Packet Payload Inspection Classifier in the Network Flow Level

N.Kannaiya Raja, Kanniya13@hotmail.co.in
Assistant Prof/ CSE Department
Arulmigu Meenakshi Amman College of Engg,
Thiruvannamalai Dt-604410,
Tamilnadu, India

Dr. K.Arulanandam, sakthisivamkva@gmail.com
Professor/CSE Department
Ganadipathy Tulsi's Jain Engg college,
Vellore, Tamilnadu, India.

P.Umadevi, umasri05@yahoo.co.in
Assistant Prof/ CSE Department
Arulmigu Meenakshi Amman College of Engg,
Thiruvannamalai Dt-604410,
Tamilnadu, India.

D.S.Praveen, praveencse37@gmail.com
Arulmigu Meenakshi Amman College of Engg,
Thiruvannamalai Dt-604410,
Tamilnadu, India.

Abstract

The network have in the world highly congested channels and topology which was dynamically created with high risk. In this we need flow classifier to find the packet movement in the network. In this paper we have to be developed and evaluated TCP/UDP/FTP/ICMP based on payload information and port numbers and number of flags in the packet for highly flow of packets in the network. The primary motivations of this paper all the valuable protocols are used legally to process find out the end user by using payload packet inspection, and also used evaluations hypothesis testing approach. The effective use of tamper resistant flow classifier has used in one network contexts domain and developed in a different Berkeley and Cambridge, the classification and accuracy was easily found through the packet inspection by using different flags in the packets. While supervised classifier training specific to the new domain results in much better classification accuracy, we also formed a new approach to determine malicious packet and find a packet flow classifier and send correct packet to destination address.

Keywords: Flow Classification, Packet Inspection, Traffic Classification, Packet Processing, Bloom Filter.

1. INTRODUCTION

The main problems are found and generated in the online classification of observed traffic flows in to the application types. The flow classification is a problematic issue because resources used for both routing, switching and bridge. Flow refers to sequences number of packets with having five tuples. They are namely source IP address, destination IP address, source port number, destination port number and protocol. Thus eliminating the approximation.

A flow classification organizes packets with different characteristics in to different classes using certain criteria. It is the basis for providing flow classification techniques. It can be applied to network and end-host security, for e.g., consider a sequence of flow, whose "server" port number

indicates one type of application, but its features reflect another that contains a anomaly, means to indicate malicious activity. At the same time, network planners may provide the quantities and types of number of packet flows could be offered. So that the users in the network decide how to expand their network to better accommodate them. Mostly, those internet users have, as they use the network to conduct their legal terms of use of policies that may need to be enforced by the foundation for this legal administrative privacy laws and internet service providers.

Flow classification schemes containing many key elements that are shown to perform well also in terms of processing flows. From the online classification First, performance of flow classifiers we created are found by features. It is the basis for providing computationally and memory wise. However, in more general usage, the features are feasible to identify in manually and which are robust and at the same time that are re4adily susceptible to obfuscation or tampering. Deep Packet Inspection(DPI) (also called complete packet inspection and information extraction-IX) is a form of computer network packet filtering that will not examines features based on packet payload information acquired online classification. Examining packet payload information may changes the terms-use-of privacy laws and additionally, difficult to examine. Also, provide simple encryption methods (e.g., a randomized substitution cipher) can render from packet payload information not considering for classification purposes. Additionally, eliminate [1], the classifier thus developed can be eliminate, and also able to be readily tampered with one set of port numbers or protocol-specific information in the payload header information e.g., push-pkts-server feature applied in the classifier of [1].which is a technique based on computing number of packets with the push bit set in the TCP option field of the layer 4-header.

On optimizing the performance and usage of port numbers, notify the flow and congestion control mechanisms of TCP, can be extended through the application can be essentially constitutes the arbitrary port numbers and UDP. There exists a QOS e.g., interactive real time applications. However, in more general usages, to solve this, the UDP based utorrent client [2] we use the UDP Bit Torrent.

Congestion control mechanism of TCP makes it possible to find missing of packets. Due to frequently large immune system to session termination by third parties via forged TCP RST (reset) or FIN packet transmission, flow classification is complicated, when and there may be a merging of additional measures that is said to be not connection orient in layer4.

Including the major problems, the second one is the problem arising by considering the port numbers as feature problem, applications are engaging in port-number “spoofing”, present increasing problems for features which employs standard port numbers requiring to nominally allowed application types leads ensuring obfuscate their activity and allow through firewalls that block certain port number ranges (which consider unwanted Application types).assumption we can emphasizes also deleterious effects of port number spoofing on the performance of classifiers that rely on ports as features. We can also able to found and develop methodology for detecting port-number spoofing.

We illustrate this method as the most important, notably issues based on the current study that corresponds to the porting of a classifier from one domain (the first domain we use supervised machine learning to a train a classifiers) that will labels(provided if supervised training is to be well performed on the new domain), the packet traces may be employed and evaluated by having contemporary publicly available packet traces with the domains, the packet traces are recorded initially one at the Cambridge, UK [2], and the second one at the at U.C. Berkeley’s Lawrence Berkeley National Laboratory (LBNL or now just LBL) [3]. The classification accuracy of over 100% when training and testing on first domain and also that accuracy of classification is applied to the second domain.

We document the features of flow by improving the measurement of this study, illustrations with publicly available network data from multiple domains, indicating and identifying most problems endemic to work. Analyze the main goal of building classifiers that can be submitted on all such,

and the classifier accuracy needs to assess on each domain. Not only obtained network traces at different sites may not include the same applications, or may not define classes in the correct way for e.g., identifying and presenting Bit Torrent traffic at one site while other site may not involve. However, the evaluation result basically group one site mi9ght define a “Database” class, where as another might group “Database” traffic with in a larger class. In order to perform, we limited less problematic, same traffic is presented at the classes defined at the two different sites e.g., mail and “email” requires its own set of protocols and there are a variety, both for sending and for receiving mail. It is increasingly popularly the class nomenclature used at the two sites will in basically differ.

Accurately, porting a classifier is said to problematic issues source site itself, which was trained to illustrate its defined classes, that may corresponds to operate on a different *target* site, In common with classes, where there is a different set of defined classes we use in the sequel to encounter this problem might have working with the Cambridge and Berkeley traces. We plan to address it, each of the site may be defined by a set of “consensus” classes, and we attained a significance that aims to best reconcile the different sites of evaluated consensus classes used by the two sites. These two sites may be defined different sets of defined classes.

Second, it is crucial, not only for supervised classifier training but also for classifier accuracy evaluation, to have ground-truth class labels for each flow. However, classifier porting is needed in the first place because there are no labeled flows for directly training a classifier for the target site. Thus, both in practice and in this study, the absence of ground truth labels for the target site, even for evaluating classifier porting accuracy, is a problem that must be overcome. In this paper, since we study classifier porting, even if class labels are not assumed to be available for supervised classifier training at the target site, we still need labels for (test) flows from this domain in order to evaluate the classification accuracy of the classifier ported from the source site. Indeed, the Cambridge trace provided ground-truth label information for all flows which was obtained via DPI (e.g., the Wireshark tool can be used for this purpose) Unfortunately, no flow-class labels or packet payloads were provided for the Berkeley trace; thus, some procedure for establishing ground-truth was needed for this trace in order to evaluate accuracy of a classifier ported to this domain. To obtain the necessary ground truth, we applied a port-to-application mapping approach that will be detailed in the sequel. For example, absent port spoofing (which is assumed in performing this mapping for the Berkeley traces), flows with destination port numbers 25, 53, and 80 are reliably bound to the applications “email”, “dns”, and “web”, respectively.

