USING SECURE REAL-TIME PADDING PROTOCOL TO SECURE VOICE-OVER-IP FROM TRAFFIC ANALYSIS ATTACKS

A Thesis

by

SASWAT MOHANTY

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

May 2011

Major Subject: Computer Science

USING SECURE REAL-TIME PADDING PROTOCOL TO SECURE VOICE-OVER-IP FROM TRAFFIC ANALYSIS ATTACKS

A Thesis

by

SASWAT MOHANTY

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Approved by:

Chair of Committee,	Riccardo Bettati
Committee Members,	Dmitri Loguinov
	Narasimha Annapareddy
Head of Department,	Valerie Taylor

May 2011

Major Subject: Computer Science

ABSTRACT

Using Secure Real-time Padding Protocol

to Secure Voice-over-IP from Traffic Analysis Attacks. (May 2011)

Saswat Mohanty, B.Tech., National Institute of Technology Silchar

Chair of Advisory Committee: Dr. Riccardo Bettati

Voice Over IP (VoIP) systems and transmission technologies have now become the norm for many communications applications. However, whether they are used for personal communication or priority business conferences and talks, privacy and confidentiality of the communication is of utmost priority. The present industry standard is to encrypt VoIP calls using Secure Real-time Transport Protocol (SRTP), aided by ZRTP, but this methodology remains vulnerable to traffic analysis attacks, some of which utilize the length of the encrypted packets to infer the language and spoken phrases of the conversation.

Secure Real-time Padding Protocol (SRPP) is a new RTP profile which pads all VoIP sessions in a unique way to thwart traffic analysis attacks on encrypted calls. It pads every RTP or SRTP packet to a predefined packet size, adds dummy packets at the end of every burst in a controllable way, adds dummy bursts to hide silence spurts, and hides information about the packet inter-arrival timings. This thesis discusses a few practical approaches and a theoretical optimization approach to packet size padding. SRPP has been implemented in the form of a library, libSRPP, for VoIP application developers and as an application, SQRKal, for regular users. SQRKal also serves as an extensive platform for implementation and verification of new packet padding techniques.

To my Parents, Ups and Chiku

ACKNOWLEDGMENTS

I sincerely thank my graduate advisor Dr. Riccardo Bettati for his guidance and support through all the difficult times in the project. I thank Dr. Narasimha Reddy, Dr. Dmitri Loguinov and Dr. Srinivas Shakkotai for their help and understanding. I thank my friends Tarun Jain, Bryan Graham, Blake Dworaczyk, Sandeep Yadav and Vinod Ramaswamy for their valuable input at crucial junctures. And special thanks to Upanita Goswami for giving me moral, intellectual and motivational support throughout my thesis studies.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

FIGURE Page

CHAPTER I

INTRODUCTION

Voice Over IP (VoIP) systems and applications have now become the norm of low-cost and business communications, and are used extensively for both personal and priority business conferences and talks. Privacy and confidentiality of the communication is of utmost priority, and the present industry standard is to encrypt VoIP calls using Secure Real-time Transport Protocol (SRTP) [4], aided by ZRTP [5, 7].

However, encrypted VoIP traffic remains vulnerable to traffic analysis attacks. For example, Variable Bit Rate (VBR) codecs like Speex [13] encode the VoIP conversation in a way that the size of each packet formed is directly dependent on the sound or phoneme it encodes. This information is carried over even when SRTP is used to encrypt the packets, and an intelligent adversary can use the packet size distributions to accurately infer information about the conversation. Wright *et al.* [1] describe techniques to monitor packet sizes and safely deduce the phrases spoken in a VBR-encoded conversation by an average of 50% and sometimes by as much as 90%. Similarly, the same team reports that the language spoken in such an encrypted VoIP call can also be determined by an accuracy of as much as 90% [2].

Unlike VBR codecs, Constant Bit Rate (CBR) codecs encode all the phonemes in a VoIP conversation at a constant bit rate, and all the VoIP packets are of similar sizes. So packet sizes of CBR-encoded conversation do not leak any extra information. However, it is still possible to infer information from such voice streams. For example, Lella and Bettati [12] utilize silence suppression packets, which are of smaller size and higher inter-arrival times than the regular packets, to identify talk spurt boundaries

The journal model is *IEEE Transactions on Automatic Control.*

in a Google Talk conversation. A simple context-unaware Bayesian classifier and a context-aware Hidden Markov Model (HMM) classifier are used to classify these isolated talk spurts, and both these classifiers can effectively determine the spoken phrases in a conversation, if the accurate talk spurt lengths are known.

There is no single effective approach to thwart all the traffic analysis attack techniques, because each technique utilizes a specific property of the network packet or the packet stream to extract information. For example, Wright *et al.* [1, 2] describe how the effectiveness of their attack methods can be reduced by appropriately padding the size of individual packets. Zimmermann *et al.* [5] point out that the use of techniques like variable VAD (Voice Activity Detection) hangover in VBR (Variable Bit Rate) codecs can help in mitigating such attacks. The silence-suppression based approach against CBR-encoded traffic [12] can be thwarted by intelligently adding extra dummy packets in the current talk spurt and extra bursts of packets in the silence period.

In this thesis, we design a framework for incorporating traffic padding into real-time multimedia streams. As an implementation venue, we design and develop Secure Real-time Padding Protocol (SRPP), a new Real-time Transport Protocol (RTP) [3] profile that incorporates the above suggestions and pads all VoIP sessions in a unique way to thwart traffic analysis attacks on encrypted calls. SRPP pads every RTP packet to a predefined packet size, adds dummy packets at the end of every burst in a controllable way, adds dummy bursts to hide silence spurts, and hides information about the packet inter-arrival timings. Importantly, SRPP will serve as an extensible platform, where one can add different padding techniques to mask any property used in a new traffic analysis attack, in order to reduce the effectiveness of the attack. We have implemented SRPP in the form of a library libSRPP and a standalone software application SQRKal. VoIP application developers can use libSRPP to incorporate the SRPP profile in their applications and SQRKal can be used by regular users to secure their voice calls.

This thesis has been organized as follows: Chapter II discusses the criteria required for a countermeasure against VoIP traffic analysis to be effective. In Chapter III, we discuss a few existing work related to this thesis. Chapter IV describes the general working of a VoIP application and the design details of SRPP. In Chapter V, we talk about the implementation of libSRPP and SQRKal. Chapter VI presents the evaluation of effectiveness, efficiency and correct functioning of SRPP. It also discusses a mathematical approach to find the optimal technique of padding packets. Finally, we conclude and discuss future work for this thesis in Chapter VII.

CHAPTER II

COUNTERMEASURES AGAINST VOIP TRAFFIC ANALYSIS

In a regular two-party Voice-over-IP call, we have a single sender and a single receiver. Any passive attacker will listen on the wire to capture all the media packets. If the packet contents are encrypted, the attacker has access only to the header and the encrypted payload. She can utilize a property of the encrypted packet, like the packet size, or a property of the session, like burst lengths or packet inter-arrival timings, to infer information about the VoIP call.

Timing based analysis attacks are used to identify or separate a packet flow from the output of an anonymizer or a traffic mix. Anonymizers or mixes, originally proposed by Chaum [15], provide anonymity by batching input packets from multiple sources to hide the correspondence between the input packet flow and the output packet flow. In a Timed Mix, all queued packets are forwarded after a periodic timeout. In a threshold mix, all queued packets are forwarded when the queue size surpasses a threshold. A continuous-time mix adds a random delay to each input packet, and thus disturbs the timing pattern of the original packet flow.

A type of timing based analysis attacks, called Flow Detection attacks, measure the similarity between the timing information of the input packet flow and multiple output packet flows to determine the output path or the receiver corresponding to the input packet flow. Similarity between the timing information of the packet flows is measured using probabilistic measures like correlation or information theoretic measures such as mutual information. If the input and output packet timing are specified using random variables X and Y, the correlation between X and Y gives the degree of similarity between X and Y. Mutual information signifies the amount of information given about X, if Y is known. Another model of timing based analysis attacks, called

Flow Separation attacks, uses techniques like Blind Source Separation to segregate each unique packet flow from the output of various mixes, in order to identify multiple call flows and the receivers. Blind Source Separation technique is a statistical signal processing method used to recover sources of a signal from observed mixtures.

A popular countermeasure against traffic analysis attacks is link padding, where additional packet frames are added to the original traffic at the link level of the source network stack. This ensures the presence of a cover traffic which obfuscates the original network traffic [9, 16]. Various link padding algorithms are available for specific traffic patterns. Some of these techniques suffer from inefficient utilization of the network bandwidth. Traffic morphing [8] applies convex optimization techniques to determine the output packet size distributions for a given input packet size distribution, and alter the packet size of the input packet accordingly.

In this thesis, we focus on padding the original traffic by modifying the packet sizes, burst lengths and packet inter-arrival timings for VoIP traffic to thwart traffic analysis attacks like timing based analysis attacks. Since these properties of the traffic are modified at the source, the attacker on the network does not have access to the original packet property and hence flow separation or flow detection analysis on the network will produce inaccurate results for the attacker.

For any solution against traffic analysis attacks on VoIP traffic to be effective, it must satisfy a number of criteria:
First, a countermeasure must be effective in padding VoIP packets, such that the original packet size cannot be inferred from the padded packet. A possible approach is to ensure that the mutual information between the output packet size distribution and input packet size distribution is minimum.
Second, it must be bandwidth efficient, so that the network does not get congested.

Moreover, the countermeasure must ensure that it does not increase the latency of the VoIP conversation due to the processing time required for padding and unpadding. Next, the implementation must be easily deployable and must have an intuitive interface for the user, so that it is very easy to use. Likewise, it must have an easily interpretable API so that application developers can utilize, extend or customize it with little effort. Portability to different platforms is also an important feature. Finally, any implementation of such a countermeasure must be scalable with respect to the load and network size. It must be resilient enough to perform efficiently, when the user makes many calls at the same time or in case of a conference call with many participants.

A. Padding Mechanisms

Any countermeasure against traffic analysis of encrypted content has to focus on perturbing the visible aspects of the message exchange, such as packet lengths, packet timing and talk spurt lengths. We propose a combination of the following four basic mechanisms:

1. **Packet-size Padding (PSP)**

   Many traffic analysis attacks exploit the packet size information. For example, Wright *et al.* [1, 2] use the packet size distribution to find the phrases and language spoken in VoIP flows. This type of attack can be mitigated by padding packets to different packet sizes than the original size. This is done by adding dummy information at the end of a packet. The challenge for the implementation of such a design is to obfuscate the padding information. Fig. 1 depicts the mechanism for Packet-size Padding.

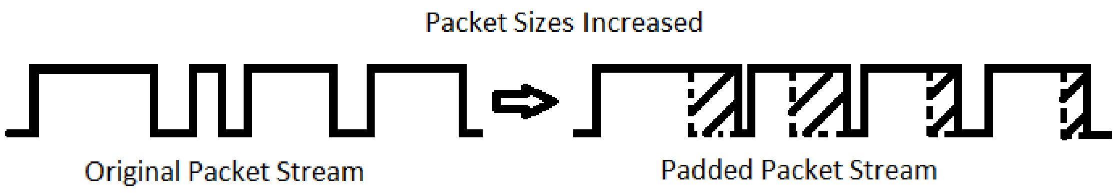


Fig. 1. Packet-size Padding (PSP)

2. **Current-burst Padding (CBP)**

   Some traffic analysis attacks utilize the length of a talk spurt in a VoIP call to estimate the spoken phrase in a VoIP conversation. So, the current burst length must be obfuscated by adding certain number of dummy packets after every packet burst. The number of dummy packets sent after every packet burst should typically depend on the burst length or talk spurt length. The longer the packet burst, the fewer the extra number of dummy packets sent. The shorter the packet burst, the higher the probability that the packet burst will be mapped to a corresponding phrase, and hence these packet bursts must be padded with more dummy packets to confuse the interceptor. Fig. 2 depicts this technique of padding current bursts.

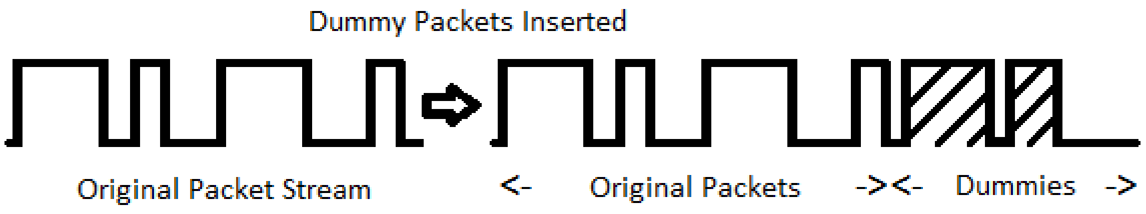


Fig. 2. Current-burst Padding (CBP)

3. **Extra-burst Padding (EBP)**

   Traffic analysis may be able to infer information from inter-burst timings as well. It is, therefore, often beneficial to "break" long intervals of silence with occasional bursts of dummy packets, as depicted in Fig. 3. This will thwart timing analysis attacks which utilize the silence information of a VoIP conversation.

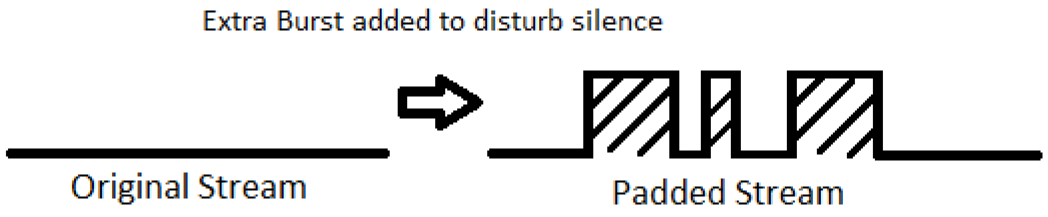


Fig. 3. Extra-burst Padding (EBP)

4. **Variable Inter-arrival Time Padding (VITP)**

   A large class of traffic analysis attacks exploit information about the inter-packet timings of a flow, as described in Chapter II. Many attacks on anonymity networks, for example, measure and correlate inter-packet time information to infer communication exchanged [10]. In order to mask the actual inter-arrival timing information of the RTP packets, a countermeasure has to add its own element of stochasticity to the inter-arrival time by selectively delaying each VoIP packet. Fig 4. depicts this mechanism.

   Care must be taken to avoid addition of extra latency to the packet. If the maximum permissible delay for a RTP packet to be sent on the network is X ms, and the current packet was received after Y ms of the last packet, then a random small delay of R ms can be added to the packet, where R ϵ {0, X-Y} ms. The maximum latency of a VoIP packet is widely considered to be 150ms, which must include the network delay. The maximum permissible delay represented by X is a design parameter, whose optimal value must be empirically determined in the evaluation phase of any new padding algorithm.

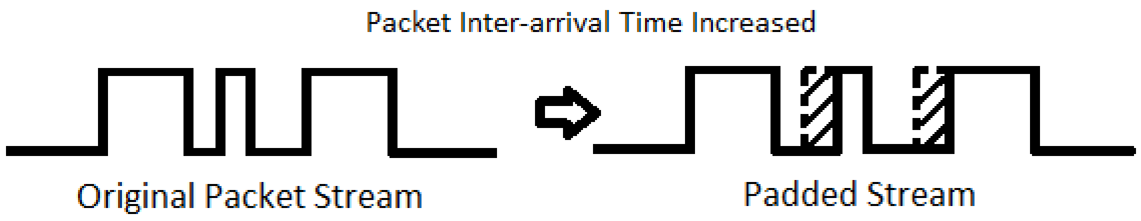


Fig. 4. Variable Inter-arrival Time Padding (VITP)

As a countermeasure, Secure Real-time Padding Protocol (SRPP) must satisfy all the above criteria and support the application of any combination of these four padding techniques.

CHAPTER III

RELATED WORK

Several papers have proposed and implemented various traffic analysis techniques against network traffic. Some papers define countermeasures against such traffic analysis attacks. Wright *et al.* [8] propose a novel technique of using convex optimization to morph a given traffic characteristic distribution into a different distribution. Given constraints on a real traffic pattern such as packet size distribution, a desired traffic pattern and traffic constraints, they determine an optimal morphing matrix, whose entities represent the probability of altering a packet with a given input size to a particular output size. For every input packet with a given size, a corresponding output packet with a different packet size is sampled from the matrix, and the packet is padded accordingly. They show that using this technique, they can reduce the efficiency of their earlier classifier [1, 2] to as less as 30%. Secure Real-time Padding Protocol (SRPP) can serve as a platform to efficiently implement and use such a morphing or padding algorithm in real time.

Danezis [10] describes statistical traffic analysis attacks based on flow correlation in continuous-time traffic mixes. This paper, along with several others [17, 14], shows the importance of information-theoretic properties like entropy in traffic analysis attacks on mix traffic, and motivates looking at optimal obfuscation of information-theoretic properties to thwart traffic analysis attacks. In SRPP, we will look at using information-theoretic techniques to optimize packet padding.

Guan *et al.* developed NetCamo [9] as a countermeasure against traffic analysis with strict quality-of-service bounds. Their system provides traffic analysis resistance by inserting padding frames at the link level and rerouting packets to confuse the attacker. NetCamo provides link padding with constant time interval, link padding with variable time interval and link padding using parasite flows as cover traffic to emulate other network packet traffic. In SRPP, we pad packets at the application level, and also give utmost importance to bandwidth efficiency as an evaluation criterion.

Wang *et al.* [11] present a dependent link padding algorithm which uses minimum sending rate to provide adequate dummy traffic to thwart traffic analysis attacks. Dummy packet frames are added at the link level depending on the input packet flow. Every dummy packet is scheduled according to the availability of input packet frames, such that the sending rate of the dummy traffic is as minimum as possible, thereby utilizing less network bandwidth. This algorithm can also be used and tested as part of SRPP. As is mentioned in the paper, this technique is not very scalable, since the rate increases with more user flows and longer sessions, and that leads to more dummy traffic and higher latency.

Our work builds on these approaches and prepares a platform to implement certain traffic padding techniques to thwart certain traffic analysis attacks on encrypted VoIP traffic. Secure Real-time Padding Protocol is designed to be extensible for application of any new padding technique and implementable in any VoIP application.

CHAPTER IV

SECURE REAL-TIME PADDING PROTOCOL

A. Architecture of VoIP Systems

A general two-party VoIP call involves the following phases of a call flow:

- **Signaling Phase:** This phase controls the creation, modification and termination of two-party VoIP calls, and is done in the following steps:
  - **Connection Establishment:** A call is setup using a signaling protocol like Session Initiation Protocol (SIP) [6], H.323 or Inter-Asterisk Exchange (IAX). This involves the sender inviting the receiver for a call and the receiver acknowledging and accepting/rejecting the request.
  - **Parameter Negotiation:** The sender and the receiver negotiate the use of various parameters like supported codecs, protocols and application/session keys, and agree upon the parameters for the ensuing call session. Most applications use techniques such as Secure DEScriptions (SDES) and Session Description Protocol (SDP) for parameter negotiation.
- **Media Transfer phase:** After the signaling is complete between the source and destination endpoints, the actual call is placed and media is transferred continuously between them. This involves sending packetized audio data from the sender to the receiver and vice versa. Fig 5. depicts the data flow at the sender side of a typical VoIP application.
  - **Voice Encoding:** At the sender side, voice input from the input device is first encoded by an appropriate audio codec. An audio codec compresses

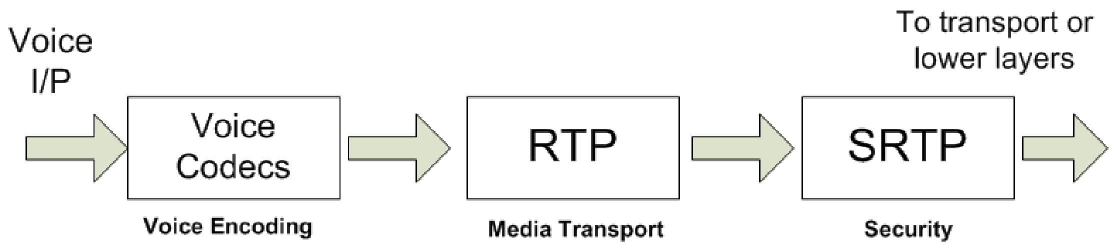


Fig. 5. Data Flow Diagram for Media Transfer Phase of a General VoIP call

and encodes digital audio data, so that it uses less storage space and bandwidth required for transmission over the network. If the codec encodes at a constant bit rate, it is called a Constant Bit Rate (CBR) codec, while a Variable Bit Rate (VBR) codec compresses audio data with a varying bit rate.

