USING GPU FOR QUERY OF EMAIL SPAM DETECTION SYSTEMS AND IDS

Aleksandar Sokolovski [*], *Saso Gelev*[**]

Abstract: The scope of this research paper is one very important aspects nowadays, the security and management of one of the most important services the email and all of the alike online services today. This paper attempts to investigate the possible benefits of using standard signature-driven spam detection logic in combination with algorithm for network intrusion detection system (NIDS). The primary objective is to verify that proposed solution (standard signature-driven spam detection logic and NIDS) will be an effective strategy for dealing with email spam detection. The main aim is to determine best possible integration of standard signature-driven spam detection logic in combination with algorithm for NIDS, for creating more effective solution for dealing with email spam and priority mail compared to the previous solutions available. This will be achieved by testing the effectiveness of the solution compared to other solutions until today, by using network simulators NS-3 and other programming tools.

1. INTRODUCTION

The email was one of the most important communication tools and it also is one of the most important communication tools nowadays. The email is also one of the most used services on the World Wide Web. According [1] to here are the general data about email:

- In 2012 144 billions emails were send per day worldwide
- In 2012 there were 2.2 billion email users worldwide.
- In 2012 4.3 billion email clients for sending email worldwide
- In 2012 and 69% of worldwide email was spam.
- In 2011 also 71% of email was email spam.

The importance of fast email delivery has change over the year and from being a reliable and fast delivery system, it has become an immediate delivery system. Many enterprises, companies and organization use various techniques and methods for dealing with spam:

- DNS whitelist and blacklists
- Load Sharing: Load is shared among several servers using Layer 4 switches or DNS round-robin.

[*] Aleksandar Sokolovski, Neotel – Macedonia, R&D Department, Skopje, Macedonia

[**] Saso Gelev, Faculty of Electrical engineering, University of Goce Delcev, Stip, Macedonia

- DNS Blacklisting: Identify whether the sender is going to send spam or not according to the sender's IP address with a help of DNS.
- Bayesian spam filtering
- Hybrid Filtering

Firewall is hardware or software based network security systems that monitors and controls inbound and outbound network traffic by analyzing the packets.

The first generation of firewalls only filter packets.

The second generation of firewalls are known as stafefull, they analyze packets like the first generation (Layer 3 – IP address) but the difference is they operate up to Layer 4 (Ports) of the ISO OSI network model.

The Third generation of firewall are known as application firewall and they work on Layer 7 (ISO OSI).

An intrusion detection system (IDS) monitors network traffic and monitors for suspicious activity and alerts the system or network administrator. In some cases the IDS may also respond to anomalous or malicious traffic by taking action such as blocking the user or source IP address from accessing the network IDS come in a variety of "flavors" and approach the goal of detecting suspicious traffic in different ways. There categories are: NIDS, HIDS, Signature Based, Anomaly Based. [1]

1.1. NIDS

Network Intrusion Detection Systems are placed at a strategic point or points within the network to monitor traffic to and from all devices on the network. Ideally you would scan all inbound and outbound traffic, however doing so might create a bottleneck that would impair the overall speed of the network.

1.2. HIDS

Host Intrusion Detection Systems are run on individual hosts or devices on the network. A HIDS monitors the inbound and outbound packets from the device only and will alert the user or administrator of suspicious activity is detected.

1.3. Signature Based

A signature based IDS will monitor packets on the network and compare them against a database of signatures or attributes from known malicious threats. This is similar to the way most antivirus software detects malware. The issue is that there will be a lag between a new threat being discovered in the wild and the signature for detecting that threat being applied to your IDS. During that lag time your IDS would be unable to detect the new threat.

1.4. Anomaly Based

An IDS which is anomaly based will monitor network traffic and compare it against an established baseline. The baseline will identify what is "normal" for that network- what sort of bandwidth is generally used, what protocols are used, and etc.

