An Ad-Hoc Smartphone-to-Smartphone Live Multimedia Streaming Application with Real-Time Constraints

Ivković, Nikola; Magdalenić, Ivan; Milić, Luka

An Ad-Hoc Smartphone-to-Smartphone Live Multimedia Streaming Application with Real-Time Constraints

Nikola Ivković, Ivan Magdalenić, and Luka Milić

***Abstract*—Widespread of smartphones which are equipped with cameras and an Internet connection allow development of applications that might be used as parking assistance, or to help humans to coordinate their actions with something that they cannot see directly. We propose an application that uses distributed architecture with carefully designed techniques to deal with Network Address Translation (NAT) issues and to allow users to, simply, temporarily and in an Ad-Hoc manner, interconnect their smartphones and achieve live video streaming with QoS feedback. Based on the proposed model, a prototype application was implemented and tested. Conducted experiments confirm the soundness of this approach.**

***Index Terms*—Video streaming, mobile application, peer-to-peer, Ad-Hoc association.**

I. Introduction

Widespread of smartphones with cameras and Internet connections permits implementation of a mobile application that can help humans in activities that require coordination with objects or events that are outside of direct sight, e.g. due to physical obstacles. Compared to specialized equipment like the interconnected camera-and-display system, the mobile application has some advantages. It is easily obtainable by simply downloading and installing the application by a user that already owns the smartphone. The mobile application is handy since people do not have to carry additional equipment, and the user does not have to spend money for buying additional hardware.

The motivating example for such application is the case of helping a driver with vehicle parking that might be difficult because of additionally attached trailers, narrow or complicated parking space, etc. In this situation, one owner of the smartphone can record a view from outside of the vehicle and live stream this video to the driver's smartphone. Unlike the live broadcast of sport events that can tolerate delays in order of tens of seconds or even minutes, the proposed application has to meet stricter real-time constraints, otherwise its usage could result with damage or injury.

To be practical, this application has to manage a temporary connection between smartphones, in an Ad-Hoc manner, and for the untrained user this has to be simple and fast.

Other use scenarios include helping humans in placing stuff on the wall that should be aligned with other objects that are not visible to them, or helping viewing an area that is hard to access but that is possible to reach with a hand and a smartphone, etc.

Streaming stored video from server to smartphone is rather common and often uses HTTP and TCP since real-time properties, although required, are not very strict [1]. Video conferencing applications like Skype have properties similar to proposed application, but they are not designed to work in an Ad-Hoc, temporary manner. Unlike the proposed application, they cannot work when both smartphones are a part of a network without Internet connectivity or when infrastructure servers are unavailable, since their main purpose is to allow communication between distant users, not to users in close proximity. Also, lacking alerting mechanisms for real-time constraint omissions makes them unsuitable for some applications.

Implementing the application with real-time properties that uses Internet infrastructure for communication is particularly challenging. Internet is a packet switching network that provides only the best-effort service to protocols in higher layers and consequently to applications. There are two research approaches to updating the current network layer protocols in order to provide versatile service models to upper layers: Differentiated Services (DiffServ) and Integrated Services (IntServ) architectures. DiffServ architecture offers different traffic classes but cannot guarantee timing properties essential for real-time applications [2], [3].

IntServ architecture can offer Quality of Service (QoS) which includes guarantees for the packet delay. Unfortunately, in order to achieve this QoS, it is necessary to allocate resources in all links and nodes (routers) on the path between the source and the destination, e.g. by using Resource Reservation Protocol (RSVP), and this approach does not scale well [4].

Both DiffServ and IntServ approaches require changes in the network core, which are not easy to implement on the global scale and certainly are not in the domain of the application developer. Implementing a real-time application in the current Internet infrastructure requires carefully designed architecture with an appropriate choice of networking protocols.

Another difficulty in the process of designing a peer-to-peer system like this stems from the fact that many end-system devices are hidden behind middleboxes which perform the Network Address Translation (NAT) [5], [6].

In this paper, we are analyzing influences on the performance of a smartphone-to-smartphone real-time streaming application and propose an application model that enables users to, simply, temporarily, and in an Ad-Hoc manner, interconnect their smartphones and achieve live video streaming with QoS feedback.

Manuscript received October 4, 2015; revised January 21, 2016.

