclic Redundancy Code)[76]. Since error rate over a wireless channel changes continuously, the coding scheme should be adaptive. For instance, in [66], the authors proposed a technique called AFECCC (Adaptive FEC Code Control) in which the amount of FEC code per packet is tuned based on acknowledgment packets. Here the channel status is implicitly indicated by arrivals of the acknowledgment packets rather than specific information such as SNR or BER (bit error rate) from the receiver.

At the application layer, video-specific error protection coding schemes are designed in [22] and in [25]. The authors of [25] investigated the effect of FEC under the throughput context. Namely, the overhead amount for error recovery is introduced by considering the channel capacity, which is implicitly expressed as the hop count. The study also proposes to use a combination of FEC and packet redundant strategy to combat with errors. On the other hand, the former study introduces a *systematic lossy error protection* (SLEP) scheme in which the transmitted video signal goes with error protection information that contains a bitstream derived from the Wyner-Ziv encoding. The Wyner-Ziv description can be optimized based on an end-to-end video distortion model and channel information, which brings up a graceful degradation over a wide range of packet loss.

In parallel to FEC, enhancing the quality of video by applying the ARQ technique has also been considered. Generally, the network should retransmit lost frames that are important since they strongly influence the playback process [21][34]. As described in Section 2.2.1, loss of these frames makes receipt of many others meaningless. In [8][21], retransmission of I/P frames are made over multiple paths to increase the reliability. The efficiency is clear when the end-to-end loss is severe. In discriminating video frames according to their types, the authors of [77] introduced an error control scheme called Differentiated Automatic Repeat reQuest (DARQ). The maximum number of retransmission attempts assigned to a frame is calculated taking into account both the attribute of the frame and its location in the group of picture.

2.6 Chapter review

Heavy and contiguous traffic creates difficulties in deploying any networking video application, despite compression techniques such as MPEG standards greatly reduce its data volume. Exploitation of temporal redundancy in video compression techniques induces video data dependence, which is another challenge to any transmission policy. Being time-sensitive, video packets cannot be conveyed over heavy overheaded transport protocol such as the conventional TCP. Deployment of real-time video over ad hoc wireless networks is even harder, since they are not only limited in resources, but are also highly unstable. Before proposing our own framework

in the next chapter, we have insightfully investigated related studies, highlighting the need to enhance the forwarding functionality of intermediate nodes.

Chapter 3
PROACTIVE FRAMEWORK

Unlike other proposed strategies that solely focus on ending hosts, this study considers the ability to enhance the functionality of intermediate nodes so that they efficiently relay video packets. This chapter considers the motivation to introduce proactiveness to the nodes. The expected benefits will be analysed from the system point of view. Additionally, the limitation of the framework is also clarified.

3.1 Overview

Let's consider an instant transmission path for real-time video from the sender to the receiver, as depicted in Figure 3.1. The video sequence is encoded, and then packetized before being sent over the intermediate nodes toward the receiver. At the receiver side, the packets are reordered, and eventually decoded into the original frames for playing back. Any loss of packets implicates additional distortion caused to the playback process; if there are dependent frames of the frame with lost packet(s), error propagation to these frames will occur as stated in the previous chapter (see Section 2.2.1). All the known proposals from the research community so far have been focused on ending nodes only, letting intermediate nodes be simple passive forwarders. Investigated transmission strategies aiming at video are essentially sender- and/or receiver-driven [17][21][23]. We observe the following at any relaying node.

- Every packet is instantly routed to the next hop based on its destination address. The hop is determined by consulting the routing table maintained at the node.
- Packets of different frame types are treated equally with respect to dropping in case of shortage of resources.
- Each intermediate node passively receives packets and then forwards them at best effort. It cannot know whether the packets are "fresh" or already miss deadline.

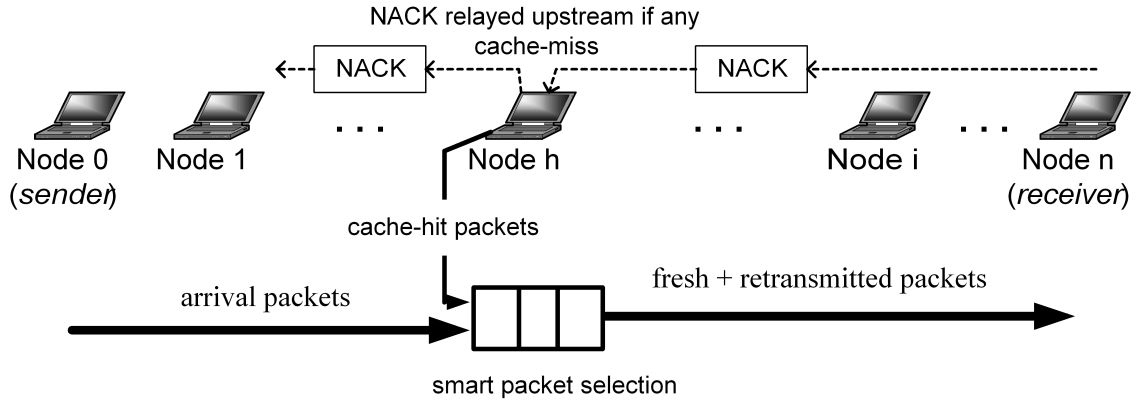

Figure 3.1: Proactive forwarding. An intermediate node h transiently "caches" video packets so that it can respond to later possible ARQ requests carried by NACK messages. Caching packets also allows to smartly forward when the channel is not good enough.

- Furthermore, it always redirects toward the sender any ARQ (Automatic Repeat-reQuest) request, which asks for retransmission of a lost packet. Even the request is for the packet that has just arrived at the node right before, it is not handled.

In brief, intermediate nodes are blind to any strategy for loss relief. Each intermediate node simply acts as a passive forwarder in which they cannot make any cooperation with the video-specific relaying process. There is no way to give higher priority to packets of important frames since relaying nodes are unaware of video frame types. Particularly, as forwarding packets at best effort, they do not care about any timing information concerning the playback deadline. Consequently, packets that are already stale may also be unawarely processed and forwarded. Namely, no admission control mechanism is forced to filter out those useless packets. In other words, intermediate nodes stay away from any strategy dealing with packet loss.

As presented in Section 2.4, any ad hoc network has rather limited and unreliable computation and communication resources while video traffic is bursty. This results in unavoidable packet loss that occurs frequently. To lighten the effect of error and loss, retransmission needs to be reinforced. At the receiving side, loss of packets can be detected from the gap of sequence numbers. Should the receiver wish

to (re-)obtain lost packets, it issues a NACK (Negative Acknowledgment) message to carry ARQ requests for those packets. Then, the node attempts to communicate this message to the sending side. Because all the forwarders are not cooperative, the NACK messages has to reach the very sender (node 0) to be processed. This mechanism is called *passive retransmission*, as seen in Figure 3.2.

Intuitively, there exists a link between packet semantics-unawareness and passiveness of the forwarders and the degradation of the whole performance. If, obviously, a forwarder understood packet-semantics, it would give higher priority to packets of I frames than to those of B frames. It would be even better if a packet could be retained at the node in a very short while so that it was able to actively respond, on behalf on the sender, to later possible ARQ requests. This would greatly shorten the retransmission trip and hence save energy considerably [19][78][79]. Figure 3.1 illustrates how such a forwarder (node h) in our proposed ***proactive*** framework deals with video packets. When a NACK message carrying ARQ arrives, some requested packets might be found "cache-hit" at this node; if the channel cannot accommodate all the traffic, a smart selection on arrival packets and cache-hit ones can be realized. Arrival packets can be either truly fresh (being sent for the first time from the sender) or a retransmitted one if it is allowed to be re-sent multiple times.

To realize the idea, we design a middleware module called RtME (***Real-time Media Engine***), which makes the forwarders more intelligent and cooperative to the relaying process. Loosely speaking, it may be considered as a plugin-like module that tops the routing layer. The introduction of RtME is supported by the following observations. Firstly, in real-time video, life-time of packets is very short (only in order of hundred milliseconds [14]). Packets suffered excessive delay should be dropped if they could be detected. Secondly, the evolution of video compression techniques (e.g. MPEG, H.264) allows a bandwidth as narrow as a hundred Kbps to accommodate a multimedia flow. Thirdly, unlike the global Internet, an ad hoc network is deployed to facilitate field operations, with limited number of users over a relatively small area only. As a result, each node serves very few, if not single, users at a time. Regarding multimedia applications, the most popular scenario is that each relaying node works with no more than one single video stream. In reality, a single video is mostly delivered concurrently over multiple paths [8][17][21][23][24][34], so

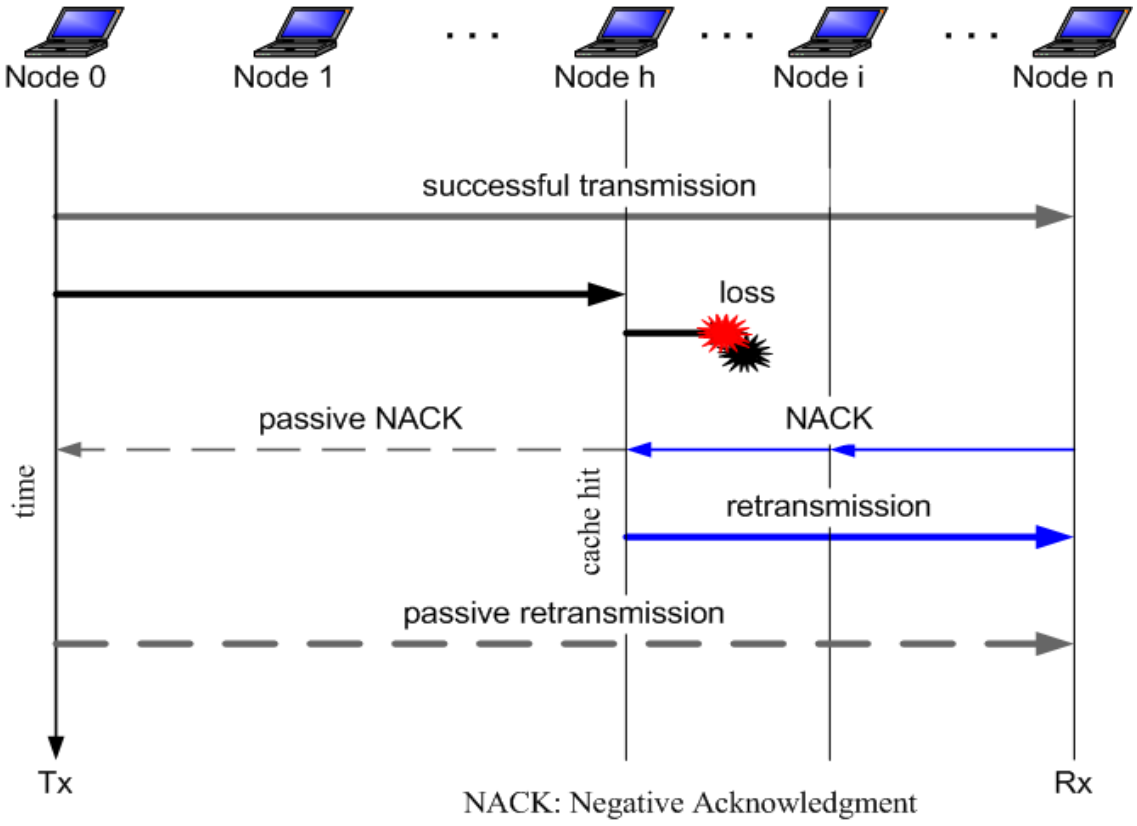

Figure 3.2: Retransmission from intermediate node. The retransmission round of the NACK message and the retransmitted packet is considerably reduced in comparison to the passive case.

the workload imposed on each relaying node is even more lightened. These facts make the following demands bearable to almost every ad hoc device nowadays:

- a small cache of a few hundred Kbits, and;
- a light computation load on packet selection for an appropriate transmission policy.

The introduction of RtME will bring benefits for both user side and network side, since it makes ad hoc forwarders smart at processing video packets. Firstly, it shortens the retransmission distance for lost packets. Note that retransmission takes place very frequently as stated in Chapter 2. Obviously, such a processing strategy does improve the success of retransmission and reduce cumulative energy consumption pertaining to the route. Secondly, caching video packets allows RtME to schedule transmission of video packets appropriately for the best quality of received video. Specifically, relaying nodes, by being aware of packet semantics, always try to keep packets of important frames (e.g. I frames) away from loss. Finally, RtME also helps the nodes detect and drop useless packets, including obsolete packets and those dependent on a corrupted frame.

3.2 Retransmission from intermediate nodes

In dealing with end-to-end loss at the packet level, conventional networking protocol stacks (e.g. TCP/IP, IPX, etc) usually rely on retransmission at the transport layer. This protocol augments a mechanism to detect packet loss at the receiver and instructs the sender how to react to ARQ requests. Only the very sender can regenerate and re-send a packet to the receiver. Should the route remain unchanged, every node along the route receives and then forwards the packet as it did in the first instance of transmission. So each time the packet is retransmitted, the cumulative load induced to the path is duplicated. In multihop wireless communication, as bandwidth is limited and unstable, careful considerations need to be taken. While retransmission helps recovery of lost packets, it also generates more overhead. Ad hoc networking devices normally run on battery power, so excessive overhead is even more unacceptable [8][21]. With the proactive relaying nodes, the distance that NACK messages and retransmitted packets travel is potentially reduced, en-

abling retransmission even in case the number of hops from the sender to the receiver is large.

Video traffic is inherently bursty, therefore retransmission of all lost packets is unrealistic, particularly during bad communication periods. In ad hoc networks, due to their lossy channels, this is even clearer. Instead, packets are selectively retransmitted, depending on their importance and timing status as well as on the network condition. Note that every packet is associated with a predefined deadline, the number of retransmission times is therefore not unlimited. The longer the distance (in term of hops) between the sender and the receiver, the fewer retransmission rounds are permitted.

3.2.1 Communication reliability

Suppose that video packets travel through n hops toward the receiver as seen in Figure 3.2, passing across nodes 1, 2, 3... and n-1. We first consider the case in which all the intermediate nodes are passive forwarders. Once the receiver sends an ARQ, the retransmitted packet must traverse every node on the same path as it did in the first transmission instance. To simplify the problem, we assume that the back-route for the packet carrying ARQ requests is the same as the forth-route. In each round of retransmission, node i receives the request from the adjacent downstream node $i+1$ and then transmits it to the adjacent upstream node $i-1$; later when the requested packet arrives, node i receives it from node $i-1$, and then transmits it down stream. So any intermediate node receives and transmits twice each for every retransmission.

On the other hand, with our video-cooperative framework, once an intermediate node h caches video packets for a while, it is able to reply to a NACK message requested for a unsuccessfully received packet that is retained at the node (called "cache-hit"). In such a case, nodes from 0 to $h-1$ are absolutely free from the retransmission process. The total length of the round trip is therefore only $2 \times (n-h)$ hops, rather than n as in the passive case, given that the back-route conveying the NACK message is the same as the forth-route. This obviously reduces the cost of retransmission in term of energy and bandwidth. If *cache-hit* occurs at node h then we observe the following benefits on the whole path.

Firstly, as fewer nodes get involved in the communication of the retransmitted

Table 3.1: Parameter definition

BER_i^f	bit error rate on the forth direction
BER_i^b	bit error rate on the back direction
$E_{tx}^i(p)$	energy consumed on transmitting packet p at node i
$E_{rx}(p)$	energy consumed on receiving packet p
$E_{nack-tx}^i$	energy consumed on transmitting the NACK message at node i
$E_{nack-rx}$	energy consumed on receiving the NACK message
L_p	packet length
L_r	NACK message length
d_i^f	delay of hop i on the forth direction
d_i^b	delay of hop i on the back direction
h	node that cache-hit occurs
i, j	generic indices
k_i	the number of nodes under the coverage of node i
n	the number of nodes along the transmission path
p	packet under consideration
ΔN_{tx}	reduction of *transmits* on proactive framework
ΔN_{rx}	reduction of *receives* on proactive framework
ΔE^{re}	reduction of energy on proactive framework
$\Delta P_{success}^{re}$	gain factor on transmission from an intermediate node
$\Delta P_{success}$	cumulative gain factor on transmission from an intermediate node

packet and the NACK message, the round-trip-time (rtt) is reduced by:

$$\Delta rtt = \sum_{i=1}^{h} (d_i^f + d_i^b) \tag{3.1}$$

where d_i^f, d_i^b are forth- and back-delay of hop i (connecting node i-1 to node i), respectively (refer to Table 3.1 for denotation of the symbols). Roughly speaking, these delay values adhere to queuing and transmitting the packet. As the distance between ad hoc nodes is practically short, the radio propagation delay is negligible. The reduction of round-trip-time consequently depends on the number of nodes that stay away from the retransmission, not on the length of each individual hop. The shortening of the round does not only decrease latency, but also enables more feasible retransmissions before the packet misses deadline.

As traversing fewer hops the packet gains higher success rate. Let's denote bit error rate (BER) in forth- and back-direction as BER_i^f, BER_i^b, and L_p, L_r as the length of the video packet and of the NACK, respectively, then the success rate per retransmission is multiplied by:

$$\Delta P_{success}^{re} = \frac{1}{\prod_{i=1}^{h} (1 - BER_i^f)^{L_p} \times (1 - BER_i^b)^{L_r}} \tag{3.2}$$

Reduction of the round-trip-time also creates more feasible retransmissions before the packet misses deadline. This results in an increased probability that the packet is successfully received. As long as the packet has not been stale yet, it can be repeatedly retransmitted until arriving correctly at the receiver. Therefore, compared to the passive case, the cumulative gain factor regarding success rate is even higher:

$$\Delta P_{success} = \Delta P_{success}^{re} \times \frac{\sum_{i=1}^{n} (d_i^f + d_i^b)}{\sum_{i=h+1}^{n} (d_i^f + d_i^b)} \tag{3.3}$$

Appendix Chapter A presents the estimation of success rates of the proactive retransmission in further detail.

3.2.2 Energy saving

Shortening the retransmission distance does not only raise the success rate of communication but also reduces the cumulative energy consumption. Note that, both *receive* and *transmit* consume considerably more power than idle status [37][38].

As already studied, both *receive* and *transmit* activities do consume additional energy; in [37] for instance, standby:receive:transmit consumption ratio is 1:1.2:1.7. It is therefore worth lowering the number of *transmits* and *receives*. Compared to the passive relaying scheme where every retransmission is made by the sender, fewer nodes join the relaying of retransmitted packets. Consequently, cumulative energy consumption is reduced. With cache-hit at node h, the numbers of *transmit* and of *receive* along the transmission path are reduced each by:

$$\Delta N_{tx} = \Delta N_{rx} = 2 \times h \tag{3.4}$$

In fact, each time a packet is sent out from a certain node, not only the destination node but also other nodes nearby get involved in the communication [16]. Thus, the actual energy-saving is potentially more noticeable.

When retransmission can be made from an intermediate node, the benefit regarding energy efficiency can be evaluated as follows. Assume that "cache-hit" takes place at node h, all its upstream nodes including the sender do not have to get involved in the retransmission of the requested packet. This saves the following cumulative power consumption amount:

$$\Delta E^{re} = \sum_{i=0}^{h-1}(E^i_{tx} + k_i \times E_{rx}) + \sum_{i=1}^{h}(E^i_{nack-tx} + k_i \times E_{nack-rx}) \tag{3.5}$$

where $E^i_{tx}, E_{rx}, E^i_{nack-tx}, E_{nack-rx}$ are respectively the consumption for transmitting (from node i) and receiving the packet by its k_i neighboring nodes, and those values associated with the NACK.

Introduction of proactiveness to relaying nodes also make them able to detect useless packets. For instance, once an I frame is corrupted, all the subsequent frames within the group of picture should be removed from the network. Dropping those packets will further save bandwidth and energy for the transmission path. This will be presented insightfully in Chapter 5.

3.3 Features of proactiveness

Our proactive framework for wireless ad hoc communication is motivated by the emergence of the active network approach [4][7][41][43], where networking devices are programmable elements, such as proxies, wireless base stations, gateways,

and active routers in conventional wired networks. As a results, the framework does incorporate advantageous features of this approach. This section analyses the bases supporting the augmentation of proactiveness from the perspective of next-generation networking design.

1. **Video packets are smartly processed**. As stated previously, video packets do not contribute equally to the decoding process. Some packets are particularly important in the sense that their loss will significantly degrade the perception quality, such as those of I frames. Additionally, loss of such a frame makes transmission of numerous packets useless. On the other hand, loss of packets pertaining to B frames affects the playback almost transiently. The effect sometimes may be invisible to human eyes. This motivates findings of smart mechanisms for processing video packets appropriately according to their semantics. The issue has been extensively investigated in conventional wired networks [4][7][41][50]. However, no study on ad hoc wireless forwarders have been conducted so far. Our proposed framework is to realize such a smart processing to ad hoc wireless forwarders, keeping in mind that communications are forced under various constraints concerning bandwidth, energy, CPU power, etc.

2. **Enforcing hop-by-hop functionality is easier**. Development of communication systems requires more and more flexibility of the network node architecture so that diverse needs can be fulfilled. Once the proactive framework is reinforced, it is simple to augment more functionalities at intermediate nodes. This is similar to the active network architecture that CISCO and other well-known manufacturers pursuit [80]. Enhanced functions at intermediate nodes can be found in data processing for security, multicast exchange of control packets and delivery of lost packets, and adaptive rate control at per-hop basis as previously described in Section 2.5.3.1.

3. **Simplicity is highly respected**. Ad hoc nodes are essentially restricted in computational power, enforcing complicated tasks for processing video packets would not be encouraged. As packets arrive at a high rate, high-complexity algorithms do not only induce unbearable load to the forwarding nodes, but they also add considerable delay to packets themselves [7]. Furthermore,

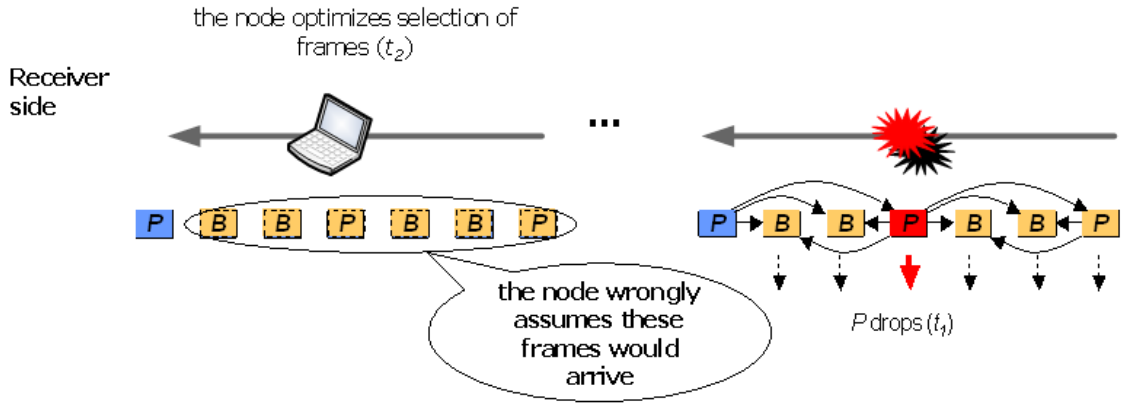

Figure 3.3: Wrong selection due to unexpected loss of the P frame upstream. At time t_1 the P frame is rejected at the upstream node. Later at time t_2 the downstream node must make some selection decision; it does not know that a cluster of frames never arrive.

heavy computation does raise the power consumption at the nodes. While optimality-oriented algorithms such as RaDiO/CoDiO reduce the distortion of the received video, we do not introduce them to intermediate nodes. The algorithms are proved to have high complexity (which is exponentially proportional to the number of packets under consideration [6]). Consequently, applying them to all forwarding nodes is not efficient. Another reason to avoid complicated optimal algorithms at relaying nodes hinges on the predictability of packet arrivals. Optimality can only be reached when all expected packets successfully arrive or are available as assumed; this is nevertheless true only at the sending node where no link loss is suffered. In ad hoc wireless communication, packet loss is caused not only by nodes (e.g. due to buffer overflow), but also by lossy links, which adds more uncertainty to any packet arrival at intermediate nodes. As illustrated in Figure 3.3, if a frame P (that is expected to arrive at the node) is lost somewhere upstream, optimality-oriented selection may be far worse than no selection at all.

4. **Relaying strategy should converge to the conventional passive forwarding when the channel is ideal**. To gain the benefits above, relaying nodes must be added more packet processing load to realize the proactiveness. When the transmission suffers loss and lack of resource (which is true most

of the time in ad hoc wireless networks), the benefits that the proactiveness brings up outweigh its cost. Nonetheless, to make the framework realistic, any forwarding strategy should guarantee that, if the network is in a good communication status, it behaves close to the conventional forwarding mechanism. This also means that there should be no major difference between the two with regard to the communication performance.

In the next chapter, we will see how these criteria are respected when realizing the proactiveness for relaying nodes. While making nodes able to respond to retransmission requests, the design keeps the cache as small and as simple as possible. Packet selection is simplified while packet queues purely follow FIFO, which leans toward best-effort nature.

3.4 Applicability range

Adding more caching and processing load to ad hoc forwarders, we expect that the efficiency of the framework is maximized regarding both communication and computation. It is obvious that the benefits of the proactive framework will be highest if the routes conveying video packets and NACK messages are identical. Namely, on the way to the sender node, NACK messages traverse proactive forwarders that potentially cache video packets. In a typical ad hoc network, as wireless links are symmetrical, the above assumption is basically correct. On the other hand, should the network contain some asymmetric links, the forth- and the back-routes might not be the same. However, retransmission from an intermediate node is usually still cheaper than from the sender itself. To take up advantage of the proactive nodes on the forth-route, the ad hoc routing protocol should give higher priority to those back-routes that cover more nodes of the forth-route, when transferring ARQ requests, even they may be longer. If no back-route consists of any proactive node, then of course there is no benefit concerning retransmission at all. Such a situation may occur when all the nodes involved have their antenna perfectly directed. Another example is a sparse network where the nodes have entirely diverse transmission power levels. In those cases, the network topology must be overwhelmed with asymmetric links to devaluate the framework. Even so movement of nodes, interference, and environment changes potentially make the scenario

transient.

Intuitively, the introduction of the enhanced forwarding functions does not pose any issue of backward compatibility. Namely, video-cooperative hosts can co-exist and efficiently inter-operate with conventional passive nodes. Arriving at proactive forwarders, only video packets are treated in a different manner while other traffic classes are relayed as over passive ad hoc routers. The next chapter will detail this behavior.

Practically, it is unlikely that a particular node joins a video session, relays only a single packet, and then goes away from the route. Note that in ad hoc routing protocols, route update cannot be made too frequently. The route update period is normally in order of seconds [65]. In addition, video traffic is bursty as stated in Section 2.2; so once a node is present at the route, the number of packets going through should be large enough for the proactiveness to take effect. Typically, a video flow has its frame rate no less than 10 fps, so the average packet inter-arrival is shorter than 100 ms. Consequently, caching and computing for a smart packet selection is always worth-doing at forwarders.

It is obvious that cache-hit rate at relaying nodes is influenced by congestion over the links. The closer to the receiving side the bottleneck link is, the higher the efficiency of the proactive framework. Specifically, on the way down to the receiver, packets tend to reach relaying nodes as far as the traversed links are in good conditions. Should retransmission requests be sent back to the sending side, cache-hits potentially occur at nodes that are close to the receiver. As a result, the more upper links are good, the shorter the retransmission distance. In this sense, retransmission from intermediate nodes potentially helps reduce traffic load over upstream links. This contributes to decrease the possibility of congestion at links close to the sending side, which in turn raises the benefits of caching video packets at relaying nodes. The worst case happens when the very first link gets jammed seriously, which blocks video packets right at the sender. In such a case, retransmission can be made only from the sender itself. Note however that, unlike wired networks, ad hoc nodes are mobile and the wireless channel is changing, that a particular link is persistently in a good or bad status is unlikely. Furthermore, QoS routing protocols naturally tend to bypass those links that are poor over a long period [73][74][81].

3.5 Chapter review

Differently from other proposals for video transmission over ad hoc networks, we consider the perspective to enhance the functionality of relaying nodes instead of ending hosts. This chapter has presented the motivation of making those nodes more cooperative to the forwarding process. The benefits regarding communication reliability and energy has been analysed. In the next chapter, we will show how to realize the proactiveness on the nodes.

Chapter 4
DESIGN OF REAL-TIME MEDIA ENGINE

As stated in Section 3.1 of the previous chapter, to realize the proactiveness of ad hoc relaying nodes, we introduced a plugin-like module called Real-time Media Engine (RtME). Basically, this module is only activated at an intermediate node if it has video packets to forward. This chapter presents the design of the module, showing how packets are accepted, cached, enqueued, and forwarded at the node. We also analyse the feasibility of the framework with respect to memory usage and computation load.

