International Journal of Innovative
Computing, Information and Control
Volume **17**, Number **3**, June **2021**

ICIC International ©2021 ISSN 1349-4198
pp. **1085–1102**

OPTIMIZING FLOW ENTRIES UPDATE IN SDN SWITCH WITH BATCH UPDATE MECHANISM

Jia Cui[1,2], Yan Jiang[1,*], Lei Song[1] and Xuewen Zeng[1,2]

[1]National Network New Media Engineering Research Center
Institute of Acoustics, Chinese Academy of Sciences
No. 21, North 4th Ring Road, Haidian District, Beijing 100190, P. R. China
{ cuij; songl; zengxw }@dsp.ac.cn; *Corresponding author: jiangy@dsp.ac.cn

[2]School of Electronic, Electrical and Communication Engineering
University of Chinese Academy of Sciences
No. 19(A), Yuquan Road, Shijingshan District, Beijing 100049, P. R. China

Received December 2020; revised April 2021

Abstract. *In software-defined network (SDN), lookup tree structure is proposed as a common organization of flow entries due to its high lookup performance. However, since tree structure takes full update, we found that every time the flow entry was updated, the lookup tree structure needed to be rebuilt, and with the increase of the tree depth, the update time was longer, which was a cumbersome process and resulted in a large number of packets lost during the update. This paper proposes a batch update mechanism, completing the batch integration based on the operation types of each Flow-Mod message, combining multiple one by one update into a single batch update. Therefore, processing multiple flow entries in the same batch only requires one centralized rebuilding, which greatly shortens the update time and enables the switch to respond to the changes of the network in time. We select the commonly used access control list (ACL) to verify the effectiveness of the proposed mechanism, and prove that it has a better performance in frequent update scenarios with high-flow. While retaining the high performance of lookup, it also meets the stringent requirements of SDN switch for updating.*
Keywords: Software-defined network, Flow entry, Update performance, Batch update

1. **Introduction.** In traditional networks, flow control and forwarding are completely dependent on routing devices, and the configuration process is cumbersome and error-prone. As a new network architecture, SDN is expected to change the traditional network pattern by adopting SDN switch based on the flow entries and centralized controller. In SDN, flexible application deployment is carried out by updating the flow entries in the switch. OpenFlow, as the mainstream southbound protocol of SDN, adopts Flow-Mod messages to update the flow entries one by one. Every time a Flow-Mod message is received, a lookup tree structure needs to be rebuilt. With the continuous operation of the network, the lookup tree structure is constantly expanding and deepening, and the time of each rebuild is also increasing, affecting the normal operation of the network.

Through the analysis of the above problems, it can be seen that the tree structure is used to improve the lookup performance, while the impact of the structure on the update performance is not considered. The update performance represents the response time to network status change, which becomes an urgent requirement in the current SDN environment. For example, failure recovery is to ensure the normal operation through strict time control. Similarly, the updating of network topology [1] and the configuration of network environment [2] also need to meet the demand of highly time-sensitives. Besides,

DOI: 10.24507/ijicic.17.03.1085

[3, 4] show that in the SDN data center, 90% of the data flows are rat flows with small data volume, while 10% are elephant flows which transfer more than 90% of the bytes, and the flow entries processing the rat flows occupy a large amount of the flow table space of the switch. By updating the flow entries which are used less frequently, we can make full use of the flow table resources.

Previous experiments have shown the average update performance ranging from 200 to 800 rules/sec [5-7]. Since the current update speed cannot meet the business requirements of high performance, several solutions have been proposed for minimizing the number of flow entries, optimizing the structure of flow entries and improving the matching efficiency [8-12]. However, these methods do not optimize the update process itself, which does not really solve the problem of slow update fundamentally.