– **Media Transport Layer:** The compressed data is then packetized into Real-time Transport Protocol (RTP) packets. RTP is used as a transport protocol for audio and video data, and contains data fields for important information about the call source and destination, and other media session parameters.

– **Encryption or Security Layer:** RTP packets are then encrypted and authenticated by the Secure Real-time Transport (SRTP) layer. SRTP is a standard VoIP encryption protocol, which is part of the RTP set of protocols or RTP profiles. It encrypts the payload of the RTP packet using the negotiated session key, authenticates the entire packet by adding an authentication hash to the end of the packet and then forwards these packets to the underlying transport layer like UDP or TCP for transfer over the network.

Likewise at the receiver end in the media transfer phase, the transport layer passes SRTP packets to the SRTP layer. The authentication hash is verified for any potential tampering of the SRTP packet, and then it is decrypted into a RTP packet. The payload from the RTP packet is then passed on to be decoded by the corresponding codec, and the voice output is then sent to the output device. This is a basic working scenario of the protocols and modules used in VoIP media stacks.

If the signaling phase is completed outside of the media phase and in a dedicated signaling channel, the type of signaling is called out-of-band signaling. SIP, H.323 and IAX support out-of-band signaling. If the signaling bits or packets are sent directly from the call source to the destination in the media channel, it is called in-band or in-media signaling. ZRTP supports in-band signaling.

It is important to note here that RTP and SIP are the most popular combination of signaling and media protocols respectively. However, there are many more signaling and media protocols which are used in VoIP applications. There are many other assisting protocols like IAX, Real-time Streaming Protocol (RTSP), and Transport Layer Security (TLS) etc. There are applications which implement session layer encryption like Secure Sockets Layer (SSL) instead of using SRTP. Skype implements its own proprietary encryption protocol. An unconfirmed statistics from the year 2008 pointed out that only about 17% of VoIP applications implement any kind of encryption. However, since SIP, RTP and SRTP are the most popularly used combination of VoIP protocols for generating encrypted VoIP traffic, we have considered this for our work.

B. SRPP Design Details

As a countermeasure to VoIP traffic analysis, SRPP must satisfy all the criteria and support all the padding techniques described in Chapter II. To ensure compatibility with existing VoIP protocols and standards, SRPP has to be implemented as a new RTP profile, so that it can be added to the family of RTP-based protocols. In order to be utilized by developers of VoIP applications, SRPP must be provided in the form of a library libSRPP, which can be easily integrated in any new VoIP application as a bump-in-the-stack module. Moreover, to ensure compatibility with legacy VoIP systems, SRPP must be implemented as a standalone application called SQRKal to be used by regular users. SQRKal must be able to serve as a bump-in-the-wire module, discover any call sessions started by the legacy VoIP application and apply SRPP padding to the media session for the entire call.

1. Analysis of SRPP Design Types

Based on the requirements, a design decision has to be made about the location of SRPP in the VoIP protocol stack. Fig. 6 depicts the different types of implementation designs for SRPP protocol.

- **Type-I Implementation:** SRPP is placed between the encoding and packetization layer. In such a scenario, the voice input from the input device at the sender endpoint is encoded by a voice codec and this encoded audio data is padded using SRPP. Packet-Size Padding (PSP) can be implemented by adding dummy data to the encoded audio data, Current-Burst Padding (CBP) can be performed by adding dummy data at the end of a burst of encoded audio data and Extra-Burst Padding (EBP) can be applied similarly by inserting dummy encoded data when there is extended silence. These padded bytes of encoded

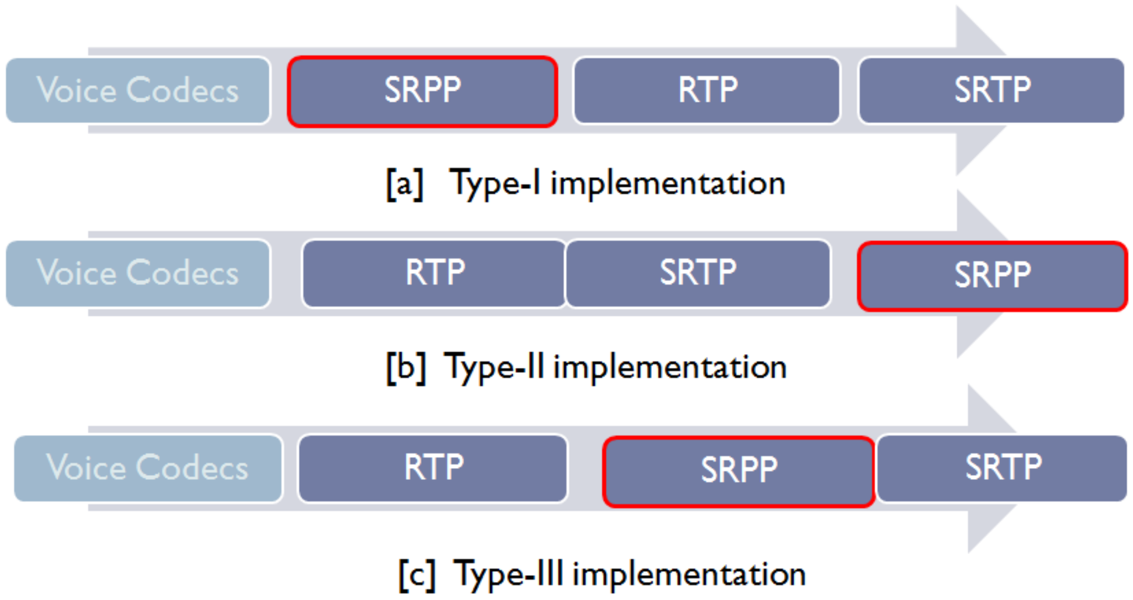

Voice Codecs
SRPP
RTP
SRTP
[a] Type-I implementation
Voice Codecs
RTP
SRTP
SRPP
[b] Type-II implementation
Voice Codecs
RTP
SRPP
SRTP
[c] Type-III implementation

Fig. 6. Types of Design of SRPP

audio data can then be packetized, encrypted and transported over the network.

Pros

– It is easy to implement since we do not need to handle any form of encryption.

– libSRPP library package will be lightweight and easily integrable with existing and new applications.

– Minimal signaling is required.

Cons

– The biggest issue with this design is that SQRKal cannot be implemented using this architecture. Since it is a bump-in-the-wire implementation, RTP packets are received from the original VoIP application and the only way to apply SRPP padding in such a design is to extract the audio data from the input RTP packet, pad this data using SRPP, form a new RTP packet for the padded data, and send it to the destination. There is extra overhead due to increased processing per packet.

- **Type-II Implementation:** SRPP is placed after the encryption layer. In such a scenario, the voice input from the input device at the sender endpoint is encoded by a voice codec, packetized into RTP packets and encrypted using SRTP. After that, SRPP applies padding by encapsulating a SRTP packet with a SRPP Header, and adding extra padding bytes for PSP in the payload of the SRPP packet. CBP can be implemented by sending extra SRPP packets at the end of a talk spurt. Likewise, EBP is implemented by sending dummy SRPP packets in case of extended periods of silence. The SRPP packets are transported over the network using a transport protocol like UDP or TCP.

  Pros

  – This design is perfect for a bump-in-the-wire scenario such as in SQRKal.

  Cons

  – The complexity of the implementation is higher since encryption and addition of new SRPP Headers with new sequence numbers needs to be handled.

  – In case of a bump-in-the-wire implementation, if the original VoIP application does not support SRTP, we will receive RTP packets instead of SRTP packets. Since RTP packets are not authenticated, there exists no need of encapsulating the entire RTP packet, since only the RTP payload can be sent in a new SRPP packet. Type-II design should be extended to include different processing for RTP packets.

- **Type-III Implementation:** In type-III implementation, the voice input from the input device at the sender endpoint is encoded by a voice codec and then packetized into RTP packets. The RTP packets are then padded using SRPP. The headers of the RTP packet can be stripped off, and only the RTP payload can be encapsulated using a SRPP Header. PSP can be implemented by adding dummy data to the payload of the new SRPP packet, while CBP and EBP can be performed similar to type-II implementation by sending dummy SRPP packets at the end of a current burst and in case of extended periods of silence in the VoIP conversation.

  Pros

  - Encryption is not required at SRPP layer, since it is handled by SRTP after SRPP processing is complete.

  Cons

  - If this design is used in a bump-in-the-wire scenario and the original VoIP application sends SRTP packets to SQRKal, SRPP will strip off the SRTP packet's headers and encapsulate its payload. At the receiver end, the SRTP packet is reconstructed from the encapsulated SRTP payload inside the SRPP packet. Since the original SRTP packet was authenticated but later modified by SRPP, it will be discarded at the receiver end. Thus, type-III implementation is unfeasible for the case of bump-in-the-wire implementation for SRTP packets.

From the above discussion, it is evident that all the three designs are insufficient. We propose a modified version of type-II implementation to serve as a design for SRPP. Fig 7 depicts the general data flow in such a design.

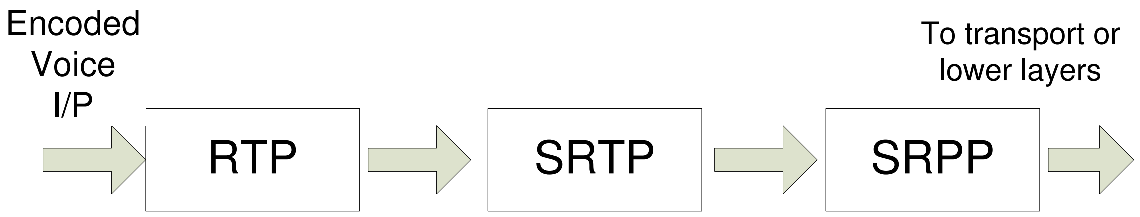


Fig. 7. Data Flow Diagram for SRPP Implementation

SRPP handles the following operations:

- **Header Obfuscation**

  At the sender side, after the voice encoding layer of the media transfer phase completes its function, the media transport layer packetizes the data into RTP packets. If the original VoIP application implements the security layer, SRTP is used to authenticate and encrypt the RTP packet. In such a scenario, the SRPP layer receives either a RTP or SRTP packet, depending on whether encryption is used or not. It must obfuscate all the visible properties of the packet to mitigate traffic analysis attacks. If it receives a RTP packet, SRPP reformats it by updating the sequence number and timestamp fields by a new sequence number and a current timestamp. Then it appends a SRPP tag consisting of padding bytes and a dummy flag signifying if the current SRPP packet is a dummy packet or not. It also contains the sequence number and timestamp of the original RTP/SRTP packet. This SRPP packet is then encrypted, authenticated and sent forward to the destination endpoint. Likewise, if the SRPP layer receives a SRTP packet, it simply encapsulates the SRTP packet in a new SRPP packet. Since SRTP packets are generally authenticated and its alteration will result in the receiver discarding the packet, we must encapsulate the entire SRTP packet. This ensures that we apply Packet-Size Padding to obfuscate the original packet size, and Extra-Burst Padding and Current-Burst Padding to obfuscate the original burst lengths.

- **Padding Identification**

  At the receiver-side SRPP module, the padding bytes and the dummy SRPP packets need to be identified and the original RTP packet must be reconstructed. If the dummy flag is set to 1, it signifies that the packet is a dummy packet and must be promptly discarded. The padding bytes are stored in the SRPP tag, and the pad count signifies the number of padding bytes added. Thus, the original packet is obtained from the received SRPP packet by removing the padding bytes, and updating the sequence number and timestamp fields with the original ones present in the SRPP tag. In the case of SRTP packets, the SRPP module simply retrieves the original SRTP packet from the payload of the received SRPP packet, and sends it forward to the encryption layer of the network protocol stack.

- **Signaling**

  Signaling between the sender and receiver is necessary to negotiate the use of SRPP and the relevant padding parameters. SRPP supports both in-band and out-of-band signaling. In in-band signaling, signaling messages are sent at the start of the media phase in the media channel. In case of SRPP, the SRPP sender module sends a HELLO SRPP message to discover if the other endpoint has a SRPP implementation. If the receiver supports SRPP, it acknowledges it by sending a HELLOACK message. The padding parameters, if any, are negotiated in the HELLO and HELLOACK messages. SRPP session teardown can be achieved by BYE and BYEACK messages and can be initiated from either side. This in-band signaling handshake, as shown in fig. 8, is perfectly feasible for both our implementations - SQRKal and libSRPP.

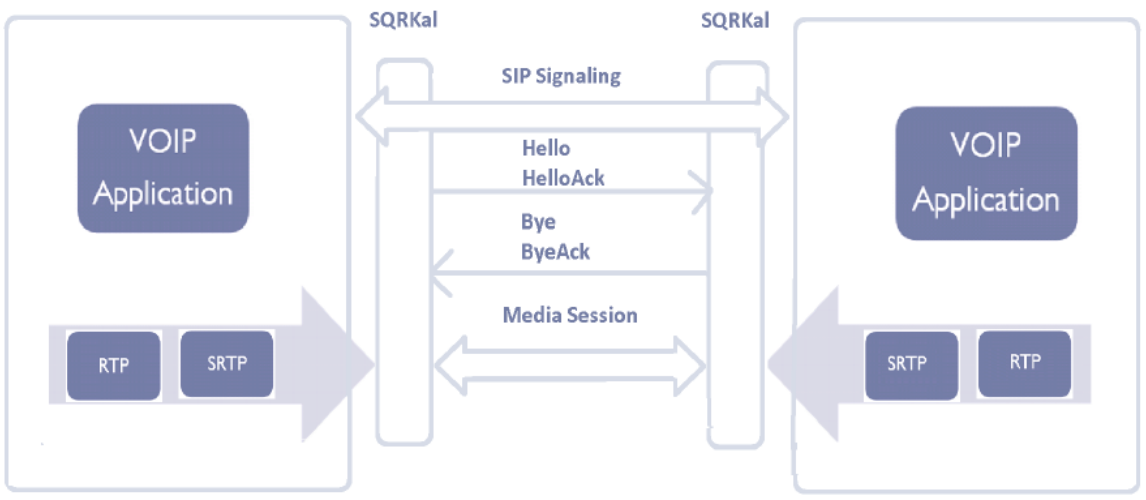


Fig. 8. In-band Signaling

SRPP also supports out-of-band signaling through SIP/SDP for integration with an external VoIP application. In out-of-band signaling, signaling messages are exchanged between the participating endpoints in a dedicated signaling channel prior to setup of the media phase. These signaling messages carry the parameters to be negotiated between both the endpoints. SIP/SDP is a popular out-of-band signaling protocol, where parameters are expressed as SDP attributes inside SIP messages. To support out-of-band signaling, SRPP provides SRPP-specific SDP attributes, which are easily extensible and the external application can embed them directly in its SDP payload.

2. Packet Format

SRPP is a RTP profile and fig. 9 represents its packet format.

The design of SRPP yields the following packet structure:

- **Header Obfuscation:** As described earlier, this is handled by the following

```
       0                   1                   2                   3
       0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+<+
     |V=2|1|X|  CC   |M|     PT      |      SRPP sequence number     | |
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ |
     |                           SRPP timestamp                      | |
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ |
     |           Synchronization source (SSRC) identifier            | |
     +=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+ |
     |            contributing source (CSRC) identifiers             | |
     |                               ....                            | |
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ |
     |      RTP extension for SRPP (used for in-band signaling)      | |
   +>+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ |
   | |                 Original RTP payload or SRTP packet ...       | |
   | |                                                               | |
   | +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ |
   | |                     SRPP packet padding  ...                  | |
   | |                               +-------------------------------+ |
   | |                               |          SRPP pad count       | |
   | +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ |
   | |P |         Dummy Flag         |    Original sequence number   | |
   +>+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+<+
   | :                 authentication tag (RECOMMENDED)              : |
   | +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ |
   |                                                                   |
   +- Encrypted Portion                       Authenticated Portion ---+
```

Fig. 9. SRPP Packet Format

fields:

- Padding Bytes: Contains the dummy information added to each packet as part of PSP.
- Pad Count: Contains the number of padding bytes inserted in the SRPP packet.
- SRPP Sequence Number: This is a new Sequence number specific to the SRPP layer.
- Payload: This contains the original RTP packet's payload or the entire original SRTP packet.
- Original Sequence Number and Timestamp: Contains the sequence num-

ber and timestamp of the original RTP/SRTP packet, which is required for the reconstruction of the original packet at the receiver endpoint.

  - As shown in the figure, the payload is encrypted and the whole packet is authenticated. The authentication hash is attached as the SRPP authentication tag.

- **Padding Identification**

  - Dummy Flag: It is set to 1 if the SRPP packet is a dummy packet and 0 otherwise.

- **SRPP Signaling**

  - RTP Extension Field for SRPP Signaling: This field is used for specifying in-band signaling message types. HELLO, HELLOACK, BYE and BYEACK SRPP signaling messages are indicated using this field.

3. Sender and Receiver Algorithms

We consider a simple two-party call between a sender and a receiver. Two SRPP timers are used for applying certain padding techniques, namely silence timer and packet timer. Silence timer must fire when Extra-Burst Padding (EBP) is required, and the packet timer must fire when Current Burst Padding (CBP) needs to be applied. In both the padding techniques, a specific number of dummy SRPP packets are sent to the destination. For every non-dummy SRPP packet, SRPP applies Packet-Size Padding (PSP) and Variable Inter-arrival Time Padding (VITP).

The flowchart for sender-side SRPP operations is depicted by fig. 10.

