PAPER Special Section on Quality of Communication Services Improving Quality of Life

OpenQFlow: Scalable OpenFlow with Flow-Based QoS

Nam-Seok KO[†,††a)], ***Student Member***, **Hwanjo HEO**[††], **Jong-Dae PARK**[††], ***and*** **Hong-Shik PARK**[†], ***Nonmembers***

SUMMARY OpenFlow, originally proposed for campus and enterprise network experimentation, has become a promising SDN architecture that is considered as a widely-deployable production network node recently. It is, in a consequence, pointed out that OpenFlow cannot scale and replace today's versatile network devices due to its limited scalability and flexibility. In this paper, we propose OpenQFlow, a novel scalable and flexible variant of OpenFlow. OpenQFlow provides a fine-grained flow tracking while flow classification is decoupled from the tracking by separating the inefficiently coupled flow table to three different tables: flow state table, forwarding rule table, and QoS rule table. We also develop a two-tier flow-based QoS framework, derived from our new packet scheduling algorithm, which provides performance guarantee and fairness on both granularity levels of micro- and aggregate-flow at the same time. We have implemented OpenQFlow on an off-the-shelf microTCA chassis equipped with a commodity multicore processor, for which our architecture is suited, to achieve high-performance with carefully engineered software design and optimization.

key words: *software defined networking (SDN), OpenFlow, flow-based networking, QoS*

1. Introduction

OpenFlow [1], a leading research project that envisions software defined networking (SDN), has caught a lot of attention recently from a variety of network research communities including major leading commercial companies. SDN is a new network paradigm that enables a remote controller to modify the behavior of network devices without hardware change or switch-by-switch configuration changes. Such innovation becomes feasible by separation between the control plane and the data plane in contrast to the tight coupling in traditional network devices. OpenFlow is initially launched to enable researchers to run experimental protocols in campus production networks, but it has become a promising SDN architecture to embrace various network functionalities in large scales.

While OpenFlow is a dominant open standard of SDN, it has several issues to resolve to meet such expectation. Scalability has been a great concern since it is not desirable to contain a large number of fine-grained flow entries due to its costly flow table data structure. Frequent invocation of the controller — for example, by sending the first packet of every flow to have a complete visibility of flows, often referred as flow setup — also brings overhead concerns. DIFANE was proposed to address the scalability issue by distributing rules across multiple switches [2]. DevoFlow tries to reduce the reliance on the control plane by engineering the trade-offs regarding flow visibility and statistics-gathering [3].

Limited action set brings another issue of flexibility. OpenFlow, of course, can always forward packets to the controller and let the controller do whatever action to the packets on its discretion; however, there are certain actions — such as oblivious routing/link failover [2], QoS [4], and flow tracking [5] — that can be implemented locally for far less overhead due to the reduced communication. Especially, the QoS support of OpenFlow is very limited — the only QoS action is the output queue assignment — and OpenFlow doesn't have an interface or related specification to have an extension of local actions.

Even though SDN is a new paradigm, a similar concept of flow-based networking has been studied from a while ago [6]–[8]. Electronics and Telecommunications Research Institute (ETRI) in South Korea has been working together with Sable Networks, previously Caspian networks, on developing flow-based routers [6] since 2004. There are also commercial products of QuantumFlow [7] by Cisco systems and the fast flow technology [8] by Saisei networks. This paper proposes OpenQFlow, a novel OpenFlow architecture that tackles the scalability and flexibility issues of OpenFlow.

Our architecture divides the flow table into three different tables: flow state, forwarding and QoS rule tables. Separation of the tables breaks unnecessary coupling which brings the scalability and flexibility problems that limit world-wide deployment and versatility. OpenQFlow also implements flow-based QoS that works on both granularity levels of micro- and aggregate-flow simultaneously. Our QoS framework is yet straightforward to implement and induces low processing overhead on the existing SDN architecture. OpenQFlow, at an implementation perspective, is very flexible and extensible in the sense that it can be easily modified or extended for additional or enhanced functionalities as it's running upon an off-the-shelf multicore processor platform. In this paper, we make the following major contributions:

- We design OpenQFlow, a novel OpenFlow architecture

Manuscript received May 31, 2012.
Manuscript revised September 12, 2012.

[†]The authors are with the Dept. of Information and Communications Engineering, Korea Advanced Institute of Science and Technology (KAIST), Daejeon, South Korea.
[††]The authors are with the Dept. of Computing Network Research, Electronics and Telecommunication Research Institute (ETRI), Daejeon, South Korea.
a) E-mail: nsko@kaist.ac.kr

DOI: 10.1587/transcom.E96.B.479

Copyright © 2013 The Institute of Electronics, Information and Communication Engineers

that enables fine-grained flow tracking with flow classification decoupled;
- We develop flow-based packet scheduling algorithm which provides performance guarantee on both granularity levels of micro- and aggregate- flow;
- We build the conclusive SDN architecture with two-tier flow-based QoS on a commercial off-the-shelf microTCA platform to show OpenQFlow is a scalable and high-performance system.

The remainder of this paper is organized as follows. In Sect. 2, we propose a framework of OpenQFlow and discuss distinguishing features of the proposed architecture. Then, flow-based QoS framework is proposed and analyzed in Sect. 3. In Sect. 4, we discuss prototype implementation and show performance results of our implementation. We finally conclude in Sect. 5.

2. OpenQFlow

2.1 Architecture

OpenQFlow divides the flow table into three different tables: flow state table, forwarding rule table and QoS rule table. Each rule table of OpenQFlow can consist of multiple internal rule tables with different key values; the tandem rule tables are searched in order until the match is found. Such configurable rule table design is chosen to make the high-performance rule table lookup possible without help of an expensive hardware such as TCAM. Each internal rule table is a best-matching table where each table entry is pre-sorted according to the priority. Each entry of internal rule tables of the forwarding rule table has a pointer to a forwarding information base that consists of forwarding action, i.e., either forward or drop, and nexthop information [1]. That of the QoS rule table has a pointer to a QoS information that comprises of the traffic type, bandwidth and priority information.