This paper focuses on improving the update performance of SDN switches. Since update performance improvement should not be made at expense of lookup performance, the lookup tree structure is still used as the flow entries organization. We analyze the update process itself, which depends on several sequential steps: generate the rules by controller, issue Flow-Mod and Table-Mod messages to switch, parse the messages in SDN switch, and build the lookup tree that need full update for packets forwarding. We find full update to build lookup tree is a major time consuming factor. The delay of the fourth step (build lookup tree) has become a main bottleneck in the SDN actual operation and maintenance. This paper proposes batch update mechanism to reduce the time of building lookup tree to improve update performance. This mechanism consists of three phases: Sequential storage table (SST) is designed to cache Flow-Mod messages of the switch, then, when the batch update condition is met, the full rule table (FRT) is used to integrate the flow entries in the switch according to the operation types in SST, and finally, the lookup tree structure (LT) is built according to the flow tables contained in FRT to improve the lookup efficiency.

The main contribution of this paper is to optimize the flow entries update process itself, and to present a batch update mechanism to combine multiple updates into a single rebuild process, solving the problem with the current one by one mechanism that cannot meet the high update performance of the switch, so that the network data can be forwarded normally in the frequent update scenario. The SDN switch is installed on a general purpose processor, and we apply the batch update mechanism on SDN switch to verifying the update performance. It is estimated that when updating 1,000 flow entries in a single flow table, our solution is 150 times faster than OpenFlow using one by one update mechanism. In addition, the percentage of building time decreased as the number of flow entries increased, eliminating the performance bottleneck caused by building the lookup tree, which shows the updating efficiency of our mechanism is more applicable in high-flow environment. While taking account of the lookup performance, the batch update mechanism also realizes the rapid response to the frequent changes of the network, which is of great significance in maintaining the normal operation of the network.

The main works of this paper are as follows.

1) Analyze the time consuming factors in the one by one update mechanism by updating different number of flow entries.

2) A batch update mechanism is proposed to improve the update speed by reducing the build time of lookup tree structure, which is more applicable in frequent update scenarios, and the three phases of this mechanism are analyzed in detail.

3) Selecting the most commonly used ACL algorithm for the practical test, the proposed mechanism is evaluated in two cases, different number of flow entries and different level of flow tables. The experiments prove that this mechanism has a significant performance improvement in actual operation compared to the one by one update mechanism.

The paper is organized as follows: the related work is shown in Section 2 and Section 3 presents the batch update mechanism, Section 4 demonstrates the effectiveness of the mechanism through experiments, and Section 5 is the conclusion.

2. **Related Work.** Aiming at the problem of fast updating flow entries, there are several solutions: (i) delegate control back to the switch; (ii) cache common flow entries and change the storage structure; (iii) SDN switch cluster allocation.

As the update speed is affected by the limited bandwidth between the SDN control plane and forward plane, to solve the performance bottleneck caused by bandwidth, some scholars propose delegating control back to the switch, reducing the burden of the SDN controller. Ramos et al. [13] propose in routing the source and destination switches install flow table, by parsing packet header information, the switches update the flow table and forward data packets, and intermediate switches do not need a controller to guide update. The method alleviates controller load and improves update speed in the entire network. Yu et al. [14] choose the Authority Switches to manage the flow table in an area. However, this method does not conform to the current specification of OpenFlow protocol, which would require large-scale modification. Xie et al. [15] propose an automatic flow table control mechanism for OpenFlow switch, which solved insufficient flow table resources caused by the slow update in high flow switches. However, the proposed optimization of the stagnation time is still static, which cannot meet the demand of the switch in a real network environment.

Some scholars reduce the time of update by caching flow entries or changing the flow table storage structure [16, 17]. Syed et al.'s algorithm [18] changes the storage structure of TCAM by keeping free space in several parts of memory. Kim et al. [19] propose an improved updating algorithm based on LRU. The idle space is used as a cache for out-of-date flow entries, the flow entry is reactivated when a packet matches it, which can make the SDN switch store as many flow entries of high usage frequency as possible. However, at the peak hour of network, the flow table of SDN switch does not have much idle space for cache use, so it is meaningless to cache. Curtis et al. [20] propose two ways to reduce the number of communication between switches and controllers through rule replication and local operation. The switches install flow entries with wildcards in advance to reduce data interaction with controllers.

Switch cluster allocation mechanism is also used to improve the update performance [21]. Some technologies [22, 23] reduce the multi-switch delay by combining and filtering instructions in a global view. In [24], the capacity of different switches is used to constrain flow entries, so as to improve the flow table update speed of the whole network. However, when all the data flow into a switch, this switch must update frequently; meanwhile, there is still available storage space in other switches, which will increase the network delay. In general, existing solutions neither significantly change the performance of a single switch nor solve the problem from the update process fundamentally.

