Design and implementation of Software Defined Hardware Counters for SDN

Tao Zhao, Tao Li, Biao Han*, Zhigang Sun, Jinfeng Huang

College of Computer, National University of Defense Technology, Changsha, Hunan, China

Abstract

In Software-Defined Networking (SDN), central controllers can obtain global views of dynamic network statistics to manage their networks. In order to support SDN controllers to obtain global information of the networks, the data planes need to maintain a large number of counters, which are typically implemented in hardware such as ASIC. However, implementation of these counters in hardware faces critical challenges: high memory consumption, control inflexibility, and low statistical accuracy. In this paper, we present the concept of Software Defined Hardware Counters (SDHC) for SDN, which offloads the management of counter updating to software and still maintains practical execution efficiency in hardware. Therefore, SDHC can allocate counter memory on demand to enhance counter utilization, which greatly reduces on-chip memory consumption. It is also able to allow controllers to flexibly control the counters through south-bound interface. Besides, with novel statistics feedback mechanisms, SDHC supports high-accuracy and active statistical requirement applications. Through a prototype implementation and performance evaluation based on FPGA and general processor, we reveal that the proposed SDHC is able to achieve high processing performance and high statistical accuracy, which incurs negligible updating delay to the switches.

© 2016 Elsevier B.V. All rights reserved.

1. Introduction

Software-Defined Networking (SDN), decoupling the control plane from the data plane, not only enables innovation and evolution, it also promises to dramatically simplify network management. In SDN, central controllers can obtain global views of dynamic network statistics to manage their networks in an efficient manner. Besides, it is required for controllers to perform various management actions, e.g., network measurement, intrusion detection, traffic engineering and so on. In order to assist the controllers in collecting these statistics to make decisions and support various network applications upon the control plane, SDN switches need to maintain a great number of various counters. For example, the OpenFlow specification (Version 1.4.0) [1] provides eight categories of counters (e.g., port counters, flow entry counters), which are furthermore divided into forty subcategories (e.g., received packets, received bytes for each flow entry). SDN controllers obtain the values of these counters through south-bound interface (e.g., OpenFlow [2]) in real time. Therefore, counters play an important role in both the data plane and the control plane in SDN.

As shown in Fig. 1(a), existing SDN counters are typically implemented in hardware such as ASIC [3]. In this paper, we call the counters implemented in hardware as hardware-defined and hardware-implemented counters, in which the relevant updating functions of counters are all implemented in hardware. In addition, most of these

* Corresponding author. Tel.: +86 0731 84575815.
E-mail addresses: taolili@nudt.edu.cn, Tao.Zhao@cs.uni-goettingen.de (T. Zhao), taoli.nudt@gmail.com (T. Li), nudtbill@nudt.edu.cn (B. Han), sunzhigang@263.net (Z. Sun), hjf@nudt.edu.cn (J. Huang).
counters only support passive statistical mode, i.e., the counters report their values to the controllers only when receiving requests.

Generally, the traditional ASIC-based counters on SDN switches are facing the following challenges:

(1) **High memory consumption.** Current SDN switches maintain a great number of statistical counters on ASIC, mainly including per-flow counters, per-port counters. For instance, there are 64K exact match flows for measurement in the V350-Centec OpenFlow switch [4], which consumes about 1.5MB counter memory besides far more memory consumed by flow table. Moreover, it will consume far more on-chip memory with the number of flows and statistical requirements increasing. However, memory space on limited ASIC area (e.g., areas of 35x35 mm [77]) is usually scarce.

(2) **Control inflexibility.** Diverse and real-time dynamic statistical requirements in SDN require counters to be more flexibly controlled. With the enhancement of SDN capability, more novel counters need to be supported (e.g., OpenFlow is always extending counters, as shown in Table 1). However, existing hardware-defined counters cannot be modified. Design and evaluation of new ASIC-based counters bring considerable costs to hardware developers. Meanwhile, with the diversity of applications, statistical requirements are becoming more dynamic. Existing counters cannot be flexibly controlled by controllers. Therefore, the counters which are not required by the applications still consume much memory space and updating overhead, leading to a waste of resources.

(3) **Low statistical accuracy.** To the best of our knowledge, current counters only support the passive statistical mode. In order to obtain periodic statistics for SDN applications under the passive mode, a controller needs to send counter-reading messages to SDN switches at some interval (e.g., every five minutes). This method brings long and unstable accessing delay so that it cannot accurately collect accumulated statistics within the interval. Meanwhile, it cannot obtain real-time and fine-grained statistics of traffic to support many more complicated applications since each time of counter updating cannot be grasped. Therefore, the passive mode will result in low statistical accuracy for various applications.

To reduce on-chip counter memory consumption, implementation of SDN counters can refer to the efforts on traditional counter implementation on network devices. For example, previous works proposed hybrid SRAM/DRAM counter architectures [8,9], SRAM-based approaches [10] and DRAM-based interleaving approaches [11,12] to implement counters. These approaches optimize the architecture of counter array to reduce on-chip memory consumption. However, as shown in Fig. 2(a), they are still hardware-defined, that is, each counter has its special and fixed memory space. Therefore, the hardware-defined counters will cost a large amount of memory with a large quantity of flows emerging. Additionally, they are inflexible, which cannot easily be modified to satisfy novel statistical requirements. In order to improve the flexibility of SDN counters, Mogul et al. proposed Software-Defined Counters (SDC) [3]. As shown in Fig. 1(b), it shifts counters from ASIC to switch-local CPU and main memory (DRAM), which we call software-defined by switch-local CPU and software-implemented counters in the paper. SDC allows flexible processing of counter-related information, and

![Fig. 1. Implementation of SDN counters: (a) traditional counters; (b) Software Defined Counters; (c) our proposed Software Defined Hardware Counters.](image1)

**Table 1.** Extension of Counters for OpenFlow.

<table>
<thead>
<tr>
<th>OpenFlow version</th>
<th>1.0</th>
<th>1.1, 1.2</th>
<th>1.3, 1.4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of counter categories</td>
<td>4</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>Number of counter subcategories</td>
<td>22</td>
<td>27</td>
<td>40</td>
</tr>
</tbody>
</table>

![Fig. 2. Mapping relationship between flow and counter memory: (a) traditional fixed mapping relationship; (b) SDHC-based flexible mapping relationship.](image2)
reduces on-chip memory consumption. However, because each update accesses DRAM, it will achieve low updating performance when a great number of flow rules exists. Meanwhile, SDC has limited flexibility, which cannot be flexibly controlled by controllers. Moreover, as far as we know, prior counter implementation approaches have not considered the problem of low statistical accuracy caused by the passive statistical mode.