The flow state table is a hash table with 21 bit index — CRC-16 hash values of flows' 5-tuple are exact-matched — with a linked list for each hash table entry to resolve hash collisions. The table has the following properties:

- Each entry of the table is dynamically created on the arrival of the first packet of a new flow and deleted if no more packet that belongs to the flow arrives for a pre-defined time interval;
- Each entry is maintains 128-byte microflow state information including forwarding, QoS, and statistics information;
- The statistics information is periodically pushed to a controller and also can be pulled on demand;
- The flow state table can be modified by local engines[†] or a remote controller.

The forwarding and QoS information are searched from the rule tables and linked to the flow state table entry upon the arrival of the first packet of a new flow. The arrival of subsequent packets then requires referring the flow state table to find the forwarding and QoS information base without rule table lookups. The packet flow of OpenQFlow is depicted in Fig. 1. OpenQFlow processes every ingress data packets according to the following steps:

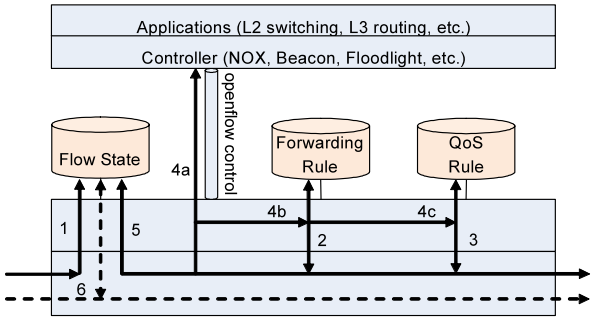


Fig. 1 Packet flow of OpenQFlow.

1. When a packet comes into the system, the flow state table lookup occurs;
2. If there is no match, it is the first packet of a new flow. The forwarding rule table is searched — it goes through the internal rule tables — until the match is found. If not found, it defaults to the unknown forwarding action that is sending it to the controller in usual configurations.
3. The QoS rule table is also searched. If no match is found, the traffic type defaults to the best-effort.
4. As soon as the packet is delivered to the controller, the controller may inspect it and add a forwarding rule entry and/or a QoS rule entry for the given flow of the packet. The wildcard expression of the flow is determined by the controller application.
5. The flow state entry is updated with the forwarding and QoS information.
6. The subsequent packets of the same microflow results in a match from the flow state table lookup. The packets will be processed according to the forwarding and QoS rule that are associated to the matched flow state table entry.

2.2 Distinguishing Features of OpenQFlow

In this subsection, we discuss the important features of OpenQFlow in comparison to OpenFlow and related studies.

2.2.1 Fine-Grained Flow Tracking and Decoupled Classification

Each flow entry in the flow tables of OpenFlow not only defines the wildcard rule for classification but also has statistics counters that is utilized for flow tracking [5]. Often,

[†]We have a QoS co-processor in our current implementation while other engines such as a deep packet inspection engine can be implemented.

flow tracking — such as detecting elephant flows or filtering malicious flows with certain threshold for workload limited deep packet inspection — is useful at the granularity level of microflow[†]. OpenFlow, however, specifies the 10-tuples of packet headers that are overkill for fine-granularity while often not enough for flow classification; for example, application identification — of P2P applications such as BitTorrent or VoIP applications such as Skype — has become a non-trivial problem involving DPI or behavioral analysis rather than simple port number matching.

Let us consider some example requirements of a service-aware OpenFlow switch: for each microflow whose size is 1M bytes, the flow is inspected through the DPI and behavioral analysis engine to see if it is a P2P or VoIP user session. If the flow is a P2P session, the flow receives a penalty of rate-limiting; otherwise, if it is a VoIP session, the flow is isolated to a certain link and receives a guaranteed rate. OpenFlow simply cannot meet the requirement for the following reasons: a) each microflow cannot be tracked for a certain threshold unless every possible 5-tuple combinations are pre-defined in the table, b) having flows being classified by adding a flow entry for every P2P and VoIP sessions is not scalable even if we assume they are identified at the controller.

OpenQFlow has a flow state table that can track every microflow, and also has forwarding information and QoS profile for every one of them without the scalability issue being concerned. The classification is separately performed by matching the forwarding and QoS rule tables or by querying the embedded DPI or behavioral analysis engine on demand if exists.

DevoFlow has a rule cloning mechanism that augments the action part of wildcard rules to have a boolean CLONE flag that indicates that every microflow matching the wildcard rule individually maintains the statistics counter; however, the latter problem of scalability is sustained [3]. Also, the classification is solely performed by the flow table matching.

Microflow tracking is also beneficial to dynamic flow scheduling when combined with OpenQFlow's per-microflow forwarding. The multipath forwarding is a functionality that can select an output port for microflows with given probability distribution. Equal-Cost Multi-Path (ECMP) is one instance of it where the probability is equally distributed among the output ports. However, we find the probabilistic distribution of flows — proportional distribution of flows in terms of flow count — does not work well since the aggregate bandwidth of flows is not proportional to the flow count.

We examined 1,000 second packet trace snippet of Abilene public packet trace [9] that is captured at the OC-192C link between Internet 2's Indianapolis (IPLS) and Kansas City (KSCY) router node where each flow is categorized into four hash bins according to the last two bits of the CRC-16 of the 5-tuple packet header fields[††].

Figure 2 shows the number of flows and octets for each 10 second aggregated hash bin. The flow count is

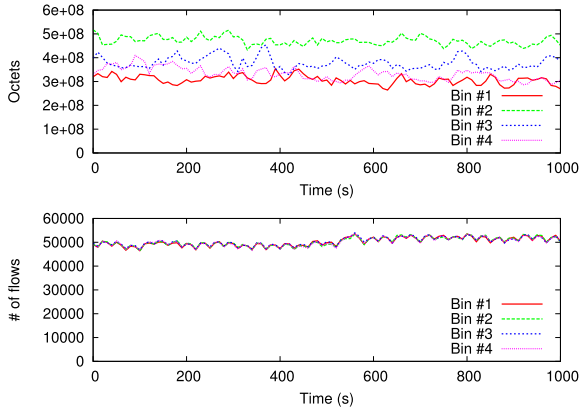


Fig. 2 The number of flows and octets per hash bin.

evenly distributed — the four lines are very close and correlated — while the discrepancy in octets is significant; there is no surprise since the flow size distribution is heavy-tailed [10]. OpenQFlow, thus, supports the dynamic flow scheduling, which identifies elephant microflows in output ports and reroutes them — in microflow granularity — on demand, that works by interfacing to a controller logic such as [11].