Once we resolved the issues of heterogeneity in class definitions and missing ground truth discussed above, we were ready to evaluate the accuracy of a classifier trained on one domain (Cambridge) but now operating on another (Berkeley).It is not surprising (as will be seen by our results) that accuracy may degrade when operating in a different domain. One possible reason is that different traces may predominantly capture different traffic types.

For example, one data set may come from a traffic monitor deployed close to a mail server, with another coming from a gateway router; email traffic will not predominate the latter dataset, while the former may not have as much inters-enterprise network management traffic. In addition to differences in class priors, there may be (even subtle) differences between class-conditional feature distributions measured at two different sites, which can degrade accuracy when a classifier trained at the source site is ported to the target site.

In summary, the contributions of this article are as follows:

- a study of classifier porting from one site to another
- network flow classification based on robust features; and
- a port-spoofing detection methodology and its evaluation.

Intrusion detection systems (IDSs) that rely on packet inspection, e.g., Snort [3] and Bro, use deterministic and simple statistical signatures to determine known threats and highly suspicious

behavior, respectively. Intrusion detection based on packet payload information has also been extensively studied, e.g., detection based on "prevalent content". Some of the decisions made by network-based IDSs clearly amount to packet-flow classification decisions. Network flow classification based on a flow's statistical features was proposed in. Using, e.g., average packet size and flow duration, they applied two supervised machine learning approaches: K-nearest neighbor (KNN) and Linear Discriminant Analysis. As typically done for packet traces recorded prior to 2005, they obtained ground-truth application labels from the standard IANA port-application mapping list.

A naive Bayes classifier combined with kernel estimation and a correlation-based feature selection strategy was used in to solve offline TCP flow classification. Using twelve different features (given in Section III-B below), they achieved a classification accuracy of 96%. One difficulty with this classifier is that it is based on both port features and TCP-dependent features, both of which can be easily tampered with to trick the classifier.

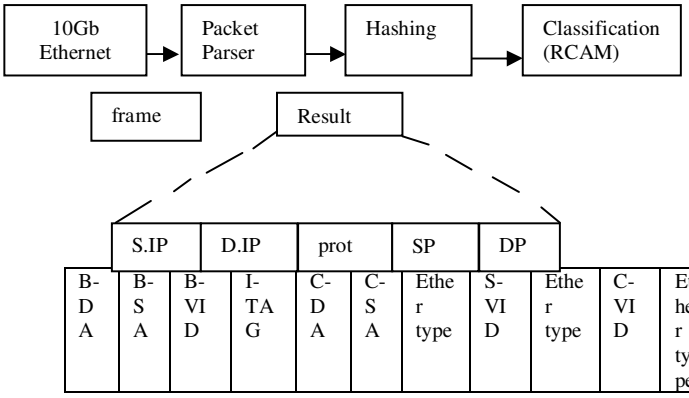
Following this investigation, reduced the complexity of the approach in to one suitable for on-line deployment by limiting the "observation window" of inspected packets per flow. This approach used C4.5 decision trees and achieved a precision greater than 92% for every application. We note that did consider classifier portability both temporally and to another (spatial) domain. However, the classifiers evaluated in used port numbers as features, which we already noted can be easily tampered with. We will demonstrate fragility of such classifiers in the presence of port spoofing in the sequel. Another limitation of is that it did not consider the case where different class definitions are used at the source and target sites. This is a genuine problem, as we note that different definitions were in fact used in defining the Cambridge and Berkeley classes. Subsequently in, the Cambridge researchers developed a ground-truth derivation tool .This tool is backed by the L7-filter, i.e., it requires layer-7 information from the packet payloads. So it has limited usage when packet payload information is not available because of laws protecting privacy, which is the case for available, publicly disseminated packet-trace data.

Recently, correlated packet-level alarms with a feature vector derived from corresponding flow-level statistics not involving payload information. Their flow-level classifier exploited ground-truth labels derived with the help of packet inspection by intrusion detection devices. Their experimental results showed little impairment of classifier performance in deployment over periods of several weeks.

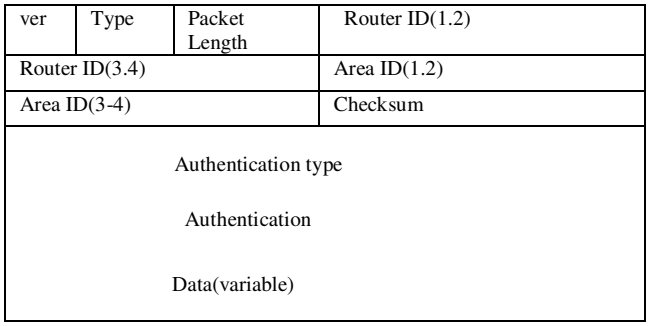
The Proposed an approach to TCP flow classification based on a flow representation using the statistical properties of an application protocol. The features they used included payload size statistics of packets composing the flows. Preliminary results for a support vector machine (SVM) classifier confirmed the effectiveness of such representation. All these previous research results demonstrate the effectiveness of classification based on extracted flow statistics; however, all these methods with the exception of, have a common

2. RELATED WORK

We briefly summarize the key ideas behind classification tools and the methodologies to test them and evaluate their performance. wherever the classifier on the network we need to change payload packet in the network eg., a,b,c are mention below.



(b)



(c)

A .Classifiers

Classifiers are defined by two main processes.

• Feature extraction: the process of extracting the subset of information that summarizes a large set of data or samples.

• Decision process: the algorithm that assigns a suitable class to an observed sample.

Examples of features are specific strings in the payload (as in DPI), packet size, or amount of exchanged bytes. Potentially, any summary of a packet stream can be used, and its choice has a deep impact on the classifier performance. In our tool, features are defined from the statistical observation of the values taken by portions of the payload.

For the decision process, any machine learning technique can be adopted. In this paper, we focus on supervised learning algorithms, in which a training set composed of known traffic is used to build a model; the model is then used during the classification task. Given a geometric representation of features in a multidimensional space, during the training phase, labeled samples are used to identify and to define the "volume" into which samples of the considered class fall. During the classification process instead, the sample to be classified has to be labeled with the most likely class according to the volume it falls into. For example, assuming that there are two classes of objects, i.e., red and yellow apples, if the features of a sample place it in a volume dense of red apples, we are inclined to classify it as a red apple, too. However, defining the surface that delimits the volumes (to later take the decision) is tricky since training points can be spread out on the multidimensional space and complex surfaces must be described. In this paper, we consider both simple geometric decision process and SVM based algorithm, which is considered to be among the most powerful supervised learning.

	Oracle Classification	
	True	False
Positive	True Positive	False Positive
Negative	False Negative	True Negative

TABLE 1.DEFINITION OF FALSE/TRUE POSITIVE AND FALSE/TRUENEGATIVE

B. Testing Methodology

Once a classifier has been designed, its performance must be evaluated and proper metrics must be defined. Assessing the performance of Internet traffic classifiers is not a trivial task due to the difficulty in knowing the "ground truth," i.e., what was the actual application that generated the traffic; for the ground truth, an "oracle" is needed. Testing the classification engine by means of artificial traffic (e.g., by generating traffic in a test bed) solves the problem of knowing the ground truth (you are the oracle), but reduces the representativeness of the experiments since synthetic traces are hardly representative of real-world traffic. Assessing the performance against traffic traces collected from operative networks is therefore mandatory. To extract the ground truth from the real traces, we developed an ad hoc oracle, based on DPI mechanisms, and we manually tuned and checked those results. However, the oracle may still be fooled.

