Compact TCAM: Flow Entry Compaction in TCAM for Power Aware SDN

Kalapriya Kannan[1] and Subhasis Banerjee[2]

[1] IBM Research, INDIA, kalapriya@in.ibm.com
[2] IIIT-Delhi, New Delhi, INDIA, subhasis@iiitd.ac.in

Abstract. High throughput access to large data structures for lookup and classification have made Ternary Content Addressable Memory (TCAM) indispensable in today's network switching devices. TCAMs offer single cycle lookup operation but at the expense of notoriously high power dissipation. While there is no suitable alternative to TCAM for maintaining line rate lookup, high power dissipation of TCAM have raised alarms in the growing power sensitive data centers. Moreover, these memories are expensive compared to other memory devices and have direct impact on cost of switching devices. The situation further worsens with the emerging network frameworks such as Software Defined Networks (SDN). SDN's large sized flow entries (15 field tuple - that requires 356 bits to define a flow) can potentially support large number of flows and consequently an SDN device will need large TCAM size to accommodate them. An SDN switching device would require 5 to 7 times the TCAM space to hold the same number of flow entries as a Layer-2 switching device. Therefore, as expected, high TCAM size in SDN enabled network switches will take a toll on power budget.
In this paper we propose *Compact TCAM*, an approach that reduces the size of the flow entries thereby managing large sized and volume of SDN flows without adding significant silicon real estate in TCAM. We use shorter tags for identifying flows than the original number of bits used to store the flow entries. We in turn leverage the dynamically programming capability of SDN to route the packets using these tags. This reduces the power dissipation per flow. We show that our approach can be easily implemented using the new SDN framework while optimizing the TCAM space an impact of which can be realized in cost and power saving. Our experiments both with real world and synthetic traffic shows that the power can be reduced on a average by 80% in SDN switching devices for a given number of flows supported by an SDN switch.

1 Introduction

The growing demand for high throughput and packet processing at line rates have made TCAMs part of every high performance network switches. To give an example of its capability, a commercially available TCAM chip in a switching device [5] can integrate 18 M-bit (configured as 500k entries $\times$ 36 bit per entry) into a single chip working at 133 MHz, which means it can perform up to 133 M lookup per second, an order that is unimaginable to be achieved with other storage devices. While TCAMs become indispensable due to their performance benefits, it comes with associated cost. TCAM

has the disadvantage of high cost-to-density ratio (US$350 for a 1 M-bit chip) and high-power consumption (about 15 Watt/1 Mbit). For these reasons, although not restrictive, TCAMs have been limited to wild card storages. In the past it has been shown that majority of the power consumption in the switching fabric (other than the chassis, active ports etc.) is due to the power hungry memory operations especially the TCAMs [2]. Recent study indicates that the cost of power consumed by the networking devices alone is around 6 Terra Watts converting to about 1 billion US dollars per year [11]. The growing demand of power conservation by the switching devices (primarily in the data center environment) motivates us to revisit power optimization in TCAMs that is an integral component of high performance switches. In the past it has been shown that majority of the power consumption in the switching fabric (other than the chassis, active ports etc.) is due to the power hungry memory operations especially the TCAMs [2]. Recent study indicates that the cost of power consumed by the networking devices alone is around 6 Terra Watts converting to about 1 billion US dollars per year [11]. The growing demand of power conservation by the switching devices (primarily in the data center environment) motivates us to revisit power optimization in TCAMs that is an integral component of high performance switches.

Power optimization in high performance network switches has always been a priority and recent introduction and adoption of SDN has reemphasized the need of the same. SDN, an emerging network framework, is designed as the next generation network architecture specifically in Data Centers (DC). SDN Features, such as dynamically programming the switching elements, micro-flow classification of incoming traffic; are expected to overcome several of the limitations posed in existing networks. Although it offers several benefits over the present network in terms of providing flexibility, it comes with a cost at the switching devices. To depict the SDN's perspective: SDN redefines flows to be identified by a 15 field tuple (unlike in IP where it is defined by 2-5 field tuples depending on the layer they belong to). This mandates each flow entry in TCAM table to be 356 bits (15 fields) which is 7 times more than that of an L2 switching device (conventional IP switch). This implies that the bit-length of flow entries stored in TCAM will now be 7 times more than that of an L2 switch. This impacts SDN switches in several ways. Firstly, it requires higher sized TCAM for storing the same number of flows that an L2 switch can hold. Secondary, this can quickly fill up the flow tables (with micro-flows being identified as flows) and lead to flow table explosion. Recent studies have established that the flow arrival to the SDN switches can be of the order of 72,000 flows/sec. Thirdly, the number of operations which have been mostly the lookup operations, will now be converted into the insert operations as there will be more misses in TCAM table. The combined effect all these results in higher power dissipation and longer access latency, many times more compared to existing TCAM in high performance switches.

In this paper we exploit SDN's functionality that leads to reduction of required bit for identifying a flow in TCAM. We introduce the notion of Flow Identifiers (Flow-ID) as replacement for the flow entries. These Flow-ID's are pushed on to the headers with the help of SDN. The switches store only the Flow-ID's in the table rather than storing the entire 15 field tuple per flow entry. All operations in the TCAM are performed on this shorter Flow-ID and the power gain are proportional to the bit size saving.

Our approach requires no modification to the network framework and all our steps are within the framework of the SDN. We show that by using our approach about 80% gain on power can be achieved and larger number of flows can be accommodated with the the same sized flow table. Unless otherwise stated, we consider DC Networks (by referring them as networks) in our explanation and experiments as SDN is likely to be more adoptable in the DC networks. However, our approach can be used in any network that comply with SDN framework.

Our contribution can be summarized as follows: (i) We investigate the impact of SDN implementation on TCAM based high performance network switches. We use OpenFlow as the implementation standard of SDN and characterize the power consumption. (ii) We propose Compact TCAM, an approach to reduce the size of flow entries in OpenFlow (SDN) implementation. The reduction in flow entry size in turn yields power saving due to shorter search line size in TCAM.

The rest of the paper is organized as follows: in section 2 we provide brief background on TCAM and set the motivation of our work in section 3. A simple power model for estimating power dissipation in switches is given in section 4. Our approach of Flow-ID based flow entry condensation is presented in section 5. We present assessment and evaluation of our approach in section 6. We describe the prior work in section 7. Finally we conclude in section 8.