OpenFlow, on the other hand, can neither track elephant flows unless their list is pre-provisioned in the flow table, nor reroute an elephant microflow but an aggregate flow that might be too coarse in the sense that rerouting of one likely makes another path congested.

2.2.2 Two-Tier QoS Provisioning

When it comes to the SDN architecture, the issue of flow QoS provisioning is rarely discussed in the literature. The only QoS action in OpenFlow is *set_queue* which assigns a packet to a specific priority queue of the output port. Kim et al. proposed a QoS control framework which extends OpenFlow API to manipulate hardware rate limiters and priority queue mappings [4].

OpenQFlow implements the two-tier QoS provisioning that comprises performance guarantee and fairness among both microflow and aggregate flow level. A guaranteed flow rate can be set for given microflow or aggregate flow. At both microflow and aggregate flow level, flow bandwidth is fairly provisioned with optional weights in case of congestion. The granularity of aggregate flow can be defined with any bit-masked subset of 5-tuples in the current OpenQFlow implementation. We find salient applications of aggregate flow fairness provisioning at both access and data center networks.

Subscription guarantee/fairness at access networks:

[†]Microflows are uniquely identified with the well-known 5-tuples.

[††]The flow count trace requires a few tens of seconds to converge. We have studied four hour long packet trace, but 1000 second long data with flow count convergence period trimmed are representative enough to show our points.

A distribution network where a number of subscribers with different IP addresses are connected to such as a broadband access network may need to guarantee some bandwidth for premium subscribers while normal users should receive the fair share of the residual capacity. Downstream aggregate flow for a given subscriber can be uniquely identified by the destination IP address that is the key of the aggregate flow table.

Tenant performance guarantee/fairness at data center networks: A multi-tenant data center network where each tenant is isolated to a different physical server can use the source IP address as the key of the aggregate flow table. In case of virtualized servers, we may distinguish tenants with the VLAN ID field and use it as the key. Provisioning of a guaranteed service rate or fair share of service bandwidth is applicable according to the service level agreement (SLA) with tenants.

2.2.3 Low-Cost Flexible Implementation

OpenFlow and flow processing engines are usually built on ASIC or FPGA [5], [12]; they also often utilize expensive TCAM [1], [3] or SRAM [5] for wildcard rule matching or flow data structure caching, respectively. OpenQFlow, on the other hand, is built on a multi-core processor without TCAM or SRAM for the current implementation; it runs on a commercial microTCA chassis with a Cavium OCTEON Plus CN5650 advanced mezzanine card (AMC) module inserted [13], [14]. The implementation details can be found in Sect. 4.

OpenFlow has a mandatory requirement of flexibility so that it can cope with various network experiments in campus size networks since its initial launch. However, experimental functionalities should not be put on switches but on the controller and thus switch implementations have become rigid and not capable of embracing extensions. Recent studies including [3] point out that the overhead due to frequent communication between switches and a controller is huge and having controller-only-identifiable flow entries in the flow tables is not scalable; implementation of intermediate functionalities, between the burdensome controller invocation and the simple switch forwarding action, in the switch data path is advised.

In comparison to [3], the rule cloning is already a part of OpenQFlow implementation as the flow state table is an exact matching table for every microflow; flow statistics counters are maintained in the granularity of microflow in OpenQFlow. As to local actions, [3] suggests two actions: rapid re-routing and multipath support. OpenQFlow supports rapid re-routing by having a failover port information, and multipath support is deliberately omitted and the dynamic flow scheduling support is implemented.

Sophisticated flow-based QoS support is one of the intermediate functionalities. With OpenFlow, the controller needs to calculate the fair share of flow bandwidths — both aggregate flow and microflow — to deliver the required queue assignment or rate adjustment actions to the switch as flow entries; it is, on the other hand, efficiently delegated to the switch data plane in OpenQFlow for much reduced overhead. The details of QoS architecture of OpenQFlow are described in the next section.

3. QoS Architecture of OpenQFlow

3.1 Motivation

QoS provisioning has not been a major issue of SDN yet. Although the controller is highly programmable — thus can equip with a wide variety of QoS policies — *set_queue*, assigning a packet to the specified output queue, is the only QoS action of OpenFlow switches according to its newest 1.2 specification as of now. Fair queuing of fine-grained flows, such as microflows, is not feasible; the number of output queues seldom exceed several hundred in commodity systems and microflows account as many as a million at backbones [5].

Without having a proper degree of QoS facilitation in switches, the persistent overhead between the controller and switches brings another scalability issue as the number of flows increases for two reasons. First, the controller should do the frequent QoS supporting calculations and decisions for every QoS enabled output ports of switches. Second, switches need to deliver its measured statistics information back to the controller over and over to keep the controller up to date. Furthermore, OpenFlow switches are not flexible as controlling of individual flow rates is not feasible either.

In this section, we present our packet scheduling scheme that provides max-min fairness without the need of output queues per flow. A rate-limiting and bandwidth guarantee are also plugged by controlling the key parameter of fair bandwidth. An extension to the two-tier QoS provisioning is finally derived.

3.2 Basic Flow Management and QoS Architecture

A number of packet scheduling algorithms have been proposed to provide delay and fairness guarantees including Weighted Fair Queueing (WFQ) and other Generalized Processor Sharing (GPS)-related fair queueuing algorithms [15]. Those packet scheduling algorithms, however, have three technical issues which prohibit using them directly in OpenQFlow. First, the fair queuing mechanisms are not feasible to schedule several millions of microflows since they assume a separate physical queue per each microflow. Second, the packet scheduling algorithms require to maintain reference times — called system virtual time or system potential — to equalize the amount of service, i.e., bandwidth, among flows as closely as possible. There is a trade-off between the complexity of the system and the accuracy of the reference time which impacts the fairness and delay characteristics of a flow. Lastly, the fair queueing algorithms do not integrate buffer management mechanisms such as random early detection (RED) and its variants [16], which encourage us to propose an integrated solution.