In this paper, to overcome the aforementioned problems, we introduce the concept of Software Defined Hardware Counters (SDHC), which is software-defined by controller and hardware-implemented. As shown in Fig. 1(c), SDHC offloads managing functions of counter updating to software and still maintains practical execution efficiency in hardware. More specifically, as shown in Fig. 2(b), the mapping relationship between the counter definition (it indicates one counter, e.g., one byte-length counter) and counter memory is maintained in software on switch-local CPU, which is defined by a remote controller. The information parsing and counter executing logics are still implemented on ASIC of SDN switch. Besides, ASIC maintains a block of non-semantic counter memory with proper size for SDN counters, which is allocated to required counters. In this paper, we call this idea “decoupling counter definition and implementation”.

Our proposed hybrid software/hardware counter architecture balances the features of hardware and software. Software has higher computing performance and flexibility while hardware has much higher memory performance. The detailed reasons are as follows:

Implementing managing functions in software rather than in hardware. Firstly, based on our idea, mapping relationship table between the counter definition and counter memory consumes much memory space while these managing logics are relatively complex. However, current ASIC area is very limited, which cannot afford such memory space and logic implementation requirements. Secondly, implementing in software can keep flexibility of SDC and afford memory and complex logic requirements. Thirdly, with multi-core processor developing, CPU will meet higher counter updating performance requirement.

Maintaining counter memory in hardware rather than in software. Updating counters needs to access counter memory frequently. However, the access performance DRAM attached to CPU is low, which cannot meet the requirement of updating counters. Therefore, we maintain the counter memory based on SRAM on ASIC, which has much higher access performance.

There are many benefits of the proposed SDHC. Firstly, it can effectively reduce counter memory consumption through allocating counter memory on demands. Existing SDN switches always keep all categories of counters to ensure the adequate statistical requirements. Actually, only some of counters are required by a controller at a certain period of time, while the others which are not required still cost much memory space. Compared with traditional counters, SDHC allocates on-chip memory to actually required counters on demands of applications. Therefore, a block of memory space with appropriate size can satisfy most applications. Secondly, it can achieve high control flexibility, which can satisfy diverse and real-time dynamic statistical requirements in SDN. SDN switches always implement many function logics in advance to support new network functions [13,14]. Like SDN switches, without increasing memory space, SDHC allows implementing statistical logics in advance to support more new counters, which avoids redesign of ASIC. Meanwhile, according to diverse statistical requirements, controllers can flexibly apply/release various counters in real time through the south-bound interface. Thirdly, it supports active statistical mode with high statistical accuracy to incorporate more application requirements. With timer-triggering and updating-triggering statistics feedback mechanisms, counters can actively report their values to the controllers. Thus, SDHC can help applications collect real-time statistics of each arriving packet and accumulated statistics at a period of time.

Moreover, compared with SDC, through offloading counter memory from CPU to ASIC, SDHC avoids accessing DRAM during updating counter memory so that it has high and stable updating performance (The performance evaluation and comparison with SDC can be found in Section 6). To be implemented in existing SDN switches, we propose a three-stage modular system design of SDHC (The details will be shown in Sections 2 and 3). Besides, through key optimization technologies (Section 4 will present the details), it is potential to deploy SDHC on both commercial and experimental SDN switches in the future.

This paper offers an extension of our previous work [15] in SDN. In contrast to [15], the main differences of this paper include introducing novel active statistical mode, proposing SDHC model design details and evaluating real performance of SDHC through a prototype implementation. The main contributions of this paper are summarized as follows.

- We propose the concept and model of SDHC, which effectively overcomes the problems of existing counters. We introduce the active statistical modes, define key communication messages between CPU and ASIC, and propose a novel SDHC south-bound interface.
- We present an efficient and flexible SDHC model design to meet the requirements of SDN counters. Through appropriate modular design and efficient processing mechanisms, we guarantee that SDHC can achieve the high processing efficiency, and provide flexible control for various applications.
- We propose several effective message compression approaches to reduce the bandwidth requirement. We take OpenFlow as a typical example to present how to extend SDHC on a SDN switch, and show several use cases to illustrate the potential advantages of our proposed SDHC.
- Through evaluating our implementation of SDHC using FPGA and general processor as a prototype development platform, we reveal that our proposed SDHC is able to achieve high processing performance and high statistical accuracy, which incurs negligible updating delay to the switches.

The rest of the paper is organized as follows. Section 2 shows the overview of SDHC. Section 3 shows the design of SDHC model in details. In Section 4, we
present key technologies of SDHC. Section 5 show SDHC-enabled OpenFlow extension and uses cases. We evaluate performance of SDHC based on prototype implementation in Section 6. Section 7 outlines related work. Finally, we conclude the paper and point out the future work in Section 8.

2. Overview of Software Defined Hardware Counters (SDHC)

This section shows the basic idea and the workflow of SDHC, and proposes active statistical mode with high accuracy. Besides, we give the definitions of two key communication messages between switch-local CPU and ASIC and present SDHC south-bound interface in this section.

2.1. Basic idea

The basic idea of SDHC is to decouple counter definition and implementation. To reduce requirements of counter memory and achieve high flexibility, the managing function of counter updating is implemented on the switch-local CPU, which is controlled by remote controllers. Practical execution functions and counter memory are still kept on the ASIC, which guarantees high updating performance. In addition, the active statistical mode with high accuracy is introduced by SDHC in order to meet more statistical requirements.

As shown in Fig. 3, SDHC consists of the following modules:

1) Record Generator Module (RGM): RGM is responsible for processing incoming packets, including extracting key information of each packet, generating corresponding event record (ER) which will be introduced in Section 2.4 and transmitting ER to the switch-local CPU. RGM is implemented on the ASIC of SDN switch.

2) SDHC Supporter Module (SSM): SSM is mainly responsible for the management of counter updating. The module provides a flexible and open SDHC south-bound interface (we will describe it in details in Section 2.5) for external controllers and maintains a counter definition table (CDT), which controls counter updating. During updating counters, SSM parses received ERs, generates update requests (UR) (more details about UR will be introduced in Section 2.4) according to matched CDT entries and transmits them to the ASIC. Due to the high-flexibility requirement, SSM is implemented on the switch-local CPU.

3) Counter Executor Module (CEM): CEM is mainly responsible for updating the corresponding counter memory according to UR. In addition, CEM feedbacks counter values to controllers according to different statistical modes. To ensure high updating performance and high statistical accuracy, CEM is implemented on the ASIC.