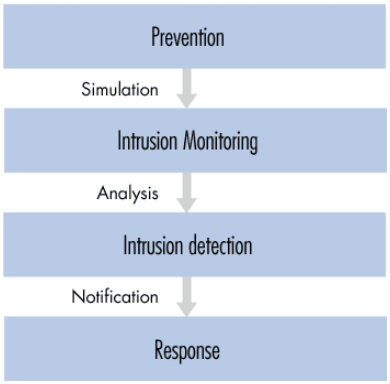

Fig. 1. Structure of an Intrusion Detection System

1.5. Snort

One of the most well-known and widely used intrusion detection systems is the open source, freely available Snort. It is available for a number of platforms and operating systems including both Linux and Windows.

Snort is a signature based IDS, lightweight and very easy to use, the code is 100 KB.

Snort architecture consists of 4 main parts: Packet Decoder, Pre-processing, Detection Engine, and Post-Process.

Many of the previously mentioned solutions for spam detection and IDS are great and in the beginning when the list is small and manageable work fast and efficient. When the lists becomes larger the processing becomes slower.

In Conclusion the main drawback in today's email spam solutions, priority mail delivery and IDS solutions is the query processing. In our paper we present solution for the problem using GPU (Graphical processing Units) more specifically CUDA programming. In part 2 and 3 of this paper we present the GPU and CUDA. In part 4 of this paper we explain the DARPA dataset. In part 5 and 6 of this paper we present our solution, and conclusion in part 7.

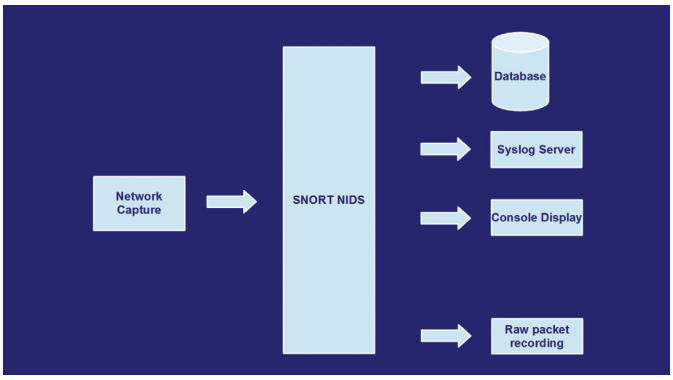

Fig. 2. SNORT NIDS

2. GPU COMPUTING

GPU (Graphical Processing Unit) computing or the usage of graphical processors for all-purpose computing, began less than 10 years ago. The research in the beginning only allow programming via graphics language, which made in un-flexible. The NVIDIA's CUDA platform according to [2], provides massively multithreaded general-purpose architecture with up to 128 processor cores and thousands of threads. The platform is programmable in the programming language C and capable of hundreds of billions of floating-point operations each second and supports running on all current and older versions of NVIDIA GPUs, this includes the HPC oriented Tesla product line and Kepler core GPUs. The NVIIDIA GPU's are sold in millions worldwide which makes them a very good platform for accelerating high-performance computing (HPC) Applications. GPU computing speeds up the real-worlds science and engineering codes up to 10 or 100 times (depending on the GPU's hardware performance). It's domains of usage range from Computer Technology and MRI, computational chemistry, astrophysics to gene sequencing. Here are few examples of CUDA usage: Friedrich-Alexander- Universidad Erlangen-Nurnberg works on biomedical imaging to accelerate CT reconstruction, working on CUDA using the FDK algorithm. University of Illinois works on accelerating advanced MRI reconstruction techniques using CUDA [3]. The last mentioned work focuses on reconstruction for non-Cartesian scan paths, which reduces image-space error from 45% for conventional reconstruction to 12% but has been considered computationally infeasible in practice. This approach uses NVIDIA Quadro FX5600 GPU and works 13-times faster than on Intel Core 2 Extreme quad-core CPU. This means that the reconstruction times is under 2 minutes for 128 volume. This important types of speedup can change science. According to [4] in medicine this types of speedups can make a significant change in clinical practice, for example an analysis in a lab can be done in hours or minutes instead of days. In biomedical imaging the processes also speedups. Another science field where GPU CUDA speeds up the work is astrophysics (mapping of stars, image processing from telescopes like Hubble and other similar tasks). In short, the potential to greatly accelerate computational techniques opens exciting avenues for biomedical imaging research, astrophysics and other fields.