Classification accuracy is often reported in terms of False Positive (FP) and True Positive (TP), and the False Negative (FN) and True Negative (TN). A test is said to be "True" if the classification result and the oracle are in agreement. A test is said "False" on the contrary. The result of a test is "Positive" if the classifier accepts the sample as belonging to the specific class. On the contrary, a test is "Negative." For example, consider a flow. The oracle states that this flow is an eMule flow. If the flow is classified as an eMule flow, then we have a True Positive. If not, then we have a False Negative. Consider instead a flow that is not an eMule flow according to the oracle. If the flow is classified as an eMule flow, then we have a False Positive. If not, then we have a True Negative. Table I summarizes the definitions.

The corresponding percentages must be evaluated as the following.

• False Positive percentage (%FP) is the percentage of negative samples that were erroneously reported as being positive.

$$\%\mathrm{FP} = 100\frac{\mathrm{FP}}{\text{Total Number of Negative Samples}}.$$

False Negative percentage (%FN) is the proportion of positive samples that were erroneously reported as negative.

$$\%\mathrm{FP} = 100\frac{\mathrm{FP}}{\text{Total Number of Negative Samples}}.$$

• True Positive percentage (%TP) is 100-%FN.
• True Negative percentage (%TN) is 100-%FP.

Indeed, if there are 100 e Mule flows and the classifier misses 10 of them, we have %FN=10%(%TP=90%).Similarly, if there are 500 non e Mule flows and the classifier returns all of them as eMule, we have %FP=100%(%TN=0%).

Finally, results are often expressed by means of a confusion matrix. In the field of artificial intelligence, a confusion matrix is a visualization tool typically used in supervised learning. Each column of the matrix represents the instances in a predicted class, while each row represents the instances in an actual class. One benefit of a confusion matrix is that it is easy to see if the system is confusing two classes (i.e., commonly mislabeling one as another).

In this section, we enumerate BF literature on high-power and throughput efficiencies. Also, a list of packet processing applications using BFs is discussed with advantages and disadvantages.

2.1 A Power-Efficient BF

An m-bit vector memory for a BF has k0 read ports despite the number of hash functions in a BF is k). For a pipelining scheme [5], let k/ k0 1⁄4 3 and there are three pipeline stages as shown in the example in Fig. 1. Among the three look ups in a BF, look up L1 does not need stage S2 and S3, because probing a BF in S1 reveals that a key is not a BF member. Although this prevention of unnecessary memory accesses in these stages reduces power, the pipelining scheme needs three clock cycles in the worst case. This is observed in a true or false positive lookup, when a lookup requires all the stages. Since them Â 1 memory supports only k0 read ports, the over lapped access to the memory in stage S2 for lookup L2 and stage S1 for lookup L3 causes a structural hazard, as shown in Fig. 1a. Although, having k read ports can resolve this hazard, the provision of a larger number of k read ports is not efficient in terms of the necessary hardware implementation. The other solution is to utilize "stall" twice, as shown in Fig. 1b. However, these stalls cause to decrease the throughput in processing the three lookups. In contrast, our MPC takes one clock to process a lookup, and an MPC of n BFs in a multitiering and pipelining configuration is designed to process multiple lookups in a clock cycle.

2.2 Fast Packet Classifier with n PIs

The packet classification goal is to identify a flow that is characterized with a five-tuple (source IP (SIP), destination IP (DIP), protocol, source port (SP), destination port (DP), and a protocol), and then to forward the flow to a corresponding output port. Several types of packet classifiers are suggested to meet this goal like those that are TCAM-based and SRAM-based. In a hash-based approach, a packet classifier in uses PIs in parallel, so that for a given packet lookup all PIs need to be checked in order to find the packet-associated flow and this packet is forwarded to a corresponding port where PIs returns "yes." However, in a high-speed lookup performed on a PI, the number of memory read ports in the PI can sufficiently provide a significantly low f-positive. Also, the number of PIs to be probed is as large as the number of a high-speed router's ports, and this means we need to access all PIs in a brute-force way. Unlike, the above schemes of the θ (n) PI access complexity among n PIs, our MPC demands probabilistically less complexity than θ (n)for a lookup, and this implies that we can save on power, which is otherwise consumed for unnecessary PI accesses.

In addition to the power saving through a sub θ (n) PI access complexity per lookup, our MPC also provides multiple lookup throughputs per clock cycle. Besides the PI applications used for packet processing, applications of other domains have utilized the benefit of PIs just as well, such as dynamic PI for data management, wide-area web caching [26],content delivery across overlay networks, IP traceback, and query routing in peer-to-peer networks. Even in a wireless sensor networks the power saving is a paramount issue; a coordinated packet traceback mechanism in is introduced with the concept of dimensions in hash algorithms in which a dimension can be expanded by the number of either hash functions, hash tables, or both. However, all these applications simply process one lookup to n PIs in parallel and are resulting in the θ (n) lookup complexity, while our MPC processes several lookups in a clock cycle for a high throughput.

2.3 System Architecture Design

Our scheme provides a flexible framework for hardware implementations, from memory minimized to performance maximized. In this section, we will explore the details of the hardware architecture of each part of the scheme, and will give two possible configurations for different requirements. In our memory size calculation, Snort (including more than 4000 patterns) is used as the experiment library.

A. Design of Overlapped Packet Flag Classifier

The most appropriate data structure for implementing OSC is Packet Inspection, which is very efficient for membership inquiry operations. By employing 2 PIs each for a packet flag set, we can

build a simple 2-set classifier. According to the match results of s, we can encode them to 0, 1 or x. A digest without any x is called a proper digest. To obtain a very small false positive rate (FPR), generally we still need much memory to build PIs. Fortunately, for parallel (or iterative) classifiers functioning upon overlapped packet flag, a very good property holds:

Theorem 2: Using the OSC approach, with f as the FPR of each OSC, the false positive rate of a proper generated flag digest (no "x") length of L is bounded by: $f_L \leq f^L$ .This indicates that we can build very low FPR system with relatively high FPR Bloom Filters, which will save even more memory. A possible yet rare hash collision may occur when both of the Packet Inspection in the OSC return "match", and a digest digit assignment is impossible. In case of this, a configurable rehash table can be accessed for final arbitration. The rehash function is described in depth in [5].The Stamping online-check functions can also be implemented using Packet Inspection.

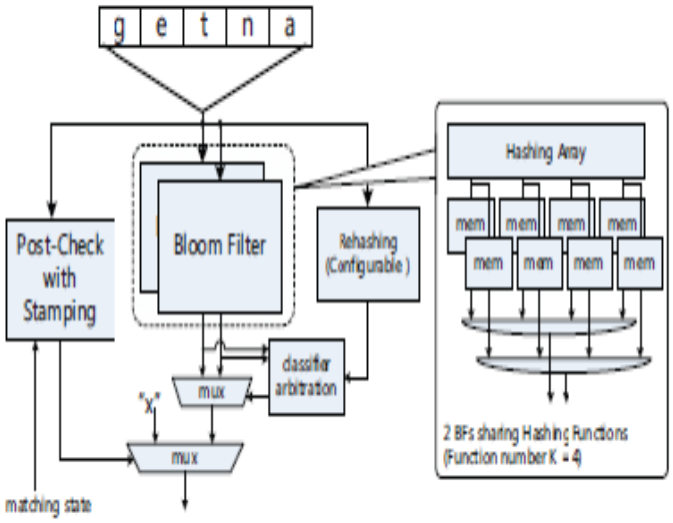


FIGURE 1. Architecture of an OSC, with on-line checker and rehashing for classification failure. Note that the 2 Packet Inspection for each set can share hash functions array.

