Scissors: Dealing with Header Redundancies in Data Centers through SDN

Kalapriya Kannan
IBM Research India New Delhi, INDIA
Email: kalapriya@in.ibm.com

Subhasis Banerjee
IIIT-Delhi, New Delhi, INDIA
Email: subhasis@iiitd.ac.in

***Abstract*—Growing concern for reduced power dissipation, cost and latency demands in next generation Data Centers (DC) motivates us to revisit header optimizations. Headers contribute to about 30-40% of DC traffic and is responsible for equal proportion of consumed power. This amounts to significant overhead on per byte transfer of payload. In the past, highly inflexible switches have limited the focus of header optimizations primarily on end-host or edge network routers. Further, strict compliance to protocols by switches/routers along the path have limited the segments of headers that can be optimized. Recent evolution of Software Defined Network (SDN) brings in several opportunities to control over switches. We believe that if SDN's finer handle to the switching devices is exploited, it can help deal with header information.**

In this paper, we exploit the capabilities of SDN and introduce a new functionality that can effectively replace the redundant and repetitive header information with shorter unique identity. We present *Scissor* that trims the headers lower in the protocol stack. As a replacement for routing, we introduce the notion of Flow-ID, where all packets belonging to a flow are identified using this unique Flow-ID. We leverage the capabilities of the SDN to dynamically allow switching devices to route the packets based on the Flow-IDs. Our approach of trimming header at the switching devices leaves the hosts unmodified making it highly adoptable for DC environment. We show that our approach can decrease switch fabric power consumption by a factor of 2.5 compared to an existing L2 switch and reduce the latency by about 30% for a significant fraction (around 30-50%) of the network traffic.

I. Introduction

Latency, Power and Cost have been the top three critical optimization parameters for DCs and will continue to be so with ever increasing demand in the future. Independently these parameters have set very stringent bounds. The increasing mandate to handle higher capacities and sophisticated network management functions have increased the cost of networking devices. A modern DC Cisco Switch providing traffic classification is at least twice as expensive as those with just switching capabilities [1]. This trends is likely to continue with even stringent bounds. There is a pressing need to address these growing concerns in a combined manner to achieve any meaningful gain.

The aforementioned parameters are well connected, an association that is less exploited in network optimization techniques. For instance, shrinking latency demands have made Ternary Content Addressable Memory (TCAM) an indispensable part of networking devices, but has higher power costs. In this paper we identify redundancy, especially header redundancies which constitute about 30-40% of DC traffic, to have the potential to provide combined gains on power, latency and cost. Headers have an impeding effect on latencies, endorsing effect on the complexities of processing thereby increasing power and cost. We believe that performing optimizations on such voluminous redundant data will provide more than miniature gains along these parameters.

In the past, header redundancies have been often revisited by researchers [2], [3] but have remained confined to selected categories of networks. This has been partly due to underlying transport protocol requirements and partly due to highly inflexible and rigid switching devices. For the same reasons, end-to-end negotiations have been adopted so that the communicating parties are *a-priori* aware of the compression and the decompression protocol. We believe that the recent introduction of the SDN [4] will unravel the inflexible switching limitations allowing more fine-grained control on handling of flows and packets at the switching devices.

In this work we exploit the benefits of OpenFlow [5], a standard for SDN by providing a framework: *Scissor*, our new functionality that trims down the packet headers much lower in the protocol stacks transparent to the end hosts. As a replacement, we provide shorter tag called *Flow-ID* and all routing is performed on the basis of these tags. Such a mechanism removes the overhead of the headers on the network and saves the real estate required for storing the flow information on hardware devices. Our approach works within the capability of the SDN framework and does not add additional functional or management overhead. We show that this trimming can provide overall gains of latency, power in DC and significant reduction to premium silicon real estate thereby lowering associated costs.