3. CUDA PROGRAMMING MODEL

The key strength of the GPU is its extremely parallel nature. The CUDA programming model enables developers to use the parallelism by using programming code written in C/C++ programming language. The code will run in thousands or millions of parallel invocations (depending on the GPU) or threads. We will present a simple example of Matrix Addition. For adding two N x N matrices on the CPU in C/C++, we would use double nested loop like in the next code:

```
void addMatrix(float *a, float *b,
float *c, int N)
{
int i, j, index;
for (i = 0; i < N; i++) {
```

```
for (j = 0; j < N; j++) {
index = i + j * N;
c[index]=a[index] + b[index];
}
void main()
{
..... addMatrix(a, b, c, N);
}
```

Example no.1

The same example written in CUDA will be like this. We write one C function, called a kernel function, to calculate one element in the matrix and invokes as many threads to run that function as the matrix has elements. In each thread the kernel runs with a predefined structure called "threadldx" indicating which of the many threads is running (example no.2):

```
_global_ void addMatrix(float *a,float *b, float *c, int N)
{
int i= threadIdx.x;
int j= threadIdx.y;
int index= i + j * N;
c[index]= a[index] + b[index];
}
void main()
{
dim3 blocksize(N, N),
addMatrix<<<1, blocksize>>>(a, b, c, N);
}
```

Example no. 2

In example no.2 the "_global_" declaration specifier indicates a kernel function that will run on the GPU, the "<<N, N>> syntax indicates that the "addMatrix()" function will be invoked across a group of threads run in parallel, this is called a thread block. Thread block may be one, two or three dimensional, which provides a natural way to invoke computation across the elements in domains like the following: matrix, fields and vectors.

CUDA makes critical improvements to the core components of running kernel function across many parallel threads: hierarchical thread blocks, shared memory and barrier synchronization. Next we explain in details.

Thread block can contain up to 512 thread if it is a NVIDIA Tesla architecture GPU, but kernels are invoked on a grid which consists of many thread block that are scheduled independently. In every thread in every block in a grid executes the kernel and then exits. Additional code will be added (example no.3):

```
_global_ void addMatrix(float *a,float *b, float *c, int N)
{
int i=blockIdx.x*blockDim.x+threadIdx.x;
int j=blockIdx.y*blockDim.y+threadIdx.y;
int index = i + j * N;
if ( i < N && j < N)
c[index]= a[index] + b[index];
}
void main()
{
dim3 dimBlock (16, 16);
dim3 dimGrid (N/dimBlk.x, N/dimBlk.y);
addMatrix<<<dimGrid, dimBlock>>>(a, b, c,N);
}
```

Example no.3

In the example above the size of the thread block which was chosen randomly is 256 (16 times 16). The created grid has enough block to have one thread per matrix element, which is same as the example before. All the threads in a sector block run in parallel. Whether multiple thread block can run one after another (in serial mode) or they can run in parallel, it all is dependable on GPU.

In the matrix threads can run independently, no threads needs to know the elements beings accessed by other threads, but when threads need to cooperate (share results the results of computations or memory fetches) in order to be more efficient. CUDA provides shared memory for this purpose. In the shared memory, the kernels can store data (variables or arrays) that is visible to other thread block. Simple example is calculating the sum value of the elements within an array, this can be archived when thread in a block places the element into an array in shared memory (adding the next and the next and so on). All the threads in a block do the same in parallel. They are calculating sum value of elements by cooperating using the shared memory. The shared memory is small because it is on a chip (16K in 2010 Models of NVIIDA GPU), but it is extremely fast, therefore it is perfect for this type of operations like calculating sum value of array elements.