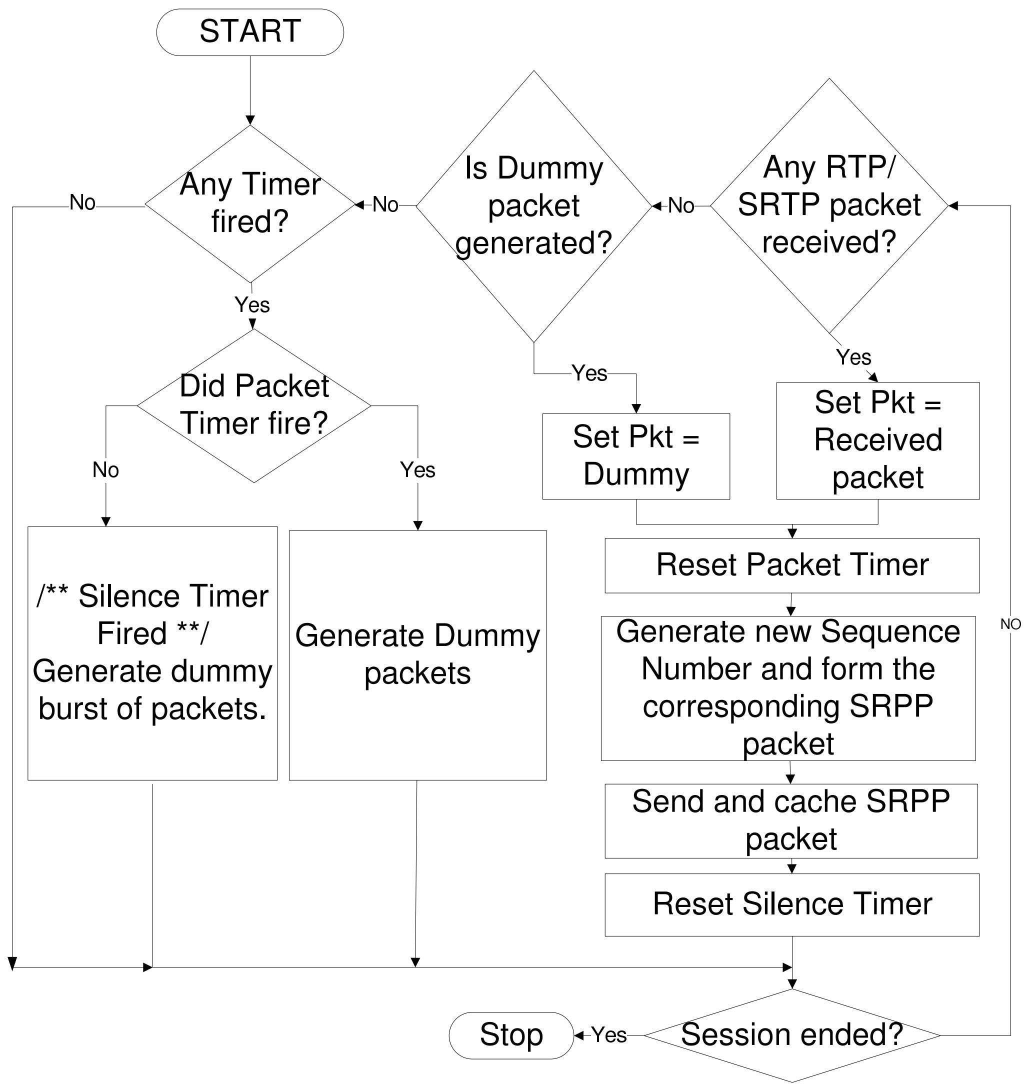

START
Any Timer fired?
No
Yes
Is Dummy packet generated?
No
Yes
Any RTP/ SRTP packet received?
No
Yes
Did Packet Timer fire?
No
Yes
Set Pkt = Dummy
Set Pkt = Received packet
Reset Packet Timer
/** Silence Timer Fired **/ Generate dummy burst of packets.
Generate Dummy packets
Generate new Sequence Number and form the corresponding SRPP packet
NO
Send and cache SRPP packet
Reset Silence Timer
Stop
Yes
Session ended?

Fig. 10. Flowchart for Sender-side Operations

For every SRPP packet to be sent to the other endpoint, the following actions are performed:

- SRPP waits for an event to occur i.e. for any of the timers to fire or for a RTP/SRTP packet to be received.
- If a packet is received, the packet timer and silence timer are reset. SRPP Padding is applied on the input packet and the new SRPP packet is sent to

the receiver. This packet is stored in a dummy cache, which serves as a local cache of sent/received SRPP packets. The dummy cache is used for generating dummy packets and padding bytes for SRPP packets.

- If the packet or the silence timer fires, a number of dummy packets for CBP or EBP respectively.

- If SRPP has not received any packet and no timers have fired, any queued dummy SRPP packets are sent to the destination.

Fig. 11 shows a flowchart representing the receiver-side SRPP process.

For every SRPP packet received from the lower layer, the following actions are performed:

1. The received packet is verified for authentication and decrypted.

2. If the packet is a dummy packet, the packet is discarded.

3. If the packet is a legitimate packet, the original RTP packet is formed by replacing Sequence number, Timestamp and Padding bit. It is then unpadded and the RTP packet is sent to the upper RTP layer.

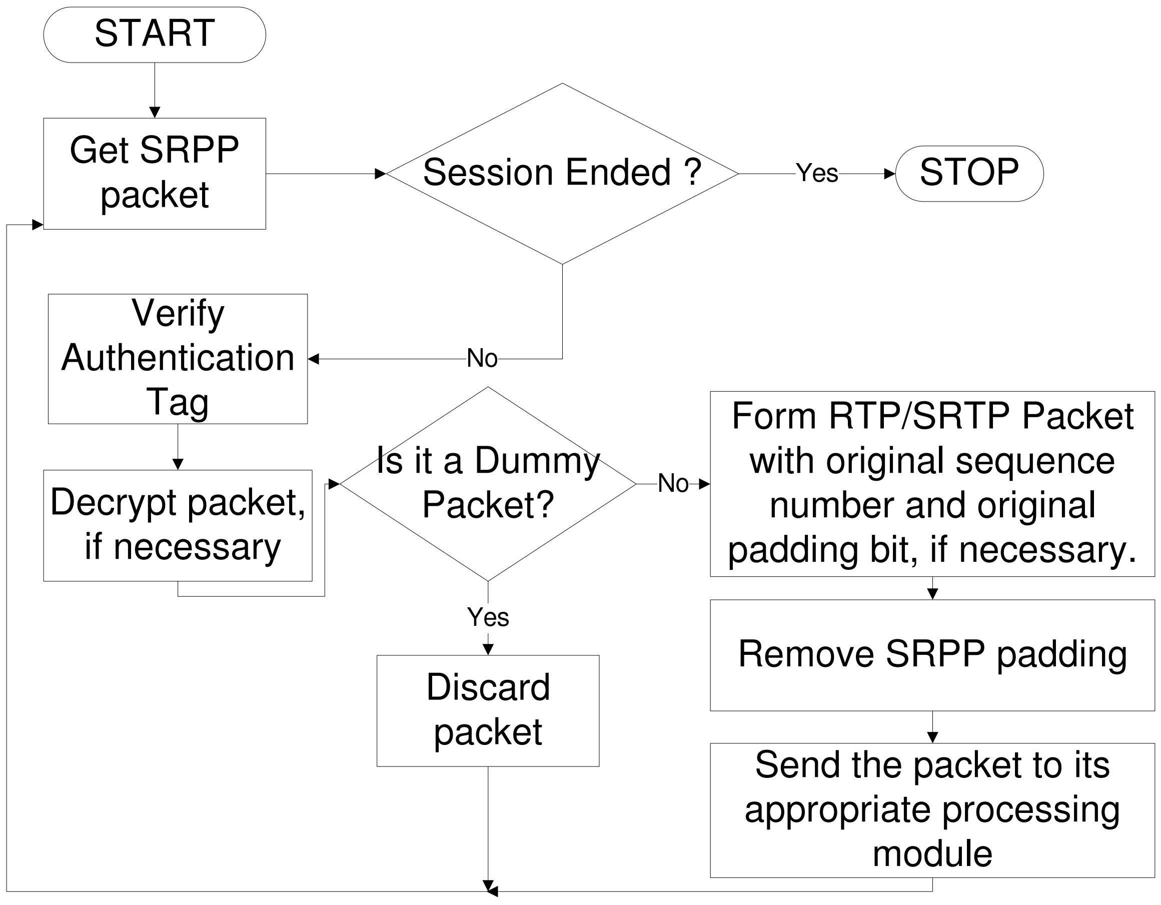

START
Get SRPP packet
Session Ended ?
Yes
STOP
No
Verify Authentication Tag
Decrypt packet, if necessary
Is it a Dummy Packet?
No
Form RTP/SRTP Packet with original sequence number and original padding bit, if necessary.
Yes
Discard packet
Remove SRPP padding
Send the packet to its appropriate processing module

Fig. 11. Flowchart for Receiver-side Operations

4. Conference Call and PC-to-phone Scenarios

A conference call involves a mixer and many endpoints, each of which can be a sender or receiver or both at any instant of time. Fig. 12 shows a RTP/SRTP mixer, a few senders and a few receivers. The general data flow can be described as:

- The senders send RTP or SRTP packets to the mixer.
- The mixer extracts the voice data from all the senders' packets.
- It then synchronizes all the audio data, repacks them as RTP/SRTP packets and sends them to all the senders and receivers.

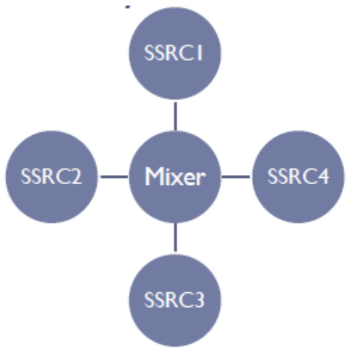


Fig. 12. Conference Call Scenario with No SRPP Support

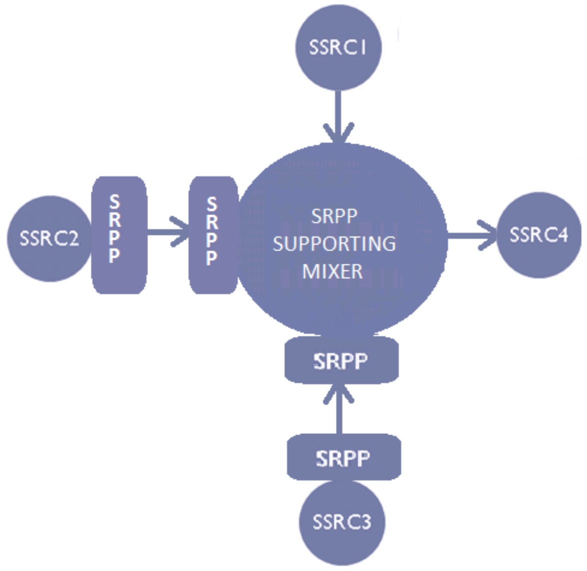


Fig. 13. Conference Call Scenario with SRPP Support at the Mixer

SRPP is implemented at the individual senders and receivers. In order for a conference call to be successful, these intermediate devices (mixers) need to support SRPP as well. Thus, as shown in Fig. 13, for every channel between the senders with SRPP support and the mixer, there is a SRPP sender module in each such sender which pads the RTP packets and a SRPP receiver module in the mixer which unpads the SRPP packets to get the original RTP packets. So the mixer is able to extract the voice data, perform its usual functions and generate the final RTP packet stream that is broadcasted to all the participants. Thus, the SRPP sender module in the mixer pads the packets and sends it to each participant. The participants unpad the packets and the VoIP media transfer is successful.

For senders which do not support SRPP, the mixer will still be able to receive the RTP/SRTP packets since SRPP signaling for these sessions will inform the mixer not to apply SRPP padding for those connections. Thus SRPP can be supported in conference calls or multicast scenarios. The mixer needs to keep a maximum of two

SRPP sessions per participant-one for receiving data and another for sending data. This increases load on the mixer and could be a possible point-of-failure in the design.

A similar scenario arises in the case of translators or Public Switch Telephone Network (PSTN) gateways. PC-to-phone can be effectively padded using SRPP only if there is support for SRPP at the translator or gateway. This is shown in the following figure.

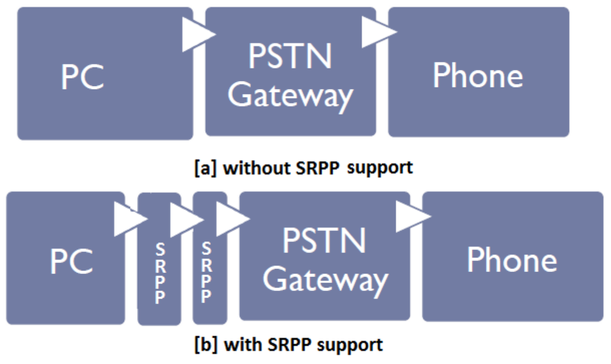


Fig. 14. PC-to-phone Call Scenario When [a] SRPP is Not Supported and [b] SRPP is Supported

When SRPP is not supported in the gateway or translator, the call is completed but the media is not padded using SRPP and is susceptible to traffic analysis attacks. Fig. 14 [b] shows the case when SRPP is supported at the PC or computer and at the PSTN gateway. In this case, the communication between the PC and the gateway are padded successfully using SRPP. Thus, the support for SRPP at the intermediate devices like translators and mixers is not mandatory for a successful call, but is recommended in order to apply SRPP padding for a secure call.

CHAPTER V

SYSTEM IMPLEMENTATION

A. Implementation Details

The library libSRPP and the standalone application SQRKal were implemented on the Ubuntu Linux v10.04 platform. The primary programming language used was C++. libSRPP API and the daemon process for SQRKal are written in C++, while the GUI for SQRKal has been designed in Java. Numerous other perl and bash scripts were used for various purposes throughout the project. The development environment comprised of Eclipse IDE CDT with Subeclipse and Doxygen plugins. Subversion, through a google code repository, was used for version control.

1. Implementation of libSRPP

The library libSRPP has the following major functions:

- Convert a RTP Packet to SRPP packet and vice versa
- Convert a SRTP Packet to SRPP packet and vice versa
- Apply different padding mechanisms
- Send and receive a SRPP Message
- Perform In-Band Signaling

To perform each of these important functionalities, libSRPP was implemented using the following major classes:

1. SRPPMessage

   This class contains information about the packet format of SRPP. It implements all the functionalities pertaining to each SRPP Message like encrypting or decrypting a SRPP message, converting a SRPP packet from network order to SRPPMessage form and vice versa.

2. SRPPSession

   A SRPP Session comprises primarily of information about the two VoIP endpoints, the RTP port used, Sequence Number of the packets, maximum packet size and the negotiated key. These information are stored in an object of SRPPSession class. It needs to be started every time a new RTP or SRTP Session is discovered or started.

3. Signaling_Functions

   This class is responsible for in-band signaling in SRPP. It implements the state machine for handling of SRPP signaling message. The major functions are sendHello, sendHelloAck, receiveHello and receiveHelloAck. This class is also called to receive information about the current state of SRPP Signaling.

4. Padding_Functions

   Padding_Functions handles the basic padding functionalities like pad, unpad, store a packet in dummy cache, prepare a dummy packet and get dummy data. These functions are widely used throughout libSRPP while padding a RTP or SRTP packet to SRPP packet or vice versa.

5. Padding_Algorithms

   This class contains the implementations of all the padding algorithms like max-random, full bandwidth etc. These algorithms are described in the next section. All the padding algorithms belong to the category of Current-Burst Padding, Packet-Size Padding, Extra-Burst Padding or Variable Inter-arrival

Time Padding. This class must be extended to include new padding algorithms.

6. RTPHeader and SRTPHeader

   These two classes, as the name suggests, store the information about the header structures of a RTP and SRTP packet respectively.

7. sdp_srpp

   This class contains the important signaling information, which are necessary to be negotiated before SRPP padding is started. These involve basic signaling parameters like maxpayloadsize and key. This is primarily used for out-of-band signaling to easily integrate it in SIP/SDP offer/answer model.

8. SRPP_Functions

   SRPP_Functions is the backbone and the most important interface of libSRPP, since it implements all the functions which are to be mostly used by a user of libSRPP. These include functions like rtp_to_srpp, srpp_to_rtp, srtp_to_srpp, srpp_to_srtp, create_srpp_message, create_and_encrypt_srpp_message, send_message and receive_message.

These classes form the library libSRPP. Any VoIP application can utilize it to incorporate the SRPP profile in his/her application.

Major functions of the libSRPP API are listed in Appendix A.

2. Padding Algorithms

A few padding algorithms have been implemented as part of SRPP. They are described as follows:

- Max-Random

  The output packet size is a random number in the interval [input packet size, maximum transmission unit]

- With Burst padding

  Current and extra burst padding are applied in order to inject dummy traffic into the stream. The number of dummy packets is a random number at this point in time.

- Full Bandwidth

  The output packet size is equal to the maximum transmission unit. All packets are padded to the maximum possible packet size and hence the system uses the maximum possible packet bandwidth.

- Gradual Ascent Padding (GAP)

  To minimize excess bandwidth used for padding, the threshold for output packet size must not be too high. Intuitively, it can be seen that the maximum output packet size must be set to a low value, and increased gradually as larger input packets are received. The maximum output packet size is always a randomly scaled value of the maximum input packet size. In other words, the output packet size is a random number in the interval [input packet size, r*(maximum input packet size received so far)], where r is a random number in the interval [0,5]. If a new packet larger than the maximum output packet size arrives, the maximum packet size is updated to a scaled value of this input packet size. The minor problem behind this technique is that every ascent of maximum packet size gives out information about a large packet coming in. This might help a passive attacker in certain scenarios.

- Slight Perturbation

  This involves padding the input packets to a random size, such that the difference between the input and output packet size is not large. The motivation behind this algorithm is to make sure that we pad the packets but to a very small extent, such that the bandwidth utilization is less. This might pose a problem where an intelligent attacker might be able to infer the original packet size information based on the correlation between the output packet size and the input packet size.

These algorithms have been implemented in SRPP and it can be extended to include newer padding algorithms.

3. Bump-in-the-wire Implementation or SQRKal

SQRKal has been implemented as a bump-in-the-wire design to intercept RTP or SRTP messages sent by a VoIP application on the local machine and pad the packet stream. Here, Twinkle [21] has been used as the third-party open-source VoIP application for testing purposes. SQRKal has been programmed in C++ as a daemon process which listens for SIP or RTP messages. SQRKal has its own SIP state machine to handle incoming and outgoing SIP messages. After parsing the standard SIP messages INVITE, 200 OK and/or ACK, it stores the information of the ensuing RTP or SRTP session. The SRPP session is started as soon as the first RTP or SRTP packet is sent from or received at the local machine. Throughout the SRPP Session, every RTP or SRTP message is padded or unpadded accordingly, and then relayed to the appropriate destination.

Fig. 15 depicts the data flow in the bump-in-the-wire implementation of SRPP.

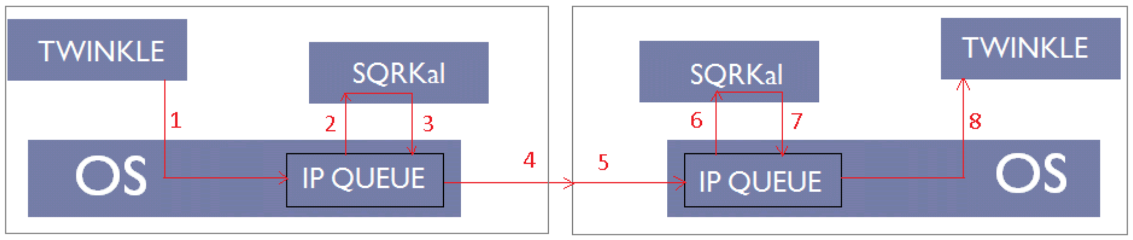


Fig. 15. Data Flow in SQRKal

The following steps describe the packet flow path when SQRKal is used:

1. Twinkle sends a SIP/RTP message to the remote machine.
2. It is queued and forwarded to SQRKal.
3. SQRKal processes the packet, creates a new packet with a spoofed header and sends it forward.
4. This new message is forwarded to the remote machine.
5. Received SIP/RTP Message is queued in the IP queue.
6. The received packet is then forwarded to SQRKal, which processes it as required.
7. The new packet is forwarded to the local machine's SIP/RTP port.
8. Twinkle receives the RTP/SRTP packet with the original payload, and processes it to receive the audio data.

SQRKal relies heavily on the correct functioning of Netfilter's iptables and ip_queue library framework [24]. An implementation related to SQRKal is the fact that it cannot function correctly if another application on the local machine is using ip_queue library. This is because it is not permissible to have more than one IP queues at the same time in a system.

The GUI of SQRKal has been implemented in Java. Fig. 16 shows a screenshot of SQRKal.