Based on the above analysis, in order to optimize the process itself for the practical application, we propose a mechanism to update flow entries quickly in SDN switch using batch update.

3. **Batch Update Mechanism.** Based on the architecture of SDN control and forwarding separation, when the network environment changes, the controller will send Table-Mod messages and Flow-Mod messages to the switches one by one through the southbound interface to implement the corresponding upper applications function logic [25]. The specific functions of these two message types are as follows.

1) Table-Mod message is used to update the flow table. All the flow entries in the switch are organized into different flow tables. There are only a few Table-Mod messages in the actual switch running, so there is no need to batch process such messages. We can process them one by one, and then load the flow tables into FRT directly, waiting for the flow entries belonging to the flow table to be updated.

2) Flow-Mod is an update flow entry message, which is one of the most important message types in OpenFlow. Each Flow-Mod message corresponds to an operation of a flow entry, and each flow entry is a forwarding rule. The main fields in the Flow-Mod message are tableId, index, and opType, which represent the flow table in which the flow entry is operated, the sequence number of the flow entry in the flow table, and the operation type of the flow entry, respectively. The opType contains three operation types for adding (Add), modifying (Mod), and deleting (Del) a flow entry. The values of these three fields are important parameters in determining how to do batch update. When the network environment changes frequently, the switch will receive a large number of Flow-Mod messages for flow entries update. This mechanism is designed to improve the processing performance of Flow-Mod messages, so as to realize the rapid response of corresponding network functions.

This section presents the implementation of batch update mechanism, and describes in detail the process of the switch from receiving the Flow-Mod messages used to update to actually using the flow entries for lookup. The mechanism includes the establishment of a sequential storage table (SST), a full rule table (FRT), and a lookup tree (LT) that can be used to forward element.

3.1. **Sequential storage tables (SSTs).** The SSTs are created in the control element of the switch. These SSTs are used to cache the Flow-Mod messages sent by the controller, and to decide the next operation by judging whether the batch update condition is met.

There are two ways for this mechanism to meet the batch update conditions.

1) Quantity condition (QC): Preset the quantity threshold and count the number of Flow-Mod messages on SSTs in real time, recorded as updateNum. When updateNum is greater than or equal to the quantity threshold, it means that QC of the batch update is met.

2) Time condition (TC): Preset the time threshold and calculate the time interval from the last batch update to the present, recorded as updateTime. When the updateTime is greater than or equal to the time threshold and there are unprocessed messages on SSTs, TC of the batch update is met.

Either the QC or TC is reached, batch update can be performed. By analyzing the main fields of Flow-Mod messages (tableId, index, and opType), batch integrates the flow entries into FRT. If the batch update condition has not been achieved, then it continues to cache the Flow-Mod messages to SSTs until the batch update condition is met.

We design SST as a list structure to ensure the orderly caching of Flow-Mod messages in the batch update mechanism. When receiving a Flow-Mod message, we should first parse its message content. Only when the tableId is less than the set maximum number of flow tables, the matching fields and actions in the message are not empty, can we insert the message to the end of the list. Otherwise, the message is dropped. Finally, we need to count the updateNum and updateTime to determine whether the batch update condition is met.

In order to ensure that the next batch of new messages can be cached normally and do not overlap with the current batch when processing messages, multiple SSTs are set in the switch, and the batch flag of each batch is marked, the sequence number is incremented

from 0. When the current batch meets the batch update condition and all Flow-Mod messages are integrated into FRT, the SST can be deleted.

3.2. **Full rule table (FRT).** FRT is used to maintain all the flow entries in the SDN switch and update the flow entries according to the Flow-Mod messages cached on SSTs. The process of FRT formation is divided into two steps: the addition of flow table and the integration of flow entries. After initializing the switch, the flow tables are directly added to the FRT based on the Table-Mod messages. FRT consists of multiple empty flow tables at this time. When the batch update condition is met, the Flow-Mod messages are taken out from the list header of SSTs successively, and multiple flow entries are integrated to the corresponding flow tables respectively according to the tableId field in the Flow-Mod messages. FRT is formed so that it contains the latest flow table and flow entries in the current switch. We count the number of flow entries that exist in the switch, denoted as entryNum.