4.1 Overview

Proactive functionality of ad hoc nodes that relay video packets lies at RtME. It includes controlling the packet acceptance, storing packets to the cache, smartly dequeuing them, and cooperatively handling retransmission requests. This module filters in video packets for further processing before transmitting them downstream. It also handles incoming NACK (Negative Acknowledgment) messages that carry ARQ requests. As illustrated in Figure 4.1, RtME includes several entities below:

1. The Rx is responsible for filtering in video packets, and discarding obsolete packets. At the sender node, it simply accepts datagram from the upper layer, whereas at the receiver node, it reorders packets if necessary;

2. The Warehouse accommodates and maintains video packets as described earlier. At the receiver node, this should be replaced by a buffer supporting the decoding process;

3. The NACK Handler processes ARQ requests, and forwards them upstream if necessary. At the receiver node, this entity is to generate and to send NACK messages instead;

4. The Tx works with packets in two separate queues ("fresh queue" for packets from upstream, and "response queue" for the packets that are cache-hit in

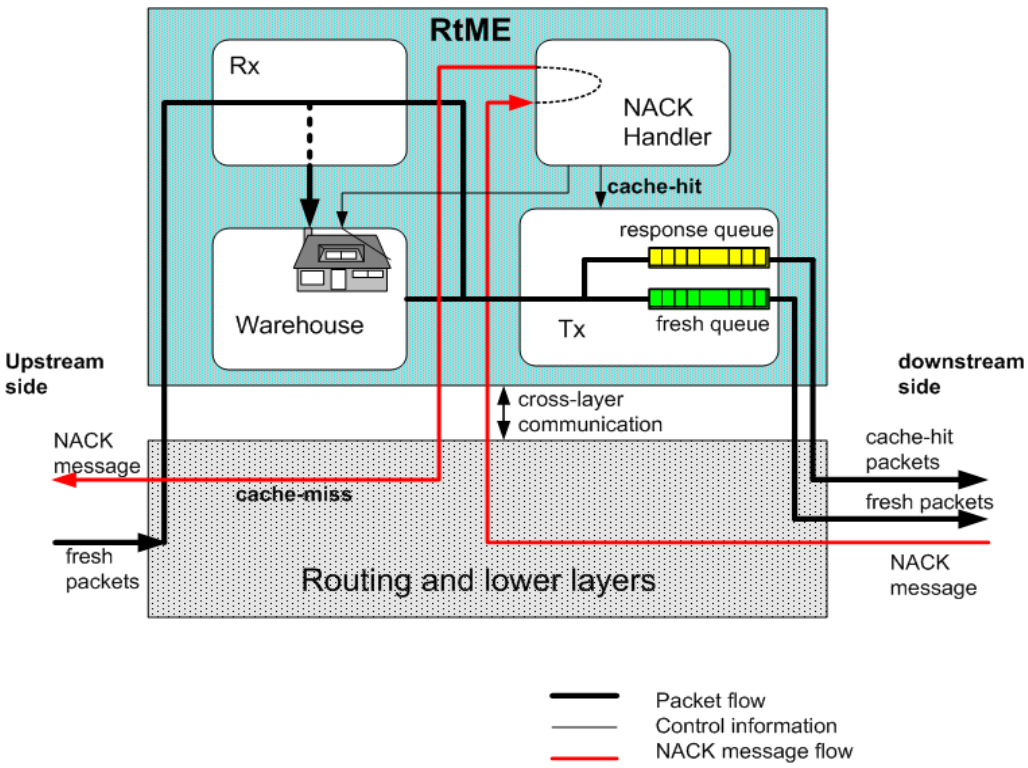

Figure 4.1: Construction of Real-time Media Engine. The middleware tops the routing layer and includes four submodules (entities).

this node); it selects the appropriate packet to transmit when the channel is available. The receiver node, of course, does not have this entity.

From the programming point of view, all the entities except the warehouse are implemented as parallel threads, synchronized by a *semaphore*. Theoretically, other implementation architectures such as inter-process communication (IPC) or signaling mechanism can also be employed. In this study, we choose the approach since the aforementioned entities have a strong need to share global variables defining the cache and the queues. Practically, multithreading is supported widely in modern OSs (e.g. POSIX *pthread* library in Unix-like OSs, and *process* library in MS Windows).

4.2 Acceptance of incoming video packets

In the conventional wired networks, each relaying node admits every packet and instantly forwards it to the next hop in a best-effort fashion. Which hop is

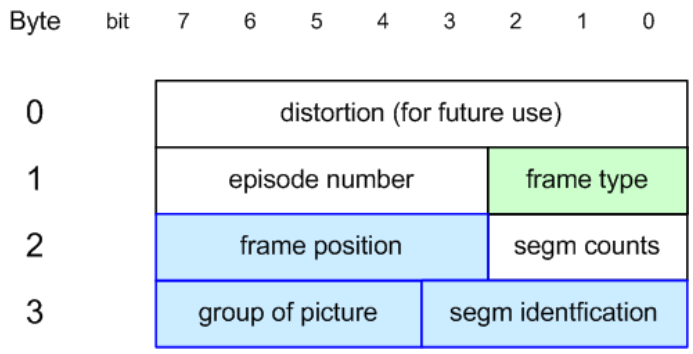

Figure 4.2: Proactive header. Four bytes are added to indicate the complete identification, the type of the packet, and the number of packets within the encompassing frame.

selected depends on its destination network address. Should the communication channel be not available, the packet is decisively destroyed without caching for any while. Namely, no packet admission control is reinforced. On the other hand, proactive relaying nodes allow to filter out useless packets thanks to the fact that packets are transiently retained in their cache.

At each intermediate node, video packets arriving from upstream are loosely called "fresh packets". Actually, a "fresh packet" may be either a packet sent for the first time from the sender or a retransmitted one that has been generated somewhere upstream. Note that a fresh packet may also be obsolete (generated earlier than the obsolete bound). Entity Rx performs the following functions upon a packet arrival: extracting header and recording packet information, sending the packet to the warehouse, and logically queuing the packet.

4.2.1 Processing proactive header

To process video packets in a way we expect, each packet must contain enough overhead information so that RtME can recognize its identity. At the sending side, the application layer adds 4 bytes to each packet so that proactive nodes can identify and process appropriately. Figure 4.2 shows the structure of this proactive header. The data include the identification of video packet, the number of packets derived from the encompassing frame, and the frame semantics. Each video packet is identified by the group of picture it belongs to, the encompassing frame position in the

group of picture, and the segment number. These are respectively represented by fields "group of picture", "frame position", and "segm identification".

Since packets are physically stored to the cache in a non-contiguous manner, these pieces of information is needed for later regeneration of each packet. Upon receiving a packet, RtME extracts this information and stores to internal variables for further computation. Namely, the identification numbers are used as an index anytime the associated packet is retrieved. Why do we use up to three identification numbers to identify a single video packet ? The answer is related to the scalability and the need for multiple level access: group of picture, frame, and packet. Each video sequence may generate a vast number of packets that are not independent of each other, so using a unique number to sufficiently identify a packet is tedious. Specifically, actions such as removing a packet from the warehouse, selecting a packet to forward, etc, need to know not only the packet identification number, but also the frame that it belongs to.

When receiving a particular packet p for the first time, RtME at a node records the arrival time (denoted as $t_p^{arrival}$) so that the residing time of the packet at the node can be estimated when needed. Later on, if this packet, due to some reason, does not correctly reach the destination node, it may be retransmitted and hence arrive again at the current node. At this retransmission and the later if any, the node estimates the elapse time since the arrival time. Note that the sender must have generated the packet earlier than its arrival time at any relaying node. As a result, if this time duration is greater than the maximum allowable delay of video packets, the packet definitely misses its deadline, and hence should be dropped by entity Rx.

4.2.2 Dispatch to cache

Since a particular packet can be retransmitted multiple times, it is likely that a node receives a packet repeatedly. In such a case, the stored header information on its first arrival helps the node identify the duplication of receipt and hence drop it. In other words, before caching any packet, the RtME checks whether it is already present in the warehouse. Once the packet is accepted, the state information of the associated frame at the node, such as total size of frame and the number of available packets must be updated.

With the assistance of multiple level accessing mechanism stated in the previous section, storing a packet to the warehouse is just a matter of dynamically allocating a memory equal to its size. The reason of using this dynamic allocation is that we do not know in advance how many packets of a frame will really arrive at the nodes, as well as how large they are. Consequently, static allocation at peak level would cost much more memory than actually needed, given that video packets are so varying in their size, as mentioned in Section 2.2.3. Chapter 6 will describe this function in detail.

Practically, the life-time of video packet in real-time applications is very short, as said previously. Obsolete packets should be removed from the warehouse as soon as they are detected. Nevertheless, to simplify the removal and hence reduce the workload, RtME does not retrieve each stale packet individually upon every packet arrival. Instead, it only locates obsolete packets to destroy if a fresh packet of a new video frame arrives. At such a time, the video frame is added to the warehouse, and entity Rx completely removes all packets of the obsolete frames.

4.2.3 Queuing packets

Once a packet is admitted by the Rx, it should be enqueued for relaying downstream whenever the channel is available. For fast processing, the RtME does not *physically* dispatch packets to the queues. Instead, valid arrival packets are *logically* pushed to a designated queue called "fresh queue". Namely, only extracted header information, which is collectively stored in a structure variable, is added to the queue, while the physical data remain untouched in the warehouse without any copying or moving.

Upon queuing each incoming packet, the semaphore controlling entity Tx has its value increased one unit, telling the Tx thread that there is one additional packet ready for transmitting. This mechanism is to minimize the CPU usage, as the thread is only awaken when there is a packet ready to transmit. Once successfully forwarding the packet downstream, entity Tx subtracts one unit from the semaphore value. Should the value become zero, the Tx goes sleeping again. Chapter 6 will describe this operation in detail.

To facilitate the warehouse maintenance, such as seeking a particular packet, deleting a frame, the warehouse is organized as follows. Each video frame, including

its data, header, and recorded information (e.g. arrival time), is associated with a pointer called frame pointer. We define a map container to hold frame pointers of all currently cached frames. As a packet arrives, its header is extracted first to identify its encompassing frame. If the packet is of a frame that is not currently available, a new frame pointer is initiated to hold the packet; otherwise, the record pointed by its encompassing frame pointer is updated with the additional packet. Accordingly, maintenance of the warehouse is actually done on the map container of the frame pointers only, rather than the physical frames.

In an ideal transmission status, the channel is always ready for every arriving packet. If this occurs, packets are forwarded downstream immediately as soon as they approach the designated queue. The proactive framework therefore converges to the conventional forwarding scheme. However, in ad hoc wireless communication, such a good situation is very rare.

4.3 Caching video packets

Unlike conventional video caching systems where packets are stored in secondary memory (e.g. HDD), the cache here should be allocated on electronic memory (e.g. DRAM), since real-time video traffic is so time-sensitive and bursty. As analysed later in this chapter, video frame's life-time is very short, while ad hoc nodes with limited communication bandwidth can normally serve only one video flow at a time, making the demand for space not prohibitive.

After extracting packet header and assigning the obtained values to internal variables, entity Rx considers allocating memory for the physical body of the arrival packet. For simplicity, the warehouse is maintained as shown in Figure 4.3. Let's assume that the maximum allowable number of frames residing in cache is N_c. Whenever a packet (or a fragment) of a new frame f (not a retransmitted frame) arrives, all the frames before frame $f - N_c + 1$ should be freed from the warehouse, since they have definitely missed deadline (obsolete). The latest frame to be deleted is $f - N_c$, which is called *obsolete bound*, whereas f is called *fresh bound*. Since there may be gaps in frame numbering due to packet loss, potentially fewer than N_c frames are in the warehouse. Upon the arrival of a packet, Rx extracts its header to identify which video frame it belongs to. If the encompassing frame was generated

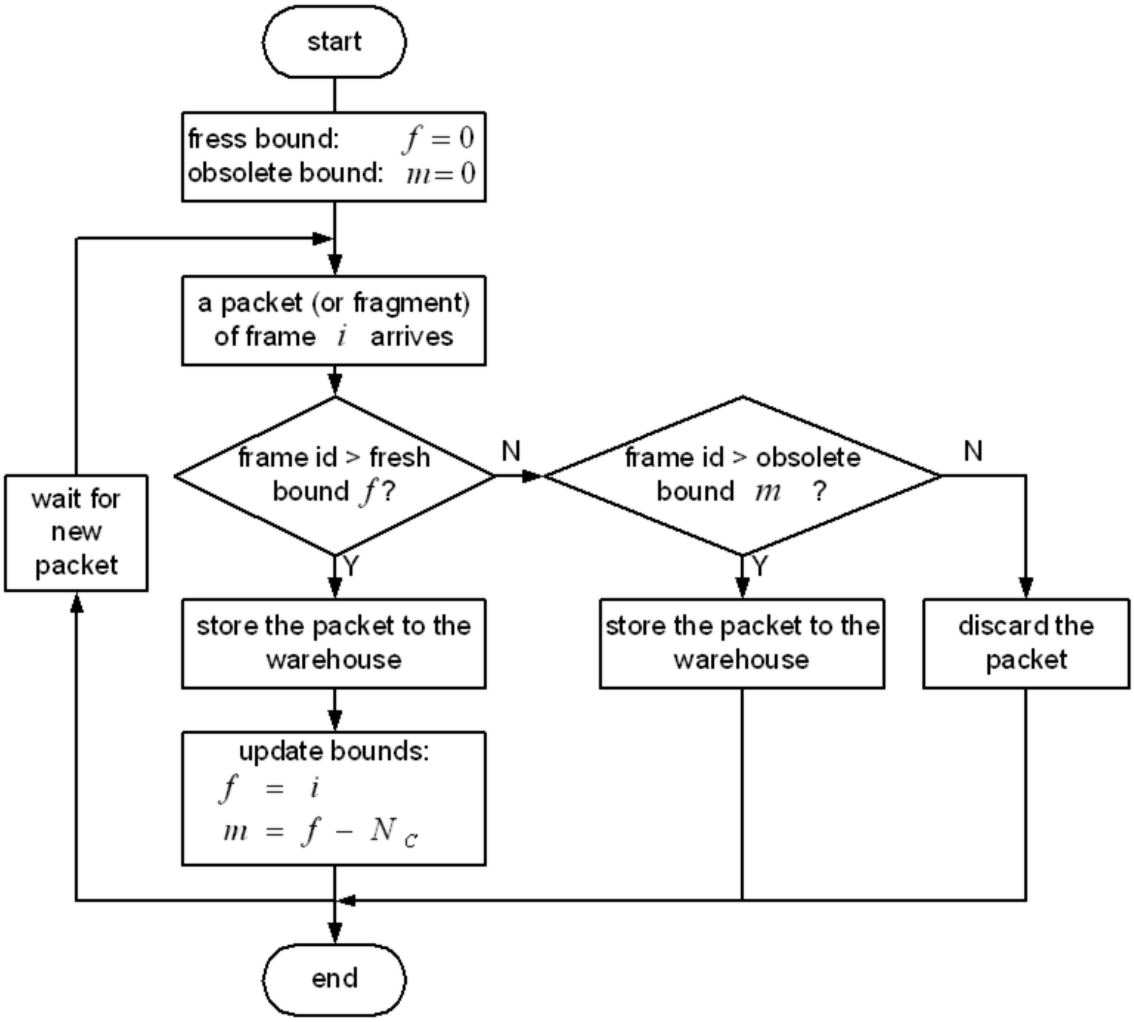

Figure 4.3: Warehouse maintenance. RtME at each node maintains two bounds: *fresh bound* f and *obsolete bound* m. Packets generated earlier than the second bound should be discarded without any processing.

earlier than the obsolete bound, it will decisively be dropped.

4.4 Processing NACK message

To avoid excessive overhead, successfully received packets are not acknowledged. NACK messages indicate only the lost packets that are not obsolete yet. Format of a NACK message is shown in Figure 4.4. Each requested packet is represented by a single byte, identifying the encompassing frame and its fragment number. The total length of the message equals to the number of requested packets plus two bytes describing the *just successfully received packet*. There are cases in which the receiver node does not receive any packet for a long time. In such cases, it would be meaningless to indicate the just successfully received packet in the NACK message. Instead, the two bytes should represent the identification of the frame that was expected to arrive at the receiver node at the time when the NACK message was generated. It is possible for the receiver node to predict the identification since video frames are numbered in an iterative natural order. In addition, from overhead information of the video or by observing arrivals of frames, the node can also estimate the frame rate.

Upon receiving, the ad hoc node checks if the encompassing video frame is still in the warehouse. If so, it attempts to update the round-trip-time to the receiver node that is calculated simply as:

$$rtt = t_p^{now} - t_p^{ltx} \tag{4.1}$$

where p is the packet that has just been successfully received (right before this NACK message was generated at the receiver node); t_p^{now} is the current time; and t_p^{ltx} is the time when packet p was last sent from this node. The updated rtt will be referred to each time entity Tx decides whether to transmit a packet downstream. This will be presented specifically in Section 4.5. In a cross-layer information sharing, proactive routing protocols can also provide statistics (from routing messages) that support the estimation of rtt.

Before interpreting the list of requested packets, the handler evaluates the elapsing time since packet p arrived at this node:

$$t_p^{elapse} = t_p^{now} - t_p^{arrival} \tag{4.2}$$

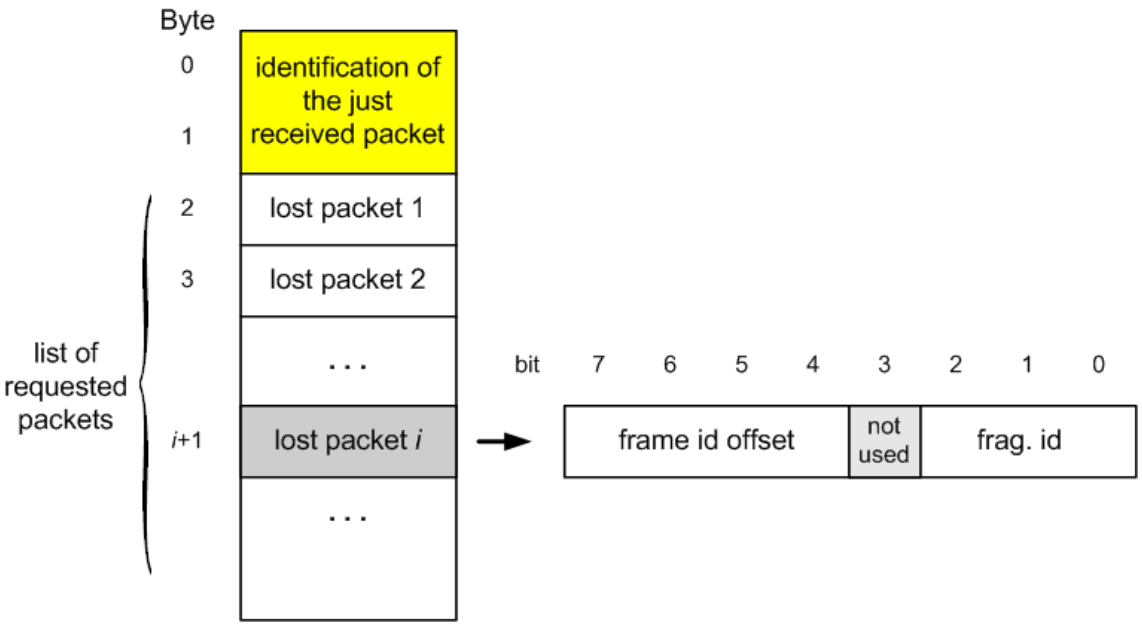

Figure 4.4: Format of NACK message. Each message contains two header bytes that indicates the just successfully received packet and the body that lists requested packets.

where $t_p^{arrival}$ is the arrival time of packet p that was recorded by entity Rx. If the time elapse is greater than delay threshold D^{thres} , then all the requested packets have certainly missed deadline, and this NACK message is obsolete and should be dropped.

Once the node determines that the NACK message is not obsolete, a list of requested packets is interpreted. Each byte carries the data indicating frame and fragment identification offsets of a requested packet. When a whole frame completely gets lost, the receiver can only tell that the whole frame needs to be retransmitted, but it does not know how many packets the frame contains. Thus, a single byte of the NACK may correspond to multiple packets. After the translation of all the bytes, a list of requested packets is completed. The node then locates those packets that are currently in its warehouse. These packets are considered "cache-hit", and are queued into the designated queue (called "response queue"). If there is any requested packet that is "cache-miss" here, the node forwards the NACK message upstream; otherwise, the message is suppressed. Before forwarding, the node cuts off all the bytes corresponding to the requested packets that are cache-hit. This cut-off will save bandwidth on the back-route. The operations above can be formally

represented in the abstract pseudo-code of List 1 below. The primary variables are `cache_miss_count` that counts the number of cache-miss packets, and formal array `byte[]` that holds the body of the NACK message.

List 1 - Processing a NACK message

```
1: cache_miss_count = 0;
2: this_NACK_suppressed = true;
3: /* scan content of NACK message */
4: for (i=2 ; i < NACK_message_length; i++)
5: {
6:  Translate byte[i] into the requested frame (called fr_ptr);
7:  if (fr_ptr is found in the warehouse)
8:  {
9:   Determine the set of requested packets (called req_ids) from
10:                                                        byte[i];
11:  Determine the set of cache-hit packets (called cache_hit_ids)
12:                                                  among req_ids;
13:  Push cache-hit ids to response queue;
14:  Increase tx semaphore value;
15:  if (all req_ids are cache hit)
16:  {
17:   Cut off byte[i] from NACK message;
18:  }
19:  else
20:  {
21:   this_NACK_suppressed = false;
22:   Increase cache_miss_count;
23:   Modify NACK message;
24:  }
25: }
26: else
27: {
28:  /*the whole requested frame is not found*/
29:  this_NACK_suppressed = false;
```

```
30:  Increase cache_miss_count;
31: }
32:}
33:if (cache_miss_count > 0)
34:{
35: Forward the modified NACK message upstream;
36:}
37:/*end of pseudo-code*/
```

As shown in the list above, for each byte of `NACK_message`, the NACK handler needs to identify the encompassing frame `fr_ptr` before determining the set of cache-hit packets (`cache_hit_ids`). Upon queuing each cache-hit packet, the semaphore controlling entity Tx also adds one unit to its value (`tx_semaphore_value`), similarly to the acceptance of a fresh packet described earlier in Section 4.2.

4.5 Relaying mechanism

In each node, there are two logical queues, one for incoming packets (fresh queue) and the other for cache-hit packets (response queue), as presented previously. Each time the channel resource is ready, entity Tx dequeues a packet from one of the queues if available. To save computation resource, the Tx thread is only activated when the semaphore value becomes positive. Once awoked, the entity first looks for the most appropriate packet from one of the queues, and then checks if it is really meaningful before sending it downstream. Obviously, in a good propagation status, the response queue should be empty, and packet selection is just a matter of taking the only packet from the fresh queue. In other words, every incoming packet is sent virtually immediately, making both queues empty most of the time. This behavior demonstrates the convergence nature of RtME that was mentioned in Section 3.3. Namely, the operation of the proactive strategy is virtually the same as the conventional forwarding scheme. Note however that, in wireless communication, video packets usually face shortage of resource; such an ideal moment, if any, is transient.

4.5.1 Packet selection

When the resource is not always enough to instantly forward every incoming packet, we need to select packets to relay taking minimization of distortion into account. Algorithms for selection should be straightforward to avoid excessive processing delay and to save computation resource. Here we introduce a frame-semantics based algorithm as follows. At first, the Tx reads the head-of-queue packets from each queue if available. Then, which packet is selected for relaying depends on (i) the semantics of the frames the packets belong to, and on (ii) their remaining life-time. Specifically, packets with more dependent ones, such as those of I frames, should be given higher priority. Among packets of equal importance, the earlier generated one should be served first. List 2 represents a cycle of packet selection in form of pseudo-code. The goal here is to return a pointer `packet_to_send_ptr` identifying the packet to forward. Note that besides I, P, and B frames, each GoP contains a "header" frame, denoted as "H".

List 2 - A cycle of packet selection

```
1:    packet_to_send_ptr = NULL;
2:    /*read out the head_of_queue packets from the queues*/
3:    fh_pk_ptr = head_packet(fresh_queue);
4:    ch_pk_ptr = head_packet(response_queue);
5:    /*decide which packet is selected for transmission*/
6:    if (both queues are not empty)
7:    {
8:      if (ch_pk_ptr->frame_type == B)
9:      {
10:       if (fh_pk_ptr->frame_type == B)
11:       {
12:         packet_to_send_ptr = ch_pk_ptr;
13:         Delete ch_pk_ptr from response queue;
14:       }
15:       else // fresh packet is of H, I, or P
16:       {
17:         packet_to_send_ptr = fr_pk_ptr;
```

```
18:         Delete ch_pk_ptr from fresh queue;
19:       }
20:     }
21:     else // cache-hit packet is of H, I, or P
22:     {
23:       if (fh_pk_ptr->GoP_id == ch_pk_ptr->GoP_id)
24:       {
25:         /* Both packets are of the same GoP */
26:         packet_to_send_ptr = ch_pk_ptr;
27:         Delete ch_pk_ptr from response queue;
28:       }
29:       else // cache-hit packet is of the previous GoP
30:       {
31:         if (fh_fr_ptr->frame_type != B)
32:         {
33:           packet_to_send_ptr = fh_pk_ptr;
34:           Delete ch_pk_ptr from fresh queue;
35:         }
36:         else
37:         {
38:           packet_to_send_ptr = ch_pk_ptr;
39:           Delete ch_pk_ptr from response queue;
40:         }
41:       }
42:     }
43:   }
44:   else //at least one of the queues is empty
45:   {
46:     if (only fresh queue is not empty)
47:     {
48:       packet_to_send_ptr = fh_pk_ptr;
49:       Delete ch_pk_ptr from fresh queue;
50:     }
```

```
51:     else
52:     {
53:       if (only response queue is not empty)
54:       {
55:         packet_to_send_ptr = ch_pk_ptr;
56:         Delete ch_pk_ptr from response queue;
57:       }
58:     }
59:   } /*end of pseudo-code*/
```

As shown in List 2, the Tx first reads the head-of-queue packets out to pointers `fh_pk_ptr` (pointing to "fresh packet") and `ch_pk_ptr` (pointing to "cache-hit packet"), using function `head_packet`. Information about frame type and group of picture is subsequently extracted, which helps the Tx decide which one to return (as pointer `packet_to_send_ptr`). In short, if both packets are of B, the cache-hit one should be served since it has shorter remaining life-time; otherwise, if only one is of B, then the other should be of course selected. If the cache-hit packet is of H, I, or P, it will also be served unless it is of the earlier group of picture while the fresh packet is also of H, I, or P. If one queue is empty, the other should be served if not empty, else the Tx thread goes sleeping.

4.5.2 Decision on forwarding

After being selected, the packet is checked whether it is still early enough to reach the receiver node. The check is done by comparing the expected time-to-destination and the remaining life-time of the packet. The former is optimistically set to $rtt/2$, where rtt is updated according to (4.1). Since a packet can be requested many times for retransmission, the selected packet should be additionally checked whether it has been forwarded within last rtt time units. In short, the selected packet is relayed downstream only if the following inequalities hold:

$$D^{thres} - (t^{now} - t^{arrival}) > rtt/2 \tag{4.3}$$

$$t^{now} - t^{ltx} > rtt \tag{4.4}$$

where $t^{now}, t^{arrival}$, and t^{ltx} stand for the current time, the arrival time, and the time of the last forwarding, respectively. If one or both of the inequalities are not

true, the selected packet should be completely dropped, and another cycle of packet selection should immediately be initiated, and so forth. Each time a packet has been sent out, the semaphore decreases one unit.

Obviously, if the path goes worse, *rtt* gets increased, breaking (4.3) more frequently, and hence more packets are dropped. So, the per-hop adaptiveness to the channel condition is enhanced. Such an exploitation of NACK messages solicits a cross-layer design.

While the RtME exploits the information concerning round-trip-time, it does not tightly depend on any other node to make forwarding decisions. If *rtt* is not updated in a long period, the node simply sets its value to the minimum value (e.g. zero) to guarantee that no packet is unnecessarily discarded. To keep the value up-to-date, the receiver node may regularly generate and send *heartbeat packets* [8]. However, this method creates more communication overhead. The algorithm is open in the sense that it can be modified more or less for different targets; it can also be combined with other strategies. The openness of the architecture guarantees interoperability of diversified nodes. Namely, proactive nodes of different relaying algorithms can work well together and with conventional passive nodes, without deterioration of the efficiency, since they make forwarding decisions autonomously.

4.6 Feasibility analysis

In exchange for the benefits regarding energy, bandwidth, and fairness at the sending side, the introduction of Real-time Media Engine (RtME) to ad hoc relaying nodes also requires memory space and poses some computation load. Each relaying node needs to reserve main memory for caching video packet transiently, as stated before. However, because the life-time of video packets is very short, the node does not have to reserve a large space. The relaying strategy, as presented in the previous section, is fairly simple, so computation load is not prohibitive to today's ad hoc devices. In this section, we consider the feasibility of the framework with respect to memory and computation resources, both analytically and practically.

4.6.1 Memory space

Upon a packet arrives at a relaying node, the RtME processes its proactive header before sending its physical body to the warehouse. In reality, caching the packet is just a matter of allocating memory equal to its size. To avoid excessively claiming memory space, packets are allocated memory on-the-fly. Namely, the RtME at each intermediate node does not reserve any space dedicated for the warehouse, but it allocates memory dynamically upon packet arrivals. Obviously, how much memory is needed to accommodate video packets depend on the following factors:

- How long each packet is allowed to remains in the warehouse. The longer each packet stays in cache, the larger space is required to accommodate all valid packets. Fortunately, in real-time video applications, the whole end-to-end delay does not exceed some hundred milliseconds [1][14].

- Inter-arrival time between adjacent packets (or arrival rate). If packets arrive sparsely one after another, then the maximum number of packets simultaneously lying in cache will be small. Video packets generally come at a rate higher than other data applications. However, limited bandwidth in multihop wireless communication does not allow to transmit at excessive frame rate, e.g. larger than 30 fps.