The paper is organized as follows. In Section II, we present some of the observation in DC through analyzing the DC traces in the background study. We briefly describe the power and latency in Section III. In Section IV, we present *Scissors* that leverages SDN to trim the header information. Expected gains using *Scissors* are evaluated and presented in Section V. We review existing literature in the context of header compressions/optimization and highlight the main differences in Section VI. We finally present our conclusions in Section VII.

II. Background

We present the impact of the header availability through observations made in the DC traffic. In an attempt to compute the overhead associated with transferring per byte of payload we estimate the payload to header ratio: *# of bytes in payload / # of bytes in header*. We use the DC traces presented in [6] to observe the per byte payload overhead. Fig. 1 shows the CDF plot of the overhead. It can be seen that around 40% of the packets contain zero bytes of payload (packets carrying only the header's). Only 30-40% of the packets are fully loaded packets.

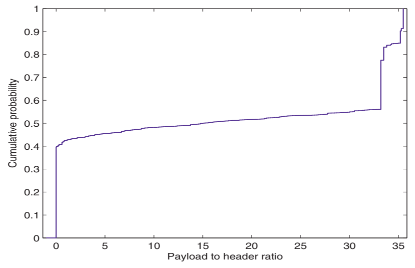


Fig. 1. CDF of overhead of header

As observed, significant portion being just headers motivates us to optimize the headers. Headers contain the necessary information for the switches to determine the routes and actions to be performed on the packets. All the information required by the router for determining actions are available in the first packet of the flow. Repeatedly performing the same actions on the subsequent packets of the flow is duplicative and leaves scope for optimization. We believe these optimizations will improve both the lantency (as it is function of network load) and power (as headers are the single most component responsible for complex processing at the switches).In rest of the paper, we use OpenFlow and SDN interchangeably.

III. Power and Latency Models

We model the (a) power consumption and (b) end-to-end latencies for a flow.

A. Power Dissipation in 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 [7]. Functionally it consists of a header parser, a input arbiter, a lookup operation, a mediator, a packet editor and the output ports out of which the lookup operation (that involves several memory operations) lie in the critical path of the switching device. Among these functional units the lookup operation is the dominating factor in terms of power consumption compared to other operations e.g., header parsing and packet forwarding to output ports.

Power Due to 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 memory structures are often used in implementing a hardware table: SRAM and TCAM. SRAMs offer low-power operations and are typically used for storing exact match entries where the lookup can be performed by asserting address line (wordline) and reading the corresponding bitlines. The latencies are higher for SRAM and therefore are lesser preferred than the TCAM. TCAMs are used to implement lookup tables and packet classifications in hardware. TCAM's are used for achieving high-speed lookup times which are important for achieving line rates in high performance networks. However, they are notoriously power-hungry. A TCAM operation can consume about 150 times more power than an equivalent operation in SRAM. 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:

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

where N_{lp} and N_{in} 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).

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

where $N_{switches}$ denotes number of switches along the path of the flow.

B. Latency in L2 Layer Networks

There are several sources of latency in L2 Layer network for a packet. They are (a) store and forward latency, (b) switch fabric latency, (c) queuing latency out of which store and forward latency is dominating factor being dependent on the number of bits being transmitted.

Store and Forward Latency (L_{sf}): Store and Forward (SF) latency refers to the basic operating principle of the L2 level switch. The switch stores the packets in its buffer until the entire frame is received. A few modern day switches start processing and transmission to output port even before the packet is completely received. The switch then processes the headers for identifying a destination. The latency introduced by this is proportional to the the size of the frame and inversely proportional to the bit rate as follows:

$L_{sf} = \frac{FS}{BR}$ where FS is the frame size in bits, BR is the bit rate in bits/sec.