The authors are with the Faculty of Organization and Informatics, University of Zagreb, Pavlinska 2, 42000 Varazdin, Croatia (e-mail: nikola.ivkovic@foi.hr, ivan.magdalenic@foi.hr, luka.milic@foi.hr).

doi: 10.18178/jacn.2016.4.1.195

II. Application Requirements

In the model of the proposed application two smartphones are essential parts; one is a streaming source, and the other is a multimedia player. To achieve desired architecture goals, other devices and processes might be employed to provide auxiliary infrastructure. The user of the application needs to be aware only of the smartphone part of the distributed application and choose its role as the source or the player of the live multimedia. Fig. 1 shows the simplified view of the system from the user's perspective.

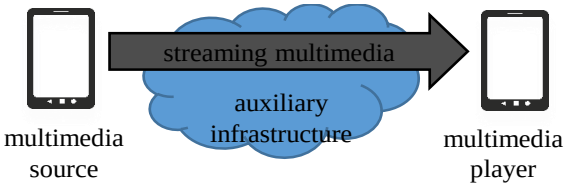

Fig. 1. Simplified view of the system architecture.

The basic design requirements for the application are the following.

The system should be simple to use for an end user and should not require any technical knowledge.

The application should allow Ad-Hoc communication between smartphones with preserved privacy, e.g. users do not have to exchange phone numbers or e-mail addresses.

It should satisfy real-time constraints, and in the case of missed deadlines the streaming player should inform the user, e.g. by drawing a red *X* across the screen or with a short beeping sound.

Helping infrastructure should scale well and be able to cope with a large number of users.

The system should function regardless of NAT devices that might complicate the communication establishing process.

A failure or a malfunction of the smartphone should not affect other users of the application.

Even if the auxiliary infrastructure is not available, the application should work whenever this is possible.

The application model should be designed to assist security.

III. System Architecture and Design Choices

The distributed application is composed from a smartphone application, a meeting server and relaying servers. The smartphone application is logically divided into two modules: a multimedia source and a multimedia player. The meeting server and the relay servers are parts of the auxiliary infrastructure.

The meeting server is used to simplify the connecting process of two smartphones and to make multimedia streaming possible when there are NAT middleboxes in the communication paths. It is also used to provide the smartphone application with a list of the relay servers if intermediation in multimedia streaming is required. In principle both TCP and UDP could be used as a transport protocol between a smartphone and the meeting server. In order to place lesser burden on infrastructure and by this to achieve better scalability, UDP is used in our model. The other reason for selecting UDP is to simplify the smartphone application design, since UDP is used to transport multimedia from the source to the multimedia player. The meeting server does not play an active role in reliable message delivery and the most of the complexity is placed on the smartphone application. Short lifetime records, with a stream identifier as the key and parameters for the associated multimedia source as other fields, are maintained at the meeting server. Parameters most essentially include a 4-tuple with private and public IP addresses and port numbers. Time to live is noted for every record to achieve a robust removal of outdated records in conformity with the soft-state protocol philosophy. To prevent DoS attacks, additional data fields might be used. Data records are stored in memory to achieve better performance, by using an associative container. Although logically one, when this is necessary or desirable, the meeting server can be deployed on multiple physical machines with data records divided through a distributed associative container.

Relay servers have a simple task of forwarding application messages, mostly from the multimedia source to the multimedia player and occasionally in the opposite direction. Because of time constraints it is necessary to have multiple servers geographically distributed to cover wide area. In order to mitigate possibility that a relay server is exploited for cyber-attacks, forwarding is performed only if the source of the message is previously registered as allowed by the designated destination.

Fig. 2 shows three possible ways by which mobile applications interact with auxiliary infrastructure.

The applications running on smartphones might communicate with the meeting server and then establish direct communication which is used for multimedia streaming. This is the case with the blue smartphones.

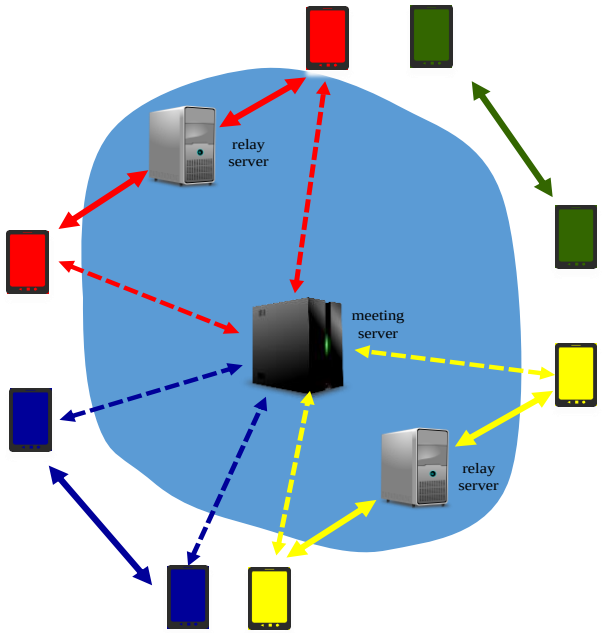

Fig. 2. Communication examples regarding auxiliary infrastructure.

