Traffic Classification over Gbit Speed with Commodity Hardware

Géza Szabó, István Gódor, András Veres, Szabolcs Malomsoky, Sándor Molnár

Original scientific paper

***Abstract*—This paper discusses necessary components of a GPU-assisted traffic classification method, which is capable of multi-Gbps speeds on commodity hardware. The majority of the traffic classification is pushed to the GPU to offload the CPU, which then may serve other processing intensive tasks, e.g., traffic capture. The paper presents two massively parallelizable algorithms suitable for GPUs. The first one performs signature search using a modification of Zobrist hashing. The second algorithm supports connection pattern-based analysis and aggregation of matches using a parallel-prefix-sum algorithm adapted to GPU. The performance tests of the proposed methods showed that traffic classification is possible up to approximately 6 Gbps with a commodity PC.**

***Index terms:* traffic classification, GPU, parallel algorithm**

I. Introduction

Certain applications, most notably belonging to the class of peer-to-peer (P2P) applications, are difficult to identify by signature search either because there are no appropriate signatures, or the signatures are difficult to match. Connection pattern-based methods, e.g. [1], [2] provide a light-weight, albeit heuristic solution to this problem.

In theory signature search and connection pattern-based methods could be combined to increase classification accuracy: finding signatures need Deep Packet Inspection (DPI), and if no signatures are found, the connection pattern method can be used. Or vice versa, if DPI marks the traffic of a host on a specific port as P2P, further connections to this host-port pair are very likely to be also P2P even if DPI has no results for them [3].

A serious downside of this combined solution is that it is not light-weight any more and raises considerable challenges to implementation. Therefore, extending a simple flow collector system with traffic classification capabilities needs more computing power than what CPUs can offer. Today there is a trend to use commodity hardware e.g., [4], [5] as it has low price/performance ratio, short development time and it is easy to get familiar with. The requirement for a dedicated hardware would slow down the wide-spreading of a particular method.

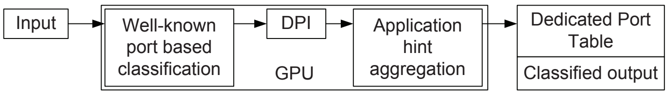

Input
Well-known port based classification
DPI
GPU
Application hint aggregation
Dedicated Port Table
Classified output

Fig. 1. Steps of traffic classification with GPU

Not long ago, Graphical Processing Units (GPUs) have also become well programmable computing elements [6]. Furthermore, the current paradigm shift in processor technology towards many core architectures[1] has already occurred in graphical processing units a few years ago. Thus highly parallel algorithms can be experimented there which will be later well suit for general processor architectures as well when they become many core architectures.

This paper addresses certain key algorithmic challenges of high-speed signature search and connection pattern-based classification. Beyond describing these tasks, we show how to implement them on GPU and present measurement results on the efficiency of the proposed system.

The contributions of the paper are the following (see also Figure 1):

- We propose a memory optimized technique for signature matching. It involves a modification of the Zobrist hashing algorithm [7] to encode the application signatures. We store the dictionaries in the cache of the GPU and apply the same encoding to check which application a packet may belong to. Due to the compactness of the hashing algorithm the data sets of the proposed DPI method reside in the fast cached memory of the GPU providing high efficiency, in contrast to [8] where the signatures do not fit to the cache. The details of the method are presented in Section III.
- We propose a fast-speed aggregation and update method to support connection pattern-based identification. We aggregate all the information necessary for the classification of flows into one data structure providing the best-available hint all the time. A Dedicated Port Table (DPT) stores the number of times a specific host-port pair was identified as the source or the destination of a specific application determined by, e.g., the above DPI method. In order to avoid a per-packet update of DPT, we propose to aggregate this information in the GPU by an extension to the parallel prefix scan [9]. This method provides massive parallelization and high efficiency with GPU. The details of the method are presented in Section

G. Szabó, I. Gódor, A. Veres, Sz. Malomsoky are with TrafficLab, Ericsson Research, Budapest, Hungary (email: {geza.szabo, istvan.godor, andras.veres, szabolcs.malomsoky}@ericsson.com)

S. Molnár is with the High Speed Networks Laboratory, Department of Telecommunications and Media Informatics, Budapest University of Technology and Economics (email: molnar@tmit.bme.hu)

[1] The terms many-core and massively multi-core are used to describe multi-core architectures with an especially high number of cores (tens or hundreds). In case of many-core processors the number of cores is so large that traditional multi-processor techniques are no longer efficient.

1845-6421/09/8185 © 2009 CCIS

IV.

Finally, Section V presents an evaluation of the GPU based algorithms. We show that we can possibly reach beyond 6 Gbps traffic classification rate using commodity hardware.

II. Related work

There are several independent traffic classification methods in current literature: (a) port-based classification [10], when classification is based on associating a well-known port number with a given traffic type (b) DPI e.g., [11], when protocol recognition is done by searching byte patterns in stateless manner, (c) connection pattern based classification [1], when the idea is to look at the communication pattern generated by a particular host, and compare it to the behavior patterns representing different activities or applications, and (d) statistics based classification e.g., [12], when some statistical feature of the network traffic of a specific application is captured and used later to classify the traffic. In our work we grab the first three type of these and show how to use them efficiently on a system with massively parallel architecture. Note that the proposed solution is general in a mean that it could also incorporate the results of a statistics based classification module.

