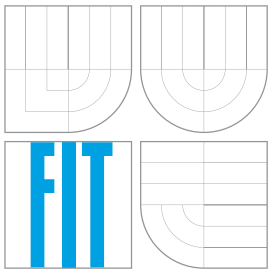
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

NÁVRH A ANALÝZA VÝKONNOSTI PARALELNÍHO ZPRACOVÁNÍ SRTP PŘENOSŮ

DESIGN AND PERFORMANCE ANALYSIS OF PARALLEL PROCESSING OF SRTP PACKETS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

JAN WOZNIAK

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. PETER JURNEČKA

BRNO 2012

Abstrakt

Šifrování multimediálních datových přenosů v reálném čase je jednou z úloh telekomunikační infrastruktury pro dosažení nezbytné úrovně zabezpečení. Rychlost provedení šifrovacího algoritmu může hrát klíčovou roli ve velikosti zpoždění jednotlivých paketů a proto je tento úkol zajimavým z hlediska optimalizačních metod. Tato práce se zaměřuje na možnosti paralelizace zpracování SRTP pro účely telefonní ústředny s využitím OpenCL frameworku a následnou analyzu potenciálního zlepšení.

Abstract

Encryption of real-time multimedia data transferes is one of the tasks for telecommunication infrastructure in order to provide essential level of security. Execution time of ciphering algorithm could play fundamental role in delay of the packets, therefore it provides interesting challenge in terms of optimization methods. This thesis focuses on paralellization possibilities of processing SRTP for the purposes of private branch exchange with the use of OpenCL framework and analysis of potential improvement.

Klíčová slova

AES, obecné výpočty na GPU, OpenCL, paralelní výpočty, SRTP, SIP, telefonní ústředna, brána, VoIP.

Keywords

AES, general-purpose GPU, OpenCL, parallel computations, SRTP, SIP, private branch exchange, gateway, VoIP.

Citace

Jan Wozniak: Design and Performance Analysis of Parallel Processing of SRTP Packets, diplomová práce, Brno, FIT VUT v Brně, 2012

Design and Performance Analysis of Parallel Processing of SRTP Packets

Prohlášení

Prohlašuji, že jsem tento semestrální projekt vypracoval samostatně pod vedením pana Ing. Petera Jurnečky.

.....
Jan Wozniak
May 2, 2013

Contents

1 Introduction **3**

2 Secure Real-time Transport Protocol **5**
2.1 Packet Structure . . . 5
2.2 Cryptographic Context . . . 6
2.3 Master Key Exchange . . . 7
2.4 Protocol Summary . . . 7
2.5 AES . . . 8
2.5.1 Mathematical Preliminaries . . . 8
2.5.2 Algorithm Description . . . 9
2.5.3 Block Cipher Modes . . . 12

3 General Purpose GPU **14**
3.1 OpenCL . . . 15
3.1.1 Platform Model . . . 16
3.1.2 Execution Model . . . 16
3.1.3 Memory Model . . . 16

4 Design **18**
4.1 Design Patterns . . . 18
4.1.1 Mediator Pattern . . . 18
4.1.2 Singleton Pattern . . . 19
4.1.3 Factory Method Pattern . . . 19
4.1.4 Protocol Stack Pattern . . . 19
4.2 . . . 19
4.3 SIP Gateway . . . 20
4.4 SRTP Stack . . . 22
4.4.1 SRTP Processing . . . 22
4.4.2 Serial Processing . . . 24
4.4.3 Massive Parallel Processing . . . 24
4.4.4 Persistent Thread Processing . . . 24

5 Implementation **25**

6 Results **26**

7 Conclusion **27**

A AES Properties **31**

Chapter 1

Introduction

One of the essential metrics for measuring VoIP gateway's performance is the number of simultaneous calls. It is affected mostly by the computational demands of used communication protocols and number of registered users. While the count of registered users provides very limited room for improvement by the nature of the problem itself, there could be wide variety of aproaches in implementing the protocol stacks.

The communication protocols for VoIP gateway can be divided into two groups. Signalization, which consists mostly of textually represented protocols, where the messages' occurance is either periodical with quite small frequency, or based on the users initiative which is a stochastical event depending on the activity of the user. However, generally the recurrance of both is rather similar. Comparably more resources during indirect simultaneous call sessions consumes processing the second group of protocols, transport of multimedia packets. Since security has recently grown to be necessary feature in VoIP communication and the encryption and decryption processes are designed with the idea of optimization, it is primary scope of interest of this thesis.

Development and results in the areas of parallel architectures shows that many procedures could be distinctively accelerated by executing the algorithm on the processing unit capable of parallel computations. Therefore, target of this thesis is implementation and analysis of parallel processing of encrypted real-time multimedia data transfere.

Chapter 2 describes the structures and algorithms used in Secure Real-time Transport Protocol. Increased attention is devoted to explanation of Advanced Encryption Standard, which is default cipher used in SRTP, including brief theoretical background and analysis of SRTP and AES. Because SRTP doesn't provide key exchange mechanism for symetric AES cipher, the chapter also includes description of selected protocol extensions for this task.

Chapter 3 provides basic information about graphic processing unit and the usage of GPU for general purpose computations. Part of the chapter is principal explanation of OpenCL framework and its elementary usage for the developer. As the parallel processing is diverse and wide study, the area of parallel paradigm that could be associated to the further implementation of this thesis is mentioned with particular interest and focus.

Chapter 4 defines the term SIP gateway for the context of this thesis, discusses the design of such gateway and includes listing of selected further implemented protocol stacks, their mutual interaction and possible improvement of processing the passing data. The highest amount of attention is devoted to the comparison of different approaches to design of SRTP stack and identification of main characteristics of native OpenCL programming pattern in contrast to persistent thread model. The advantages of both parallel implementations over

serial code executed on the same hardware is mentioned as well. Short introduction and description of used design patterns is included in order to provide better comprehensibility of the application schemes.

Chapter 5 covers the reference implementation of the previous theoretical part of this thesis, used techniques and algorithms and reasonig behind their selection. Even though the focus of the thesis is primarily research of available contemporary methods there were many restrictions. The requirements of this chapter arise from currently used implementation and hardware limitation of the gateway.

Finally chapters 6 and 7 sumarize the potential benefits of usage the GPGPU for the number of maximal simultaneous calls and shows visualization of achieved results in improvement and decrease of latency. Also these chapter discusses possible contribution to related topics, such as transcoding of media compressing codecs which parallel implementation may provide even higher level of improvement.

Chapter 2

Secure Real-time Transport Protocol

To achieve confidentiality and neccessary security for real-time multimedia transmission over TCP/IP connection there has been invented SRTP[12]. Except previously mentioned, it provides message authentication and replay protection for both RTP and RTCP traffic, however, the thesis is going to focus on the implementation and computation time of the security. The default cipher is AES in counter mode.

2.1 Packet Structure

SRTP packet can be described as RTP extension. It keeps the RTP fields of the packet such as:

- Version (V) – two bit number which currently is equal to 2.
- Padding (P) – boolean value whether the padding is set.
- Extension (X) – if this field is set, fixed header must be followed by exactly one extension header.
- CSRC count (CC) – number of CSRC identifiers that follow the fixed header.
- Marker (M) – interpretation defined by a profile.
- Payload Type (PT) – identifies the type of payload
- Sequence Number (SEQ) – incremets by one for each RTP packet.
- Timestamp (TS) – reflecting the exact moment the payload was sampled.
- Synchronization Source Identifier (SSRC) – identifier of RTP synchronization source within the single RTP session.
- Contributing Source Identifiers (CSRC) – list of 0 to 15 items identifying contributing sources.