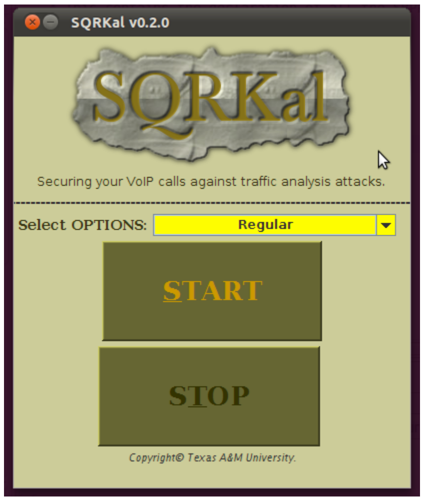

SQRKal v0.2.0
SQRKal
Securing your VoIP calls against traffic analysis attacks.
Select OPTIONS:
Regular
START
STOP
Copyright© Texas A&M University.

Fig. 16. Screenshot of SQRKal Application

CHAPTER VI

EVALUATION

SRPP has been implemented as a library libSRPP and as the application SQRKal. SQRKal operates in a bump-in-the-wire fashion. We proceed to evaluate these deliverables on the basis of evaluation criteria described earlier in Chapter II.

A. Results from the Bump-in-the-wire Implementation

The operations, effectiveness and efficiency of SQRKal as a bump-in-the-wire implementation is evaluated in this section. The implementation setup using Twinkle VoIP application and the different padding algorithms have been described in Chapter V.

1. Operation

We analyze the functioning of SQRKal in each of the following scenarios: All the wireshark [20] traces are shown in Appendix C.

- Session where we connect Twinkle to a public music server

  This is the case where the sender has SRPP support while the receiver does not support SRPP. A call is placed from our SIP account in Twinkle to a public music server at music@iptel.org. The traces verify that SQRKal tries to signal its presence to the other endpoint at the start of the media session, but it fails. No packet-level padding is performed, and the session goes on as usual. There is no degradation in quality.

- Session between two SQRKal supported endpoints implementing Max-Random algorithm

  From the trace, it was verified that the packets are padded to a random size less than the maximum packet size i.e. 1500 bytes. It was noted that there is a slight degradation in the quality of the received audio.

- Session between two SQRKal supported endpoints implementing Full Bandwidth algorithm

  From the trace, it was verified that all the packets are padded to a random size exactly equal to 1466 bytes. Quality of the received audio degraded more than the previous case.

- Session between two SQRKal supported endpoints implementing Gradual Ascent Padding algorithm

  From the trace, it was verified that the packets are initially padded to 150 bytes, and then the padding size increases as the packet size becomes greater than this threshold. The quality of received audio was satisfactory.

- Session between two SQRKal supported endpoints implementing Slight Perturbation algorithm

  From the trace, it was verified that the packets are padded with small number of dummy data. The quality of received audio was satisfactory.

- Session between two SQRKal supported endpoints implementing Dummy Bursts technique

  The trace shows that the dummy packets are indistinguishable from the original packets. It was verified that the endpoints are sending dummy packets as part of Current Burst Padding and Extra Burst Padding, from the application logs. The quality degraded slightly.

These sessions were repeated for the case of SRTP, with similar results.

2. Effectiveness

The effectiveness of SQRKal can be depicted in the following histogram. It shows that when SQRKal is not used,the packet sizes for a stream of RTP packets range between 50 to 80. When SQRKal is used in Gradual Ascent Padding (GAP) mode, the packets are padded effectively and we see SRPP Packets of sizes ranging between 145 to 220. This proves the effectiveness of SQRKal, as depicted by fig. 17.

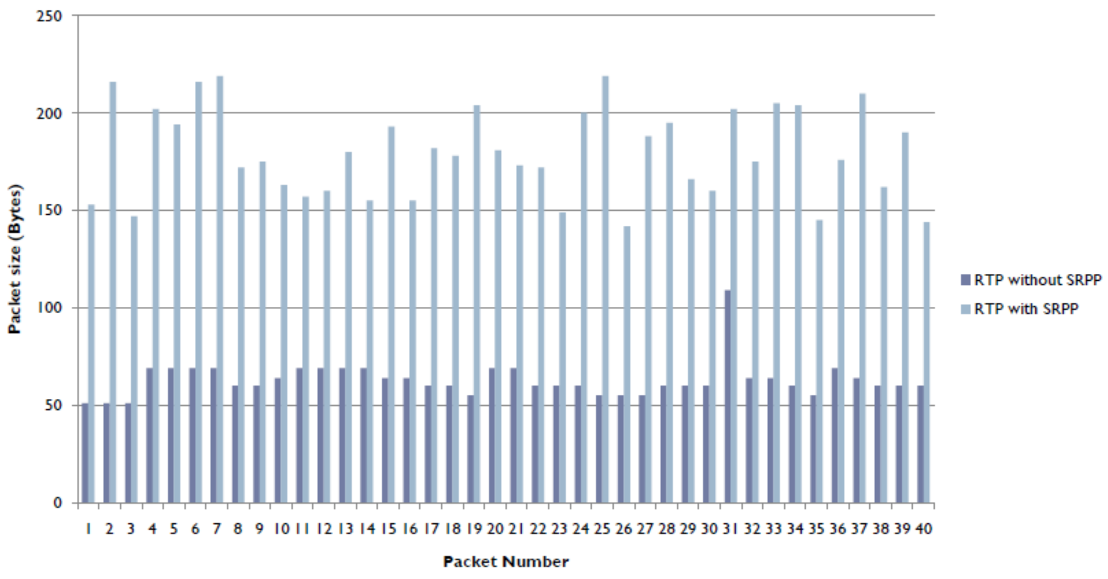


Fig. 17. Effectiveness of SRPP for RTP Packets

A similar experiment was done for a stream of SRTP packets. The following figure shows that while the input packet sizes range from 50 to 80, they are effectively padded to packets of size ranging from 150 to 250. Fig. 18 shows the results for SRTP packet padding using SRPP.

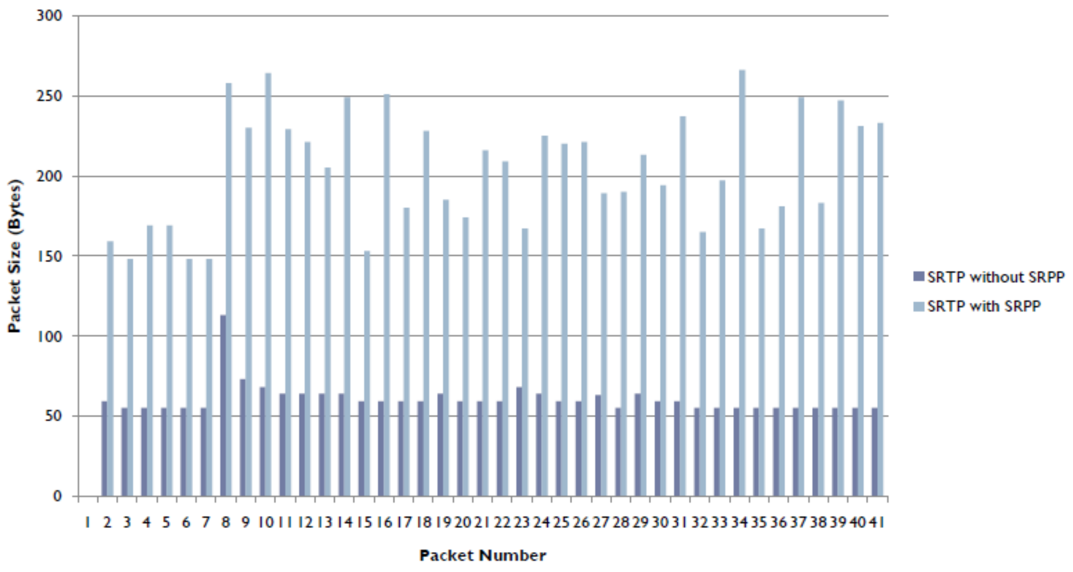


Fig. 18. Effectiveness of SRPP for SRTP Packets

3. Efficiency

1. *Jitter*

Formally, jitter is defined as the statistical variance of the RTP packet inter-arrival timings. In order to quantify the efficiency of SQRKal, we ran SQRKal a number of times for each padding technique and collected traces for each case. Then, the jitter of the session was calculated using the formula,

$J_i = J_{i-1} + (|(R_i - R_{i-1}) - (S_i - S_{i-1})| - J_{i-1})/16,$

where R_i represents the received time of packet i, S_i represents the RTP timestamp of the packet signifying the sent time of the packet, J_i represents the instantaneous jitter for packet i.

If we take the jitter value of the last packet, we get the approximate mean jitter of the session. This is the standard method of calculation of jitter for RTP packets. We applied this formula to determine the jitter for every session corresponding to each technique. The following graph depicts the average jitter values for all the padding techniques.

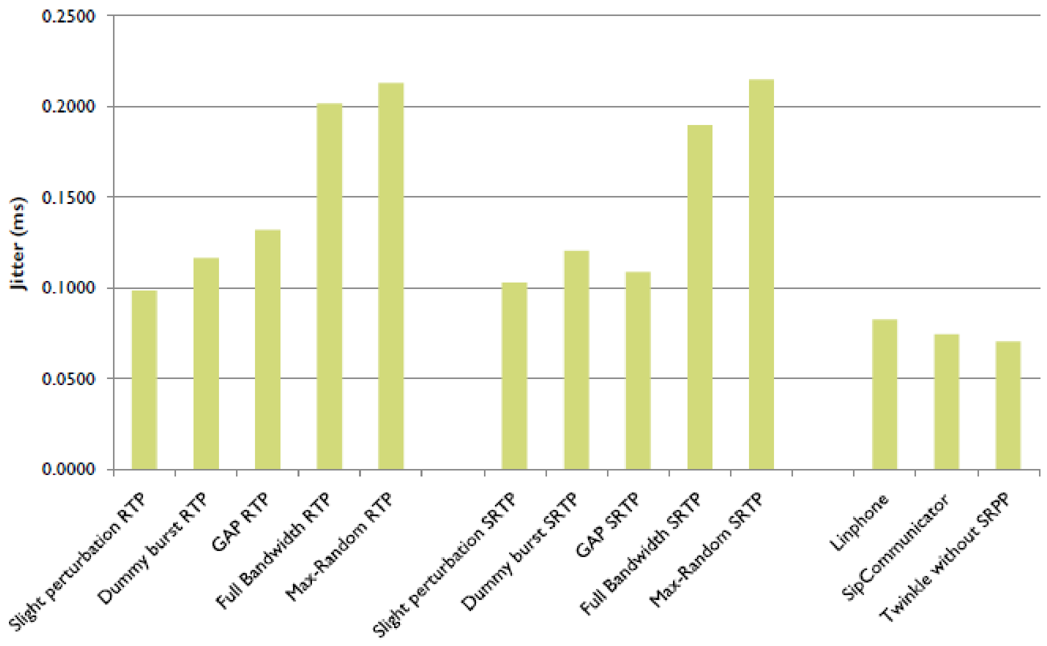


Fig. 19. Mean Jitter for Different Padding Techniques

It was seen that the regular VoIP sessions using linphone [22], sip communicator [23] or twinkle [21] exhibit a jitter value between 0.06ms-0.08ms. For cases of both RTP and SRTP, it was observed that the jitter values are well within the permissible value of 0.5ms. It was also noted that max-random mode and full-bandwidth mode add more jitter to the stream, while GAP technique, burst padding and slight perturbation mode have jitter ranging between 0.1ms - 0.13ms. This satisfactory jitter suggests that SQRKal is efficient in carrying VoIP calls without addition of extra jitter.

2. *Average Increase in size per packet*

   Increase in bandwidth due to padding can be roughly considered as the average increase in packet size per RTP/SRTP packet. For every trace, we calculated the average increase in packet size per packet, and then took the mean of these values for each padding technique. Fig. 20 depicts the distribution of the increase in bandwidth.

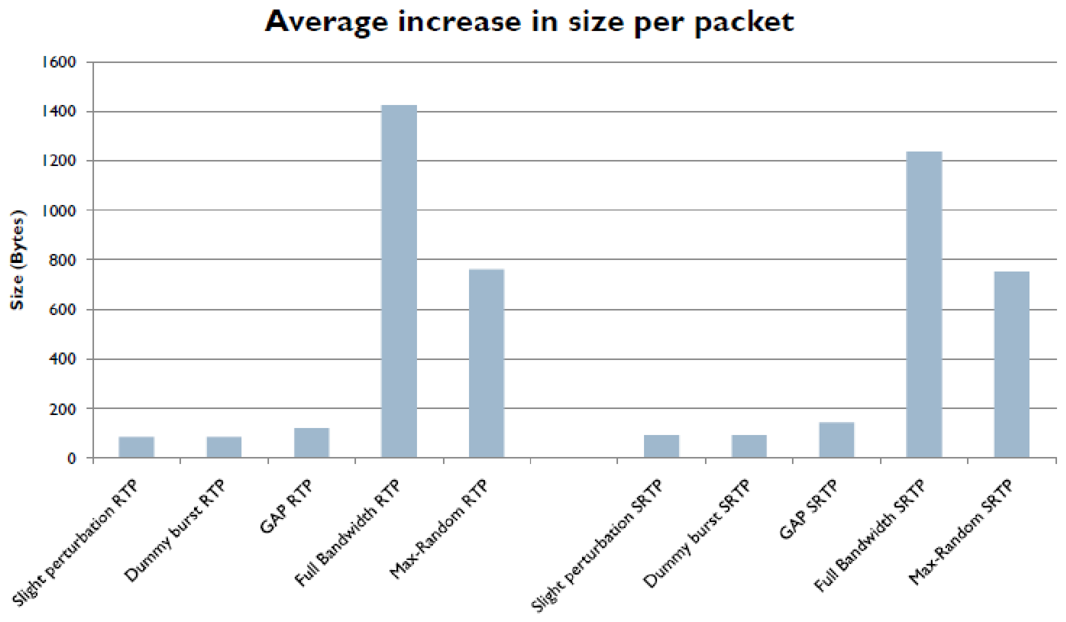


Fig. 20. Average Increase in Packet Size per Packet

As expected, we have max-random and full bandwidth modes of operation adding a huge amount of dummy data to the packets, thereby utilizing a high bandwidth. On the contrary, since slight perturbation, dummy burst and GAP add less amount of padding bytes, we see less increase of bandwidth for these algorithms. This holds true for both RTP and SRTP.

3. *CPU and Network Load*

Experiments were run on a single-core CPU with Twinkle phone as the only running application. In the first case, we ran Twinkle with RTP option without running SQRKal and found that the CPU load was in the range of 35%-40%. This can be considered as a coarse-grained CPU utilization measure. Next, we ran Twinkle and SQRKal, and noticed that while SQRKal was effectively padding the media stream, the CPU load was marginally increased to 38% to 40%. In case of SRTP, a similar observation was made. Fig. 21 and Fig. 22 depict the CPU load for RTP and SRTP packet padding respectively.

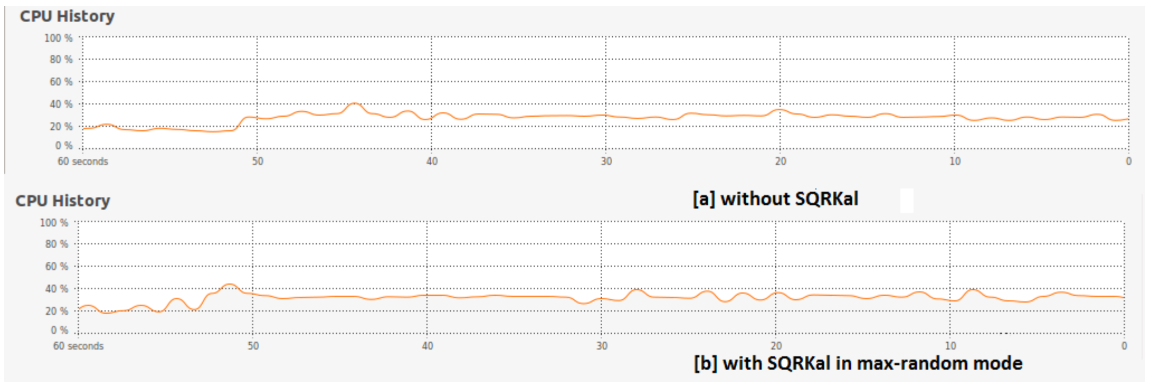


Fig. 21. CPU Load for RTP, with and without SQRKal

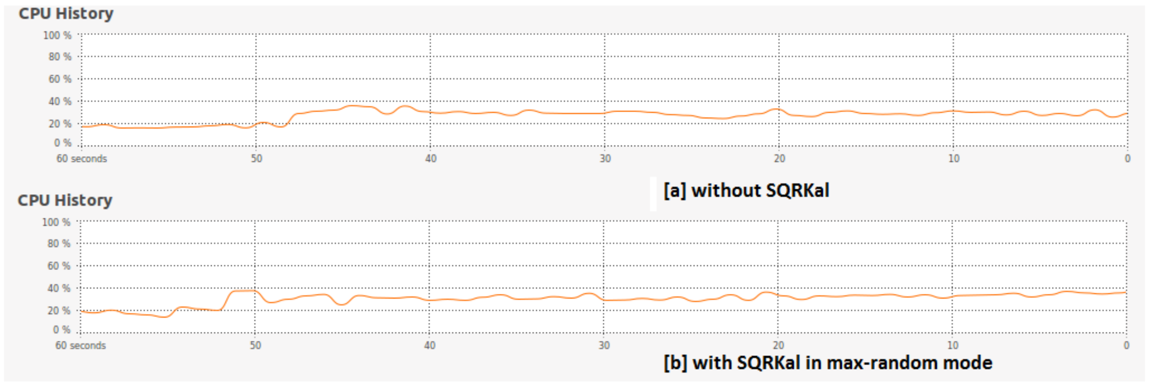


Fig. 22. CPU Load for SRTP, with and without SQRKal

As expected, SQRKal adds padding bytes to a packet, thereby increasing the bandwidth usage. Fig. 23 compares the bandwidth usage for RTP when SQRKal is not used, with the bandwidth usage when SQRKal is used with max-random mode. It is seen that in the worst case, bandwidth usage increases four-fold. Fig. 24 shows a similar comparison for SRTP packet padding.