In the case that direct communication between smartphones is not possible, after communicating with the meeting server, the smartphones use one relay server to stream multimedia indirectly. This is the case for red and yellow smartphone pairs on Fig. 2. When choosing an appropriate relay server,

timing constraints and server's load have to be considered.

If the auxiliary infrastructure is not available, temporarily or permanently, then smartphones establish direct communication, in the case that this is possible to achieve.

The most of the complexity and computational burden is placed on smartphone applications to achieve better scalability of auxiliary infrastructure. Since UDP does not provide a reliable data transfer, when this is necessary, e.g. in interaction with a meeting server, the mobile application has to use retransmissions and timeouts.

After performing preparatory procedures in the initialization phase, both the multimedia source and the multimedia player go through different states that can be divided into four phases: multimedia stream advertising/lookup, communication establishing, multimedia streaming, and the shutdown phase.

In the multimedia stream advertising/lookup phase, network parameters for establishing communication are exchanged or generated by using an association code. The communication establishing phase has to deal with NAT issues, and a possibility that auxiliary infrastructure is not available. In the multimedia streaming phase the multimedia source sends chunks of multimedia, the multimedia player is measuring the delay to confirm that real-time constraints are satisfied, and it plays received content until it goes to the shutdown phase.

A. Coupling the Source and the Player Smartphones

In other to establish a communication channel between the multimedia source and the multimedia player, it is necessary to set up the address parameters, IP address and the port number, that determine the network socket. The port number can be predefined, but the IP address needs to be communicated from one user to another, either directly or indirectly through some specially designed mechanism. Since NAT can change original port numbers, predefined port numbers are not practical either.

Currently, the most deployed IPv4 addresses, which use 32 bits, are represented in a human readable format with 12 decimal digits, and newer, partially deployed IPv6 addresses use 128 bits and are usually represented with a hexadecimal notation. One user could directly inform another user about his address, but this is not practical even in the case of shorter IPv4 addresses and the situation is further complicated by NAT mechanisms. To mitigate these problems, we propose an association code composed from up to 10 characters that is uniquely mapped to a 64-bit stream identifier by the scheme displayed on Fig. 3. Each character is taken from a modified base64 index table and requires 6 bits to be stored. Storing the association code of maximal length (10 characters) requires 60 bits, which leaves 4 bits to store information about the length of the association code.

Length (4 bits)	char1 (6 bits)	char2 (6 bits)	. . .	char10 (6 bits)

Fig. 3. Encoding of association code to stream identifier.

As a fallback mechanism that can be used in some situations when the auxiliary infrastructure fails, the association code, that is 8-characters long, encodes an IPv4 address and a port number. All other association codes, that are longer or shorter than 8 characters, can be chosen in any manner, e.g. randomly.

Using only 3-characters long association code, which is easy for a user to enter, can allow more than 250000 simultaneous smartphone couplings. The association code that is 5-character long allows more than 10^9 couplings which seem enough for the massive world scale application.

The multimedia source application generates an association code, whose uniqueness is assured by the auxiliary infrastructure, and displays this code to the user as shown on Fig. 4. The user that has the multimedia source communicate this code to a user of the multimedia player. Both the source and the player compute a stream identifier and use it to establish communication.

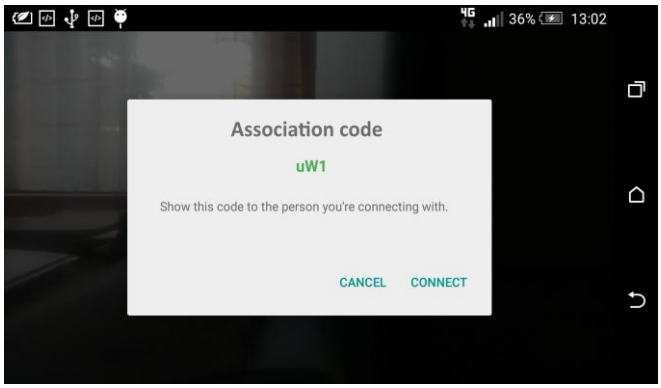

Fig. 4. Screen of the smartphone showing an association code to a user.