IV. Scissors for Dealing with Header Redundancies

We introduce the notion of "Flow-ID". Flow-ID is a numeric identifier. This is used as a replacement for headers. However, some fields in the original header (e.g., Internet header, total

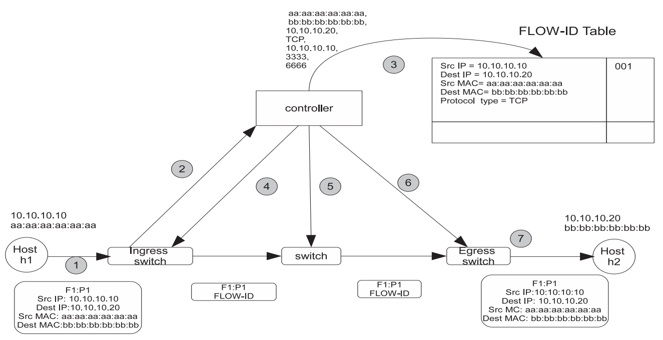


Fig. 2. System showing *Flow-ID* utilization through different steps.

length, IP identification etc.) that varies with each packet of the flow are retained in the modified header along with the Flow-ID. In the switch table, the flow entry can be reduced to the size of the 'Flow-ID' and associated actions. It should be noted that, apart from the fields representing a flow, switch stores other fields such as counters, priority fields etc. which remain as-is as they are dynamically utilized for each packet. Table I show an example mapping of flow entires to their corresponding Flow-ID's. All routings are performed based on the Flow-ID's.

TABLE I
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

The programming control provided by the SDN makes it viable for us to design a system that can be benefited from Flow-ID's. We propose *Scissor* framework that combines the flexibility provided by SDN along with Flow-ID's to deal with redundant header information. Fig. 2 shows the *Scissor* architecture consisting of the controller, OpenFlow switches and host machines.

In OpenFlow the first packet of a new flow arriving at an interface of a switch (Step 1 of Fig. 2)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 *Scissor*, we augment the 'action' part of the rule with a boolean 'SCISSOR' flag. If the flag is clear, the switch performs the standard actions on the packets and *Scissor* operation is bypassed. For instance, the flag is set to be clear if the destination address is connected to the same switch. Otherwise (SCISSOR flag set), the switch performs the *Scissor* operations on the packets and output it to specific port as given in the action part.

Scissor operation consists of two parts (a) replacement of the header information with Flow-ID and (b) trimming down the packet headers before it is forwarded to the output port. 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. Trimming down of the header is performed by a specific micro-architectural hardware that selectively discards bits of the header before forwarding it to the output port. Reduction can be achieved up to **254** bits in the header. This includes **144** bits from the Ethernet frame (Ethernet Src[6 bytes], Ethernet Dst[6 bytes], Ethernet Type [2 bytes], VLan Tag [4 bytes]), **76** bits of data in IP header (IPv4 Src address[4 bytes], IPv4 destination address[4 bytes], IPv4 type [2 bytes] and IPv4 [6 bits]) and about **32** bits of TCP header (TCP Src Port [2 bytes], TCP Dest Port [2 bytes]). Only selected fields (those that are extracted as part of flow information in the controller) from IP and TCP can be trimmed, as this information is made available by the controller at the egress switch to reconstruct the packet fully before delivering it to the end host.

The controller sends a flow insert operation to all the switches along the path of the packet (Step 5 of Fig. 2). 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 scissor 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 replace the specific fields of the packet with the information extracted as flow at the IS before forwarding it to output port. Step 6 in Figure illustrates the flow insert operation specified by the controller to the egress switch. These values are inserted at specific bit positions and packet is forwarded to the output port to deliver to the end hosts (Step 7).

Example Demonstration: In Fig. 2 let us assume that host $h1$ initiates a communication (a flow) to $h2$. The figure also shows a sample packet being generated from $h1$ to $h2$. For a flow, the Ingres Switch (IS) is the switch through which the packet enters into the network and Egress Switch (ES) is the switch by which it exits out of the network. Shown below are 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 'Src MAC', 'Dst IP' and 'Dst MAC' and few fields such as IP Checksum and Identification number that are unique to 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	

We refer to this packet as ‘f1:p1’. At the IS, the selected fields from the header of ‘f1:p1’ is extracted and is used for locating a matching entry in the flow table. 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 ‘SCISSOR’ bit set. Figure below shows the flow entry in the IS.