2 Background Work

Fig. 1 shows a conceptual model of TCAM in a switch. Usually a combination of TCAM and SRAM is used to forward a packet based on appropriate actions. The TCAM is a fully associative array of bits which are used for storing flow entries. Any packet arriving to the switch will undergo lookup operation to find a match in TCAM. The incoming packet may contain wildcard entries in the address field which may lead to multiple matches in TCAM. In such cases, the entry that matches with longest prefix is selected. This logic is in built along with the TCAM cells by incorporating a priority encoder. Adjacent to TCAM, there is an array of SRAM that contains the action part to guide the packet through appropriate port in the switch.

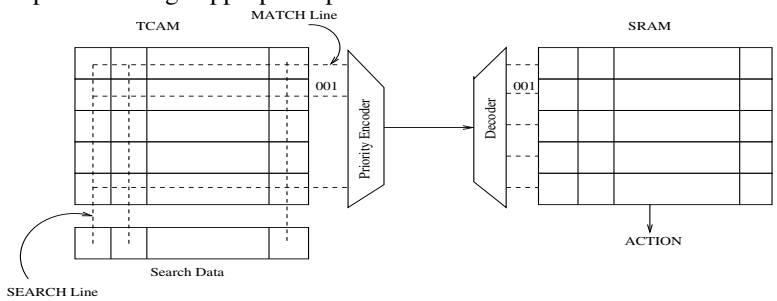


Fig. 1: TCAM/SRAM operation in a switch

An alternative to TCAM/SRAM lookup is to use dedicated processing of packet through ASIP with local caching of data in SRAM. This requires an efficient algorithm to find longest prefix match. There are few attempts to organize data cache for such implementation to improve lookup time [17]. However, this cannot match the line rate

as there are multiple access to fetch the entire data element from the cache and therefore we focus on TCAM/SRAM.

Current day switches have TCAM of size ranging from 1Mbit - 2Mbit. This indicates that current sizes of TCAM is insufficient to hold the number of active entries required for an OpenFlow enabled switch. In general commercial switching devices employ two kinds of memory cells, one a SRAM only chip for storing full entries (also referred to as 'Exact Match Entries') which can be looked up using a direct hashing algorithm and another a TCAM/SRAM combination which exploits don't care state to store the wild card entries. The SRAM consumes about 50 times less power with 20 times more access latency compared to the TCAM. Since in SDN environment (e.g., DCs) low latency lookup is preferred due to the nature and criticality of the applications, TCAM wins over SRAM even for exact match entries.

TCAM operations are primarily categorized into read and write operations and each of these operations are associated with different power consumption. Basic Flow entry operations are insert, delete, modify and search. Using the TCAM power simulator [1], average power dissipation for 100 k entry TCAM in a L2 switch consumes $195,167nJ$ and $447,065nJ$ respectively for write/insert and search/lookup operation. The estimated power number for an OpenFlow switch having same TCAM size is $792,677nJ$ and $1,807,142nJ$.

Table 1: Number of flow table entries

Arrival rate (IAT distribution)	Switch	# of machine	# of Flows	Comments
Weibull	TOR switch	40	72,000 flows/min	20% of flows are generated with IAT of 10 μ sec and 70% with IAT of 35 μ secs
Weibull	Aggregate switch	40	$\approx$ 1,00,000 flows/min	about 70-80% of the traffic is contained within TOR's
Weibull	TOR switch	40 * 15 VM's/host	$\approx$ 1050000 flows/min	Volume of flows is increased by the times equivalent to number of VM's
Weibull	Aggregate switch	40 *15 VM's/host	$\approx$ 1,30,000 flows/min	about 70-80% of the traffic is contained within TOR's
Median arrival rate	TOR Switch	40	78,000 flows/min	Median arrival rate of 30 μ secs
Median arrival rate	Aggregate switch	40	$\approx$ 90,000 flows/min	about 70-80% of the traffic is contained within TOR's

3 Motivation: Impact of SDN on OpenFlow Enabled Switches in Data Centers

We first highlight the salient features of the SDN networks and its impact on the lookup/flow tables. We use OpenFlow as implementation standard of SDN and use OpenFlow and SDN interchangeably. OpenFlow characterizes a flow using 15 field tuple. The design philosophy behind choosing the large size tuple is to attain fine grained traffic classification. Traffic classification in existing switching devices had been coarse grained. For instance, majority of them employs per destination based traffic classification and therefore the flow/lookup table consisted of entries equivalent to the number of the destination ports. In this case one can conclude that only few flow entries (as many as number of destination ports) can classify all the incoming traffic. Therefore only limited fields are required in the tuple to describe a flow. On the other side, best case resource utilization is shown to be achieved at the level of classifying individual packets as unique flow

and dynamically routing them (as done in Equal Cost Multi-Path routing (ECMP) [16]). Clearly, per destination based approaches *cannot* achieve efficient resource utilization. Categorizing individual packets into unique flow is extremely difficult to implement due to high overhead. The next best traffic categorization can be defined at the level of flows that categorize lesser traffic into a flow called 'micro-flows'. OpenFlow achieves this by defining larger sized tuples. One can certainly build a macro-flow from several micro-flow by replacing exact bit pattern (modifying flow tuple) with wild card, but this attempt to condense the flow entries into macro flows will only limit the benefits of SDN to large extent. In an attempt to retain this property of fine grained flow definition we encourage "exact match full sized" entry in the flow table.

There are two primary design challenges for a OpenFlow switch. First, the volume of flows recognizable by the switches explodes. Theoretically, 2^{356} flow entries are possible at any point in time. This is prohibitively high number for any hardware to manage. However, the actual number of flow entries and the operations performed on them are directed by several parameters of the network such as number of ports in switching devices, actions, hosts, flows etc. The practical number is something around 1 million, although it is quite high compared to the conventional switch. Second, the nature of operations in the OpenFlow switches are quite different. In conventional switches, the number of flows being small, one would expect majority of the traffic arriving at the switch to be categorized into one of the flows and the operation typically performed (and kept alive) is a lookup operation. With OpenFlow, the scenario is expected to change. The number of flows being higher, the number of possible entries to be stored is large. Therefore, operations would be more than just lookup given the fact that flow table size is limited. Operations such as writes, invalidations, update of flow table entries will now become prominent apart from just lookup.

To understand the stress experienced by the OpenFlow switches in an SDN network, we walk through an DC network and study the volume of flow entries generated and the nature of the operations performed. In this work, we mainly concentrate on the DCs primarily for two reasons: (a) their characteristics are well studied and (b) SDN has already been adopted in DC environments. We simulate the behavior of the switches for specific topologies of DCs to perform these studies.

