Formal Analysis and Verification of Packet Recovery Protocols for Multicast Video

Muhammad Atif†, Nadia Mushtaq† , M. Sohaib Mahmood†, Muhammad Naeem††, Amjad Riaz†††

†Department of Computer Science and Information Technology, The University of Lahore, Pakistan
††Department of Electronics and Electrical Systems, The University of Lahore, Pakistan
†††Horizon Concepts Limited, Stockport, Manchester, SK7 5EJ, United Kingdom

Summary
Internet based transmission of real-time videos has become one of the major components of online multimedia applications. A real-time video delivery system may face several problems that include possible delay, packet loss and limited bandwidth. Unfortunately, not much work has been done on the improvement of the quality of service (QoS) to ensure the smooth video transmission. The cooperative packet recovery protocol for the multi-casting of audio/video was presented in [Maxemchuk, Nicholas F., K. Padmanabhan, and S. Lo. A cooperative packet recovery protocol for multicast video, Proceedings of 1997 IEEE International Conference on Network Protocols] and claimed significant improvement in QoS through a packet recovery mechanism. Our work formally specified this protocol together with its principal functional requirements and proved that this protocol failed to recover packet(s) in certain situations. The diagnostic traces have been reported as the proof, and this protocol has been specified in automata-theoretic formalism.

Key words:
Multicasting Video/Audio, Formal Specification, Model Checking, Packet Recovery protocol, QoS in Multimedia Transmissions.

1. Introduction

For the Internet multimedia applications such as Internet television, video conferencing, distance learning, digital libraries, multicast video transmission is required. The main problem with such applications is the heterogeneity of networks which makes it difficult to achieve a good Quality of Service (QoS). In designing video transmission protocols, the challenging issues are dealing with bandwidth limitations, packet losses and arbitrary delays in packet transferring. Moreover, excessive traffic and congestion collapse also have negative effects on QoS for video transmission protocols. To achieve acceptable visual quality, the packet loss ratio should be below a certain threshold (say 1%) but due to more packet loss the multicast video transmission protocols face the quality degradation.

Manuscript received March 5, 2018
Manuscript revised March 20, 2018

In the cooperative packet recovery protocol for audio/video multicasting described in [17], a significant improvement of QoS is claimed by error control approach which is proved a better packet-recovery mechanism of retransmission. We verify this claim and the functional requirements of the protocol by using formal analysis techniques. Formal methods offer a large potential to provide correctness measuring techniques [4]. We apply model-checking, i.e. a formal method’s technique for formal verification. During the last few years, a number of modeling and automatic verification tools for hybrid and real-time systems [8], [16], [19], [25], and [2] have been developed.

The participants of the protocol described in [17] are source, a retransmit server, a client and set of receivers. For implementation of this protocol experiment conducted via transmission between two real time locations, one at Indian Hill (IH), near Chicago, and the other at Murray Hill (MH), New Jersey. At Indian Hill live video transmission started with the objective to improve the reception for viewers in Murray Hill. The source transmits audio and video packets to all via global multicast channel. In [17], the global multicast depicts the complete multicast group and local multicast means multicasting for a specific set of receivers connected with a client. After reception of packets the retransmit server stores these packets in a buffer named retransmit buffer. The client as repair server adds these packets in a buffer named as repair buffer. In case of packet loss from source, the client sends negative acknowledge (NACK) message to retransmit server, which recovers it and again unicasts back that lost packet to the client. After that the repair server (client) transmits repaired packets to all receivers using local multicast channel.

This basic architecture is also extended to the hierarchy of retransmission servers positioned around expensive or over-utilized links. In this architecture the servers operate a NACK based reliable protocol where the receivers use a similar scheme for requesting lost packets.

2. Related Work

Various real-time systems such as [16], [19], [8], [25], [1] and [11] use formal methods and model verification approaches for proving their correctness.

UPPAAL is used to model many real-time systems for checking their reachability and validity. It is rich with various features including real-values clocks, invariants, diagnostic traces, communicating channels and committed states [13].