When threads are operating in parallel on same memory it is important to have a mechanism that ensures that one thread will not attempt to read a result before another thread has ended writing it or changing it. CUDA provides the __syncthreads() intrinsic function for this purpose. "_syncthreads ()"acts as a barrier at which all threads in the block must wait before any are allowed to proceed. [2]

4. THE ALGORITHM FOR THE PROPOSED IDS

Solutions for IDS are many, our solution is a hybrid version of an, network based / host based IDS and signature based and anomaly based IDS. Our system or the algorithm presented for managing the proposed ISP system on Fig. 3, is based upon HIN procedure, presented in the next section

4.1. Procedure for epidemic containment and control created by NIH (National Institute of Health)

Procedure for epidemic containment and control
Quarantine procedure (NIH POLICY MANUAL, 3043-1)

- Dislocating VIP persons from the quarantine zone
- Isolating the sick from the healthy patients
- Immunization of the healthy patients
- Creating Quarantine Zone
- Detecting "patient zero"
- Eliminating the threats

This procedure is used in medicine, and is proven as a very successful procedure in 2009, tested for the H1N1 virus containment in North Carolina, U.S.A. This is also known as procedure CDC H1N1.

Many methods for containment were tested in June the above mention procedure was implemented and the containment of the virus was 100% or 0 newly exposed patients.

4.2. UML Design of the IDS

In this section we will present the implementation of the ISP system, and the proposed algorithm that manages the modules of the ISP and their communication.

The entire ISP is not presented in details, only the parts / modules that will be same for the any implementation to any type of computer network, as a separate system of add-on to an existing one.

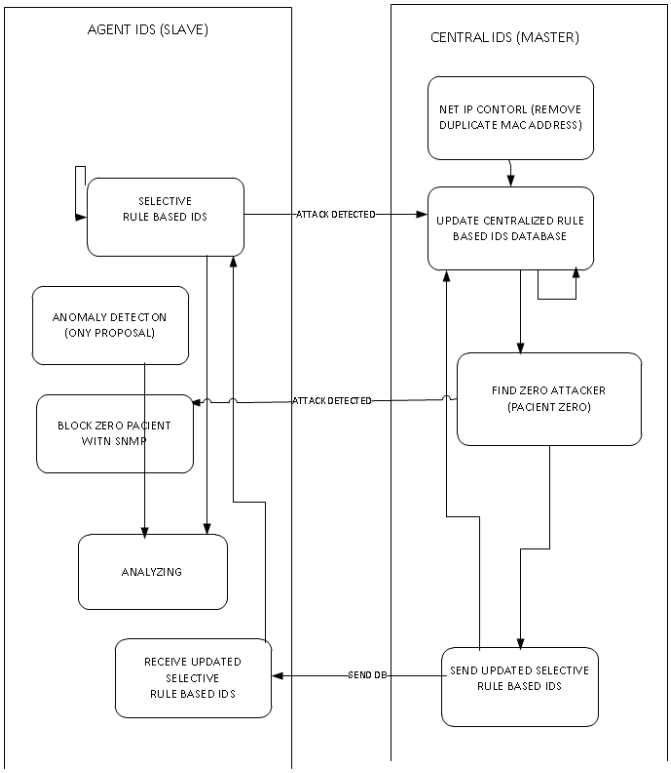

Fig. 3. Elements and modules of the ISP

4.3. Detecting duplicate MAC address in one LAN network

In one network if there is a duplicate MAC address, it can only be detected if there is one physical network with one VLAN's. If there are multiple VLAN's or multiple address pools with one or many DHCP services, the detection of duplicate MAC addresses cannot be detected.