Flow table explosion: Our aim is to show the number of flow entries in a switch that can be generated per second for the flow characteristics described in [3] and [7]. We take an example of a simplistic DC network topology. We use a Fat Tree topology proposed in [15]. The topology consists of three layers: Top of the Rack (ToR), aggregate and core switches (Fig. 5.a shows the fat tree topology). In general a three stage fat tree built from k-port switches can support non-blocking communication among $k^3/4$ end hosts using $5k^2/4$ individual k-port switches [15]. As an example consider a layout of 11,520-port data center network. In this example, the data center consists of 24 rows, each with 12 racks. Each rack has 40 machines interconnected by a ToR switch. In a non-virtualized environment this could mean just 11,520 machines, but with virtualization each machine can hosts minimum of 15 VM's thereby increasing the number of end-hosts to approximately 172000 nodes. The number of flows arriving at the TOR is given by : *flow arrival rate* $*$ *# of servers* $*$ *# of VM* (considering bridged mode in VM). Table 1 shows the number of flows arriving at ToR for different arrival rates. It can be seen

that the volume of flows arriving at a ToR switch can be in the order of 75,000-100,000 flows/min for a rack consisting of 40 machines and the same would be of the order of 1.3 million for the machines hosting around 15 VM/host.

Given the current day TCAM sizes (1 Mbit), the number of 356-bit entry (a flow entry size in SDN network consisting of 15 field tuple) that can be stored is around 2800. This is too small a number compared the number of active flows arriving at a switch.

TCAM operation overhead due to SDN: Another issue imposed by the deployment of the OpenFlow is the increase in power budget for the nature of operations performed on the TCAM devices. Operations in the TCAM of OpenFlow switch are predominantly insert or lookup operations. Table 2 provides an overview of the nature and number of operations performed in a L2 switching device and SDN switching device assuming a TCAM that can hold about 2800 entries, for the same data center characteristics as the above study. These numbers were roughly estimated using the characteristics of packet arrival at switches shown in [3] and sample size of VM under a ToR to be 40 using 1 Mbit TCAM on the switching device. It can be seen that the number of insert operations changes due to the flows getting generated in high volume. Considering the fact that the power consumption of the insert operations in the TCAM is significantly high (section 2) it can be seen that the overall power consumption in TCAM is likely to increase by multiple folds.

Table 2: Nature of the TCAM operations

Switch type	Nature of operations	# of operations in L2	# of operations in SDN
ToR	Insert operations	40 (once)	4000/sec
ToR	Lookup operations	39000/sec	35000 /sec

4 Power Model for Switching Fabric of a OpenFlow switch

In this work we are interested in modeling the impact of the header processing on the switching fabric. Power dissipation by the switching fabric is a function of operations and the underlying hardware/software associated with performing these operations. In order to understand the functional units of a switching device we refer to a pipeline reference architecture provided in [14] . Fig. 2 presents the pipeline architecture of a switching device with both functional units and the associated hardware. Functionally it consists of a header parser that extracts specific bits from the header, a input arbiter that direct these bits to SRAM/TCAM for lookup operations, a mediator for prioritizing the output of lookup operations for appropriate actions. The packet is further edited if required by the packet editor and forwarded to the output port. Switches incorporate several off-the-shelf hardware components of varying technology to achieve these operations in line speed. For lookup operations, hardware flow tables are used. Flow tables stores flow entries with varying size depending on the protocol enabled on the switching device. For instance an L2 switching device would use 60 bit sized flow entries (48 bit destination MAC address + 12 bit VLAN tag) and an OpenFlow enabled switch will

utilize 356 bits for storing flow entries. We model the power consumption for different functional units of a switch which is described below.

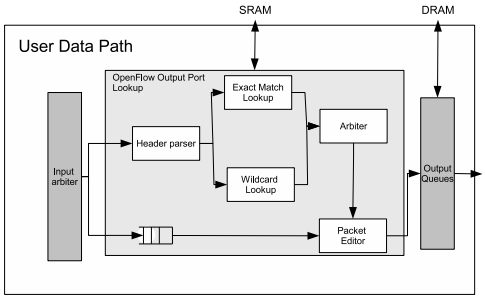


Fig. 2: Reference pipeline of a switching device as shown in [12]

1. Header parser (P_{hp}): The parts of header is extracted from the packet through a hardware unit primarily consisting of few 'select' lines and 'multiplexers'. The packet is typically stored in a RAM and the power consumed for this operation is dependent on the number of bits retrieved from the header. It is denoted by: $P_{hp} = P(S_{bitline})$, where $S_{bitline}$ is the size of the bit line for the selected fields from the header.
2. Lookup Operations (P_{lp}): The extracted fields from the header are used to perform a lookup operation for a matching entry in the flow table. Two kinds of tables are often used in implementing a hardware table: SRAM's and TCAM's.
   A combination of SRAM and TCAM is typically used in modern day switches to store tables. Power is dependent on the nature of operation performed on these tables and is either a write or search operation. Therefore the total power consumption can be written as follows:

$$P_{lp} = E_{s-search}N_{lookup} + E_{s-wr}N_{insert} + E_{t-search}N_{lookup} + E_{t-wr}N_{insert} \quad (1)$$

   where N_{lookup} and N_{insert} are the number of lookup and insert operations performed per second, $E_{s-search}$ and E_{s-wr} are the energy associated with search and write of an entry in the SRAM, $E_{t-search}$ and E_{t-wr} are the energy associated with search and write of an entry in TCAM. SRAM and TCAM search and write operations are functions of bitline size (number of cell columns) and word length (number of rows).
3. Forwarding to output ports (P_{fp}): Forwarding to output ports includes transmission of bits to output ports. This is dependent on the number of bits to transfer to output port.
   $P_{fp} = P_{bit\_transmission} * N_{bits}$
   The total power consumed by a packet processing is equivalent to the summation of the power consumed by the packet on the individual switches and is given as follows:

$$P_{total} = (P_{hp} + P_{lp} + P_{fp}) * N_{switches} \quad (2)$$

where $N_{switches}$ denotes number of switches along the path of the flow. In Equation 2, contribution by P_{hp} and P_{fp} is insignificant compared to the power dissipated by P_{lp}. Power drawn by a memory access (read/write) is few thousands time more than that of a register read/write. P_{fp} is constant for fixed throughput and the overall power gain remains unchanged (assuming that the switch will be sending at a fixed throughput). Therefore we focus our effort in measuring P_{lp} in our results.