2.2. Workflow of SDHC

As shown in Fig. 4, a SDHC-enabled controller can flexibly control counters to obtain required statistics. The whole workflow of SDHC contains four stages:

1) Counter applying. When an application on a SDN controller wants to obtain several counter statistics, the controller sends counter-applying messages, each of which includes a 3-tuple, to SDN switches through the south-bound interface. The 3-tuple indicates a counter definition, which is defined as a {RecordID, StatisticalMode, CounterType} tuple. Here, RecordID indicates a location number, e.g., a port number, a flow entry number on SDN switches, StatisticalMode denotes a statistical mode and CounterType indicates a type of a counter, e.g., packet-length. After receiving and parsing these counter-applying messages, SSM allocates a free counter memory address for each counter definition and installs a new 4-tuple {RecordID, StatisticalMode, CounterType, CounterAddress} as a new table entry in CDT.

Fig. 3. The architecture of SDHC.

Fig. 4. The involved behaviors of SDHC in SDN.
2.3. Active statistical mode

Comparing traditional counters only support the passive mode, we introduce two active statistical mode in SDHC.

Real-time mode: The counter will immediately report its value to the controller when being updated, called updating-triggering statistics feedback mechanism. The mode is specifically tailored for the network applications which need fine-grained and real-time statistics, e.g., network intrusion detection [16], fine-grained traffic analysis, Media Delivery Index (MDI) [17] measurement.

Timer mode: The counter will report its current value to the controller when the statistical timer of the counter expires, called timer-triggering statistics feedback mechanism. The timer mode can be extended to the periodic mode if configured. The value of the timer can be reset immediately when the timer expires, if a periodic flag for the timer has been set to be valid. The mode is especially useful for SDN applications, which require the periodic statistics of traffic, e.g., load-balancing [18] and traffic engineering [19].

As shown in Fig. 5(a), based on the traditional passive mode, to obtain the periodic statistics, a controller needs to use its internal software timer to initiate an interruption. Then the controller-local CPU responds to the interruption and sends a counter-accessing message to SDN switches. The method involves two stages: CPU interruption processing and network transmission. We assume that in the first counter accessing, CPU interruption processing delay is \( t_{\text{interruption}} \) and network transmission delay is \( t_{\text{trans}} \). So the total duration of the first counter accessing is \( t_{\text{interruption}} + t_{\text{trans}} \). Assume that the timer of the controller initiates an interruption every \( T \) second(s). Due to the unstability of CPU interruption processing and network transmission, in the second accessing, CPU interruption processing delay is \( t_{\text{interruption}} + \Delta t_{\text{interruption}} \) and network transmission delay is \( t_{\text{trans}} + \Delta t_{\text{trans}} \). So the duration of the second counter-accessing process is \( t_{\text{interruption}} + t_{\text{trans}} + \Delta t_{\text{trans}} + \Delta t_{\text{interruption}} \). Therefore, after these two accessing, the controller actually obtains the accumulated statistics in 
\[ T + (\Delta t_{\text{trans}} + \Delta t_{\text{interruption}}). \]

Therefore, if the mean rate of a flow is \( R_{\text{pkt}} \), the statistical error of obtained packet counts is \( R_{\text{pkt}} \times (\Delta t_{\text{trans}} + \Delta t_{\text{interruption}}). \) Meanwhile, the traditional mode cannot timely collect statistics in high-performance networks, where most of flows last a short time (the lifetime of some flows are even shorter than the delay) [20]. Therefore, this will lead to statistical errors. Moreover, in order to collect the real-time and fine-grained statistics, the controller must send counter-accessing messages at a quite high frequency. This will result in a large number of counter-accessing messages, which consumes high network bandwidth and cannot ensure the statistical accuracy. Therefore, the traditional implementation method based on the passive mode cannot afford the statistical requirements of current SDN applications.

Compared with the traditional method, SDHC can actively report current counter values to the controller. Fig. 5(b) shows that the controller-local CPU does not carry out the complex operations to access counters, dramatically reducing the number of counter-accessing messages and avoiding long and unstable counter-accessing delay.

2.4. Key messages between ASIC and CPU

Our proposed SDHC is designed as a hybrid software and hardware counter architecture. As key internal communication messages between the ASIC and the switch-local CPU during updating counters, the definitions of ER and UR are introduced in the section.

Event record. The message is responsible for delivering RecordID and key updating information (e.g., packet-length) from the ASIC to the switch-local CPU. On current SDN switches, counters at different locations may be in charge of collecting different statistics. Hence, ERs generated at different locations may contain different updating information, e.g., each flow entry typically collects received packet-count and byte-length while each port often collects the number of packet errors. Thus in our paper, we...
define a corresponding ER for each location. The locations in the OpenFlow specification [1] include port, flow entry, group, meter, etc. Note that there is no need for a duration field in ER since this field would cost much bandwidth. The duration counters can be implemented on the ASIC easily. In order to explain the definition of each ER well, we take the ER for each flow entry as a representative example.

When a packet matches a flow entry successfully, an ER is generated. The counters for each flow entry always focus on two statistics (i.e., received packets, received bytes) besides duration, so this ER has three fields:

- **RecordID**: the index of a flow entry matched by a packet successfully.
- **PktCount**: the number of the packet, which can be omitted because each ER can indicate one received packet.
- **ByteCount**: the byte-length of the packet.

The size of an ER depends on some switch parameters. For example, 16-bit RecordID supports over 60K location index range and 11-bit ByteCount supports 1518-byte packets.

According to the same principle, we also give the definitions of the other ER categories. Note that the format of an ER can be reconfigured only to slightly modify some hardware logics (no modifying is possible if original logics support them). Besides, although defining these ERs for different locations, there may be no need to implement all of them for limitations of ASIC area and CPU power.

**Update request**: The message is responsible for telling CEM which counter memory to be updated, how to be updated and what the statistical mode of the counter is. Each ER may trigger one or more counters to be updated, so CPU may generate one or more URs for each ER. Each UR triggers one counter to be updated, which has these following fields:

- **StatisticalMode**: the statistical mode.
- **CounterAddress**: the address of the counter memory.
- **CounterIncrement**: the increment of the counter value.

Here, StatisticalMode takes 2 bits to satisfy three statistical modes, the size of CounterAddress depends on the size of non-semantic counter memory and the size of CounterIncrement needs to afford all increments.

In Section 4, we discuss several effective techniques to compress ER and UR, in order to reduce the bandwidth requirement between the ASIC and the switch-local CPU.