According to [13], the most resource consuming task is signature matching in traffic classification systems. String matching can be accelerated with ASICs, FPGAs [14], [15] and previous generations of videocards [16], [17], [18] (as texture operations), but they are difficult to modify and extend with new signatures and functions, which would be essential for traffic classification systems.

In [8], the deterministic finite automata (DFA) and the extended finite automata (XFA) based signature matching was analyzed. The authors found that the G80 GPU implementation was 9x faster than the Pentium4 CPU implementation. They emphasized the problem that data structure of the automata (in the order of MBytes) does not fit in the cache of current GPU architectures which would be essential for optimal operation. The packet processing speed of this solution was about 80,000 packets/sec (with FTP, SMTP and HTTP signatures).

In [19], the signature matching of an Intrusion Detection System (IDS) was done with GPU providing system throughput of 2.3 Gbps with synthetic traces and 600 Mbps with real traffic. In [20], the authors further developed the work of [19] by reimplementing the DPI engine to the new CUDA architecture [6]. The overall system througput increased by 30% on real traffic compared to [19]. In [21] further refined the problem of load balancing the signature matching procedure of a large dictionary for multiprocessors. They proposed pattern set partition and input text partition together to fully utilize GPUs during the pattern matching.

Note that in [8], [19], [20], [21] the main tasks were to implement IDS on the GPU. This requires (a) the recognition of a huge set of protocol signatures with their numerous exploitation signatures, (b) find the signature anywhere in the byte stream, and (c) false positive hit is not allowed. I.e., the proposed methods for traffic classification task should not be directly compared to the above methods. As it can be seen later, our methods outperform them in raw packet processing speed, but on a more focused problem set of traffic classification (see Section V and V-C for details).

III. Signature Matching in GPU

In case of DPI, application specific bitstrings are searched in the packet payloads. DPI is a well-parallelizable task considering either the analysis of several packets in parallel or several signatures on the same packet. Moreover, traffic classification has a more focused requirement set comparing to an IDS functionality:

- The 'only' goal is to identify the applications, so less signatures is needed focusing on handshake messages typically found in the first few packets of the flows. For example, it is enough to identify an SMTP protocol from the first EHLO message and there is no need to parse any of its later messages.
- In most cases the signatures can be found in the first few bytes, thus the method can focus only on them.

In order to achieve high processing speed with a GPU-based solution, the memory access overhead should be minimized. Our goal was to store the signatures on a minimized memory footprint to achieve this. State machines are processor-efficient methods for signature matching but not space-efficient [8]. Use of hashes results in data compression, but it is difficult to perform wild-card character matching. E.g., typical solutions store the same signature with all the possible wildcard values in the hash [22]. This results in the significant increase of the hash table size and in false positive hits as well. In the following paragraphs we propose an encoding method, which eliminates the above problems. I.e., the input and output data of the DPI process occupy only the fast access memory of the GPU and the method provides wildcard support.

A. Zobrist hashing

Our proposal applies the idea of Zobrist hashing [7]. This technique was developed for creating hash codes in board games and storing the states of the players. Each piece has a unique identifier corresponding to its actual position on the board. The hash code of the actual state is calculated by XORing the unique position identifier of each piece. Instead of storing the whole board in each step of the game, only the hash values should be stored. I.e., the hash itself stores the data and not only a pointer to an external data field.

B. Input data for string matching

The proposed string matching technique uses the following input data (see Table I for illustration).

- The application signature dictionary is denoted by S (see Table I(a)). One element of this dictionary, S_i^j denotes the j^{th} character of the i^{th} application.
- The list of bitmasks of non-wildcard characters in the application signatures is denoted by B (see Table I(b)). One element of this list, B_i^j denotes whether the j^{th} character of the i^{th} application is wildcard or not.

TABLE I
INPUT DATA OF THE DPI METHOD

(a) Input application signature dictionary (S)

App#1	a?b
App#2	aaa
App#3	?a?
App#4	??a

(b) Bitmask of applications signatures (B)

101
111
010
001

(c) Alphabet-position dictionary (α)

	0	1	2
a	1100	1011	1000
b	1010	1110	1001

(d) Encoded signature database (E)

0101
1111
1011
1000

- The alphabet-position dictionary is denoted by α and represented as a matrix (see Table I(c)). The rows of the matrix represent the characters of the alphabet. The columns represent the positions of the characters in the signatures. One element of this matrix, $\alpha^j_{S^j_i}$ denotes the character code of the j^{th} character of the i^{th} application. These codes are represented as random numbers with a bigger range than the size of the dictionary in order to minimize the collision of encoded signatures.
- The encoded signature database calculated from the above input data and denoted by E (see Table I(d)).

The encoded signature database of a particular application is calculated by XORing together the codes of its characters if their corresponding bitmask is 1. In general it can be written as $E_i = \bigoplus_{j=0}^{|S_i|-1}(\alpha^j_{S^j_i} \bullet B^j_i)$. For illustration, let us see a specific example based in Table I: e.g., `a?b` is encoded into: $1100 \oplus 1001 = 0101$.