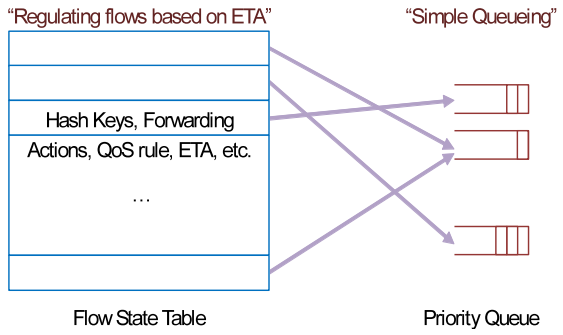


Fig. 3 ETA-based packet scheduling.

OpenQFlow takes a similar approach to address the above three issues as Rate Controlled Static Priority (RCSP) [17]. The packets pass through the eligibility test according to a regulation policy and only the eligible packets are queued into the priority queues as shown in Fig. 3; this removes the necessity to maintain several millions of per-flow queues. The packets are regulated by the estimated time — which is called estimated time of arrival (ETA) — when the packet is supposed to arrive according to its allocated bandwidth. The simple architecture addresses the first issue discussed in the preceding paragraph; we just need to maintain several priority queues rather than several millions of physical queues per individual flow.

The ETA is based on the max-min fair share of each flow; the fair-share is a time-varying value which is dynamically adjusted according to the link utilization [18]. The time-varying fair share traces the exact bandwidth share that each flow is entitled to have at the moment. The exact trace of fair share makes the proposed ETA-based packet scheduling algorithm use simple system real time clock rather than complex reference time; the second issue discussed in the preceding paragraph is addressed. The ETA-based packet scheduling algorithm performs simple eligibility test by comparing the ETA of each packet with its actual arrival time. We could drop packets on the fly if they do not pass the eligibility test since the fair share is the maximum bandwidth up to which a flow can use in the moment; we do not need a separate buffer management mechanism to control traffic congestion in the microflow level.

The ETA of k-th packet of a flow i, E_i^k is formally defined as follows.

$$E_i^k = E_i^{k-1} + l_i^k / b_i(t), \tag{1}$$

where $k = 1, 2, \cdots, l_i^k$ is the length of k-th packet of flow i, $b_i(t)$ is the fair share of flow i, and E_i^0 is the current system time. The ETA for an incoming packet is pre-calculated before the packet enters the system as per the Eq. (1) and is compared with the system time at the time of packet arrival. If the system time is larger than or equal to the ETA of the packet, the packet is scheduled to a priority queue; otherwise the packet is dropped since it enters the system violating the fair share.

We call this algorithm the basic ETA-based packet scheduling algorithm or BETA for the convenience of telling it from the extended version of the algorithm in next subsection. The fair share of BETA has the following properties which are different from the original max-min fairness'.

– $C/N \le \inf(b_i(t)) \le C,$ (2)
where $b_i(t)$ is the fair share of flow i and C is the total bandwidth;
– $b_i(t) = b_j(t)$ (3)
for all i, j;
– $\sum_{i=0}^{N} b_i(t) \ge C,$ (4)

Equation (2) explains the fair share of a flow is larger than or equal to the minimum share, C/N while not exceeding the total bandwidth. Eq. (3) means that he fair share of each flow is the same with each other. Equation (4) is self clear from Eqs. (2) and (3); the sum of all $b_i(t)$ is always larger than or equal to the total bandwidth.

The fair share in our algorithm can be described more clearly in an example. We take an example where a flow i whose initial bandwidth request is smaller than the minimum bandwidth share, C/N and the flow dynamically changes its bandwidth request during the lifetime of the flow. In the original max-min fair share, the $b_i(t)$ of the flow i starts from a value less than C/N. Therefore, C/N is not guaranteed because some packets may be dropped before the $b_i(t)$ is adjusted to a sudden bandwidth increase. In BETA, however, the fair share of a flow is at least equal to the minimum share, C/N and is guaranteed in any circumstances. The more details are explained in Sect. 3.3 using Algo. 1.

3.3 Two-Tier Flow Management QoS Architecture

This subsection provides the two-tier flow management architecture which is extended from BETA. We call this architecture the Complete version of ETA-based scheduling algorithm or CETA. BETA treats all the microflows equally without considering the origin and destination of the flows, which brings unfairness issue at aggregated flow level. For example, if there are two users, user A and user B, who are using five sessions and one session to download a file, respectively. The six sessions are equally treated and share the total bandwidth fairly in BETA. However, if we see this in each user's perspective, user A receives four times more bandwidth than user B. The two-tier flow management addresses this unfairness issue at aggregate flow level by managing aggregate flows as well as fine granular microflows. The ETA calculation and eligibility test in BETA are based on the fair share of the microflows regardless of their aggregation groups. CETA, however, calculates each microflow's fair share based on the fair share of each aggregate flow which is calculated separately.

The sequence is described in detail using pseudo code shown in Algo. 1. When a packet arrives at the system, the two hash values for both of microflow and its aggregate flow are generated using the combination of packet header fields as hash keys (GenerateHash). We use source or destination

IP address to identify an aggregated flow depending on its direction: source IP address for upstream groups and destination IP address for downstream groups.

Aggregate flows and microflows are managed in the aggregation flow table (AFT) and microflow table (MFT), respectively. For every input packets, the two tables are searched first. If a packet is the first packet of an aggregate flow j, a new AFT entry is created (CreateAFT) and the number of aggregate flows, N, is incremented by one. Then if the packet is the first packet of a microflow, a new MFT entry is created and the number of microflows of the aggregate flow j, n_j, is incremented by one. Forwarding rule tables and QoS rule tables are searched next to retrieve forwarding action and QoS profile information, respectively, then the ETA of the next packet of microflow i, E_i^1, is calculated based on the fair share, $b_i(t)$, and the arrival time of current packet, t_{now}. Finally, an MFT entry is created using all the above information for the next packet processing.