Contribute a multiplicative factor to the whole system FPR; a relative high FPR is also acceptable, which makes its memory usage trivial. The Stamping check will be activated or deactivated according to the matched state. To combine all these techniques above, a digest encode module is illustrated in Fig.1. Note the two Packet Inspection can share the hash function array. More implementation techniques on Packet Inspection can be found in [6].

B. Matching On Flags Digest

To locate the exact match (or match set) according to the Flags digest generated using previously described approach; we can use DFA-style architecture for constant speed processing. First of all we can generate the DFA using the new" pattern set" composed of the digests of the original pattern strings. This DFA is much compact since only b-bit characters (with 2b as the set number) are accepted. The rest of the architecture design is very similar to DFA-based scheme, in which next state of the matching is fetched at every cycle from the transition memory using the current state and the input b bits as index. Optimization for this procedure has been one of the main focus of recent research; many of them are applied in our scheme [23][24][25]. Basically, it is a trivial matter to design a DFA based architecture with 1-bit processing step; and the memory consumption is also very small(with our Snort example, no more than 50KB is needed). On the other hand, building hardware for DFA which takes multiple bits as input will need more careful design. First of all, to match at all offset within a step, multiple DFAs have to be accessed; second, since wildcard "x" can occur at the end of a digest match (for instance, for digest "10010001011x", a "011x" could be matched at the last step), wildcard need well support; third, to minimize the memory usage, more sophisticated techniques need to be employed for next state

lookup.As in Fig. 2(a), transitions in a typical DFA are imbalanced. In fact, all the transitions of the DFA can be divided to three parts with degrading priorities: normal transitions, partial match transitions (with wildcards), and Fail-Back-To-Layer1 (FBTL) transitions. The isolation the FBTL transitions reduce the transition number dramatically since many original transitions are combined. In addition, we can assign a “default” transition.

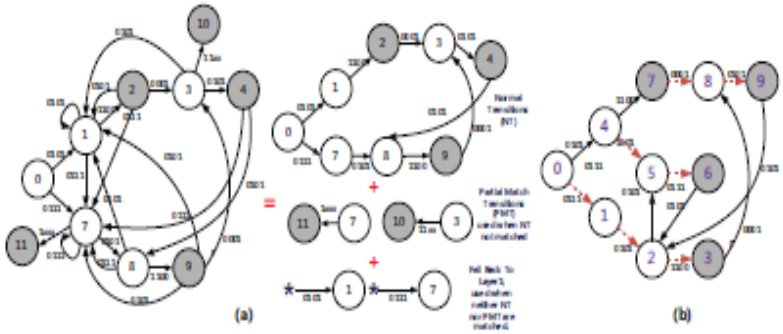


FIGURE 2. Digest DFA optimization intuitions. (a) Demonstrate how a DFA Can be detached to different parts with different priorities; note the FBTL Transitions use wildcard as starting states; (b) demonstrates that the non-leaf Node of a DFA can assign a “default” transition (the red dotted arrows), so that the next-state lookup can be omitted by using current-state + 1 instead.

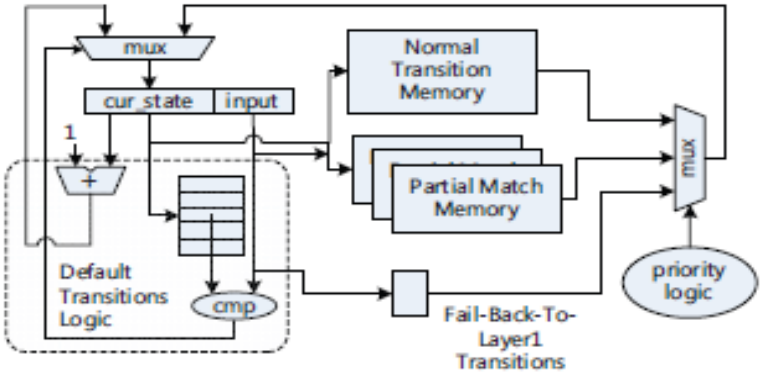


FIGURE:3 Architecture of the DFA next state logic, corresponding to Fig.2.

For each state by carefully numbering the states, as shown in Fig.2(b) so that a number of transitions are “computed” rather than “looked up”, saving more storage. Using these two techniques, Fig.3 illustrates architecture for DFA taking 4-bit step. For normal and partial match transitions, set associative style memory layout [23], as shown in Fig.4, can result in even better storage efficiency.

C.System Configurations for Memory or Performance

With the efficiently implemented digest encoder and DFA match engine, the design of the whole system is a straightforward task. We can configure the system at will, from the least memory usage to ultra high throughput.

If the memory resources are critical, the design in Fig.5 (a) can be used. With the digest step s = 2, we have two matching systems accessing the same dual-ported memory. Running at 150MHz and consuming less than 50KB memory, the system

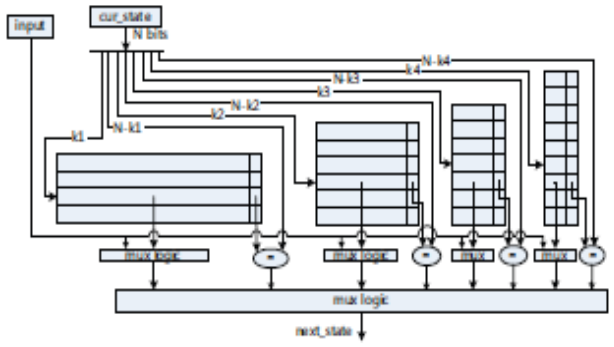


FIGURE:4.A cache - style memory access architecture for efficient memory use. States with more out-bounded transitions can reside in the "fat" memory set, while most leaf states can reside in the "thin" memory set since most of them only have one out-bounded transition. [23] details similar design.

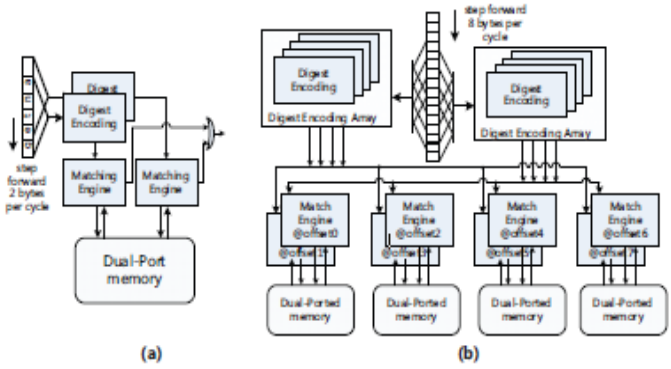


FIGURE:5 System designs with distinct requirement. (a) A 2-byte per cycle architecture; (b) A 8 byte per cycle scheme, which use multiple match engines searching on different offsets.

Has a constant scanning rate of about 2.4Gbps, which would suffice in many applications. In fact, with simple duplicate of this scheme, it would be trivial matter to design a system supporting 10Gbps aggregated rate (multi-threaded i.e. against multiple simultaneous flows).