Life-time of real-time video packets in any node should not be longer than a predefined threshold D^{thres}. The maximum required capacity of the cache is therefore calculated as follows:

$$C = B \times D^{thres} \tag{4.5}$$

where B is the bit rate of the video flow. Following [11], D^{thres} hereafter is set to 0.5 seconds, and C is accordingly just a half of B. In reality, no one would expect a bit rate of encoded video exceeding 1 Mbps in a ad hoc wireless network, the required capacity is consequently less than 62.5Kbytes. Additionally, as video compression techniques evolve, the bit rate gets decreased. If path diversity [8][21] is exploited, the required cache is even smaller. In the experiments presented in Chapter 7, the peak value is just around 24 KB. Practically, allocation of such a limited RAM space is not unrealistic in today's mobile computers or any embedded devices.

If the frame rate is R, the maximum number of frames residing in the warehouse is as follows:

$$N_c = D^{thres} \times R \tag{4.6}$$

Likewise, over such a resource-limited environment as ad hoc wireless networks, transmission of video with frame rate greater than 30 fps is almost unlikely, so the maximum number of frames retained in the warehouse, as calculated in (4.6), is less than 15.

4.6.2 Complexity

The complexity of our framework is mainly derived from handling NACK messages and the packet selection process, as presented the previous section. Regarding the former, it can be inferred that the complexity of the translation of the NACK content is $O(N^{nack})$, where N^{nack} is the number of requested packets, and:

$$N^{nack} < D^{thres} \times R \times k \tag{4.7}$$

where k is the maximum number of packets (fragments) of a video frame. This value pertains normally to I frames. As far as we have experienced, it is not greater than 10. In the experiments with *Akiyo* [82] presented in Chapter 7, this value is only 5.

In a typical case, the packet selection algorithm only considers two head-of-queue packets, its complexity is therefore inconsiderable. In a worse case, it may pass several cycles to successfully select a packet for transmitting. The maximum cycles, nevertheless, cannot exceed the total number of packets in the two queues which is also less than $N_c \times k$. Figure 4.5 shows a snapshot of system monitoring screen. As indicated, the increment of memory usage and CPU occupancy when RtME was activated is virtually inconsiderable, even the machine ran under the graphic mode.

So, the requirements on memory and computation resource are not prohibitive. In reality, ad hoc networks are normally deployed in limited areas, and each node serves very few, if not single, users at a time. This makes the framework even more practical.

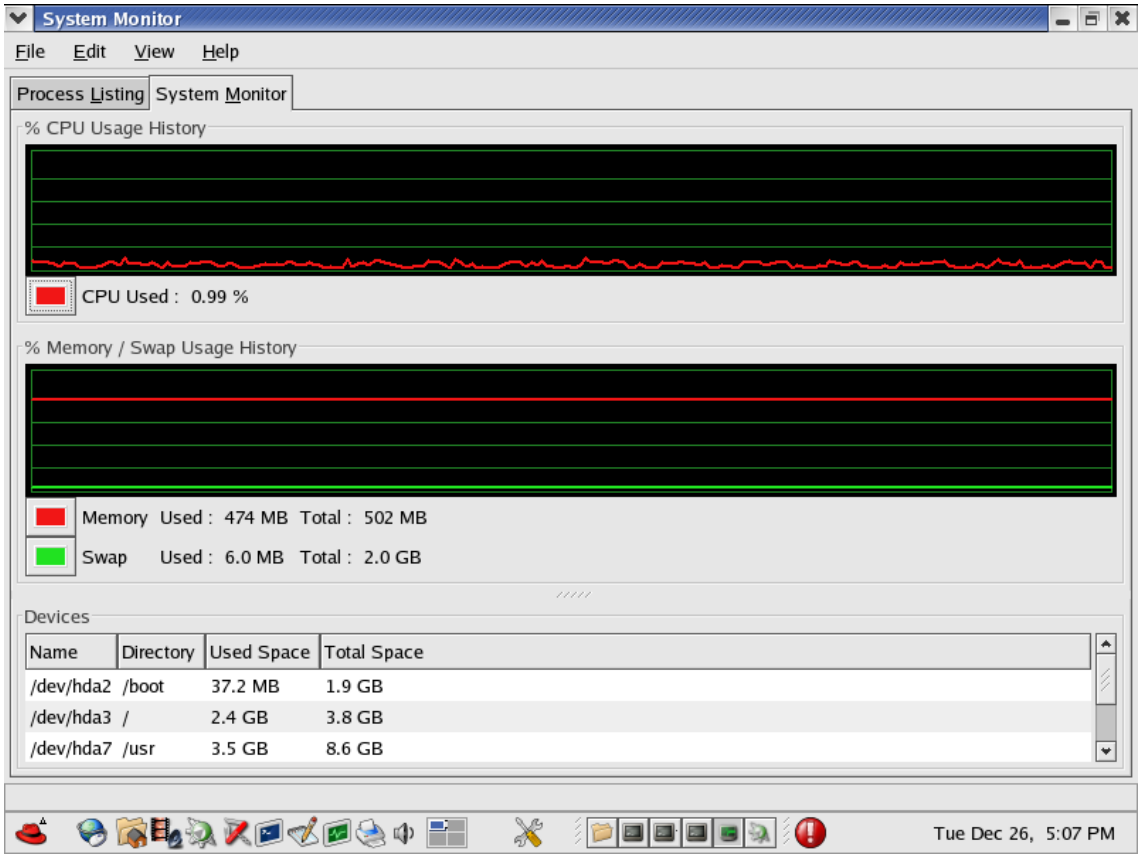

Figure 4.5: System monitoring: single path transmission of *Akiyo* sequence. Even the machine ran under the graphic mode, the increased CPU usage and memory occupation are not much considerable.

4.7 Chapter review

On realizing the proactive framework, this chapter has described the construction of Real-time Media Engine - a middleware that is responsible for caching video packets and controlling the relaying process. We have shown how video packets are accepted and forwarded as well as how the caching mechanism works. The applicability of the framework has also been evaluated through considering memory usage and complexity in the worst-case scenario.

Chapter 5
DISCARDING USELESS PACKETS

In the previous chapter, we have presented the design of Real-time Media Engine. This plugin-like module is responsible for processing video packets as well as negative acknowledgment (NACK) messages. In this chapter, we describe a mechanism to detect and to drop down useless packets. As stated before in Chapter 3, avoidance of forwarding useless packets does not only save bandwidth, but also reserves energy budget for other useful packets. Discarding useless video packets at wired networking devices have been investigated in several studies [1][4][7][41]. In [1], the author recommended to reinforce a packet loss observation mechanism at routers. Should the loss rate exceed a predefined threshold, the routers stop forwarding subsequent packets, since even these packets were forwarded, the perception quality would be still unacceptable. Application of this mechanism requires flow-state information and cannot detect useless packets that are dependent on a lost frame.

As the active networking technology [43] emerges, the video packet semantics-aware approach has extensively been studied. Strategies proposed in [4][50] specifically address the issue of video packet dependence, where they prescribe the optimal policy on selecting patterns of packets to forward. These proposals help minimize distortion but also impose heavy computation on the forwarders. Furthermore, they assumed that communication links are reliable, which is not true in ad hoc wireless networks. In reality, loss of packets attributed to link failures is significant [17][23][78], in addition to buffer overflow that is the common cause of packet loss at conventional routers. Considering lossy wireless multihop networks, the authors of [20] propose an application-aware link layer in which hosts stop forwarding incomplete frames. A node does not forward any packet until the whole frame is available. The study does not address the useless packets that are dependent on other lost frames. Obviously, this approach introduces significant delay, especially when the frame is fragmented into many packets. If the network topology or routes change, forwarding nodes potentially make incorrect discarding decisions. From the implementation point of view, it is hard to design such a link layer protocol, which

must also guarantee the backward compatibility (the strategy was evaluated via simulations). Remarkably, in all the strategies above, forwarding nodes locate and drop useless packets separately and independently. Furthermore, they cannot detect packets that do not have enough time-to-live to reach the receiver node. This chapter shows how RtME addresses those issues.

5.1 Overview

As stated in Section 2.2.1, video packets have different roles in the decoding process. Specifically, employment of MCP (Motion Compensated Prediction) in video coding techniques at the application layer creates inter-frame dependency among video frames as well as data units of lower representation levels (i.e macrobloc, slice [12]). This on one hand lowers the data volume to transmit, but complicates any packet selection on the other hand. Loss of an I frame voids all the frames of the same GoP while corruption of a P frame makes all subsequent frames within the GoP useless. This raises the issue of transmitting useless packets.

In fact, useless packets can be either ones that have been stale or those that cannot be decoded since they depend on another missing packet. As ad hoc relaying nodes record packet arrival time, they allow to determine stale ones at any time. Note that a particular packet may be transmitted multiple times. The arrival time of a packet is defined as the time when the packet first arrives at the node. Upon dropping an important packet (e.g. of I or P frame), the node memorises its identification to help detect and destroy its dependent packets in the future.

Beyond the local rejection of useless packets, we have realized a *joint discarding* mechanism in which relaying nodes can inform each other of dropping important packets. This would further raise transmission efficiency of the whole path. Basically, the mechanism relies on transferring NACK messages to carry notifications on discarding those packets.

5.2 Detecting useless packets

In order to avoid relaying useless packets, each forwarder needs to perform an observation mechanism in which it autonomously detects useless video packets. Studies investigating conventional active/programmable networking devices

propose to identify useless video packets/frames basically by means of locating dependent packets upon discarding I or P frames. In this approach, active routers or programmable store-and-forward devices (e.g. proxies, wireless base stations, etc), which understand packet semantics, mark the discarding of an important packet/frame and watch for any dependent packets. In [4][50], the authors propose to selectively relay a pattern of packets in accordance with the residual bandwidth of the outlink. Among selected video frames is no frame that is dependent on an excluded frame. In addition to such a semantics-based observation, we implement a temporal marking scheme which helps the intermediate node autonomously identify stale packets.

- *Definition 1*: **An orphan packet** is the one that depends on a lost frame within the same group of picture. Orphan packets are either of dependent frames, i.e. P or B.

- *Definition 2*: A packet arriving or residing at a certain node is called **stale** if it would not arrive at the receiver node in time once forwarded. Practically, such a packet might have been generated earlier than the obsolete bound (see Section 4.5.2) or it resides too long at the node.

Similarly to the selective discarding model in wired active networks, orphan packets are easily located at any node. Given the side information adhered to video packets, a relaying node can keep track of GoP identification of the video packet flow. Any time an important frame is discarded, it records the identification number so that later dependent packets can be determined. Nevertheless, to know if a packet is really stale is not trivial. Theoretically, if all the ad hoc nodes including the sender are synchronized to the same clock, the sender, upon generating every packet, can add the time stamp which later helps relaying nodes autonomously determine when it is stale. In reality, however, asking for clock synchronization among different ad hoc nodes is unrealistic in most of cases, particularly with an accuracy that the processing of short life-time video packets expects. Nonetheless, given that video packets may be retransmitted repeatedly when the receiver requests and that congestion takes place frequently, it is still worth enough for a relaying node to know how much time more, optimistically, a packet remains unexpired (optimistic time-to-live d^{ttl}). In principle, should the node find that a packet has an optimistic

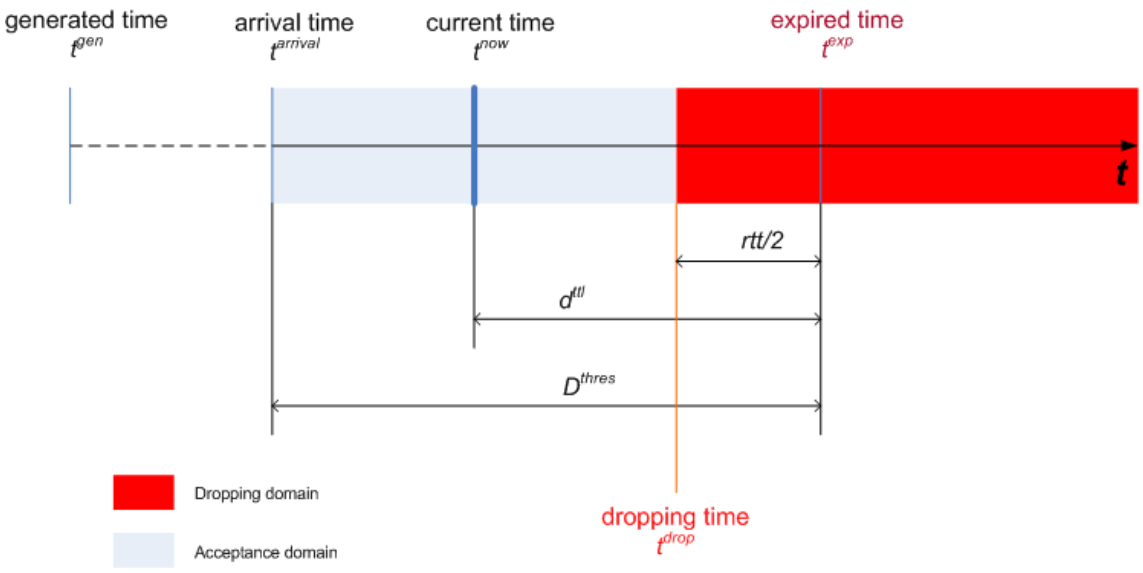

Figure 5.1: Autonomous estimation of time points at relaying nodes. Upon a packet arrives, the RtME can determine the dropping domain, starting from time point t^{drop}.

time-to-live no greater than zero, it can discard the packet without any concern, since it must certainly have been obsolete. As introduced in Chapter 4, we realize a simple packet timing watch at intermediate nodes as follows. Each intermediate node records the time when a video packet arrives for the first time; this time is called "arrival time" ($t^{arrival}$). Since then, any time the node wants to know the residing time of the packet, it simply subtracts the arrival time from the current time. Actually, there is a possibility that a node wrongly translates a subsequent arrival of a well stale packet as its arrival time. This happens when the true arrival time of this packet at the node is so long time ago that it has completely been destroyed from the warehouse. Namely, information about the first arrival of the packet is no longer kept at the node. Occurrence of this phenomenon, nevertheless can be diminished by an appropriate control in issuing ARQ requests at the receiver node. Appendix Chapter B details this mechanism.

Figure 5.1 illustrates the packet timing watch that relaying nodes perform. Right after the packet is accepted, the RtME can autonomously estimate critical time points, including the time from which on the packet should be dropped (drop-

ping time t^{drop}) and the time that the packet surely becomes expired (t^{exp}):

$$t^{exp} = t^{arrival} + D^{thres} \tag{5.1}$$

$$t^{drop} = t^{exp} - rtt/2 \tag{5.2}$$

Theoretically, if the node knows the round-trip-time toward the sender node, it can also infer the time that the packet was generated (t^{gen}). However, we do not consider applying such a mechanism to estimate that round-trip-time at the current study. Assume that at time t^{now}, the RtME refers to the packet, the optimistic time-to-live is determined as $d^{ttl} = t^{exp} - t^{now}$. If this time duration is no greater than $rtt/2$, the packet now falls in the dropping domain (the red area in Figure 5.1) and hence is classified as useless. As implemented in the current RtME design, the time when the node refers to the packet timing information can be at its acceptance or at dequeuing. The former case occurs when the packet arrives at the node again as a retransmitted version. Should it come at the dropping domain, it is considered useless. Actually, the latter case has been presented in Section 4.5.2, which is the very function that checks if a selected packet has enough remaining time-to-live (4.3).

Briefly speaking, in the following cases, packets will be classified as useless ones:

1. Packets that become orphan when an important frame is dropped. Previous studies on active network devices solely took this type of packets into considerations. Note however that dropping decisions are made at each node in a solitary manner. Beyond that, we propose a joint discarding mechanism in which orphan packets are resolved more thoroughly thanks to notifications among relaying nodes.

2. Packets that remain too long in the cache of a particular node. Upon a first packet of a frame arrives, the node marks the arrival time of the frame. Later on, the RtME at this node can estimate the residing time of packets of this frame at any time. Once the node refers to the packets and realizes that the dropping time t^{drop} has passed, they will be considered stale. Chapter 6 will discuss this in detail.

3. Packets that have the identification number no greater than the obsolete bound. As presented in Chapter 4, each relaying node maintains this bound to filter out obsolete packets. Note that the node usually does not store any information about these packets any more; they have been removed from the warehouse. It is the obsolete bound, rather than the recorded arrival time, that helps the RtME recognize their staleness.

4. Packets that are received again. If some packet has been available at the warehouse and another instance arrives, the packet is classified as useless. However, it may be logically enqueued to the fresh queue. The next section will explain this behavior in detail.

When a packet arrives at a node for the first time, the RtME checks its identification numbers. Should it be neither obsolete nor orphan, the packet will be admitted. This means that useful packets are not filtered out. Otherwise, they are considered meaningless. In the following sections, we will specifically analyse situations in which packets are discarded. Basically, packets that have been classified as useless will be subject to rejection. However, what the RtME does after discarding is not identical from packet to packet.

5.3 Local discarding

Activation of the RtME module allows to autonomously check the validity of arriving packets. In general, packets that are found to be useless will be prevented from being forwarded. Local dropping is executed in the cooperation of entity Rx and Tx and happens either when a packet arrives or when it is dequeued from one of the two queues (fresh queue and response queue). A packet that is accepted upon its arrival is not necessarily forwarded when the channel is available. Specifically, during congestion periods, packets are not instantly forwarded after their arrival. Even being enqueued, a packet may be discarded at the dequeuing time if it is detected as useless.

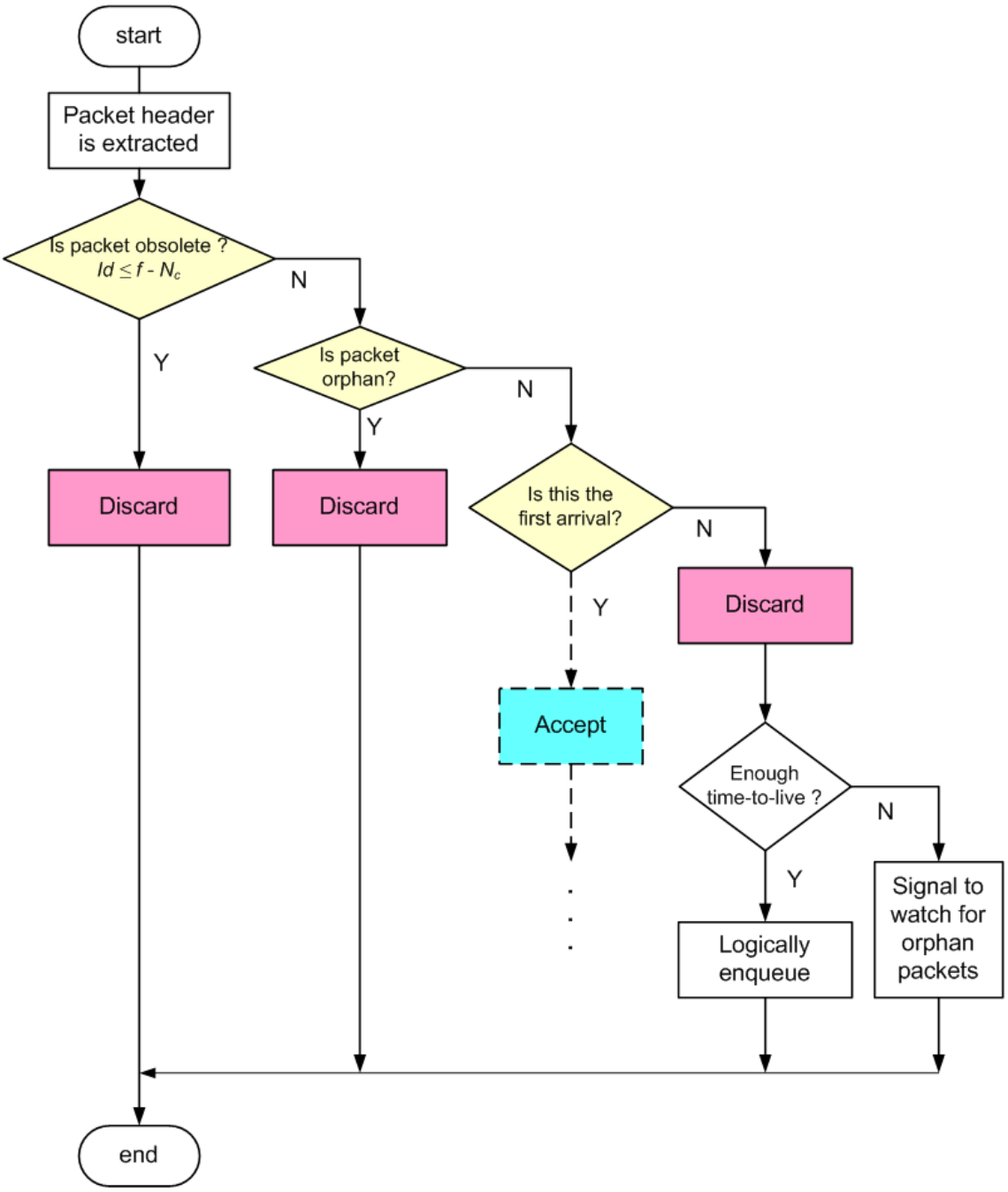

Figure 5.2: Local dropping cases. Packets might be discarded since they are obsolete, orphan, or arrive again.

5.3.1 Rejection upon arrival

Upon arriving at the node, a packet may be rejected by entity Rx for several reasons. Figure 5.2 shows the cases in which packets are to be discarded. In the first case, a packet is found useless since it belongs to a frame with the identification less than the obsolete bound. The packet itself (and its encompassing frame) should be destroyed from the cache if any. Basically, the RtME in this case does not proceed to any further processing. This case occurs when upstream links get worse. The second case of dropping happens when a packet is orphan. Namely, it depends on some other packet that this node has already dropped. After dropping the packet, the RtME does not proceed to any further processing, too. In essence, this case is similar to the dropping mechanisms that have been proposed to active networks [7][41][43].

The last case of dropping takes place if the packet arrives again while it is already in the warehouse. Obviously, only retransmitted packets fall in this case. The packet in this case is physically dropped while its previous version in the cache and the queues if any remains untouched. Nonetheless, the packet might be *logically* enqueued to the fresh queue if it has enough optimistic time-to-live (d^{ttl}). This case happens when the packet was lost somewhere downstream. While this dropping case is possible, it rarely occurs. The reason is that the request for retransmission of this packet would have *likely* been processed at this node (cache-hit); in the NACK message, the byte corresponding to this packet should have accordingly been cut off, i.e. the request would not go upstream. Furthermore, recall that packets are only forwarded from a node if it has not been sent within a period of round-trip-time to the receiver node (see Section 4.5.2). A possible situation of this dropping case is when the upstream link suddenly deteriorates, resulting in massive retransmission in the link layer. Imagine, a NACK message carrying a request for some lost packet arrives at the node right before the arrival of the packet. The request cannot be processed at the node since the delivery of the packet is still underway. Later, on handling the request, the upstream host resends the packet. The packet arrives at the node while its previous instance might also be successfully obtained.

In the last dropping case, if the packet does not have enough time-to-live, the RtME remembers to block subsequent packets of the same frame. These packets should be classified as useless if they arrive, since their arrival definitely falls in the

dropping domain. Remarkably, if this is a packet of an important frame, entity Rx is signaled to watch for future dependent packets.

5.3.2 Rejection upon forwarding

As stated before, being accepted at Rx does not mean that a packet will be forwarded. At the dequeuing time, Tx may decide not to forward the packet due to numerous reasons. Figure 5.3 shows how entity Tx discards a video packet. Firstly, when the downstream link gets worse, packets may experience a long waiting time. This may happen when nodes are moving far away from each other, raising the path loss, or when the channel is lossy due to some reason (e.g. fading), which increases the retransmission frequency in the link layer, the effective bandwidth is therefore reduced [27][63]. After waiting too long in the queue, the packet may not have enough time-to-live any more, or even get expired.

Secondly, because a particular packet can be requested repeatedly, especially when severe congestion occurs, it might logically be enqueued multiple times within a short moment. If all these instances are processed equally, the packet may be forwarded more times than actually needed; namely, multiple copies of the same packet uselessly reach the receiver node. This does not only waste energy and bandwidth, but also intensifies congestion. Therefore, any selected packet is additionally checked if it has just been forwarded recently before being forwarded. After forwarding the packet, it takes at least one round-trip-time (to the receiver node) for this relaying node to receive another legitimate request. Therefore, we set the time window to a period of round-trip-time, i.e., the node refrains from relaying the packet again if it has been sent out within last rtt time units. This has been stated in Chapter 4, where how a relaying node makes forwarding decisions is described.

In a very rare case, the downstream link is severely bad, making one or both queues excessively long, and some enqueued packets may get obsolete before being read out. The packet selection function, as presented in Chapter 4, should drop out such packets. Note that those packets should have been removed from the warehouse anyway. Any attempt to re-forward a non-cached packet will break down the operation of RtME.

As described previously in Section 4.5, entity Tx is only activated when there are packets to forward downstream, which is controlled by the semaphore. Specifi-

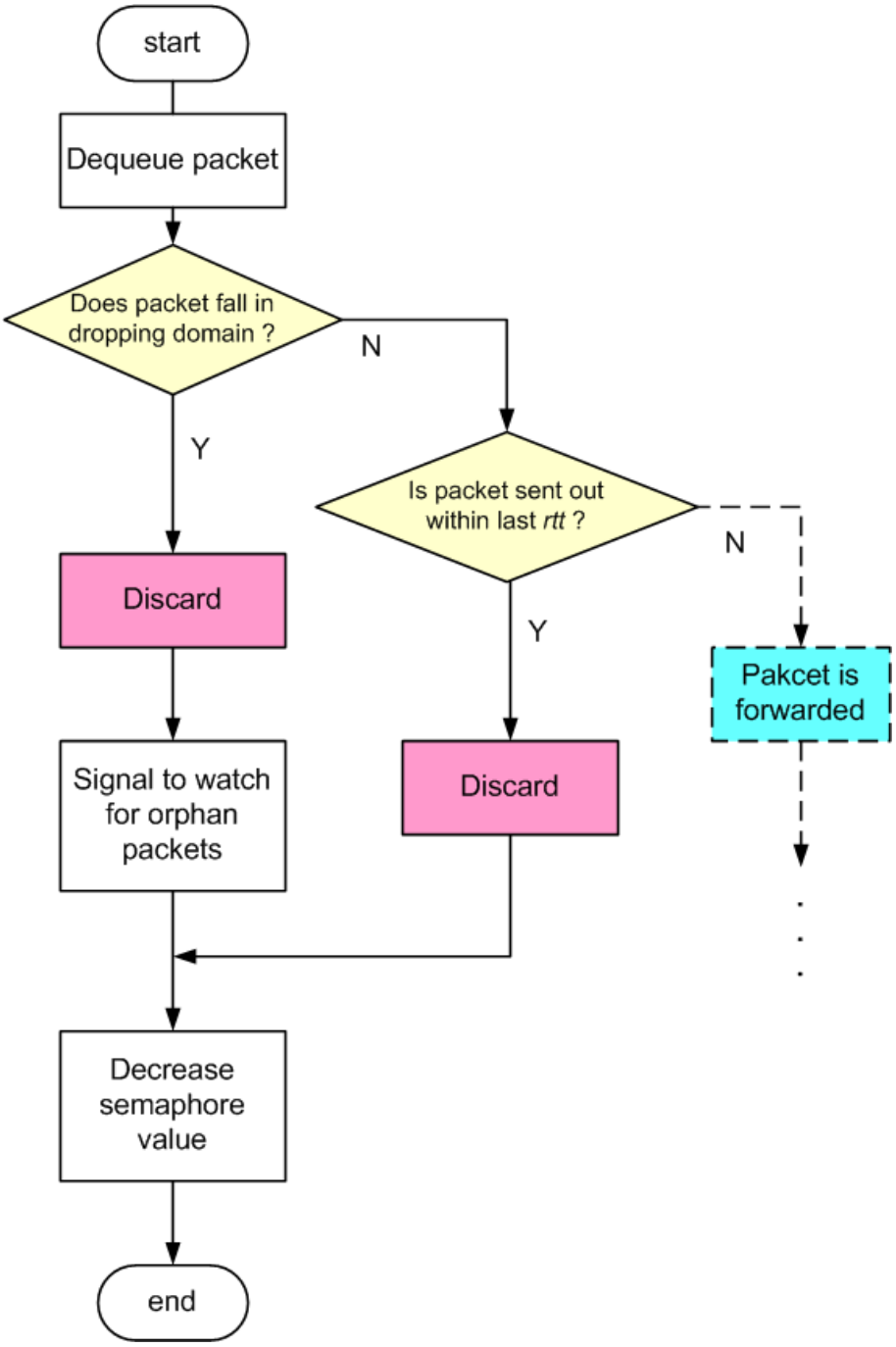

Figure 5.3: Discarding packet at forwarding time. Packets are discarded if they fall in the dropping domain or have just been sent out.

cally, whenever a packet is logically inserted to one of the queues, the value of the semaphore is added one unit. In parallel, any time a packet is dequeued, Tx subtracts one unit from the semaphore. This is done even when the packet is discarded, as shown in Figure 5.3. Otherwise, Tx might be uselessly awoked, even there is no waiting packet to relay.