Our Proposed solution is every IDS slave, (One IDS slave host is placed on every physical switch or VLAN, this is a strategic decision in order to get host and network type of IDS) using the ARP protocol to map all of its neighbor host PC's. The list of all the IDS slaves are send to the master IDS. Then the master makes one jointed list, and the possible duplicate MAC addresses are detected. If One MAC address is in many network than that is an virtual interface made by some type of WORM. Form this list the preserved IP/MAC addresses are ignored, like broadcast.

Then the MAC address is blocked on ALL VLAN's or address pool's, using SNMP TRAP.

4.4. Selective DB with attacks

One of the universal or unique solutions of this paper is the selective Database (DB) with attacks. The entire DB with signatures of attacks is kept at the master IDS.

Locally at all the IDS slaves only a selective DB are being kept. The algorithm for exchange is LRU.

The network traffic is recorded in by the slave IDS, the packet are copied, and are converted in understandable form for the our ISP. The data is first converted from HEX to ANCII, by using HEX to ANCII decoder. Later on the value of every variable from the IP header for each packet is written in a predefined class IP packet or an object is derived from that class IP packet.

If the packet matches one of the rules / signatures of attacks from the selective DB in the slave IDS.

If the packet does not match any of the signatures of attacks from the selective DB in the slave IDS, the packet is then send to the slave IDS for more detailed analyses.

The difference is that when the IDS slave detects an attack, it can be block there is time for that action, therefore the slave IDS is the Intrusion Prevention System.

Using analogy from the medical quarantine procedure (mentioned in this paper above) the slave IDS make so called vaccination of their host neighbors, this is an important task in order to prevent the network attack from spreading in other parts of the network.

The detected signature in the slave IDS is put on the top of the signature based DB and send update for the number of detected attacks from those signature to the master IDS, and the master IDS updates the selective DB of all the slave IDS with the new information.

The used algorithm for exchange of rules in LRU least recently used, the reason for choosing this algorithm and not FIFO or LIFO, are explained in more details in appendix 4 of [2].

The master IDS compare the packet with all of the signatures in the database, so the attacks will be detected after it will have happened. Using sort function, the packet are sorted according to the logic first the LAN, and then WAN because the LAN packet are "faster" then the WAN packets. With this methodology and with replacing all the public IP addresses with the LAN IP address of the default gateway using (Reverse NAT), the attacks are detected faster for the fastest packets.

The experiment for the sort algorithm and which sort algorithm for this module of the ISP is chosen is explained in more details in [7], and in section 5 (Experiment) in this paper.

4.5. Finding Patient Zero

In scenario of multiple attacks or medically an "epidemic outbreak", the module for finding patient zero is used; the detailed explanation for this part is presented in this section.

From implementation aspect every switch has a slave IDS and analyses the traffic only for the host on the same switch. With this implementation the number of computers / host in the network will not affect computational speed of the IDS, this is solution for the problem of any network based IDS, that have slower computational speed with the increase number of new hosts in the network.

Our proposed algorithm for the ISP also does not the disadvantage of any Signature based IDS, by using a selective database for the signature based attacks in the slave IDS part of the ISP.

In a classical signature based IDS with time the number of rules in the signature DB increases and with that the computational speed/time of the packets, and so the possible

attacks are detected more and more later. So with time the classical signature bases IDS are losing their efficiency.

Our algorithm in the slave IDS part of the ISP uses fixed number of signature.

The algorithm for exchange of rules between the master and slave IDS's is Least Recently Used, explained in more details in appendix 4 of [2]. In section 5 (Experiment) of this paper, an experiment is presented which determines the right or optimal number of rules that should be kept in the selective signature DB.

Using this DB the slave IDS's compare the packet for possible attacks.

The packet marked as attacks by the slave IDS are send with TIME STAMP of the detected attacks are then send to the master IDS for detailed analyses.