Under ultra high single-threaded throughput requirement where multiple bytes must be processed in one cycle, both digest step larger than 1 and DFA step larger than 1 can be combined to achieve higher performance. Fig.5 (b) shows such a configuration. In this architecture, 8 bytes are processed per cycle. For the encoder part, two digest-encoder arrays each with step of 2 are generating two 4-bit digest each cycle. Also, two DFA matching units, each of which consume 4 bits of digest and contains for parallel engines, are employed. Again with dual-ported memory, 4 memory blocks can support 8-way parallel processing. Running at 150MHz and consuming about 260 KB memories, the system can support a single threaded 10Gbps scanning rate.

3. CONSTRUCTION AND RESULTS

3.1 Problem Statement of Fast Packet Classification

The issue of how to reduce the number of expensive off chip accesses through n on-chip BFs is a paramount concern in processing a packet [7], [8],[9],[10],[11] as well as network application including wireless sensor network .However, in this section, we formalize and restrict this issue to only addressable to the packet classification domain. A parallel lookup with n BFs is a common

configuration in packet processing. This is shown in Fig. 2, where a five-tuple of SIP, DIP, protocol, SP, and DP is extracted from a packet and a lookup of the five-tuple is composed among the n BFs. Fast on-chip packet processing with n BFs is beneficial, because this approach not only reduces the number of expensive off-chip hash probes but also enhance the load balance in a set of off-chip hash tables [12],[13]. Due to f-positives from the BFs, all positives are required to be confirmed by a hash table of the recorded flows. It is emphasized that providing a perfect match in the off-chip hash table is necessary in packet classification for QoS and security concern, and consequently, this produces a BFs' access that is in contention to the n hash tables. BFs can be fabricated on-chip due to their memory efficiency.

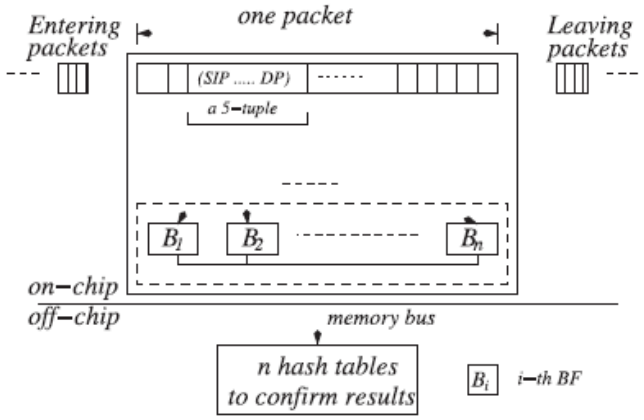


FIGURE:6 Parallel packet classifier engine of n BFs in a given packet.

Their hash tables are located off-chip due to large memory requirement as in other schemes [14],[15],[16]. In this configuration of on/off chip separation, the packet lookup throughput is bounded to the processing time in the off-chip hash table. We can calculate the worst-case throughput of a parallel packet classifier engine in Fig. 6 in the following way: Given a lookup of a minimum 40-byte packet, there are two kinds of lookups, an unsuccessful lookup (UL) in which a key is relentlessly searched although it does not exist in BFs, and a successful but time-consuming lookup (SL) in which a key is to be searched in PIs. Let t_s and t_u denote the processing times in an off-chip hash table (HT) for an SL and a UL, respectively. Then, the packet lookup throughput in n BFs is calculated as follows:

$$T = \frac{40 \cdot 8}{p_s\{t_s + t_u \cdot (n-1)f\} + (1-p_s)\{t_u \cdot nf\}} \text{bits/sec.},$$

Where p_s is an SL rate, and the nf and (n-1)f terms explain the expected numbers of f-positives, which are based on the binomial distribution of identical and independent PIs in an SL and a UL, respectively.Fig.7 shows the throughput where HT's processing time in an SL,t_s, is 1.001 times of 2 ns in a modern T-RAM and t_s is set to 0.5 times of 2 ns. In the Worst case of p_s =1, the lookup throughput with PIs of k=10 read ports shows that this configuration can barely keep up with 160 Gbps, while PIs of k =15 read ports can Meet the bandwidth requirement. Thus, a large number of read ports in a PI memory is required for obtaining a high throughput, and is also preferable for avoiding accessing irrelevant PIs of such a large number of ports for a lookup is preferable. In the following section, we present the aforementioned avoidance with an MPC by distributing lookups through small-sized BFs of a few ports, so that a subset of the lookups is processed in large-sized BFs in one clock cycle for higher power and throughput efficiencies.

3.2 A MULTITIERED PACKET CLASSIFIER WITH n BFS

In this section, we first present a basic theory behind a BF and an f-positive. We then introduce the steps to build an MPC.

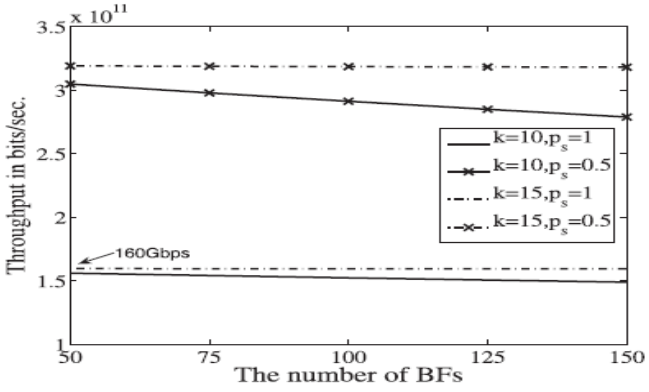


FIGURE:7 Throughput comparison with a different number of BFs, p_s, and k.

3.3 Packet Inspection Theory

A legacy PI for representing a set S of n_i items (or keys) is described by an m-bit array memory with each initially set to 0. A PI uses k independent hash functions $h_0; \ldots ; h_{k-1}$ within the range of [0: m _ 1]. For mathematical convenience, we make a natural assumption that these hash functions map each key in the universe to a random number uniform over the range as the authors [17] claim. For insertion of each key e_j', the bits indexed by $h_k'(e_j')$ are set to 1 for $0 \leq k' \leq k-1$, $0 \leq j' \leq n_i - 1$. To query that key e0 is in S, k bits by k memory reads through $hk'(e')$ should all be 1. If that is the case, a PI returns "yes" about a query of key e0. If that is not the case, then clearly e0 is not a member of S. Even if a PI returns "yes," there exists a probability of an *f*-positive, such that key y is falsely believed to belong to set S due to the random gathering of k bits of value 1 set by independent keys.

The above probability f of an f-positive can be formulated in a straightforward way, given our assumption that hash functions are perfectly random. Among m bits, the chance of a bit being value 0 by one h_k is 1/m. After all n_i elements of S are hashed k times into the PI, i.e., totaling $k.n_i$ times, the probability that a specific bit is still 0 is asymptotically $p = (1-1/m)^{kn_i} \approx e^{-kn_i/m}$. Then, the probability of an f-positive by randomly choosing k bits among m bits is

$$f \geq \{1-(1-1/m)^{kn_4}\}^k \varepsilon (1-p)^k \geq (1/2)^{m \ln 2/n_4}$$

This probability is bounded, and the optimal k, the number of hash functions that minimizes f, is easily found $k = \ln 2\ (m/n_i)$. After some algebraic manipulation, it is clear that the requirement of $f \leq \epsilon = 2^{-w}$ where w is called lookup precision, suggests

$$m \geq n_i \log_2(1/\varepsilon)/\ln 2 \approx 1.44 n_i \log_2(1/\varepsilon) = 1.44 n_i w.$$

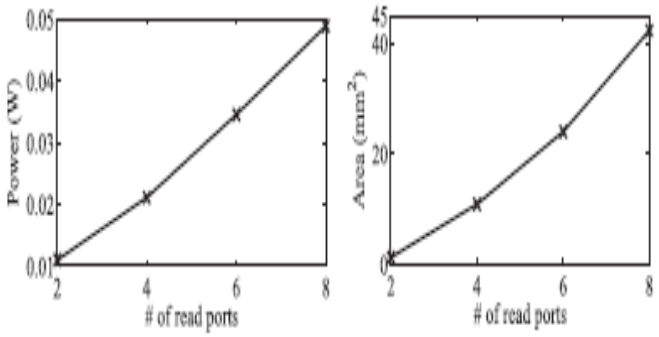


FIGURE:8 Power and area in multi memory read ports for 64K _1 bit memory.