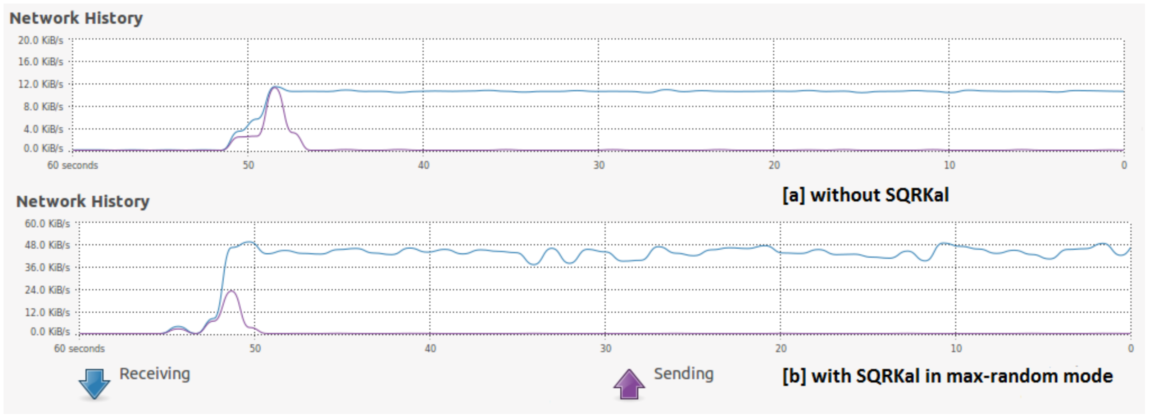


Fig. 23. Network Load for RTP, with and without SQRKal

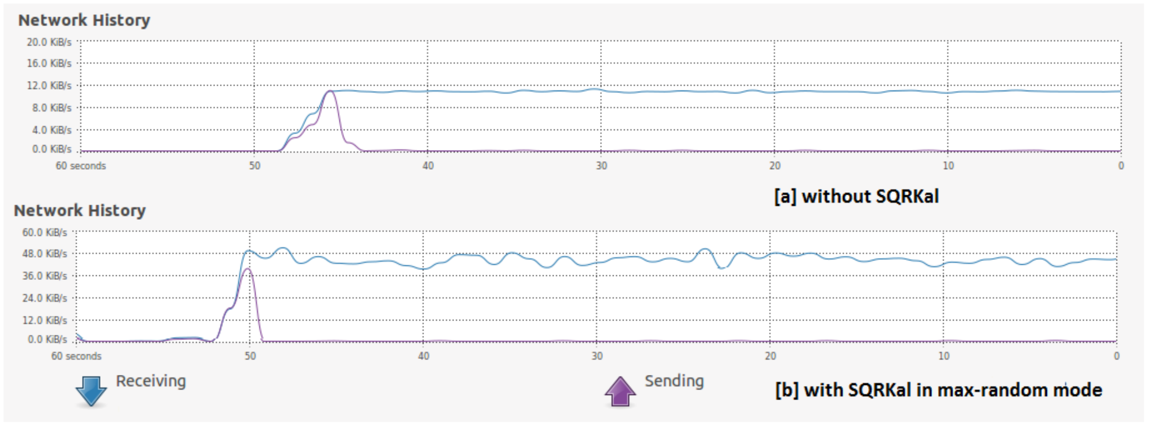


Fig. 24. Network Load for SRTP, with and without SQRKal

4. *Mean-Opinion Score*

Mean opinion score (MOS) provides a numerical indication of the perceived quality of received media after transmission. The Mean-Opinion Score (MOS) was calculated by asking listeners to rate the quality of a recorded audio for each padding technique, on a scale of 1 to 5, 5 being the highest. MOS is the average rating for each case and the results as shown in Table 1. Algorithms like GAP, Burst Padding and Slight Perturbation have a very good MOS, suggesting that the perceptive quality of the audio is not getting degraded.

Table I. Mean Opinion Score

System	MOS
Without SQRKal	3.33
With Full Bandwidth	3.00
With Max-Random	2.00
With Gradient Ascent Padding	3.33
With Burst Padding	3.66
With Slight Perturbation	4.00

5. *Mutual information*

We also measure certain information theoretic measures for each of the padding technique to determine the best way of padding among them. Mutual information is determined using a MATLAB toolbox called Information Theory Toolkit v1.0 [18]. Fig. 25 represents the average value of mutual information for all the traces in each padding technique.

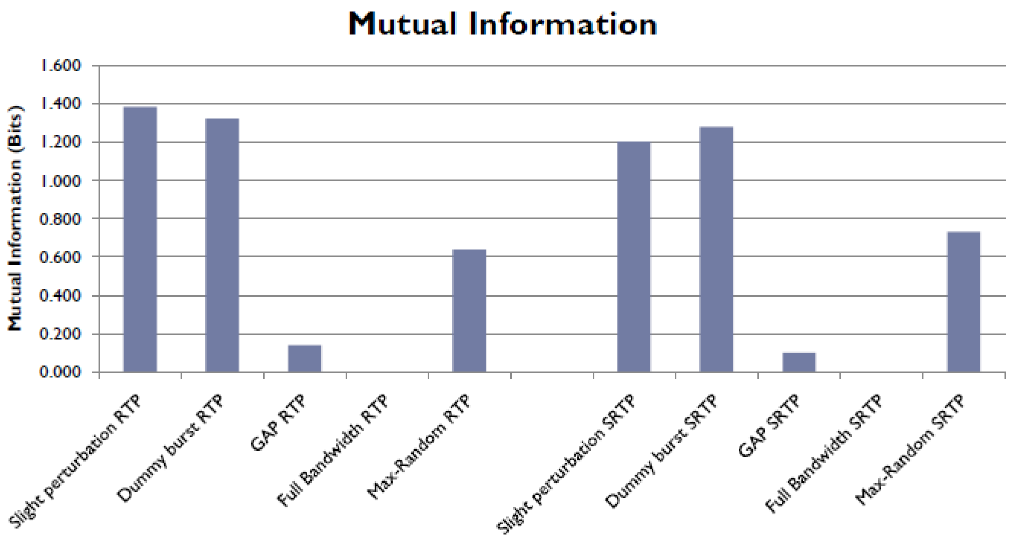


Fig. 25. Mutual Information for RTP and SRTP with SQRKal

It can be observed that mutual information is 0 for streams padded with full bandwidth. Since the output packet size is constant, it can be intuitively deduced that inferring the input packet size from a constant output packet size is not feasible. Likewise, it can be seen that mutual information is very less for GAP technique, but is pretty high for slight-perturbation. It can be argued that since we are adding small amount of stochasticity in the case of slight perturbation, there is high mutual dependance between the output packet size and the input packet size.

6. *Relative Entropy*

   Relative entropy between the input and output packet size distributions was determined for each trace in each padding algorithm. Since the relative entropy values varied from each other, we plot the cumulative distribution function of Relative Entropy for each padding algorithm in fig. 26.

   It can be observed that there are values of high relative entropy when we consider max-random padding algorithm. The other algorithms are not appreciably different from each other. It can be safely assumed the efficiency with respect to Relative Entropy is not a great measure to distinguish between the given padding algorithms.

All the results clearly favor the usage of the Gradual Ascent Padding algorithm, because it adds less jitter to the packet stream, have considerably less mutual information between the output packet size and input packet size, have a satisfactory increase in bandwidth usage and a satisfactory quality score.

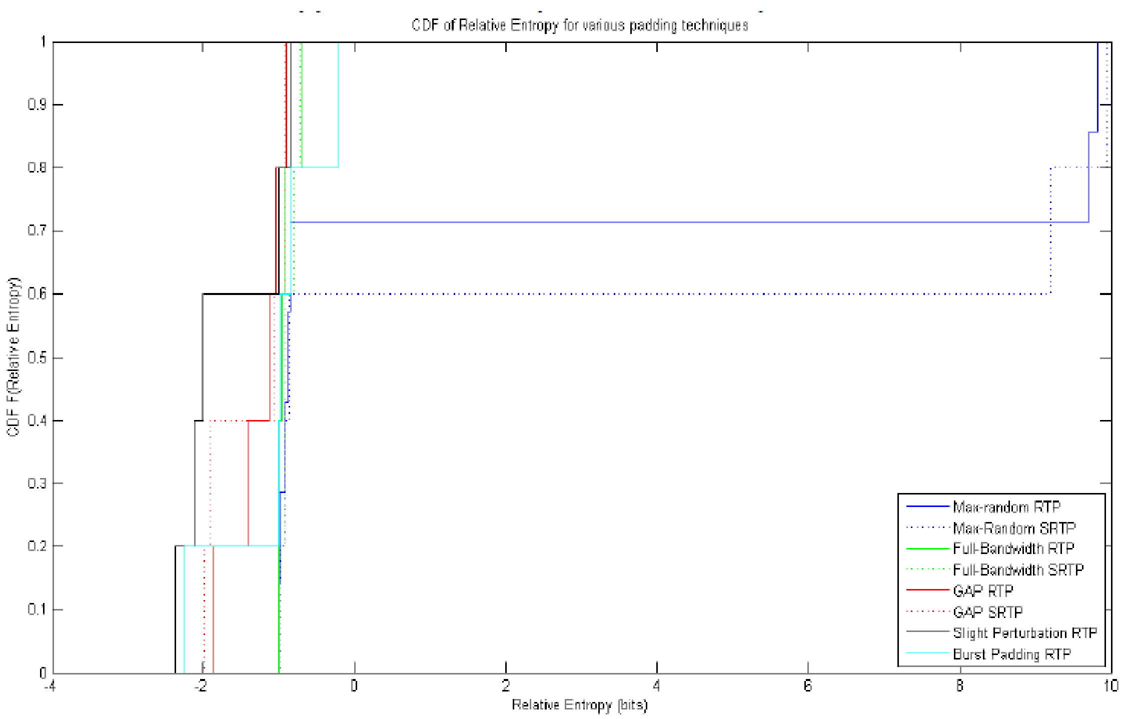


Fig. 26. C.D.F. of Relative Entropy for RTP and SRTP, with SQRKal

B. Determination of Output Packet Size Distribution

Statistical traffic analysis attacks attempt to extract information about the input packet sizes from the output packet size distribution. Our motivation is to ensure maximum divergence between the output packet size distribution and the input packet size distribution, such that traffic analysis provides minimum information to the attacker. We consider two probability mass distribution functions to be different from each other if there exists minimum mutual information or maximum relative entropy among them.

Let us say that an input stream of VoIP packets is morphed into an output stream, such that the size of the output packet OP_i is greater than the input packet size IP_i. There are n distinct packet sizes, $s_1, s_2, s_3, ...s_n$, for both input and output streams. The input packets have a packet size distribution p(S) and the output packets have a packet size distribution q(S), where S represents the set of distinct packet sizes.

We can ensure that it is unfeasible to infer p(S) from observations of q(S) using the following methods:

- For every input packet size IP_i, an output packet size $OP_i \quad \epsilon \quad [IP_i, MPS]$ is generated using a pseudo-random number generator, where MPS is the maximum allowed packet size or maximum transmission unit of the datagram. Most pseudo-random number generators provide uniformly distributed output, and intuitively we can say that a uniform output distribution makes it harder to infer the input distribution from it.

- For every input packet size IP_i, the output packet size OP_i is calculated using a one-way function such as a cryptographic hash function. The problem of determining IP_i from OP_i becomes "unsolvable" in polynomial time.

- For every input packet size IP_i, we calculate the optimum value for output packet size OP_i, such that it maximizes the relative entropy between the input and output distributions.

C. Optimization of Relative Entropy

For the input packet size distribution p(S) and output packet size distribution q(S), we have the relative entropy or K-L Divergence as:

$$D(p||q) = \sum_{S=1}^{n} p(S) \log \frac{p(S)}{q(S)} \tag{6.1}$$

When we receive an input packet IP_i, we need to find the output packet size OP_i

where $D(q||p)$ is maximum. This translates to the following optimization problem.

Maximize,

$$D(p||q) = \sum_{S=1}^{n} p(S) \log \frac{p(S)}{q(S)} \tag{6.2}$$

An output distribution consisting of all output packets padded to the maximum packet size MPS will result in the maximum relative entropy at the cost of high bandwidth usage. To ensure that padding does not exceptionally increase the bandwidth usage, we must also minimize a distance measure between the output packet size and input packet size for all packets.

In addition, the following constraints must also hold, since p(S) and q(S) must be probability distributions:

$$\sum_{S=1}^{n} p(S) = 1 \tag{6.3}$$

$$\sum_{S=1}^{n} q(S) = 1 \tag{6.4}$$

$$\tag{6.5}$$

Let us approach this optimization problem as an inductive process. We can assume that initially we start with the output distribution as uniform, and as we receive a subsequent input packet, we can calculate the optimal output packet size, pad the output packet to the output size and send it forward.

When we receive input packet IP_i, the input packet size distribution p(S) is already known, since we have all the input packet sizes from IP_0 to IP_i. So, p(S) can be considered to be a constant. The output packet sizes till i-1 i.e. OP_0 to OP_{i-1} are also known, and the output packet size distribution q(S) will be dependent on OP_i. Hence, the known values are:

1. p(S) $\forall S$

2. $IP_j\ \forall j \epsilon (0, i)$

3. $OP_j\ \forall j \epsilon (0, i-1)$

Equation 6.2 simplifies to,

$$D(p||q) = \sum_{S=1}^{n} p(S) \log p(S) - \sum_{S=1}^{n} p(S) \log q(S)$$

Since the first term $\sum_{S=1}^{n} p(S) \log p(S)$ is a constant, we must minimize the second term $\sum_{S=1}^{n} p(S) \log q(S)$ in order to maximize $D(p||q)$.
So the problem can be simplified to,

$$\begin{aligned} \textit{Minimize} \quad & \sum_{S=1}^{n} p(S) \log q(S) \text{ and} \\ & \text{increase in bandwidth usage.} \end{aligned}$$

1. Minimize the Difference between Output Packet Size and Input Packet Size

If we specify the increase in bandwidth usage as the difference between output packet size and input packet size for every packet, we must minimize,

$$\sum_{j=0}^{i} (OP_j - IP_j)$$

The optimization problem now becomes,

Minimize,

$$\sum_{S=1}^{n} p(S) \log q(S) + \mu(\sum_{j=0}^{i} (OP_j - IP_j)),\ \textit{where } \mu \textit{ is a weight constant} \tag{6.6}$$

The only constraint is:

$$\sum_{S=1}^{n} q(S) = 0 \tag{6.7}$$

Thus, after applying Lagrange Multiplier λ, the objective function can be stated as:

$$\Lambda(q(S), OP_j, \lambda) = \sum_{S=1}^{n} p(S) \log q(S) + \mu(\sum_{j=0}^{i} (OP_j - IP_j)) + \lambda(\sum_{S=1}^{n} q(S) - 1) \tag{6.8}$$

Since IP_j is constant for all values of j and $\sum_{j=0}^{i} OP_j$ can be expressed as $\sum_{S=1}^{n}(Sq(S))$, eq. 6.8 simplifies to:

$$\Lambda(q(S),\lambda) = \sum_{S=1}^{n} p(S)\log q(S) + \mu\sum_{S=1}^{n}(Sq(S)) - k + \lambda(\sum_{S=1}^{n} q(S) - 1),\ \textit{where}\ k = \textstyle\sum_{j=0}^{i} IP_j \tag{6.9}$$

The following equations need to be solved:

$$\frac{\partial\Lambda}{\partial q(S)} = 0 \tag{6.10}$$

$$\frac{\partial\Lambda}{\partial\lambda} = 0 \tag{6.11}$$

Solving eq. 6.9 for all values of S, we have,

$$\frac{\partial\Lambda}{\partial q(s_1)} = 0 \tag{6.12}$$

$$\frac{\partial\Lambda}{\partial q(s_2)} = 0 \tag{6.13}$$

$$\frac{\partial\Lambda}{\partial q(s_3)} = 0 \tag{6.14}$$

$$\vdots\vdots$$

$$\frac{\partial\Lambda}{\partial q(s_n)} = 0 \tag{6.15}$$

The above equations can be expanded as,

$$p(s_1) + \mu s_1 q(s_1) + \lambda q(s_1) = 0 \tag{6.16}$$

$$p(s_2) + \mu s_2 q(s_2) + \lambda q(s_2) = 0 \tag{6.17}$$

$$p(s_3) + \mu s_3 q(s_3) + \lambda q(s_3) = 0 \tag{6.18}$$

$$\vdots\vdots$$

$$p(s_n) + \mu s_n q(s_n) + \lambda q(s_n) = 0 \tag{6.19}$$

Each of the equations 6.16-6.19 can be simplified as:

$$q(s_i) = -\frac{p(s_i)}{\mu s_i + \lambda} \quad \forall i \epsilon [1, n] \tag{6.20}$$

Since $\sum_{S=1}^{n} q(S) = 1$ (from eq. 6.7 and by solving eq. 6.11), the above set of equations simplifies to,

$$\sum_{i=1}^{n} \frac{p(s_i)}{\mu s_i + \lambda} = 0 \tag{6.21}$$

The value of λ can be determined by solving the polynomial equation eq. 6.21. Abel-Ruffini Theorem states that the solution of a polynomial equation of degree equal to or greater than 5 cannot be expressed algebraically i.e. in radicals. But it can be numerically determined using Groebner's basis or other methods. The output size distribution with optimal relative entropy will then be determined using eq. 6.20.

2. Minimize the Square Distance between the Average Size of Output and Input

If we specify the increase in bandwidth usage as the square distance between the average output size and average input size, we must minimize,

$$(\sum_{j=0}^{i} OP_j - \sum_{j=0}^{i} IP_j)^2$$

The optimization problem now becomes,

Minimize,

$$\sum_{S=1}^{n} p(S) \log q(S) + \mu(\sum_{j=0}^{i} OP_j - \sum_{j=0}^{i} IP_j)^2, \textit{ where } \mu \textit{ is a constant} \tag{6.22}$$

The only constraint here is:

$$\sum_{S=1}^{n} q(S) = 0 \tag{6.23}$$

Thus, after applying Lagrange Multiplier λ, the objective function can be stated as:

$$\Lambda(q(S), OP_j, \lambda) = \sum_{S=1}^{n} p(S) \log q(S) + \mu(\sum_{j=0}^{i} OP_j - \sum_{j=0}^{i} IP_j)^2 + \lambda(\sum_{S=1}^{n} q(S) - 1) \tag{6.24}$$

Since IP_j is constant for all values of j and $\sum_{j=0}^{i} OP_j$ can be expressed as $\sum_{S=1}^{n}(Sq(S))$, eq. 6.24 simplifies to:

$$\Lambda(q(S),\lambda) = \sum_{S=1}^{n} p(S)\log q(S) + \mu(\sum_{S=1}^{n}(Sq(S)) - k)^2 + \lambda(\sum_{S=1}^{n} q(S) - 1),\ where\ k = \sum IP_j \tag{6.25}$$

The following equations are required to be solved:

$$\frac{\partial \Lambda}{\partial q(S)} = 0 \tag{6.26}$$

$$\frac{\partial \Lambda}{\partial \lambda} = 0 \tag{6.27}$$

Solving eq. 6.26 for all values of S, we have,

$$\frac{\partial \Lambda}{\partial q(s_1)} = 0 \tag{6.28}$$

$$\frac{\partial \Lambda}{\partial q(s_2)} = 0 \tag{6.29}$$

$$\frac{\partial \Lambda}{\partial q(s_3)} = 0 \tag{6.30}$$

$$\vdots\vdots$$

$$\frac{\partial \Lambda}{\partial q(s_n)} = 0 \tag{6.31}$$

The above equations can be expanded as,

$$p(s_1) + 2\mu s_1 q(s_1)(\sum_{S=1}^{n} Sq(S) - k) + \lambda q(s_1) = 0 \tag{6.32}$$

$$p(s_2) + 2\mu s_2 q(s_2)(\sum_{S=1}^{n} Sq(S) - k) + \lambda q(s_2) = 0 \tag{6.33}$$

$$p(s_3) + 2\mu s_3 q(s_3)(\sum_{S=1}^{n} Sq(S) - k) + \lambda q(s_3) = 0 \tag{6.34}$$

$$\vdots\vdots$$

$$p(s_n) + 2\mu s_n q(s_n)(\sum_{S=1}^{n} Sq(S) - k) + \lambda q(s_n) = 0 \tag{6.35}$$

Taking $X = \sum_{S=1}^{n} Sq(S) - k$, the equations 6.32-6.35 simplify to:

$$p(s_1) + 2\mu s_1 X q(s_1) + \lambda q(s_1) = 0 \tag{6.36}$$

$$p(s_2) + 2\mu s_2 X q(s_2) + \lambda q(s_2) = 0 \tag{6.37}$$

$$p(s_3) + 2\mu s_3 X q(s_3) + \lambda q(s_3) = 0 \tag{6.38}$$

$$\vdots\vdots$$

$$p(s_n) + 2\mu s_n X q(s_n) + \lambda q(s_n) = 0 \tag{6.39}$$

Each of the equations 6.36-6.39 can be simplified as:

$$q(s_i) = -\frac{p(s_i)}{2\mu s_i X + \lambda} \quad \forall i \epsilon [1, n] \tag{6.40}$$

Adding the equations 6.36-6.39, we get,

$$\sum_{S=1}^{n} p(S) + 2X(s_1 q(s_1) + s_2 q(s_2) + ... + s_n q(s_n)) + \lambda(\sum_{S=1}^{n}(q(S)) = 0 \tag{6.41}$$

This equation simplifies to $X = \sqrt{-\frac{\lambda+1}{2}}$.