2.5. SDHC south-bound interface

SDHC south-bound interface is the key of realizing SDHC. To let controllers flexibly control the SDHC-related behaviors, we define the SDHC south-bound interface between controllers and switches. As shown in Table 2, the SDHC south-bound interface mainly contains applying-related, releasing-related and reading-related messages. Note that the counter-applying message for the timer mode contains some timer parameters (e.g., cycle, the number of cycles) to configure the timers of the corresponding counters, besides a 3-tuple for a counter definition. Via these SDHC-related messages, applications on controllers can efficiently obtain their required statistics.

To implement the idea of SDHC in current SDN switches, we should extend OpenFlow, as current main SDN south-bound interface, to support these SDHC-related messages (we will describe the extension of OpenFlow in Section 5).

3. SDHC design details

Thus far, we have shown the SDHC architecture. In this section, we propose SDHC design details.

For convenience, the main notations used in this section are first illustrated in Table 3.

3.1. Record generator module

The main generator of RGM is to generate ER for each arriving packet and transmit these records to the switch-local CPU of the SDN switch. To guarantee its desirable processing efficiency, according to Fig. 6, an ASIC-based record generator module is proposed, which contains:

1. **Generator blocks**: Each location, which needs to collect statistics, is equipped with a generator block. Once a packet arrives at a location, its corresponding generator extracts key updating information and generates an ER. Then the generator stores this ER into its record buffer. For example, as for each port in SDN switches, once a packet arrives at a port, the generator for the port obtains some information of the packet (e.g., CRC error, byte-length) and generates an ER.

2. **Record transmitter**: The record transmitter is in charge of extracting ERs from a record buffer and transmitting these records to the switch-local CPU.

In order to meet the bandwidth requirement between ASIC and switch-local CPU, each generator block has a buffer, which is split into several blocks (Section 4.1 will describe the principle). When a block is full, the block requests the record transmitter to transmit its ERs to the CPU. To avoid the special case that no block of a buffer is always full, each buffer has a time threshold, indicating that the non-full but non-empty block must request to be transmitted once the buffer is overtime. The record transmitter transmits ERs using FRFT (First Request First Transmit) mechanism.
3.2. SDHC supporter module

SSM is responsible for processing SDHC-related messages and managing counter updating. Fig. 7 shows our design of SSM, which contains the following sub-modules:

(1) **Definition manager**: The definition manager is responsible for two main functions: i) Once receiving `apply_counter` messages from the controller, it extracts 3-tuples for counter definitions and allocates free counter addresses for these 3-tuples. Then it adds new table entries (each entry contains a 3-tuple and a `CounterAddress`) in the CDT. Meanwhile, it sends `apply_ack` messages to the controller. Additionally, if receiving `apply_timer_counter` messages, it needs to send another messages to the ASIC to configure counter timers. ii) Once receiving `release_counter` messages, it deletes the corresponding table entries according to 3-tuples and recycles their counter addresses. Then it returns `release_ack` messages to the controller.

(2) **Updating manager**: The updating manager is responsible for managing counter updating. It unpacks each ER and searches matched table entries according to the content of ER. Once finding matched entries successfully, it combines the corresponding information of ER and StatisticalMode, CounterAddress of the matched table entry, then generates and transmits the corresponding URs to the ASIC.

(3) **Statistics agent**: The statistics agent is in charge of handling statistics-related operations. As for the passive statistical mode, it receives `read_counter_req` messages and sends reading requests, which mainly includes counter addresses, to the ASIC. Then it receives counter values and sends them to the controller. As for the active statistical mode, it actively reports counter values to the controller.

As the kernel module of SDHC, the event processing performance of SSM is critical, which affects the whole updating performance. As shown in Fig. 8, the paper introduces a hash-table-based mechanism to reduce the number of comparisons for finding
Algorithm 1 shows the workflow of handling applied counter definitions. The algorithm starts from a list of counter definitions C. SSM takes a counter definition c from C and uses the hash function hash_func(c.rid, c.t) to find the index of table entry. If a free address is available (i.e., |A| > 0, where A indicates a set of free counter memory addresses), SSM allocates a free counter memory address a and builds a new table entry. Then SSM sends an ack message to tell the controller that the applying is successful. If no free address is allocated, SSM sends an ack message to tell the controller that the applying is failed.

**Algorithm 1 Applied Counter Definitions Handling.**

**Input:**
- C: a list of counter definitions c (|C| = n > 0);

**Output:**
- Ack: a list of ack;

```plaintext
1: for each c ∈ C do
2:     index = hash_func(c.rid, c.t); // hash mapping
3:     if |A| > 0 then
4:         entry[index].a = alloc(c); // allocate address
5:         entry[index].rid = c.rid;
6:         entry[index].t = c.t;
7:         entry[index].m = c.m;
8:         ack.append(c, 1); // successful
9:     else
10:        ack.append(c, 0); // failed
11:    end if
12: end for
```

In Algorithm 2, for each updating information u of each er, SSM firstly finds the matched table entry through the hash function hash_func(er.rid, type(er.u)), and compares RecordID of the er and RecordID of the entry. Here, type(er.u) indicates the counter type of u. If there exists matched entry, an ur is generated.

**Algorithm 2 Counter Definition Table Matching.**

**Input:**
- ER: a list of event records er (|ER| = n > 0);

**Output:**
- UR: a list of updating requests ur;

```plaintext
1: for each er ∈ ER do
2:   for each er.u ∈ Uer do
3:     index = hash_func(er.rid, type(er.u)); // hash mapping
4:     if er.rid = entry[index].rid then
5:         ur.append(entry[index].a, entry[index].m, er.u);
6:     else
7:         continue;
8:     end if
9: end for
10: end for
```

To further improve the processing performance of SDHC, we exploit the parallelized processing capability of multi-core CPUs. The approach for exploiting multi-core CPUs is the round-robin processing among multiple threads, which can achieve the load balancing among threads. More specifically, RGM sends generated ERs to each thread on the average, then each thread processes them using the RTC (Run To Completion) processing model.

We evaluate the paralleled version of SDHC (pSDHC) in Section 6 and find pSDHC can exploit the multi-core CPU to dramatically improve the performance of SDHC.

### 3.3. Counter executor module

CEM is responsible for the practical execution of counter updating, timer configuring and counter value reading. Meanwhile, we empirically analyze the reason for low memory consumption of SDHC. The key challenge for CEM design is how to correctly and effectively handle different statistical modes. As shown in Fig. 9, in our design, CEM contains:

1. **Counter updater:** The sub-module is mainly responsible for updating counters. The counter updater receives URs and stores them in an awaiting queue. Then the counter updater extracts and parses URs once. After that, it adds Counterincrement to the corresponding counter memory. In addition, the counter updater also judges whether StatisticalMode is real-time. If the mode is real-time, it will send a real-time triggering signal to the statistics manager.