The SRTP protocol defines that only payload is encrypted and also describes new fields in the RTP header.

- Master Key Identifier (MKI) – unique identifier of the master key (previously signaled) to be used in session key derivation.
- Authentication Tag – carries message authentication data. If both encryption and authentication are used, encryption should be applied first.

The packet length is variable and depends on number of CSRC used and length of payload. The following scheme describes the packet with proportional sizes of each field.

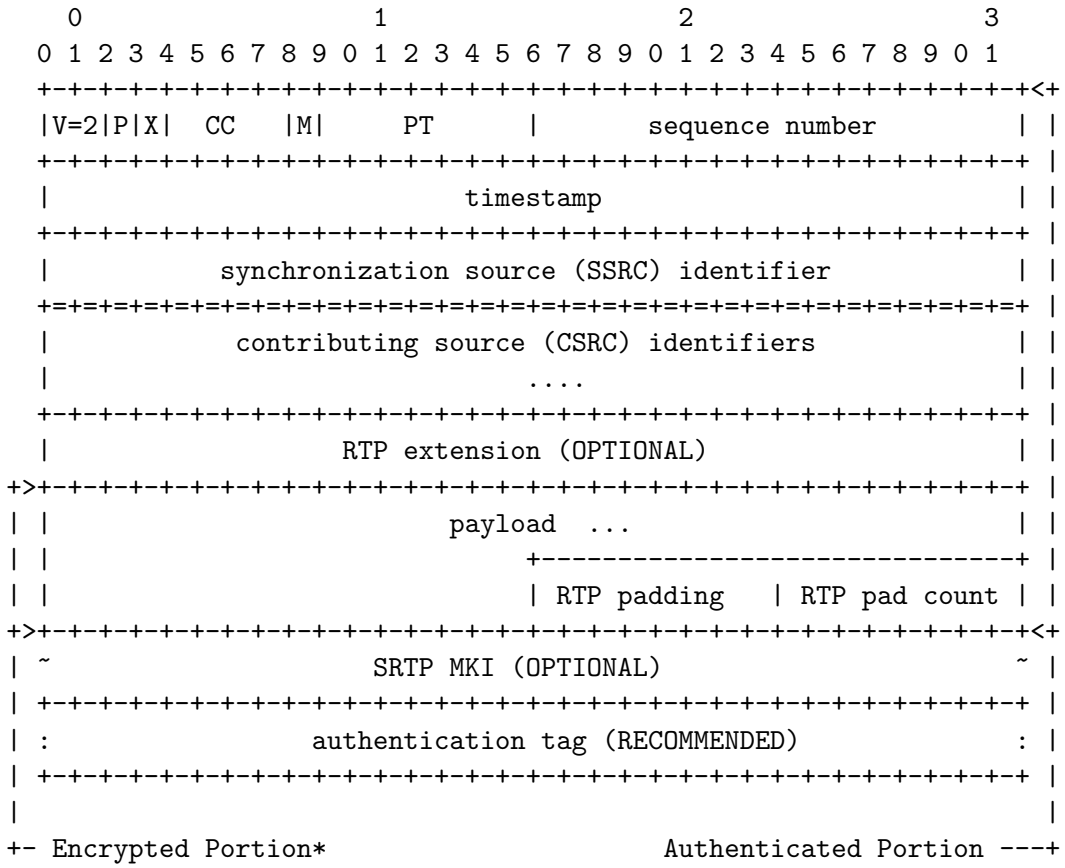


Figure 2.1: SRTP packet structure.

2.2 Cryptographic Context

In order to implement SRTP stack in the application, it is necessary to preserve certain information about each encrypted session, which is called *cryptographic context*. It must consist of the following:

- Rollover Counter – 32-bit unsigned number, records how many times has the RTP sequential number been reseted to zero passed the value 65 535.
- Highest Received SEQ – 16-bit unsigned number
- Identifier of the Encryption Algorithm – the cipher and its mode

- Replay List – containing indices of recently received and authenticated SRTP packets
- MKI – if the MKI is present in current session, the length of the MKI field in octets, actual value of currently used MKI
- Master Keys – enumeration of random and secret master keys and counter for each key of how many packets have been sent with that key. Single Master Key identifies SRTP stream and corresponding SRTCP stream.
- Session Keys – current key for encryption and authentication including stored their lengths in `n_e` and `n_a`

And for every master key, the cryptographic context may contain also random but possibly public *Master Salt* which will be used in key derivation.

2.3 Master Key Exchange

There are three most common protocols for key exchange in SRTP session between the end users – SDES, MIKEY, and ZRTP. They differ in what protocol in VoIP communication they extend, provided security guarantees and possible communication overhead.

ZRTP is a protocol extension of RTP for secure establishing session key using Diffie-Hellman key exchange improved for detection of man-in-the-middle attack, which is briefly described in section 2.4, [28]. Another advantage of the improvement is that it doesn't require any prior shared secret nor public key infrastructure.

SDES is protocol extension of SDP[19, 11] typically in SIP[26] message. It is responsibility of the SIP stack to protect the key as secured secret, which is possible via TLS connection for instance.

MIKEY defines the key exchange as part of SDP payload in SIP message. The algorithm is basic Diffie-Hellman which requires either prior shared secret or PKI[1]. The SIP stack doesn't have to protect the transfered information any further.

2.4 Protocol Summary

Main concers about the use of SRTP are whether the increase of computational complexity and packet size don't make RTP hardly usable and what degree of security does it provide.

Computational Overhead

In VoIP communication the time has essential impact on the quality of transmitted information, therefore it is important that ensuring the security of RTP wouldn't increase the latency over the acceptable level. Among commont limitations of real-time communications belong[25]:

- Maximal tolerable latency round-trip time 300ms.
- Smaller packet loss than 5%.
- Sensitivity to factors that are difficult to objectively measure such as jitter.

[1] Public Key Infrastructure for digital certificates

It has been proven that increase in size of the packet SRTP is insignificant compared to the RTP[9, 10]. Average throughput of secured VoIP is usually around 2% more than unsecured VoIP.

Security

VoIP suffers from many simillar security threats as other standard internet services.

Man in the middle in computer security is form of active eavesdropping. The attacker creates connections to both endpoints of the session which allows him to monitor, record or modify the packets in communication making the endpoints believe that the conversation is secured. Protection against such attack could be achieved by key negotiating protocol *ZRTP* which is able to detect this activity[28].

Denial of service is considered an attempt to make target machine unavailable to to its intended users. Typical method of this attack is to saturate the target machine with excessive requests that could lead to overloading the machine. Replay protection mechanism of SRTP with replay lists and authentication headers provide sufficient protection against DoS attack[12, 7].

2.5 AES

This section treats neccessary theoretical background of Advanced Encryption Standard, which is the default cipher, and as the text has been written the only cipher, of Secure Real-time Transport Protocol used in VoIP communication.