If there is a matching MFT entry, the packet goes through the eligibility test to decide whether the packet enters the system in time or not by comparing E_i^k against t_{now} considering burst tolerances: lower bound (BT_L) and upper bound (BT_U). There are three cases to consider.

Case 1) If a flow sends packets much faster than its fair rate, i.e., $t_{now} \leq E_i^k - BT_L$, the packets are dropped.

Case 2) If packets arrive within $E_i^k - BT_L \leq t_{now} \leq E_i^k + BT_U$, then the packets are accepted and the next E_i^{k+1} is calculated as in Eq. (1). The next ETA, E_i^{k+1}, is the time that it would take if the packet were serviced at the exact rate of b_i from the current ETA, E_i^k. Since E_i^k is larger than the current system time, t_{now}, it would have an effect of accumulating credits to the flow for the bandwidth that the flow did not use for its later uses.

Case 3) If a flow sends packets much slower than its fair rate, i.e., $t_{now} \geq E_i^k + BT_U$, the packets are accepted. However, if we calculate E_i^{k+1} as in case 2, the flow may accumulate its credits indefinitely for its future packet transmission which would cause unfairness. So the credit is reset and the next ETA, E_i^{k+1}, is set to the time that it would take if the packet were serviced at the exact rate of b_i from the current system time, t_{now}.

The above description assumed that fair share for a microflow i is given before a packet processing function runs the Algo. 1. We maintain a function for inducing the fair share of a microflow i, $b_i(t)$ separately from packet processing function. The fair share of a microflow is calculated by a two-step procedure: 1) calculating the fair share of an aggregate flow, 2) calculating the fair share of a microflow as the following equation.

$$b_i(t) = B_j(t)/n_j. \tag{5}$$

The above two-step procedure is self-clear since the fair share of a microflow in an aggregate flow should be equally distributed to all the microflows of the aggregate flow.

The fair share of an aggregate, $B_i(t)$, is obtained through an iterative process. Initially, each aggregate flow is allocated with the minimum bandwidth which is equally distributed to all the aggregate flows without considering any unused bandwidth. The minimum bandwidth is simply computed as such: C/N, where C is the total bandwidth that can be used by the flows. Then, if there is any aggregate flow j that does not use the assigned bandwidth, the unused bandwidth of the aggregate flow, U_j, is distributed equally to all the other aggregate flows. We should note that the unused bandwidth which is reassigned to other aggregate flows could be reclaimed by the original owners any time whenever they want. Therefore, if the throughput starts to exceed the total bandwidth as any aggregate flow starts to reclaim its fair share, then the equal share of the excess bandwidth is subtracted from each aggregate flow. So the fair share of an aggregate flow j at each iteration stage, B_j, is formally calculated as follows.

$$B_j = \left(C + \left(\sum_{k=0}^{N} U_k\right)\right)/N, \tag{6}$$

where the minimum value of $\sum_{k=0}^{N} U_k$ is zero and it is limited to the value such that B_j does not go over user defined maximum value of a flow. The value B_j keeps being adjusted by updating the unused bandwidth U_k, repeatedly.

Algorithm 1 Two-tier flow scheduling algorithm

Require: $AFT, MFT, b_i(t)$
while (*Packet*) **do**
  MakeHashKeys(*Packet*)
  t_{now} = GetSystemTime()
  match = LookupAFT() /* exact matching */
  if (!*match*) **then**
    $N++$
    CreateAFT()
  end if
  match = LookupMFT()
  if (!*match*) **then**
    n_i++
    $E_i^1 = t_{now} + l_i^0/b_i(t)$
    LookupFwdTbl() /* best matching */
    LookupQoSTbl() /* best matching */
    CreateMFT()
  else
    if $\left(t_{now} \leq E_i^k - BT_L\right)$ **then**
      DropPacket()
    else if $\left(E_i^k - BT_L \leq t_{now} \leq E_i^k + BT_U\right)$ **then**
      $E_i^{k+1} = E_i^k + l_i^k/b_i(t)$
    else
      $E_i^{k+1} = t_{now} + l_i^k/b_i(t)$
    end if
  end if
  UpdateMFT()
  PacketSechdule()
end while

3.3.1 Performance Analysis

We show that the fairness property of CETA comparing with other algorithms. We performed simulation using the NS-2 network simulator with the following conditions [19]. The topology is a simple dumbbell network with 4 subscribers, i.e., A, B, C, and D, with the 10 Mbps bottleneck link which is the same as in [20]. There are a total of 32

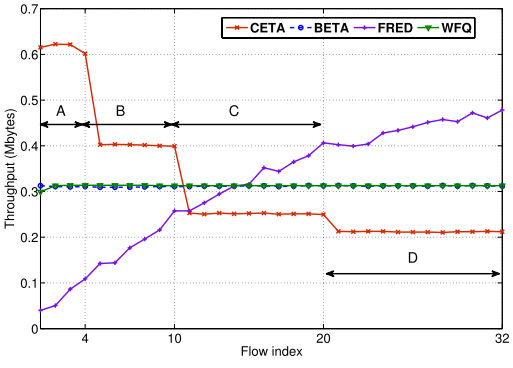
flows; the subscribers, A, B, C and D generate 4, 6, 10 and 12 UDP constant bit rate (CBR) flows, respectively. The flows are indexed from 1 to 32, where flow i sends i times of 0.3125 Mbps traffic; flow 1 sends 0.3125 Mbps traffic, flow 2 sends 0.6125 Mbps traffic, and so on.

We compare the average throughput of microflows in the BETA, CETA, Flow Random Early Drop (FRED) [16] and WFQ [15]. Figure 4 shows the average throughput of individual microflows and average throughput of each subscriber, respectively. Figure 4(a) shows that BETA and WFQ guarantee fairness among microflows almost perfectly while others do not. In BETA and WFQ, the max-min fair share of 10 Mbps bandwidth, i.e., 0.3125 Mbps, is equally allocated to each of 32 microflows. However, greedier flows receive more bandwidth share in FRED; the more traffic a micorflow generates, the higher average throughput it receives. In CETA, however, the microflows from the subscribers with larger number of microflows receive less throughput than others, which is fair in subscriber's point of view. The fairness property of CETA is more clearly shown by comparing the average bandwidth of each subscriber as in Fig. 4(b). CETA is the only one which could provide fairness among subscribers; BETA does not guarantee fairness among subscribers while microflows across all subscribers are fairly treated.