2. **Statistics manager:** The sub-module handles counter values in accordance with different statistical modes. (i) For the real-time mode, it actively reads and reports counter values to SSM once receiving the real-time triggering signal. (ii) For the timer mode, if receiving timer configuration message from SSM, the statistics manager will configure the corresponding timer. Additionally, once the timer expires, the sub-module reports the corresponding counter values to SSM. (iii) For the passive mode, on each reading request arrival, the sub-module reports the counter values to SSM.

The module also maintains a non-semantic counter memory of modest size. The size of this memory theoretically depends on the maximum counter requirement at
the same time. As far as we know, applications on a SDN controller may only need some specific counters to collect statistics at a time. In other words, currently the counter requirement at a time is often much less than the number of counters on a traditional SDN switch. For example, many measurement studies showed a common observation that a small percentage of flows institute a large percentage of the traffic [21]. Besides, [22] shows that the top 9% of flows institute 90% of the traffic in bytes between all AS pairs. Therefore, we assume that in many specific network scenario, applications mainly focus on some specific flows like these “large flows” such that manufacturers actually only need to implement a portion of prior traditional counter memory. Given this principle, in general our proposed SDHC can greatly reduce the requirement of counter memory compared with traditional counters.

Based on SDHC, a great number of counters can share the on-chip counter memory without semantic. We think that on-chip SRAM of modest size can satisfy most statistical requirements. What if even ASIC cannot afford the requirement of the counter memory, the previous proposed hardware counter optimization methods can be applied to the non-semantic counter memory, e.g., hybrid SRAM/DRAM counter architectures [8,9,23], counter braids [10]. Of course, to avoid some specific cases that need much more counter memory, the counter memory can be extended to main memory (DRAM) if on-chip counter memory space is not enough.

4. Key technologies of the proposed SDHC

In this section we present several key technologies to optimize our proposed SDHC design.

4.1. Message compression between ASIC and CPU

In order to guarantee the updating performance, the link between switch-local CPU and ASIC must afford the bandwidth requirements of ER and UR. Current PCI Express (PCIe) can provide a quite large available bandwidth, e.g., PCIe 2.0 can provide 5.0 Gigabits per second per lane per direction raw bandwidth and support up to 32 lanes [24]. Thus in this paper, we choose PCIe 2.0 as the link between ASIC and CPU for SDHC implementation. However, high bandwidth requirement incurred by high-speed arriving packets in some networks (especially in high-performance data center) may exceed the capacity of PCIe, so several effective message compression approaches should be applied to compress these key messages between ASIC and CPU.

**Compressing event records.** ER can be compressed to reduce the upstream bandwidth from ASIC to CPU. In this paper, we propose a compression approach to reduce the bandwidth, whose basic idea is to enhance the available data utilization of PCIe as soon as possible. On a PCIe link, TLP (Transaction Layer Packet) is in charge of taking data to the memory attached to the CPU [24]. Meanwhile, the payload of each TLP must be no more than a maximal value, called \( MPL_{TLP} \) in this paper. Firstly, we can get the equation for computing the available data utilization of PCIe:

\[
U_{PCIE} = \frac{PL_{TLP}}{S_{Header} + PL_{TLP}}
\]  

(1)

Here, \( S_{Header} \) indicates the size of a TLP header and \( PL_{TLP} \) indicates the data payload of a TLP. As mentioned above, RGM sends a block of ERs every time. Hence, in order to achieve the maximum available data utilization, the size of each block should take maximum. We can obtain the maximal size through the following inference:

\[
N_{ER} = \left\lfloor \frac{MPL_{TLP}}{S_{ER}} \right\rfloor
\]  

(2)

\[
S_{Block} = N_{ER} \times S_{ER}
\]  

(3)

Therefore, according to Eqs. (2) and (3), we can get the maximal size of the record block:

\[
S_{Block} = S_{ER} \times \left\lfloor \frac{MPL_{TLP}}{S_{ER}} \right\rfloor
\]  

(4)

Here, \( S_{ER} \) indicates the size of each ER, \( N_{ER} \) indicates the number of ERs in a record block.

We assume that the size of each ER takes 32 bits (18-bit RecordID and 14-bit ByteCount). Considering the case that the size of a TLP header takes 16 Bytes and the maximal data payload of each TLP is 128 Bytes, we can figure out the optimal size of each record block, i.e., \( S_{Block} = 128 \) Bytes. Hence, the maximal available data utilization is \( 128/(16 + 128) \approx 89\% \) and the available bandwidth that PCIe provides reaches up to about 4.4 Gbps. However, does RGM send one ER in each TLP, the available data utilization is \( 4/(16 + 4) = 20\% \) and the available bandwidth is only 1 Gbps.

Through enhancing the available data utilization of PCIe, the practical capacity of PCIe can be boosted effectively. In addition, the compression approaches discussed in [3] can be exploited to compress ER, such as Huffman coding, shorter representation.

**Compressing update requests.** In order to reduce the downstream bandwidth from CPU to ASIC, we present three simple but effective compression approaches to compress UR.

One compression approach utilizes the statistical feature that the one-increment counter updating always accounts for the major proportion of the entire counter updating, such as error-count, packet-count, flow-count, and so on. The basic idea of this approach is that when the one-increment updating arises, there is no need for a CounterIncrement field since each UR can imply one increment. Therefore, this compression approach can effectively reduce the bandwidth requirement, especially in the case that the one-increment updating occurs frequently.

The second approach benefits from the spatial locality of applied counter address. Its main idea is that as for the counter definitions which have the same RecordID, the definition manager of SSM will allocate continuous CounterAddresses for them. Therefore, during the counter updating, URs generated by the same ER have continuous CounterAddresses, which will be gathered as the following compression format before being sent to the ASIC: \( \{ a_{init}, n, m_1, i_1, m_2, i_2, \ldots, m_n, i_n \} \). Here, \( a_{init} \) indicates an initial CounterAddress, \( n \) presents the number of URs, \( m_k \) and \( i_k \) respectively indicates StatisticalMode and CounterIncrement of the \( k \)th UR. In fact, each StatisticalMode and CounterIncrement corresponds to one CounterAddress in order. For example, the CounterAddress of \( m_k \) and \( i_k \) is \( a_{init} + k - 1 \).
The third approach is to keep a small cache, aimed at the passive statistical mode with low real-time demand. For URs which have the same CounterIncrement, SSM can accumulate the values of their CounterIncrement. When a cache-entry is full, the coalesced CounterIncrement will be sent to the ASIC.