Advanced Encryption Standard is symetric block cipher which means it uses the same key for both encryption and decryption and encodes the input in uniform sized blocks. The algorithm was developed to supersede *Data Encryption Standard* due to various security reasons[2] in electronic data transmission.

For this purpose National Institute of Standards and Technology (NIST) announced public competition for new encryption standard in 1997 and considering multiple requirements the *Rijndael*[3] was selected as the most suitable algorithm for the task[8].

2.5.1 Mathematical Preliminaries

All the bytes in AES are interpreted as 8-bit values in finite field 2^8. For better readability the values are printed using hexadecimal notation. Following mathematical therms and operations are used in AES aglorithm:

Galois field

In algebra Galois field is finite field with finite number of elements. Common notation is $GF(p^k)$ where p is prime number and k is positive natural number. Therefore it is possible to classify the Galois fields by their size, because only single $GF(p^k)$ exists for each p and k. Characteristics of the field is equal to the p.

[2] For instance COPACOBANA is FPGA based machine that could find an exhaustive key for DES in no longer than a week[20].

[3] Rijndael was original name of the AES as abreviation of authors' names – Joan Daemen and Vincent Rijmen.

Each byte is in fact a polynomial with degree equal to 7 with coeficients b_i 0 or 1 and this notation $b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x^1 + b_0$. The decimal number 95 could be represented as:

- $5F$ in hexadecimal
- 0101 1111 in binary as a byte
- $x^6 + x^4 + x^3 + x^2 + x^1 + 1$ as polynomial with degree equal to 7

Addition

Addition is defined as addition of coeficinets of both polynomials modulo 2. This operation has the same result as bitwise XOR and because each value is its own inversion, addition and substraction are equal operations.

Multiplication

Multiplication is defined as multiplication of both polynomials modulo irreducible polynomial of degree eight. For AES the irreducible polynom is defined as

$$m(x) = x^8 + x^4 + x^3 + x + 1 \tag{2.1}$$

Multiplication by x

Multiplication of binary polynomial by polynomial x results in polynomial of higher degree therefore the result must be reduced modulo $m(x)$. Following equation is the binary polynomial multiplied by polynomial x.

$$b_7x^8 + b_6x^7 + b_5x^6 + b_4x^5 + b_3x^4 + b_2x^3 + b_1x^2 + b_0x \tag{2.2}$$

If $b_7 = 1$ the result must be XORed with the polynomial $m(x)$. This operation can be accomplished as bitwise left shift and XOR with $1B$.

2.5.2 Algorithm Description

The AES is block cipher, therefore both encryption and decryption processes are performed on a matrix of 4x4 bytes called *state*. Even though state has fixed block size 128-bit, supported key sizes are 128-bit, 196-bit and 256-bit.

Encryption process as described in pseudocode 1 has 4 operations perfomed on each state of the data in specific number of cycles which varies from key length.

- 10 cycles for 128-bit key
- 12 cycles for 196-bit key
- 14 cycles for 256-bit key

Algorithm 1 AES encryption

Cipher(State, Key)
state $\leftarrow$ *AddRoundKey*(State, Key[0])

for $i \leftarrow (1..n-1)$ **do**
    state $\leftarrow$ *SubBytes*(state)
    state $\leftarrow$ *ShiftRows*(state)
    state $\leftarrow$ *MixColumns*(state)
    state $\leftarrow$ *AddRoundKey*(state, Key[i])
end for

state $\leftarrow$ *SubBytes*(state)
state $\leftarrow$ *ShiftRows*(state)
state $\leftarrow$ *AddRoundKey*(state, Key[n])

return state

Key Expansion

Round keys are derived from cipher key through process called *key expansion*. For the ciphering and deciphering purposes, the round keys could be thought as array of 4x4 8-bit values, which length is 10, 12 or 14 according to the used key size. The first matrix is copy of first 128 bits of cipher key. The following round keys are always calculated from the previous key and *rcon* array as explained in the algorithm 2.

Algorithm 2 Key Expansion

ExpandRoundKey(Key, size)
rk[0] $\leftarrow$ Key[0]

for $i \leftarrow (1..size)$ **do**
    k.col(0) $\leftarrow$ Key[$i-1$].col(3).rotate(1).map(sbox $\oplus$ Key[$i-1$].col(0)) $\oplus$ rcon
    for $j \leftarrow (1..3)$ **do**
        k.col(j) $\leftarrow$ Key[i-1].col(j) $\oplus$ k.col($j-1$)
    end for
    rk[i] $\leftarrow$ k
end for

return rk

Ciphering Process

`AddRoundKey` is XOR operation on the state with specific round key. Round key is extracted from the cipher key in *ExpandRoundKey*. Since this operation uses XOR, it is its own inverse form as well.

s_{00}	s_{01}	s_{02}	s_{03}	$\oplus$	k_{00}	k_{01}	k_{02}	k_{03}	$=$	a_{00}	a_{01}	a_{02}	a_{03}
s_{10}	s_{11}	s_{12}	s_{13}		k_{11}	k_{12}	k_{13}	k_{10}		a_{11}	a_{12}	a_{13}	a_{10}
s_{20}	s_{21}	s_{22}	s_{23}		k_{22}	k_{23}	k_{20}	k_{21}		a_{22}	a_{23}	a_{20}	a_{21}
s_{30}	s_{31}	s_{32}	s_{33}		k_{33}	k_{30}	k_{31}	k_{32}		a_{33}	a_{30}	a_{31}	a_{32}

Table 2.1: AddRoundKey on state s with key k where $a_{ij} = s_{ij} \oplus k_{ij}$.

`ShiftRows` is performed on each row of the state matrix. The first row is not shifted, second row is shifted by one byte to the left, third row is shifted by two bytes to the left and fourth row is shifted by three bytes to the left. Inverted ShiftRows for decryption is simply reversion.

a_{00}	a_{01}	a_{02}	a_{03}	$\longrightarrow$	a_{00}	a_{01}	a_{02}	a_{03}
a_{10}	a_{11}	a_{12}	a_{13}		a_{11}	a_{12}	a_{13}	a_{10}
a_{20}	a_{21}	a_{22}	a_{23}		a_{22}	a_{23}	a_{20}	a_{21}
a_{30}	a_{31}	a_{32}	a_{33}		a_{33}	a_{30}	a_{31}	a_{32}

Table 2.2: State on the right is the first state after ShiftRows is performed.

`MixColumns` together with ShiftRows provides diffusion in the AES algorithm. During this operation each column of the state is multiplied in Galois field 2^8 by matrix 2.3.

$$\begin{pmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{pmatrix} \tag{2.3}$$

As a result of this multiplication, each column $[s_{0c}, s_{1c}, s_{2c}, s_{3c}]$ is replaced by the column $[a_{0c}, a_{1c}, a_{2c}, a_{3c}]$ which could be calculated:

$$\begin{aligned} a_{0c} &= 2 \cdot s_{0c} \oplus 3 \cdot s_{3c} \oplus s_{2c} \oplus s_{1c} \\ a_{1c} &= s_{1c} \oplus 2 \cdot s_{0c} \oplus 3 \cdot s_{3c} \oplus s_{2c} \\ a_{2c} &= s_{2c} \oplus s_{1c} \oplus 2 \cdot s_{0c} \oplus 3 \cdot s_{3c} \\ a_{3c} &= 3 \cdot s_{3c} \oplus 2 \cdot s_{2c} \oplus s_{1c} \oplus s_{0c} \end{aligned} \tag{2.4}$$

`SubBytes` is non-linear transformation of the input *state*. Each byte in the state matrix is replaced with byte from substitution array of 256 8-bit values called S-box. The S-box A for encryption is generated by determining the multiplicative inverse for a given number in $GF(2^8)$ Rijndael's finite field and then affine transformation. The S-box A for decryption uses the same matrix but has first applied addine transformation and then the multiplicative inverse. For implementation purposes both S-boxes are precomputed.