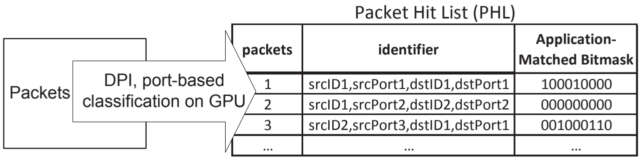
C. The proposed string matching method

The proposed string matching method is as follows. Each thread of the GPU deals with the content of one packet. This choice is a direct consequence of the Single Instruction Multiple Data (SIMD) architecture. Further, the GPU can deal efficiently with thousands of lightweight threads [6]. This is a big difference to an e.g., x86 architecture where the threads are monolithic and both task switching and scheduling are complex tasks.

The content of packets is considered as character strings. Apply the same procedure on these character strings as on the application signatures: the codes of the characters are XORed together according to the corresponding bitmasks. If the encoded packet data matches with the corresponding encoded application signature, then the packet is considered to belong to the particular application. The hit by the DPI process sets the proper bit of the specific application to 1 in the Packet Hit List (see Figure 2 and Section IV for further discussion).

D. DPI data structures in GPU memory architecture

The *global memory* (uncached) space is filled with the array of network packets. During the initialization of each thread, the corresponding array of the packet bytes are copied from the global memory to the registers or to the *shared memory* (cached) of the thread. Thus global memory access does not reduce the speed of arithmetic calculations with the same data.



packets	identifier	Application-Matched Bitmask
1	srcID1,srcPort1,dstID1,dstPort1	100010000
2	srcID1,srcPort2,dstID2,dstPort2	000000000
3	srcID2,srcPort3,dstID1,dstPort1	001000110
...	...	...

Fig. 2. Input/output of signature matching on GPU

The *constant memory* space is cached so a read from constant memory costs one memory read from device memory only on a cache miss, otherwise it just costs one read from the constant cache. The pre-calculated input data structures are loaded into the constant memory space (the alphabet-position dictionary, the bitmasks and the encoded signatures). The allocable constant memory size is 64 Kbyte for the whole kernel in the test configuration (see Appendix). If we consider our example implementation where the signature database consists of 4 byte long values, then about 10 thousands of signatures could fit into the constant memory.

IV. FLOW CLASSIFICATION BASED ON THE DEDICATED PORT TABLE

Some specific traffic types are difficult to be identified simply based on DPI without considering flow groups and their history. E.g. according to [1], in case of small P2P flows trying to establish connection to non-existent peers, partially encrypted P2P traffic or Skype. In this section we propose how to combine efficiently the DPI and connection pattern based classification methods for flow classification on GPU.

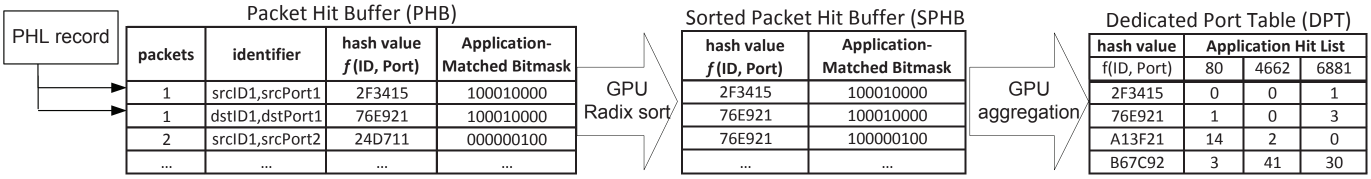
The **purpose** of the DPT is to store the application hints of host-port pairs based on their communication history and provide the traffic classification engine with the best available hint all the time. The DPT is represented as a list of records containing the number of hits per application for each host-port pairs.

Note that a simple port-based classification is also applied on the GPU after the DPI process. In [23] it was shown that in many cases the port based classification has become obsolete but it is still a useful aid in a traffic classification system. As it is a simple existence check of a port number in a given list thus no further discussion is given here. The hits of the port based classification module are also stored by setting the proper bit of the specific application to 1 in the Packet Hit List (see Figure 2).

The DPT is created and updated as follows.

A. Starting from per-packet application hits

The starting point of the process is the Packet Hit List (PHL). Whether a packet comes from an application or not is a yes/no question, thus the PHL is represented as a bitmask. In order to deduce per host per port information, each record of PHL is split into source and destination identifiers (hash(srcID, srcPort), hash(dstID, dstPort)), for illustration see Figure 3.

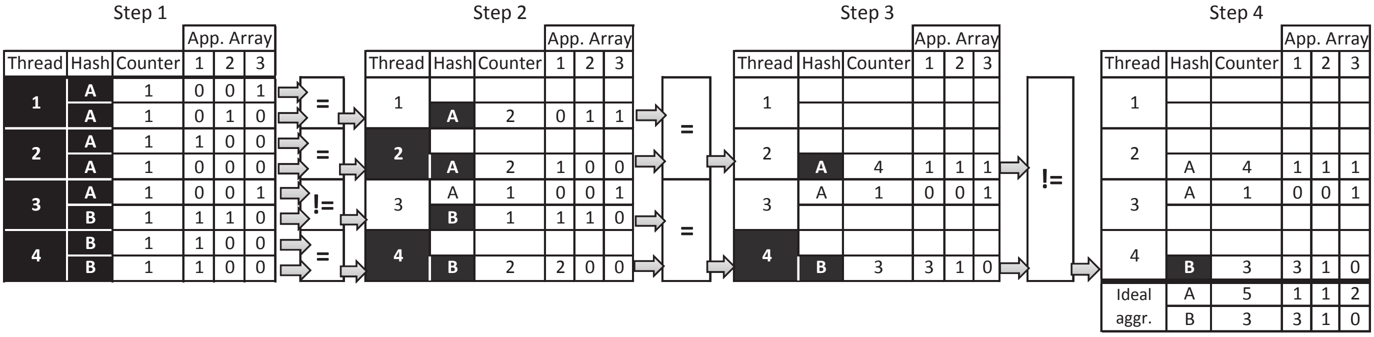


packets	identifier	hash value ƒ(ID, Port)	Application-Matched Bitmask
1	srcID1,srcPort1	2F3415	100010000
1	dstID1,dstPort1	76E921	100010000
2	srcID1,srcPort2	24D711	000000100
…	…	…	…