5.4 Joint discarding mechanism

Imagine, an I frame has arrived at some relaying node. Later on, before it leaves, the node realizes that packets of the frame do not have enough optimistic time-to-live anymore. As stated in the previous section, they will not be forwarded. At this time, it is likely that many packets of subsequent frames have also been sent out from the sender, but due to congestion, they are still somewhere upstream. As the node decides to drop out the I frame, those dependent frames automatically become useless. Obviously, should this node actively notify other upstream nodes where those frames are still in cache, the benefit will be considerable. There are of course other cases that notifying may also help nodes jointly detect useless packets, for instance, when the very entity Rx drops an I or P frame.

5.4.1 Notifications on discarding important packets

When an important packet is intentionally discarded at a node, other relaying nodes should be notified as soon as possible. This would help stop forwarding useless packets immediately, and potentially more useless packets would be identified. To do so, the node should generate and distribute an instant message. Note however that communication cost in term of energy and bandwidth for every message is considerable as stated in Chapter 4. Fortunately, dropping of important packets practically coincides with burst of lost packets, which in turn leads to the communication of NACK message(s). To avoid overhead, we propose to take up advantage of NACK messages themselves to carry notifications on suppression of important packets.

Basically, along the communication path between the sender and the receiver, the forth direction is more busy than the opposite, unless the video session is bidirectional and the two video flows are conveyed over the same physical path. The

load of transmitting NACK messages over the back direction is normally not high compared to that of video packets, since NACK messages themselves are not heavy and the receiver node does not acknowledge successfully arriving packets. Consequently, notifications can be relayed among intermediate nodes easier than video packets. However, we do not propose to generate messages that are exclusively used for notifying upstream nodes. Instead, RtME at relaying nodes adds notifications to NACK messages upon dropping important frames, due to three reasons below.

1. Firstly, there is a trade-off between the efficiency of notifications and the overhead derived if they are carried over exclusive messages. Note that while a notification potentially helps upstream nodes stop forwarding more useless packets, its effect is not certain, due to the unpredictability in the nature of the ad hoc wireless network. Using exclusive messages may detect more useless packets, but there exists a risk if the load of those notifications outweighs that of would-be identified packets.

2. Secondly, once relaying nodes decide to discard important packets, it is much likely that congestion over the path is major. In such a case, NACK messages will be generated quickly. This means that exploiting NACK messages to notify upstream nodes is not a tardy solution.

3. Thirdly, unlike incoming and cache-hit video packets, any NACK message arriving at a forwarding node is processed and forwarded upstream instantly if necessary, rather than being enqueued.

Figure 5.4 illustrates the overview of the joint discarding mechanism. In the current implementation of the mechanism, we add two bytes to NACK messages for carrying any notification on dropping of important frame. They are to indicate the identification number of the rejected frame. Upon receiving a NACK message, the relaying node extracts its header to obtain the identification number of the packet that the receiver node has just successfully received. Then, rejection of important frame if any is identified based on these identification numbers.

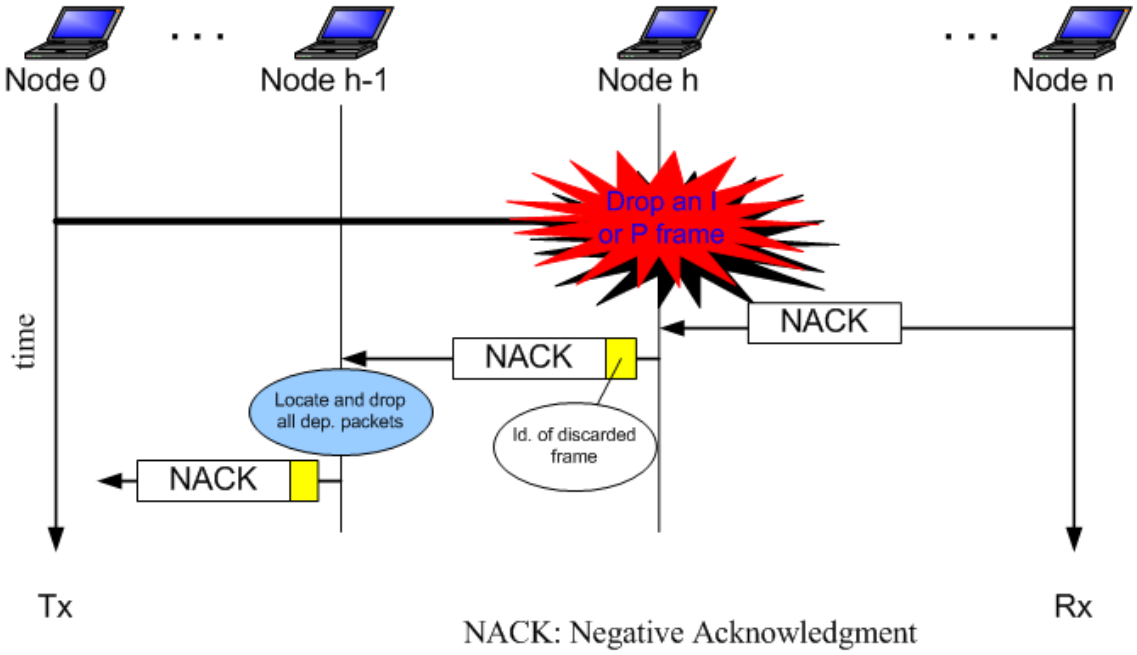

Figure 5.4: Operation of the joint discarding mechanism. Notifications on discarding important packets are carried over NACK messages.

5.4.2 Joint operations

To mark the event of dropping important frames, each relaying node maintains a so-called *signal record*. This record simply holds the identification of the discarded frames, including the group of picture they belong to. Updating the record might be made either when the node decides to drop an important frame, or when it receives a notification from downstream node. Note that the content of this record also influences the locally dropping function presented in the previous section. Specifically, both entities Rx and Tx refer to the record to detect orphan packets at their arrival and when they are about to be forwarded, respectively.

An important frame (either I or P) is called more *critical* than another if the former has more dependent packets than the other. Obviously, the closer to the I frame within the same group of picture, the more critical an important frame is. In reality, updating the signal record is only carried out if the node drops out a more critical frame, or it receives a notification indicating that a more critical frame has been discarded somewhere downstream. In an unpopular case, the node may observe a discard of two or more important frames that are not of the same group

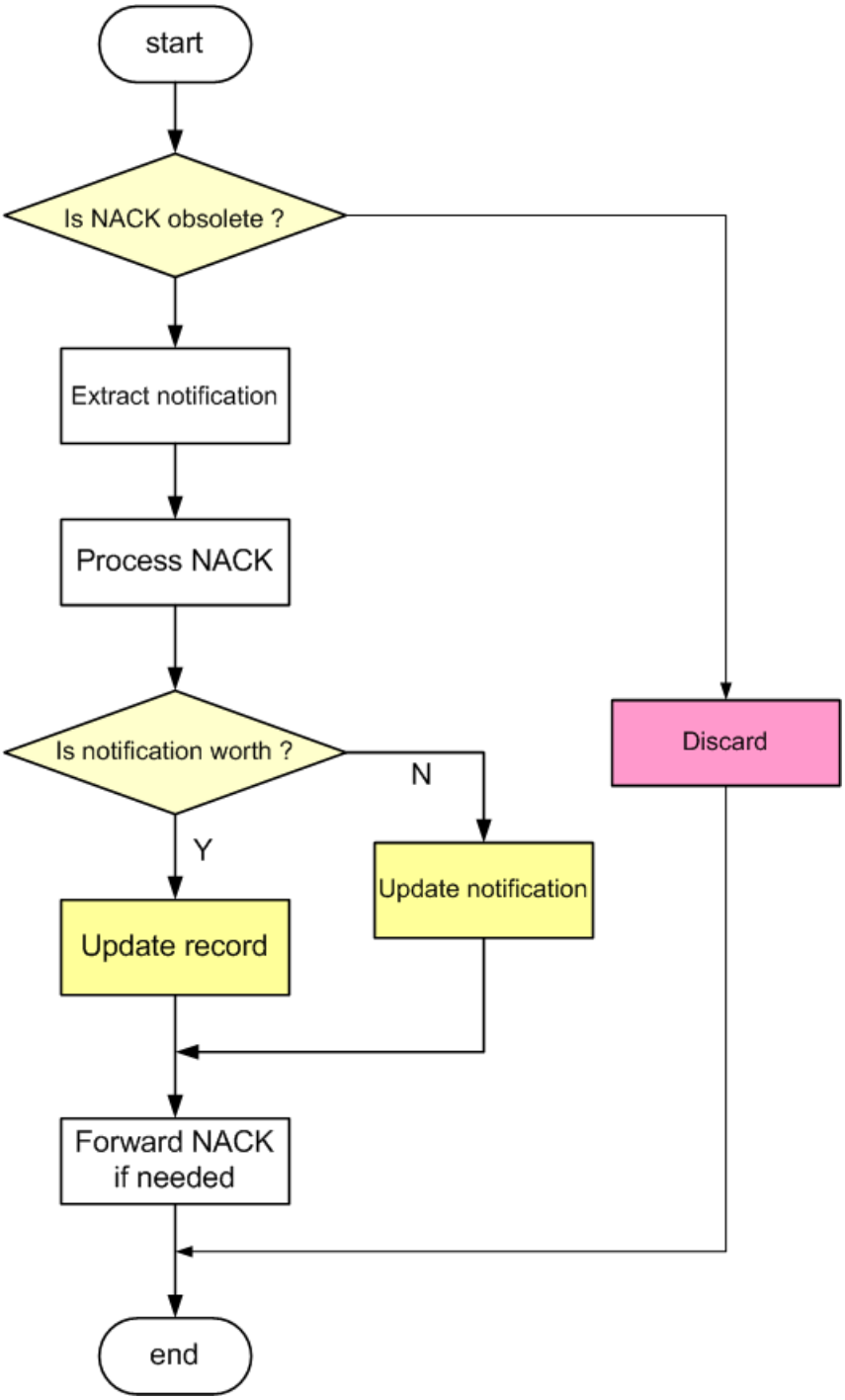

Figure 5.5: Processing notification attached to NACK message. There may be an information exchange between the signal record and the notification.

of picture. If so, the record retains identification of them all until they get obsolete.

Figure 5.5 illustrates the operation of the joint discarding mechanism when a NACK message is received. As presented in Section 4.4, if the just received packet (or the expected packet to be received) is determined to be obsolete, the NACK message will be discarded without any further processing. Otherwise, after extracting the notification and interpreting the list of requested packets (see Chapter 4), the RtME at this node decides to update either the signal record or the notification attached to the NACK message. The former case occurs when the notification indicates a more critical frame. Meanwhile, the latter happens when this node has discarded a more critical frame. In a rare case, dropping of a frame of another group of picture may be marked, and hence is additionally written to the record.

The presence of the signal record at each relaying node results in the fact that how orphan packets are detected is transparent to dropping decisions. Dropping of important frames at either this node or some other node downstream causes the same effect: the signal record is updated. Decisions to discard associated orphan packets are made based on referring to the content of the record, regardless of how it is created (locally or jointly).

5.5 Efficiency analysis

Detecting and dropping useless packets does release intermediate nodes from relaying packets that do not contribute to the decoding process at the receiver node. This brings up not only energy saving but also bandwidth for other useful packets. In Chapter 4, we already know the benefits regarding success rate and energy from the retransmission point of view. Now that the comprehensive dropping mechanism has been presented, let's consider its effect concerning energy consumption and bandwidth.

5.5.1 Energy saving

Once detecting that a packet p is not meaningful to transmit anymore, the node will discard it decisively. This brings up an energy saving for itself and for the downstream nodes, since they do not have to relay the packet. The cumulative amount can accordingly be calculated as follows (refer to Table 5.1 for denotation

Table 5.1: Parameter definition

$E^i_{tx}(p)$	energy consumed on transmitting packet p at node i
$E_{rx}(p)$	energy consumed on receiving packet p
E^i_{rts-tx}	energy consumed on transmitting RTS from node i
E_{rts-rx}	energy consumed on receiving RTS
E^{i+1}_{cts-tx}	energy consumed on transmitting CTS from node $i+1$
E_{cts-rx}	energy consumed on receiving CTS
E^{i+1}_{ack-tx}	energy consumed on transmitting an acknowledgment from node $i+1$
E_{ack-rx}	energy consumed on receiving an acknowledgment
h	node that cache-hit occurs
i, j	generic indices
k_i	the number of nodes under the coverage of node i
k^b_i n	the number of nodes along the transmission path
p	packet under consideration
Ω	set of dependent packets of p
ΔE^{dis}	cumulative amount saved when a single packet is dropped
ΔE^{Ω}	cumulative amount saved when a cluster of packets are dropped

of the symbols).

$$\Delta E^{dis} = \sum_{i=h}^{n-1} [E^i_{tx}(p) + k_i \times E_{rx}(p)] \tag{5.3}$$

Let's assume that, as soon as packet p is determined to be stale, the whole encompassing frame should be dropped. Detection of stale packets have been explained in Section 5.2. After discarding the frame, the node keeps watching for future dependent packets if packet p is of an important frame (e.g. I or P). Those useless packets should immediately be rejected upon their arrival. Once the frame is followed by subsequent dependent ones (denoted as set Ω), the node notifies upstream devices of the discard. After being notified, these nodes may jointly drop useless packets if any. This would save another amount of energy:

$$\Delta E^{\Omega} = \sum_{j \in \Omega} \sum_{i=h_j}^{n-1} [E_{tx}^{i}(j) + k_i \times E_{rx}(j)] \tag{5.4}$$

where h_j is the node that detects and drops dependent packet j.

Should RTS/CTS and/or acknowledgment be adopted in the link layer, the values estimated in (5.3) and (5.4) are even higher since more control packets are exchanged upon transmission of each video packet [37]. Note that transmitting these overhead packets does not only consume more power, but also lowers the effective bandwidth. This is particularly clear if nodes are densely distributed. As the four-way handshake RTS-CTS-DATA-ACK [37] is used, in a network of k nodes where transmission range of each node covers all the other nodes, relaying a packet from a node to another consumes the following amount [37]:

$$\begin{aligned} & E_{rts-tx} + (k-1) \times E_{rts-rx} + E_{cts-tx} + (k-1) \times E_{cts-rx} \\ + \quad & E_{tx}(p) + (k-1) \times E_{rx}(p) + E_{ack-tx} + (k-1) \times E_{ack-rx} \end{aligned}$$

where E_{rts-tx} and E_{rts-rx} are the energy consumption respectively for transmitting and receiving a *request-to-send* message; E_{cts-tx} and E_{cts-rx} are those values associated with a *clear-to-send* message; E_{ack-tx} and E_{ack-rx} represent those values for sending and receiving an acknowledgment message.

In such a case, when a packet p is discarded, the reduction of cumulative consumption of the path can be estimated as follows.

$$\begin{aligned} \Delta E_{four-way}^{dis} \quad = \quad & \sum_{i=h}^{n-1} [E_{rts-tx}^{i} + k_i \times E_{rts-rx} \\ & + \quad E_{cts-tx}^{i+1} + k_{i+1} \times E_{cts-rx} + E_{tx}^{i}(p) + k_i \times E_{rx}(p) \\ & + \quad E_{ack-tx}^{i+1} + k_{i+1} \times E_{ack-rx}] \end{aligned} \tag{5.5}$$

In (5.5), i - subscripted parameters are associated with hop i that connects node $i-1$ to node i; subscripts $rts-tx$ and $rts-rx$ refer to *transmit* and *receive* of a request-to-send message; subscripts $cts-tx$ and $cts-rx$ refer to *transmit* and *receive* of a clear-to-send message; subscripts $ack-tx$ and $ack-rx$ refer to *transmit* and *receive* of a acknowledgment message.

Similarly, the energy saving when the joint discarding mechanism removes orphan packets from the transmission path is approximately calculated below.

$$\Delta E_{four-way}^{\Omega} \quad = \quad \sum_{j \in \Omega} \sum_{i=h_j}^{n-1} [E_{rts-tx}^{i} + k_i \times E_{rts-rx} + E_{cts-tx}^{i+1}$$

$$+ \quad k_{i+1} \times E_{cts-rx} + E^{i}_{tx}(j) + k_i \times E_{rx}(j)$$
$$+ \quad E^{i+1}_{ack-tx} + k_{i+1} \times E_{ack-rx}] \tag{5.6}$$

5.5.2 Bandwidth saving

Basically, bandwidth saving can be estimated similarly to what we have done with energy consumption. Should a node h detect and stop forwarding a useless packet p, the path observes a saving of L_p in bandwidth of all the $(n - h)$ hops down to the receiver from this relaying node. In case the four-way handshake is applied into the link layer, the number of RTS, CTS, and acknowledgment messages is reduced each by $(n - h)$.

When all the video packets of set Ω are detected and removed, the saving amount on the forth direction can be expressed as follows. For each packet $j \in \Omega$ detected at node h_j, all the $(n - h_j)$ downstream hops are free from the relaying. At the same time, the the number of RTS, CTS, and acknowledgment messages is reduced each by:

$$\Delta M^{\Omega} = \sum_{j \in \Omega} (n - h_j) \tag{5.7}$$

Additionally, avoiding forwarding a packet/message from a particular node may release resources of its multiple neighborhoods. Therefore, the actual efficiency is even more considerable.

Note that the above calculations are to clarify the benefits of the discarding mechanism, not to find a mandatory solution for identifying packets. In reality, relaying nodes do not estimate any saving amount on making forwarding decisions. To locate and drop useless packets, each intermediate host just needs to check their timing information and some identification numbers. The complexity of the presented approach is therefore inconsiderable compared to other strategies, e.g. the RaDiO-oriented selection algorithm in [4].

5.6 Chapter review

Retaining video packets at relaying nodes does not only allow them to smartly select best packets, but also creates a chance to detect useless packets. We have presented a comprehensive discarding strategy that helps intermediate nodes thor-

oughly stop forwarding packets. NACK messages are exploited to carry dropping notifications among relaying nodes so that they jointly detect orphan packets. The introduction of the signal record does make dropping decisions more simple.

Chapter 6
IMPLEMENTATION

Up to now the architecture of the proactive framework has been presented. In this chapter, we report the realization of RtME (Real-time Media Engine) that can be seen as a plugin-like module dealing with video packets. As stated in previous chapters, this module includes several entities: Rx, Tx, NACK Handler, and Warehouse. Except the warehouse, all the entities are implemented as parallel threads synchronized using a special semaphore. This chapter presents key functions of the entities from the code designing point of view. Within the scope of this dissertation, we do not intend to go through all the source code in a statement-by-statement manner. Instead, only system design-related aspects are analysed at an abstract coding level.

6.1 Overview

Basically, the realization of the framework lies in RtME (Real-time Media Engine) as stated in Chapter 4. From the programming point of view, among four aforementioned entities that make up RtME three are *functional threads*. The remainder is a "static submodule" that is responsible for maintaining the transient cache. The reason behind this parallel fashion is that the functions of receiving video packets, forwarding them, and processing NACK messages are independent of each other; note that arrivals of video packets and NACK messages are highly unpredictable. Furthermore, as already explained in Section 4.1, the adoption of the multithreading architecture is to satisfy the need of sharing data among the functions of RtME. Figure 6.1 depicts the coding design of RtME, including units of the functional threads and their interactivity. These threads implement the operations of entities Tx, Rx, and NACK Handler.

- **Thread Rx** is composed of *Validation* and *Caching & Queuing* units. The former is to control admission of video packets arriving at this node while the latter, as it is named, performs the caching and queuing functions on accepted packets. When no packet arrives, this thread is blocked by a function call that

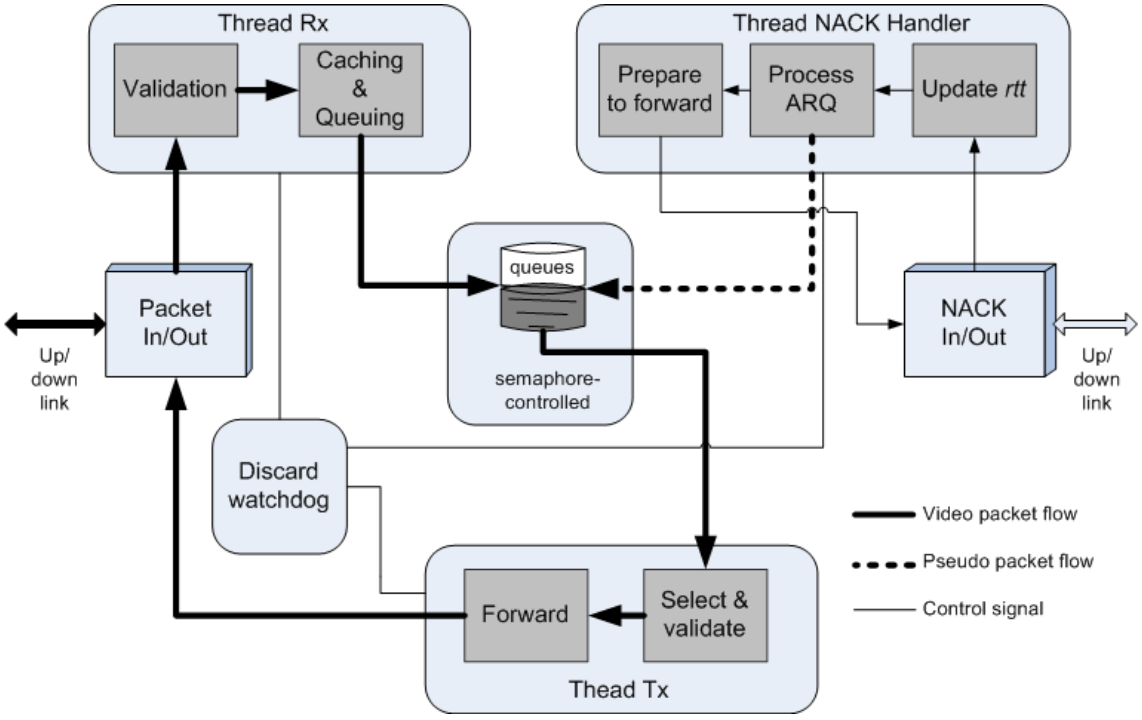

Figure 6.1: Abstract coding digram. Three threads are implemented to process video packets and NACK messages.

is responsible for capturing packets.

- **Thread NACK Handler** awaits NACK messages that carry retransmission requests. As a valid NACK message arrives, unit *Update rtt* extracts the header to identify the just successfully received packet and then calculates the current round-trip-time to the receiver if applicable. Subsequently, a list of requested packets is interpreted by unit *Process ARQ*; cache-hit packets derived form a *pseudo packet flow* to one of the two queues. As explained in Section 4.4, those packets are not physically originating from the NACK Handler. Indeed, the formation of the flow stems from re-inserting the header of the packets into the response queue. Finally, unit *Prepare to forward* decides whether to forward a NACK message upstream; it cuts off the processed bytes (see Chapter 4) and integrates discarding notifications into the NACK message before relaying (see Chapter 5). At idle time, this thread is blocked by another function call waiting for NACK messages.

- **Thread Tx** consisting of units *Select & validate* and *Forward*, once activated

by the semaphore, selects the most suitable packet from the waiting packets in the queues for considering transmission downstream. If not signaled by unit *Discard watchdog*, the packet is sent out.

The discarding mechanism as presented in Chapter 5 is executed with the assistance of a *discard watchdog* that holds the signal record. This abstract unit interacts with all the above threads, taking care of detecting and dropping useless packets. To the side of the NACK Handler thread, it cooperates to add discarding notifications on important packets.

6.2 Data organization

The core object that RtME processes is video packets. They are both the input and the output of the module. In the conventional passive forwarding framework, packets are treated independently of each other. They are simply forwarded at a pure best-effort. Differently, the proactive strategy keeps in mind the data dependency among video frames and the tightly coupled relations among packets of the same frame. In addition to data objects, we need functional items (cache and queue) to store and maintain packets, either physically or logically. Basic data items that RtME employs include *video frame*, *video packet*, *logic queues*, and *frame warehouse*. Figure 6.2 illustrates the relations among these items.

- **Video frame**. Most studies on minimizing distortion [4][6][23] consider frames as elementary objects of the optimization problem. In video data representation, a frame is the middle point between the truly independent data unit - group of picture - and the "networked" unit - packet. As shown in Figure 6.2, a video frame is defined in data type `struct MPEGFrame`, containing pointer `segments_ptr` that points to the array of its packets.

- **Video packet**. Each video frame may be composed of multiple packets that are actually transmitted over the network each time the channel is available. In our implementation, packets are declared as elements of data type `struct Segment` as seen in Figure 6.2. They do not exist as stand-alone items. Upon acceptance, each packet is logically integrated into its frame.

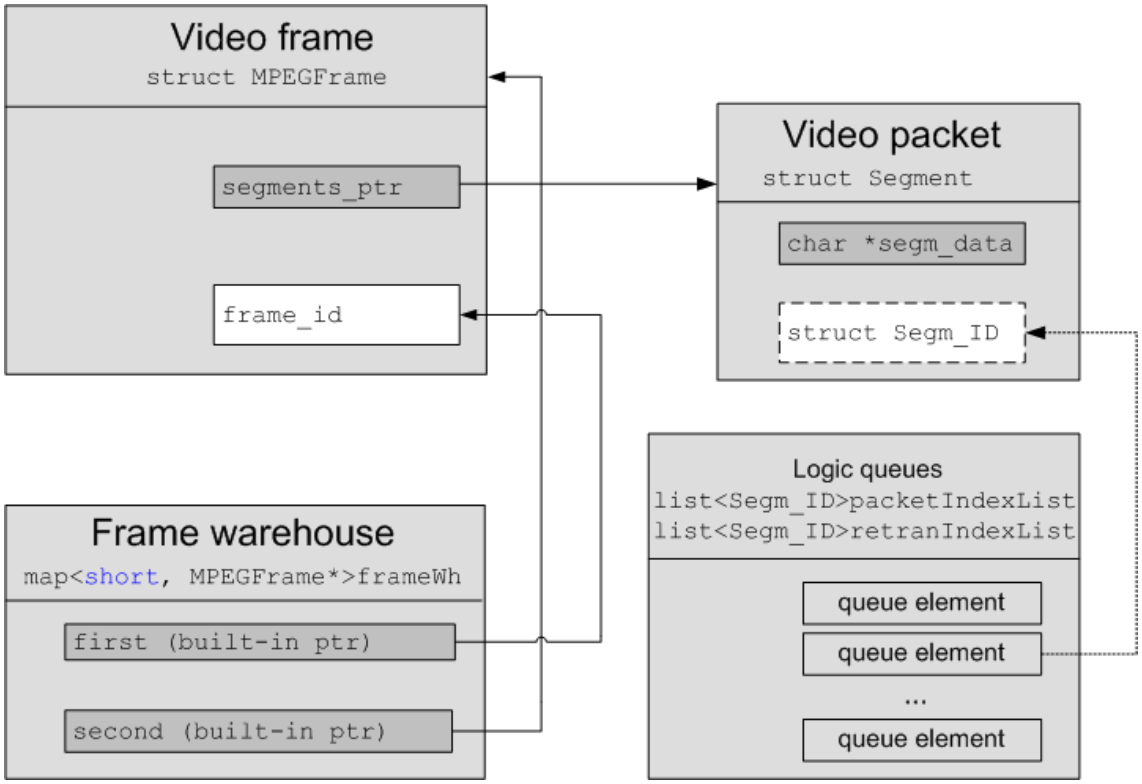

Figure 6.2: Relation among data items. Besides items for storing video data units, the cache and the queues are defined as collective data types.

- **Queue** and **queue element**. The fresh queue and the response queue are defined as *list* items, indicated in Figure 6.2 as `list<Segm_ID>packetIndexList` and `list<Segm_ID>retranIndexList`, respectively. Each queue element is an item of data type `struct Segm_ID` that is composed of a set of identification numbers of the packet, the encompassing frame, and the group of picture. These numbers help entity Tx to later fetch the packet. We choose data type *list* for the queues because of its simplicity; it does not require packets to be stored contiguously. Note that video packets are not physically enqueued, but only their header information that is defined by data type `struct Segm_ID`. This is to eliminate any packet replication inside the node and to fasten packet searching. The queues simply employ the FIFO fashion, which further lowers the complexity of processing video packets as well as NACK messages.

- **Frame warehouse**. Video frames are stored collectively inside the warehouse. They are logically bundled up into a *map* that is shown in Figure 6.2 as

`map<short, MPEGFrame*>frameWh`. The first pointer of each *map* element is actually the identification number of the frame (`frame_id`) while the second pointer refers to the video frame. Under this relation, video frames can be quickly searched from the warehouse, decreasing the computation load each time a packet is selected.

There exists a hard tie between a frame and their packets. Within the frame, packets share numerous items of header information, such as identification number of the frame, the group of picture, timing data, etc. As indicated in Figure 6.2, `struct Segm_ID` partially defines these very items. Subsequent sections will detail the role of each item and further clarify their tie.

6.3 Proactive encapsulation for video packets

At the sending side, a header has to be added to each video packet to identify itself within the video sequence. Main data are frame sequence number, frame type, packet count, and the segment number. Figure 4.2 lists these fields and their size. When a video packet is encapsulated for sending down to the transport layer, the fields are constructed as in the code below.

```
1:  header[0] = (unsigned char) frame_distortion; //currently not used
2:  header[1] = (unsigned char)((episode << 3) | (7 & frame_type));
3:  header[2] = (unsigned char) ((frame_pos << 3) | (segm_num & 7));
4:  header[3] = (unsigned char) (frame_gop << 4); //4 LSB bits is
5:                                                 // for segm_id later
```

To flexibly identify a packet from a video frame, we segment the frame sequence number into the identifier of group of picture, denoted as `gop_num`, the frame position identifier `frame_pos`, and the segment identification number of the packet `segm_num`. The next section will further detail this hierarchical identifying mechanism.