From (7), the following important lemma can be derived: Lemma 1 (Linear Property). Linear property between m and n exists in (7) because given f requires that variable n_i is linearly proportionate to variable m. Furthermore, in an optimal configuration, k becomes w according to the following derivation:

$$k = \ln 2 \frac{m}{n_i} = \ln 2 \left(n_i \frac{\log_2(1/f)}{\ln 2} \right) / n_i = w, \qquad (4)$$

and to be a scheme of a deterministic O(1) lookup processing 500 M packets a second for a 160 Gbps router, k needs to be at least 29 (_log2 1=500 M).Each hash function corresponds to one random lookup in an m-bit PI. Thus, a PI having k hash functions for high throughput needs the exact same k number of memory read ports in an m-bit memory module. Although, the state-of the art VLSI technology can fabricate memory modules with multiple ports, supporting more than ten ports is tremendously challenging to implement, as noted in a concise summary of the recent embedded memory technologies [18]. Fig. 8 shows such a difficulty in terms of the power and area costs measured by CACTI [19], according to the number of read ports in a single memory module. The conclusion from the figure is that the power and area costs are super linear with respect to the number of read ports. Thus, a PI is considered as a high computation element due to the large value of k for the high-speed router, and, thereby, reconfiguring such PIs for a power and throughput-efficient lookup is proven necessary.

3.4 Basic Principles of an MPC

Fig.9 shows the basic principles in constructing an MPC, and this layout demonstrates how an MPC is superior in Power efficiency than a PPC. Suppose, there are four PIs in a PPC, as shown in Fig. 9a and each PI is equipped with k memory read ports. In this parallel configuration, we need to access k bits in each PI and the access is performed in one Clock cycle. Thus, the PPC's lookup throughput is one per clock cycle and the PPC needs a power for 4k-bit PI memory access, in order to process one lookup.

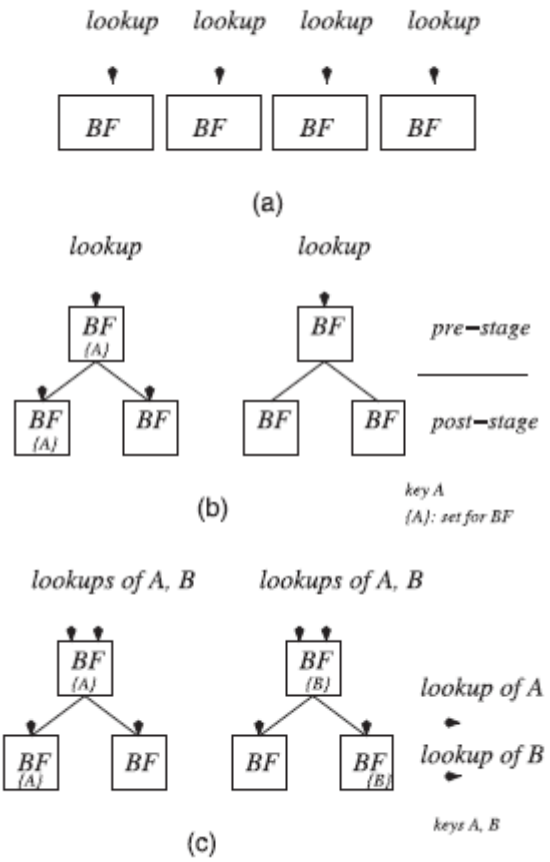


FIGURE: 9 Basic principle and two benefits (i.e., power and throughput) of an MPC in on-chip memory. (a) PPC configuration. (b) MPC configuration for power efficiency. (c) MPC configuration for throughput efficiency.

In contrast, our MPC can reduce the aforementioned power usage by probing only a subset of k bits in a PI. Suppose, there are two smaller PIs of one read port and we put them in the prestage of four larger PIs of k _ 1 read ports, as shown in Fig. 9b. Then, we conceptually connect each smaller PI in a prestage to two larger PIs in a poststage via a tree relationship. That is, if a smaller PI (or a parent PI) returns a positive in a lookup, we need to probe two larger PIs (or children PIs) that are connected to the smaller PI. Suppose, a key A is encoded in a smaller PI and a larger PI, as shown in Fig. 5b, and we search for the key from the prestage. Since there is no false negative, a PI, which encodes the key A, should return “yes” in the key lookup. The second smaller PI in the prestage may return “yes” with a false positive, and its probability is 1/2 based on (2). If there is no false positive in the second smaller PI, then the total number of probed bits in our MPC is 1+ 1 +2(k _ 1). Even if there is a false positive, the bit count is (1+1+4(K–1). Thus, our MPC configuration requires additional power to probe 2k bits if no false Positive shows up, and 4k – 2 bits if it does. On average, our MPC probes 3k–1(=2k·1⁄2+4k –2·1⁄2) bits in order to process a lookup while a PPC probes 4k bits, confirming that our MPC can reduce the power usage for 4k-bit memory access in a PPC.

In addition to the power saving, our MPC can increase the lookup throughput by using dual read ports for the two smaller PIs in the prestage, as shown in Fig. 9c. Suppose, we encode keys A and B into the first and fourth larger PIs in the poststage and into both smaller PIs in the prestage. Next, we assign one read port for a lookup A and the other read port for a lookup B. Since we can process two lookups in two smaller PIs in one clock cycle, we can place two lookups for keys A

and B in the four larger PIs during the next clock cycle under the condition that there is no false positive in the smaller PIs. Thus, at the complementary probability of an f-positive, i.e., 1/2, our MPC can increase the lookup throughput.

In a nutshell, our MPC reduces the power required for PI memory access by preprocessing a lookup with the smaller PIs in the prestage and by confirming the lookup with the larger PIs in the poststage. Note that if we increase the number of read ports in smaller PIs in the prestage, we can further minimize the power consumption, since the f-positive probability decreases according to (9).

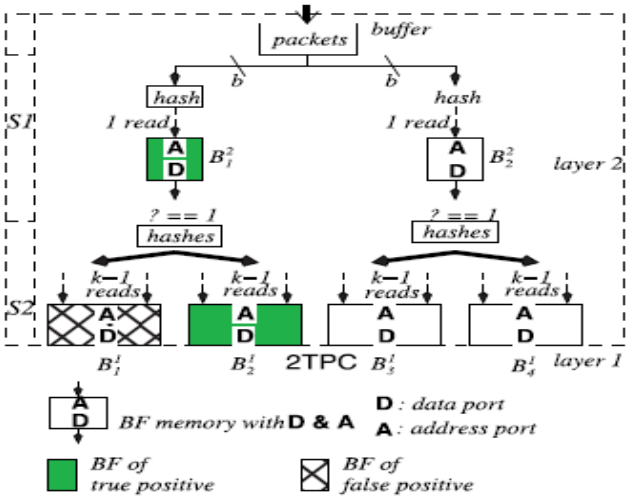


FIGURE: 10. Pipeline memory architecture of a 2TPC in a forest. S1 and S2 are pipeline stages. B_j^i means the *j*- th BF at layer *i*.n= 4.K= w due to (4). $W_{2=}$ 1, $w_{1=}$k –1.b is a buffer size.