All the same attacks with different IP sender and IP receiver packers are put into a single array, the array is sorted from smaller to larger according to the value of the variable TIME STAMP. The IP sender address of first element in the array is the zero patient or the first attacker.

That IP address is BLOCK from the network and its associating MAC address, this tasks is performed using SNMP TRAP.

5. TEST DATA SETS

In 1998 and 1999 The Information Systems Technology Group of MIT Lincoln Laboratory with the support of the Defense Advanced Research Projects Agency and the Air Force Research Laboratory (all from the USA), had worked on a new innovative experiment in the field of intrusion detection systems. They had done a cutting edge experiment for the time, creating an Intrusion detection system that monitors the state of an active computer network, looking for some form of attack like denial of service, form of abuse like unauthorized usage, or rear and strange behavior like some forms of so called anomalous behavior.

The experiment was set in a real military base with real computers, but the attack were simulate (it was known what was attack what was a normal connection, this was used later on to evaluate the effectiveness).

The experiment in 1999 (1999 DARPA Intrusion Detection Evaluation Data Set [5]) was small improvement on the experiment done in 1998. In the 1999 the "simulated" attacks lasted 5 weeks, the first and third week was normal traffic, the second week Contained Labeled Attacks. The attacks were divided into five main categories: Denial of Service Attacks, User of Root Attacks, and Remote to Local Attacks, Probes and Data. The full list of attacks is presented on [REF-04]. Then the system was tested with random network packets (some attacks, some normal traffic), there were 201 instances of about 56 types of attacks distributed throughout these two weeks. At the time the main purpose of the experiment was creating the intrusion detection system, but the real "hided" value of this experiment was the 1998/1999 DARPA Intrusion Detection Evaluation Data Set.

The collected data from this are made available for all the researchers that needed a test data set for their intrusion detection system. This data set had made possible the creation of many future intrusion detection systems. Proof of the value of this data set is the number of publications using this data set in their research project. This is the reason why we intended to use the 1998/1999 DARPA Intrusion Detection Evaluation Data Set. [5]

The 1998/1999 DARPA Intrusion Detection Evaluation Data Set, is a very data set containing around 4-5 GB of data, our main purpose was testing our intrusion detection genetic algorithm, so in order to minimize the time for analyzing the data set and maximizing the testing type, we used the optimized versions of the DARPA data set, the KDD CUP 99 Data Set. KDD CUP 99 Data Set is compiled from the 1998 DARPA Intrusion Detection Evaluation Data Set. The database contains normal connections and 24 types of attacks, the types of attacks are presented on [5]. Evaluation on the KDD CUP 99 Data Set and Summary report with type of attacks and number of connections for each type of attacks is presented on the table in [6]. For this paper and our algorithm it is very important to present the KDD CUP 99 Data Set Schema properly and precisely, for this we will use the tables from the tasks for the KDD CUP 1999 [6].

6. ALGORITHM AND IMPLEMENTATION

In [7] we presented signature based IDS which can dynamically allocate the number of rules that are kept in the detection system based on the LAN complexity and number of packets in the network at the moment. The idea is to process and analyze the packet in the distributed agents IDS before the time of arrival at the destination computer and processing the packet by the host computer. This is done using complex mathematical mechanism for calculating the packet travel time. Using Least Recently Used algorithm to remove the not used rules are after the buffer of rules in the DB of the distributed agents is over its pre-defined limit. This makes the hosts in the LAN protected with intrusion detection and prevention for the more recent and frequent type of attacks, but leaves the hosts vulnerable to new attacks (that are not in the current buffer of rules in the DB of the distributed agents). The new attacks will be detected with deeper analyses at the IDS server side (after analyzing the whole "historical" packet in the LAN) and the rule for that attack will be added in the buffer of rules in the DB of the distributed agents. The main problems is that the first new type of attack or intrusion will be only detected but not prevented. In order to increase the efficiency we need to keep more rules in the buffer of rules in the DB of the distributed agents. This can be done by using CUDA programming. To explain the process of using CUDA in IDS analyses, we will use simple matrix (matrix no.1).