Hence, substituting the value of X in eq. 6.40, we get,

$$q(s_i) = -\frac{p(s_i)}{\mu s_i \sqrt{-2(\lambda+1)} + \lambda} \quad \forall i \epsilon [1, n] \tag{6.42}$$

Since $\sum_{S=1}^{n} q(S) = 1$ (from eq. 6.7 and by solving eq. 6.27), the above set of equations simplifies to,

$$\sum_{s_i \epsilon S} \frac{p(s_i)}{\mu s_i \sqrt{-2(\lambda+1)} + \lambda} + 1 = 0 \tag{6.43}$$

The value of λ can be determined by solving the polynomial equation eq. 6.43. This cannot be solved algebraically but can be numerically determined using Groebner's basis or other methods. The output size distribution with optimal relative entropy can then be determined by eq. 6.42.

3. Specifying a Bandwidth Threshold

If we specify the bandwidth constraint with a specific bandwidth threshold B, then the constraint as shown in eq. 6.44 will be also be considered.

$$\frac{\sum_{j=1}^{i} OP_j}{T} < B, \textit{ where } B \textit{ is the bandwidth threshold and } T \textit{ is the total time.} \quad (6.44)$$

This optimization problem can be solved using Karush-Kuhn-Tucker or KKT conditions and is beyond the scope of this study.

4. Results Using Matlab Optimization Toolbox

Since all the above cases fall under the domain of non-linear minimization problem, we used the MATLAB Optimization Toolbox to find the optimal output packet size distribution. After specifying the respective objective function and the constraints, we use the *fmincon* function to use the interior-point algorithm for non-linear minimization. This process is repeated for random start values and different types of discrete distributions for the input packet size distribution p(S) and output packet size distribution q(S).

Probability distribution fitting was applied on the generated output distributions using the *dfittool* in MATLAB. 73% (24 from 30) of the optimal output distributions were found to be uniform in nature, 33% were negative binomial distributions with parameter r = 1 and the remaining belonged to the class of hypergeometric distributions. The optimal output packet size distribution is most likely to be uniform in nature. However, since MATLAB is unable to process input arrays of size greater than 70, this result is not entirely reliable, and this approach is not scalable in real-time.

CHAPTER VII

FUTURE WORK AND CONCLUSION

A. Future Work

SRPP as an application-level padding protocol can be applied, leveraged or extended in the following few scenarios:

- Implementation of SRPP on an appliance: This will allow VoIP calls made from IP Phones and other hardware to be padded.
- Implementing new and existing padding algorithms in libSRPP, like morphing algorithm.
- SQRKal implementation for Windows.
- VoIP Steganography using SRPP padding: SRPP Padding provides an effective platform to hide data and send information within the padding bytes.
- Study of approximation algorithms or further heuristic measures to reduce mutual information between input and output packet size distribution.
- Threat analysis study on utilizing SRPP timers to reveal identity of dummy packets.

The above salient points can serve as proper avenues of carrying this work forward.

B. Conclusion

In this thesis, we have designed and implemented SRPP, a new RTP profile for securing Voice-over-IP traffic from traffic analysis attacks. It pads every RTP packet to a predefined packet size, adds dummy packets at the end of every burst in a controllable way, adds dummy bursts to hide silence spurts, and hides information about the packet inter-arrival timings. SRPP has been implemented as a library libSRPP for use by VoIP application designers and as a standalone application named SQRKal for use by regular users. A few practical approaches to padding were implemented, and a mathematical approach to optimal padding was discussed. Implementation of SQRKal and libSRPP was found to be effective and efficient, since SRPP adds minimum jitter within the permissible amount of 0.5 ms and does not increase the operational load on the CPU. It does increase the network bandwidth usage, but implementation of a proper padding algorithm like Gradual Ascent Padding algorithm minimizes that overhead as well. It is hoped that with a more robust and portable implementation of SQRKal, SRPP will be readily accepted by the VoIP community.

REFERENCES

[1] C.V. Wright, L. Ballard, S.E. Coull, F. Monrose, and G.M. Masson, “Spot me if you can: Uncovering spoken phrases in encrypted VoIP conversations,” in *IEEE Symposium on Security and Privacy*, pp. 35–39, May 2008.

[2] C.V. Wright, L. Ballard, F. Monrose, and G.M. Masson, “Language identification of encrypted VoIP traffic: Alejandra y Roberto or Alice and Bob?,” in *16th Annual USENIX Security Symposium*, Boston, MA, USA, August 2007, pp. 1–12.

[3] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, “RTP: A transport protocol for real-time applications,” Retrieved March 2009 from http://www.ietf.org/rfc/rfc3550.txt.

[4] M. Baugher, D. McGrew, M. Naslund, E. Carrara, and K. Norrman, “The secure real-time transport protocol (SRTP),” Retrieved March 2009 from http://www.ietf.org/rfc/rfc3711.txt.

[5] P. Zimmermann, A. Johnston, and J. Callas, “Zrtp: Media path key agreement for secure RTP,” Retrieved March 2009 from http://www.ietf.org/ietf/1id-abstracts.txt.

[6] J. Rosenberg and H. Schulzrinne, “SIP: Session initiation protocol,” Retrieved March 2009 from http://www.ietf.org/rfc/rfc3261.txt.

[7] “The Zfone Project,” Retrieved March 2009 from http://zfoneproject.com/.

[8] C. Wright, S. Coull and F. Monrose, “Traffic morphing: An efficient defense against statistical traffic analysis,” in *Proc. of the 14th Annual Network and Distributed Systems Symposium*, San Diego, CA, USA, February 2009.

[9] Y. Guan, X. Fu, R. Bettati, and W. Zhao, "NetCamo: Camouflaging network traffic for QoS guaranteed mission critical applications.," *IEEE Transactions on Systems, Man and Cybernetics. Special Issue on Information Assurance*, vol. 31, no. 4, pp. 253–265, July 2001.

[10] G. Danezis, "The traffic analysis of continuous-time mixes," *Privacy Enhancing Technologies*, pp. 35–50, 2004.

[11] W. Wang, M. Motani and V. Srinivasan, "Dependent link padding algorithms for low latency anonymity systems," in *Proc. of 15th ACM Conference on Computer and Communications Security*, Alexandria, VA, USA, October 2008, pp. 323-332.

[12] T. Lella, R. Bettati, "Privacy of encrypted voice-over-IP," in *Proc. of the 2007 IEEE International Conference on Systems, Man and Cybernetics*, Montreal, Canada, October 2007, pp. 3063–3068.

[13] M. Valin, "The Speex codec manual," Retrieved May 2010 from http://www.speex.org/docs/manual/speex-manual.pdf.

[14] Y. Zhu, X. Fu, B. Graham, R. Bettati, and W. Zhao, "On flow correlation attacks and countermeasures in mix networks," in *Proc. of Workshop on Privacy Enhancing Technologies*, Toronto, Canada, May 2004, pp. 207–225.

[15] D. Chaum, "Untraceable electronic mail, return addresses, and digital pseudonyms," *Communications of the ACM*, vol. 24, no. 2, February 1981, pp. 84–88.

[16] B. W. Graham, Y. Zhu, X. Fu, and R. Bettati, "Using covert channels to evaluate the effectiveness of flow confidentiality measures," in *Proc. of the 11th IEEE*

International Conference on Parallel and Distributed Systems, Washington, DC, USA, June 2005, pp. 57–63.

[17] Y. Zhu, X. Fu, and R. Bettati, “On the effectiveness of continuous-time mixes under flow-correlation based anonymity attacks,” in *Proc. of the Fourth IEEE International Symposium on Network Computing and Applications*, Washington, DC, USA, July 2005, pp. 215–218.

[18] J. Goi, I. Martincorena, “Information Theory Toolbox v1.0.,” Retrieved March 2010 from http://www.mathworks.com/matlabcentral/fileexchange/17993.

[19] JRTPLIB, “RTP Library in C++,” Retrieved August 2010 from http://research.edm.uhasselt.be/ jori/page/index.php?n=CS.Jrtplib

[20] “Wireshark frequently asked questions,” Available: http://www.wireshark.org/, 2010.

[21] “Twinkle,” Retrieved August 2010 from http://www.xs4all.nl/ mfnboer/twinkle/

[22] S. Morlat, “Linphone, an open-source SIP video phone for Linux and Windows,” Retrieved August 2010 from http://www.linphone.org/.

[23] “Sip-Communicator,” Retrieved August 2010 from http://sip-communicator.org/

[24] “Netfilter IPTables and ip_queue,” Retrieved August 2010 from http://netfilter.org/documentation/HOWTO//packet-filtering-HOWTO.html

[25] A. S. Mihai, “Voice over IP security a layered approach,” Available: http://www.xmcopartners.com

APPENDIX A

LIBSRPP API

VoIP application developers can use libSRPP to incorporate SRPP in their applications. The major functions of the libSRPP API are present in the header file SRPP_functions.h and can be listed as:

```
/***--------   Processing functions:  --------------***/
// To initialize SRPP
 int init_SRPP();

// To verify if SRPP has been disabled or not. Returns 0, if disabled
 int SRPP_Enabled();

// To create a SRPP session
 SRPPSession* create_session(string address, int port,
  CryptoProfile crypto);

// To start the srpp session
 int start_session();
 int start_session(sdp_srpp sdp);
// To stop the srpp session
 int stop_session();
 void stop_abnormally(int i);
```

```
// To start the SRPP Signaling process
 int signaling();

// To convert a RTP packet to SRPP packet
 SRPPMessage rtp_to_srpp(RTPMessage* rtp_msg);
 SRPPMessage rtp_to_srpp(RTP_Header rtp_hdr, char* buf, int length);

// To convert a SRPP packet back to RTP packet
 RTPMessage srpp_to_rtp(SRPPMessage* srpp_msg);

// To convert a SRTP packet to SRPP packet
 SRPPMessage srtp_to_srpp(SRTPMessage* srtp_msg);

// To convert a SRPP packet back to SRTP packet
 SRTPMessage srpp_to_srtp(SRPPMessage* srpp_msg);
 int srpp_to_srtp(SRPPMessage * srpp_msg, char * buff,int length);

// To create a SRPP Message with the data and encrypt it.
 SRPPMessage create_and_encrypt_srpp(string data);
// To create a SRPP Message and return it.
 SRPPMessage create_srpp_message(string data);

// Only create a RTP Message and return it.
 RTPMessage create_rtp_message(string data);
// Only create a SRTP Message and return it.
 SRTPMessage create_srtp_message(string data);
```

```
// To encrypt the given SRPP packet
 SRPPMessage encrypt_srpp(SRPPMessage * original_pkt);
// To decrypt the given SRPP packet
 SRPPMessage decrypt_srpp(SRPPMessage * encrypted_pkt);
// To get the padding functions object used here
 PaddingFunctions* get_padding_functions();
// To get the current Session object
 SRPPSession * get_session();

/***------------   Utility functions:  --------------***/

// SRPP Pseudo-Random number between min and max
 int srpp_rand(int min,int max);

// Used by the interior functions to send or receive a SRPP message
 int send_message(SRPPMessage* msg);
 SRPPMessage receive_message();
 SRPPMessage processReceivedData(char * buff, int bytes_read);

// To set the Process Function (callback functions) which
// handle sending and receiving of a message.
 int setSendFunctor(int (*process_func)(char*,int));
 int setReceiveFunctor(SRPPMessage (*process_func)());

// To parse the received message. Returns -1 if its a media packet
// and 1 if its a signaling packet.
```

```
 int isSignalingMessage (SRPPMessage * message);
 int isSignalingMessage (char * buff);

// To check or set whether the signaling is complete
 int isSignalingComplete();
 int setSignalingComplete();

// To check whether media session is complete
 int isMediaSessionComplete();
 int set_starting_sequenceno(int seq_no);

/***--------  END OF FILE  --------------***/
```

APPENDIX B

MONTE CARLO SIMULATIONS FOR MINIMIZATION OF MUTUAL INFORMATION

Monte Carlo methods (or Monte Carlo experiments) are a class of computational algorithms that rely on repeated random sampling to compute their results. First, we define a domain of inputs. Next, we generate inputs randomly , and perform a computation on each input. Finally, we aggregate all the results into our final result. Our objective is to pad the original packets of size x to a new size y, such that the mutual information (MI) [18] between distributions of y and x is minimum.

Observations based on minimum MI for all combinations of input-output distributions: We ran a series of Monte-Carlo experiments, with the aim to find out the combination of probability distributions of x and y, for which we can get minimum MI. The skeleton of this algorithm is as follows:

1. For a random number of experiments 'r', repeat steps 2-4.
2. Generate a random maximum packet size 'm' and a random number 'n'.
3. Generate random samples of length n from uniform, binomial, poisson and geometric distributions for the discrete case and uniform, normal, exponential, beta, gamma and chi-square distributions for continuous case. The parameters to all these distributions are appropriately tuned, on the basis of 'm'. For e.x. I generate uniform numbers in the range [1, m], and exponential numbers with mean λ, such that λ is a random number in the range [1, m].

4. Calculate the mutual information between each of these samples, and find the combination of input-output distributions which has the minimum MI in this experiment.

5. After all the 'r' experiments are complete, plot the combination of distributions vs the number of experiments in which they had the minimum MI among all the combinations.

The results, as shown in Fig. 27-28, were straightforward in the case of discrete distributions. The minimum MI was found to be when both input and output distributions were geometric in nature.

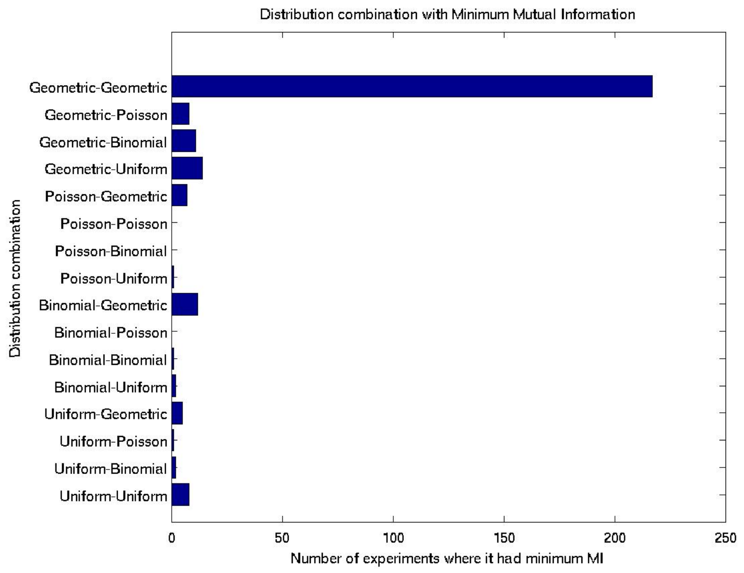


Fig. 27. Observations for Discrete Distributions (Run 1)

For continuous distributions, it was seen that beta or gamma distributions reduce the MI when they are considered as the output distribution.

Observations based on individual input distributions: Next, we observed the optimal distribution for specific cases of input distributions. It was observed that if the output

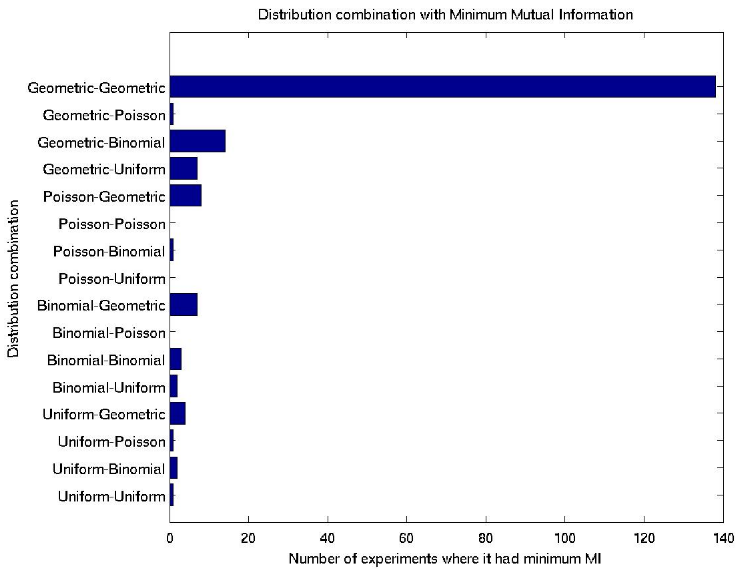


Fig. 28. Observations for Discrete Distributions (Run 2)

is geometrically distributed, it had the minimum MI compared to the other output distributions. For example, if the input is uniformly distributed, 165 experiments where the output distribution was geometrical had the minimum MI, and this was higher than all such other combinations. So, it is a good idea to simply transform any discrete input distribution to geometric distribution, in order to minimize the mutual information, based on these Monte Carlo experiments. Fig. 29-32 show the results for various discrete input distributions.

In the case of continuous distributions, it was observed that for most of the runs and for various input distributions, output distributions belonging to the beta or gamma family had minimum MI most number of times, while exponential distribution was a close second in some cases. From these Monte Carlo experiments, we can say that for most of the distributions, it is a good idea to transform them to gamma or beta distributions, while for gamma input distribution, we can also safely transform it to exponential or uniform distribution as well.