5 Compacting Flow Entries using Flow-ID

The flow entries in a table classify incoming packets into different flows. Once a flow entry is identified, subsequent packets belonging to the flow need not match the entire flow entry. The size of the entry can be now compacted and replaced with a shorter ID for classification. Hence we introduce the notion of "Flow-ID". Flow-ID is a numeric identifier used to identify the flows uniquely. In the switch flow table, the flow entry can be reduced to the size of the 'Flow-ID' and associated actions. Table 3 show an example mapping of flow entires to their corresponding Flow-ID's. All routings are performed based on the Flow-ID's.

SDN's programming capability is leveraged to take advantage of the Flow-ID. Packets are identified to belong to a flow based on the Flow-ID and actions corresponding to the flow is applied on the packets. This requires that the headers of the packet carries the Flow-ID information to be able for the switches to identify the flow to which the packet belongs. Switches perform lookup operation based on the Flow-ID rather than full sized flow tuple extracted from the header. Fig. 3 shows the SDN architecture consisting of the controller, OpenFlow switches and host machines.

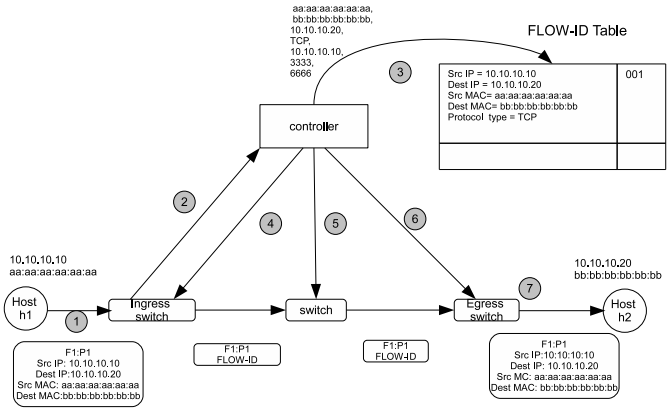


Fig. 3: System overview showing *Flow-ID* creation, utilization through different steps.

In OpenFlow the first packet of a new flow arriving at an interface of a switch (Step 1 of Fig. 3)is forwarded to the controller (Step 2). The controller generates a *Flow-ID* for the flow and stores it in its local reference table as illustrated in Step 3. The

controller responds back to the switch with an OpenFlow action message consisting of the actions to be performed on the packets of the flow as demonstrated in Step 4 of the Figure. In our work, we augment the 'action' part of the rule with a boolean 'COMPACT' flag. If the flag is clear, the switch performs the standard actions on the packets. For instance, the flag is set to be clear if the destination address is connected to the same switch. Otherwise (COMPACT flag set), the switch utilizes the Flow-ID and performs operations on the packets corresponding to the Flow-ID and output it to specific port as given in the action part.

Compact operation consists inserting the Flow-ID into the header of the packet. Leaving aside the 'preamble' and 'start of delimiter' fields in the Ethernet Frame, the Flow-ID is stored in 2 bytes of the frame. The 'preamble' and the 'start of delimiters' are used by the link layers for purpose of determining frame boundaries and are left intact. In OpenFlow enabled switches, the 'PUSH' allows addition of bits of data to specified segments of the packet. We use the 'PUSH' tag of the OpenFlow architecture to add a new header on to the packet containing just the 'Flow-ID'. The original header and packet payload is encapsulated within this header.

The controller sends a flow insert operation to all the switches along the path of the packet (Step 5 of Fig. 3). This flow entry consists of a "Flow-ID" and associated actions to it. When a packet arrives at the intermediate switches, the 'Flow-ID' is extracted from the header and is matched against the entries in the flow table (proactively installed by the controller on the switches). These switches do not perform any COMPACT operations and do not require any additional hardware. They perform the normal operations of a OpenFlow switch except that they extract the Flow-ID's from the header and search for a matching entry in the lookup tables.

For the egress switches, the controller sends a flow insert operation that contains the "Flow-ID" and actions to remove the 'Flow-ID' from the header of the packets. Step 6 in Figure illustrates the flow insert operation specified by the controller to the egress switch. The header that was in the original packet withouth the Flow-ID is forwarded to the output port to deliver to the end hosts (Step 7).

Example Demonstration: In Fig. 3 let us assume that host $h1$ initiates a communication (a flow) to $h2$. The figure also shows a sample packet ($F1 : P1$) being generated from $h1$ to $h2$. For a flow, the Ingres Switch is the switch by which the packet enters into the network and Egress Switch (ES) is the switch by which it exits out of the network. Table 4 shows the fields of the packet with the values populated. For sake of brevity we only show a few fields that are common to a flow such as src IP and MAC, dest IP and MAC and few fields such as IP Checksum and Identification number that are unique to a packet.

Table 3: Example mapping of flow information to *FLOW-ID*

Flow information	Flow-ID
in_port = 1, dl_vlan = 0xffff, dl_vlan_pcp = 0x00,dl _src = 00:00:00:00:00:0b, dl_dst = 00:00:00:00:00:0c, nw_src = 10.0.0.11, nw_dst = 10.0.0.12, icmp_type = 8, icmp_code = 0, actions = output:2	001
in_port = 2,dl_vlan = 0xffff, dl_vlan_pcp = 0x00, dl_src = 00:00:00:00:00:0c, dl_dst = 00:00:00:00:00:0b, nw_src = 10.0.0.12,nw_dst = 10.0.0.11, icmp_type = 0, icmp_code = 0, actions = output:1	002

We refer to this packet as ‘f1:p1’. At the Ingress Switch (IS), the selected fields from the header of ‘f1:p1’ is extracted and is used for locating an matching entry in the flow tables. As this is the first packet of a new flow, the lookup will result in a miss and the packet is forwarded to the controller. The controller extracts the flow entry (Src and Dst IP addresses, Src and Dst TCP Port) and generates a corresponding Flow-ID ‘001’. The controller sends a flow insert operation to the IS along with ‘COMPACT’ bit set. Fig. 5 shows the flow entry in the IS.