B. Application Messages

All the messages used by the distributed application start with the message-type field, followed by additional fields. Selected message types, together with a short description, are listed in Table I. All messages are transported by UDP, using only one bidirectional channel between communicating processes, i.e. multimedia chunks and control messages use same sockets. Messages of the type MULTIMEDIA have many fields borrowed from the Real-Time Transport Protocol (RTP), although some fields are not used since they are not relevant for this type of applications [7].

The message flow in Fig. 5 illustrates a typical message exchange, without lost messages or retransmission. At the beginning of the multimedia stream advertising phase, the multimedia source sends STREAM_ADVERTISEMENT to the meeting server. This message contains a stream identifier, a private IP address and a port number. The public IP address and the port number are available to the meeting server from the IP and UDP headers. Since the chosen stream identifier is available, the meeting server stores the data record and responds with STREAM_REG message which indicates the registration was successful.

After the association code is entered into the multimedia player, it is used to decode the stream identifier and FIND_STREAM_SRC is sent to the meeting server.

The meeting server responds with STREAM_SRC_DATA that contains public and private IP addresses and ports of the multimedia source. After that, the multimedia player requests streaming directly from the multimedia source, which is confirmed with SOURCE_READY and followed by chunks of multimedia.

TABLE I: LIST OF MAIN APPLICATION MESSAGES

Message type	Short description
PING	Sent by the mobile application to the relay server for measuring timing
PONG	Response from the relay server for measuring timing
STREAM_ADVERT	Sent by the multimedia source to the meeting server
STREAM_REG	Response from the meeting server if the stream identifier is accepted
ID_NOT_USABLE	Response from the meeting server if the stream advertisement is rejected
FIND_STREAM_SRC	Sent by the multimedia player to the meeting server
STREAM_SRC_DATA	Meeting server responds with connection-establishing parameters
STREAM_PLYR_DATA	Meeting server forwards connection-establishing parameters
STREAM_REMOVE	Sent by the multimedia source to the meeting server to delete the record
MULTIMEDIA	Multimedia chunk sent from the multimedia source to the player
REQUEST_STREAMING	Multimedia player requests streaming from the multimedia source
FORWARD_PLYR_RDY	Multimedia player sends to the meeting server to inform the multimedia source about its public IP address and port
PLAYER_RDY	Sent by the multimedia player to the multimedia source
SOURCE_RDY	Sent by the multimedia source to indicate that a message from the multimedia player was successfully received
REQ_RELAY_LIST	Requesting a relay list from the meeting server
RELAY_LIST	List of relay servers
SHUTTING_DOWN	Request to end communication
PLEASE_FORWARD	Message encapsulating another message sent to the relay server
REG_FORWARDING	Destination registers source to the relay server

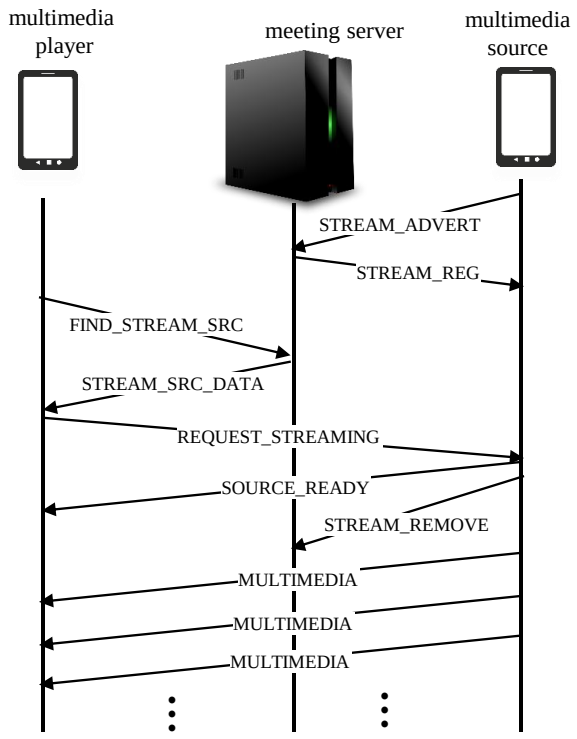

Fig. 5. Example of message flow.