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} \oplus \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} \tag{2.5}$$

In this transformation $[x_0, .., x_7]$ is the multiplicative inverse as vector, and $\oplus$ is XOR operation.

2.5.3 Block Cipher Modes

During encyption the same key is applied repeatedly on the uniform lenght blocks of data to whose the message is separated into. Large amount of ciphered data with the same encryption key might present security threat unless the ciphering algorithm provides form of randomization the output value. Such procedure might be achieved by additional input value.

There are many variations on block cipher to provide this confidentiality[16], for AES algorithm the most often used are *counter mode* and *f8-mode*. Both algorithms keep standard high level of confusion of the AES algorithm and provides necessary diffusion[4].

Both algorithms share some similar terminology and acronyms:

- IV – initial value used for encrypting the first block
- C_i – ciphertext block number i
- P_i – plaintext block number i
- E_K – encryption function
- D_K – decryption function

Counter Mode

The counter mode (CTR) turns AES block algorithm into stream cipher with possibility for parallel computations[27]. It is possible to decrypt the cipher text even with loss of number of blocks because the encrypted blocks are not dependent on the previous blocks. Instead the additional diffusion value are achieved by specific counter.

Equation 2.6 describes computation of counter value, equation 2.7 describes ciphering the counter value, equation 2.8 is encryption process – XOR operation of plaintext with encrypted counter value which produces ciphered text and equation 2.9 is decryption process.

$$CTR_i = (IV + i - 1) \mod 2^B \tag{2.6}$$
$$H_i = E_K(CTR_i, key) \tag{2.7}$$
$$C_i = P_i \oplus H_i \tag{2.8}$$
$$P_i = C_i \oplus H_i \tag{2.9}$$

[4] *Confusion* and *diffusion* are basic two properties of secure cipher introduced by Claude Shannon[13].

The last block of the plaintext doesn't have to be padded[5], it is common to use only the most significant bits of ciphered counter to be XORed with plaintext in cipher algorithm (and in similar way for deciphering).

F8-mode

The f8-mode is a variant of commonly known Output Feedback Mode (OFB) [16] with more elaborate initialization and feedback function [12]. The first output block O_1 is computed from IV, then it is XORed with plaintext to produce the first ciphertext block. The output block from previous step O_{j-1} is used to compute the current output block O_j which is always XORed with current plaintext in encryption algorithm.

The equation 2.10 describes the improved initializing function where m is the mask. The equations 2.11 and 2.12 describes computation of value, which is used for ciphering algorithm to produce output values in equation 2.13. Equation 2.14 describes ciphering and equation 2.15 describes deciphering.

$$IV' = E_K(IV, key \oplus m) \tag{2.10}$$
$$I_1 = IV' \tag{2.11}$$
$$I_j = O_{j-1} \oplus IV' \oplus j \tag{2.12}$$
$$O_j = E_K(I_j, key) \tag{2.13}$$
$$C_j = P_j \oplus O_j \tag{2.14}$$
$$P_j = C_j \oplus O_j \tag{2.15}$$

[5] Padding can be used for the plaintext that is not alligned to the multiplies of the block.

Chapter 3

General Purpose GPU

This chapter describes the basic ideas and techniques behind GPU parallel programming model and architecture. Following text will focus on possibilities of efective implementation for GPGPU and integrated GPU in modern CPU using OpenCL framework, brief description of selected principles and development of parallel applications.

Parallel machines have impressive performance to cost ratio compared to the common sequential machines[15], but bring well known problems for software development such as run-time resource allocation and resource sharing. Mapping parallel program to multiprocessor machine is complex problem that needs to decide about task allocation, scheduling of processes, communication patterns and much more.

While current CPUs are powerfull and sofisticated chips, their design must be focused on wide variety of tasks, therefore vast majority of resources might not be as fully utilized as could have been. The GPU chips provide much better theoretical performance for certain tasks for smaller price[24]. Interest among developers has grown in using the power GPUs provide for other tasks than graphics pipeline.

In order to achieve improvement in certain algorithm it is neccessary to analyze the procedures and find possibilities for parallelization and take under consideration that usage of additional processing unit brings computational overhead. The characteristics of such application are[23]:

- Utilization of data-parallelism – many non-graphical problems might be separated into fractional procedures and computed separately, such as matrix calculations individially for each cell.
- Large portion of computation – GPU porcessors are optimized for computations over handling conditional evaluations.
- Throughput over Latency – computations on GPU are designed for large overall throughput of entire data rather than short response time of each individual operation.

The current trend in development shows that parallel computations either in the form of GPU computations and APU[1] are worth examination and research. SIMD[2] has already proven it's value on improving performance with parallelization of various algorithms[4, 2].

[1] Accelerated Processing Unit – in this context it means CPU with GPGPU chip.

[2] Single Instruction Multiple Data – multiple processing elements that perform the same operation on multiple data points simultaneously[17].

APU

Usual solutions with graphics card can have high power consumption. The modern trend and need of transportable forced development to reduce negative effects of GPUs while keeping as much of latest visual experience as possible[14]. Both solutions utilize a portion of computer's system RAM memory.

APU is Accelerated Processing Unit that is designed to accelerate certain type of computations outside of CPU in single chip. It could include GPU, FPGA or similar specialized processing unit. Among the best known there are Intel HD Graphics[3], AMD Fusion[1] and NVIDIA Project Denver[5].

3.1 OpenCL

Development for parallel computation brought need for infrastructure. OpenCL is an industry standard framework for programming heterogenous systems composed of a combination of CPUs, GPUs, DSP and other processing units[21]. With OpenCL it is possible to write a software that will run on wide variety of platforms from cell phones or computers to massive supercomputers.

The *OpenCL programming language* has syntax based on the language C with few additions and limitations arising from the design and architecture of heterogenous platforms. Among most important limitations it omits the use of recursion, function pointers and header files. On the other hand, the language is extended to the use of parallelism with build in types and synchronization. Also it defines many functions and four new keywords as memory region qualifiers: `__global`, `__local`, `__constant` and `__private`.

For further reading of the text and better comprehensibility, there are listed neccessary words from OpenCL terminology[21].

- Context – contains one or more devices used for kernel execution and are used for managing command queues, memory and program.
- Kernel – function written in OpenCL programming language that is executed on OpenCL device.
- Work-item – instance of executing kernel.
- Work-group – organisation of work-items.
- Command-queue – interaction between the host and OpenCL device through commands posted by the host and provides synchronization methods for the execution of the commands.

OpenCL platform includes single *host* that comunicates with the user and the OpenCL program. The host is connected to one or more OpenCL *devices* where the *kernels* are executed. *Kernel* could be considered as the entry point between host and GPU. In order to achieve parallelism, the device consists of many *work-items* whose execute multiple instances of kernel at the same time. The work-items are organized in integer indexed orthogonal grid where the unique index of a work-item is called gloabl ID. The identification of work-item is possible through combination of its local ID inside a specific *work-group* and the work-group global ID.