6.4 Processing packet arrivals

Upon arriving from upstream nodes, video packets are handled by thread Rx. Its main task is to extract the packet header, control packet admission, update obsolete/fresh bounds, and order to store the packet data to the warehouse if needed.

Memory for packets is allocated dynamically upon their arrivals. This does save storage space but complicates handling each packet individually. To simplify the process, we follow a frame-oriented fashion to maintain video packets. Namely, packets of the same frame are grouped into a single element of data type `struct MPEGFrame`. Note that their physical data might not be stored contiguously, since memory allocation is made dynamically on-the-fly. The list below shows the structure of a `MPEGFrame` element.

```
1:  struct MPEGFrame
2:  {
3:     char     *frame_data; /* only used for frames
4:    containing one packet */
5:     Segment  *segments_ptr;
6:     char     segm_num;
7:     char     header;
8:     char     frame_type; // 0.5 byte
9:     char     frame_gop; // 1 byte
10:    short    frame_id;
11:    short    episode;  // 0.5 byte
12:    unsigned int frame_distortion; // for future use
13:    unsigned int frame_size;
14:    unsigned int time_stamp; // millisecond
15:    unsigned int last_tx_stamp;
16:    char packetState;
17:    /* 0-initial (when new episode starts); 1-packet received;
18:    * 2-packet forwarded; 3-NACK received; 4-retransmitted;
19:    5-obsolete (deleted) */
20:    unsigned short rtt;
21: };
```

The total number of packets of a frame is represented in the definition as field `segm_num`. Upon receiving any packet of the frame, the relaying node gets to know how many packets were generated from this frame. However, the number of packets that actually arrive successfully at this node may be smaller because of loss

or route changes. To save memory, the RtME declares a pointer (`*segments_ptr`) to hold packets on-the-fly instead of initiating all `segm_num` packets and reserving `frame_size` bytes for the whole frame. When receiving the first packet of the frame, the time is recorded in field `time_stamp` and is translated as the arrival time of the frame. Later on, estimation of residing time of this frame will refer to this value.

Video packets are data items of data type `struct Segment` that looks as follows:

```
1:   struct Segment
2:   {
3:     char         *segm_data; /* hold packet physical data */
4:     short        segm_size; /* indicate the size of this packet */
5:     unsigned int last_tx_stamp; /* recording the last time stamp
6:                                    this packet was sent*/
7:     char         segm_state; /* indicate if this packet is sent
8:                                 out or NACKed */
9:   };
```

The first `char` pointer is to hold the physical body of the packet when it is eligible for dispatching to the warehouse. Once packet loss is detected, the receiver node may generate NACK messages where a particular packet might be requested repeatedly. As explained in Section 4.5, before sending out any packet, entity Tx should check if the packet has just been sent within *rtt* time units. This check is to reduce the possibility of receiving multiple copies of the same packet. In this need, `last_tx_stamp` helps control the time stamp of last *transmit* of the packet. The last field (`segm_state`) is used to indicate the status of the packet such as being sent out, being successfully received (indicated in the header of NACK message), etc.

As explained in the previous section, to ease the maintenance of the warehouse, we choose to use a *map container*, which is a collective data type that accelerates accessing any element. The container, declared as `map<short, MPEGFrame*>frameWh`, flexibly holds elements whose members include the frame sequence number and a pointer to the associated frame. The main maintenance tasks are adding, either partially or entirely, video frames to, and removing them completely from the ware-

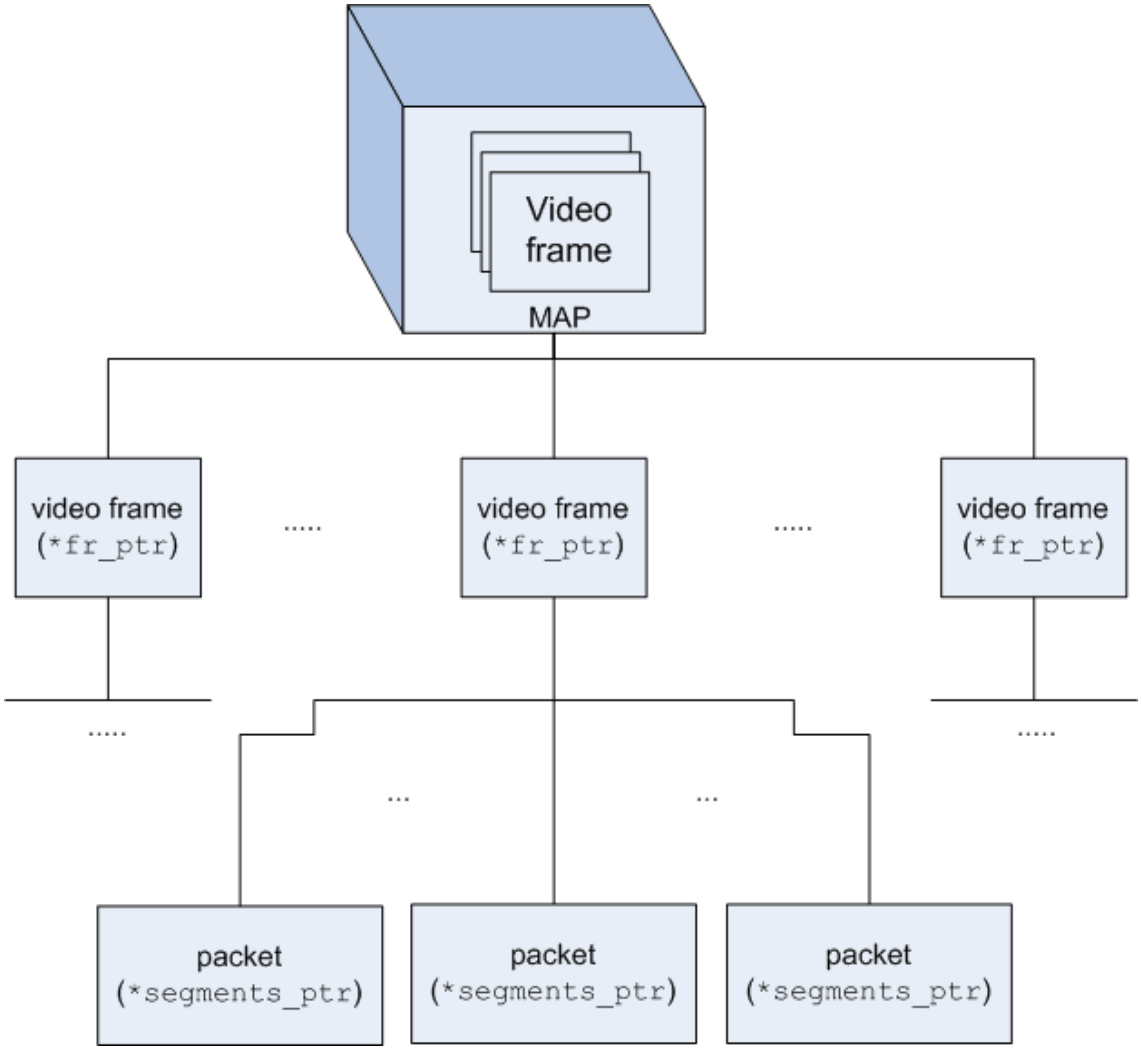

Figure 6.3: Hierarchical architecture of video packet management. Logically, the cache is realized as a map container that holds video frames. Each video frame is in turn a collection of packets.

house. Figure 6.3 illustrates the hierarchical management for video packets inside RtME.

At the sending side, the function of Rx is simplified; what it does is just receiving packets from the application layer and then adding header. For the RtME in relaying nodes to fully understand video packets, this entity must encapsulate them with a valid header. At the receiver node, the function is even more simple: admitting packets to the playback buffer and reordering them if necessary.

The structure of packet header has already been illustrated in Chapter 4 (Figure 4.2). It contains a handful of bytes added to indicate the identification of the packet, the video frame semantics and others. When receiving a packet, the RtME extracts the proactive header by simply calling the following function:

```
1:  void header_trans(unsigned short *dist, short *type, short *epi,
2:                    short *pos, short *gop, short bytes)
3:  {
4:    /* Distortion reduction associated with the encompassing frame
5:       - this will be used in future studies */
6:    *dist  = (unsigned short) RecvBuf[bytes-4];
7:    /* Video frame semantics : I, P, or B */
8:    *type  = (short)(RecvBuf[bytes-3] & 7);
9:    /* Episode - this field is currently used for identifying the
10:      cycle of the video sequence transmitted. In the real
11:      application, it can be used for other identifications,
12:      e.g. stream */
13:   *epi   = (short)(RecvBuf[bytes-3] >> 3);
14:   /* Position of the frame within the GoP */
15:   *pos   = (short) RecvBuf[bytes-2];
16:   /* Identification of the GoP */
17:   *gop   = (short) RecvBuf[bytes-1];
18: }
```

In reality, calling the function above will store the extracted header of the packet to internal local variables. From now on, any attempt to retrieve the packet will rely on these pieces of information. As explained later, queuing the packet is done over these values, rather than the physical body of the packet. An actual call

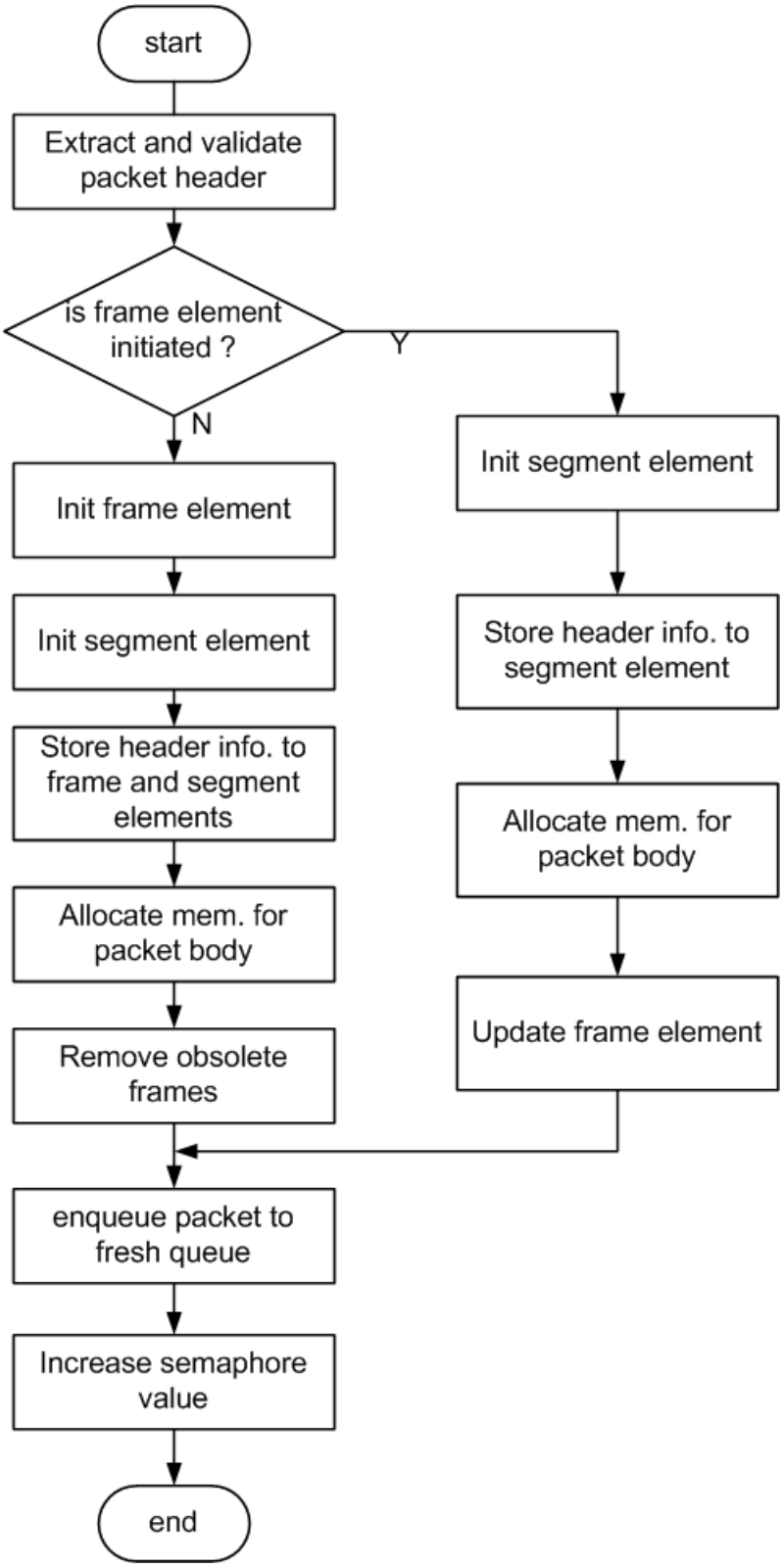

Figure 6.4: Operation of Rx in RtME. A packet can be of either a new frame or already initiated one.

of the function looks as follows:

```
header_trans(&distortion, &fr_type, &episode_num, &fr_pos, &fr_gop,
            &segm_id, &segm_num);
```

where the most remarkable items are `fr_type` that indicates the packet semantics; `fr_gop` that identifies the group of picture; `fr_pos` that positions the frame within the group of picture; and `segm_id` that shows the packet identification within the frame.

The operation of Rx is summarized in the flowchart of Figure 6.4. When a packet arrives, the header is first extracted and the packet is checked if it is valid. Should the packet belong to a frame generated earlier than the current obsolete bound, it will not be admitted. Basically, there are two possible situations once the packet is accepted. In the first case, the packet belongs to a completely new frame that is not available in the warehouse. The RtME needs to initiate a frame pointer (`*fr_ptr`) for this frame. The pointer for this packet is then created to hold the header information and the body of the packet as the list below:

```
1:   /* Initiate frame pointer */
2:   fr_ptr = (MPEGFrame*) malloc(sizeof(MPEGFrame));
3:   /* Store header information of the frame to the pointer*/
4:   fr_ptr->frame_type = fr_type;
5:   fr_ptr->frame_gop  = fr_gop;
6:   fr_ptr->segm_num   = segm_num;
7:   fr_ptr->frame_distortion = distortion;
8:   fr_ptr->time_stamp = TIME_NOW;
9:   /* Allocate memory for the packet body */
10:  if (is_frame) // a single-packet frame
11:  {
12:    fr_ptr->frame_size = bytesRecved;
13:    fr_ptr->frame_data = (char*)malloc(bytesRecved);
14:    memcpy(fr_ptr->frame_data, RecvBuf, bytesRecved);
15:  }
16:  else //a segment
17:  {
```

```
18:    segms  = (Segment*) malloc(segm_num * sizeof(Segment));
19:    segms[segm_id].segm_size = bytesRecved;
20:    segms[segm_id].segm_state = 1;
21:    memcpy(segms[segm_id].segm_data, RecvBuf, bytesRecved);
22:    fr_ptr->segments = segms;
23:  }
```

As stated above, video frames are collectively stored into a *map*. This new frame is added to the map accordingly. Before the addition, all the packets of obsolete frames should be cleared away from the cache. To lower the computation load, we just keep the number of frames fixed (equal to N_c); the clearance is done at pessimistic assumption by removing the oldest frame of the map each time a new one is pushed. This is executed simply as follows:

```
1:  if (frameWh.size() > N_C)
2:  {
3:    free(frameWh.begin()->second);
4:    frameWh.erase(frameWh.begin());
5:  }
6:  frameWh.insert(make_pair(frame_id, fr_ptr));
```

In the second case, the arriving packet is of some frame element that has already been initiated. The frame pointer is sought from the map rather than being created newly. Subsequently, what the RtME does is fairly similar. However, instead of removing obsolete frames, it updates relevant fields of the frame element indicating that another packet of the frame has been accepted.

Once the packet has allocated memory, it is logically enqueued to the fresh queue that is dedicated to packets incoming from upstream nodes (the fresh queue). Namely, the packet is not wholly added to the queue, but only some of its header information. The data to be enqueued is defined by the following structure:

```
1:  struct Segm_ID
2:  {
3:    short frame_id;
4:    char  segm_id;
```

```
5:    char  frame_type;
6:    char  frame_gop;
7:    char  frame_pos;
8:  };
```

Actually, we do not need so many pieces of information for dequeuing the packet later. However, to quickly access the map when the packet is sent out, two fields `frame_gop` and `frame_pos` are added. In order to speed up the packet selection by entity Tx (previously presented in Section 4.5.1), the frame semantics (`frame_type`) is also needed. Recall that which packet is selected depends on their frame semantics. In fact, enqueuing a packet is just a matter of adding the `Segm_ID` item to list `packetIndexList` that defines the fresh queue. Before dispatching the element to the queue, the RtME assigns the extracted header information to the fields of the item:

```
1:  segm_str.frame_id = frame_id;
2:  segm_str.segm_id = is_frame ? 7 : segm_id;
3:  /* segment_id is carried over 3 bits */
4:  segm_str.frame_type = fr_type;
5:  segm_str.frame_gop =  gopIdGet(frame_id);
6:  segm_str.frame_pos = fr_pos;
7:  packetIndexList.push_back(segm_str);
8:  sem_post(&pk_sem);
```

After pushing the item into the queue, the last statement adds one unit to the semaphore value (line 8).

6.5 Processing NACK messages

In parallel to Rx, another thread (NACK Handler) is created to take responsibility of the NACK Handler. Once receiving a NACK message, the NACK Handler first extracts its header to identify the packet that the receiver node has just successfully obtained (or the packet that was expected to reach the receiver by the time this NACK message was created). Similarly to video packets, the identification

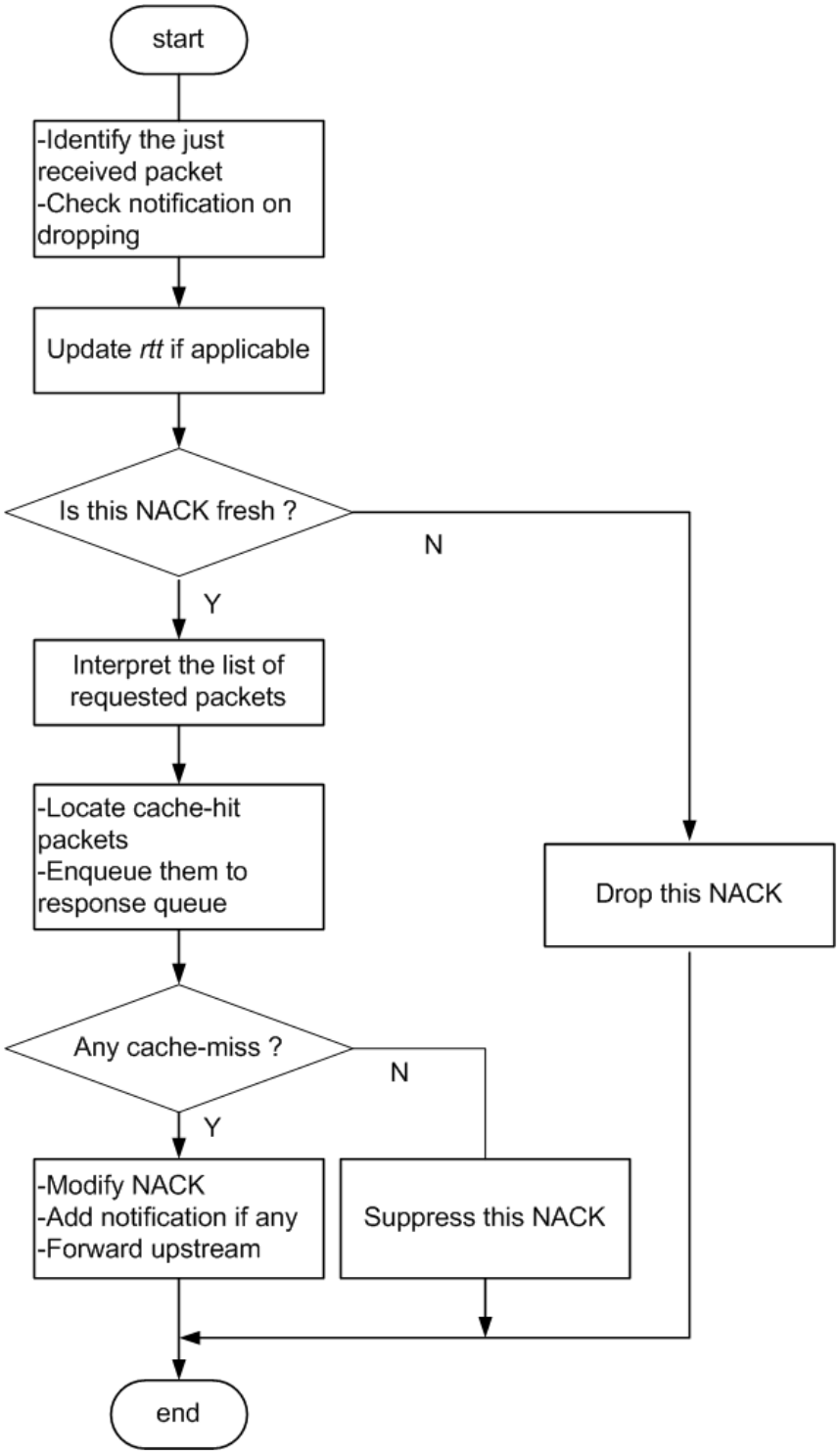

Figure 6.5: Operation of NACK Handler. A NACK message can be dropped, processed and/or forwarded upstream.

encapsulated in the header of NACK messages is also represented by three numbers: `gop_num`, `pos_num`, and `seg_num`. The extraction of these numbers is executed simply as follows:

```
1:  /* Extract packet ID number */
2:  seg_num = (unsigned char) NackBuf[0] & 7;
3:  pos_num = (unsigned char) NackBuf[0] >> 3;
4:  gop_num = (unsigned char) (NackBuf[1] & 15); /* use only 4
5:      LSB bits */
```

The operation of the NACK Handler is visually illustrated in the flowchart of Figure 6.5. Shortly speaking, once receiving a NACK message, it extracts the header to identify the just received packet (or the expected packet to arrive). The identification number of this packet is needed to update the round-trip-time to the receiver node and to interpret the list of requested packets (if the NACK message is fresh). After obtaining the list, the RtME locates and enqueues (logically) cache-hit packets. If any cache-miss occurs, the NACK message is forwarded upstream with the hope that upstream nodes will handle the request; otherwise, it will be suppressed.

As presented in Chapter 3 and Chapter 5, notifications on dropping of important frames are conveyed by NACK messages. As part of the joint discarding mechanism, the NACK Handler checks the received NACK message to see if there is any such a notification from downstream nodes. If so, it records the information to global variables `gop_to_drop` and `pos_to_drop` as seen in the code below. These values will be referred to when entity Tx selects a packet to transmit or when a new packet arrives at the node.

```
1:  if ((NackBuf[bytes-2] != 124) && (gop_to_drop
2:      != NackBuf[bytes-2]))
3:  {
4:    gop_to_drop = NackBuf[bytes-2];
5:    pos_to_drop = NackBuf[bytes-1];
6:    nack_notification_is_valuable = true;
7:  }
```

Recall from Section 4.4 that, the RtME takes advantage of arrival of NACK messages to update the round-trip-time to the receiver from this relaying node. The new round-trip-time is calculated by referring to the arrival time of the just received packet. Note however that there are chances that update should not take place. For instance, when the receiver node does not receive any packet for a very long time, the last received packet is now well obsolete. In such a case, it is useless to encapsulate the identification number of the last received packet into the NACK message. The receiver node instead adds the identification number of the frame that is expected to arrive. Should relaying nodes receive such a NACK message, they do not go updating the round-trip-time. One bit in the second byte of the NACK header is used for the receiver to signal "please do not update *rtt* !". The following code illustrates the behavior.

```
1:   /*Ask the system for the current time */
2:   ftime(&tm);
4:   time_now = TIME_NOW;
5:   /* Update the current round-trip-time if possible */
6:   if (frameWh.count(frame_id))
7:   {
8:     fr_ptr = frameWh[frame_id];
9:     if (fr_ptr->segm_num == 1) /* the frame contains only
10:                                  one packet */
11:    {
12:      /* Before updating, make sure that the receiver node does
13:         not signal anything */
14:      if (!(NackBuf[1] & 64))
15:      {
16:        current_rtt = time_now - fr_ptr->last_tx_stamp;
17:        last_rtt_time = time_now;
18:      }
19:    }
20:    else /* the just successfully received packet is of a
21:            multi-packet frame */
22:    {
```

```
23:      /* Make sure that this packet has ever arrived at this node,
24:      or null pointer exception may occur before going updating */
25:      if (fr_ptr->segments[seg_num].segm_state)
26:      {
27:        /* Before updating, make sure that the receiver node
28:           does not signal anything */
29:        if (!(NackBuf[1] & 64))
30:        {
31:          current_rtt = time_now -
32:          fr_ptr->segments[seg_num].last_tx_stamp;
33:          last_rtt_time = time_now;
34:        }
35:      }
36:    }
```

If this NACK message is fresh, the subsequent step of the NACK Handler is to interpret the list of requested packets from which cache-hit packets are identified. Those packets then are enqueued to the response queue (implemented as `list<Segm_ID>retranIndexList`). These behaviors are shown in the abstract code below.

```
1:    /*  NackBuf[]: array to hold data of the NACK message
2:        bytes: the length of NACK content
3:        is_any_cache_miss: count cache-miss packets
5:        bs: temporary variable that hold queue elements
6:        nack_fr_ptr: pointer to the requested frame
7:        suppressed: indicated if this NACK is to suppress
8:        retransIndexList: response queue
9:        MAX_NACK_FRAMES: maximum frames within one NACK
10:   */
11:   is_any_cache_miss = 0;
12:   suppressed = true;  //by default, suppress this nack
13:   //scanning bytes of nack
14:   for (i=2 /*first two bytes are for seg, pos, gop*/;
```

```
15:        i<MAX_NACK_FRAMES; i++)
16:   {
17:     if (i == bytes-2) break; /* last two by carry dropping
18:                                 notification */
19:     bs.segm_id = NackBuf[i] & 7; // 3 LSB bits hold segm id
20:     //4 MSB bits hold lost frame distance
21:     bs.frame_id = frame_id - 1 - ((NackBuf[i] >> 4) & 15);
22:     if (frameWh.count(bs.frame_id))  /* found pointer in
23:                                         the warehouse */
24:     {
25:       nack_fr_ptr = frameWh[bs.frame_id];
26:       bs.frame_type  = nack_fr_ptr->frame_type;
27:       bs.frame_gop   = nack_fr_ptr->frame_gop;
28:       //cache really hit
29:       retranIndexList.push_back(bs);
30:       Cut off NackBuf[i] from the NACK message;
31:       //update semaphore
32:       sem_post(&pk_sem);
33:     }
34:     else
35:     {
36:       is_any_cache_miss++;
37:     }
38:   }
39:   /* Only forward this NACK message upstream if there
40:       is still any cache-miss   */
41:   if (is_any_cache_miss>0)
42:   {
43:     if (this_node_wants_to_notify)
44:     {
45:       NackBufSent[last_byte-2] = gop_to_drop;
46:       NackBufSent[last_byte-1] = pos_to_drop;
47:     }
```

```
48:     Forward the modified NACK message upstream;
49:   }
```

On realizing the joint discarding mechanism, before forwarding the NACK message upstream, the RtME checks if there is a need to update the content of the notification. This is done via statements from line 41 to line 49 in the above abstract code. Additionally, should any cache-hit take place, the processed bytes will be removed from the NACK message before it is sent out. Suppression and partial cut-off of NACK messages are to lower the processing load at upstream load, and to reduce communication overhead.

6.6 Selection and transmission

Now that two queues are containing packets added by entity Rx (incoming packets) and the NACK Handler (retransmitted packets), entity Tx needs to dequeue the most appropriate packet to send downstream. As presented in Section 4.5.1, Tx first attempts to select one of head-of-queue packets from the queues. Selection is made considering packet semantics and their age. Basically, the selection occurs as illustrated in List 2 of Chapter 4. Subsequently, the selected packet is checked whether it is worth transmitting. As presented in Chapter 5, Tx refers to the *signal record* at this node to see if this packet is orphan before sending out. Figure 6.6 visually illustrates the operation of Tx.

If there is no enqueued packet to transmit, Tx is blocked on the calling of function `sem_wait(&pk_sem)`. Once activated by the positive value of the semaphore, Tx first calls function `indexToTransmit(Segm_ID& sgm_id, bool& retrans)` to obtain the appropriate packet from one of the queues. The first variable is to hold the returned reference to the selected packet.