3.5 Building a Multitiered Packet Classifier

Fig. 6 shows the detailed configuration example of an MPC, a two-tiered PC (2TPC) built on top of 4 PIs; this is in place of a PPC used in a dashed box of Fig. 10. Letters A and D denote the address and data ports in a PI memory, respectively. A PI in the layer 2, i.e., the prestage, has one read port while a PI in the layer 1, i.e., the poststage, has k– 1 read ports. Since, we organize an MPC in a pipeline configuration; we can access two PIs in stage S2, if a parent PI in stage S1 returns “yes” in a lookup. Similarly, we follow the same lookup steps in a three-tiered PC (3TPC), which is constructed on top of eight PIs, as shown in Fig. 11.Note that all small-sized PIs in S1 and S2 have one read port, while the large-sized PIs in S3 have k –2 read ports, and these setups are purposely built this way in order to make a fair memory comparison with a PPC with eight BFs of k read ports.

In addition to these two architecture examples, we derive one mathematical proof that an MPC uses the same memory size as that of a PPC in a general case. For example, given desired f-positive $f = 2^{-w}$, the total PPC memory in bits with n PIs is n·m, where m is a BF’s memory based on (3). However, with the linear property between m and n_i an additive operation on memory size m_t, we can reconfigure PIs in an (r+1)-tiered way, r> 0, while the same memory size, m_M, for an MPC is used as follows:

$$n \times m = n \times \{1.44.n_i.\log_2(1/f)\}$$

$$= n \times \{1.44.n_i.w\} = n \times \{1.44.n_i.(w\text{-}r + r)\}$$

$$= n.144.\ n_i.(w\text{-}r) + \sum_{t=1}^{r} \{n.\ 1.44.n_i.1\}$$

$$= \sum_{i=1}^{n} (1.44.\ n_i.(w\text{-}r)) + \sum_{t=1}^{r} \sum_{i=1}^{n/2} (1.44.(2^t n_i).1)$$

$$= m_1 + \sum_{t=1}^{r} mt + 1 = m_{M,}$$

where mt is the total memory of PIs on layer t, r+1 is the number of tiers, $2^t n_i$ is the number of keys in , and the lookup precisions of a PI on layer 1 and t, w1, and wt, are w–r and 1, respectively. Based on (3), the f-positives of PIs on layer 1 and 2 in a 3TPC are expected to be $2^{-(w-2)}$ and 2^{-1}, respectively, and the second term in (5a) is the sum of small-sized PIs from layer 2 to layer r+1. Also, a PI from layer 1 covers n_i elements, and a PI from layer 2 covers $2n_i$ keys. In general, B_i^j covers all keys from, $1 \le i \le n/2, 1 < j \le r$ in an MPC.In this multitiered and pipelined configuration with b ¼ 1, power in accessing memory (or probing BFs) can be eliminated. For example, B12 has a key, and there is a lookup.

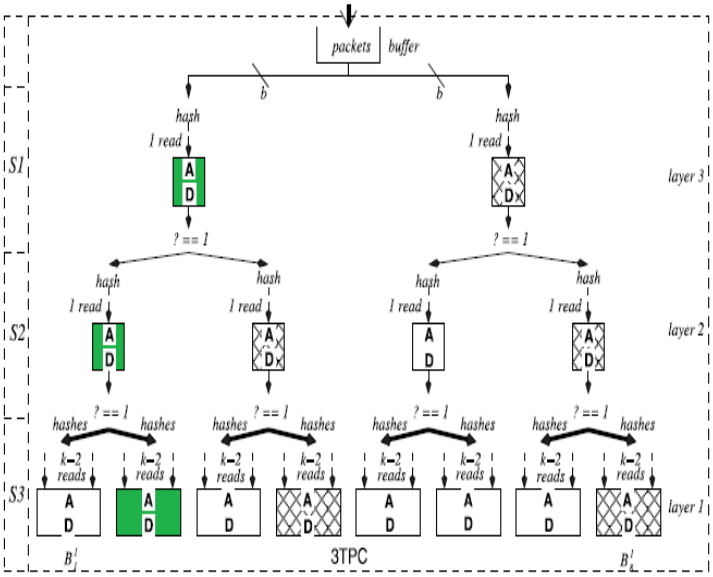
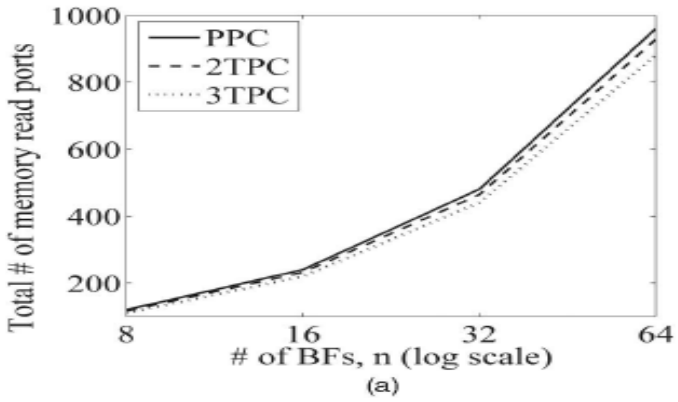


FIGURE: 11 Memory architecture of a 3TPC in a forest and in pipeline. Bij means the j-th BF at layer i. n ¼ 8. k ¼ w due to (10).



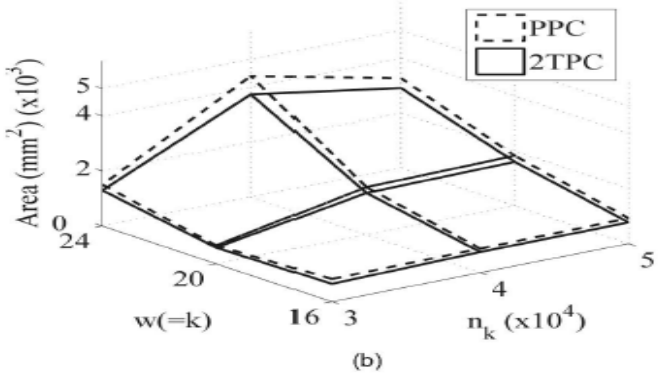


FIGURE: 12 (a) The total number of read ports in different number of BFs. w3=w2=1, w1=3 for a 2TPC.w2=1, w1=14 for a 2TPC. f = 2^{-15}.(b)2TPC and PPC die area costs with n= 8 in.13μm process technology. (a) The read port number.(b)The area cost (w2 ¼).