3.1.1 Platform Model

The OpenCL provides a high-level abstraction model representing any heterogenous platform. The *host* is a bridge between parallel computations on one or more devices and interaction with external environment. *Device* could be CPU, GPU, DSP or any other processing unit supporting OpenCL and consists of compute units which are further divided into processing elements. *Processing element* is abstraction of a work-item and *compute unit* is in similar way representation of work-group.

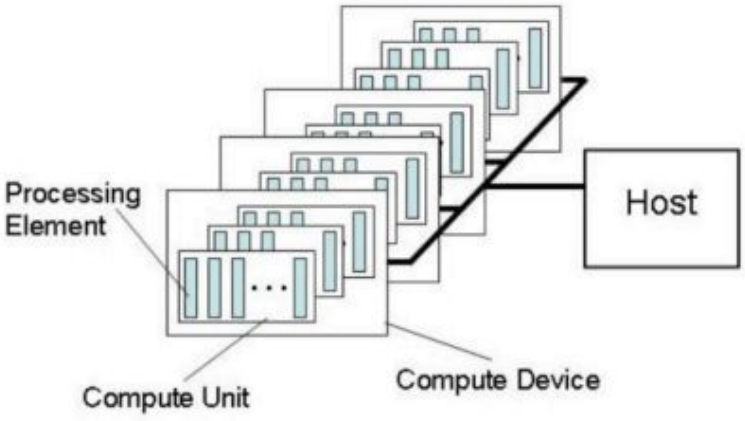


Figure 3.1: OpenCL platform model with one host and multiple devices[21].

3.1.2 Execution Model

The OpenCL software executes on two levels

- **Host code** – the OpenCL doesn't define any restrictions about the host part of the application, it defines only the interaction between host and devices. It consists of selection and initialization of the context – selected platform and devices.
- **Device code** – written in OpenCL programming language in the form of short functions, kernels, that usually transform an input array through series of processes into output array. It is compiled via OpenCL compiler and executed on the device's work-items.

The host program takes care of synchronization and plans the execution of each kernel on the devices. Each instance of kernel runs in separate work-item and the work-items within each work-group execute concurrently.

3.1.3 Memory Model

OpenCL defines two types of memory objects. The *buffer object* is versatile type that could be used for representation of any data type available in C or OpenCL language. The *image object* is restricted to containing pictures only and is optimized for the specific needs of image processing.

OpenCL uses a hierarchicaly structured memory. The types differ in access time, availability and types of usage[21]:

- **Private Memory** – each work-item has it's own private memory which could be thought of as analogy to CPU's registers. It is the fastest type of memory used in OpenCL.
- **Local Memory** – designed for sharing data between work-items who belong to the same work group. It is used to reduce the number of accesses to the global memory. Local memory is slower than private memory but faster than global memory. The programmer is denied both direct access and control over local memory. The analogy could be the cache in CPU.
- **Global Memory** – shared among all work-items in the same context.
- **Host Memory** – memory visible only for the host, OpenCL only defines how the host interacts with OpenCL objects and constructs.

There could be another type of memory in graphic cards that OpenCL doesn't define

- **PCI Memory** – type of memory that could be used by the program and GPU, part of host memory. It is slower than global memory.

Chapter 4

Design

The aim of the implementation is to determine whether the parallel processing of SRTP could increase the limitations on modern VoIP softgates. The development of sophisticated softgate requires elaborate engineering and implementation of various communication protocols that would overshadow the effort in parallel processing. Therefore only narrow selection of well know communication protocols has been implemented. For VoIP telephony, registration and maintenance of users serves *SIP* protocol, for the media transmission description and session description *SDP* protocol, and for secure media transport *SRTP* with *ZRTP*. There is also implementation of *LCP* stack[1].

4.1 Design Patterns

More complex the application is the higher level of considerate design it requires. There are plenty of already well tested design patterns from which the implemetation could be based on and as the field of VoIP communication has been known for decent amount of time, there are currently couple of advised design patterns, from which the particular implementation for this thesis stands on four – mediator pattern, singleton pattern, factory method pattern [18] and protocol stack pattern [6]. None of these design patterns could be thought as contribution of the thesis as they all belong to common public knowledge and their examination was not the main topic of the research. However, their explanation is provided in order to make the rest of the chapter more comprehendable.

4.1.1 Mediator Pattern

In object oriented design the common problem be the large number of classes and their mutual interaction. One of the possible solutions for the latter can be behavorial pattern called mediator, which is named after the way it alters the running behavior. The pattern consits of following participants:

- **Mediator** – defines an interface for communicating with colleague objects
- **ConcreteMediator** – implements cooperative behavior by coordinating colleague objects, it knows and maintains its colleagues

[1] Light-weight Control Protocol – communication protocol for Siemens prototype VoIP phone.

- **ConcreteColleague** – each colleague knows its mediator object and it communicates with its mediator whenever it would have otherwise communicated with another colleague

The mediator object communicates with multiple colleague object through shared interface. http://en.wikipedia.org/wiki/Mediator_pattern

4.1.2 Singleton Pattern

http://en.wikipedia.org/wiki/Singleton_pattern

4.1.3 Factory Method Pattern

http://en.wikipedia.org/wiki/Factory_method_pattern

4.1.4 Protocol Stack Pattern

http://www.eventhelix.com/realtimemantra/PatternCatalog/protocol_stack.htm#.UYEfRbXQovo

4.2

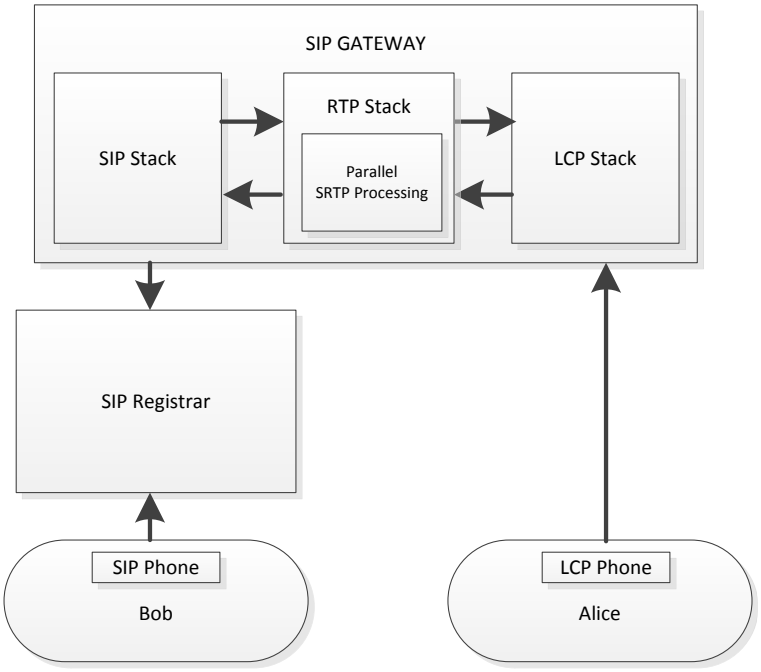


Figure 4.1: SIP Gateway with two VoIP telephones accessible, LCP phone directly connected and SIP phone through SIP Registrar.