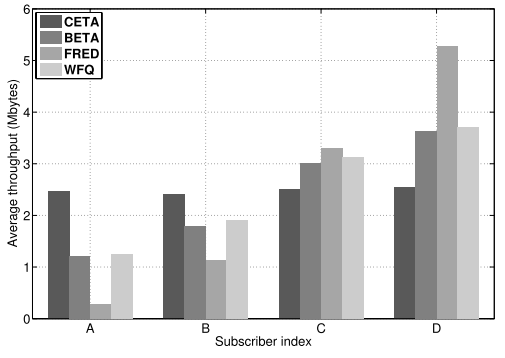
To represent fairness properties of CETA in a more formal form, we use the modified Jain's fairness index as follows [21].

$$FI = \left(\sum_{i=0}^{N} x_i\right)^2 / \left(N \times \sum_{i=0}^{N} {x_i}^2\right), \tag{7}$$

where x_i is measured throughput of subscriber i and N is the total number of subscribers. In case of CETA, FI is 0.99, whereas FIs of BETA, FRED and WFQ are 0.86, 0.62, and 0.86, respectively.



(a) Average throughput of each flow.



(b) Average throughput of each subscriber.

Fig. 4 Simulation results.

4. Implementation of Prototype System

4.1 Hardware Platform

We used Kontron OM5080 chassis, a popular off-the-shelf microTCA platform, which can be equipped with up to eight AMC modules of various kinds [22]. The two kinds of AMC modules, summarized in Table 1, are installed to the chassis for our prototype implementation [14], [23]. The modules can communicate with each other through an internal Ethernet network of the microTCA chassis.

The microTCA platform could work as two different kinds of systems depending on its configuration: a network system with multiple line cards where multiple data plane modules work as line cards and one controller module as a main controller card, or an SDN emulation network which consists of multiple packet processing nodes, i.e., SDN switches, and a remote controller.

4.2 Software Platform

4.2.1 Control Plane Architecture

We have two versions of the control plane software as our system supports both a legacy router with multiple line cards and SDN emulation network. XORP, an open source routing protocol suite, is chosen to provide IP routing protocols for the legacy router control plane [24]. XORP supports most of existing IPv4 and IPv6 routing protocols and has a well-organized object oriented software architecture implemented in C++, which makes it easy to add new software components on it. NOX or alternative OpenFlow controller software such as Floodlight can be adopted to support the SDN emulation network mode [25], [26]. Figure 5 shows the basic architecture of our prototype system. Both XORP and the OpenFlow controller module are running on the controller AMC while the agent component of each is running on the first core, i.e., No. 0 of CN5650, of the OCTEON AMC data plane module. The rest of cores in each data plane module are allocated for packet processing, QoS co-processing, scheduler, etc. More details are described in the

Table 1 The two AMC modules.

	Data plane module	Controller module
Name	AM4204	AM4011
Processor	Cavium OCTEON Plus CN5650, 600 MHz	Intel Core 2 Duo L7400, 1.5 GHz
Memory	2 GB DDR SDRAM, 800 MHz	4 GB DDR2 SDRAM, 400 MHz

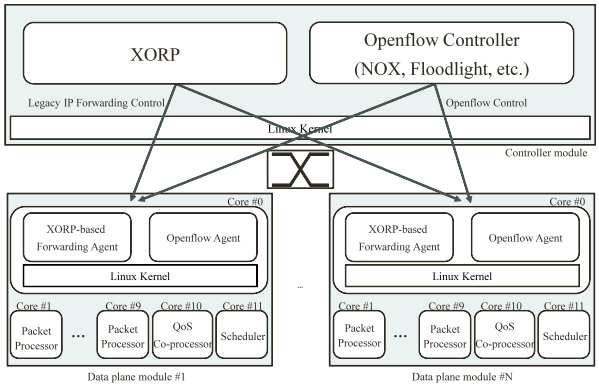


Fig. 5 Prototype system architecture.

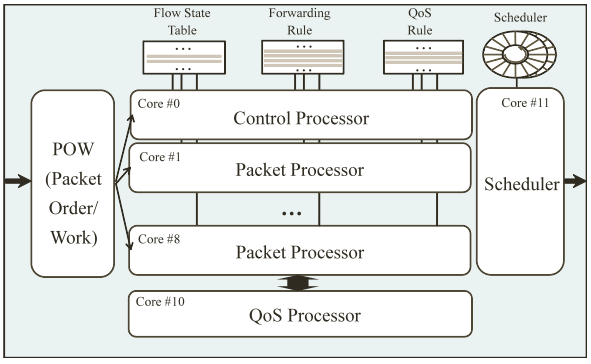


Fig. 6 Prototype data plane architecture.

next subsection.

4.2.2 Data Plane Architecture

Figure 6 depicts the data plane architecture of the system. The data plane module has the multicore processor, i.e., Cavium OCTEON CN5650, whose cores are partitioned as follows: 1 core (Core No. 0) for the OpenFlow agent, 8 cores, i.e., Core No. 1 ∼ Core No. 8, for flow-based packet processing, 1 core (Core No. 9) for future use of DPI and other value added functions, 1 core (Core No. 10) for QoS co-processing, and the last core (Core No. 11) for the scheduler. The fundamental data structures that are shared among different cores are forwarding rule table, QoS rule table, flow state table, and calendar queues. They are allocated by the core No. 0 as the type $CVMX\_SHARED$ so that they can be shared among the cores. The forwarding rule table is implemented in a 3-stage multibit trie with fixed strides of 16-8-8 whose memory footprint depends on distribution of IP prefixes [27]. Each leaf node of the multibit trie points to one of 2^{10} 40-byte-long nexthop information entries which is composed of forwarding action, i.e., either forward or drop, and nexthop interface information. The QoS rule table is implemented according to the Distributed Cross-producting of Field Label (DCFL) algorithm [28] whose each leaf node of the final composite label tree points to one of 2^{10} 24-byte-long QoS information entries which is composed of flow type, i.e., either a best effort flow or a guaranteed flow, bandwidth information for the guaranteed flow, and priority information. The flow state table is a hash table with 2^{20} 128-byte-long flow state information entries which is described more precisely in Sect. 2.