In the next step, the selected packet is checked whether it should be relayed downstream or not. As described in Section 4.5.2, the check is carried out by comparing its remaining time-to-live and the expected time-to-destination. Additionally, entity Tx needs to make sure that the selected packet has not been transmitted any time within the last round-trip-time period (rtt). To realize this check, the entity records the time when the packet leaves the node. The packet is really sent out only if inequalities (4.3) and (4.4) hold. The code segment in the following text

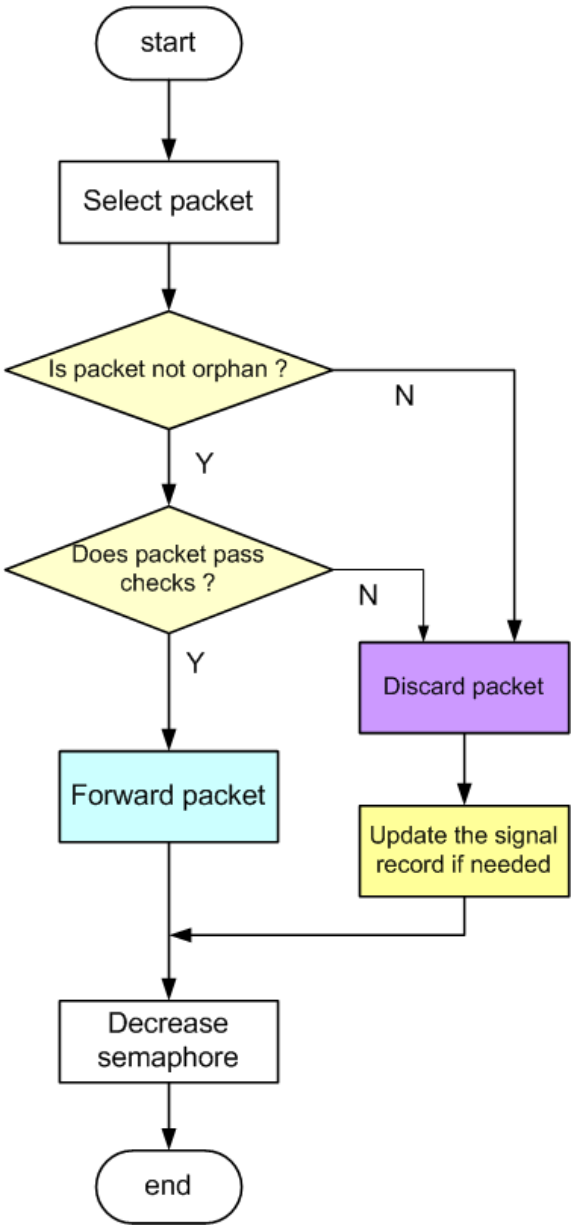

Figure 6.6: Operation of entity Tx. Basic functions are selecting, validating, forwarding, and dropping packets.

illustrates the behavior.

```
1:   if ((time_now - fr_ptr->time_stamp < DELAY_THRES -
2:      (unsigned short)(current_rtt/2)) && (time_now -
3:      fr_ptr->segments[segm_str.segm_id].last_tx_stamp
4:      > current_rtt))
5:   {
6:     bytes_to_tran = fr_ptr->segments[segm_str.segm_id].segm_size;
7:     Send the packet downstream;
8:     fr_ptr->segments[segm_str.segm_id].last_tx_stamp = time_now;
9:   }
```

Note that the expected time-to-destination has already been estimated optimistically (`DELAY_THRES - (unsigned short)(current_rtt/2)`), since the back-route conveying NACK messages is practically more idle than the forth-route that carries video packets. This is attributed to the fact that the load of video traffic is heavier than that of NACK messages. Theoretically, we can estimate the one-way delay (OWD) more precisely by some compensation strategy [8]. However, as introducing considerable overhead, it is not employed. Once the selected packet does not have enough optimistic time-to-live, it means that all other enqueued packets, if any, of the same frame also face the same situation; namely, they fall in the *dropping domain* (see Chapter 5). Entity Tx then signals the node to watch for future orphan packets as follows.

```
1:        if ((fr_ptr->frame_type != 3) && (fr_ptr->frame_gop !=
2:           gop_to_drop) && (segm_str.segm_id == 0))
3:        {
4:          gop_to_drop = segm_str.frame_gop;
5:          pos_to_drop = segm_str.frame_pos;
6:        }
```

6.7 Chapter review

Receiving video packets, processing NACK messages, and relaying packets downstream are implemented as parallel threads that are synchronized by a semaphore.

Due to data dependency of video packets, Real-time Media Engine organizes data representation into different levels to ease accessing and maintenance. The design of data types, including frame warehouse and queue, does take the matter of performance into account. In this chapter, we have concretely described how proactive functions are realized.

Chapter 7
EXPERIMENTAL RESULTS

To evaluate how sound the proactive framework practically is, we have deployed experiments with a testbed. Differently from the majority of related studies on video streaming over ad hoc networks, we do not rely on simulations to verify our proposed framework. Instead, a real-life testbed has been implemented on mobile computers. As stated in Section 1.3.4, unpredictability of wireless communication links in multihop connections potentially alters simulation results [27][28]. It was already recommended that studies on higher layers should base on testbeds for verifying the network performance. This chapter extensively reports results collected from different experiments.

7.1 Testbed overview

Physically, the testbed was composed of up to 8 ad hoc nodes of which 7 were mobile machines (laptop computers). Note that the study does not aim at testing behaviors of any routing algorithm, so there is no strong need to build a topology of vast number of nodes. Testing behaviors of the application layer in fact emphasizes the operation of each individual node. Compared to other known video testbeds [21][25], ours has a pretty-large scale.

To prove that the proactive framework works efficiently and is robust, the testbed devices should not be powerful. Indeed, the laptops were mostly Intel Pentium III-based platform. Except only one Pentium IV laptop was equipped with the built-in wireless network interface, all the machines employed merely removable wireless network cards for ad hoc communications, including PCMCIA and USB cards. Successfully deploying a testbed of such performance-moderate devices means that the strategy will operate well in most of systems, nowadays as well as tomorrow.

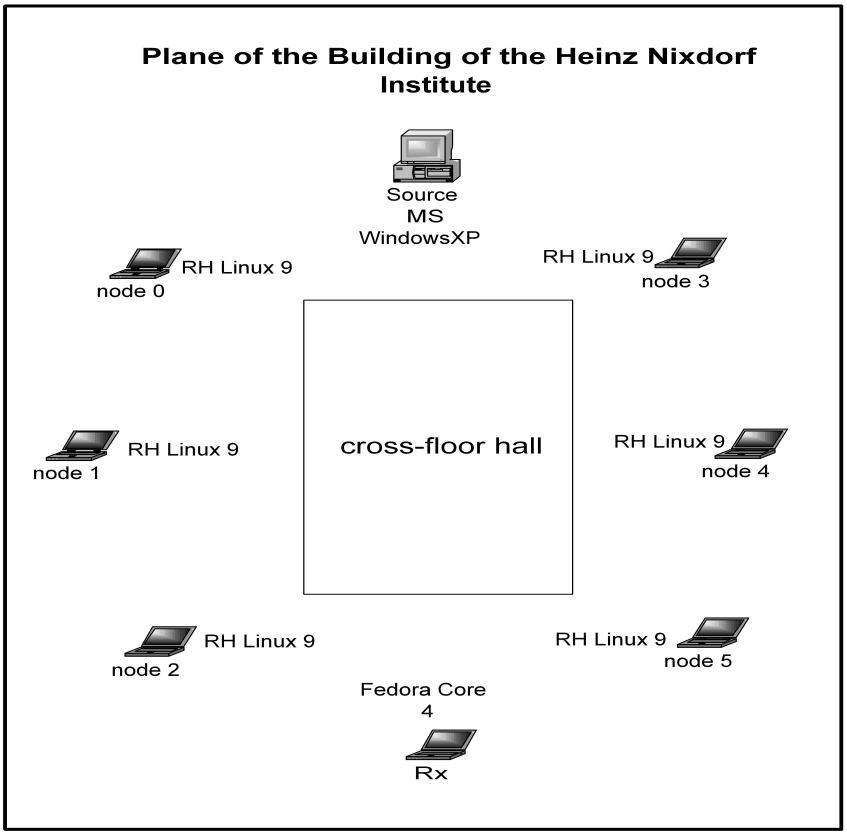

Figure 7.1: Layout of testbed.

7.2 Setup of the testbed

Geographically, the testbed was deployed in the building of the Heinz Nixdorf Institute, the University of Paderborn. In the place, it suffered interference and noise from the WLAN Access Points, other ad hoc networks, and laptop computers of students.

The deployed network was composed of up to 8 nodes as laid out in Figure 7.1. The sender machine was a Desktop PC running WindowsXP, and was equipped with Netgear MA111 USB card. All the other nodes were laptop computers running either RedHat Linux 9 or Fedora Core 4, and used Lucent Technologies PCMCIA cards for wireless video communication. Details on the machines are indicated in Table 7.1. The wireless network cards, which adopt IEEE802.11b standard, have settings shown in Table 7.2. Basically, we accepted the values that the operating systems WindowsXP and Linux suggest.

To demonstrate that the framework is open to other proposals, the approach

Table 7.1: Hardware configuration of relaying nodes

	CPU	**Main memory (MB)**	**Wireless NIC**
node 0	Pentium IV	512	built-in 3Com 3C920(*)/ PCMCIA WaveLAN LC
node 1	Pentium III	256	PCMCIA orinoco LC
node 2	Pentium III	256	PCMCIA orinoco LC
node 3	Pentium III	256	PCMCIA orinoco LC
node 4	Pentium III	256	PCMCIA WaveLAN LC
node 5	Pentium III	256	PCMCIA orinoco LC

(*) *The built-in card was used for network maintenance tasks, not for video communication.*

Table 7.2: Parameters of wireless settings

parameter	**value**
Bit rate	2Mb/s
Frequency	2.4GHz
Tx-Power	15dBm
Sensitivity	1/3
Retry limit	7
Fragment threshold	2347 bytes

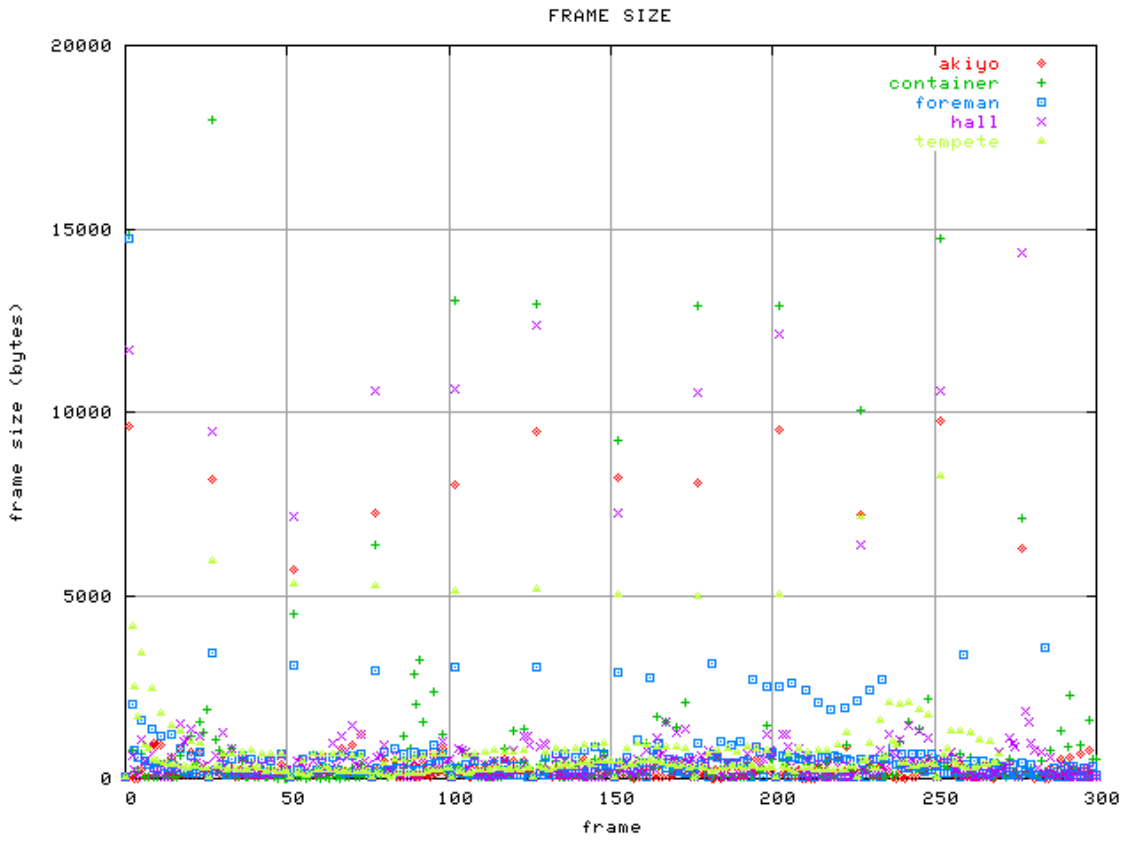

Figure 7.2: Frame sizes of different video sequences. *Akiyo* has a medium volume, smaller than *Container* and *Hall*, but larger than *Foreman* and *Tempete*.

of Multiple Path Transport (MPT) [8][21][83][84] was also integrated. Namely, video packets were delivered over two disjoint paths toward the receiver.

7.3 Deployment of experiments

For comparison, three experiments were implemented; one was on MPT with conventional passive nodes, another was on RtME-enabled nodes but without the joint discarding mechanism (hereafter referred to as "active processing"), and the last was on full-featured RtME. All the experiments were made on MPEG4 CIF video sequence *Akiyo* [82] that has 300 video frames and a nominal rate of 200 Kbps. The sequence was selected considering its popularity and its traffic. As shown in Figure 7.2, while I frames of *Akiyo* are not so large as *Container* or *Hall*, they are greater than those of *Tempete* and *Foreman* [82]. Remarkably, relative

Table 7.3: Video sequence transmitted

parameter	**value**
Nominal bit rate	200 Kb/s
Frame rate	30 fps
Frame count	300 video frames
Format	CIF
Codec	MPEG4

Figure 7.3: Sample pictures from three experiments.

difference among *Akiyo* video frames in their size is much considerable compared to the others. This is important since we wished to see how our framework reacts to instability of the traffic. Further details about the sequence are listed in Table

7.3. In each experiment, 30 episodes of the sequence were transmitted repeatedly, so that up to 9000 video frames in total were sent out. The experiments had identical settings and each node followed the same movement profile.

7.4 Distortion evaluation

The received video sequences in the three experiments were measured distortion after error concealment. Figure 7.3 shows sample pictures in the three received video of the experiments. All-frame PSNR (Peak Signal to Noise Ratio) statistics of the three transmission schemes are plotted in Figure 7.4, 7.5, and 7.6, respectively. In all the experiments, the perception quality was highly unstable. However, it is pretty clear that, for the majority of 30 episodes, RtME did improve PSNR. The quality difference between the active processing strategy and the passive forwarding case was significant. Compared to the active processing case, introduction of the joint discarding mechanism did not improve much; the benefit with respect to energy consumption, however, was clear (presented later in Section 7.5). Indeed, the all-frame average PSNR values in the three experiments were 32.67dB, 34.94dB, and 36.18 dB, respectively. So, in long term, the full-featured RtME brought up an improvement of up to 3.51 dB. The all-frame statistics also show that our framework stabilized the perception quality. Namely, it reduced the number of periods as well as the total time of severely bad perception quality.

The soundness of our framework is more visible in Figure 7.7. In this figure, each point represents the episode-averaged PSNR value measured for an episode. One can realize that the number of episodes with better PSNR in our strategy is higher, and that the proactive forwarders bring up generally better perception quality.

As depicted in Figure 7.8, the minimum PSNR values in the RtME-enabled experiments are higher than in the passive case, and so are those values in worst episodes.

7.5 Power consumption

We evaluate the power consumption of each laptop computer by monitoring its battery status. To make the comparison more accurate, we ran OSs of all the

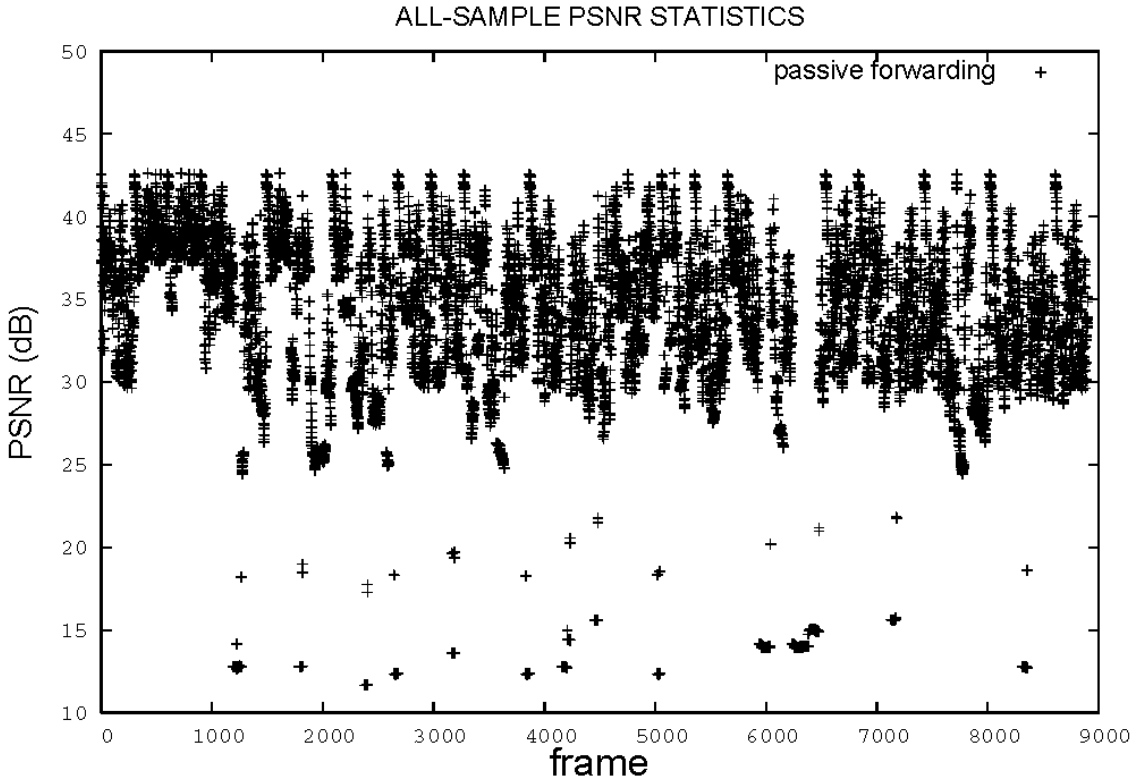

Figure 7.4: All-frame PSNR of 30 episodes: passive forwarding. In general, the playback quality was substantially fluctuated from frame to frame. In most of the video episodes, we observe severely bad frames.

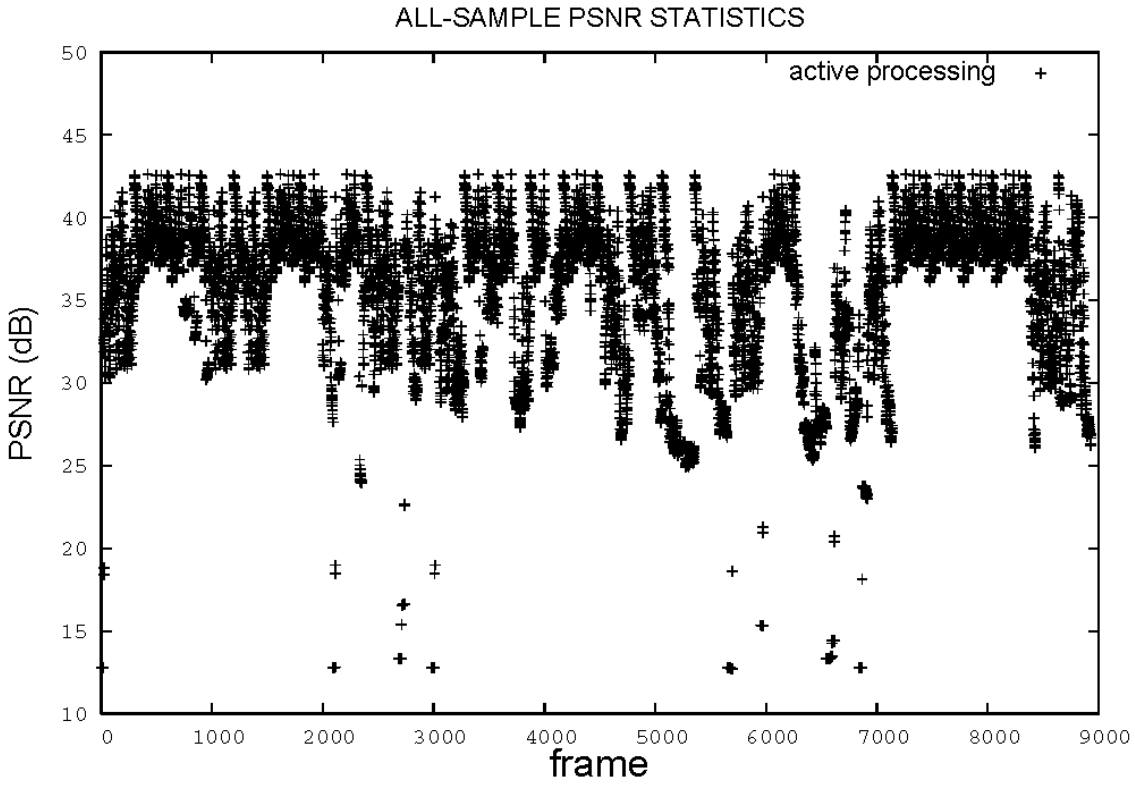

Figure 7.5: All-frame PSNR of 30 episodes: active processing without joint discarding. Activation of Real-time Media Engine has clearly improved the playback quality. It does not only stabilize distortion, but also reduces the number of severely bad frames and episodes.

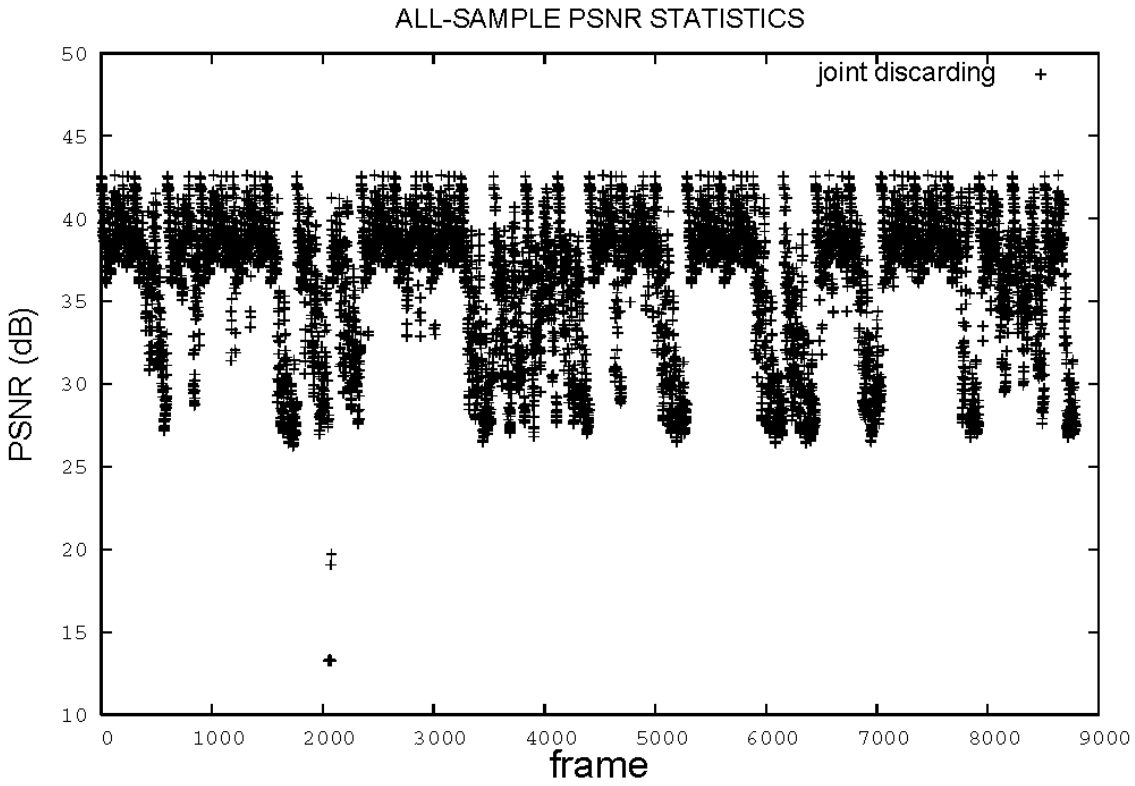

Figure 7.6: All-frame PSNR of 30 episodes: Real-time Media Engine was fully activated. Compared to the second experiment with active processing, the joint discarding mechanism did not substantially improve the average distortion, though the number of bad frames was decreased. However, the benefit regarding energy was more recognizable.

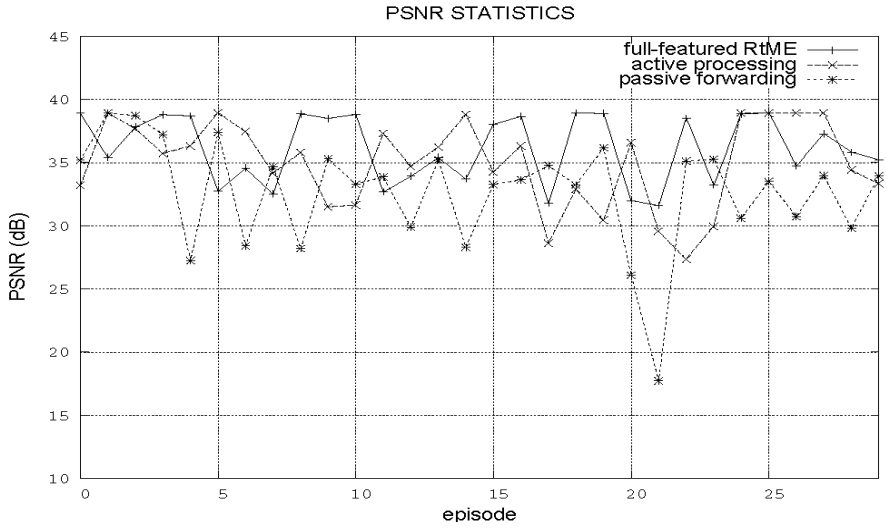

Figure 7.7: Episode-averaged PSNR. When Real-time Media Engine was activated, whether partially or fully, the average frame quality got improved in a considerable number of episodes.

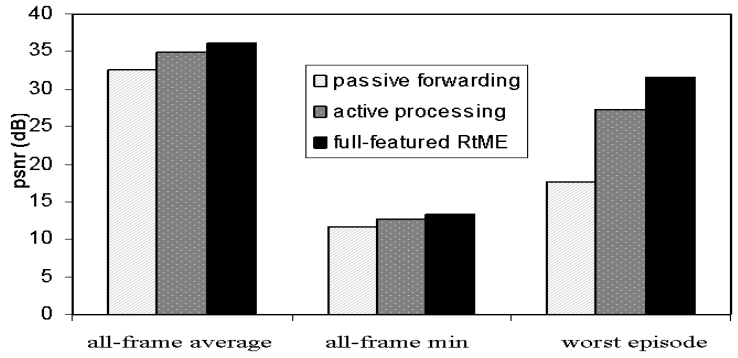

Figure 7.8: PSNR: min/average values and worst-episode values. It is clear that activation of Real-time Media Engine raises the quality overall as well as during adverse communication periods.

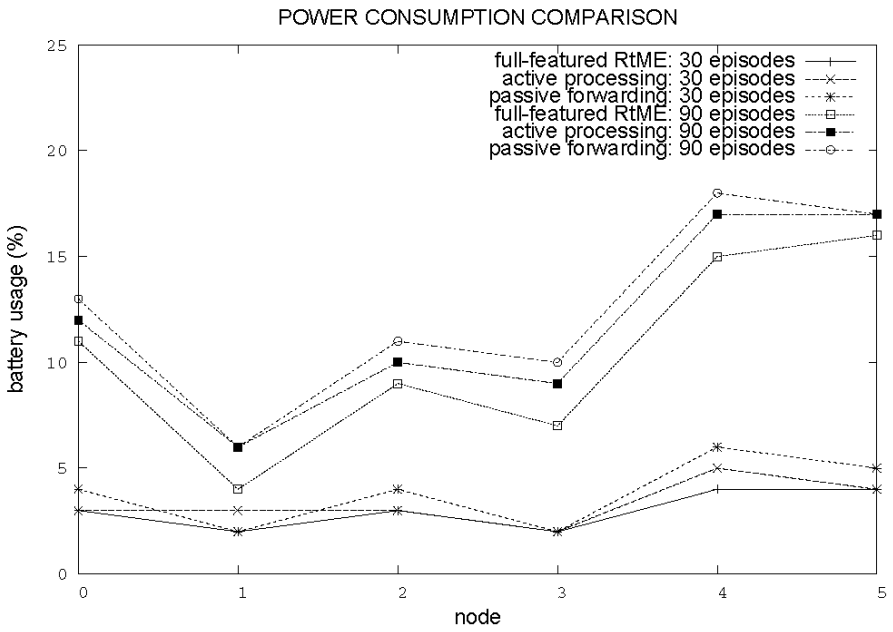

Figure 7.9: Power consumption. The lower part represents consumption in the experiments of 30 episodes. Lengthening the communication time (90 episode), the reduction when Real-time Media Engine was activated was more visible (the upper part).

intermediate nodes in the text mode, killing all irrelevant services/processes, except daemon *olsrd* [65], which is a proactive routing protocol. Figure 7.9 charts the power consumption of all the intermediate nodes. Note that since the goodness of the batteries in different computers is not identical, it should not be inferred that the difference of percentages between any two nodes truly reflects the difference of energy amounts they consumed. As indicated in the chart (the lower part), without the joint discarding, we see that only one node (node 1) consumed more power in our framework than in the passive case, even though the difference is not major; node 3 observed the same usage. All the other relaying nodes consumed less power in the proactive strategy. When the intermediate nodes activated the joint discarding mechanism, the reduction was clearer. Noticeably, though we could not measure the power consumption of the Windows-based sender machine, we realized that, with proactive intermediate nodes, the sender machine must respond to ARQ requests much less than the passive case. Section 7.8 will show the retransmission load that the machine dealt with.