When the Flow-Mod message is received again, the flow table to which the flow entry belongs is first found according to tableId. Then, the existing flow entries in FRT should be integrated based on the index and opType of the flow entry. After obtaining the index of the flow entry to be operated, it is necessary to clarify whether the flow table in the FRT contains the flow entry of this index. If this index already exists, and when the opType is Add or Mod, check whether the matching fields of this flow entry are consistent with the flow entry in the FRT. If so, the flow entry memory in SSTs is copied to FRT for overwriting and updating. If not, drop the message directly. When the opType is Del, the memory occupied by the flow entry in the FRT is cleared, and the entryNum minus one. If the index does not exist and the opType is Add or Mod, add the flow entry to the FRT and the entryNum plus one. When the opType is Del, the message is dropped directly. The current total number of flow entries in the switch is obtained at entryNum.

When the message in SST is empty, it represents the message processing of this batch is completed, and the number of batches, denoted as batchCount is counted plus one. At this time, the number of messages successfully processed in this batch is recorded updateNum. During the operation of the switch, the total number of messages updated is the accumulation of updateNum, denoted as updateTotalNum, which is commonly used to measure the complexity of the network environment.

3.3. **Lookup tree (LT).** When the FRT is generated, the forward element of switch can forward packets based on the rule of FRT. However, in practical application, the linear lookup time of the FRT is too slow. So when the switch deploys the flow entries, the tree structure [26] is usually adopted to improve the lookup efficiency. Therefore, the FRT needs to be compiled according to the relevant table lookup algorithm to generate the tree structure, so as to conduct efficient lookup.

In the process of building the LT, as the flow entries in FRT are classified into each flow table according to tableId, the LT is built in the unit of the flow table. The final update version of the flow table is formed by the LTs, which are compiled by multiple flow tables in the switch. The forward element forwards the data flow according to the rule in the flow table version, which is stored and maintained by the control element of the switch.

From the introduction and explanation of the above three phases, the pseudocode of batch update mechanism is summarized in Algorithm 1.

3.4. **Comparison.** Figure 1 shows a comparison of the two update mechanisms, and more intuitively shows the similarities and differences between the two mechanisms when updating flow entries.

Algorithm 1: The implementation of batch update mechanism

```
Input: Flow-Mod messages (contain tableId, index, opType), quantity threshold,
       time threshold
Output: Flow table version, batchCount, updateTotalNum
Add Flow-Mod messages to the sequential storage table (SST)
    if tableId < TableMax and entry != NULL then
        item = Storage node(entry, opType, NULL);
        if !head then
            head = item;
            tail = head;
        else
            tail → next = item;
            tail = item;
        updateList → entriesListInfo[tableId].head = head;
        updateList → entriesListInfo[tableId].tail = tail;
        updateList → entriesListInfo[tableId].updateNum ++;
Consolidate flow entries into full rule table (FRT)
    while updateNum ≥ quantity threshold ||
    updateNum && (nowTime − updateTime ≥ time threshold) do
        while item != NULL do
            if preUpdateRuleTable → entry.index == item.index then
                if item → opType == Add || Mod then
                    if preUpdateRuleTable → entry.match == item.match then
                        consolidateRuleTable → entryInf = item.entry;
                    else
                        drop this message;
                else
                    consolidateRuleTable → entryInf = item.next;
                    entryNum −−;
            else
                if item → opType == Add || Mod then
                    consolidateRuleTable → entryInf = item.entry;
                    entryNum ++;
                else
                    drop this message;
            item = item → next;
        batchCount ++;
        updateTotalNum = updateTotalNum + updateNum;
FRT to build the lookup tree (LT)
    for tableId = 0 → TableMax do
        Build LT for SDN switch to lookup;
    flow table version is formed by the LTs;
    updateTime = nowTime;
    return flow table version, batchCount, updateTotalNum.
```