hash value ƒ(ID, Port)	Application-Matched Bitmask
2F3415	100010000
76E921	100010000
76E921	100000100
…	…

hash value	Application Hit List		
f(ID, Port)	80	4662	6881
2F3415	0	0	1
76E921	1	0	3
A13F21	14	2	0
B67C92	3	41	30

Fig. 3. Dedicated port search



Step 1

Thread	Hash	Counter	App. Array		
			1	2	3
1	A	1	0	0	1
	A	1	0	1	0
2	A	1	1	0	0
	A	1	0	0	0
3	A	1	0	0	1
	B	1	1	1	0
4	B	1	1	0	0
	B	1	1	0	0

Step 2

Thread	Hash	Counter	App. Array		
			1	2	3
1					
	A	2	0	1	1
2					
	A	2	1	0	0
3	A	1	0	0	1
	B	1	1	1	0
4					
	B	2	2	0	0

Step 3

Thread	Hash	Counter	App. Array		
			1	2	3
1					
2					
	A	4	1	1	1
3	A	1	0	0	1
4					
	B	3	3	1	0

Step 4

Thread	Hash	Counter	App. Array		
			1	2	3
1					
2					
	A	4	1	1	1
3	A	1	0	0	1
4					
	B	3	3	1	0
Ideal aggr.	A	5	1	1	2
	B	3	3	1	0

Fig. 4. The aggregation of packet information with parallel reduction

The number of stored application types and the size of the application-matched bitmasks are the results of a trade-off due to the memory addressing of the GPU. The GPU can handle 2 or 4-byte long data most efficiently [6], thus 16 or 32 possible application categories can be distinguished in these cases. I.e., if P2P is chosen as one category, its 'variants' like BitTorrent and Gnutella cannot be differentiated later. But our experiences support that 16/32 application types are practically enough to cover the main types such as P2P file sharing, web browsing, streaming, VoIP, etc. If needed, this part of the algorithm can be extended to support more application types by switching to longer non-hardware supported data length representation. However, this solution results in performance degradation. An alternative solution is to switch between signature lists. Once the packets filled in the global memory are evaluated based on the first list, then these packets can be evaluated based on a second, third, etc. lists.

The theoretical speed of the proposed DPI engine with current hardware is approx. 2 million packets/sec (see V-A2 for the performance test) that is 20k flow presuming 100 packets/flow in average [24]. This means that a 2-bytes-long hash representation will not fully occupy its array on the GPU which is stored only for one DPI iteration per batch. Since the duration of the flow is approx. 60 sec [24] there are 60 times more flows to be stored in the DPT (hosted in the main memory). Therefore the hash representation on the CPU side can be increased to 3 bytes which results in a 256 times larger hash array occupying in the order of 100 MBytes memory.

In order to avoid the overhead of frequent GPU initializations, we propose to store the PHLs in a Packet Hit Buffer (PHB) in the operative memory until the aggregation phase.

B. Aggregating the per-packet application hits

Before updating the DPT, we propose to aggregate the PHB based on parallel reduction. The idea of parallel reduction is to slice a big problem to smaller tasks, distribute it to processing units and execute the computation of the partial tasks parallel. The algorithm proceeds to recursively reduce the problem size.

The reduction of the PHB is more efficient if its input is sorted. The sorting ensures that the same [srcID; srcPort] and [dstID; dstPort] pairs come after each other. This way the shared memory blocks of the GPU will contain application information about packets that likely belong to the same [host; port] pair. Therefore the chance of effective aggregation per block is high. We have applied parallel radix sort [9] to create a Sorted PHB (SPHB) based on the hash value of the packets.

The SPHB is aggregated with parallel reduction into the Aggregated Packet Hit Buffer (APHB) similarly to the parallel prefix sum [9]. Extending [9], we propose to use conditional aggregation while sweeping through the data structure. The idea is to handle the SPHB as a balanced binary tree and sweep it from the leaves to compute the aggregated sums.

The steps of the aggregation can be followed in Figure 4 and Algorithm 1. The input of the algorithm is the SPHB extended with a counter in each row. This counter is initialized to 1 and it represents the number of successfully aggregated packets (i.e., rows). In the first step in Figure 4, each thread compares the hash keys of the two rows they are responsible for. If the keys are equal then the aggregation occurs by summing the aggregation counters and the proper application hit array elements. The results are stored at every second row. In the following step, every second thread does actual work, comparing and adding data in the recently updated rows, and storing them in every fourth array position. This is repeated until there is no more row left untouched. The advantage of parallel reduction with such a data structure is that there are no concurrent memory reads or writes. The threads have to be synchronized after each step to ensure memory content consistency.

Operations are performed in place in the shared memory, thus no additional memory allocations are needed. If the binary tree has n leaves and $\log(n)$ levels, then only $O(n)$ operations are needed by the algorithm (it performs $(n-1)$ adds).