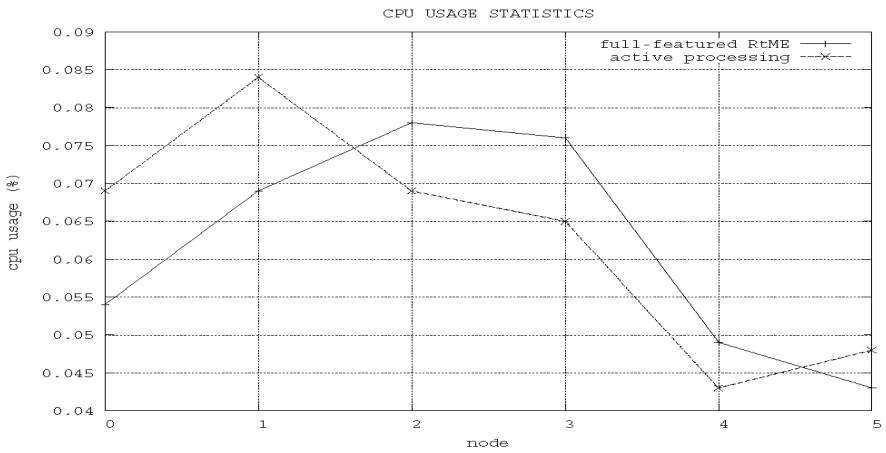

Figure 7.10: CPU usage when RtME was activated partially and fully. The usage was less than 0.09%; the difference between the two cases was basically not clear.

Extending the number of episodes experimented to 90, we observed that the soundness of our framework was more considerable, as seen in the upper part of Figure 7.9. Compared to the passive forwarding case, the proactive framework without the joint discarding mechanism did reduce the battery usage at nodes 0, 2, 3 and 4 while nodes 1 and 5 did not observe any reduction at all. When RtME was fully activated at all the forwarding nodes, the energy saving was clearer; each node consumed from 1% to 3% less than the passive forwarding strategy.

7.6 CPU usage

Even RtME was forced to collect heavy statistics, the CPU usage in our framework was extremely low, no greater than 0.09%. This further demonstrates the feasibility of the framework. Figure 7.10 plots the CPU usages of all the intermediate nodes, which had either Intel Pentium III or IV platform. Basically, reinforcement of the joint discarding mechanism does not clearly add more computation load. The reason is likely that dropping useless packets induces extra header processing, but it also decreases the number of packets to relay. In the future, upon completion of the middleware, removal of statistics collection, and code optimization, we expect even lower CPU usage.

Again, since the laptops are not identical in hardware and OS configurations, difference between any two nodes does not reflect the difference of their real com-

putation load.

7.7 Cache space

As RtME was activated, the maximum cache levels measured in the primary and the secondary paths were respectively 24646 bytes and 10294 bytes (corresponding to the second experiment). Obviously, these values are not prohibitive to today's computers or embedded devices. The episode-averaged caching statistics estimated on each path are respectively plotted in Figure 7.11 and Figure 7.12, further demonstrating that the memory allocation for caching video packets is highly feasible. From the figures we learn that the joint discarding mechanism slightly reduces the cache occupancy.

As the charts show, it can be inferred that the average values are considerably smaller than the peak values mentioned above. Over the primary path, no episode observed an average cache level higher than 17,500 bytes. The difference in the secondary path is even more recognizable; all the episodes had their average cache level less than 2,500 bytes. As seen in the charts, the joint discarding mechanism lessened the average data volume in the warehouse, though not significantly. The average level was lower in most of 30 episodes compared to the active processing case. This can be explained that the application of the mechanism has taken effect. It did help the ad hoc forwarders detect and drop out useless packets right at their arrival.

7.8 Sender response

As the intermediate nodes are responsive, the sender does not have to process all ARQ requests from the receiver node. This reduces the usage of both computation resource and energy in the sender, which is busy enough with processing video. Figure 7.13 shows that, the activations of RtME at the forwarding nodes significantly reduced the number of ARQ requests that the sender machine must respond to. Specifically, in the passive forwarding case, up to 53 NACK messages were handled at the worst episode (number 11) while this number was only 14 (at episode number 18) when RtME was active. Should RtME be fully activated, the reduction was even clearer; no episode had to handle more than one reply. Note however that

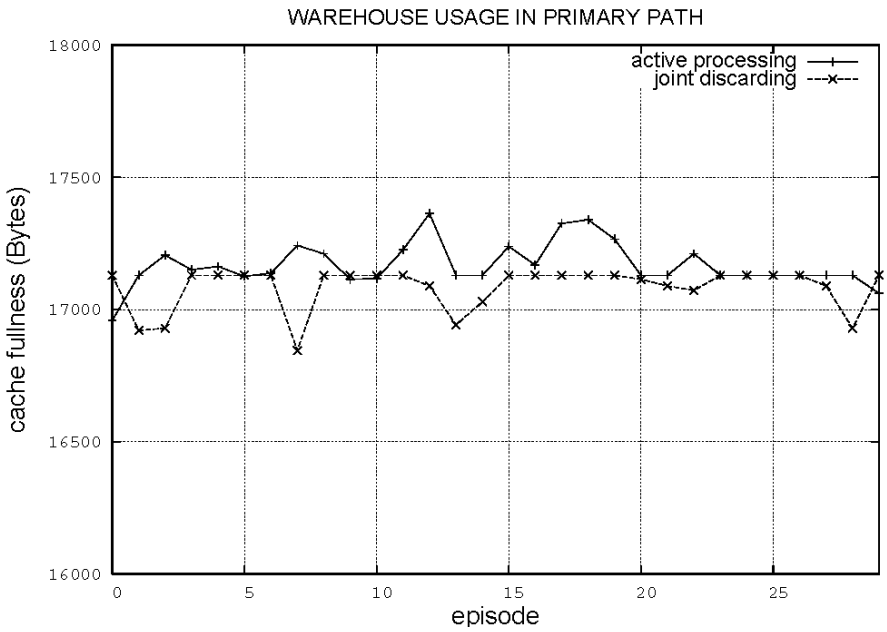

Figure 7.11: Cache occupancy: primary path. No node observed an average level greater than 17,500 bytes.

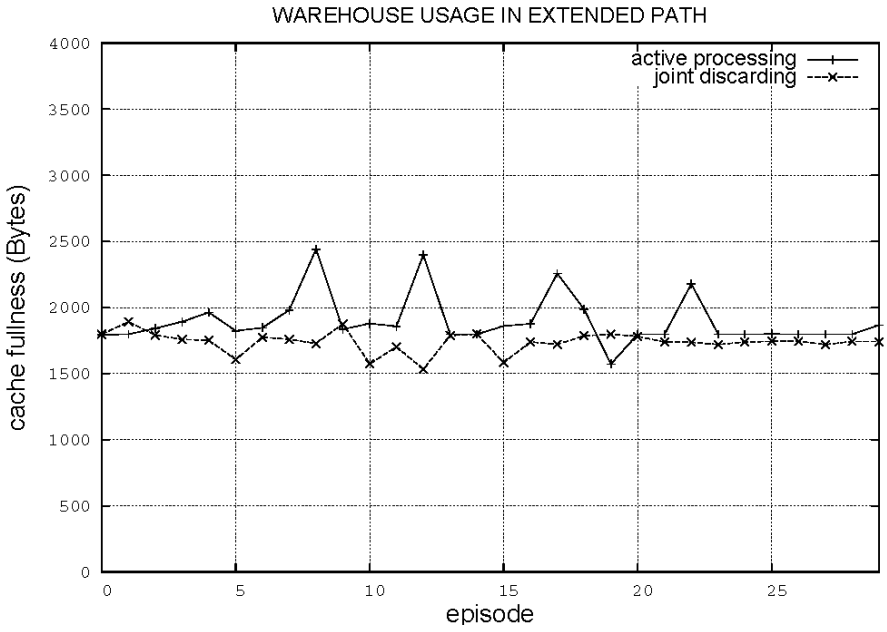

Figure 7.12: Cache occupancy: secondary path. No node observed an average level greater than 2,500 bytes.

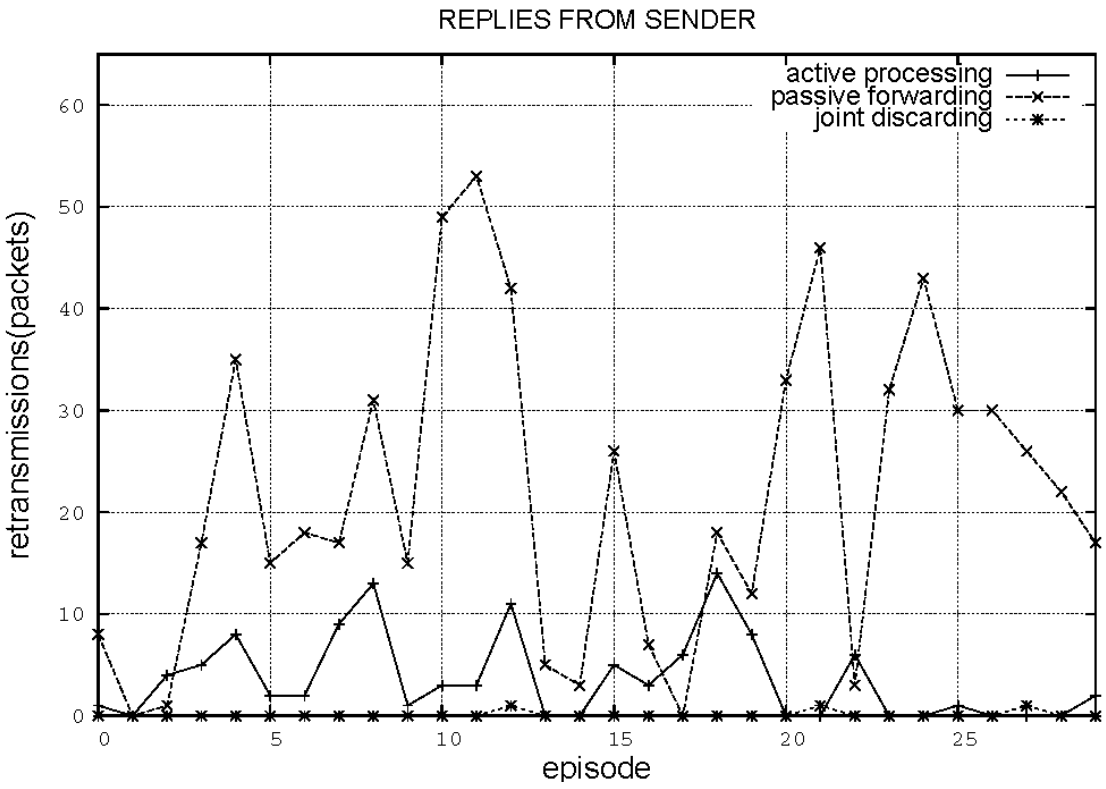

Figure 7.13: Retransmission load from the sender: count of replies. Activations of Real-time Media Engine significantly reduced the number of times that the sender responded to NACK messages.

the number of NACK messages arriving at the sender might be higher than actually being processed. The relaying nodes with the joint discarding function did suppress the NACK messages that requested stale packets.

The above reduction results in decreasing the retransmitted data load from the sender node. As seen in Figure 7.14, the load in the worst episode of the passive case was 72,385 bytes, corresponding to episode number 10; at the active processing experiment, the value was 13,038 bytes, while only 63 bytes were retransmitted from the sender in a couple of bad episodes when RtME ran the joint discarding function.

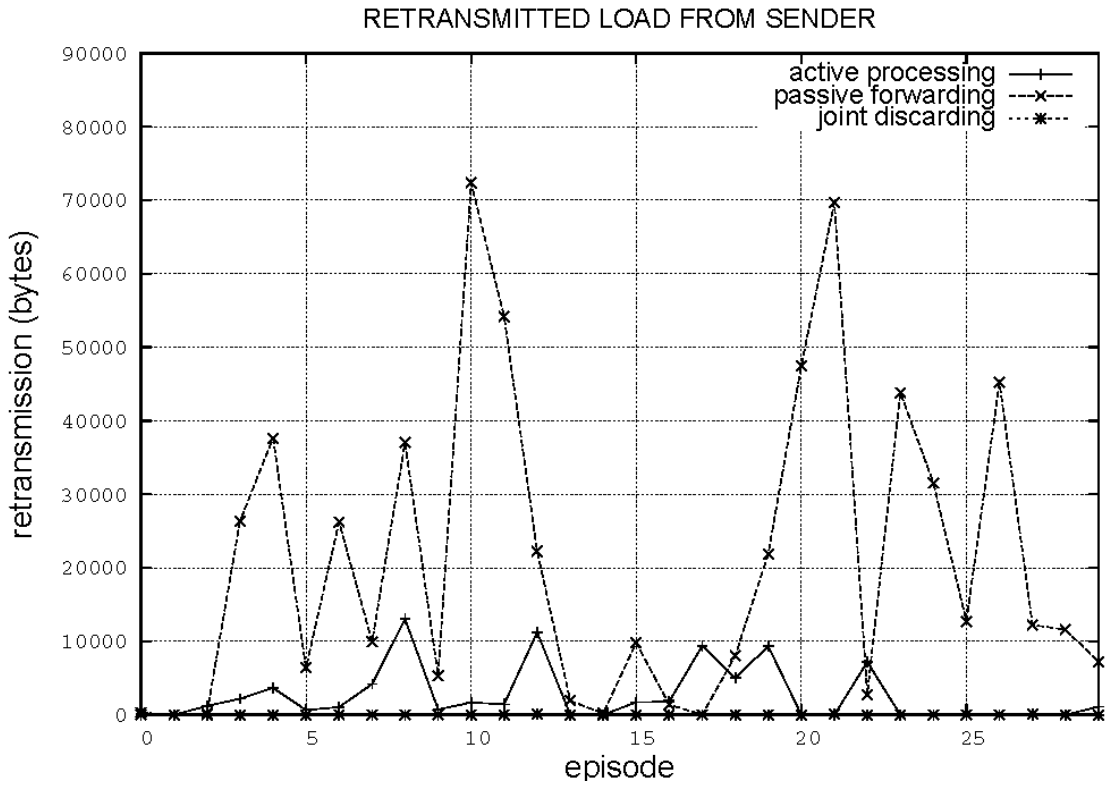

Figure 7.14: Retransmission load from the sender: load of data. The data volume for retransmission also got decreased when Real-time Media Engine was activated.

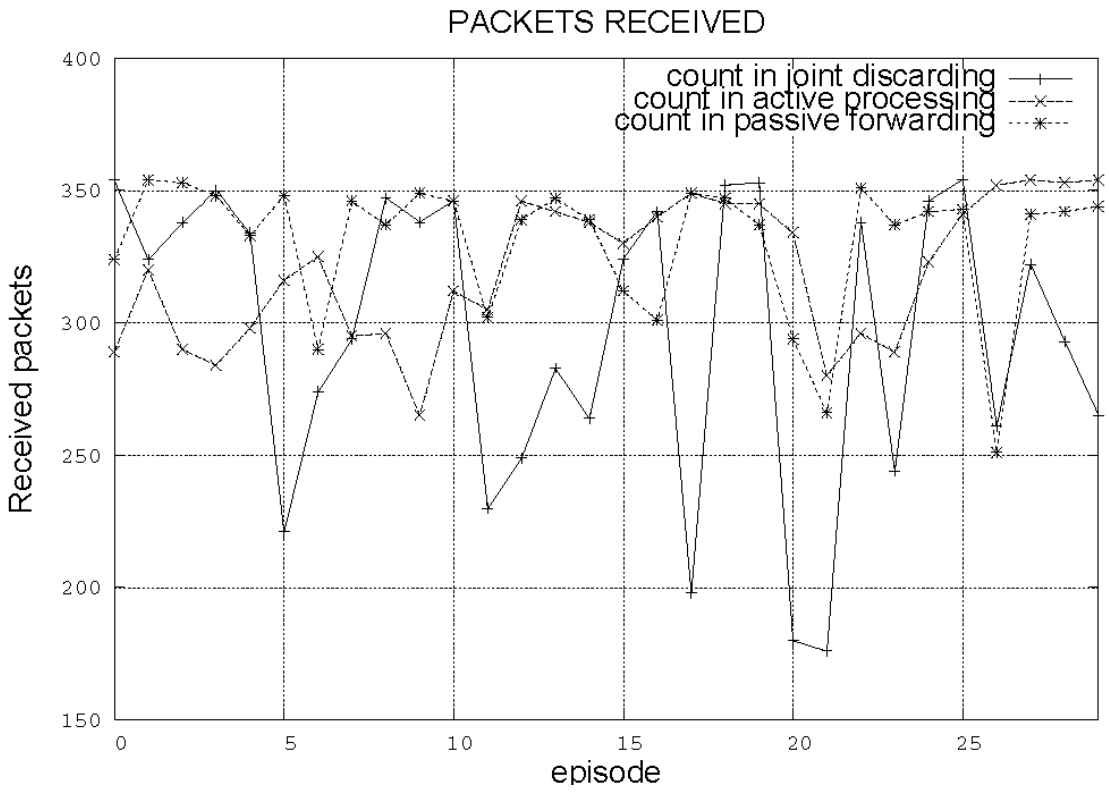

Figure 7.15: Received packet counts. It is recognizable that introduction of Real-time Media Engine did decrease the number of packets received.

7.9 Traffic load

As presented in Section 2.2.1, video packets are not independent of each other. Loss of packets pertaining to an I or P frame may devalue transmission of many others, even they arrive correctly at the receiver node. This means that the quality of the received video is not linearly proportional to the number of packets reaching the receiver node. In fact, what really matters is the number of *useful* packets received and how much they contribute to the decoding process. Chapter 4 already shows that RtME allows intermediate nodes to smartly select packets based on their semantics and remaining time-to-live. Additionally, we also know in Chapter 5 that RtME helps block useless or stale packets that conventional forwarders would continue to process. So the proactiveness introduced to the forwarding nodes does potentially decrease the traffic load coming to the receiver node.

Let's look at Figure 7.15 that plots the received packet counts in each experiment. Generally, RtME lowered the count in most of episodes. The influence of the joint discarding mechanism was considerable, especially when the propagation seriously degraded, e.g. during episode 21. Over all the experiments, it remarkably reduced the counts in the majority of 30 episodes. This partially explains why energy consumption in our framework was reduced. Note that cumulative consumption also depends on other factors such as expected transmission count (ETX) [65], load of NACK messages, and how many hops retransmission packets traverse.

It is clear that while receiving more packets in comparison to our proposed approach, the passive forwarding strategy did not obtain a better performance. One of the main reason is that conventional node architecture could not eliminate useless packets. Figure 7.16 shows how many packets that arrived at the receiver node were unworthy. These include late packets - ones that arrived after their deadline, and redundant packets - ones that were received multiple times. Note however that orphan packets were not counted. As seen in the figure, the proactive framework consistently and clearly decreased the number of useless arrivals. In the first experiment, up to 115 packets were useless at episode 21, which was the worst case. This count in the second experiment was 19, observed at episode 13. Meanwhile, the joint discarding mechanism lowered the value down to only 2 useless packets.

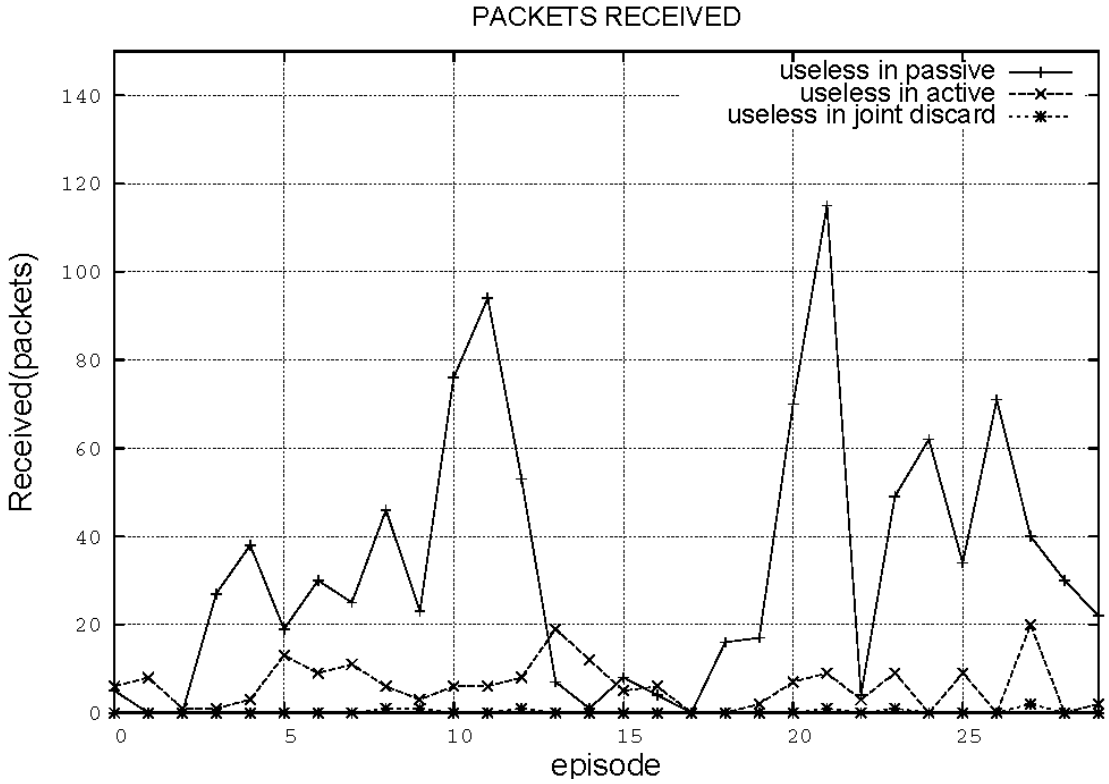

Figure 7.16: Useless packets received in all the experiments. As stale and redundant packets were detected, the activations of Real-time Media Engine lowered the number of useless packets that arrived at the receiver node.

7.10 Chapter review

The efficiency of the proactive framework has practically been demonstrated. The experimental results shows that the quality of the received video is improved while the cumulative energy consumption of the relaying hosts is reduced. The testbed with inexpensive devices also proves the feasibility regarding memory usage and computation load.

Chapter 8 CONCLUSION AND OUTLOOK

8.1 Conclusion

Realizing real-time video communication over ad hoc wireless networks is not a trivial task. As wireless channels in ad hoc networks are capacity-limited, deployment of video applications is problematic. Differently from related works, this study has investigated intermediate nodes on relaying real-time video packets. In the proposed proactive framework, we have designed a node architecture in which each relaying node transiently retains packets so that they can be appropriately selected to forward when congestion occurs. Particularly, retransmission can be made from intermediate nodes instead of the very sender. The shortening of retransmission rounds does result in higher end-to-end success rate of delivery and lower the cumulative power consumption.

Proactive nodes can also detect and drop packets that are useless for the decoding process. Once rejecting important frames, a video-cooperative node may also notify other nodes to jointly destroy their dependent packets (that are definitely useless). This mechanism does save both bandwidth and energy for useful packets. To avoid generating communication overhead, NACK messages themselves are used to carry notifications on discarding such important frames.

The framework has been implemented via the introduction of Real-time Media Engine (RtME) that makes relaying nodes more intelligent and responsive. Our lightweight framework is open to other strategies, and does guarantee the interoperability of heterogeneous network nodes, including conventional video-passive forwarders.

The deployment of the experiments helped us understand how the framework behaves in reality. Extensive results collected from the testbed do demonstrate the soundness and the applicability of the proposed relaying strategy.

8.2 Outlook

Advances in hardware technology make embedded devices and mobile computers more and more powerful. However, as ad hoc nodes mainly operate on their battery power, memory usage and computation load should be lowered as much as possible. In the future work, we will look into the implementation to optimize the coding design.

Secondly, as needs of users may be diverse, we consider developing more algorithms for packet selection to satisfy different goals. For instance, QoS definition might be different from user to user; some operations such as military communication may be strict on delay whereas professional applications, e.g. surveillance and monitoring, are sensible to packet loss.

Finally, like what currently takes place over the Internet, video communication in ad hoc networks will likely accommodate multipoint-to-multipoint calls in a lot of cases. Extension of the framework for multicasting will hence be considered. With the proactive functionality, it is not hard to develop flexible strategies for efficiently delivering packets to multiple nodes.

Bibliography

[1] J. Wu, *Techniques to Avoid Useless Packet Transmission in Multimedia over Best-effort Networks*, Dr. Dissertation, University of New South Wales, 2003.

[2] IPTV Focus Group, *http://www.itu.int/ITU-T/IPTV/.*

[3] J. Chakareski, P. Chou, and B. Girod, *RaDiO Edge: Rate-Distortion Optimized Proxy-driven Streaming from the Network Edge,* IEEE/ACM Transactions on Networking, vol. 14, no. 6, pp. 1302-1312, 2006.

[4] T. Wei, C. Jacob, S. Eckehard, *Rate-Distortion Optimized Frame Dropping and Scheduling for Multi-user Conversational and Streaming Video,* Journal of Zhejiang University SCIENCE A, vol. 7, no. 5, pp. 864-872, 2006.

[5] Y. Lu, K.J. Christensen, *Using Selective Discard to Improve Real-time Video Quality on an Ethernet Local Area Network,* International Journal of Network Management, 9(2), pp. 106-117, 1999.

[6] P. A. Chou and Z. Miao, *Rate-Distortion Optimized Streaming of Packetized Media,* IEEE Transactions on Multimedia, vol. 8, no. 2, pp. 390-304, 2006.

[7] T. Pham Van, *Real-time Video over Programmable Networked Devices,* Proceedings of the IFIP International Conference on Network and Parallel Computing (NPC'05), LNCS 3779, pp. 409 416, Beijing, China, Nov. 30 - Dec. 3, 2005.

[8] A. A. E. Al, T. Saadawi, M. Lee, *Improving Interactive Video in Ad-hoc Networks Using Path Diversity,* Proc. IEEE International Conference on Mobile Ad-hoc and Sensor Systems, October 2004.

[9] H. Shojania, B. Li, *Experiences with MPEG-4 Multimedia Streaming,* in Proceedings of the 9th ACM Multimedia Conference (ACM Multimedia 2001), pp. 492-494, Ottawa, Canada, September 30 - October 5, 2001.

[10] S. Kaiser and K. Fazel, *Comparison of Error Concealment Techniques for an MPEG-2 Video Decoder in Terrestrial TV-broadcasting,* in Special Issue on Error Resilient Video of Signal Processing: Image Communication journal, vol. 14, no. 6-8, pp. 655-676, May 1999.

[11] C. Huang and P. Salama, *Error Concealment for Shape in MPEG-4 Object-Based Video Coding,* IEEE Transactions on Image Processing, vol. 14, no. 4, pp. 389-396, 2005.

[12] MPEG Pointers and Resources, http://www.mpeg.org/MPEG/index.html.

[13] D. Wu, T. Hou, W. Zhu, H.-J. Lee, T. Chiang, Y.-Q. Zhang, and H. J. Chao, *On End-to-End Architecture for Transporting MPEG-4 Video Over the Internet,* IEEE Transactions on Circuits and Systems for Video Technology, 10(6), pp. 923-941, 2000.

[14] X. Meng, H. Yang, S. Lu, *Application-oriented Multimedia Scheduling over Lossy Wireless Networks,* Proc. IEEE International Conference on Computer Communications and Networks, October 2002.

[15] IEEE 802.11, *The Working Group Setting the Standards for Wireless LANs*, www.ieee802.org/11/.

[16] C. S. R. Murthy and B. S. Manoj, *Ad Hoc Wireless Networks Architectures and Protocols,* Prentice Hall, 2004.

[17] X. Zhu, E. Setton and B. Girod, *Congestion-Distortion Optimized Video Transmission over Ad Hoc Networks,* Journal of Signal Processing: Image Communications, vol. 20, pp. 773-783, September, 2005.

[18] N. Blundell, *Overlays for Live Internet Multimedia Streaming Systems,* 1st year PhD student report, Lancaster University, November, 2002.

[19] T. Pham Van, *Video-Cooperative Design for Ad hoc Networks,* IEEE Wireless Communications and Networking Conference (IEEE WCNC'07), Hong Kong, China, March 11-15, 2007 (to be presented).

[20] A. Harris, C. Sengul, R. Kravets, P. Ratanchandani, *Energy-Efficient Multimedia Communications in Lossy Multi-Hop Wireless Networks,* IFIP TC6 / WG6.8 Conference on Mobile and Wireless Communication Networks (MWCN 2004), Paris, France pp. 461-472, October 25-27, 2004.

[21] S. Mao, S. Lin, S. S. Panwar, Y. Wang, EmbreCelebi, *Video Transport over Ad hoc Networks: Multistream Coding with Multipath Transport,* IEEE Journal On Selected Areas In Communications, vol. 21, no. 10, pp. 1721-1737, 2003.