Table 4: Fields in a packet

Src MAC = aa:aa:aa:aa:aa:aa	Dst MAC= bb:bb:bb:bb:bb:bb
.....	
Src IP = 10.10.10.10	Dst IP = 10.10.10.20
IP checksum	IP Identification field
.....	
Src Port= 5555	Dst Port = 6666
TCP checksum	TCP sequence number
payload	

Table 5: Flow entry in the ingress switch

Flow entry in the IS
cookie=0, duration_sec=0s, duration_nsec=0s, table_id=1, priority=32768, n_packets=0, n_bytes=0, idle_timeout=60,hard_timeout=0,in_port=1, nw_src =10.10.10.10, nw_dst=10.10.10.20,dl_src=aa:aa:aa:aa:aa:aa, dl_dst=bb:bb:bb:bb:bb:bb,ether_type=ip, actions = ’COMPACT’, mod_dl_dst[]:0x00001,output:3

Bit size of the ’Flow-ID’ is set to hold maximum number of active flows at any point in time. As shown in [3] the number of active flows is about 10000. Therefore, number of bits for storing Flow-ID can be set to 16 bits (Byte aligned). This is a static value and is incremented by the controller for each incoming new flow. The flows are stored at the controller along with its mapping to ‘Flow-ID’ in a map table (shown in fig.6).

Table 6: Outgoing packet

Src MAC = (16 bits) 0x001	Dst MAC= bb:bb:bb:bb:bb:bb
.....	
Src IP = 10.10.10.10	Dst IP = 10.10.10.20
IP checksum	IP Identification field
.....	
Src Port= 5555	Dst Port = 6666
TCP checksum	TCP sequence number
payload	

In SDN, controller is assumed to have complete knowledge of the network state and so are the paths for the flows. The controller also sends the ‘Flow-ID’ along with the actions to all the switches on the path except the ES. The flow entry in the intermediate switches is shown below. It should be noted that as the Ethernet header field (Dst MAC address) will carry the *Flow-ID*, the match is set against the Dst MAC address. Although the MAC address is 6 bytes, the packet header parser should be modified to extract just the *Flow-ID* bits (shown in fig.7).

Table 7: Flow entry in the ingress switch

Flow information in the intermediate switches
cookie=0, duration_sec=0s, duration_nsec=0s, table_id=1, priority=32768, n_packets=0, n_bytes=0, idle_timeout=60,hard_timeout=0,in_port=1, dl_SRC = 0x001, actions=output:3
Flow information in the ES
cookie=0, duration_sec=0s, duration_nsec=0s, table_id=1, priority=32768, n_packets=0, n_bytes=0, idle_timeout=60,hard_timeout=0,in_port=1, dl_SRC = 0x001, actions= 'RECONSTRUCT', mod_src_dst='aa:aa:aa:aa:aa:aa', mod_dl_dst:'bb:bb:bb:bb:bb:bb', mod_nw_src='10.10.10.10', mod_nw_dst='10.10.10.20',output:3

Table 8: Flow entry in the Egress switch

When an intermediate switch receives a packet with the header containing a Flow-ID, it extracts the Flow-ID from the header and performs a lookup operation on the flow tables for Flow-ID match. As this will result in a hit the actions associated with the entries are applied to the packet. All intermediate routings are performed based on the Flow-ID's.

The flow information shown in fig. 8 is inserted by the controller at the ES. When a packet arrives at an ES, the packet have to be reconstructed before dispatching it to the end-host. The packet is reconstructed with the information that has been extracted by the controller at the Ingress (other bits remains intact and is carried in the header along the path) and made available at the ES. A packet similar to the one originated by the source host is dispatched to the end-hosts.

Table 9: Characteristics of DC traffic used for simulation on the topologies

Flow characteristic	Range of values/Distribution		
	Cloud DC	Enterprise DC	University DC
Flow Inter arrival time	80% $<$ 1ms	2-3% $<$ 10μs and 80% $<$ 1 ms	4 ms - 40 ms
Flow duration	80% $<$ 11 secs	80% $<$ 10-11 secs	80% $<$ 10-11 secs
Flow of traffic	80% within racks	40-90% leave the rack	60-90% leave the rack
Flow sizes	$<$ 10 KB	$<$ 10 KB	$<$ 10 KB
Packet Size	40-30% $<$ 200 Bytes	30 % $<$ 1 Byte, 30-40% $<$ 200 Bytes, 30-40% $>$ 1200 Bytes	30 % $<$ 1 Byte, 30-40% $<$ 200 Bytes, 30-40% $>$ 1200 Bytes
Packet inter arrival times	Weibull	Lognormal	Weibull

5.1 COMPACT Hardware

Fig. 4 illustrates the implementation of Compact TCAM micro-architecture as part of switching device (refer to the pipeline reference architecture in Fig. 2). As described in the previous section, the approach operates only on packets for which 'COMPACT' flag is set. The hardware consists of a buffer that temporarily holds the packet along with the complete header. The assigned Flow-ID is copied at the selected bits of the buffer (2 bytes following the frame delimiter) and the packet is dispatched through the output port. The power consumption by these multiplexer are negligible compared to the hardware flow table operations. Packets arriving to the switching devices pass through the packet header parser which will now extract the 'Flow-ID' bits that is instructed through the bit selection logic.

Intermediate switches / core switches contain TCAM array that has entries with Flow-ID. A query to these TCAMs will require fewer bits to search in parallel. These are the switches where we reap maximum benefit on TCAM power saving due to shorter header size.

An edge switch acts both as ingress for one set of flow and egress for another set of flows. When a packet arrives, the header either contains a Flow-ID (as an egress) or a full header (as an ingress). This will require two flow tables, one with Flow-IDs and another with unmodified flow entries. We propose to use two separate TCAMs (Egress

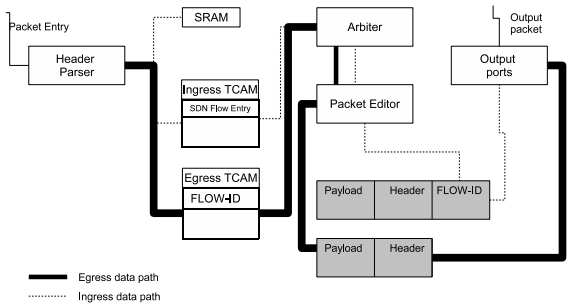


Fig. 4: Compact TCAM hardware in context of reference switch architecture given in Fig. 2

TCAM and Ingress TCAM as shown in Fig. 4) each capable of storing half of the target number of entries. One TCAM stores the Flow-ID and associated action and the other stores the complete flow entry with complete header along with COMPACT flag. The edge switch as directed by the controller either performs a COMPACT operations (where the 'Flow-ID' is inserted when the packet contains the full header) or remove operation (for those packets containing Flow-ID [ES]) where the extra information added by the 'PUSH' tag is removed. When a edge switch receives the packet, a special bit is used to identify whether the packet contains the 'Flow-ID' or the usual full header. Due to this separate TCAM's power gains on the edge switches are lesser than 50% than those obtained from the core switches.