Matrix no.1: [1 4 5]; [1]; [9 7].

Firstly in order to be processed by GPU the matrix must have same number of elements in every row. In order to do that in the shared memory (of GPU) every thread adds the number 1 for every element in the row. Then we use the Max_Number() function, that way we calculate the most elements per row, and then the value is placed (MAX_NUMBER) in the shared memory in GPU. Afterward we process the matrix using thread for each row if there is a missing number of elements (compared to the compared to the MAX_NUMBER) we add "empty element" (the number -1 is ignored in processing). Then we get this matrix (matrix no.2). Adding the "empty element" must be done if we want be process the data with CUDA.

matrix no.2: [1 4 5]; [1 -1 -1]; [9 7 -1].

This in our simplistic example representing the buffer of rules in the IDS distributed agent. Then we and packet s from the network traffic (matrix no.3) in the shared memory, pre-process them (adding “empty element” if the number of elements is lower the MAX_NUMBER).

The last step is comparison of the DB IDS rules (matrix no.2) with the packets from the network traffic recorded (array [7, 9,-1]) placed in the shared memory of the GPU in order to be accessed by every thread at once. This speedup the comparison by 10 times at least.

6.1. Code Samples in NS-3

```
Class IPv6Sender();
{
   Int ProcessID;
   Int UserID;
   Int OsID;
   Int SubnetID;
   Int InterfaceID;
}


Class IPv6Recever ();
{
   Int SystemRunTime;
   Int SubnetID;
   Int InterfaceID;
}
```

Example 3. IPv6 Addition

```
#include "ns3/core-module.h"
#include "ns3/simulator-module.h"
#include "ns3/node-module.h"
#include "ns3/helper-module.h"
using namespace ns3;
NS_LOG_COMPONENT_DEFINE ("FirstScriptExample");
  int main (int argc, char *argv[])
{          LogComponentEnable("UdpEchoClientApplication",          LOG_LEVEL_INFO);
LogComponentEnable("UdpEchoServerApplication", LOG_LEVEL_INFO);

  NodeContainer nodes;  nodes.Create (2);
  PointToPointHelper pointToPoint;
  pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("5Mbps"));
  pointToPoint.SetChannelAttribute ("Delay", StringValue ("2ms"));
  NetDeviceContainer devices;
  devices = pointToPoint.Install (nodes);
  InternetStackHelper stack; stack.Install (nodes);
```

```
 Ipv6AddressHelper address;
 address.SetBase ("AutoAssign(IPv6Sender();)");
 Ipv4AddressHelper address;
 address.SetBase ("AutoAssign(IPv6Recever();)");
 Ipv6InterfaceContainer interfaces = address.Assign (devices);
 UdpEchoServerHelper echoServer (9);
 ApplicationContainer serverApps = echoServer.Install (nodes.Get (1));
 serverApps.Start (Seconds (1.0));
 serverApps.Stop (Seconds (10.0));
 UdpEchoClientHelper echoClient (interfaces.GetAddress (1), 9);
 echoClient.SetAttribute ("MaxPackets", UintegerValue (1));
 echoClient.SetAttribute ("Interval", TimeValue (Seconds (1.)));
 echoClient.SetAttribute ("PacketSize", UintegerValue (1024));
 ApplicationContainer clientApps = echoClient.Install (nodes.Get (0));
 clientApps.Start (Seconds (2.0));
 clientApps.Stop (Seconds (10.0));

 Simulator::Run (); Simulator::Destroy (); return 0;
}
```

Example 4. Examle in NS-3

6.2. Experiment

The main purpose of this experiment is to test the effectiveness of the prososed ISP algorithm, using dynamic number of selective rules in the IDS sensors (intelligent agent IDS).