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 = ’SCISSOR’, 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 [6] 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. The IS trims the header information.

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 select only two bytes to extract the *Flow-ID*.

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_dst[] = 0x001, actions = output:3

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 table for *Flow-ID* match. As this will result in a hit the actions associated with the entries are applied to the packet. All intermediate routing are performed based on the Flow-IDs. 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.

Scissor Hardware

Fig. 3 illustrates the implementation of Scissor micro-architecture as part of packet editor (refer to the pipeline reference architecture in [7].). As described in the previous section, the scissor operates only on packets for which ‘SCISSOR’ flag is set. The hardware consists of a buffer that temporarily holds the packet along with the complete header.

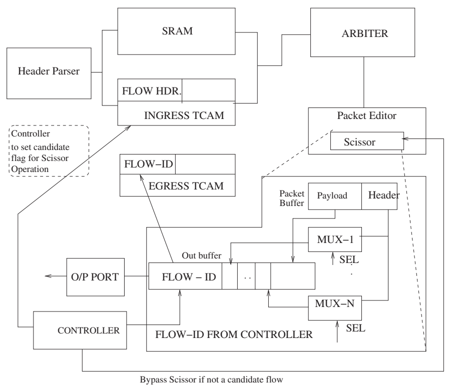


Fig. 3. Scissor hardware shown in context of reference switch architecture given in [7].

A set of Multiplexers (MUX-1, .. MUX-N) are used to select appropriate field that are to be retained by Scissor. SEL signal that controls the selection of bits from the header is predefined (as the fields to be retained are known *apriori*). Multiplexers select only the required fields and copy it into a output buffer along with the payload. The assigned *Flow-ID* is copied at the selected bits of the buffer (2 bytes following the frame delimiter) and the packet is ready to dispatch through the output port. The power consumption by these multiplexer are negligible compared to the hardware flow table operations. Intermediate switches / core switches contain TCAM array that has entries with Flow-ID.

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 TCAM and Ingress TCAM as shown in Fig. 3) 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 flow entry with complete header along with SCISSOR flag. The edge switch as directed by the controller either performs a SCISSORS operations (where it cuts down the header information [IS]) or insert operation (for those packets containing Flow-ID [ES]) where the extracted flow information is inserted into the original position for reconstructing the packets. Due to these separate TCAMs, power gains on the edge switches are lesser than 50% than those obtained from the core switches.

V. 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, (b) end-to-end latency gain obtained due to reduction of packet size being transferred. Measurement of the end-to-end latency and power gain require fine grained flow level data traces both at the switches and the end hosts. In [8] it has been established that obtaining such fine grained flow level details and related data traces are difficult due to expensive instrumentation required

TABLE II
CHARACTERISTICS OF DC TRAFFIC USED FOR SIMULATION ON THE TOPOLOGIES

Flow character-istic	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

TABLE III
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	≈ 35000-40000	≈ 19000-21000	< 20000	< 50000	< 20000	¡50000
Packet of new flow	Insert	≈ 1200-1300	≈ 800-850	< 900	≈ 1200	< 900	≈ 1200

on the servers/switches. We resort to simulation of DC traffic given the flow characteristics of real world DCs. For all our experiments we consider the switch-to-switch measurements.

A. 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) OpenFlow standard based switching device consisting of 356 bits flow entries (15 tuple flow entries) and (c) *Scissor* based switching device consisting of 16 bits flow entries. We have considered three different topologies (Fat tree (taken from [9], 2-tier multi rooted tree and multi-tiered multi rooted tree presented in [6]) representing diverse set of DC environments such as Enterprise DC, University DC and Cloud DC's. Flows are generated by our simulator according to traffic characteristics for various DC's adopted from the existing studies presented in [6] [10] and is presented in Table II.