Table 10: Number of operations performed in a bin of 1 sec in a L2 layer switch

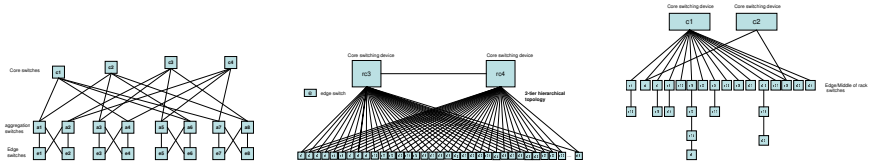
Packet category	Operation type	Number of occurrence					
		Cloud DC		Enterprise DC		University DC	
		Edge Switches	Core Switches	Edge Switches	Core Switches	Edge Switches	Core Switches
packet of an existing flow	Lookup	$\approx$ 35000-40000	$\approx$ 19000-21000	$<$ 20000	$<$ 50000	$<$ 20000	$<$ 50000
Packet of new flow	Insert	$\approx$ 1200-1300	$\approx$ 800-850	$<$ 900	$\approx$ 1200	$<$ 900	$\approx$ 1200

6 Assessment and Evaluation

Our objective is to measure the following: (a) Power gain due to the reduced number of bits for storing the Flow-ID in the hardware tables and operations performed on it, and (b) Overall cost reduction. Measurement of power gain requires fine grained flow level data traces both at the switches and the end hosts. In [4] it has been established that such fine grained flow level detail and related data traces are difficult to obtain due to expensive instrumentation required on the servers/switches. We resort to simulating DC traffic given the flow characteristics of real world DC's. For all our experiments we consider the switch-to-switch measurements.

6.1 Simulation Setting

We consider three configurations for our experiments: (a) L2 based switching device consisting of 60 bit (48 bit for source MAC + 12 bit for VLAN tag) flow entry, (b)



(a) Cloud DC - Fat Tree topology(ref:[6])

(b) Enterprise DC- Multi rooted 2 tier(ref: PRV1 in [3])

(c) University DC- Multi rooted multi tier(ref: Univ 2 of [3])

Fig. 5: DC topologies used for simulation

OpenFlow standard based switching device consisting of 356 bits flow entries (15 tuple flow entries) and (c) Compact based switching device consisting of 16 bits flow entries. We have considered three different topologies (Fat tree, 2-tier multi rooted tree, multi-tiered multi rooted tree) representing diverse set of DC environments such as and Cloud DC, Enterprise DC and University DC respectively. Fat tree topology has been recently proposed in [6] and is being viewed as widely adoptable for Cloud DC's. 2-tier and multi-tier single/multi rooted trees are widely used in todays DC's such as Universities and Enterprises. Fig. 5 (a), (b), (c) show a Fat-Tree, multi-rooted 2-tier and Multi-tier topologies that we have used in our simulation.

Traffic characteristics for various DC's have been suitably adopted from the existing studies presented in [3, 7] and is presented in Table 9. Our simulator is a discrete event simulator implemented in Java. It generates flows according to the DC traffic characteristics presented in Table 9. Flows are generated with the following tuple $<$*flow id, src, destination,duration, startTime, endTime, flow sizes, packet sizes, flow length, flow of traffic, start switch, end switch*$>$. The start and the end switches are dependent on the chosen topology. The distribution of traffic is dependent on the particular DC environment they represent. We collect the statistics of bin granularity of one sec.

6.2 Power Gain

Here our goal is to study two important effects: (a) power saving in core and edge switches which is measured by generating the flows/packets arriving at a switch for bin unit of 1 sec and using the flow and packet arrival distribution given in the Table 9. In this case the topology of the traffic less important but the 'Flow of Traffic' (refer Table 9) is important to obtain these numbers. (b) Power saving per flow which is measured by generating certain number of flows that follows arrival distribution as shown in the flow statistics table and associating a path for these flows in the specified topology (topology plays important role here). For determining the paths we consider the 'Flow of traffic' to determine the probability of a flow leaving the rack or targeted within the rack. It should be noted that the power gain on the switches are measured based on the flow arrival as observed on the switches while the power saving per flows concentrates on overall power saving along the path of the flows.

Power gain on switches: According to Equation 2 we need the number of packets that require lookup and insert operations. We fix the number of switches to one (either representing the core or a edge switch). The first packet of a Flow is considered to

trigger a miss, resulting in a corresponding flow entry being inserted into the hardware table. Table 10 shows the number of packets arriving at a switch representing a new flow (first packet of a new flow) and that of an existing flow.

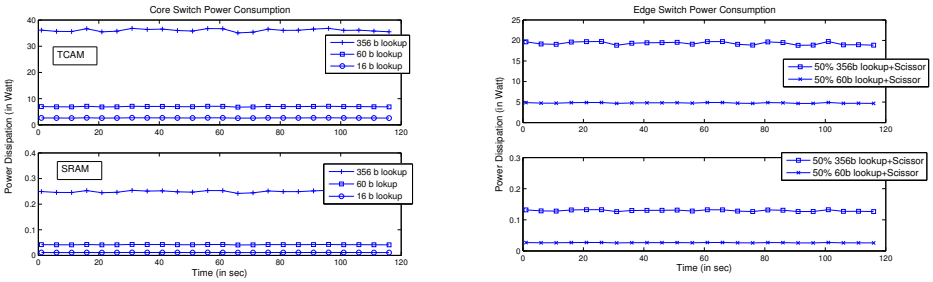
We use TCAM modeling tool available in [1] to observe the power consumption for different sizes. This TCAM power model is based Cacti[22] and have been widely used in the research community for measuring power consumption on the network devices [2]. In order to normalize the number of entries across the three categorize of switches (L2, SDN and Compact TCAM based) we fix the number of flow entries across all the three to a arbitrary higher value and measure the power saving due to the bit line reduction. In our experiments we assume about 100000 entries (as our aim is not to simulate the misses) and measure the power consumption for varying sizes of bit-line. We use traffic characteristics of 3 different DC environments (shown in Table 10) to simulate the TCAM power consumption. Table 11 shows the power consumed by various operations in TCAM for 16-bit line used by the *Compact TCAM*. As obvious, the power consumed to perform a write/insert operation with 16 bits is about 4 times lower than the 60 bit (L2 switch) and 15 times lower than the 356 bit entry (OpenFlow switch). A lookup/search operation with 16 bits can consume about 4 times lesser than 60 bit and 13 times lesser than the 356 bit entry.