In between, the multimedia source explicitly informed the meeting server to remove the record related to the streaming identifier. If this message is lost, the soft-state principle ensures that this record is removed. The meeting server accepts STREAM_REMOVE message only from the socket, identified by the IP address and the port number, which registered the stream identifier with STREAM_REG message, to make cyberattacks more difficult.

C. Dealing with the Network Address Translation

Shortage of IPv4 addresses has motivated wide deployment of NATs. Middleboxes that perform NAT change IP addresses and port numbers in headers of packets that transport data between end-system applications. Many devices behind one NAT can have different addresses from some private IP range, but all of them often use only one public IP address that is visible to the outside world. This makes it hard to initiate communication toward a device that is inside a private IP space hidden behind a NAT.

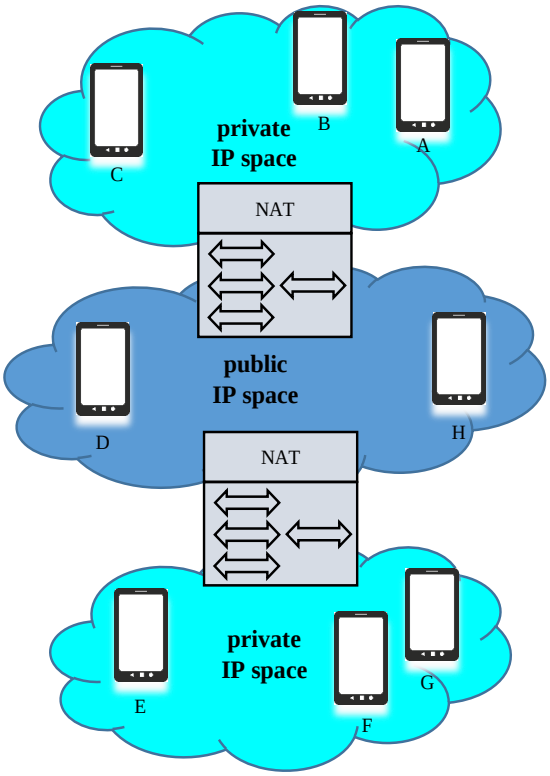

Fig. 6. Devices in public and private IP spaces.

Fig. 6 shows two networks with private IP spaces that are interconnected with the public IP space with NATs. Regarding NAT and address spaces, there are five different scenarios important for this application:

i) The multimedia source and the multimedia player are inside different private IP spaces, e.g. smartphones A and E, or smartphones F and C,
ii) The multimedia source is inside a private IP space and the multimedia player has a public IP address, e.g. smartphone B as the source and smartphone H as the player,
iii) The multimedia source has a public IP address and the multimedia player is in a private IP space, e.g. smartphone D as the source and smartphone F as the player,
iv) The multimedia source and the multimedia player have public IP addresses, e.g. smartphones D and H,

v) The multimedia source and the multimedia player are inside one private IP space, e.g. smartphones A and B, or smartphones E and G.

Although all five scenarios could be successfully resolved by using a relay server, all unnecessary intermediation in communication should be avoided to satisfy real-time requirements. The multimedia player is responsible for initiating communication with the multimedia source, and this task is performed as described by a flowchart on Fig. 7.