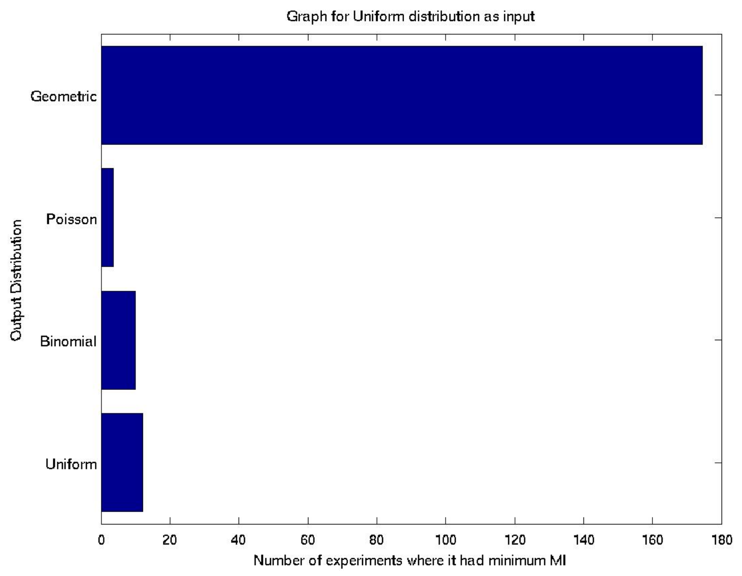


Fig. 29. Discrete Case for Uniform Distribution

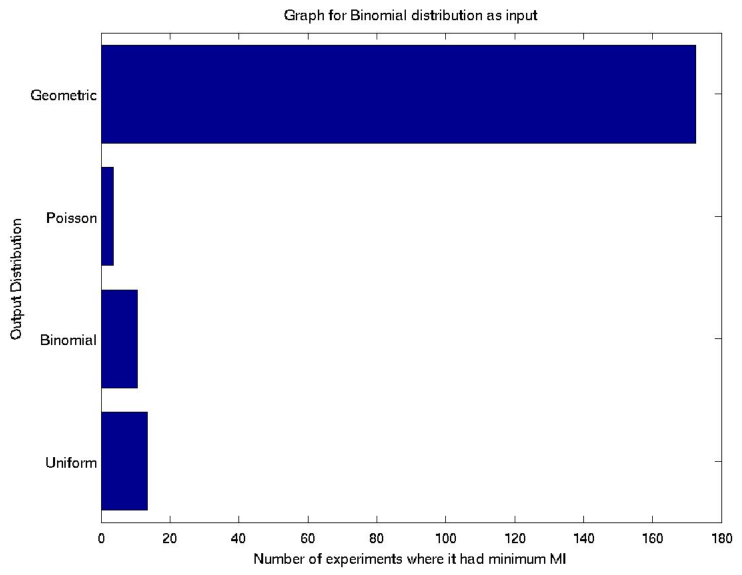


Fig. 30. Discrete Case for Binomial Distribution

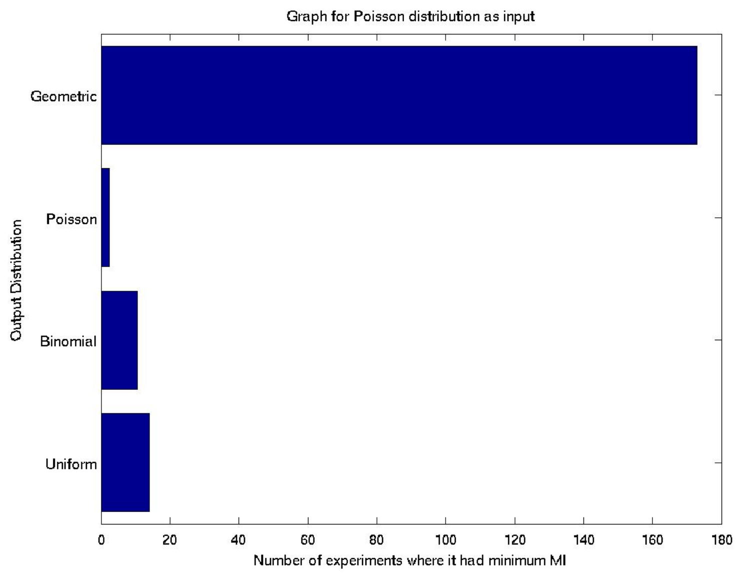


Fig. 31. Discrete Case for Poisson Distribution

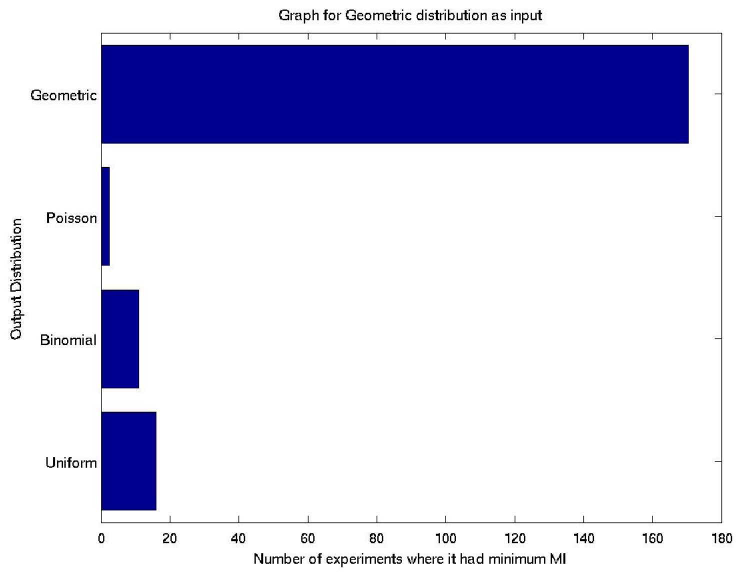


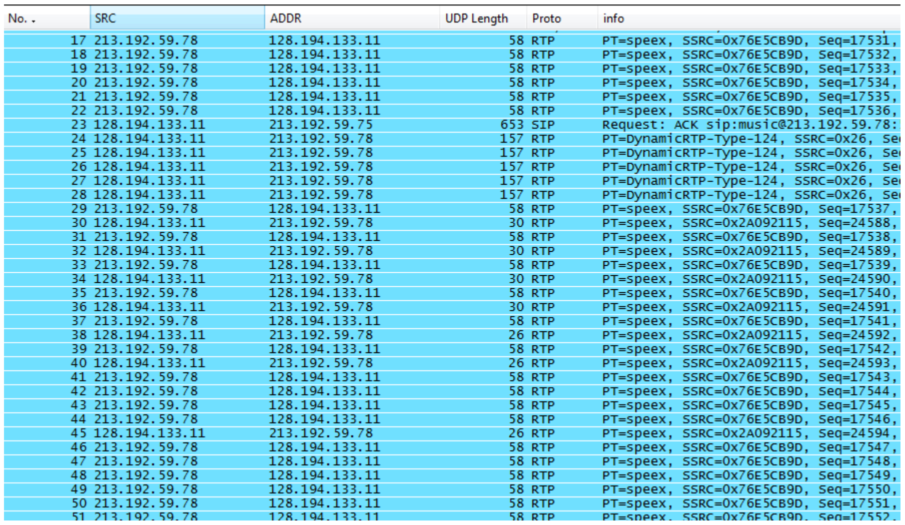
Fig. 32. Discrete Case for Geometric Distribution

All these Monte-Carlo experiments do not take into account the actual relationship between the input packet size x and output packet size y, and therefore are not very conclusive. These results can be used to formulate a padding algorithm, and its efficiency and effectiveness can be evaluated using SRPP.

APPENDIX C

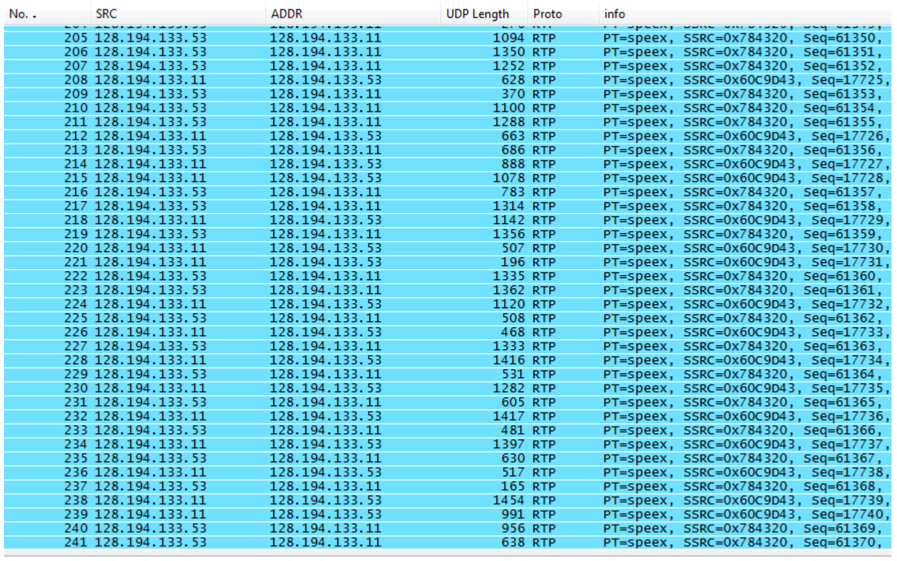
SCREENSHOTS OF WIRESHARK TRACES

Fig. 33-38 are screenshots of wireshark traces taken for SQRKal in various modes of operation, which were used for evaluation of operations as described in Chapter VI.



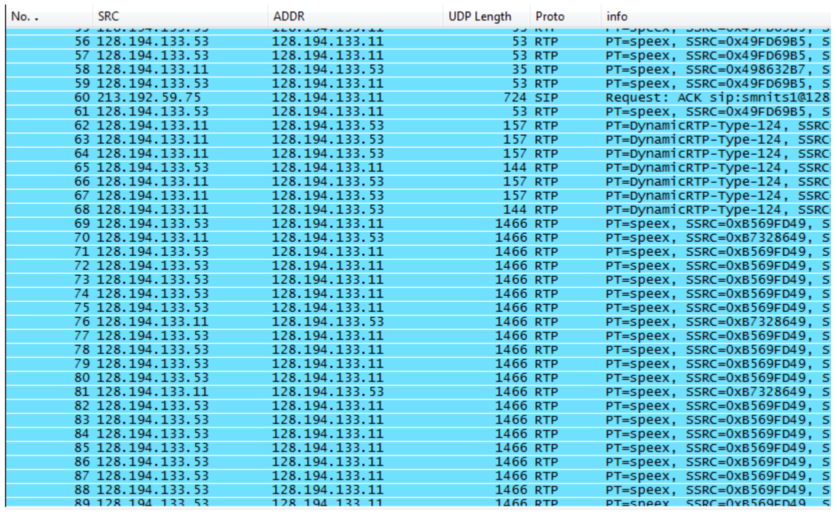
No.	SRC	ADDR	UDP Length	Proto	info
17	213.192.59.78	128.194.133.11	58	RTP	PT=speex, SSRC=0x76E5CB9D, Seq=17531,
18	213.192.59.78	128.194.133.11	58	RTP	PT=speex, SSRC=0x76E5CB9D, Seq=17532,
19	213.192.59.78	128.194.133.11	58	RTP	PT=speex, SSRC=0x76E5CB9D, Seq=17533,
20	213.192.59.78	128.194.133.11	58	RTP	PT=speex, SSRC=0x76E5CB9D, Seq=17534,
21	213.192.59.78	128.194.133.11	58	RTP	PT=speex, SSRC=0x76E5CB9D, Seq=17535,
22	213.192.59.78	128.194.133.11	58	RTP	PT=speex, SSRC=0x76E5CB9D, Seq=17536,
23	128.194.133.11	213.192.59.75	653	SIP	Request: ACK sip:music@213.192.59.78:
24	128.194.133.11	213.192.59.78	157	RTP	PT=DynamicRTP-Type-124, SSRC=0x26, Se
25	128.194.133.11	213.192.59.78	157	RTP	PT=DynamicRTP-Type-124, SSRC=0x26, Se
26	128.194.133.11	213.192.59.78	157	RTP	PT=DynamicRTP-Type-124, SSRC=0x26, Se
27	128.194.133.11	213.192.59.78	157	RTP	PT=DynamicRTP-Type-124, SSRC=0x26, Se
28	128.194.133.11	213.192.59.78	157	RTP	PT=DynamicRTP-Type-124, SSRC=0x26, Se
29	213.192.59.78	128.194.133.11	58	RTP	PT=speex, SSRC=0x76E5CB9D, Seq=17537,
30	128.194.133.11	213.192.59.78	30	RTP	PT=speex, SSRC=0x2A092115, Seq=24588,
31	213.192.59.78	128.194.133.11	58	RTP	PT=speex, SSRC=0x76E5CB9D, Seq=17538,
32	128.194.133.11	213.192.59.78	30	RTP	PT=speex, SSRC=0x2A092115, Seq=24589,
33	213.192.59.78	128.194.133.11	58	RTP	PT=speex, SSRC=0x76E5CB9D, Seq=17539,
34	128.194.133.11	213.192.59.78	30	RTP	PT=speex, SSRC=0x2A092115, Seq=24590,
35	213.192.59.78	128.194.133.11	58	RTP	PT=speex, SSRC=0x76E5CB9D, Seq=17540,
36	128.194.133.11	213.192.59.78	30	RTP	PT=speex, SSRC=0x2A092115, Seq=24591,
37	213.192.59.78	128.194.133.11	58	RTP	PT=speex, SSRC=0x76E5CB9D, Seq=17541,
38	128.194.133.11	213.192.59.78	26	RTP	PT=speex, SSRC=0x2A092115, Seq=24592,
39	213.192.59.78	128.194.133.11	58	RTP	PT=speex, SSRC=0x76E5CB9D, Seq=17542,
40	128.194.133.11	213.192.59.78	26	RTP	PT=speex, SSRC=0x2A092115, Seq=24593,
41	213.192.59.78	128.194.133.11	58	RTP	PT=speex, SSRC=0x76E5CB9D, Seq=17543,
42	213.192.59.78	128.194.133.11	58	RTP	PT=speex, SSRC=0x76E5CB9D, Seq=17544,
43	213.192.59.78	128.194.133.11	58	RTP	PT=speex, SSRC=0x76E5CB9D, Seq=17545,
44	213.192.59.78	128.194.133.11	58	RTP	PT=speex, SSRC=0x76E5CB9D, Seq=17546,
45	128.194.133.11	213.192.59.78	26	RTP	PT=speex, SSRC=0x2A092115, Seq=24594,
46	213.192.59.78	128.194.133.11	58	RTP	PT=speex, SSRC=0x76E5CB9D, Seq=17547,
47	213.192.59.78	128.194.133.11	58	RTP	PT=speex, SSRC=0x76E5CB9D, Seq=17548,
48	213.192.59.78	128.194.133.11	58	RTP	PT=speex, SSRC=0x76E5CB9D, Seq=17549,
49	213.192.59.78	128.194.133.11	58	RTP	PT=speex, SSRC=0x76E5CB9D, Seq=17550,
50	213.192.59.78	128.194.133.11	58	RTP	PT=speex, SSRC=0x76E5CB9D, Seq=17551,
51	213.192.59.78	128.194.133.11	58	RTP	PT=speex, SSRC=0x76E5CB9D, Seq=17552,

Fig. 33. Wireshark Trace for non-SRPP supporting endpoint



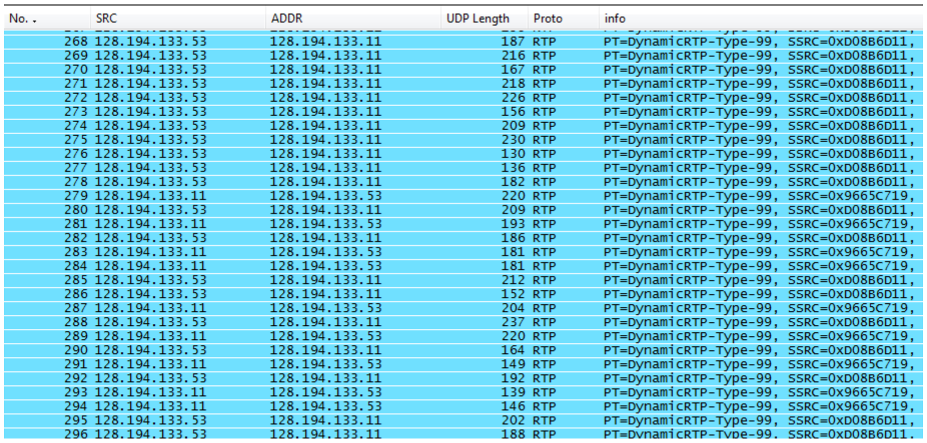
```
No.    SRC              ADDR             UDP Length Proto info
   205 128.194.133.53   128.194.133.11         1094 RTP   PT=speex, SSRC=0x784320, Seq=61350,
   206 128.194.133.53   128.194.133.11         1350 RTP   PT=speex, SSRC=0x784320, Seq=61351,
   207 128.194.133.53   128.194.133.11         1252 RTP   PT=speex, SSRC=0x784320, Seq=61352,
   208 128.194.133.11   128.194.133.53          628 RTP   PT=speex, SSRC=0x60C9D43, Seq=17725,
   209 128.194.133.53   128.194.133.11          370 RTP   PT=speex, SSRC=0x784320, Seq=61353,
   210 128.194.133.53   128.194.133.11         1100 RTP   PT=speex, SSRC=0x784320, Seq=61354,
   211 128.194.133.53   128.194.133.11         1288 RTP   PT=speex, SSRC=0x784320, Seq=61355,
   212 128.194.133.11   128.194.133.53          663 RTP   PT=speex, SSRC=0x60C9D43, Seq=17726,
   213 128.194.133.53   128.194.133.11          686 RTP   PT=speex, SSRC=0x784320, Seq=61356,
   214 128.194.133.11   128.194.133.53          888 RTP   PT=speex, SSRC=0x60C9D43, Seq=17727,
   215 128.194.133.11   128.194.133.53         1078 RTP   PT=speex, SSRC=0x60C9D43, Seq=17728,
   216 128.194.133.53   128.194.133.11          783 RTP   PT=speex, SSRC=0x784320, Seq=61357,
   217 128.194.133.53   128.194.133.11         1314 RTP   PT=speex, SSRC=0x784320, Seq=61358,
   218 128.194.133.11   128.194.133.53         1142 RTP   PT=speex, SSRC=0x60C9D43, Seq=17729,
   219 128.194.133.53   128.194.133.11         1356 RTP   PT=speex, SSRC=0x784320, Seq=61359,
   220 128.194.133.11   128.194.133.53          507 RTP   PT=speex, SSRC=0x60C9D43, Seq=17730,
   221 128.194.133.11   128.194.133.53          196 RTP   PT=speex, SSRC=0x60C9D43, Seq=17731,
   222 128.194.133.53   128.194.133.11         1335 RTP   PT=speex, SSRC=0x784320, Seq=61360,
   223 128.194.133.53   128.194.133.11         1362 RTP   PT=speex, SSRC=0x784320, Seq=61361,
   224 128.194.133.11   128.194.133.53         1120 RTP   PT=speex, SSRC=0x60C9D43, Seq=17732,
   225 128.194.133.53   128.194.133.11          508 RTP   PT=speex, SSRC=0x784320, Seq=61362,
   226 128.194.133.11   128.194.133.53          468 RTP   PT=speex, SSRC=0x60C9D43, Seq=17733,
   227 128.194.133.53   128.194.133.11         1333 RTP   PT=speex, SSRC=0x784320, Seq=61363,
   228 128.194.133.11   128.194.133.53         1416 RTP   PT=speex, SSRC=0x60C9D43, Seq=17734,
   229 128.194.133.53   128.194.133.11          531 RTP   PT=speex, SSRC=0x784320, Seq=61364,
   230 128.194.133.11   128.194.133.53         1282 RTP   PT=speex, SSRC=0x60C9D43, Seq=17735,
   231 128.194.133.53   128.194.133.11          605 RTP   PT=speex, SSRC=0x784320, Seq=61365,
   232 128.194.133.11   128.194.133.53         1417 RTP   PT=speex, SSRC=0x60C9D43, Seq=17736,
   233 128.194.133.53   128.194.133.11          481 RTP   PT=speex, SSRC=0x784320, Seq=61366,
   234 128.194.133.11   128.194.133.53         1397 RTP   PT=speex, SSRC=0x60C9D43, Seq=17737,
   235 128.194.133.53   128.194.133.11          630 RTP   PT=speex, SSRC=0x784320, Seq=61367,
   236 128.194.133.11   128.194.133.53          517 RTP   PT=speex, SSRC=0x60C9D43, Seq=17738,
   237 128.194.133.53   128.194.133.11          165 RTP   PT=speex, SSRC=0x784320, Seq=61368,
   238 128.194.133.11   128.194.133.53         1454 RTP   PT=speex, SSRC=0x60C9D43, Seq=17739,
   239 128.194.133.11   128.194.133.53          991 RTP   PT=speex, SSRC=0x60C9D43, Seq=17740,
   240 128.194.133.53   128.194.133.11          956 RTP   PT=speex, SSRC=0x784320, Seq=61369,
   241 128.194.133.53   128.194.133.11          638 RTP   PT=speex, SSRC=0x784320, Seq=61370,
```