B. Power Gain

We simulate the arrival of packets at the switches based on the distribution presented in Table II. According to Equation 2 we require the number of packets that requires lookup and insert operations. Table III 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 [11] to observe the power consumption for different sizes. We fix the number of flow entries to 100000 and measure the power consumption for varying sizes of bit-line. We use traffic characteristics of 3 different DC environments (shown in Table III) to simulate the TCAM power consumption. Table IV shows the power consumed by various operations of TCAM. It can be seen that power consumed varies significantly with the size of the TCAM (both the number of entries and the bit-line). The power consumed to perform a write/insert operation with 16 bits is about 4 times lower than the 60 bit and 15 times lower than the 356 bit entry. 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.

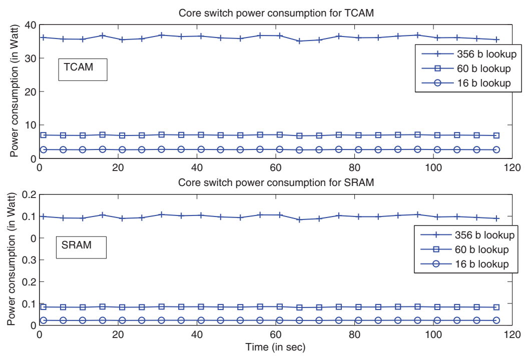
Fig. 4 (a) presents the power consumption (according to Equation 2) on the core and Fig. 4 (b) show the power consumption at edge switches for Enterprise and Univ DC's. At the core switches, a L2 switching fabric would consume about 2.5 times more power and a OpenFlow enabled switch will consume about 12 times more compared to a *Scissor* switch. Therefore, in core switches Scissor switch gives best power saving. This gains are lower in the edge switches as the power saving is obtained only for the egress flows. In Enterprise / Univ DC's around 40-90% of traffic leaves top of the rack switch and therefore the overall gains would be about 35% of the best case (saving of 12 W with respect to the best case of 35 W in core switch). For the Cloud DC's around 75% of the traffic would be observed within the rack, where no scissoring operation will be performed. Therefore the observed power gains is only about 20% of the best case.

TABLE IV
TCAM POWER CONSUMPTION FOR VARYING BIT-LINE SIZES, # OF FLOW ENTRIES = 100,000

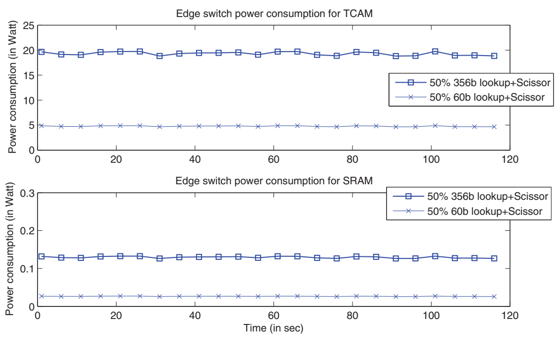
Operation	L2 sw. (nJ)	SDN sw. (nJ)	Scissor sw. (nJ)
Write/Insert	195167	792677	56614
Search/Lookup	447065	1807142	131685

C. Latency Reduction

Latency is a function of network load. We consider bandwidth of 10 Gbps for computing the bit rates supported by the links. We measure the time taken for the flows to complete based on the number of switches that the packets traverses and assuming a constant queuing delay for all packets at the switches. Fig. 5 plots the CDF of flows and the latency reduction in Enterprise / Univ DC and Cloud DC environment. It can be observed that in the case of Enterprise / University environments about 40-50% of the flows experience no gains (as 30-40% is within the racks) in latencies. Flow traversing outside the top of rack switch which constitutes about 60-90% have reduced latencies by about 10-30% compared to non-scissors environment. In case of the Cloud DCs around 70-85% of the flows do not experience reduced latencies. This is primarily due to the fact that around 70% of the traffic being intra-rack and does not traverse the network. The traffic that