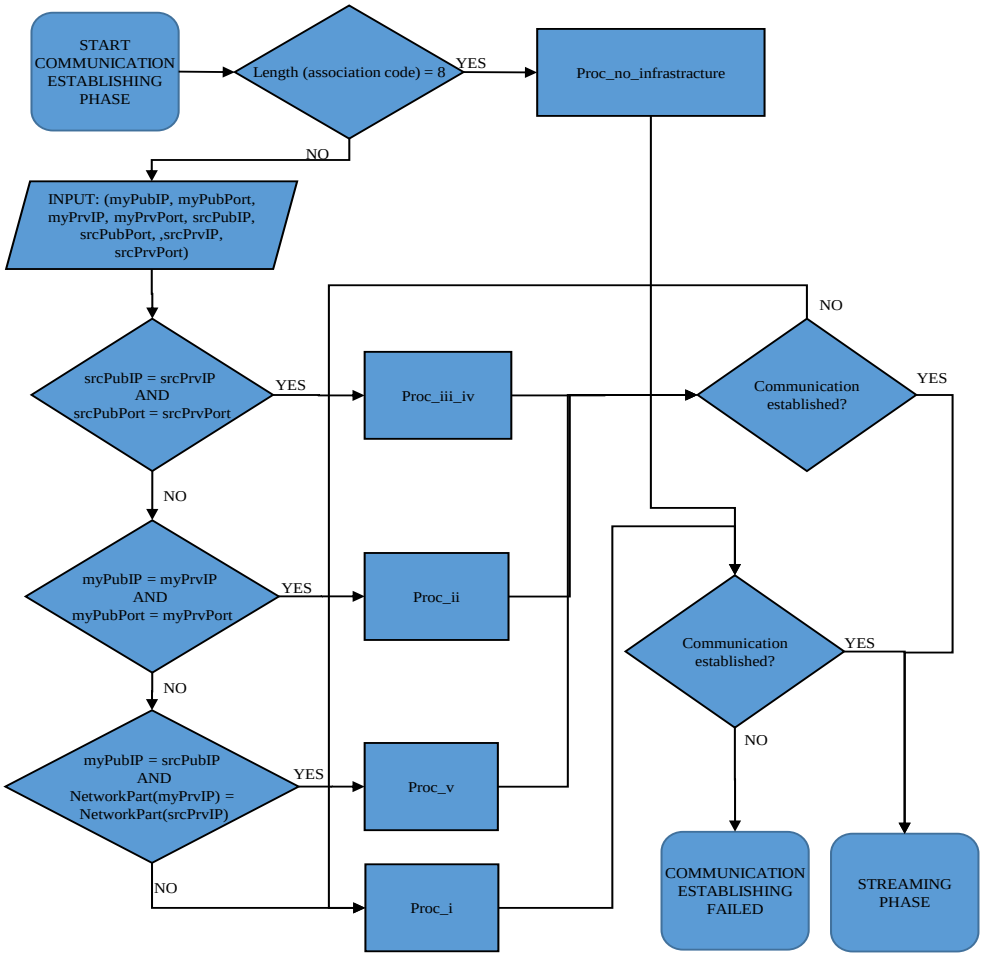

Fig. 7. Program flowchart of the communication establishing phase performed by the multimedia player.

After examining the association code, the multimedia player decides whether to use auxiliary infrastructure to receive input parameters from a meeting server, which is considered as a default behavior, or to use the backup technique by executing the procedure Proc_no_infrastructure.

In Proc_no_infrastructure, the multimedia player tries to establish direct communication by sending a REQUEST_STREAMING message to the IPv4 address and the port decoded from the 8-character long association code. If the multimedia player receives SOURCE_RDY message, communication is established and both the multimedia player and the multimedia source go to the multimedia streaming phase.

This procedure can establish communication in the case of scenarios iii, iv, and v, and could be extended to work in the case of a scenario ii, but this would require a second association code and additional user actions. This procedure cannot work with IPv6 addresses and requires rather long association codes.

When the meeting server provides necessary address parameters with STREAM_SRC_DATA message to the multimedia player, it is easy to identify the actual scenario. Proc_iii_iv is used for scenarios iii and iv. The multimedia player sends REQUEST_STREAMING message and waits for SOURCE_RDY message. If such arrives, the communication is established.

Proc_ii is performed in the case of the scenario ii. The multimedia player sends FORWARD_PLYR_RDY message to the meeting server who in turn sends PLAYER_RDY message to the multimedia source. This message contains a public IP address and a port number of the multimedia player, and so the multimedia source initiates direct communication with the multimedia player.

In the case that the scenario v is presumed, the multimedia player sends REQUEST_STREAMING to a private IP address and a port of the multimedia source at the beginning of Proc_v. If this is successful, the multimedia source responds with SOURCE_RDY and communication is established.

If the scenario i is presumed, the multimedia player starts with Proc_i. It requires assistance from the meeting server to establish communication with the multimedia source. After learning each other's public IP addresses and ports, the multimedia player and the source try to communicate directly by sending each other messages (REQUEST_STREAMING and SOURCE_RDY). Same messages are discarded and

should not reach inside the private address space, but since application uses UDP, after a few tries both NATs could allow messages to reach the right destination. If this procedure does not succeed, both the multimedia source and the multimedia player should use a relay server with a public IP address to establish indirect communication. Agreement about which relay server will they use is communicated with the help of the meeting server.

It is not always easy for the smartphone application to distinguish between scenarios i and v. In both cases, both end applications use private addresses and ports which are translated by NATs to public addresses and ports. The necessary condition for the end applications to be inside the same private IP space like in the scenario v. is that both devices have equal network part of IP address, unfortunately this is not sufficient since IP ranges behind NATs are not unique. If devices also have equal public IP addresses, then devices are indeed in the same private IP space (scenario v), but if they have different IP addresses this does not necessary imply the scenario i. Using indirect communication for multimedia streaming in the case of the scenario *v.* that is falsely recognized as scenario i, would introduce unnecessary network delays that could cause the application to fail its real-time constraints. The description of these procedures was somewhat simplified, e.g. some messages are sent periodically until predefined maximal number of attempts is reached or the expected response message is received.