In [5] Kim G. Larsen et al. report the significance of features provided by UPPAAL by using it to perform formal verification of an audio protocol with bus collision.

Klaus Havelund et al. use real-time verification tool set of UPPAAL to formally analyze an Audio/Video protocol [10], in order to trace error in a complex model that involves 1998 transition steps.

In [15] Magnus Lindahl et al. report a formal analysis of a gear control system and provide an application in UPPAAL, based on a gear change algorithm, for this particular industrial case study.

Atif et al. in [3] and [2] formalize and verify a heartbeat protocol, and perform formal verification of its various variants including expanding heart beat protocol, periodic heart beat protocol, and dynamic heart beat protocol.

[17], [18], [22], [23], [9], [21] and [12] propose different algorithms and techniques to optimize the quality of audio/video transmission. Moreover several surveys such as [7], [24], [6], [14] and [20] claim the quality enhancement of audio/video transmission on the basis of system hardware.

In [17] the N. F. Maxemchuk et al. introduce a technique to improve the quality of audio/video transmission by recovering lost packets, and claim to have witnessed significant improvement through an experiment.

In our work we study and precisely describe [17], and present it formal requirements by specifying corresponding properties in UPPAAL in the form of formulas. Then we verify these properties over a designed system model and generate simulation results. These simulation results show whether a particular property is satisfied or not, and if not then verifier generates a counter-example that shows the correctness of protocol with regard to that property.

3. Introduction of Cooperative Recovery Protocol

The cooperative recovery protocol is based on a collaborative agreement between some of the receivers of audio/video multicast in a network. This protocol aims at recovering the packets lost by a subset of receivers.

The key players who take part in a simplest form of its implementation are as follows:

Source

It continuously multicasts packets with a sequence number.

Retransmit Server

It is located near to the source on same local network, and directly receives packets from the source. It is therefore, most likely to receive all the transmitted packets. This server retransmits the packets as and when it receives a retransmission request for them.

Clients

These are retransmitting clients which receive packets directly from the source but are on a different location therefore can miss packets. If a client receives a packet that has a higher sequence number than the expected sequence number, then it issues a retransmission request to the retransmit server for each lost packet. It repeatedly resends such request for a certain number of times if the lost packet is not recovered. The retransmission request is implemented as a negative acknowledgement (NACK) and issued for each lost packet. After waiting for the missing packet for a certain time period, the retransmitting client acts as a repair server and transmits the repaired stream (with all the packets in proper sequence) on a new multicast address. Such multicast is transmitted only when there exist potential receivers. In such cases, all of the receivers that are under that repair server receive that repaired signal, and the signal from the source is not transmitted to those receivers.

Moreover, by using NACKs, this protocol retains the number of control messages as few as possible, which is very useful particularly when number of repair servers grows.

Playback Buffer

It is a fixed size memory buffer based on a circular queue is used to hold the coming packets.

Receivers

These either receive multicast from the source or receive repaired stream from the associated retransmitting client.

An important characteristic of this protocol is that it improves the quality of existing MBone sessions without requiring operational changes at the source and receivers. Recovery of missing packets takes place between a subset of network players. These players multicast repaired signals on a channel other than that used by the original source signal. However, the receivers use the same software to receive both repaired signal and the signal from the original source. Moreover, this protocol does not require all the receivers on a network to use it rather only a subset of receivers use it to improve their quality of service; hence for instance an internet service provider (ISP) may use this protocol and then can sell an improved reception to its customers.