(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. 4. Power consumption at the core and edge switches

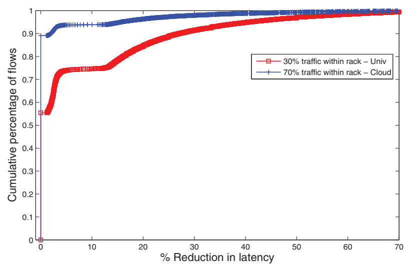


Fig. 5. Cumulative percentage of the flows and corresponding latency reduction due to trimming of headers by Scissors

traverses the network (30%) shows reduction of latency again by about 15-20%. These gains are highly significant especially when the latency bounds are stringent.

D. Cost of Signaling

Scissors do not add additional stress on the signaling required to setup the rules for header trimming. According to OpenFlow specification [4], the first packet of flow is sent to the controller. The controller is assumed to be a logically centralized entity that provides every switch with the actions required to be performed on the packets of the flow. In our case, the action is associated with inserting a Flow-ID, instructions to trim the header, routing according to the Flow-ID, restoring back the header. Therefore the number of operations involved in signaling does not change due to Scissors functionality. However, currently the action consisting of trimming the headers requires specialized hardware to selectively output bytes of data to the output ports in the ingress switch and inserting specific bytes of data in the packet at the egress switch. These operations incur negligible latency and power (as shown in the Section (IV)).

VI. Related Work

In this section we position our work among related systems, which we classify under the following three categories: header compressions, deduplication and dynamic routing systems. A survey of header compression work have been presented in [12]. ROHC [13], IPHC [14] attempt to compress the segments of header (IP headers/ TCP header segment). ROHC and IPHC operate by exploiting the repetitiveness found in the information carried in the header. Compression technique have also been applied to latency sensitive applications such as multimedia data [15]. In [15], the RTP headers have been identified to have redundant information and latency gains have been shown by compressing them. We distinguish our work along three different ways. Firstly, tagging helps removing header redundancies much lower in the protocol stack while retaining routing capabilities. Secondly, our approach is performed transparently in the network. Such an architecture would be highly beneficial to DC environments where several thousands of end-hosts run critical applications. Ultimately, our work is complimentary to those approaches that deal with the redundancy through compressions techniques.

De-duplication techniques have not only been confined to the headers but have been applied to segments of payload in [16] [17]. Such techniques identifies repeating payload segments in packets and eliminate them through indexing. Our approach applies similar concept to extract the header information with the support of emerging SDN framework. In [18] dynamic addresses are assigned to nodes and routing is performed based on these addresses. However, the focus of such work have been to provide transparent routing for migrating nodes and is not applied in the context of the header redundancies. Our approach is along those lines of providing dynamic tags for routing the packets while keeping our focus on removing redundant headers.

VII. Conclusions

In an attempt to have combined gains on power, latency and cost we have identified header redundancy as an optimization target. We propose *Scissor* that deals with the redundant header information by trimming them at the lower most stack. This provides significant reduction in the size of the header information. *Scissor* exploits the emerging SDN framework and utilizes OpenFlow a standard for SDN to eliminate the redundant header information. Scissors shows significant latency improvement (as high as 30%) while switch fabric power gain can be about 2.5 times of a standard L2 switch.

References

[1] "The Total Economic Impact of Cisco Catalyst Access Switches," http://www.cisco.com/en/US/prod/collateral/switches/C11-698012-00_Cisco_Access_Switching_Forrester_TEI_Study.pdf.