As to the packet flow at the data plane, an input packet comes into the packet order/work (POW) unit of the processor at which packets are distributed into the eight packet processing cores according to the hash value of the 5-tuple packet header fields so that the packet order is preserved in the same flow[†]. As soon as the input packet processing at the cores is completed, packets are inserted and waiting for being dequeued for output packet processing by the scheduler in an appropriate time slot of the calendar queue.

A calendar queue is allocated per each packet processor so that expensive shared memory contention among different packet processors can be avoided. Each calendar queue is composed of 16,384 time slots each of which spans eight microseconds. Multiple packets can be linked-listed in a single time slot if they are eligible to be sent in the same eight microsecond period. The scheduler runs an infinite loop that repeatedly checks the timer tick and sends out packets of the expired time slot from the calendar queues in a round-robin way.

4.2.3 Performance Analysis

In this subsection, we evaluate our OpenQFlow prototype implementation based on the control and data plane architecture described in above two subsections. To validate the scalability, the average number of instruction cycles spent on processing a packet is measured by varying the number of active flows. We used Cavium Octeon simulator oct-sim [29] for accurate instruction cycle measurement with up to 1 million synthetic flow generation[††] ; the measured instruction cycles are also verified in a testbed generating the flows with Ixia IxN2X test solution to our microTCA chassis [30]. As the instruction cycles cannot be measured in the testbed, we measured the system throughput with IxN2X and the cycles are calculated via Eq. (8).

Figure 7 shows the average number of instruction cycles for packet processing as a function of flow count. As processing the first packet of a flow requires a flow setup — while the subsequent packet processing can pass the setup procedure with faster flow lookup — we categorize the packet processing cycles into the two average groups of flow setup and flow lookup. Although the number of flows increases exponentially from ten to one million, we only notice at most sub-linear increase in the flow lookup group and even rather decrease in some intervals of the flow setup

[†]The hash based packet distribution also benefits the memory performance as there is no need for a mutex in the flow state table access as different cores access different hash entries.

[††]Each synthetic flow consists of a thousand 64-byte TCP packets and each flow is given with a unique 5-tuple by incrementing the destination IP address. Packets are fed to the simulator by round-robin in terms of flow.

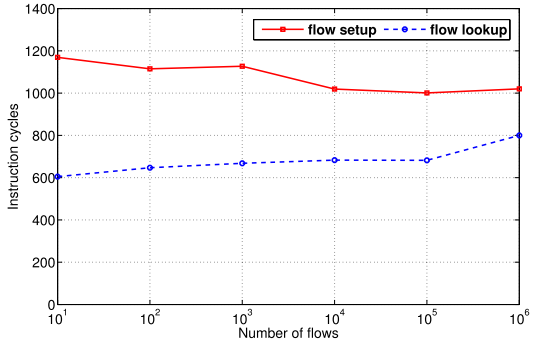


Fig. 7 The average number of instruction cycles for packet processing as a function of flow count.

group. Finally, we conclude OpenQFlow is scalable in terms of the number of flows.

The throughput, T in Gbps can be estimated as follows assuming packets are evenly distributed to the packet processing cores:

$$T = ((H \times N \times L))/C \times 8 \times 10^{-3}, \quad (8)$$

where H is multicore CPU clock in MHz, N is the number of packet processing cores, L is the number of packet bytes, and C is the number of instruction cycles spent on packet processing. The system throughput of our microTCA, measured by the IxN2X test solution, was around 3 Gbps regardless of the number of flows generated with the fixed packet bytes of 64. By Eq. (8) with 600 MHz multicore CPU clock, 8 packet processing cores and 64-byte packet size, the calculated number of instruction cycles is 800 which is consistent with our simulation result.

While we used Cavium OCTEON Plus CN5650 for our OpenQFlow implementation as this work has launched a few years ago, Cavium OCTEON II CN6880 processor, running 32 cores at 1.3 GHz, is recently released to the public. The system throughput of around 20 Gbps is extrapolated by the equation assuming the same number of instruction cycles for packet processing and 24 packet processing cores.

5. Conclusion

We proposed OpenQFlow, a novel SDN architecture that is a scalable and flexible variant of OpenFlow. OpenQFlow has a flow state table for flow tracking together with forwarding and QoS tables for flow classification instead of a single unified flow table that hinders both scalable deployment and flexible operation in various network environments. Flow-based QoS that provides performance guarantee and fairness among flows on both micro- and aggregate-flow level runs efficiently by utilizing the newly proposed CETA scheduling algorithm on a dedicated QoS co-processor and software calendar queues. Performance evaluation of OpenQFlow prototype on an off-the-shelf multicore platform shows that high-performance OpenQFlow data plane implementation is viable on a user friendly programming environment. We also expect that extensions of OpenQFlow that exploits the flexible multi-core processor implementation to provide value-added services or even drive a new networking in high-performance would line up in the near future.

Acknowledgments

This research was partly supported by the MKE (The Ministry of Knowledge Economy), Korea, under the ITRC (Information Technology Research Center) support program supervised by the NIPA (National IT Industry Promotion Agency. (NIPA-2012-H0301-12-1004)