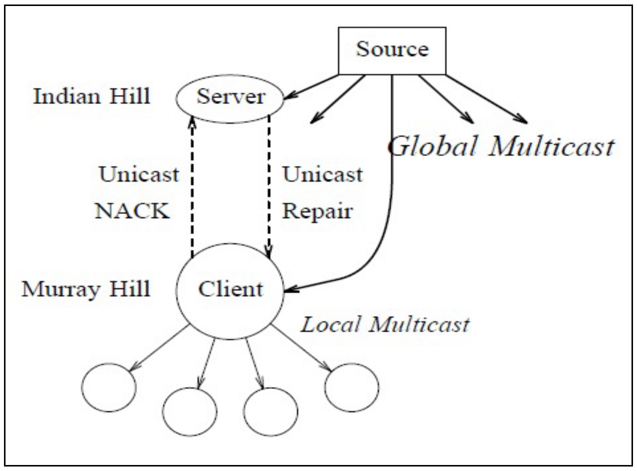

Fig. 1 Basic Architecture of Cooperative Packet Recovery Protocol [17]

Furthermore, number of active participants is very few. For instance, only a subset of receivers send retransmission requests for the lost packets to the retransmit server, and these special receivers are known as retransmitting clients, and the rest of the receivers rely on these clients for recovering the lost packets. Another important characteristic of this protocol is that it learns, and then sets and changes associated network parameters instead of predicting them. For instance if amount of multicast delay is resulting into a bad signal then it changes and corrects the delay amount to yield an improved signal.

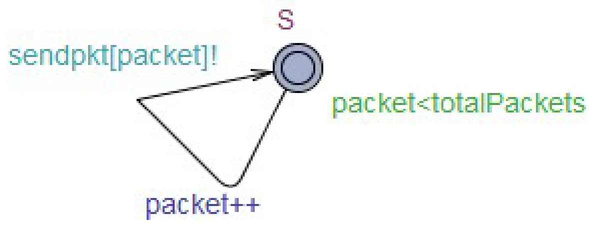

Fig. 2 Source Process

This protocol is appropriate for different types of real-time audio/video transmission systems for example news, conference sessions and courses. Such transmission systems tolerate a delay of few seconds. However, an interactive audio/video session permits only a very short delay to keep the conversation live, even if it has to compromise on video/audio quality [17].

4. Formal Specifications of Cooperative Recovery Protocol

We use the UPPAALL toolset for formal specification of the cooperative packet recovery protocol.

The UPPAAL is model-checker tool suit, which performs validation and verification of the real-time systems [17]. The UPPAAL provides a graphical interface to design an automata model for specification of a distributed system. The automata model is based on circular locations and so called transition (transition are also called edges). The locations indicates the states of a model and edges are used to connect the locations. The location can be an initial, a committed or an urgent location. In UPPAAL, an initial location is represented by a doubled circle and it indicates the stating state of a process. The actions from a committed location is taking on priority before taking another action form any other process's location. Process can move from one location to another by taking an action. The action can be an update in variables (integers, arrays, Booleans, etc.), call functions for updates or using channels (unicast or multicast) for communication with other processes. The processes use '!' and '?' symbol for sending and receiving respectively. An action can be guarded by adding a condition for execution.

In the cooperative packet recovery protocol, there are four distributed processes (a source, a client, a retransmit server and some receivers).

4.1 The Automaton of the Source Process

The automaton the source process is shown in Figure 2. The functional requirement of the source process is broadcasting packets to the client and the retransmit server. The source process contains a location named Source and an edge. We use twelve total packets in the specification of the protocol. We use a broadcast channel sendpkt [packet]! for sending packets . There is a counter for counting number of broadcast packets from the source. The source process stop broadcasting packets when the counter reaches the total number of packets.

4.2 The Automaton for the Client Process

The automaton for the client process is shown in Figure 3. The client process contains two location named Client and Sening2Receiver and five edges. The Client is an initial location and the Sening2Receiver is a committed location. We model the Sening2Receiver location as committed because the client must complete the action back to the Client location before executing another edge when it execute the edge from the Client location to the Sening2Receiver location. Following are the actions of the client process:

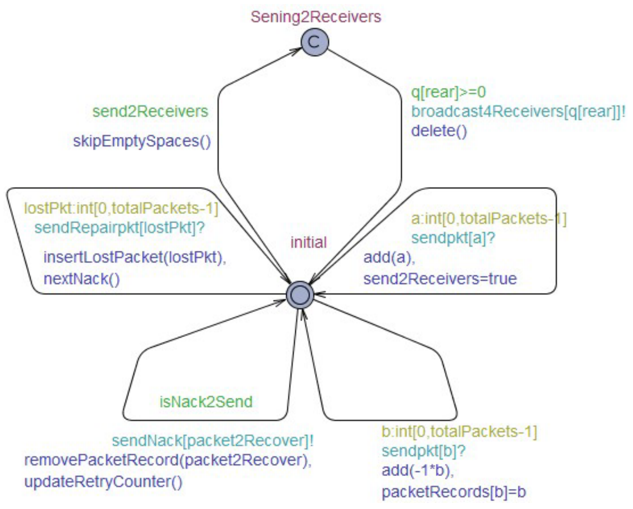

Fig. 3 The Client Process

It receives packets form the source via sendpkt[packet]?, a broadcast channel and add the received packet by calling a function add(). In add() function, we add received packet in the playback buffer of the client and update the buffer pointers. The buffer pointers point out the packets to send and receive. A Boolean variable send2Receivers is also becomes true to indicate the presence of packet(s) in the clients buffer for broadcasting.

The bottom right edge of the client process is also synchronized with the source for receiving packets but this action indicate the packet is lost by the client. It adds a negative sequence number in the playback buffer for a lost packet and also maintain the record of lost packets for recovery process.

The client process executes the bottom left edge for sending a NACK request to the retransmit server for recovery of a lost packet via sendNack[packet2Recover]! channel. This edge is guarded with the isNack2Send Boolean variable and it is true when there are lost packet(s) in the playback buffer.

The middle left edge of the client process executes to receive a repair packet form the retransmit server via sendRepairpkt[lostPkt]? channel. In this action, the client adds that repaired packet in the playback buffer by calling insertLostPacket(lostPkt) function and calls nextNack() function to check for another lost packet in the playback buffer. It also calls removeNackRecord(lostPkt) function to remove the lost packet record of that lost packet on successful recovery.

In the upper action, the client executes an edge and move the Sening2Receiver committed location. In this edge, the client moves the playback buffers pointer to the successfully received packet and removes the lost packets entries for broadcasting by calling skipEmptySpaces() function. After that it executes an edge from Sening2Receiver location to the Client location for broadcasting packets to the receivers via broadcast4Receivers[]! broadcast channel. In this edge, it also calls delete() function to remove the sent packet from the playback buffer and updates buffers pointers.

4.3 The Automaton for the Retransmit Server Process

The automaton for retransmit server process is shown in Figure 4. The process contains two locations (RS and NackReceived) and four edges. RS is an initial location of the process. We model the NackReceived location is committed because the retransmit server must send a repair packet before executing another edge when it receives a NACK request for a lost packet. Followings are the actions of the retransmit server process:

It receives packets for the source process via sendpkt[a]? broadcast channel and it calls add() function to add the received packet into its repair buffer.

The process change its location from RS to NackReceived by execution an edge. In this execution, it receives a NACK request for a lost packet form the client via sendNack[t]? channel. It calls isAvailable(t) function to check the availability of the requested packet. In this function, it also update isPacketinBuffer boolean variable to true when the requested packet is available. In this edge, the process also assigns the requested packet to the lostPacket integer variable to send repaired packet to the client.

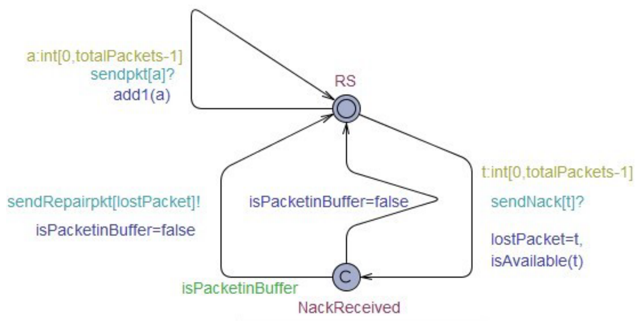

Fig. 4 Retransmit Server Process

The process executes an edge to simply move back to the RS location from the lostPacket location when the requested packet is not available in the repair buffer (isPacketinBuffer = false).