[22] X. Zhu, S. Rane and B. Girod, *Systematic Lossy Error Protection for Video Transmission over Wireless Ad Hoc Networks,* SPIE Visual Communications

and Image Processing (VCIP-05), vol. 5960, pp. 1849-1860, Beijing, China, July 2005.

[23] E. Setton, T. Yoo, X. Zhu, A. Goldsmith, B. Girod, *Cross-layer Design of Ad hoc Networks for Real-time Video Streaming,* IEEE Wireless Communications Magazine, vol. 12, no. 4, pp. 59-65, 2005.

[24] E. Setton, X. Zhu and B. Girod, *Congestion-Optimized Scheduling of Video over Wireless Ad Hoc Networks,* IEEE 2005 International Symposium on Circuits and Systems, vol. 4, pp. 3531-3534, Kobe, Japan, May 2005.

[25] H. Gharavi, K. Ban, *Cross-layer Feedback Control for Video Communications via Mobile Ad-hoc Networks,* Proc. IEEE Vehicular Technology Conference, vol. 5, pp. 2941-2945, October 2003.

[26] H. Gharavi, K. Ban, *Rate Adaptive Video Transmission over Ad-hoc Networks,* Electronics Letters, vol. 40, no. 19, pp. 1177-1178, 2004.

[27] J. Kuruvila, A. Nayak, and I. Stojmenovic, *Hop Count Optimal Position-based Packet Routing Algorithms for Ad hoc Wireless Networks with a Realistic Physical Layer,* IEEE Journal On Selected Areas In Communications, vol. 23, no. 6, pp. 1267-1275, June 2005.

[28] D. Cavin, Y. Sasson, and A. Schiper, *On the Accuracy of Manet Simulators,* Proceedings of the ACM Principles of Mobile Computing (POMC 2002), Toulouse, France, October 30-31, 2002.

[29] NS-2, *The Network Simulator,* http://www.isi.edu/nsnam/ns.

[30] OPNET Modeler, http://www.opnet.com/products/modeler/home.html.

[31] R. Bagrodia, and M. Gerla, *Glomosim: A Scalable Network Simulation Environment,* Technical Report 990027, UCLA Computer Science Department, May 1999.

[32] Z. He, Y. Liang, L. Chen, I. Ahmad, and D. Wu, *Power-Rate-Distortion Analysis for Wireless Video Communication under Energy Constraints,* IEEE Transactions on Circuits and Systems for Video Technology, Special Issue on Integrated Multimedia Platforms, vol. 15, no. 5, pp. 645-658, May 2005.

[33] S. Chakraborty and D. K. Y. Yau, *Predicting Energy Consumption of MPEG Video Playback on Handhelds,* Proc. IEEE International Conference on Multimedia and Expo, pp. 317-320, 2002.

[34] S. Mao, Y. T. Hou, X. Cheng, H. Sherali, S. Midkiff, and Y. Zhang, *On Routing for Multiple Description Video over Wireless Ad hoc Networks,* IEEE Transactions on Multimedia, vol. 8, no. 5, pp. 1063-1074, 2006.

[35] C. Wu, T. Hou, *The Impact of RTS/CTS on Performance of Wireless Multihop Ad Hoc Networks Using IEEE 802.11 Protocol,* IEEE SMC, 2005.

[36] K. Langendoen and G. Halkes, *Energy-efficient Medium Access Control,* Book chapter in the Embedded Systems Handbook, R. Zurawski (editor), CRC press, August 2005.

[37] Ahmed Safwat et al, *Power-Aware Wireless Mobile Ad hoc Networks,* Handbook of Ad hoc Wireless Networks, CRC Press, December 2002.

[38] S. Singh, CS Raghavendra, *Power efficient MAC protocol for multihop radio networks,* Proc. IEEE International Personal, Indoor and Mobile Radio Communications Conference, pp. 153-157, 1998.

[39] I. Bouazizi, *Size-Distortion Optimized Proxy Caching for Robust Transmission of MPEG-4 Video,* In LNCS 2899, Proc. International Workshop on Multimedia Interactive Protocols and Systems, November 18-21, 2003.

[40] E. Masala, H. Yang, K. Rose and J. C. De Martin, *Rate-Distortion Optimized Slicing, Packetization and Coding for Error Resilient Video Transmission,* Proc. IEEE Data Compression Conference, 2004.

[41] T. Pham Van, *Proactive Optimization of Real-time Video,* Proc. International Conference on Wireless Communications, Networking and Mobile Computing 2005, IEEE Press, Wuhan, China, September 23-26, 2005.

[42] T. Pham Van, *A Practical Approach for Real-time Video Streaming,* Proc. 13th International Conference on Software, Telecommunications and Computer Networks, (SoftCOM'05), Split, Croatia, September 15-17, 2005.

[43] Y. Bai and M. Robert Ito, *QoS Control for Video and Audio Communication in Conventional and Active Networks: Approaches and Comparison,* IEEE Communications Surveys & Tutorials, vol. 6, no. 1, 2004.

[44] RFC 2475, *An Architecture for Differentiated Services,* http://rfc.net/rfc2475.html.

[45] Y. C. Chang, C. C. Huang, H. C. Chang, H. C. Fang, L. G. Chen, *Error-Propagation Analysis and Concealment Strategy for MPEG-4 Video Bitstream*

with Data Partitioning, IEEE International Conference on Multimedia and Expo (ICME 2001), Tokyo, Japan, August 2001.

[46] M. Christiansen, K. Jeffay, D. Ott, and F.D. Smith, *Tuning RED for Web Traffic,* IEEE/ACM Transactions on Networking, vol. 9, no. 3, pp. 249-264, 2001.

[47] M. A. Labrador and S. Banerjee, *Packet Dropping Policies for ATM and IP Network,* IEEE Communications Surveys, vol. 2, no. 3, Third Quarter 1999.

[48] W. Feng, D. Kandlur, D. Saha, K. Shin, *A Self-Configuring RED Gateway,* IEEE INFOCOM '99, March 1999.

[49] David D. Clark and Wenjia Fang, *Explicit Allocation of Best Effort Packet Delivery Service,* ACM Transactions on Networking, vol. 6, no. 4, 1998.

[50] J. Chakareski and P. Frossard, *Rate-Distortion Optimized Distributed Packet Scheduling of Multiple Video Streams Over Shared Communication Resources,* IEEE Transactions on Multimedia, Special Issue on Distributed Media Technologies and Applications, vol. 8, No 2, pp. 207-218, April 2006.

[51] Z. He and S. K. Mitra, *A Unified Rate-Distortion Analysis Framework for Transform Coding,* IEEE Transactions on Circuits and Systems for Video Technology, vol. 11, no. 12, December 2001.

[52] N. Itaya and S. Kasahara, *Dynamic Parameter Adjustment for Available-Bandwidth Estimation of TCP in Wired-Wireless Networks,* Computer Communications, vol. 27, pp. 976-988, 2004.

[53] A. Balk, M. Gerla, M. Sanadidi, D. Maggiorini, *Adaptive Video Streaming: Pre-encoded MPEG-4 with Bandwidth Scaling,* Computer Networks: The International Journal of Computer and Telecommunications Networking archive Volume 44 , Issue 4, 2004.

[54] A. Aguiar, C. Hoene, J. Klaue, H. Karl, H. Miesmer, and A. Wolisz, *Channel-aware Schedulers for VoIP and MPEG4 based on Channel Prediction,* Proc. 8th International Workshop on Mobile Multimedia Communications (MoMuC'03), October 2003.

[55] J. Klaue, J. Gross, H. Karl, and A. Wolisz, *Semantic-Aware Link Layer Scheduling of MPEG-4 Video Streams in Wireless Systems,* Proc. 3rd Workshop on Applications and Services in Wireless Networks (ASWN), Bern, Switzerland, July 2003.

[56] G. B. Akar, N. Akar, E. Gurses, *Selective Frame Discarding for Video Streaming in TCP/IP Networks,* Packet Video, 2003.

[57] N. Laoutarisa, B. Houdt, and I. Stavrakakis, *Optimization of a Packet Video Receiver under Different Levels of Delay Jitter: an Analytical Approach,* Elsevier Performance Evaluation, vol. 55, issues 3-4, pp. 251-275, 2004.

[58] A. M. Safwat, H. S. Hassanein, H. T. Mouftah, *Energy-aware Routing in MANETs: Analysis and Enhancements,* Proc. 5th ACM International Workshop on Modeling, Analysis and Simulation of Wireless and Mobile Systems, pp. 46-53, September 2002.

[59] M. Krunz, A. Muqattash, and S. Lee, *Transmission Power Control in Wireless Ad Hoc Networks: Challenges, Solutions, and Open Issues,* IEEE Network, September/October 2004.

[60] J. Gomez and A.T. Campbell, *A Case for Variable-range Transmission Power Control in Wireless Multihop Networks,* Proc. IEEE INFOCOM, vol. 2, pp. 1425-1436, December 2004.

[61] G. Gaertner and V. Cahill, *Understanding Link Quality in 802.11 Mobile Ad Hoc Networks,* IEEE. Internet Computing, vol. 8, pp. 55-60, 2004.

[62] J. Li, C. Blake, D. De Couto, H. Lee, and R. Morris *Capacity of Ad Hoc Wireless Networks,* Proc. 7th ACM International Conference on Mobile Computing and Networking, Rome, Italy, pp. 61-69, July 2001.

[63] X. Chen, D. Jayalath and H. M. Jones, *A Cross-Layer Design for mobile AD HOC Networking in Fast Fading Channels,* in Proc. of WITSP05, pp. 159164, December 2005.

[64] B. Sklar, *Rayleigh Fading Channels in Mobile Digital Communication Systems Part 1: Characterization,* IEEE Communication Magazine, July 1997.

[65] A. Tonnesen, *Olsr Daemon,* http://www.olsr.org/.

[66] Jong-Suk Ahn, Seung-Wook Hong, and John Heidemann, *An Adaptive FEC Code Control Algorithm for Mobile Wireless Sensor Networks,* Journal of Communications and Networks, 7 (4), pp. 489-499, 2005 (to appear).

[67] M. Jiang, R. Jan, C. Wang *An Efficient Multiple-path Routing Protocol for Ad hoc Networks,* Computer Communications, 25(5), pp. 478-484, 2002.

[68] J. Liu, F. Sailhan, D. Sacchetti, and V. Issarny, *Group Management for Mobile Ad hoc Networks: Design, Implementation and Experiment,* Proc. 6th IEEE International Conference on Mobile Data Management (MDM'2005), May 2005.

[69] RFC 2205, *Resource Reservation Protocol,* www.ietf.org/rfc/rfc2205.txt.

[70] Q. Ren, W. Guo, *A Novel Medium Access Control (MAC) Protocol for Ad Hoc Network,* Proc. 17th International Conference on Advanced Information Networking and Applications (AINA'03), 2003.

[71] J. Sheu, C. Liu, S. Wu, Y. Tseng, *A Priority MAC Protocol to Support Real-Time Traffic in Ad Hoc Networks,* Wireless Networks, 10(1), pp. 61-69, 2004.

[72] T. Hui, Y. Yang, H. J. Dong, Z. Ping, *A MAC Protocol Supporting Multiple Traffic over Mobile Ad Hoc Networks,* IEEE VTC Spring, 2003.

[73] Chen, S., and K. Nahrstedt, *Distributed Quality-of-Service Routing in Ad-hoc Networks,* IEEE Journal on Selected Areas in Communications, vol. 17, no. 8, pp. 1488-1505, 1999.

[74] H. Badis, I. Gawedzki and K. Al Agha *QoS Routing in Ad hoc Networks Using QOLSR with no Need of Explicit Reservation,* IEEE Vehicular Technology Conference, Los Angeles, USA, September 2004.

[75] E. Visotsky, Y. Sun, V. Tripathi, M. L. Honig, and R. Peterson, *Reliability-Based Incremental Redundancy with Convolutional Codes,* IEEE Transactions on Communications, vol. 53, no. 6, pp. 987-997, 2005.

[76] R. Kays, K. Jostschulte, W. Endemann, *Wireless Ad-Hoc Networks with High Node Density for Home AV Transmission,* IEEE Trans. on Consumer Electronics, vol. 50, no. 2, pp. 463-471, 2004.

[77] F. Hou, Pin-Han Ho, Xuemin, and Y. Zhang, *Performance Analysis of Differentiated ARQ Scheme for Video Transmission over Wireless Networks,* Proc. 1st ACM workshop on Wireless Multimedia Networking and Performance Modeling, 2005.

[78] T. Pham Van, *Efficient Relaying of Video Packets over Wireless Ad hoc Devices,* Proc. 8th annual IEEE Wireless and Microwave Technology Conference (IEEE WAMICON'06), Clearwater, Florida, USA, Dec 4-5, 2006.

[79] T. Pham Van, *Proactive Ad hoc Nodes for Real-time Video,* Proc. 10th IEEE International Conference on Communications Systems (IEEE ICCS'06), Singapore, Oct 30 - Nov 1, 2006.

[80] Cisco Active Network Abstraction,

http://www.cisco.com/en/US/products/ps6776/index.html.

[81] C. R. Lin, J. S. Liu, *QoS Routing in Ad hoc Wireless Networks,* IEEE Journal on Selected Areas in Communications, vol. 17, no. 8, pp. 1426-1438, August 1999.

[82] J. Klau, *YUV CIF Video Sequences,* http://www.tkn.tu-berlin.de/research/-evalvid/.

[83] J. Chen, S.-H. G. Chan, V. O. K. Li, *Multipath Routing for Video Delivery over Bandwidth-limited Networks,* IEEE Journal on Selected Areas in Communications, vol. 22, no. 10, pp. 1920-1932, 2004.

[84] Y.S.Liaw, A.Dadej, A.Jayasuriya, *Throughput Performance of Multiple Independent Paths in Wireless Multihop Network,* Proc. IEEE International Conference on Communications, pp. 4157-4161, June 2004.

List of own publications related to this dissertation

[1] Tien Pham Van, *Video-Cooperative Design for Ad hoc Networks,* IEEE Wireless Communications and Networking Conference (IEEE WCNC'07), Hong Kong, China, March 11-15, 2007.

[2] Tien Pham Van, *Efficient Relaying of Video Packets over Wireless Ad hoc Devices,* Proceedings of the 8th annual IEEE Wireless and Microwave Technology Conference (IEEE WAMICON'06), Clearwater, Florida, USA, Dec 4-5, 2006.

[3] Tien Pham Van, *Proactive Ad hoc Nodes for Real-time Video,* Proceedings of the 10th IEEE International Conference on Communications Systems (IEEE ICCS'06), Singapore, Oct 30 - Nov 1, 2006.

[4] Tien Pham Van, *Proactive Optimization of Real-time Video,* Proceedings of International Conference on Wireless Communications, Networking and Mobile Computing 2005 (WCNM'05), IEEE Press, Wuhan, China, September 23-26, 2005.

[5] Tien Pham Van, *Real-time Video over Programmable Networked Devices,* Proceedings of the IFIP International Conference on Network and Parallel Computing (NPC'05), LNCS 3779, pp. 409-416, Beijing, China, Nov. 30-Dec. 3, 2005.

[6] Tien Pham Van, *A Practical Approach for Real-time Video Streaming,* Proceedings of the 13th International Conference on Software, Telecommunications and Computer Networks, (SoftCOM'05), Split, Croatia, September 15-17, 2005.

Chapter A
BIT ERROR RATE OVER WIRELESS MULTIHOP CONNECTIONS

Given a multi-hop connection consisting of n hops as shown in Figure A.1 below. Video packets travel along the forth direction all the way from the sender (node 1) to the receiver (node n) while NACK messages are sent from the receiver toward the sender over the back direction. Denote the bit error rate at a generic hop i that connects node $i-1$ to node i as BER_i^f and BER_i^b respectively for the forth and back direction. In case the MAC layer adopts multiple retransmission, these values are understood as the cumulated rate that is estimated for all retransmissions. Over the path, each video packet has its length fixed, denoted as L_p. In reality, NACK messages may be cut off partially as they are forwarded toward the sender, depending on how many cache-hits are found at each intermediate node (see Chapter 5). However, for simplicity, we assume that the length of a NACK message under consideration is L_r.

The probability that a packet of length L_p successfully passes hop i is calculated as:

$$P_i^p = (1 - BER_i^f)^{L_p} \tag{A.1}$$

while the probability for a NACK message of length L_r to successfully traverse the hop is:

$$P_i^n = (1 - BER_i^b)^{L_r} \tag{A.2}$$

In case of passive retransmission, only the sender node can regenerate the lost packet(s). The end-to-end successful rate for the video packet and the NACK message accordingly are as follows:

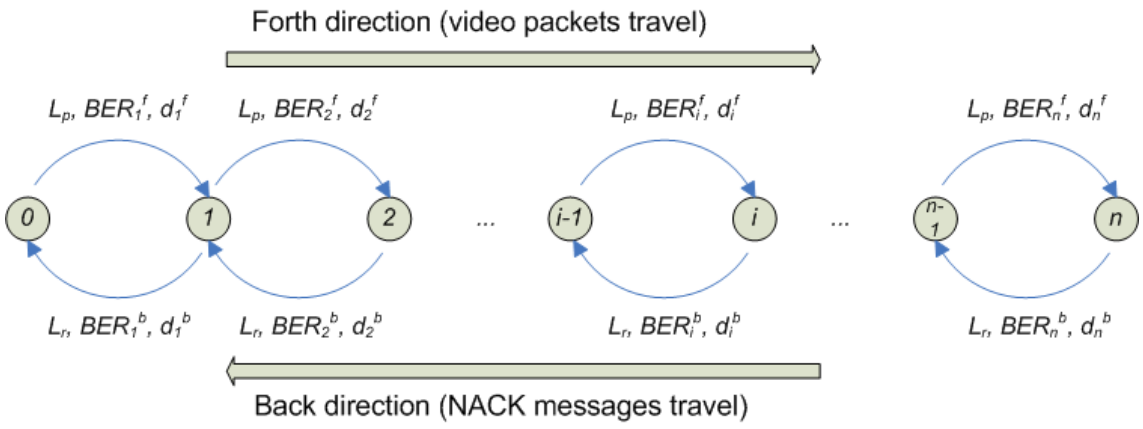

Figure A.1: Multi-hop transmission path of n nodes.

$$P^p = \prod_{i=1}^{n}(1 - BER_i^f)^{L_p} \tag{A.3}$$

$$P^n = \prod_{i=1}^{n}(1 - BER_i^b)^{L_r} \tag{A.4}$$

A retransmission round is successful when the NACK message is correctly received by the sender node and the packet successful arrives at the receiver node. So, the success rate of the retransmission round is calculated as:

$$P_{success}^{re} = \prod_{i=1}^{n}(1 - BER_i^f)^{L_p} \times \prod_{i=1}^{n}(1 - BER_i^b)^{L_r} \tag{A.5}$$

In the proactive framework, when cache-hit occurs at a node h, all the hops from 1 to h do not get involved in transmission of both the packet and the NACK message. The success rates for the packet and the NACK message are determined as:

$$P^p = \prod_{i=h+1}^{n}(1 - BER_i^f)^{L_p} \tag{A.6}$$

$$P^n = \prod_{i=h+1}^{n}(1 - BER_i^b)^{L_r} \tag{A.7}$$

The success rate for the retransmission is accordingly expressed as:

$$P_{success}^{re} = \prod_{i=h+1}^{n}(1 - BER_i^f)^{L_p} \times \prod_{i=h+1}^{n}(1 - BER_i^b)^{L_r} \tag{A.8}$$

This means that the success rate is multiplied by:

$$\Delta P_{success}^{re} = \frac{1}{\prod_{i=1}^{h}(1 - BER_i^f)^{L_p} \times (1 - BER_i^b)^{L_r}} \tag{A.9}$$

compared to the passive retransmission.

For the packet to correctly arrived at the receiver node after a retransmission round, both the NACK message and the packet must successfully pass all the hops. The success is therefore approximately estimated as:

$$\Delta P_{success}^{re} = \frac{1}{\prod_{i=1}^{h}(1 - BER_i^f)^{L_p} \times (1 - BER_i^b)^{L_r}} \tag{A.10}$$

As long as a particular packet does not get stale, it can be retransmitted again and again until it correctly reaches the receiver node. The maximum allowable number of retransmissions is determined by how long does a retransmission take. In the passive retransmission case, the number is calculated as:

$$\kappa^{passive} = \frac{D^{thres}}{\sum_{i=1}^{n}(d_i^f + d_i^b)} \tag{A.11}$$

where D^{thres} is the maximum allowed delay of video packets, d_i^f and d_i^b respectively denote the delay values on hop i for the packet and for the NACK message, in forth and back direction.

Now consider the proactive case in which cumulated delay is reduced since both the packet and the NACK messages traverse fewer hops. Actually, as retransmission of a given packet happens again and again, cache-hit may take place at some node closer and closer to the receiver in subsequent times; however, we pessimistically assume that only node h holds the packet to response to the retransmission requests. In this worst case, the maximum allowable retransmission times is calculated as:

$$\kappa^{proactive} = \frac{D^{thres}}{\sum_{i=h+1}^{n}(d_i^f + d_i^b)} \tag{A.12}$$

As a result, the overall gain factor regarding success rate can be estimated as:

$$\Delta P_{success} = \frac{1}{\prod_{i=1}^{h}(1 - BER_i^f)^{L_p} \times (1 - BER_i^b)^{L_r}} \times \frac{\sum_{i=1}^{n}(d_i^f + d_i^b)}{\sum_{i=h+1}^{n}(d_i^f + d_i^b)} \tag{A.13}$$

Chapter B

GENERATING NACK MESSAGES AT RECEIVER

As stated in Chapter 4, NACK messages are only generated when the receiver node detects loss of packet(s). To avoid excessive overhead, successfully received packets are not acknowledged. The source code file that generates NACK messages is shown below.

```
/***************************************************************
                          NackHandler.cpp  -  description
                             -------------------
    begin                : Thu Jan 26 2006
    copyright            : (C) 2006 by Tien Pham van
    email                : tienpv@nb-ram25
 ***************************************************************/

#include <NackHandler.h>
using std::list;
using std::map;
/* functions prototypes */
void predict_frame_type(short& id, char& t);
char predict_gop(short& fr_id);
char predict_pos(short& fr_id);
/* global variable */
extern map<short, MPEGFrame*> frameArray;
extern Frame_str mpegPacketArray[];
extern int s, ns;
extern sockaddr_in sa, nsa;
extern char up_host_pri[], up_host_sec[], last_segm_id;
extern short curr_id_conv, last_id_conv, expected_id;
extern pthread_mutex_t a_mutex;
extern pthread_cond_t got_request;
extern sem_t pk_sem;
short last_nacked_frame_offset;
unsigned int last_nacked_time;
extern bool too_many_forward_frames_not_arrive;
void*  nh_process(void* data)
{
    short i,  res;
    short frame_offset;
    char segm_offset, pre_fr_type; // 1: BL, 2: EL, 3: both
    bool to_bl, to_el, stalled;
```

```
  char nack_data[MAX_NACK_FRAMES+2];
  unsigned int time_now;
  timeb tm;
  char *NackBuf;
  NackBuf = (char*)malloc(NACK_SIZE * sizeof(char));
  nsa.sin_addr.s_addr = inet_addr(up_host_pri);
#ifdef DEBUG
  printf("Nack Thread started. Uphost: %s\n", up_host_pri);
#endif
  //initialization before processing NACK
  last_nacked_frame_offset = -1;
  while(1)
  {
    pthread_cond_wait(&got_request, &a_mutex); // from frame_timer
    segm_offset  = last_segm_id;
    stalled      = too_many_forward_frames_not_arrive;
    frame_offset = stalled ? expected_id % TRACE_SIZE :
     last_id_conv % TRACE_SIZE;
    pthread_cond_signal(&got_request);
    ftime(&tm);
    time_now = TIME_NOW;
    if (stalled)
    {
      printf("stalled\n");
      nack_data[0] = (predict_pos(frame_offset) << 3) | 7;
      nack_data[1] = (predict_gop(frame_offset) & 15);
    }
    else
    {
      if (frameArray[frame_offset]->segm_num == 1)
        nack_data[0] = (frameArray[frame_offset]->frame_pos << 3)
         | 7;
      else
        nack_data[0] = (frameArray[frame_offset]->frame_pos << 3)
          | (7 & segm_offset);
      //actually, gop need only 4 bits for coding
      nack_data[1] = (frameArray[frame_offset]->frame_gop & 15);
    }
    if ((frame_offset <= last_nacked_frame_offset) || stalled)
    {
      //routers should not update RTT with this nack
      nack_data[1] = 64 | nack_data[1];
    }
    res = 1; // count lost frames/segments  #
    to_bl = false; to_el = false;
```

```
      //start to scan for lost frame
      for (i = frame_offset - MAX_NACK_FRAMES; i < frame_offset-3;
        i++)
      {
        if (i < 0) continue;
        if (!frameArray.count(i))  //frame definitely lost
        {
          //predict the frame type;
          predict_frame_type(i, pre_fr_type);
          //decide if nack is sent over bl, el, or both
          switch(pre_fr_type)
          {
            case 0: //fail to predict
            case 1: //type I
            case 2: //type P
            case 4: //type H
              to_bl = true;
              to_el = true;
              break;
            case 3:
              to_el = true;
              break;
            default:
              printf("predict_frame_type() return weird value\n");
              break;
          }
          //only request for retrans if the frame still not
           missing deadline
          if ((time_now < mpegPacketArray[i].deadline -
FRAME_PERIOD) && (time_now - mpegPacketArray[i].
last_request_stamp >= NACK_PERIOD))
          {
            res++;
            nack_data[res] = (char)(((frame_offset-i-1) << 4)
             | 7);
            mpegPacketArray[i].last_request_stamp = time_now;
          }
        }
        else // found the pointer of the frame
        {
          //found, but segments may be lost
          if (frameArray[i]->segm_num > 1)
          {
            for (int j = 0; j < frameArray[i]->segm_num; j++)
            {
```

```
          if (frameArray[i]->segments[j] != NULL)
            if (frameArray[i]->segments[j]->segm_state == 0)
            {
              //this segment is really lost
              predict_frame_type(i, pre_fr_type);
              //decide if nack is sent over bl, el, or both
              switch(pre_fr_type)
              {
                case 0: //fail to predict
                case 1: //type I
                case 2: //type P
                case 4: //type B
                    to_bl = true;
                    to_el = true;
                    break;
                case 3:
                    to_el = true;
                    break;
                default:
                    printf("predict_frame_type() return weird
                     value\n");
                    break;
              }
              //only request for retrans if still not deadline
              if ((time_now < mpegPacketArray[i].deadline -
               FRAME_PERIOD)
              && (time_now - mpegPacketArray[i].segments[j].
               last_request_stamp
                >= NACK_PERIOD))
              {
                res++;
                nack_data[res] = (char)(((frame_offset-i-1) <<
                 4) | (j & 7));
                mpegPacketArray[i].segments[j].last_request_stamp
                   = time_now;
              }
            }
        }
      }
    }
  }
}
#ifndef MULTIPATH
to_bl = true;
to_el = false;
#endif
```

```
if ((res > 1) && (time_now - last_nacked_time >= NACK_PERIOD))
{
  //nsa.sin_family = AF_INET;
  nsa.sin_port = htons(nack_port);
  if (to_bl)
  {
    //add information to indicate that this nack sent over
    //bl path
    nack_data[1] = (32 | nack_data[1]);
    nack_data[res+1] = 124;
    nack_data[res+2] = 124;
    nsa.sin_addr.s_addr = inet_addr(up_host_pri);
    i = sendto(ns, nack_data, res + 3,  0,
       (struct sockaddr *) &nsa, sizeof(nsa));
    if (i < 0) printf("nack sent over BL fail\n");
    if (last_nacked_frame_offset != frame_offset)
      last_nacked_frame_offset = frame_offset;
  }
#ifdef MULTIPATH
  if (to_el)
  {
    nsa.sin_addr.s_addr = inet_addr(up_host_sec);
    nack_data[res+1] = 124;
    nack_data[res+2] = 124;
    i = sendto(ns, nack_data, res + 3,  0,
       (struct sockaddr *) &nsa, sizeof(nsa));
    if (i < 0) printf("nack sent over EL fail\n");
    if (last_nacked_frame_offset != frame_offset)
       last_nacked_frame_offset = frame_offset;
  }
#endif
  last_nacked_frame_offset = frame_offset;
}
// if it is HEART_BEAT, then send anyway
if ((frame_offset % HEART_BEAT == 0) && (res == 1)
    &&(last_nacked_frame_offset != frame_offset))
{
  nack_data[1] = (32 | nack_data[1]);
  nsa.sin_port = htons(nack_port);
  nsa.sin_addr.s_addr = inet_addr(up_host_pri);
  i = sendto(ns, nack_data, 2,  0,
     (struct sockaddr *) &nsa, sizeof(nsa));
  if (i < 0) printf("nack sent over BL fail\n");
  if (last_nacked_frame_offset != frame_offset)
    last_nacked_frame_offset = frame_offset;
```

```
      #ifdef MULTIPATH
        nsa.sin_addr.s_addr = inet_addr(up_host_sec);
        i = sendto(ns, nack_data, 2,  0,
           (struct sockaddr *) &nsa, sizeof(nsa));
        if (i < 0) printf("nack sent over EL fail\n");
      #endif
      }
      last_nacked_time = time_now;
    } // of while (1)
}
/* End of code */
```