D. Timing Properties

The consequence of real-time requirements is that all sorts of delays have to be considered when system architecture is designed. In general, there are processing delays, queuing delays, transmission delays, propagation delays, packetization delay, protocol induced delays, resending delays and reproduction delay. The sum of all these delays makes the total source-to-player delay.

Processing delays are occurring at the end systems (multimedia source, multimedia player and infrastructure servers), but also at routers and any other network nodes. The application developer and owners of the end systems can influence the processing delay on end systems, but routers are inside the core network out of their reach. Luckily the processing delays in the network are often negligible, often in order of microseconds [8].

Queuing delays depend on the network load and capacity, which are both out of the application developer's control, and are the main cause of variation in the end-to-end delay (jitter).

The transmission delay is proportional to message size and inversely proportional to the capacity of the link. The message size can be reduced by choosing appropriate frame resolution and compression rate. The owner of the smartphone can have influence on the access link capacity, but the other links in the network core, which are operated by internet service providers, normally have higher capacities and do not create bottlenecks. A basic prerequisite for this application to meet real-time constraints is that a link on the path that has the lowest throughput can transmit frames at least as fast as they are created.

The propagation delay is proportional to the length of the path through which signals are traveling and inversely proportional to the speed of signal propagation, which is 3×10^8m/s for wireless links and about 2×10^8m/s for wired links.

For short distances, path propagation delays are negligible, but using satellite and intercontinental links can make the proposed application unusable. This means that relay servers have to be in a limited perimeter of both smartphone users; therefore multiple relay servers are necessary for global usage.

The packetization delay occurs because the multimedia source has to wait for a while, until enough audio data or video frames are accumulated to create a packet. Choosing smaller packetization delay can reduce total source-to-player delay and if the streamed multimedia contains only video, without audio, then the video could be sent frame by frame similar to motion-JPEG (MJPEG) [9].

Using particular protocols can induce additional delays, e.g. for media access contention or because of waiting for acknowledgment. The lower layer protocols generally cannot be selected by the developer or the end user, but the appropriate transport layer protocol is an important design choice that can have a significant influence on overall real-time properties.

Transmission Control Protocol (TCP) offers the application a reliable transfer of bytes in the right order with congestion and flow control, which makes it a good choice for many applications that communicate over the Internet. Unfortunately, a reliable transfer in the right order means that, when some packet of data is lost in the network, the TCP sender will resend the packet and the destination has to wait for it. When this packet finally arrives it might not be usable anymore and the whole resending process was undesired in that case.

More troubling is the fact that all other out-of-order data packets that might arrive on the receiver's side on time are not delivered to the player's side of the application by TCP, until the re-sent packet arrives [10]. As a consequence, these out-of-order packets can miss their deadlines too. Also, the congestion control of TCP can temporarily prevent data transmission and cause an unacceptable delay. Due to these timing properties, TCP is not appropriate for live multimedia streaming with real-time constraints. This leaves the User Datagram Protocol (UDP) as a better choice for a smartphone-to-smartphone live multimedia streaming application.

Although UDP has smaller overhead than TCP, it requires more careful application design since data sent over UDP can be lost or can arrive in a wrong order. Considering the fact that some packet can arrive too late or can be lost, it requires that every multimedia packet is self-sufficient, e.g. starts with the I-frame [11].

The reproduction delay is introduced intentionally on the multimedia player side to deal with jitter. The higher reproduction delay means more buffered frames and lower probability of interruption in playing a video, but too high reproduction delay can make the total source-to-player delay too large and thus the entire application unusable.

Choosing an appropriate relay server from the list is performed by the multimedia source. The stream player measures round trip times from the relay server by sending

PING and receiving PONG messages. PING is small in this case, and PONG has a size of the expected multimedia chunk, to reflect asymmetries in links on the path. The multimedia player sends RELAY_LIST message with timing measurement to the meeting server who in turn forwards this to the multimedia source.

The multimedia source uses PING messages with the size of multimedia chunks, and receives a small PONG message from relay servers from the list, also to reflect asymmetries in links on the path. By combining timing measurements received by the multimedia player with its own, the multimedia source selects the most appropriate relay server. These measured durations can be used to predict whether the application will be able to deliver multimedia within required deadlines.