Algorithm 1: Packet Aggregation

```
Input: $SPHB$
Output: $APHB$
for $d = 0, d < \log_2 n - 1, d++$ (for the size of $n$ see Section V-B) do
    for $k = 0, n - 1 > k, k+ = 2^{d+1}$ parallel do
        if $x[k + 2^d - 1].hash == x[k + 2^{d+1} - 1].hash$ then
            $x[k + 2^{d+1} - 1].counter+ = x[k + 2^d - 1].counter;$
            for $i = 0, i < size(bitmask), i++$ do
                $x[k + 2^{d+1} - 1].app[i]+ = x[k + 2^d - 1].app[i];$
```

C. Sequential update of the DHT

The APHB is moved back to the operative memory and the DPT is updated sequentially by adding the corresponding applications hit values to the rows of the DPT. The update is started from the last row of each block of the APHB since these rows have the highest aggregation level. The counter of these rows shows how many rows should be jumped upwards to get to the next useful row to be included in the DPT. The speed of updating the DPT is also increased by the fact that the SPHB remains sorted after the parallel reduction. This results in good spatial locality of the hash values in the DPT, which means high hit rate in the L2-cache of the CPU during update [25]. Also note that when querying the DPT during traffic classification, the best matching application for a [host; port] pair can be obtained in one step, if the application hit list in the DPT is a heap, where the number of hits for different application types is in decreasing order.

A flow is **classified** by querying the DPT for the most probable application type corresponding to the [srcID; srcPort] pair and the [dstID; dstPort] pair, and comparing their results. In case of equivalence the application type is unambiguous. In case of difference, we decide for the more probable and more specific category.

V. Evaluation

In this section we evaluate the efficiency of the GPU based DPI and flow aggregation, and the total system throughput.

A. Evaluation of the GPU based DPI method

1) Compression vs. accuracy: The collision probability of the encoded signature database (i.e., misclassification of packets, false positive hit) can be analyzed as a function of the length of the signatures and their representations in the alphabet-position dictionary.

The size of the alphabet-position dictionary is $a * p$, where a is the possible number of characters and p is the possible number of positions (i.e., length of the signatures). To represent the signatures completely collision free, each character of the alphabet should be represented with $\log_2 a$ bit, and the character coding should be rotated according to the position. I.e., the size of one element of the dictionary should be $m \geq p \log_2 a$. In our case $a = 256$, i.e., $m \geq 8p$. In practice, however, much smaller m can provide negligible collision probability.

TABLE II
The collision probability of the compression

m	10	11	12	13	14	15	16
$p = 16$	0.2527	0.0867	0.0250	0.0069	0.0017	0.0004	0.0001

Due to the complexity of analytical analysis of the collision probability based on XOR functions [7], [26], [27], we applied the common practice for choosing a possible hash value representation length (m) by analyzing the collision probability with simulation. We tested the collision probability with 1000 signatures with different payload length ($p = 1, 2 \ldots, 256$) which is enough for practical usage [8]. The test was done in the function of m from 10 to 32 bit representation with 100 independent runs by setting different random values in the alphabet-position dictionary. Note that m should be at least 10 because $2^m > 1000$. For illustration, see Table II showing the collision probability in case of 1000 signatures of 16 characters for $m = 10, \ldots, 16$. Our experiments show that the collision probability is practically independent of p and falls below 10^{-4} in case of $m = 16$ (2 bytes).

2) Performance: The proper initialization setup of the GPU kernel regarding the number of threads processed parallel by a multicore unit on the GPU has a high impact on the overall performance [6]. We created several measurements with different setups where the parallel processed packet batch size was varied from 16k to 500k packets. In case of CPU, the batch size would be the number of packets processed one-by-one without the interruption of any other operation, while in the case of GPU it means the number of parallel threads.

In the performance tests, the following implementations of the DPI task were analyzed:

- `regexp`: A perl-based implementation of the DPI method with standard regular expressions.
- `CPU`: The CPU-based version of the proposed DPI method.
- `GPU`: The GPU-based version of the proposed DPI method without shared memory caching of the packet headers and payloads.
- `GPU_shared`: The GPU-based version of the proposed DPI method with shared memory caching of the packet headers and payloads.

In our tests we replayed recent traces captured in live broadband mobile networks during busy hour traffic. The current traffic mixture of broadband mobile networks is similar to those presented in [24], [28]. The applied signature database contained an extended version of [29] consisting of approx. 300 signatures in total. The average signature length is 16 bytes characterizing 35 different application protocols grouped into 16 application types. Table III summarizes the performance tests focusing on the average packet processing speed. Note that the processing time is linearly increasing with the increase of input signature list. The slowest solution is `regexp`. Over this, `CPU` provides 5 times faster solution.

TABLE III
DPI PERFORMANCE COMPARISON – PACKETS PROCESSED PER SECOND

	regexp	CPU	GPU	GPU_shared
Average [x1000 packets]	14	72	138	1844
Relative variance *f(batch size)* [%]	1.1%	3.1%	25%	0.2%
Relative variance *f(payload)* [%]	49%	36%	37%	27%

GPU provides additional 2 times speed increase. Additional magnitude of speed can be gained by GPU_shared with exploitation of cached shared memory. In this latter case, the global memory is processed in small well-cacheable data blocks.