Table 11: Power dissipation comparison for TCAM operations: L2, OpenFlow and Compact TCAM

No. of entries = 100,000			
Operations	L2 Sw. (60 bit)	OpenFlow Sw. (356 bit)	**Using Compact TCAM** (16 bit) (in nJ)
Write/Insert	195167	792677	**56614**
Search/Lookup	447065	1807142	**131685**

Fig. 6 (a) presents the power consumption (according to Equation 2) in terms of Watts on the core switches and Fig. 6 (b) show the power consumption at edge switches for Enterprise and Univ DC's. At the core switches, a L2 switching fabric would consume about 2.5 times more power and a OpenFlow enabled switch will consume about 12 times more compared to a *Compact* switch when the number of flows were observed in the range of 1200-1300 flows/sec. Therefore, in core switches Compact switch gives best power saving. This gain are lower in the edge switches as the power saving is obtained only for the egress flows. It can be seen that by employing *Compact TCAM* approximately 80% of the power can be saved on the core and about 15% power saving in the edge switches.

Power gain on flows: Our aim is to understand the per flow consumptions/savings by using the *Compact TCAM*. Fig. 7 shows the average power consumption for the flows considering different topology. On a average about $2700 flows$ were generated every bin. For each of these flows we associate a path with probability determined by the nature of the flow of traffic in a particular DC environment and the number of switches as given in the topology (Fig. 5). In the university topology, by using this approach savings can be achieved upto 30-40% compared to a 60 bit L2 switching device and about 87% compared to a 356 bit line SDN switching. This huge saving is primary due to the fact that in university data Centers the amount of traffic that leaves the rack and travels through the network is about 80% ie., increasing the number of hops visited by



(a) Power consumption at the core switches for Enterprise and Univ DC's)

(b) Power consumption at the edge switches for Enterprise and Univ DC's

Fig. 6: Power consumption at the core and edge switches

the flows. The saving are lesser (about 10% compared to L2 switching) in a enterprise DC setting. This is due to the fact that majority of the traffic (about 80%) is intra rack and therefore it uses regular L2 switch of 60 bit entry (the plot uses a default L2 switch with 60 bit entry). If the default switch is 356 bit rather than the L2 switching device, the power consumption observed was nearing the 356 bit values only saving in about 20% of the flows that leaving the rack. In the fat tree topology the same have been observed as the number of flows leaving the rack is considered to be 50% as cloud DC's have a mix of application that have inter rack communications and applications (web servers) facing the clients. In such a scenario the saving is again about 10% of that compared to a L2 switching device and about 83% compared to a SDN switching device.

6.3 Cost Saving

Apart from the cost saving due to reduced power consumption, another important metric to consider in the DCs is the cost of the switching devices. An important fall out of this optimization is the reduction in the size of the flow entry. This implies that by applying Compact TCAM one can accommodate more number of flow entries in the table compared to the one that does not apply the COMPACT operations. For instance, a 2Mbit TCAM can accommodate around 33,000 flow entries of size of 60 bit while it can accommodate 125,000 entries of 16 bits. TCAM are expensive. 2Mbit TCAM chip costs about 500 USD and cost of 1Mbit TCAM chip is 50% of the cost of 2Mbit chip [9]. Our approach provides significant gain in power and capital cost.

7 Related Work

We classify the existing literature along three broad categories and compare our work. They are: (a) algorithmic optimizations (b) reducing the amount of information stored in the high performance memory(c)architectural solutions. We provide a overview of the work along these three categories and compare our approach with them.

Algorithmic approaches for faster lookup's: As an alternative to TCAM's [8] [21] etc., provide fast algorithms for computing hashes. IPstash [8] exploits the fact that full

associativity is not necessary for IP-lookup. In [21] proposes an algorithm for longest prefix matching where prefixes are represented as trees and the tree is traversed according to a given key. These techniques reduce a set of arbitrary length prefixes to a predefined set of lengths. By this reduction, the number of depths in the prefix tree for traversal is reduced. We differentiate our work from these existing work along two dimensions. Firstly, these algorithms assume prefixes and propose algorithms to optimize the traversal, storage and lookup of these prefixes. In SDN networks, there is lesser incentive (although not ruled out completely) for storing prefixes. SDN is designed to provide the flexibility of fine grained traffic management by defining the concept of 'micro-flows' by increasing the number of tuples. Secondly, these algorithms suffer from unacceptably slow updating (insert delete) of the forwarding table. In SDN as insert operations are likely to be dominant (1 in every three packets can result in a insert operation [13]) these algorithms are less scalable.

Reducing information in TCAM: Reducing the number of entries in TCAM was considered in [19] [10]. In [10], an algorithm have been propose to condense the flow table entries but retaining only those entries that subsume the other entries. In [19] a similar algorithm have been proposed for prefix expansion and aggregation. Our work is along the lines of reducing the information stored in the flow table. In our approach we exploit the capability of SDN to remove the redundant flow information used to identify the flow.

Architecture of TCAM: Architectural changes in the TCAM design and usage have been proposed to the power consumption of TCAM's. CoolCAM [23], EaseCAM [18], Routing table partitioning proposed in [20] are some the work that propose architectural changes. In CoolCAM, a bit selection forwarding engine architecture that exploits the fact that prefixes in the core routing tables (less than 2%) are either very short or very long (24 bits) proposed. In EaseCAM [18], prefix aggregation and expansion techniques is used to compact the effective TCAM size in a router. An architectural extension have been proposed in [20] to organize TCAM blocks perform searches in parallel. Majority of the above work exploit the common prefixes that can be used to store per destination based/per port based wild card flow entries. As mentioned earlier, in SDN prefixes will have lesser preference and therefore will have limited adoption.