Multiple SIP or LCP telephones are connected to the SIP Gateway whose appear as users to the SIP registrar[2]. The SIP gateway in this scenario works as bridging point between the SIP telephones and SIP registrar, LCP phones and SIP registrar or LCP phones directly.

[2] Used registrars were Asterisk and Siemens HiPath 8000

The modules of SIP gateway are implemented in different programming languages and each serve specified purpose.

- **Gateway core** – implemented in Java, provides communication between each module and encapsulates basic functionality of a Gateway.
- **SIP Stack** – Java API for SIP stack called JAIN SIP[22], implemented basic SIP features.
- **LCP Stack** – implemented in Java fully covering LCP protocol v1.0.
- **RTP Stack** – for non-direct connections where the telephones couldn't agree on communication chanel for the session, RTP stack implemented in C/C++ and OpenCL provides neccessary briging point.

4.3 SIP Gateway

Mesurement of utilization of computational resources during execution of ciphering algorithm does provide correct and exact results, however in real deployment the efectivity could be negatively affected by the other processes running on the softgate. SIP Gateway is a collection of programs and utilites whose together implement a server for lighweight LCP phones and suplement a SIP functionality for each phone to be able to connect to an actual SIP registrar.

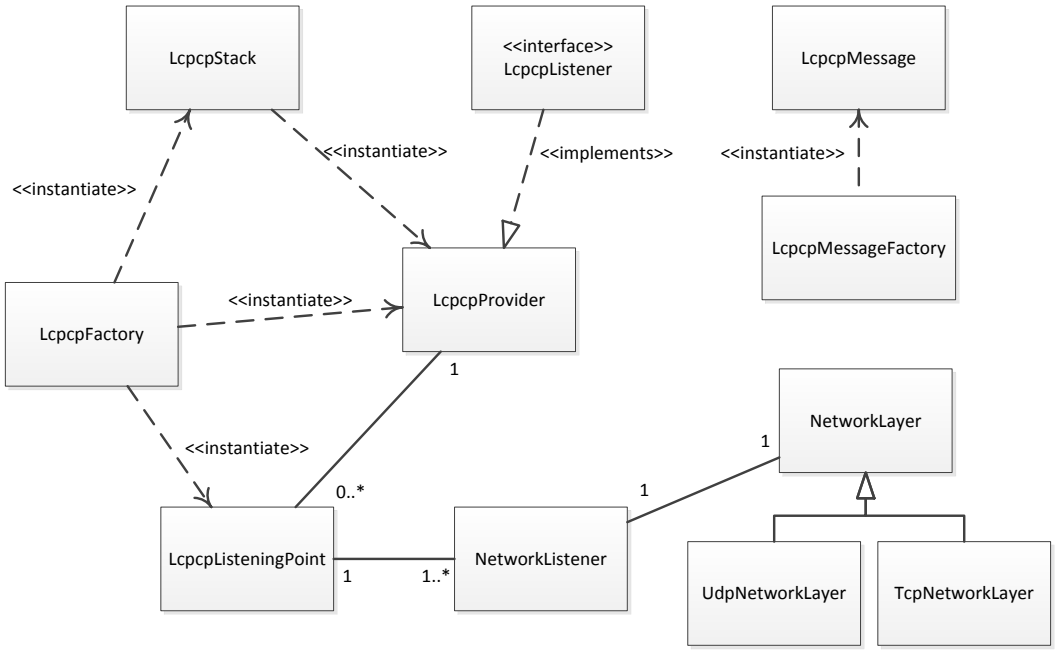


Figure 4.2: Architecture of LCP stack, design was inspired by the JAIN-SIP api[22].

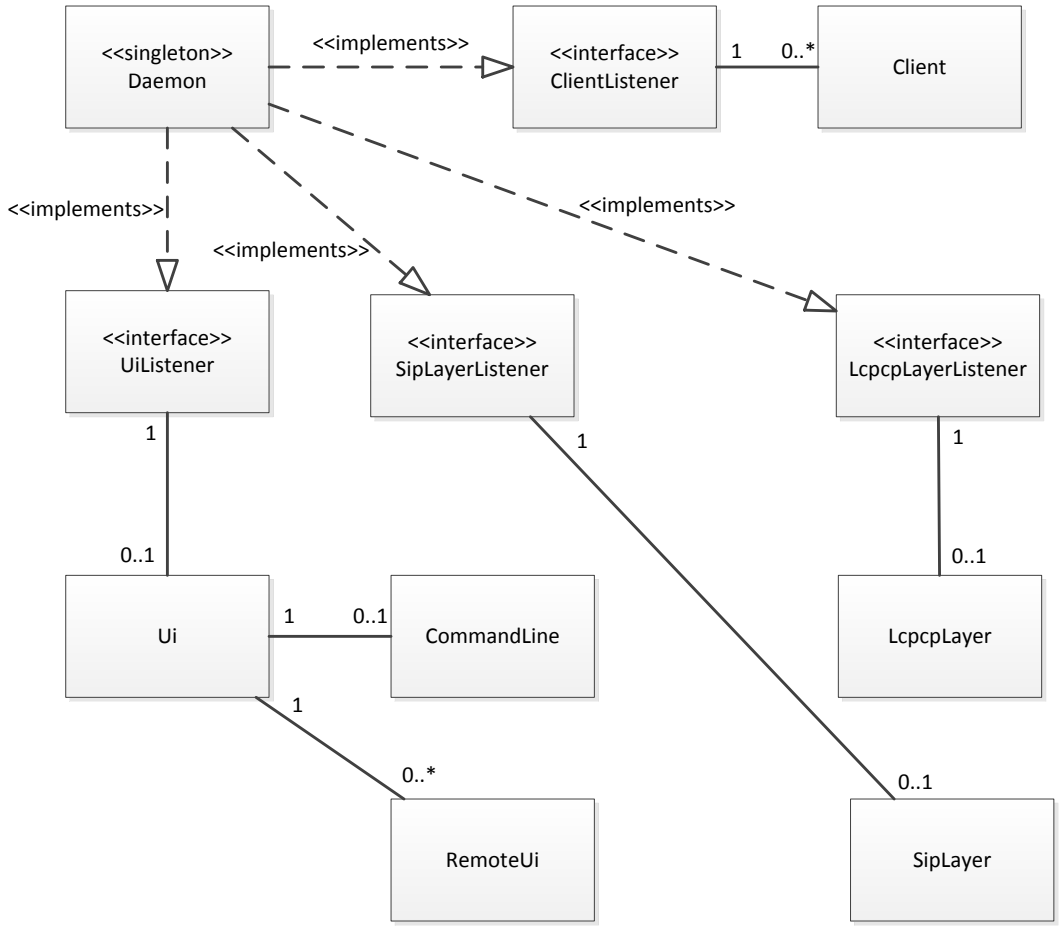


Figure 4.3: Architecture of SIP Gateway with singleton design pattern. Consists of *Daemon* class, multiple LCP phones connected via LCP stack and represented as instances of *Client* class and multiple user interfaces for control over the gateway.

The core application should offer simple management via command line for both development and tracing of the flowing communication and basic functionality for communication and session management.