The process sends a repaired packet by executing an edge via sendRepairpkt[lostPacket]! channel and it changes its location from lostPacket location to RS location. This edge is guarded with the isPacketinBuffer variable and it only executes when this variable is true.

In this edge, the isPacketinBuffer is set to false because we have sent a repaired packet.

4.4 The Automaton for the Receiver Process

The automaton for receiver process is shown in Figure 5. The functional requirement of the receiver process is receiving packets from the client. The receiver process contains an initial location named R and an edge. It receives packets form the client via broadcast4Receivers[z]? a broadcast channel.

There are three receivers (receiver (1), receiver (2) and receiver (3)) in our system model.

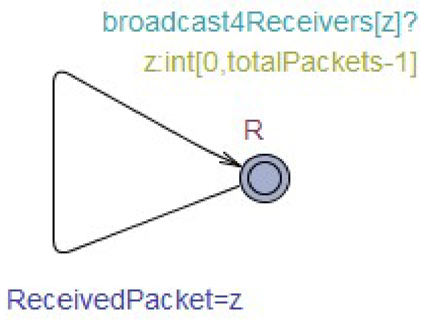

Fig. 5 Receivers Process

5. Verifying Protocol Requirements

We derive following four functional requirements (R1 - R4) for formal verification. We also give formulae for these functional requirements. Section 5.1 lists and discusses these formulae.

R1: Deadlock does not occur during the audio/video broadcast from the source.

R2: The client either receives packets transmitted by source or sends a NACK for each lost packet to the retransmit server.

R3: There exists a NACK for every lost packet, and every NACK recovers a lost packet.

R4: The client locally multicasts both the repaired signal and the original signal from the source, to associated receivers.

5.1 Formal Specifications of the Requirements

We use a constant, totalPackets to specify the total number of packets at source. The source process continually transmits a packet and increments by one, a counter variable packet. This process continues till the packet becomes greater than totalPackets. There must not occur any deadlock during the above mentioned processing (when packet <= totalPackets).

Hence we write the following formula for R1:

A[] deadlock imply Source:packet == totalPackets

This formula states that a deadlock may occur if and only if the counter packet is greater than the constant totalPackets.

In order to formalize the R2, we write the following function to keep track of the received packets and the lost packets.

```
bool allPacketsRcvdOrNacked(){

int i;

for (i=0;i<=totalPackets;i++)

   if (packetRecords[i]!=0)

     return false;

   return true;

}
```

In this function, packetRecords is a global array. The source adds to this array, the packet ID of each packet it transmits. The client removes from this array, the packet ID of each packet it either receives or misses.

The presence of a packet ID that is a non-zero value, in packetRecords array means that the associated packet is still missing from the client's buffer and its NACK message is not yet generated.

After transmitting totalPackets number of packets the packetRecords array must be empty. We write the following formula for R2:

R2 : A[] source:packet == totalPackets imply allpacketReadorNacked()

Similarly, for R3's verification we use the following function.

```
bool allNacksRecoeverOrNot(){

int i;

for (i=0;i<=totalPackets;i++)

  if (NackRecords[i]!=0)

    return false;

  return true;

}
```

This function uses a NackRecords array to keep track of NACK requests. The presence of a packet ID that is a non-zero value, in NackRecords array means that the associated requested packet is yet to be transmitted.

We express R3 as:

A[] source:packet == totalPackets imply allNacksRecoeverOrNot()

In our model there are three receivers for a client i.e. Receivers(0), Receivers(1) and Receivers(2).

To formalize R4 we write the following formula:

Client:q[0] == 1 → Receivers(0):ReceivedPacket == 1

^ Receivers(1):ReceivedP acket == 1

^ Receivers(2):ReceivedP acket == 1

This formula states that all the packets which are present in client's buffer q, become the received packets of all the associated receivers.