In practice, the batch size has only effect on the GPU [6]. Since the memory access overhead in the shared memory case is 2 magnitudes less than the global memory case, thus the effect is visible only in the GPU case without the shared memory. Regarding the payload dependence, the signature matching process for a specific packet ends in case of a hit in the signature database. Depending on the position of the signature in the signature database the threads finish in different times. According to [6], the threads in the same block are scheduled to run parallel thus they wait for each other to finish. This means that the slowest thread would determine the run time of the block. Therefore the more the parallel the solution is, the less variance occurs.

Summarizing the results, the GPU-based shared memory version of the proposed method introduces a performance increase of two orders of magnitudes compared to the original regular expression based method. The average packet processing speed of the GPU is approx. 1,800,000 pkts/sec. This speed can be translated into network line speed by calculating with 500 byte average packet size which results in 6.7 Gbps speed.

B. Evaluation of GPU based aggregation

We can measure the compression ratio of the aggregation as the size of the APHB divided by the size of the SPHB. When evaluating the compression ratio, the following issues should be considered:

- *Ideal compression*: All possible items are aggregated, i.e., every element occurs only once.
- *Parallel reduction*: Due to the binary tree based execution model, this solution cannot aggregate nodes from different branches of different roots. So this solution provides lower compression ratio than the ideal case by definition. For illustration see Figure 4.
- *Practical HW constraints*: The number of SPHB rows that can be aggregated in a block of the GPU is limited by the maximum number of threads per block (recommended maximum is 512) and the size of the shared memory (16 Kbytes) [6]. In our example, a row is stored on 19 bytes (2 bytes for the hash key, one for the aggregation counter and 16 for the application hit counter). Thus at most 862 rows could fit in the shared memory. But the number of leaves of the binary tree is the power of 2 and each thread examines 2 rows at the same time, so only 512 rows with 256 threads can be handled per block. I.e., if X adjacent rows could be ideally aggregated to 1 row, the GPU can aggregate it to $\lceil X/512 \rceil$ rows.

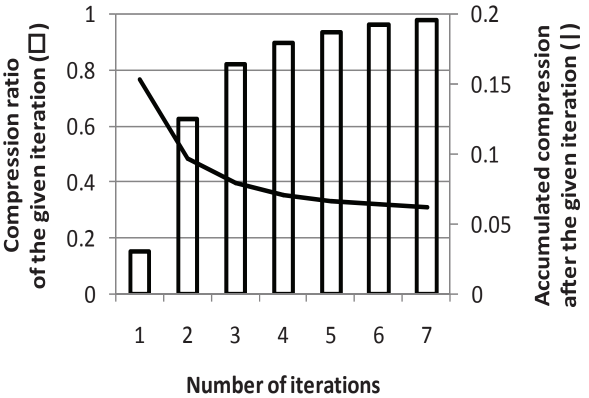


Fig. 5. The compression ratio of the GPU based aggregation as a function of execution iteration number

In an example from a live traffic trace, the SPHB contained 256,000 rows. By ideal aggregation, APHB could be aggregated into 4,560 rows meaning that the compression ratio is 1.78%. By using the GPU, the APHB could be aggregated to 41,095 rows meaning a compression ratio of 16%. However, the aggregation ratio can be improved by repeating the aggregation several times. The compression ratio as a function of the number of iterations is showed in Figure 5. It can be seen that the initial compression ratio can be approx. halved in the next few iterations resulting a total compression ratio of 8%.

The next question is the performance of the aggregation, i.e., how much time the aggregation needs. Our experiments show that the aggregation tasks of the GPU are more than 4 magnitudes faster than the DPT update done by the CPU (hash insert and update). That is, the aggregation causes no practical overhead in the overall traffic classification process.

C. Total system throughput

In this section we discuss the overall performance of the capture and classification system in case of one CPU core and one GPU. The task of the CPU is to obtain the data from the Network Interface Card (NIC), compress the data, feed the GPU with the data and create a classified flow log. The GPU is responsible for the classification.

According to our test (see Figure 6), the proposed system can classify real-time traffic at approx. 1.7 Gbps per CPU core. The components of the CPU load are as follows.

- 20% CPU load to capture the traffic: we use our own developed kernel module (similar to [30]) to avoid the built-in packet handling mechanism of the kernel. This results in a significant reduction of CPU load during the NIC read.
- 70% CPU load to create flow logs: the output of the kernel module is passed to a user level process which collects the packets into flows.

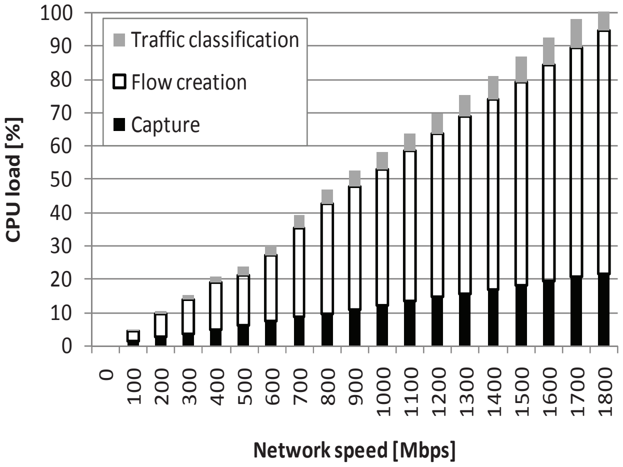


Fig. 6. Total CPU load as a function of network speed

- 10% CPU load is connected to the feeding of the GPU and process the classified data.