The core class of the gateway is `Daemon`, which controls the flow of data inside the application and provides interfaces to communicate with external applications. `SIP Stack` provides the interface to communicate with SIP Registrar. The single SIP Stack is shared for all users, implicitly runs on well known port for SIP communication 5060, which could be explicitly changed if neccessary. `LCP stack` visualized on the figure 4.2 was designed to reflect the elaborate design used in JAIN SIP[22]. While SIP is much richer protocol than LCP, the design of the stack was extremely shortened but the basic structure of elementary components and their interaction remained the same. LCP stack runs implicitly on the recommended port 4066, but as well as SIP stack port, the port could be variable if needed. Each SIP/LCP client is instance of `Client` class, and universal interface for remote communication and administration shall be provided as well.

4.4 SRTP Stack

The essential point of implementation improvement lies in design of SRTP stack as it has been mentioned in previous text that it consumes majority of resources of the gateway during indirect media sessions. Proper implementation must not lack following properties:

- **encryption module** – implementation of AES-128b cipher as defined in RFC-3711 [12] in at least CRT mode that provides protection of transfered data with different keys for each endpoint in all concurrent sessions.
- **input and output buffers** – in order to avoid exhaustive allocation and deallocation of structures for input and output packets, the data storage should be implemented as thread safe pool of buffers with sufficient size and both, synchronization techniques and memory override protection.
- **transcoding module** – due to vaious reasons, endpoints may not be able of negotiate the same media compressing codec. The SRTP stack should allow the transcoding and then encapsulate the process without unneccessary additional demands for the gateway.
- **integration interface** – most of the procedures implemented in SRTP stack should be encapsulated to minimize overloading data transferes with the gateway providing only essential and minimal interface with callback features to simplify and unify the integration process.

An advanced techniques like **jitter buffer** may improve overal quality of VoIP communication, however, each end device capable of such communication must implement these techniques as well, therefore, it may render itself redundant and generating minimal, but still additional latency.

4.4.1 SRTP Processing

Advantage of usage AES in CM is that it allows out-of-order processing. Because majority of RTP implementations are build on UDP transport layer, which is simple model with minimal protocol mechanisms, neither order nor delivery of the packets are guaranteed in exchange for smaller average delay and smaller traffic.

The exact size of payload in SRTP packet can differ widely according to the used codec, its bit rate, and sampling frequency. The selection of used voice codecs, their sampling periods and payload size are mentioned in table 4.1.

Codec and Bit Rate	Payload Size	Sampling Period	Packets Per Second
G.711 – 64 Kbps	160 bytes	20 ms	50
G.729 – 8 Kbps	20 bytes	20 ms	50
G.726 – 32 Kbps	80 bytes	20 ms	50
G.726 – 24 Kbps	60 bytes	20 ms	50
G.728 – 16 Kbps	60 bytes	30 ms	33

Table 4.1: Selected codecs and payload information.

Fixed block size of AES is 16 bytes, which means that one or more states could be mapped to the packet using any of the mentioned common codecs. Parallelization of the encryption process could be performed either on a single state, where value during every method of the AES of each cell of the state is computed separately, therefore a work-item can be mapped on computing for each cell. Theoretical common hardware should be capable of utilizing 16 work-items in a single work-group which is the maximal number of needed by this design.

Another possible approach for codecs with larger payload size, such as G.711, could be to map multiple states for the parallel execution of entire packet, which for the particular codec would require significantly more computational units.

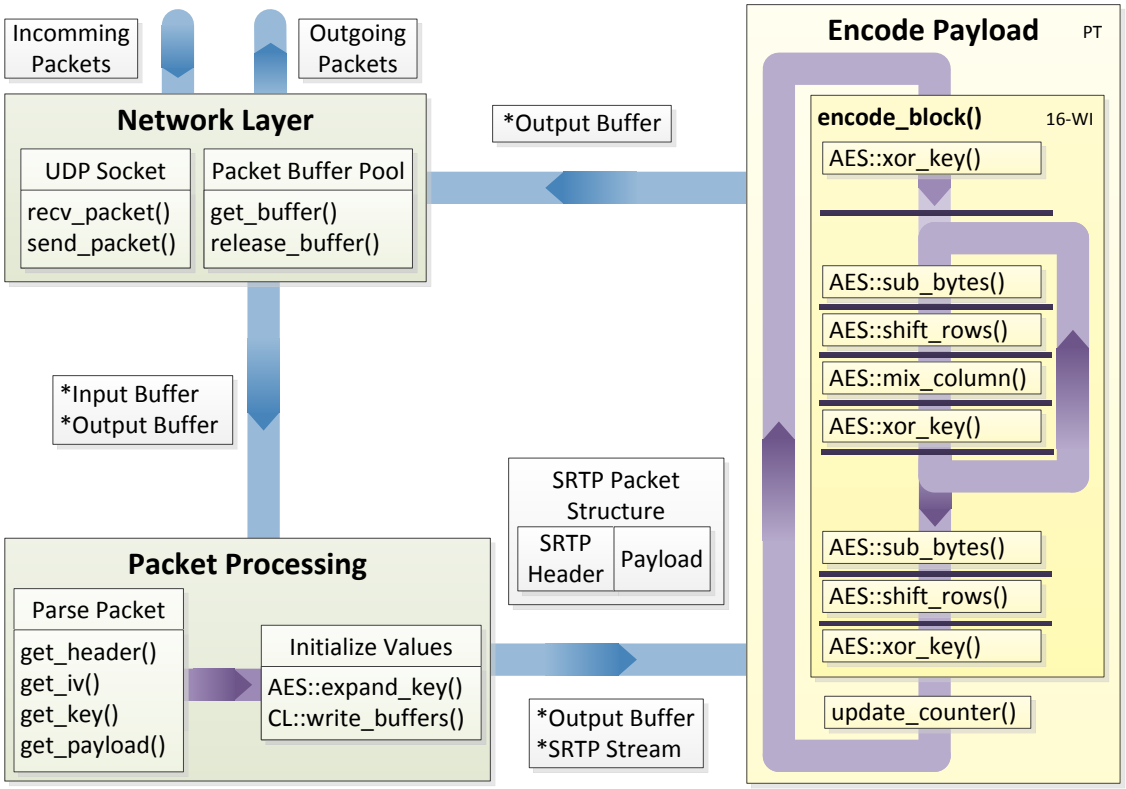


Figure 4.4: SRTP Processing Scheme.

The SRTP processing scheme from figure 4.4 visualizes the ideas behind the design of SRTP stack and encapsulates implementation details for easier explanation from the multi-threaded application design point of view. The entire stack runs in three separate threads which shall minimize the delay caused by waiting on modules with varying time of execution per packet.

- **Network thread** – the incomming and outgoing data are captured via two sockets, for RTP and RTCP. This thread includes a pool of buffers for the storage of packets and another the processed data.
- **Daemon thread** – the implementation of the entire stack is based on the daemon design pattern.

- textbfPacket Processing Thread –

The threads can be used for bijective mappings to the SRTP layers as shown in the scheme.

- Network Layer
- Packet Processing Layer
- Payload Encoding

4.4.2 Serial Processing

For better understanding of improvement this thesis is provided with reference serial implementation which design will be analyzed as well.