References

[1] "Openflow switch specification, version 1.1.0." online. http://www.openflowswitch.org/documents/openflow-spec-v1.1.0.pdf
[2] M. Yu, J. Rexford, M.J. Freedman, and J. Wang, "Scalable flow-based networking with difane," SIGCOMM Comput. Commun. Rev., vol.40, no.4, pp.351–362, Oct. 2010.
[3] A.R. Curtis, J.C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "Devoflow: Scaling flow management for high-performance networks," SIGCOMM Comput. Commun. Rev., vol.41, no.4, pp.254–265, Aug. 2011.
[4] W. Kim, P. Sharma, J. Lee, S. Banerjee, J. Tourrilhes, S.J. Lee, and P. Yalagandula, "Automated and scalable qos control for network convergence," USENIX INM/WREN, pp.1–6, April 2010.
[5] P. Tian, X. Guo, C. Zhang, J. Jiang, H. Wu, and B. Liu, "Tracking millions of flows in high speed networks for application identification," INFOCOM, pp.1647–1655, March 2012.
[6] N.S. Ko, S.B. Hong, K.H. Lee, H.S. Park, and N. Kim, "Quality-of-service mechanisms for flow-based routers," ETRI J., vol.30, no.2, pp.183–193, April 2008.
[7] "The cisco quantumflow processor: Cisco′s next generation network processor," online. http://www.cisco.com/
[8] "Fast flow technology," online. http://www.anagran.com/
[9] "Abilene-III packet trace," online. http://pma.nlanr.net/Special/ipls3.html
[10] Y. Zhang, L. Breslau, V. Paxson, and S. Shenker, "On the characteristics and origins of internet flow rates," SIGCOMM Comput. Commun. Rev., vol.32, no.4, pp.309–322, Aug. 2002.
[11] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks," NSDI'10, pp.19–19, USENIX Association, Berkeley, CA, USA, 2010.
[12] "HP ProCurve 5400zl switch series," online http://www.hp.com/rnd/support/manuals/5400zl.htm
[13] "Octeon plus cn54/5/6/7xx hardware reference manual," online. http://support.cavium.com/
[14] "AM4204 user guide," online. http://emea.kontron.com/products/boards+and+mezzanines/advancedmc/io/am42xx.html
[15] A. Parekh and R. Gallager, "A generalized processor sharing approach to flow control in integrated services networks: The single-node case," IEEE/ACM Trans. Netw., vol.1, no.3, pp.344–357, June 1993.
[16] D. Lin and R. Morris, "Dynamics of random early detection," SIGCOMM Comput. Commun. Rev., vol.27, no.4, pp.127–137, Oct. 1997.
[17] H. Zhang and D. Ferrari, "Rate controlled static priority queueing," INFOCOM, pp.227–236, April 1993.
[18] J.Y.L. Boudec, "A tutorial on rate adaptation, congestion control and fairness in the internet." online. http://ica1www.epfl.ch/PS_files/LEB3132.pdf
[19] "The network simulator - ns-2," online. http://www.isi.edu/nsnam/ns/

[20] I. Stoica, S. Shenker, and H. Zhang, "Core-stateless fair queueing: Achieving approximately fair bandwidth allocations in high speed networks," SIGCOMM Comput. Commun. Rev., vol.28, no.4, pp.118–130, Oct. 1998.
[21] R. Jain, The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling, John Wiley & Sons, 1991.
[22] "OM5080 carrier grade amc platform," online. http://emea.kontron.com/products/systems+and+platforms/microtca+integrated+platforms/om5080.html
[23] "AM4011 user guide," online. http://emea.kontron.com/products/boards+and+mezzanines/advancedmc/processor/am4011.html
[24] M. Handley, O. Hodson, and E. Kohler, "XORP: An open platform for network research," SIGCOMM Comput. Commun. Rev., vol.33, no.1, pp.53–57, Jan. 2003.
[25] "FloodLight OpenFlow Controller," online. http://floodlight.openwhub.org/
[26] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, "NOX: Towards an operating system for networks," SIGCOMM Comput. Commun. Rev., vol.38, no.3, pp.105–110, July 2008.
[27] S. Sahni and K.S. Kim, "Efficient construction of multibit tries for ip lookup," IEEE/ACM Trans. Netw., vol.11, pp.650–662, 2003.
[28] D. Taylor and J. Turner, "Scalable packet classification using distributed crossproducing of field labels," INFOCOM 2005, pp.269–280, March 2005.
[29] "Octeon programmers guide: The fundementals." online. http://support.cavium.com/
[30] "Ixia IxN2X multiservice test solution." online. http://www.ixiacom.com/products/ixn2x/index.php

Nam-Seok Ko received the B.S. degree in computer engineering from Chonbuk National University, Jeonju, South Korea, in 1998 and the M.S. degree in information and communications engineering from Korea Advanced Institute of Science and Technology (KAIST), Daejeon, South Korea, in 2000. In 2000, he joined Electronics and Telecommunication Research Institute (ETRI), Daejeon, South Korea and has participated in several projects including developments of ATM switching systems and flow-based routers. Currently he is working towards the Ph.D. degree with the Department of Information and Communications Engineering, KAIST, Daejeon, South Korea. His research interests include network architecture, network protocols, traffic engineering and software defined networking.

Hwanjo Heo received the B.S. degree in electrical engineering from Korea University, Seoul, South Korea in 2004, and M.S. degree in computer science from Purdue University, West Lafayette, IN, in 2009. He is currently a researcher with Electronic and Telecommunications Research Institute (ETRI), Daejeon, South Korea. His research interests include network measurement, network protocols, and software defined networking.

Jong-Dae Park received his B.S., M.S. and Ph.D. degrees in electronic engineering from Yeungnam University, South Korea, 1985, 1987 and 1994, respectively. He was a research fellow in the department of electrical and electronics, Toyohashi University of Technology, Japan from 1995 to 1996. In 1997, he joined Electronics and Telecommunication Research Institute (ETRI), Daejeon, South Korea, where he is currently with the Net-computing Convergence Team as a team leader of the engineering staff. His current research includes packet forwarding engine and software defined networking.

Hong-Shik Park received the B.S. degree from Seoul National University, Seoul, South Korea, in 1977, and the M.S. and Ph.D. degrees from Korea Advanced Institute of Science and Technology (KAIST), Daejeon, South Korea, in electrical engineering in 1986 and 1995, respectively. In 1977, he joined Electronics and Telecommunication Research Institute (ETRI) and worked on the development of the TDX digital switching system family, including TDX–1, TDX–1A, TDX–1B, TDX–10, and ATM switching systems. In 1998, he moved to Information and Communications University, Daejeon, South Korea as a member of faculty. Currently he is a professor of the Department of Electrical and Electronics Engineering, KAIST, Daejeon, South Korea. His research interests are network architecture, network protocols, and performance analysis of telecommunication systems. He is a member of the IEEE, IEEK, and KICS of South Korea.