Note that the GPU is idle in most of the time, thus it is able to serve several traffic classification threads parallel. The current bottleneck is the CPU part of the process. With a multicore environment, the achievable speed scales up to the number of processors linearly achieving an approx. 6.7 Gbps total system throughput in a 4 core environment. For a slight comparison calculating with 500 bytes average packet length [8] and [19] achieved 0.3 and 0.6 Gbps system throughput respectively supporting only a DPI based classification.

VI. Conclusion

This paper discusses a GPU assisted solution working on multi-Gbps speeds on commodity hardware. A modification of Zobrist hashing for signature matching problem was introduced, which uses compact signatures allowing to fit the entire dictionary into fast memory of the GPU. The achievable speed was measured over 6 Gbps for a typical traffic mix. The connection pattern analysis method utilizes a parallel-prefix-sum algorithm adapted to GPU, which is four magnitudes faster than the CPU based aggregation. It was also shown, that when the above algorithms are placed in a commodity PC, the system performance including traffic capture, flow aggregation, GPU algorithms, and GPU communication allows an overall typical speed exceeding 6 Gbps.

Appendix

The measurements were done on a PC with two Dual-Core Intel Xeon Processor 5130/2.00 GHz, Intel Pro/1000 NICs, Asus ENGTX260 video card. The operation system was a 32-bit Ubuntu Linux 8.04 with 2.6.26-1-686 kernel and NAPI supported driver. The applications were compiled with `gcc-4.1`. The perl is an 5.10.0 multi-threaded version. The CUDA API used the CUDA SDK 2.0 with an NVIDIA Driver for Linux with CUDA Support (177.73) and the CUDA Toolkit for Ubuntu 7.10. The NICs were bridged by `brctl`, thus they appeared as one virtual adapter towards the system.

References

[1] T. Karagiannis, K. Papagiannaki, and M. Faloutsos, "BLINC: Multilevel Traffic Classification in the Dark," in *Proc. ACM SIGCOMM*, Philadelphia, Pennsylvania, USA, August 2005.

[2] "M. Iliofotou, P. Pappu, M. Faloutsos, M. Mitzenmacher, S. Singh, G. Varghese: Network Traffic Analysis using Traffic Dispersion Graphs (TDGs): Techniques and Hardware Implementation, Technical Report, 2007," http://www.cs.ucr.edu/~marios/Papers/UCR-CS-2007-05001.pdf.

[3] G. Szabó, I. Szabó, and D. Orincsay, "Accurate traffic classification," in *Proc. IEEE WOWMoM*, Helsinki, Finnland, June 2007.

[4] "Stephen Shankland: Google uncloaks once-secret server, 2009," http://news.cnet.com/8301-1001_3-10209580-92.html.

[5] "Penguin Computing Delivers University Of Delawares Fastest Supercomputer to Global Computing Laboratory , 2009," http://www.penguincomputing.com/press/press_releases/Delaware.

[6] "CUDA Programming Guide 2.0," http://developer.download.nvidia.com/compute/cuda/2.0-Beta2/docs/Programming_Guide_2.0beta2.pdf.

[7] A. L. Zobrist, "A hashing method with applications for game playing," *Tech. Rep. 88, Computer Sciences Department, University of Wisconsin*, 1969.

[8] "N. Goyal, J. Ormont, R. Smith, K. Sankaralingam, C. Estan: Signature Matching in Network Processing using SIMD/GPU architectures," http://www.cs.wisc.edu/techreports/2008/TR1628.pdf.

[9] M. Harris, S. Sengupta, and J. D. Owens, "Parallel prefix sum (scan) with cuda," *Hubert Nguyen, editor, GPU Gems 3*, Aug 2007.

[10] "IANA TCP and UDP port numbers," http://www.iana.org/assignments/port-numbers.

[11] S. Sen and J. Wang, "Analyzing peer-to-peer traffic across large networks," in *Proc. Second Annual ACM Internet Measurement Workshop*, November 2002.

[12] A. W. Moore and D. Zuev, "Internet Traffic Classification Using Bayesian Analysis Techniques," in *Proc. SIGMETRICS*, Banff, Alberta, Canada, June 2005.

[13] J. Cabrera, J. Gosar, W. Lee, and R. Mehra, "On the statistical distribution of processing times in network intrusion detection," in *In 43rd IEEE Conference on Decision and Control*, Dec 2004, pp. 75–80.

[14] S. Dharmapurikar, P. Krishnamurthy, T. Sproull, and J. Lockwood, "Deep packet inspection using parallel bloom filters," *Hot Interconnects*, pp. 44–51, Aug 2003.

[15] N. Weaver, V. Paxson, and J. M. Gonzalez, "The shunt: an fpga-based accelerator for network intrusion prevention," in *FPGA '07: Proceedings of the 2007 ACM/SIGDA 15th international symposium on Field programmable gate arrays*. New York, NY, USA: ACM, 2007, pp. 199–206.

[16] N.-F. Huang, H.-W. Hung, S.-H. Lai, Y.-M. Chu, and W.-Y. Tsai, "A gpu-based multiple-pattern matching algorithm for network intrusion detection systems," in *Advanced Information Networking and Applications - Workshops*, Okinawa, Japan, Mar 2008.