4.2. Message compression between switch and controller

SDHC-related messages between switches and controllers allow controllers to flexibly control counters on switches. However, in real traffic, burst of flow often arises (i.e., a great number of back-to-back packets from a same flow arrive continuously) [25], e.g., a video stream. If some application needs to collect the real-time and fine-grained statistics of these flows, the real-time mode counters would report their values to a controller continuously and frequently. This will consume amounts of resources in switches and controllers, and much network bandwidth between switches and controllers, which may defer the processing of some OpenFlow messages and cause message loss.

To solve this problem, we present a compression approach to limit the rate of sending active_counter_return messages. SSM firstly stores these active_counter_return messages for the same real-time mode counter in a buffer. When the buffer is full, SSM merges these messages into one message, which contains the same 3-tuple for the counter definition and each counter value of each message. As a result, through configuring the threshold of the message buffer capacity, we can control the rate of sending real-time messages and avoid the case that there suddenly causes a great number of messages between switches and controllers for a burst of flow.

5. SDHC-enabled OpenFlow extension and use cases

In this section, we present the extension of OpenFlow to support SDHC and show several typical use cases to prove the benefits of SDHC.

5.1. SDHC-enabled OpenFlow extension

OpenFlow has become the de facto standard SDN south-bound interface [13]. Therefore, extending OpenFlow to support SDHC is promising and meaningful. From the implementation perspective, the SDHC-enabled OpenFlow extension is feasible and convenient, which contains two aspects: the extension of OpenFlow switch and the extension of OpenFlow controller.

(1) OpenFlow switch extension. It is easy to extend an OpenFlow switch to support SDHC, which need to implement two ASIC-based modules and one software module of SDHC in the switch. The paper has described the design of RGM and CEM in details. These two ASIC-based modules are mainly responsible for some simple functions, some of which (e.g., extracting the byte-length, checking errors of packets in each port) have been supported in current OpenFlow switches. Hence, only slightly adding some other simple logics (e.g., generating ER, parsing UR and actively reporting counter values) on the ASIC may complete the implementation of these two modules in OpenFlow switches. Additionally, implementing SSM on the switch-local CPU is easy to complete because software can be modified much faster and more easily than hardware.

(2) OpenFlow controller extension. The extension of OpenFlow Controller mainly involves enabling the SDHC south-bound interface. Our proposed extension approach is to add a SDHC driver module on the network operation system (NOS) (e.g., NOX [26]) of controllers. The SDHC driver module is mainly responsible for handling SDHC-related messages between controllers and switches. Besides, it provides the SDHC API for applications on the controllers. Because API is associated with the specific NOS, the paper does not refer to the definition of API.

Finally, we point out that although some details of our solution have been specifically designed for OpenFlow switches (e.g. to take into account the OpenFlow protocol), its general ideas are applicable to a class of recent SDN switches. Therefore, the applicability of our proposed SDHC is not limited to the OpenFlow switches but is far more general.

5.2. Use cases

In addition to supporting all existing traditional counters that OpenFlow supports, SDHC brings in flexible configuration, novel statistics and active statistical mode, which can meet the statistical requirements of more network applications.

(1) Flexible configuration. SDHC provides “reconfigurable hardware counters” for SDN applications. Through the SDHC south-bound interface, applications can reconfigure counters, i.e., apply/release counters on demand. For example, at time T1, application A1 on a controller wants to obtain statistics of flow set \( \{ f_1, f_2, f_3 \} \). The controller sends apply_counter/apply_timer_counter messages to add required counters on SDN switches. At time T2, application A1 does not need these statistics while application A2 needs to collect statistics of flow set \( \{ f_4, f_5, f_6, f_7 \} \). The controller firstly sends release_counter messages to delete counters for flow set \( \{ f_1, f_2, f_3 \} \), and then applies new counters of flow set \( \{ f_4, f_5, f_6, f_7 \} \). Though this is a simple example, this example represents statistical features of many applications in SDN, i.e., some of applications need some counters to obtain statistics within a period of time. Therefore, SDHC can meet most applications on SDN controllers at many scenarios, especially some scenarios where the on-chip memory space on SDN switches is limited.

(2) Novel statistics. Existing counter categories on SDN switches are limited, only collecting several basic statistics (e.g., byte-length, packet-count). Even though these statistics can support a wide range
of statistical functions, current SDN applications require more statistics that cannot be collected or calculated using existing counters. The proposed SDHC can introduce novel counter categories, which can support novel statistics, such that it is able to meet more application requirements. For instance, video stream detection like MDI [17] can be easily supported in a SDHC-enabled controller, because SDHC is able to count the RTP (Real-time Transport Protocol) [27] payloads, the number of loss packets for specific video flows, etc. Another example is that it can support traffic monitoring functions like RMON [28], which often collect the comprehensive statistics like the number of undersize/oversize packets. Note that in order to support diverse counters, SDN switches need to implement ASIC-based logic in advance that extract more key information of flows.

3. Active statistical mode. The paper has shown that the traditional passive statistical mode may lead to many statistical errors. Therefore, two active statistical modes are introduced in this paper. Most applications need these two high-accuracy active modes. For example, traffic load balancing often needs to obtain statistics at a period of time. Network security detection like DDoS detection is another example. With SDHC, they can accurately obtain traffic statistics (e.g., the number of loss packets) at a period of time. Additionally, video stream detection also needs these two active modes. Here we take MDI measurement application as an example to show how these two modes work. To obtain the real-time statistics of MDI (i.e., Delay Factor (DF) and Media Loss Rate (MLR)) [17] to evaluate the video quality, the application can apply some timer mode counters for the number of loss packets and some real-time mode counters for the byte-length of each packet.

6. Performance evaluation

In this section, we evaluate our implementation of SDHC using FPGA and general processor as a prototype development platform. Our evaluation focuses on three main aspects of SDHC: (1) processing performance; (2) statistical accuracy; (3) updating delay.

6.1. Experiment setup

To evaluate the performance of SDHC, we implement SDHC prototype based on self-developed network processing engine, containing Altera EP4SGX360 FPGA and PCIE 2.0, and general multi-core CPU (Intel Xeon CPU E5-2680 2.80 GHz) as shown in Fig. 10. In the SDHC prototype implementation, the record generator module and the counter executor module are implemented in the specialized network processing engine while the SDHC supporter module is running on the commodity multi-core CPU. Each thread processes the relevant functions using RTC processing model. Besides, the hardware platform also includes counter timer implemented in the counter executor module and a block of SRAM-based counter memory. PCIE 2.0 provides the bandwidth available for streaming key messages between the hardware platform and the software platform.