Corresponding to the key.By preprocessing the lookup in stage S1 with B21 and B22, if B22 returns "no" in the lookup there is no need to probe B13 and B14. Thus, the power used to probe them can be saved.In addition to the power concern, we design a throughput efficient scheme in an MPC configuration. However, the higher throughput efficiency cannot be achieved in this setup simply setting b to a value greater than 1. Although (5)'s derivation shows that an MPC has the same memory size as a PPC, but processing a lookup in small-sized PIs of one read port does not provide a higher throughput in large-sized PIs on a lower layer. For instance, even if b in Fig. 6 with w2 =1 is set to two, a one-read-port PI on layer 2 cannot process two lookups in one cycle. Thus, the number of read ports in the small-sized PI needs to be the same as b. In general, the number needs to be b·w2 for a throughput-efficient MPC. As suggested in [21], using mini- PIs with fewer read ports is the solution without degrading lookup accuracy. However, even if a PI is broken into several mini-PIs, the total number of read ports in the mini-PIs is the same as that of a PPC. Thus, breaking a PI into mini-PIs only gives the possibility of fabricating PIs for packet processing, but does not incur the benefit of high throughput. However, our MPC has two benefits of fewer numbers of read ports and an area cost reduction, which can lead to fabricate small-sized PIs of multiread ports for a high throughput without introducing area overhead.Figs. 12a and 12b show such two benefits: the smaller number of fabricated read ports and the smaller die area for a 2TPC. Fig. 12a shows the required read port numbers in fabricating different numbers of BFs for a PPC, a 2TPC, and a 3TPC, respectively. In fabricating, a 2TPC and a 3TPC use 4 percent and 10 percent less read port count than a PPC in all cases. Fig. 12b shows 2TPC and PPC area costs in different numbers of w and ni, and the area costs using four mini-PIs for a PI in each case using CACTI model [22] are measured.

Now, we show how to fabricate multiport in a small sized PI without incurring hardware overhead. There is a noticeable gap between dotted and solid meshes in Fig. 12b, and the reason is that fabricating multiports in a small-sized memory does not require area as much as in a large-sized memory. In the figure, there is a small area increase for the multiport memory, compared to a PPC's area. Thus, it is explained that the buffer size b can amount to five at most.Also, utilizing dual reads on falling and rising edges in a clock [20] can double the memory read capacity and a lookup throughput (i.e., double data rate scheme implemented in DRAM and AMD Athlon64). Thus, the buffer size becomes twice larger and the maximum b is 10 without incurring the memory overhead in an MPC.

4. CONCLUSION

This paper we have developed and evaluated TCP/UDP/FTP/ICMP and payload information and port numbers. The primary motivations of our study are easily abfuscated by the enduser through the payload encryption and port spoofing, we proposed and experiment the evaluated hypothesis

testing approach , finally and most important we have discussed the issue of porting and packet classifier from one domain to another and also investigated the performance of tamper resistant classifier in the network domain. It was found that the classifier accuracy was highly graded the new domain and sources of performance loss identifier while supervised classifier training in the new domain result in much better classifier accuracy.

REFERENCES

[1] W. Li and A. Moore. A machine learning approach for efficient traffic classification.In *Proc.IEEE MASCOTS*, 2007.

[2] utorrent.http://utorrent.en.softonic.com/.

[3] Wireshark Wireshark go deep.http ://www. wireshark.org/.

[4] "Skype testbed traces," [Online]. Available: http://tstat.tlc.polito.it/ traces-skype.shtml

[5] A. W. Moore and D. Zuev, "Internet traffic classification using bayesian analysis techniques" in *Proc.ACM SIGMETRICS*, Banff, Canada,Jun. 2005, pp. 50–60.

[6] N. Cristianini and J. Shawe-Taylor*, An Introduction to Support Vector Machines and Kernel-Based Learning Methods*. New York: Cambridge Univ. Press, 1999.

[7] S. Dharmapurikar and J. Lockwood, "Fast and Scalable Pattern Matching for Network Intrusion Detection Systems," IEEE J. Selected Areas in Comm., vol. 24, no. 10, pp.1781-1792, Oct. 2006.

[8] W.-C.F.F.Chang and K. Li,"ApproximateCaches for Packet Classific ati on ,"Proc.IEEE INFOCOM '04, vol. 4, pp. 2196-2207, Mar. 2004.

[9] H.C. Deke Guo, J. Wu, and X. Luo,"Theory and NetworkApplications of Dynamic Bloom Filters,"Proc.IEEE INFOCOM'06,pp. 1233-1242, 2006.

[10] M.Waldvogel, G.Varghese, J.Turner, and B.Plattner,"Scalable High Speed IP Routing Lookups,"Proc.ACM SIGCOMM'97,pp. 25-36, 1997.

[11] I. Kaya and T. Kocak, "Energy-Efficient Pipelined Bloom Filters for Network Intrusion Detection," Proc. IEEE Int'l Conf. Comm.,pp. 2382-2387, 2006.

[12] J.T. Sailesh Kumar and P. Crowley,Peacock Hashing: Deterministic and Updatable Hashing for High Performance Networking,"Proc. IEEE INFOCOM '08, pp. 101-105, 2008.

[13] S.Kumar and P.Crowley,"Segmented Hash:An Efficient Hash Table Implementation for High Performance Networking Subsystems," Proc. ACM Symp. Architecture for Networking and Comm.Systems (ANCS '05), pp. 91-103, 2005

[14] S. Dharmapurikar, P. Krishnamurthy, and D.E. Taylor, "Longest Prefix Matching Using Bloom Filters," Proc. SIGCOMM '03, pp. 201-212, 2003.

[15] T.S.SarangDharmapurikar,P.Krishnamurthy,andJ.Lockwood,"Deep Packet Inspection Using Parallel Bloom Filters,"Proc. 37th Ann. AC M/IEEEI nt'lSymp . Microarchitectuepp.52-61, 2004.

[16] H. Song, J. Turner, and S. Dharmapurikar, "Packet Classification Using Coarse-Grained Tuple Spaces," Proc. ACM/IEEE Symp.Architecture for Networking and Comm. Systems (ANCS '06), pp. 41-50, 2006.

[17] M. Mitzenmacher and S. Vadhan, "Why Simple Hash Functions Work: Exploiting the Entropy in a Data Stream," Proc. 19th Ann.ACM-SIAMSymp.Discrete Algorithms (SODA '08), pp. 746-755, 2008.

[18] B. Dipert, "Special Purpose SRAM Smooth the Ride," June 1999.

[19] ACTIhttp://www.hpl.hp.co.uk/personal/Norm an_Jouppi/cacti5.html,2010

[20] D.A. Patterson and J.L. Hennessy, Computer Architecture: A Quantitative Approach,Morgan Kaufmann Publishers Inc., 1990.

[21] S. Dharmapurikar, P. Krishnamurthy,and D.E. Taylor, "Longest Prefix Matching Using Bloom Filters," Proc. SIGCOMM'03, pp. 201-212, 2003.

[22] CACTI.http://www.hpl.hp.co.uk/persl/Nor an Jouppi/ cacti5.html, 2010.

[23] Tian Song, Wei Zhang, Dongsheng Wang,Yibo Xue, A Memory Efficient Multiple Pattern Matching Architecture for Network Security.Proceedings of IEEE Infocom, 2008

[24] Jan.van.Lunteren,*High performance pattern matching for intrusion detection*.Proceedings of Infocom'06, 2006.

[25] Hongbin Lu, Kai Zheng, Bin Liu, Xin Zhang, and Yunhao Liu, *A Memory-Efficient Parallel String Matching Architecture for High- Speed Intrusion Detection*. IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS,VOL.24,NO.10,OCTOBER 2006

[26] L. Fan, P. Cao, J. Almeida, and A.Z. Broder, "Summary Cache: A Scalable Wide-Area Web Cache Sharing Protocol," IEEE/ACM Trans. Networking, vol. 8, no. 3, pp. 281-293,June 2000.