Materials used for the experiment: Hardware: 1 desktop PC with the following components, 1 virtual machine on the same PC (using from the Host Machine: 1 core of the CPU, 1 GB RAM)

- 3 GHz CPU (Intel Core 2 Duo E7500)
- 3GB RAM DDR2 (400 MHz),
- Motherboard Intel P43, System Bus 800 MHz
- HDD 160 GB (SATA 2, @7200 RPM)

6.3. Methods

First the DARPA DATASET'99 training collection is used to define the rule set of attacks in the central IDS, then using the distributed agents and the DARPA DATASET'99 training collection the algorithm is tested for the most optimal number of attack rules (minimal number, maximum number of detected attacks). The test is reputed multiple times using different number of attack rules in the distributed database (the attack rules are replaces using LRU).As a reference for indenting is the packet an attack or not, we use SNORT as a 99, 9 % effective signature based IDS system.

6.4. Data

Number of attack rules in Distributed Agents	% of attacks detected by the proposed algorithm for IDS
20	11,25%
40	36,41%
60	46,71%
80	63,57%
100	65,39%
120	68,92%
140	69,12%

Table 1: Number of rules in Distributed Agents, % of attacks detected

6.5. Results

The system for testing is the following: One PC: (Intel® Core™ i7-3770K with 8M Cache @ 3.90 GH; Mobile Intel® QM77 Express Chipset; 16 GB DDR3 @ 1600 MHz; HDD 500 GB @ 7200 RPM; GPU: NVIDIA Quadro K2000M - 384 CUDA CORES), RHEF Linux 6 with C/C++, NS-3 and Windows 8.1 with C# (VS 2010).

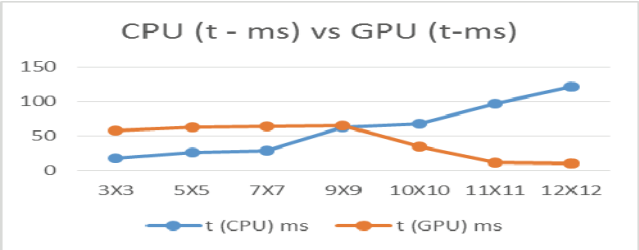

Fig. 4. CPU vs GPU compared (time – ms)

From Fig. 4 we can clearly see that GPUs work much faster with larger data sets, compared to CPUs. Therefore they are ideal candidates for using them as main processing power for big data analytics (IDS, email spam).

7. CONCLUSION

NVIDIA GPUs provide massive computation potential in the field of computer science. Our proposal is a new type of using the GPU in the field of computer science more

specifically in the field of system administration. Our Future work includes creating an interface for data exchange between CUDA and OPENGL in order to use Intel GPUs as well, also we will try to create dataset of mail for testing email spam (like DARPA DATA for IDS).

REFERENCES

[1] Internet 2012 in numbers - Royal Pingdom U.K., http://royal.pingdom.com/2013/01/16/internet-2012-in-numbers/
[2] NVidia CUDA platform, http://www.nvidia.com/
[3] W. Jeong, P. T. Fletcher, R. Tao, and R.T. Whitaker, "Interactive Visualization of Volumetric White Matter Connectivity in DT-MRI Using a Parallel-Hardware Hamilton-Jacobi Solver, 2007.
[4] F. Xu and K. Mueller. "Real-Time 3D Computed Tomographic Reconstruction Using Commodity Graphics Hardware," Medicine and Biology, 2007.
[5] MIT Lincoln Labs DARPA DATA SET 1998/1999 http://www.ll.mit.edu/mission/communications/cyber/CSTcorpora/ideval/data/
[6] KDD Cup 1999 Data https://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html
Aleksandar Sokolovski, Saso Gelev ALGORITHM FOR DISTRIBUTED AGENT BASED NETWORK INTRUSION DETECTION SYSTEM (NIDS), CIIT 2011 Bitola, Macedonia