This paper focuses on verifying the feasibility of SDHC in the real world. Two types of experiments were performed: one is to verify the feasibility of SDHC from a hardware point of view, and the other is to verify the feasibility of SDHC from a software point of view. For the hardware evaluation, we use the same platform and environment as in the software implementation. The software evaluation is performed on the same platform and environment as in the hardware implementation. Meanwhile, due to the need to support more SDN functions, switch CPUs will become more powerful [3]. Moreover, high-performance multi-core processor is a powerful and promising platform for supporting complex SDN functions, some work [5,6] has been studied and deployed. Therefore, the evaluation results based on the multi-core processor platform also provide an important guiding meaning for SDN counters in the future.

6.2. Processing performance

Processing performance on the CPU is an important indication of SDHC performance, which decides the updating performance. We perform the event-processing performance evaluation from three aspects: (1) performance comparison between SDHC and SDC; (2) performance of pSDHC (the paralleled version of SDHC); (3) performance of different statistical modes.

Performance comparison between SDHC and SDC. To compare the event-processing performance of SDHC and SDC, we also implement a SDC prototype based on the same platform. The difference between SDHC and SDC is that SDC updates counters in DRAM attached to the CPU when receiving event records. In the paper, we emulate the real trace as soon as possible, we run RGM to generate event records for arriving packets, each of which contains a RecordID and two updating information (byte-length and packet-count) according to current typical statistical requirements. To emulate statistical location sets of various sizes on the SDN switch, we specify the range of randomly-generated RecordID. Thus based on these two
prototypes, we evaluate the event-processing performance of SDHC and SDC. All experiments are single-threaded.

From Fig. 11, we can observe that with the range of RecordID increasing, the event-processing rate of SDC decreases. The rate is probably limited by memory bandwidth. However, the rate of SDHC is relatively stable. This is because SDHC cannot be affected by the memory bandwidth. We can also observe that the processing rate of SDHC is inversely proportional to the number of counter updates triggered by each ER. This is because more updates consume more CPU overheads. Besides, Fig. 11 shows that the rate of SDHC are faster than that of SDC when the range of RecordID is relatively large. With the statistical requirements increasing, the range of RecordID will increase. Therefore, SDHC can support the ever-increasing statistical requirements better.

As shown in Fig. 11, the best-case event processing performance is about 11.2 event-records per microsecond and the worst-case performance is about 4.7 event-records per microsecond. From several data of OC192 traces provided by CAIDA [29,30] and some typical data centers [31,32], we can obtain that a mean packet size encountered by switches is between 400 and 800 bytes in fact. Therefore, SDHC can achieve the processing rate from 15.04 Gbps to 35.84 Gbps using 400 bytes as a conservative mean packet size. Meanwhile, because only some applications need to obtain statistics at a time, the case that each ER triggers one update on the average is regarded as a conservative case. From Fig. 11, the processing rate of SDHC can achieve 21.86 Gbps in the conservative case. Therefore, SDHC have high processing performance.

Performance of pSDHC. Fig. 12 shows the performance enhancement of the paralleled version of SDHC compared with SDHC. We can find that pSDHC effectively exploits the paralleled processing ability of the multi-core CPU to improve the updating performance. By enabling two threads, the processing rate almost doubles that of one thread. With four threads enabled, in the case that each ER triggers one update on the average, pSDHC reaches about 15.48 event-records per microsecond, which can achieve 49.54 Gbps processing performance using 400 bytes as a mean packet size.

Performance of different statistical modes. In the section, we evaluate the event-processing rate of each statistical mode in the conservative case. As for the timer mode, we respectively set 1 s, 1 ms and 1 μs as the timer value to evaluate the performance under different intervals.

Fig. 13 shows that (1) the performances of the 1-s and 1-ms timer modes are very close to the passive mode, this is because low frequency of counter-value returning hardly affects the processing performance; (2) the performance of the 1-μs timer mode is lower than that of the passive mode, this is because high frequency of counter-value returning consumes much more CPU overheads; (3) the real-time mode has a relatively low performance, in fact, its event-processing rate is inversely proportional to the rate of counter updating. This is because faster updating rate leads to more frequent counter-value returning messages which consume more CPU overheads. Therefore, the processing performance of the timer mode is often
the same as that of the passive mode because 5 min is a typical statistics report interval [33]. Meanwhile, the processing rate of the real-time mode can also achieve about 4.4 records/μs. In our future work, to enhance the performance of the real-time mode, we can set an update threshold to trigger the value reporting. Only when there are significant changes/updates, counter values are reported to the applications. The threshold can be configured by the SDHC API.

6.3. Statistical accuracy

Active statistical modes of SDHC bring in high statistical accuracy. In this section, we respectively evaluate the statistical accuracy of SDHC under the real-time mode and the timer mode to prove its high accuracy.

Real-time mode. To prove the statistical accuracy of the real-time mode, we measure the packet-count statistics of some specific flows using our SDHC prototype under the real-time mode. So we can obtain the returning values of counters in real time. Through analyzing these values, we can observe that the values of each flow are continuous to prove that we obtain statistics of each packet in these flows under the real-time mode.

Timer mode. Based on the timer statistical mode, we measure the packet-count statistics of a flow (its arriving rate is 3.125 Mpps) every 5 min. We choose to observe the statistical accuracy under the 5-min timer mode because 5 min is a typical statistics reporting interval [33]. The experimental result shows that the obtained statistics based on the timer mode are the same as the actual value (i.e., 9.3754 × 10^8 packets every 5 min). To demonstrate the low statistical accuracy of the traditional passive mode, we also measure the same statistics of the flow based on the passive mode. We implement a software timer in the CPU to obtain the statistics every 5 min. From Fig. 14, we can find that the statistics based on the passive mode are less than the actual value and continuously changing, which is due to the low accuracy of the software timer. If adding long transmission delay from controller to switch, the obtained statistics would be more incorrect and unstable.

6.4. Updating delay

While bringing in low memory consumption, high control flexibility and high statistical accuracy, the 3-stage processing model of SDHC leads to updating delay compared with traditional counters because of the communication between software and hardware. To correctly evaluate the influence made by the updating delay, we measure the updating delay by implementing a hardware timing logic in the SDHC prototype. The logic records the time of generating each event record and the time of updating the corresponding counter so that we can obtain each updating delay.