4.4.3 Massive Parallel Processing

4.4.4 Persistent Thread Processing

Chapter 5

Implementation

Chapter 6

Results

Chapter 7

Conclusion

There are lot of possibilities for optimization of SRTP processing. Selected approach focuses on methods of parallelization of encryption and decryption processes of default AES cipher, which offers large potential thanks to recent development in the field of parallel computational units.

Proposed architectures and designs are currently far from being complete. The most effort was invested in correct analysis and understanding of basic principles of further implemented algorithms and elementary knowledge of parallel programming paradigm focused on usage of OpenCL framework for general-purpose computations on graphical processing unit. Modern GPU concentrate large amount of computational power, which could be to a certain extent utilized, if the algorithm is correctly mapped for parallel execution. That brings unusual complications in design whose must be carefully considered.

Another important milestone is definition of integration of RTP stack with SRTP processing into implemented SIP Gateway and their mutual interaction. The SRTP processing is only a fraction of overall load on the gateway and if measured separately while producing exact experimental results may not be equal to the actual results on deployed machine experiencing real traffic.

For the further development number of issues must be taken for notice. For instance the delay generated by the processing of separate SRTP packets should be reliably masked and interpolated across the SRTP stream to reduce possible jitter. On the other hand stands the actual delay of incomming packet, since after certain absolute value the conversation quality becomes unbearable.

Nevertheless, partial value of this thesis lies in the understanding of current technologies for future potential direction of development and exploration of new options in the field communication infrastructure.

Bibliography

[1] AMD Accelerated Processing Units [online]. http://www.amd.com/us/products/technologies/apu/Pages/apu.aspx. Published 2011-6-8, accessed 2012-12-28.

[2] AMD and Leading Software Vendors Continue to Expand Offerings Optimized for OpenCL Standard [online]. http://www.amd.com/us/press-releases/Pages/offerings-optimized-for-opencl-2011jun08.aspx. Published 2011-6-8, accessed 2012-12-28.

[3] Intel HD Graphics [online]. www.intel.com/content/www/us/en/architecture-and-technology/hd-graphics/hd-graphics-developer.html. Accessed 2012-12-28.

[4] NVIDIA OpenCL SDK Code Samples [online]. http://mlso.hao.ucar.edu/hao/acos/sw/cuda-sdk/OpenCL/Samples.html. Published 2012-10-1, accessed 2012-12-28.

[5] Project Denver [online]. http://blogs.nvidia.com/2011/01/project-denver-processor-to-usher-in-new-era-of-computing/. Published 2011-1-5, accessed 2012-12-28.

[6] Protocol Stack Design Pattern [online]. http://www.eventhelix.com/realtimemantra/PatternCatalog/protocol_stack.htm#.UYFRqbXQp-p. Accessed 2013-5-1.

[7] Securing Internet Telephony Media with SRTP and SDP [online]. www.cisco.com/web/about/security/intelligence/securing-voip.html. Accessed 2012-1-2.

[8] Specification for the Advanced Encryption Standard (AES). Federal Information Processing Standards Publication 197, 2001.

[9] T. Adomkusv and E. Kalvaitis. Investigation of VoIP Quality of Service using SRTP Protocol. pages 195–209, 2008.

[10] A. L. Alexander, A. L. Wijesinha, and R. Karne. An evaluation of secure real-time transport protocol (srtp) performance for voip. In *Network and System Security, 2009. NSS '09. Third International Conference on*, pages 95 –101, oct. 2009.

[11] F. Andreasen, M. Baugher, and D. Wing. Session Description Protocol (SDP) Security Descriptions for Media Streams. (RFC 4568), 2006.

[12] M. Baugher, D. McGrew, M. Naslund, E. Carrara, and K. Norrman. The Secure Real-time Transport Protocol (SRTP). (RFC 3711), 2004.

[13] C. E. Shannon. Communication Theory of Secrecy Systems. vol.28-4:656 – 715, 1949.

[14] M. Daga, A.M. Aji, and Wu chun Feng. On the Efficacy of a Fused CPU+GPU Processor (or APU) for Parallel Computing. In *Application Accelerators in High-Performance Computing (SAAHPC), 2011 Symposium on*, pages 141 –149, July 2011.

[15] J. Darlington, M. Ghanem, and H. W. To. Structured Parallel Programming. In *In Programming Models for Massively Parallel Computers*, pages 160–169. IEEE Computer Society Press, 1993.

[16] M. Dworkin. Recommendation for Block Cipher Modes of Operation. Federal Information Processing Standards Publication 800-38A, 2001.

[17] M. J. Flynn. Some computer organizations and their effectiveness. *IEEE Trans. Comput.*, 21(9):948–960, September 1972.

[18] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

[19] P. Handley, V. Jacobson, and C. Perkins. SDP: Session Description Protocol. (RFC 4566), 2006.

[20] S. Kumar, C. Paar, J. Pelzl, G. Pfeiffer, and M. Schimmler. Breaking ciphers with copacobana – a cost-optimized parallel code breaker. In *Workshop on Cryptographic Hardware and Embedded Systems – Ches 2006, Yokohama*, pages 101–118. Springer Verlag, 2006.

[21] A. Munshi, B.R. Gaster, T.G. Mattson, J. Fung, and D. Ginsburg. *OpenCL Programming Guide*. OpenGL Series. Prentice Hall, 2011.

[22] P. O'Doherty and M. Ranganathan. JAIN SIP Tutorial - Serving the Developer Community. Technical report.

[23] J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone, and J. C. Phillips. GPU Computing. *Proceedings of the IEEE*, 96(5):879–899, May 2008.

[24] J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. KrĂźger, A. Lefohn, and T. J. Purcell. A Survey of General-Purpose Computation on Graphics Hardware. *Computer Graphics Forum*, 26(1):80–113, 2007.

[25] C. Perkins. *RTP: Audio and Video for the Internet*. Addison-Wesley, June 2003.

[26] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. SIP: Session Initiation Protocol. (RFC 3261), 2002.

[27] N. P. Tran, M. Lee, S. Hong, and S. J. Lee. Parallel Execution of AES-CTR Algorithm Using Extended Block Size. In *Computational Science and Engineering (CSE), 2011 IEEE 14th International Conference on*, pages 191 –198, August 2011.

[28] P. Zimmermann, A. Johnston, and J. Callas. ZRTP: Media Path Key Agreement for Unicast Secure RTP. (RFC 6189), 2011.

Appendix A

AES Properties

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
00	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
10	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
20	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
30	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
40	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
50	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
60	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
70	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
80	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
90	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a0	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b0	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c0	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d0	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e0	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f0	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Table A.1: S-box for SubBytes transformation in hexadecimal notation.

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
00	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
10	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
20	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
30	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
40	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
50	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
60	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
70	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
80	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
90	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a0	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b0	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c0	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d0	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e0	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f0	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Table A.2: Inverse S-box for SubBytes transformation in hexadecimal notation.

Algorithm 3 AES decryption

Decipher(State, Key)
state ← $AddRoundKey$(State, Key[n])
state ← $ShiftRows$(state)
state ← $SubBytes$(state)

for $i \leftarrow (n-1..1)$ **do**
    state ← $AddRoundKey$(state, Key[i])
    state ← $MixColumns$(state)
    state ← $ShiftRows$(state)
    state ← $SubBytes$(state)
end for

state ← $AddRoundKey$(state, Key[0])

return state