[2] M. Degermark, M. Engan, B. Nordgren, and S. Pink, "Low-loss tcp/ip header compression for wireless networks," in *Proceedings of the 2nd annual international conference on Mobile computing and networking*, ser. MobiCom '96. New York, NY, USA: ACM, 1996, pp. 1–14. [Online]. Available: http://doi.acm.org/10.1145/236387.236388

[3] K.-s. Kim, M.-s. Kang, and I.-t. Ryoo, "Header compression of rtp/udp/ip packets for real time high-speed ip networks," in *Proceedings of the 1st international conference on Advances in hybrid information technology*, ser. ICHIT'06. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 576–585. [Online]. Available: http://dl.acm.org/citation.cfm?id=1782654.1782716

[4] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, mar 2008. [Online]. Available: http://doi.acm.org/10.1145/1355734.1355746

[5] "The OpenFlow Switch Specification," http://www.openflow.org/documents/openflow-spec-v1.1.0.pdf.

[6] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proceedings of the 10th annual conference on Internet measurement*, ser. IMC '10. New York, NY, USA: ACM, 2010, pp. 267–280. [Online]. Available: http://doi.acm.org/10.1145/1879141.1879175

[7] J. Naous, D. Erickson, G. A. Covington, G. Appenzeller, and N. McKeown, "Implementing an openflow switch on the netfpga platform," in *Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, ser. ANCS '08. New York, NY, USA: ACM, 2008, pp. 1–9. [Online]. Available: http://doi.acm.org/10.1145/1477942.1477944

[8] T. Benson, A. Anand, A. Akella, and M. Zhang, "Understanding data center traffic characteristics," *SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 1, pp. 92–99, jan 2010. [Online]. Available: http://doi.acm.org/10.1145/1672308.1672325

[9] A. Greenberg, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, "Towards a next generation data center architecture: scalability and commoditization," in *Proceedings of the ACM workshop on Programmable routers for extensible services of tomorrow*, ser. PRESTO '08. New York, NY, USA: ACM, 2008, pp. 57–62. [Online]. Available: http://doi.acm.org/10.1145/1397718.1397732

[10] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, "The nature of data center traffic: measurements & analysis," in *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, ser. IMC '09. New York, NY, USA: ACM, 2009, pp. 202–208. [Online]. Available: http://doi.acm.org/10.1145/1644893.1644918

[11] "TCAM Delay and Power Model," http://www.cs.ucsb.edu/ arch/mem-model/.

[12] T. C and F. G, "A review of ip packet compression techniques," in *PGNet*, 2003.

[13] C. Bormann, C. Burmeister, M. Degermark, H. Fukushima, H. Hannu, L.-E. Jonsson, R. Hakenberg, T. Koren, K. Le, Z. Liu, A. Martensson, A. Miyazaki, K. Svanbro, T. Wiebke, T. Yoshimura, and H. Zheng, "Robust header compression (rohc): Framework and four profiles," United States, 2001.

[14] M. Degermark, B. Nordgren, and S. Pink, "Ip header compression," 1999.

[15] P. Fortuna and M. Ricardo, "Header compressed volp in ieee 802.11," *Wireless Commun.*, vol. 16, no. 3, pp. 69–75, jun 2009. [Online]. Available: http://dx.doi.org/10.1109/MWC.2009.5109466

[16] A. Anand, C. Muthukrishnan, A. Akella, and R. Ramjee, "Redundancy in network traffic: findings and implications," in *Proceedings of the eleventh international joint conference on Measurement and modeling of computer systems*, ser. SIGMETRICS '09. ACM, 2009, pp. 37–48.

[17] A. Anand, A. Gupta, A. Akella, S. Seshan, and S. Shenker, "Packet caches on routers: the implications of universal redundant traffic elimination," in *Proceedings of the ACM SIGCOMM 2008 conference on Data communication*, ser. SIGCOMM '08. New York, NY, USA: ACM, 2008, pp. 219–230. [Online]. Available: http://doi.acm.org/10.1145/1402958.1402984

[18] R. Niranjan Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat, "Portland: a scalable fault-tolerant layer 2 data center network fabric," *SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 4, pp. 39–50, aug 2009. [Online]. Available: http://doi.acm.org/10.1145/1594977.1592575