As shown in Fig. 15 and Table 4, we obtain the delays of 1 × 10^8 updates in total. From the delay statistics, we found 648 ns–680 ns updating delays account for approximately 90%. However, even in high-performance data centers, the network transmission delay between controllers and switches can be much more than hundreds or
thousands of nanoseconds [34]. Therefore, the updating delay incurred by SDHC can be ignored, which cannot affect the statistical accuracy of SDHC.

The above prototype implementation and performance evaluation reveal that our proposed SDHC is able to achieve high processing performance and high statistical accuracy, and the updating delay incurred by SDHC is negligible.

7. Related work

7.1. Traditional counters

Before SDN comes into being, many methods have been studied and proposed to optimize statistical counters on network devices (e.g., switches and routers). Most of these methods can be applied to the implementation of SDN counters.

To reduce expensive SRAM memory space, hybrid SRAM/DRAM counter architectures [8,9,23] were proposed. One typical example is a counter architecture introduced by Shah et al. [8]. The architecture implements small counters as a cache in SRAM and complete large counters in DRAM. The values of SRAM-based counters are periodically updated to the DRAM according to a counter management algorithm called LCF. Although these hybrid SRAM/DRAM architectures are proved to reduce the SRAM requirement, a great number of counters still consume a great amount of SRAM, which are difficult to be implemented on chip.

Besides hybrid SRAM/DRAM architectures, several SRAM-based approaches have been proposed, which make implementing a great number of SRAM-based counters feasible through some optimized representations. Lu et al. proposed a SRAM-based counter architecture called Counter Braids [10], which aims to reduce the SRAM requirement through optimizing its construction. Another SRAM-based approach called BRICK [35] is based on an efficient variable-length counter representation. But a large number of counters still require amounts of SRAM.

Additionally, the DRAM-based interleaving methods to implement large counter arrays were proposed in [11,12]. For example, Zhao et al. presented a randomized counter architecture, which only requires a little SRAM. Although these DRAM-based methods could reduce on-chip memory consumption and keep high update speed, high-bandwidth DRAM is not common on SDN switches. Meanwhile, a large number of counters still cost amounts of DRAM.

These implementation approaches can reduce on-chip memory consumption to some extent through optimizing architecture of counter array. However, these architectures are hardware-defined, which are fixed and cannot easily be modified to satisfy novel statistical requirements.

7.2. Software-defined counters for SDN

To optimize SDN counters, Mogul et al. proposed a new idea called SDC [3] for SDN counters. The authors think ASIC-based counters are inflexible. So they shift per-flow counters from ASIC to switch-local CPU and main memory (DRAM). Once a packet arrives, ASIC can generate a record containing updating information and store it in a buffer which is split into several blocks. When a block is full, the ASIC transmits the block to the CPU. Then the CPU unpacks these records and updates corresponding counters in DRAM. SDC makes the processing of counter-related information more flexible and reduces the ASIC area and complexity. However, each update needs to access DRAM, which involves cache hit rate. With the number of flow rules increasing, the cache hit rate will decrease so that the event-processing performance becomes lower. Besides, the flexibility of SDC is limited, which is only aimed at counters for each flow-table rule and cannot be controlled by controllers.

In this paper, we introduce a novel architecture of counters, which is software-defined by controller and hardware-implemented. While reducing the on-chip memory requirement and enhancing the flexibility of controlling counters, our proposed SDHC introduces new active statistical modes to increase the statistical accuracy.

8. Conclusion and future work

Our paper proposes the concept of SDHC, which offloads management function of updating counters from hardware to software and still maintains practical execution functions in hardware. Through allocating counter memory on demands, it effectively reduces the requirements of on-chip counter memory. By providing the SDHC south-bound interface, SDHC enhances the flexibility of introducing and controlling counters. Meanwhile, active statistical modes bring in high statistical accuracy compared with the traditional passive mode. Therefore, SDHC can meet diverse statistical requirements of applications in SDN. Prototype implementation and evaluation reveal that SDHC has high processing performance and high statistical accuracy, and negligible updating delay. We believe that SDHC is an exciting and promising idea for SDN counters, which accords with the development of SDN that makes network more flexible and manageable.

In future work, there are several research directions we intend to pursue to improve SDHC: (1) in order to vastly improve the updating performance, we aim to investigate better ways to parallelize SDHC; (2) we intend to theoretically analyze statistical requirement of some specialized scenarios using mathematical modeling, in order to qualify the practical requirement of un-sematic counter memory; (3) we need to further standardize the interface of SDHC to better promote the development of SDHC-enabled OpenFlow.

Acknowledgements

The authors thank the anonymous reviewers for their valuable and constructive comments. This work was
supported in part by the National 863 Program under Grant 2015AA016103, and in part by the National Natural Science Foundation of China under Grant 61202483, and partially supported by the Research Plan of NUDT under Grant JC15-06-01 and the Prospective Research Project on Future Networks of Jiangsu Future Networks Innovation Institute (BY2013095-1-17).

References

[15] T. Zhao, T. Li, B. Han, et al., Design of software defined hardware counters for SDN, Local and Metropolitan Area Networks (LAN-MAN), 2014 IEEE 20th International Workshop on, 2014, pp. 1–6.

Tao Zhao received the master degree in computer science from National University of Defense Technology (NUDT) in 2014. He is currently working toward the PhD degree in the Institute of Computer Science, University of Göttingen, Germany. His main research interests include network architecture and measurement for next generation Internet, social network, big data. He received the IEEE LANMAN Best Paper Award in 2014.

Tao Li is currently an assistant professor of computer science at the National University of Defense Technology, China. He received the PhD degree in computer science from National University of Defense Technology. He has been a visiting PhD candidate in Nanyang Technological University, Singapore in 2008. His main research interests include network architecture, network processor and router.

Biao Han is currently an assistant professor of computer science at the National University of Defense Technology, China. He received the PhD degree in computer science from University of Tsukuba, Japan in 2013. Before that, he received the B.E. degree and finished the master program in computer science in 2007 and 2009 respectively, both from NUDT. He has been a visiting scholar in the Department of ECE at University of Florida, USA. His main research interests include wireless network, software defined networking and network processor. He has published several refereed papers including...
Zhigang Sun is currently a professor at the School of Computer, National University of Defense Technology. He received his PhD in NUDT in 2001. His main research interests include architecture and node design for next generation Internet, computer communications, network processor, network protocol.

Jinfeng Huang is currently an associate professor at the School of Computer, National University of Defense Technology, China. His main research interests include network architecture and protocol.