Fig. 34. Wireshark Trace for RTP max-random mode



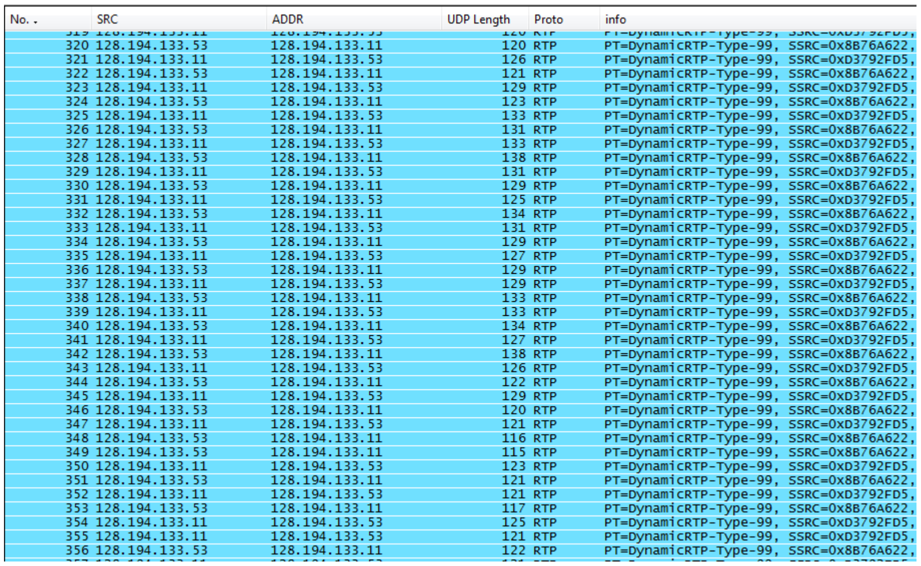
```
No.    SRC              ADDR             UDP Length Proto info
    56 128.194.133.53   128.194.133.11           53 RTP   PT=speex, SSRC=0x49FD69B5, S
    57 128.194.133.53   128.194.133.11           53 RTP   PT=speex, SSRC=0x49FD69B5, S
    58 128.194.133.11   128.194.133.53           35 RTP   PT=speex, SSRC=0x498632B7, S
    59 128.194.133.53   128.194.133.11           53 RTP   PT=speex, SSRC=0x49FD69B5, S
    60 213.192.59.75    128.194.133.11          724 SIP   Request: ACK sip:smnits1@128
    61 128.194.133.53   128.194.133.11           53 RTP   PT=speex, SSRC=0x49FD69B5, S
    62 128.194.133.11   128.194.133.53          157 RTP   PT=DynamicRTP-Type-124, SSRC
    63 128.194.133.11   128.194.133.53          157 RTP   PT=DynamicRTP-Type-124, SSRC
    64 128.194.133.11   128.194.133.53          157 RTP   PT=DynamicRTP-Type-124, SSRC
    65 128.194.133.53   128.194.133.11          144 RTP   PT=DynamicRTP-Type-124, SSRC
    66 128.194.133.11   128.194.133.53          157 RTP   PT=DynamicRTP-Type-124, SSRC
    67 128.194.133.11   128.194.133.53          157 RTP   PT=DynamicRTP-Type-124, SSRC
    68 128.194.133.11   128.194.133.53          144 RTP   PT=DynamicRTP-Type-124, SSRC
    69 128.194.133.53   128.194.133.11         1466 RTP   PT=speex, SSRC=0xB569FD49, S
    70 128.194.133.11   128.194.133.53         1466 RTP   PT=speex, SSRC=0xB7328649, S
    71 128.194.133.53   128.194.133.11         1466 RTP   PT=speex, SSRC=0xB569FD49, S
    72 128.194.133.53   128.194.133.11         1466 RTP   PT=speex, SSRC=0xB569FD49, S
    73 128.194.133.53   128.194.133.11         1466 RTP   PT=speex, SSRC=0xB569FD49, S
    74 128.194.133.53   128.194.133.11         1466 RTP   PT=speex, SSRC=0xB569FD49, S
    75 128.194.133.53   128.194.133.11         1466 RTP   PT=speex, SSRC=0xB569FD49, S
    76 128.194.133.11   128.194.133.53         1466 RTP   PT=speex, SSRC=0xB7328649, S
    77 128.194.133.53   128.194.133.11         1466 RTP   PT=speex, SSRC=0xB569FD49, S
    78 128.194.133.53   128.194.133.11         1466 RTP   PT=speex, SSRC=0xB569FD49, S
    79 128.194.133.53   128.194.133.11         1466 RTP   PT=speex, SSRC=0xB569FD49, S
    80 128.194.133.53   128.194.133.11         1466 RTP   PT=speex, SSRC=0xB569FD49, S
    81 128.194.133.11   128.194.133.53         1466 RTP   PT=speex, SSRC=0xB7328649, S
    82 128.194.133.53   128.194.133.11         1466 RTP   PT=speex, SSRC=0xB569FD49, S
    83 128.194.133.53   128.194.133.11         1466 RTP   PT=speex, SSRC=0xB569FD49, S
    84 128.194.133.53   128.194.133.11         1466 RTP   PT=speex, SSRC=0xB569FD49, S
    85 128.194.133.53   128.194.133.11         1466 RTP   PT=speex, SSRC=0xB569FD49, S
    86 128.194.133.53   128.194.133.11         1466 RTP   PT=speex, SSRC=0xB569FD49, S
    87 128.194.133.53   128.194.133.11         1466 RTP   PT=speex, SSRC=0xB569FD49, S
    88 128.194.133.53   128.194.133.11         1466 RTP   PT=speex, SSRC=0xB569FD49, S
    89 128.194.133.53   128.194.133.11         1466 RTP   PT=speex, SSRC=0xB569FD49, S
```

Fig. 35. Wireshark Trace for RTP full-bandwidth mode



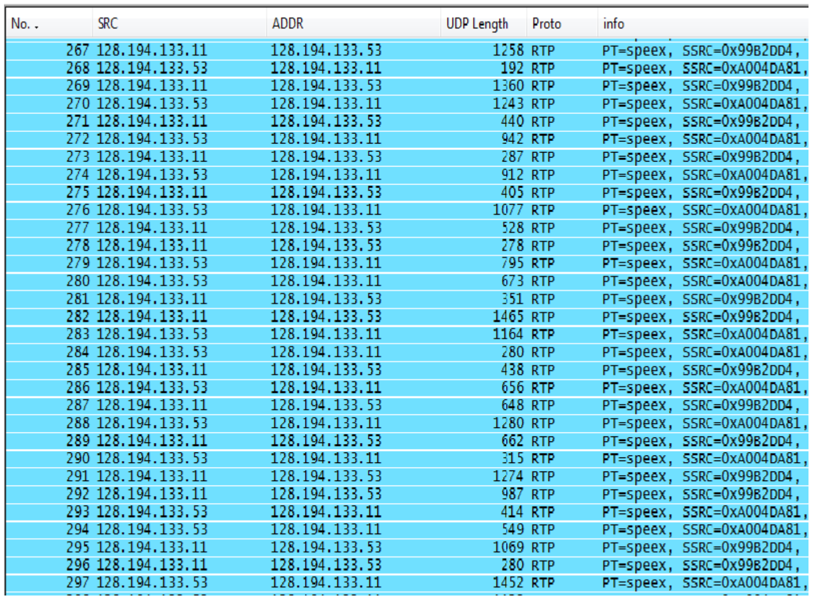
No.	SRC	ADDR	UDP Length	Proto	info
268	128.194.133.53	128.194.133.11	187	RTP	PT=DynamicRTP-Type-99, SSRC=0xD08B6D11,
269	128.194.133.53	128.194.133.11	216	RTP	PT=DynamicRTP-Type-99, SSRC=0xD08B6D11,
270	128.194.133.53	128.194.133.11	167	RTP	PT=DynamicRTP-Type-99, SSRC=0xD08B6D11,
271	128.194.133.53	128.194.133.11	218	RTP	PT=DynamicRTP-Type-99, SSRC=0xD08B6D11,
272	128.194.133.53	128.194.133.11	226	RTP	PT=DynamicRTP-Type-99, SSRC=0xD08B6D11,
273	128.194.133.53	128.194.133.11	156	RTP	PT=DynamicRTP-Type-99, SSRC=0xD08B6D11,
274	128.194.133.53	128.194.133.11	209	RTP	PT=DynamicRTP-Type-99, SSRC=0xD08B6D11,
275	128.194.133.53	128.194.133.11	230	RTP	PT=DynamicRTP-Type-99, SSRC=0xD08B6D11,
276	128.194.133.53	128.194.133.11	130	RTP	PT=DynamicRTP-Type-99, SSRC=0xD08B6D11,
277	128.194.133.53	128.194.133.11	136	RTP	PT=DynamicRTP-Type-99, SSRC=0xD08B6D11,
278	128.194.133.53	128.194.133.11	182	RTP	PT=DynamicRTP-Type-99, SSRC=0xD08B6D11,
279	128.194.133.11	128.194.133.53	220	RTP	PT=DynamicRTP-Type-99, SSRC=0x9665C719,
280	128.194.133.53	128.194.133.11	209	RTP	PT=DynamicRTP-Type-99, SSRC=0xD08B6D11,
281	128.194.133.11	128.194.133.53	193	RTP	PT=DynamicRTP-Type-99, SSRC=0x9665C719,
282	128.194.133.53	128.194.133.11	186	RTP	PT=DynamicRTP-Type-99, SSRC=0xD08B6D11,
283	128.194.133.11	128.194.133.53	181	RTP	PT=DynamicRTP-Type-99, SSRC=0x9665C719,
284	128.194.133.11	128.194.133.53	181	RTP	PT=DynamicRTP-Type-99, SSRC=0x9665C719,
285	128.194.133.53	128.194.133.11	212	RTP	PT=DynamicRTP-Type-99, SSRC=0xD08B6D11,
286	128.194.133.53	128.194.133.11	152	RTP	PT=DynamicRTP-Type-99, SSRC=0xD08B6D11,
287	128.194.133.11	128.194.133.53	204	RTP	PT=DynamicRTP-Type-99, SSRC=0x9665C719,
288	128.194.133.53	128.194.133.11	237	RTP	PT=DynamicRTP-Type-99, SSRC=0xD08B6D11,
289	128.194.133.11	128.194.133.53	220	RTP	PT=DynamicRTP-Type-99, SSRC=0x9665C719,
290	128.194.133.53	128.194.133.11	164	RTP	PT=DynamicRTP-Type-99, SSRC=0xD08B6D11,
291	128.194.133.11	128.194.133.53	149	RTP	PT=DynamicRTP-Type-99, SSRC=0x9665C719,
292	128.194.133.53	128.194.133.11	192	RTP	PT=DynamicRTP-Type-99, SSRC=0xD08B6D11,
293	128.194.133.11	128.194.133.53	139	RTP	PT=DynamicRTP-Type-99, SSRC=0x9665C719,
294	128.194.133.11	128.194.133.53	146	RTP	PT=DynamicRTP-Type-99, SSRC=0x9665C719,
295	128.194.133.53	128.194.133.11	202	RTP	PT=DynamicRTP-Type-99, SSRC=0xD08B6D11,
296	128.194.133.53	128.194.133.11	188	RTP	PT=DynamicRTP-Type-99, SSRC=0xD08B6D11,

Fig. 36. Wireshark Trace for RTP Gradual Ascent Padding Algorithm



No.	SRC	ADDR	UDP Length	Proto	info
320	128.194.133.53	128.194.133.11	120	RTP	PT=DynamicRTP-Type-99, SSRC=0x8B76A622,
321	128.194.133.11	128.194.133.53	126	RTP	PT=DynamicRTP-Type-99, SSRC=0xD3792FD5,
322	128.194.133.53	128.194.133.11	121	RTP	PT=DynamicRTP-Type-99, SSRC=0x8B76A622,
323	128.194.133.11	128.194.133.53	129	RTP	PT=DynamicRTP-Type-99, SSRC=0xD3792FD5,
324	128.194.133.53	128.194.133.11	123	RTP	PT=DynamicRTP-Type-99, SSRC=0x8B76A622,
325	128.194.133.11	128.194.133.53	133	RTP	PT=DynamicRTP-Type-99, SSRC=0xD3792FD5,
326	128.194.133.53	128.194.133.11	131	RTP	PT=DynamicRTP-Type-99, SSRC=0x8B76A622,
327	128.194.133.11	128.194.133.53	133	RTP	PT=DynamicRTP-Type-99, SSRC=0xD3792FD5,
328	128.194.133.53	128.194.133.11	138	RTP	PT=DynamicRTP-Type-99, SSRC=0x8B76A622,
329	128.194.133.11	128.194.133.53	131	RTP	PT=DynamicRTP-Type-99, SSRC=0xD3792FD5,
330	128.194.133.53	128.194.133.11	129	RTP	PT=DynamicRTP-Type-99, SSRC=0x8B76A622,
331	128.194.133.11	128.194.133.53	125	RTP	PT=DynamicRTP-Type-99, SSRC=0xD3792FD5,
332	128.194.133.53	128.194.133.11	134	RTP	PT=DynamicRTP-Type-99, SSRC=0x8B76A622,
333	128.194.133.11	128.194.133.53	131	RTP	PT=DynamicRTP-Type-99, SSRC=0xD3792FD5,
334	128.194.133.53	128.194.133.11	129	RTP	PT=DynamicRTP-Type-99, SSRC=0x8B76A622,
335	128.194.133.11	128.194.133.53	127	RTP	PT=DynamicRTP-Type-99, SSRC=0xD3792FD5,
336	128.194.133.53	128.194.133.11	129	RTP	PT=DynamicRTP-Type-99, SSRC=0x8B76A622,
337	128.194.133.11	128.194.133.53	129	RTP	PT=DynamicRTP-Type-99, SSRC=0xD3792FD5,
338	128.194.133.53	128.194.133.11	133	RTP	PT=DynamicRTP-Type-99, SSRC=0x8B76A622,
339	128.194.133.11	128.194.133.53	133	RTP	PT=DynamicRTP-Type-99, SSRC=0xD3792FD5,
340	128.194.133.53	128.194.133.11	134	RTP	PT=DynamicRTP-Type-99, SSRC=0x8B76A622,
341	128.194.133.11	128.194.133.53	127	RTP	PT=DynamicRTP-Type-99, SSRC=0xD3792FD5,
342	128.194.133.53	128.194.133.11	138	RTP	PT=DynamicRTP-Type-99, SSRC=0x8B76A622,
343	128.194.133.11	128.194.133.53	126	RTP	PT=DynamicRTP-Type-99, SSRC=0xD3792FD5,
344	128.194.133.53	128.194.133.11	122	RTP	PT=DynamicRTP-Type-99, SSRC=0x8B76A622,
345	128.194.133.11	128.194.133.53	129	RTP	PT=DynamicRTP-Type-99, SSRC=0xD3792FD5,
346	128.194.133.53	128.194.133.11	120	RTP	PT=DynamicRTP-Type-99, SSRC=0x8B76A622,
347	128.194.133.11	128.194.133.53	121	RTP	PT=DynamicRTP-Type-99, SSRC=0xD3792FD5,
348	128.194.133.53	128.194.133.11	116	RTP	PT=DynamicRTP-Type-99, SSRC=0x8B76A622,
349	128.194.133.53	128.194.133.11	115	RTP	PT=DynamicRTP-Type-99, SSRC=0x8B76A622,
350	128.194.133.11	128.194.133.53	123	RTP	PT=DynamicRTP-Type-99, SSRC=0xD3792FD5,
351	128.194.133.53	128.194.133.11	121	RTP	PT=DynamicRTP-Type-99, SSRC=0x8B76A622,
352	128.194.133.11	128.194.133.53	121	RTP	PT=DynamicRTP-Type-99, SSRC=0xD3792FD5,
353	128.194.133.53	128.194.133.11	117	RTP	PT=DynamicRTP-Type-99, SSRC=0x8B76A622,
354	128.194.133.11	128.194.133.53	125	RTP	PT=DynamicRTP-Type-99, SSRC=0xD3792FD5,
355	128.194.133.11	128.194.133.53	121	RTP	PT=DynamicRTP-Type-99, SSRC=0xD3792FD5,
356	128.194.133.53	128.194.133.11	122	RTP	PT=DynamicRTP-Type-99, SSRC=0x8B76A622,

Fig. 37. Wireshark Trace for RTP Slight Perturbation Algorithm



```
No. .        SRC                 ADDR                UDP Length  Proto   info
     267 128.194.133.11     128.194.133.53           1258 RTP    PT=speex, SSRC=0x99B2DD4,
     268 128.194.133.53     128.194.133.11            192 RTP    PT=speex, SSRC=0xA004DA81,
     269 128.194.133.11     128.194.133.53           1360 RTP    PT=speex, SSRC=0x99B2DD4,
     270 128.194.133.53     128.194.133.11           1243 RTP    PT=speex, SSRC=0xA004DA81,
     271 128.194.133.11     128.194.133.53            440 RTP    PT=speex, SSRC=0x99B2DD4,
     272 128.194.133.53     128.194.133.11            942 RTP    PT=speex, SSRC=0xA004DA81,
     273 128.194.133.11     128.194.133.53            287 RTP    PT=speex, SSRC=0x99B2DD4,
     274 128.194.133.53     128.194.133.11            912 RTP    PT=speex, SSRC=0xA004DA81,
     275 128.194.133.11     128.194.133.53            405 RTP    PT=speex, SSRC=0x99B2DD4,
     276 128.194.133.53     128.194.133.11           1077 RTP    PT=speex, SSRC=0xA004DA81,
     277 128.194.133.11     128.194.133.53            528 RTP    PT=speex, SSRC=0x99B2DD4,
     278 128.194.133.11     128.194.133.53            278 RTP    PT=speex, SSRC=0x99B2DD4,
     279 128.194.133.53     128.194.133.11            795 RTP    PT=speex, SSRC=0xA004DA81,
     280 128.194.133.53     128.194.133.11            673 RTP    PT=speex, SSRC=0xA004DA81,
     281 128.194.133.11     128.194.133.53            351 RTP    PT=speex, SSRC=0x99B2DD4,
     282 128.194.133.11     128.194.133.53           1465 RTP    PT=speex, SSRC=0x99B2DD4,
     283 128.194.133.53     128.194.133.11           1164 RTP    PT=speex, SSRC=0xA004DA81,
     284 128.194.133.53     128.194.133.11            280 RTP    PT=speex, SSRC=0xA004DA81,
     285 128.194.133.11     128.194.133.53            438 RTP    PT=speex, SSRC=0x99B2DD4,
     286 128.194.133.53     128.194.133.11            656 RTP    PT=speex, SSRC=0xA004DA81,
     287 128.194.133.11     128.194.133.53            648 RTP    PT=speex, SSRC=0x99B2DD4,
     288 128.194.133.53     128.194.133.11           1280 RTP    PT=speex, SSRC=0xA004DA81,
     289 128.194.133.11     128.194.133.53            662 RTP    PT=speex, SSRC=0x99B2DD4,
     290 128.194.133.53     128.194.133.11            315 RTP    PT=speex, SSRC=0xA004DA81,
     291 128.194.133.11     128.194.133.53           1274 RTP    PT=speex, SSRC=0x99B2DD4,
     292 128.194.133.11     128.194.133.53            987 RTP    PT=speex, SSRC=0x99B2DD4,
     293 128.194.133.53     128.194.133.11            414 RTP    PT=speex, SSRC=0xA004DA81,
     294 128.194.133.53     128.194.133.11            549 RTP    PT=speex, SSRC=0xA004DA81,
     295 128.194.133.11     128.194.133.53           1069 RTP    PT=speex, SSRC=0x99B2DD4,
     296 128.194.133.11     128.194.133.53            280 RTP    PT=speex, SSRC=0x99B2DD4,
     297 128.194.133.53     128.194.133.11           1452 RTP    PT=speex, SSRC=0xA004DA81,
```

Fig. 38. Wireshark Trace for RTP with Dummy Bursts

VITA

Name: Saswat Mohanty

Address: Department of Computer Science and Engineering, Texas A&M University, TAMU 3312, College Station TX 77843-3112

Permanent Address: 447 Mahanadi Vihar, Cuttack, Orissa, India 753004

Email Address: smohanty@cs.tamu.edu

Education:

B. Tech., Computer Science, National Institute of Technology, Silchar, India, 2007

M.S., Computer Science, Texas A&M University, 2011

The typist for this thesis was Saswat Mohanty.