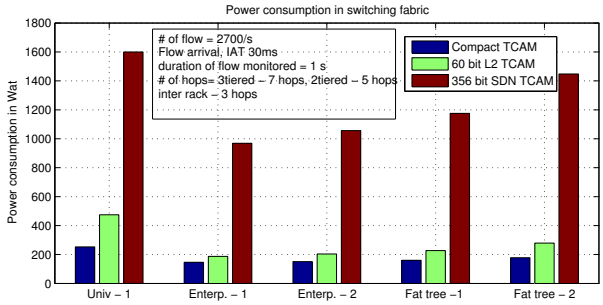


Fig. 7: Power consumption in switching fabric with different configuration. Univ-1: University 3 tier hierarchical topology, Enterp.-1: Enterprise 2 tier hierarchical topology, Enterp.-2: Enterprise 3 tier hierarchical topology, Fat tree-1: Fat tree topology (cloud) - interrack, Fat tree-2: Fat tree cloud 2 tier.

8 Conclusions

In an attempt to have combined gain on power and cost we have identified TCAMS as an optimization target. We propose *Compact TCAM* that deals with the redundant information stored in the TCAM's required for the flow processing. Flows are generally used to classify a packet and perform operations specific to flows. We show that for classifying packets, the information in the flow entry can be condensed to just the size of unique flows. This provides significant reduction in the size of the flow entries. *Compact TCAM* exploits the emerging SDN framework and utilizes OpenFlow a standard for SDN to eliminate the redundant information. We show that the switch fabric power gain can be about 2.5 times of a standard L2 switch and 80% gain in power compared to SDN switches. These gains are significant in stringent power bound DC Environments. Our approach is transparent to the end hosts and therefore highly adoptable for DCs. We are building our Compact TCAM framework on a OpenFlow enabled NetFGPA platform and our future work involves fine grained power measurements on this real system.

References

1. Agrawal, B., Sherwood, T.: TCAM Delay and Power Model. `http://www.cs.ucsb.edu/~arch/mem-model/`
2. Agrawal, B., Sherwood, T.: Modeling TCAM power for next generation network devices. In: Proc. of IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS). Austin, TX (Mar 2006)
3. Benson, T., Akella, A., Maltz, D.A.: Network traffic characteristics of data centers in the wild. In: Proceedings of the 10th annual conference on Internet measurement. pp. 267–280 (2010)
4. Benson, T., Anand, A., Akella, A., Zhang, M.: Understanding data center traffic characteristics. SIGCOMM Comput. Commun. Rev. 40(1), 92–99 (jan 2010)
5. Chao, H.J., Liu, B.: High performance switches and routers. John Wiley & Sons
6. Greenberg, A., Lahiri, P., Maltz, D.A., Patel, P., Sengupta, S.: Towards a next generation data center architecture: scalability and commoditization. In: Proceedings of the ACM workshop on Programmable routers for extensible services of tomorrow. pp. 57–62. PRESTO '08, ACM, New York, NY, USA (2008)
7. Kandula, S., Sengupta, S., Greenberg, A., Patel, P., Chaiken, R.: The nature of data center traffic: measurements & analysis. In: Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference. pp. 202–208 (2009)
8. Kaxiras, S.: Ipstash: A set-associative memory approach for efficient ip-lookup. IEEE Infocom pp. 992–1001 (2005)
9. Liu, A.X., Meiners, C.R., Torng, E.: TCAM razor: a systematic approach towards minimizing packet classifiers in TCAMs. IEEE/ACM Trans. Netw. 18(2), 490–500 (apr 2010)
10. Liu, H.: Routing table compaction in ternary cam. IEEE Micro 22(1), 58–64 (jan 2002)
11. Mahadevan, P., Sharma, P., Banerjee, S., Ranganathan, P.: A power benchmarking framework for network devices. In: Proceedings of the 8th International IFIP-TC 6 Networking Conference. NETWORKING '09 (2009)
12. McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., Turner, J.: Openflow: enabling innovation in campus networks. SIGCOMM Comput. Commun. Rev. 38(2), 69–74 (mar 2008)
13. Mogul, J.C., Tourrilhes, J., Yalagandula, P., Sharma, P., Curtis, A.R., Banerjee, S.: DevoFlow: cost-effective flow management for high performance enterprise networks. In: Proceedings of the Ninth ACM SIGCOMM Workshop on Hot Topics in Networks. pp. 1:1–1:6. Hotnets '10, ACM, New York, NY, USA (2010)

14. Naous, J., Erickson, D., Covington, G.A., Appenzeller, G., McKeown, N.: Implementing an openflow switch on the netfpga platform. In: Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems. pp. 1–9. ANCS '08, ACM, New York, NY, USA (2008)
15. Niranjan Mysore, R., Pamboris, A., Farrington, N., Huang, N., Miri, P., Radhakrishnan, S., Subramanya, V., Vahdat, A.: Portland: a scalable fault-tolerant layer 2 data center network fabric. SIGCOMM Comput. Commun. Rev. 39(4), 39–50 (aug 2009)
16. Raiciu, C., Barre, S., Pluntke, C., Greenhalgh, A., Wischik, D., Handley, M.: Improving datacenter performance and robustness with multipath tcp. In: Proceedings of the ACM SIGCOMM 2011 conference. pp. 266–277. SIGCOMM '11, ACM, New York, NY, USA (2011)
17. Rajan, K., Govindarajan, R.: A novel cache architecture and placement framework for packet forwarding engines. IEEE Trans. Comput. 58(8), 1009–1025 (aug 2009)
18. Ravikumar, V.C., Mahapatra, R.N., Bhuyan, L.N.: Easecam: An energy and storage efficient TCAM-based router architecture for ip lookup. IEEE Trans. Comput 54, 2005 (2005)
19. Ravikumar, V., Mahapatra, R.N.: TCAM architecture for ip lookup using prefix properties. IEEE Micro 24(2), 60–69 (mar 2004)
20. Tzeng, N.F.: Routing table partitioning for speedy packet lookups in scalable routers. IEEE Trans. Parallel Distrib. Syst. 17(5), 481–494 (2006)
21. Waldvogel, M., Varghese, G., Turner, J., Plattner, B.: Scalable high speed ip routing lookups (1997)
22. Wilton, S., Jouppi, N.: Cacti: an enhanced cache access and cycle time model. Solid-State Circuits, IEEE Journal of 31(5), 677 –688 (may 1996)
23. Zane, F., Narlikar, G., Basu, A.: Coolcams: Power-efficient TCAMs for forwarding engines. In: IN IEEE INFOCOM. pp. 42–52 (2003)