Table 1: Verification results of the cooperative packet recovery protocol [17]

Total Packets: 10 Retransmit buffer size: 5 Client buffer size: 5			**Computation time**	**Memory used**
R1	True	Property is Satisfied	> 15h	> Gb
R2	True	Property not Satisfied	0:56s	13,764kb – 34,988kb
R3	True	Property not Satisfied	0:015s	13,792kb – 35,032kb
R4	True	Property not Satisfied	0:016s	7,618kb – 26,532kb

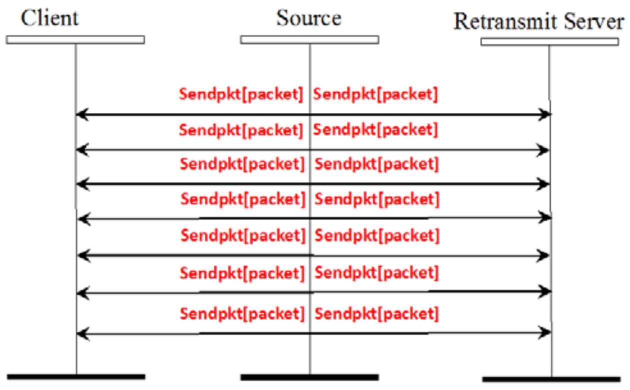

Fig. 6 Counter Example of Requirement no. 2

5.2 Verification Results

With the aim of verifying properties specified in 5.1, we use Verifier, a feature of UPPAAL model checker. The Query section of this feature generates the result of each property and displays the same in Status section. The result clearly states, whether a property is 'satisfied' or 'not satisfied'.

5.3 Verification Results with Counter Examples

We use constant global variables to generalize the buffer's size and total number of packets. The protocol [17] works under a constraint i.e. the playback buffer size is less than the retransmit buffer size. In our model, we keep these sizes equal to perform best case analysis.

R1: This property is satisfied. The simulation results show that R1 is not violated, that means maxemchuk1997cooatiperve causes no deadlock during the audio/video transmission.

R2: This property is violated and not satisfied. The simulation results of R2 show the violation of this requirement. It means that the client does not generates NACK for every lost packet. There is a counter example which shows message sequence chart of some traces. The counter example is shown in Figure 6.

R3: This property is also not satisfied, the simulation results show that R3 is not correct that means not every NACK recovers the related lost packet. The counter example which shows message sequence chart of some traces is shown in the Figure 7.

Assume that the client’s buffer size is ’n’, and the client misses a multicast of last ’n’ packets.

In such a situation the client remains ignorant and hence fails to recover and repair the last ’n’ lost multicast packets, and meanwhile transmission ends.