[17] N. Jacob and C. Brodley, "Offloading IDS Computation to the GPU," in *ACSAC 2006*, Washington, DC, USA, Dec 2006.

[18] E. Seamans and T. Alexander, "Fast virus signature matching," *GPU Gems*, vol. 3, pp. 771–783, 2007.

[19] G. Vasiliadis, S. Antonatos, M. Polychronakis, E. P. Markatos, and S. Ioannidis, "Gnort: High performance network intrusion detection using graphics processors," in *RAID '08: Proceedings of the 11th international symposium on Recent Advances in Intrusion Detection*. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 116–134.

[20] G. Vasiliadis, M. Polychronakis, S. Antonatos, E. P. Markatos, and S. Ioannidis, "Regular expression matching on graphics hardware for intrusion detection," in *RAID*, 2009, pp. 265–283.

[21] C. Wu, J. Yin, Z. Cai, E. Zhu, and J. Chen, "A hybrid parallel signature matching model for network security applications using simd gpu," in *APPT '09: Proceedings of the 8th International Symposium on Advanced Parallel Processing Technologies*. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 191–204.

[22] L. Deri, "High-speed dynamic packet filtering," in *Journal of Network and Systems Management*, vol. 15, no. 3. New York, NY, USA: Plenum Press, 2007, pp. 401–415.

[23] A. W. Moore and K. Papagiannaki, "Toward the Accurate Identification of Network Applications," in *Proc. PAM*, Boston, MA, USA, March 2005.

[24] "Traffic Measurements and Models in Multi-Service Networks (TRAMMS): Project Achievements, 2009," http://projects.celtic-initiative.org/tramms/files/TRAMMS_leaflet_final_lq.pdf.

[25] A. Papadogiannakis, D. Antoniades, M. Polychronakis, and E. P. Markatos, "Improving the performance of passive network monitoring applications using locality buffering," in *15th Annual Meeting of the IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, vol. 36, no. 2, Istanbul, Turkey, 2007.

[26] R. M. Hyatt and A. Cozzie, “The effect of hash signature collisions in a chess program,” *ICGA Journal*, vol. 28, no. 3, pp. 131–139, 2005.

[27] “D. M. Breuker: Memory versus search in games, PhD thesis, 1998–2005,” http://www.dennisbreuker.nl/thesis/thesis_pdf.zip.

[28] “Ipoque: Internet Study 2008/2009,” http://www.ipoque.com/resources/internet-studies/internet-study-2008_2009.

[29] “Application specific bit strings,” http://www.cs.ucr.edu/~tkarag/papers/strings.txt.

[30] “PF_RING, snapshot date: 2008.07,” http://www.ntop.org/PF_RING.html.

Géza Szabó Géza Szabó received the degree of Master in Computer Science in 2006 from the Budapest University of Technology and Economics, in Budapest, Hungary. His main interests include internet traffic classification and modeling. He works as a research engineer in Trafficlab of Ericsson Research Hungary and also pursues a PhD degree in the High Speed Networks Laboratory of the Budapest University of Technology and Economics.

István Gódor István Gódor received both his M.Sc. and Ph.D.degree in electrical engineering from Budapest University of Technology and Economics, Budapest, Hungary in 2000 and 2005, respectively. He is a research fellow at Ericsson Research, Traffic Analysis and Network Performance Laboratory of Ericsson Hungary. His research interests include network design, combinatorial optimization, cross-layer optimization and traffic modeling. He is a member of IEEE and the Scientific Association for Infocommunications of Hungary.

András Veres András Veres is a senior researcher at Ericsson’s Traffic Analysis and Network Performance Laboratory. His research interests include analysis of fixed and mobile networks, applications and protocols.

Szabolcs Malomsoky Szabolcs Malomsoky is the manager of Research at Ericsson Telecommunications Hungary, Budapest. His research experience covers performance analysis of and connection admission control for UMTS Radio Access Networks, traffic modeling, network dimensioning and real-time bandwidth estimation as well as passive monitoring based performance management of 3G networks. He received his PhD degree from the Budapest University of Technology and Economics in 2004. During his PhD studies he was a visiting researcher at NTT in Musashino, Tokyo.

Sándor Molnár Sándor Molnár received his M.Sc. and Ph.D. in electrical engineering from the Budapest University of Technology and Economics (BME), Budapest, Hungary, in 1991 and 1996, respectively. In 1995 he joined the Department of Telecommunications and Media Informatics, BME. He is now an Associate Professor and the principal investigator of the teletraffic research program of the High Speed Networks Laboratory. Dr. Molnár has been participated in several European research projects COST 242, COST 257, COST 279 and recently in COST IC0703 on ”Traffic Monitoring and Analysis: theory, techniques, tools and applications for the future networks”. He is a member of the IFIP TC6 WG 6.3 on ”Performance on Communication Systems”. He is participating in the review process of several top journals and is serving in the Editorial Board of the Springer Telecommunication Systems journal. He is active as a guest editor of several international journals like the ACM/Kluwer Journal on Special Topics in Mobile Networks and Applications (MONET). Dr. Molnár served on numerous technical program committees of IEEE, ITC and IFIP conferences working also as Program Chair. He was the General Chair of SIMUTOOLS 2008. He is a member of the IEEE Communications Society. Dr. Molnár has more than 130 publications in international journals and conferences (see http://hsnlab.tmit.bme.hu/~molnar/ for recent publications). His main interests include teletraffic analysis and performance evaluation of modern communication networks.