E. Security Properties

The system was designed to mitigate some cyber-attacks, as explained in Section III. The proposed application is expected to be used sporadically and in short durations which make a potential attack on particular communication between the smartphones harder to perform. The close proximity of two smartphones allows verbal communication between users, so it is easy for the user to detect suspicious activities. If higher security standards are required, then cryptographic techniques can be employed by using the Datagram Transport Layer Security (DTLS) or a specially designed lightweight cryptography protocol.

By using QR code, cryptographic keys along with other parameters can be exchanged in a secure manner through the out-of-band secure communication channel. The multimedia source would generate keys and display them on the screen together with the association code by using QR code. The smartphone with the multimedia player would use a camera to automatically take over the keys and the association code. By employing cryptographic keys, all the messages between smartphones are encrypted. The messages sent to the meeting server can be encrypted by using the public key from the certificate that is downloaded together with the smartphone application. Decryption of these messages is possible only with the private key owned by the meeting server.

IV. Conclusion

The application proposed in this paper can have different uses in everyday life. It is much cheaper and handier than specialized hardware. It can be used in an Ad-Hoc manner, without a need for users to exchange contacts, which also promotes privacy. Temporary smartphone communication is achieved by a simple few-character code, and when this is needed, the application can work without infrastructure with limited functionality. The proposed application model solves communication problems in the presence of NATs, provides guidance for solving timing issues and for an implementation of stronger security features.

A prototype application based on the proposed model and implemented for Android showed that this approach is sound.

References

[1] X. Xie, X. Zhang, S. Kumar, and L. E. Li, "piStream: Physical layer informed adaptive video streaming over LTE," in *Proc. the 21st Annual International Conference on Mobile Computing and Networking*, Paris, France, 2015, pp. 413-425.

[2] S. Blake *et al*., "An architecture for differentiated services," RFC 2475, IETF, December 1998.

[3] M. E. Villapol and J. Billington, "Internet service quality: A survey and comparison of the IETF approaches," *Telecommunications Journal of Australia*, vol. 50, no. 2, pp. 57-69, 2000.

[4] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala, "RSVP: A new resource reservation protocol," *IEEE Network Magazine*, vol. 7, no. 9, pp. 8-18, Sept. 1993.

[5] P. Srisuresh and K. Egevang, "Traditional IP network address translator (Traditional NAT)," RFC 3022, January 2001.

[6] L. Zhang, "A retrospective view of NAT," *The IETF Journal*, vol. 3, issue 2, Oct. 2007.

[7] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RTP: A transport protocol for real-time applications," RFC 3550, July 2003.

[8] J. F. Kurose and K. W. Ross, *Computer Networking: A Top-Down Approach*, 6th ed. Pearson, 2012, p. 37.

[9] L. Berc, W. Fenner, R. Frederick, S. McCanne, and P. Stewart, "RTP payload format for jpeg-compressed video," RFC 2435, October 1998.

[10] S. Kadry and A. E. Al-Issa, "Modeling and simulation of out-of-order impact in TCP protocol," *Journal of Advances in Computer Networks*, vol. 3, no. 3, September 2015.

[11] L. L. Peterson, B. S. Davie, *Computer Networks: A Systems Approach*, 5th ed. Elsavier, 2012, pp. 610-613.

Nikola Ivković received the MS degree in computer engineering and PhD degree in computer science from the Faculty of Electrical Engineering and Computing, University of Zagreb. He is a member of the research and teaching staff at the Department of Information Technologies and Computing of the Faculty of Organization and Informatics, University of Zagreb. His research interests include computational intelligence and optimization, parallel programming, formal methods, operating systems, and computer networks.

Ivan Magdalenić received the BS degree and MS degree in electrical engineering, in 2000 and 2003, respectively, and received the Ph.D. degree in computer science in 2009, all from the Faculty of Electrical Engineering and Computing, University of Zagreb. Currently, he is working as an assistant professor at the Department of Information Technologies and Computing of Faculty of Organization and Informatics, University of Zagreb. His interests include generative programming, semantic web technologies, e-business, and computer system security.

Luka Milić received the MS degree in computer science from the Faculty of Electrical Engineering and Computing, University of Zagreb, he is currently a PhD candidate. He is a member of the research and teaching staff at the Department of Information Technologies and Computing of the Faculty of Organization and Informatics, University of Zagreb. His research interests include operating systems, computer architectures, computer networks and internet of things.