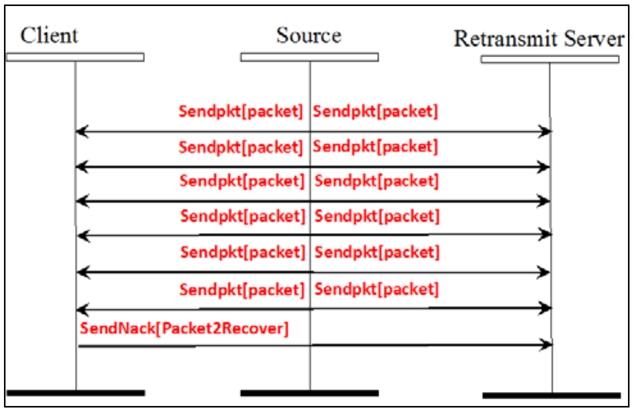

Fig. 7 Counter Example of Requirement no. 3

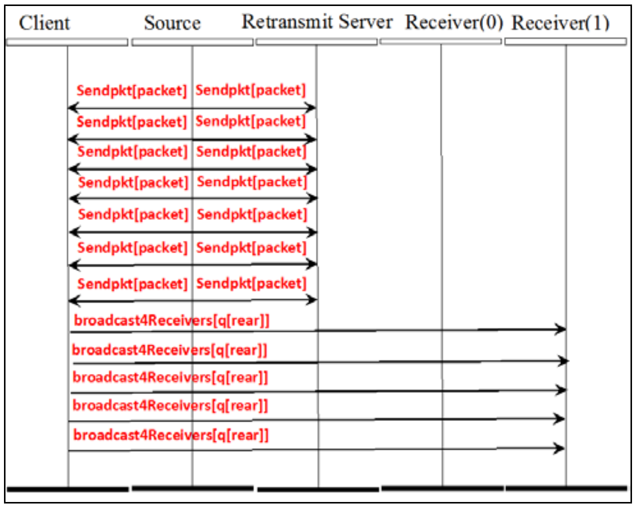

Fig. 8 Counter Example of Requirement no. 4

R4: This property is violated and not satisfied. The simulation results of R4 show the violation of this functional requirement as the client fails to send all received packets to associated receivers. There is a counter example which shows message sequence chart of some traces. The counter example is shown in Figure 8. According to this example the client process starts overwriting the previously received packets with the newly received packets.

In addition to the above violations, the [17] does not elaborate the following:

1. The way retransmit server should react after receiving a NACK from a client. If it reacts after some time interval then how this time internal is determined? Our model however, binds the retransmit server to reply immediately in response to a NACK.

2. Whether the retransmit server should maintain a separate queue for the NACKs or not.

3. The scheme to decide and prioritize which lost packets must be recovered first in a situation when multiple packet loss occurs at client side. Our model however, recovers packets on First Come First Server basis.

4. Dealing with a situation when due to any reason the retransmit server fails to timely respond to NACK requests and at the time when it retransmits the relevant packets it finds that the associated clients are transmitting to the end receiver’s packets with greater sequence numbers. In such situation the retransmit server may also encounter buffer overflows from its clients.

5. In situations when the following types of packets loss occurs in the network during video transmission.

(a) If the client receives the first half of the video and then due to a network error or due to any other reasons client fails to receive next packets with greater sequence numbers and the transmission session ends. Consequently, client cannot recover the rest of the video.

(b) Assume that ’n’ is the buffer size of retransmit server, and a client misses ’n’ packets. The client then receives ’n + 1’ packet. In such situation the client cannot repair the first lost ’n’ multicast packets due to limited buffer size.

(c) The client cannot recover the last lost packet because no packet with greater sequence number will arrive. Therefore the client remains unaware of having lost the last packet.

6. Limitations and Challenges

Modeling encounters three types of limitations. First is to limit the number of receivers to three, second is to reduce the value of the max limit of buffer size to five in the transmission and third is the limited amount of number of packets. These limitations prevent the system in generating a very huge state space and also in avoiding the state space explosion problem. These limitations are imposed due to limited memory of the machine. The machine can crash during execution of query verification phase. Model's properties are not affected due to these limitations because a few number of packets and processes can reflect the behavior of a huge system. This small system is transparent reflection of a huge system with maximum buffer size and huge number of packets with a large group of receivers processes. The main challenge is faced during the modeling phase of the protocol is to handle RTP packet representation format. The problem is that protocol uses the various fields that were available in RTP format but few of those are packet-sequence-number, retry-count and a valid tag that contain a packet and a 'Hole'(in our model hole is simulated as negative integer value). On the other side packet delay, last-request-time, packet-type are related to real-time clocks and their use is out of scope defined in our objective of verification process. This problem is addressed by initializing two arrays such as intq[bufferSize], retry_count[bufferSize] and making the 'q' array as circular buffer of packets. A source sends a packet that is added in that array after reception, but each value that stored in array is represented by packet sequence number.

7. Conclusion

We formally specified the cooperative packet recovery protocol described by [17] in UPPAAL model checker-timed automaton. Further, few properties for checking functional requirements are formalized and analyzed in UPPAAL. We know that the verifications and specifications model for cooperative packet recovery for multicast video/audio protocol can be very useful for verification of other video/audio multicast protocols. Formal specification and analysis of quality claimed in video/audio multicast transmission is the future research goal. Our basic research question is to investigate the behavior and efficiency claimed in the protocol [17]. For this purpose, we formalize the protocol using tool set UPPAAL and verify the claim of efficiency described in the protocol. We develop process and automaton, formal description on the basis of underlying concepts or informal description of this protocol. This formal model can be reused for other protocols that are related to this protocol. To the best of our knowledge, the formal model on this protocol is not designed earlier.

References

[1] Rajeev Alur and David Dill. Automata for modeling real-time systems. In Automata, languages and programming, pages 322–335. Springer, 1990.
[2] Muhammad Atif. Formal modeling and verification of distributed failure detectors. Faculty of Mathematics and
[3] Computer Science, TU/e, 10, 2011.
[4] Muhammad Atif and MohammadReza Mousavi. Formal specification and analysis of accelerated heartbeat protocols. In Proceedings of the 2010 Summer Computer Simulation Conference, pages 403–412. Society for Computer Simulation International, 2010.
[5] Christel Baier, Joost-Pieter Katoen, et al. Principles of model checking, vol. 26202649. MIT Press Cambridge, 26:58, 2008.
[6] Johan Bengtsson, WO David Griffioen, Kare J Kristoffersen, Kim G Larsen, Fredrik Larsson, Paul Pettersson, and Wang Yi. Verification of an audio protocol with bus collision using uppaal. In Computer Aided Verification, pages 244–256. Springer, 1996.
[7] Vijay K Bhagavath, Joseph Thomas O'neil, David Hilton Shur, and Aleksandr Zelezniak. Network-based service for recipient-initiated automatic repair of IP multicast sessions, May 20 2003. US Patent 6,567,929.
[8] Georg Carle and Ernst W Biersack. Survey of error recovery techniques for ip-based audio-visual multicast applications. Network, IEEE, 11(6):24–36, 1997.
[9] Pedro R D'Argenio, J-P Katoen, Theo C Ruys, and Jan Tretmans. The bounded retransmission protocol must be on time! Springer, 1997.
[10] Stephane Gruber, Jennifer Rexford, and Andrea Basso. Protocol considerations for a prefix-caching proxy for multimedia streams. Computer Networks, 33(1):657–668, 2000.
[11] Klaus Havelund, Arne Skou, Kim Guldstrand Larsen, and Kristian Lund. Formal modeling and analysis of an audio/video protocol: An industrial case study using uppaal. In Real-Time Systems Symposium, 1997. Proceedings, The 18th IEEE, pages 2–13. IEEE, 1997.
[12] Henrik Ejersbo Jensen, Kim G Larsen, and Arne Skou. Modelling and Analysis of a Collision Avoidance Protocol using SPIN and UPPAAL. BRICS, 1996.
[13] Isidor Kouvelas, Vicky Hardman, and Jon Crowcroft. Network adaptive continuous-media applications through self organised transcoding. In Proceedings of the Network and Operating Systems Support for Digital Audio and Video. Citeseer, 1998.
[14] Kim G Larsen, Paul Pettersson, and Wang Yi. Uppaal in a nutshell. International Journal on Software Tools for Technology Transfer (STTT), 1(1):134–152, 1997.
[15] Xue Li, Mostafa H Ammar, and Sanjoy Paul. Video multicast over the internet. Network, IEEE, 13(2):46–60, 1999.
[16] Magnus Lindahl, Paul Pettersson, and Wang Yi. Formal design and analysis of a gear controller. In Tools and

Algorithms for the Construction and Analysis of Systems, pages 281–297. Springer, 1998.
[17] Henrik Lonn and Paul Pettersson. Formal verification of a tdma protocol start-up mechanism. In Fault-Tolerant Systems, 1997. Proceedings., Pacific Rim International Symposium on, pages 235–242. IEEE, 1997.
[18] Nicholas F. Maxemchuk, K. Padmanabhan, and S. Lo. A cooperative packet recovery protocol for multicast video.
[19] In 1997 International Conference on Network Protocols (ICNP '97), 28-31 October 1997, Atlanta, GA, USA, pages 259–266. IEEE Computer Society, 1997.