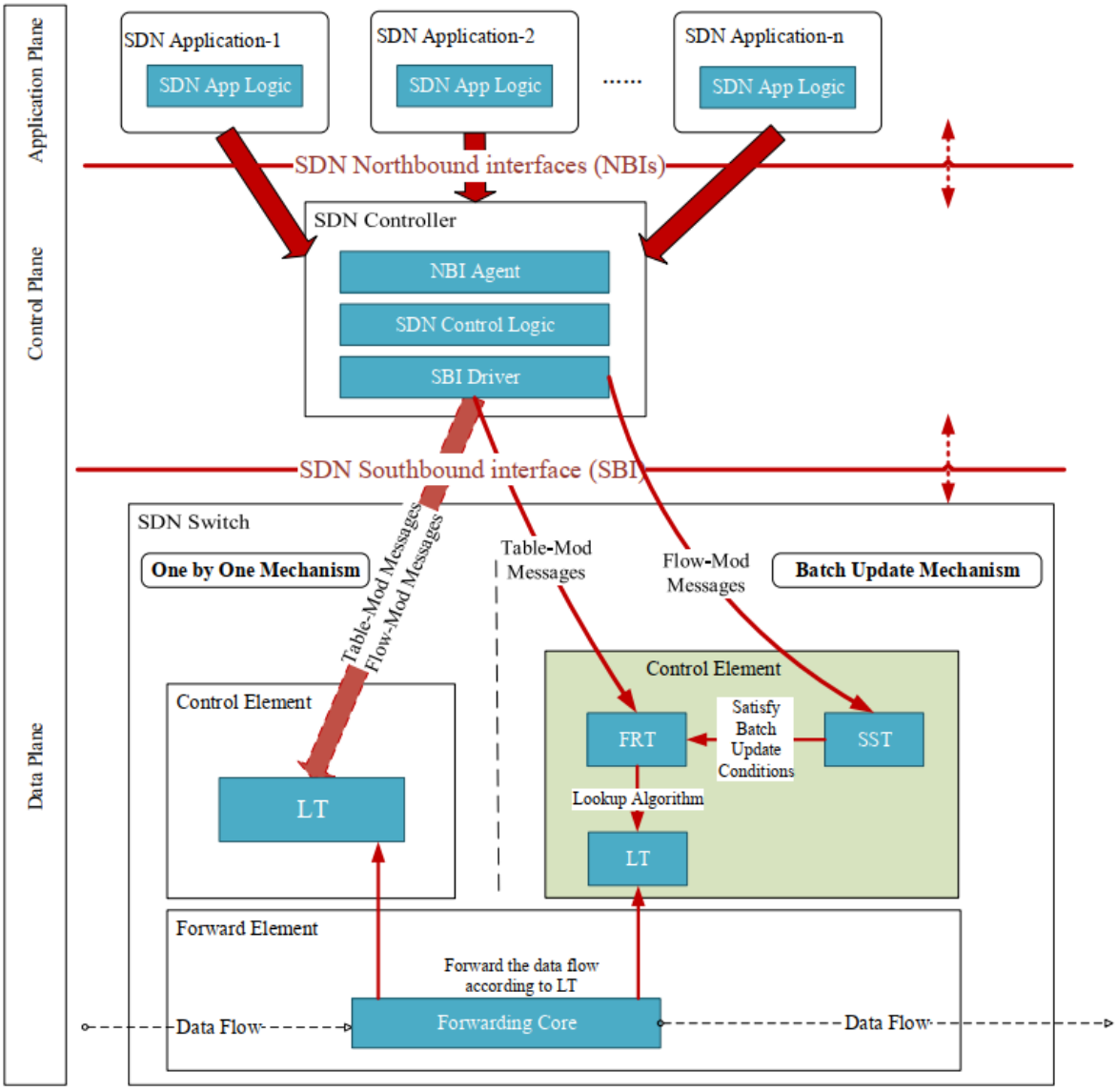


FIGURE 1. A comparison of the two update mechanisms

According to the above analysis and the architecture of SDN, when the application plane processing logic changes, the user transmits the network configuration to the control plane through the northbound interface protocol. The controller abstracts the whole network view as a network service, and uses the southbound interface to send the flow entries to the switch in the data plane, so as to realize the rapid response of the applications. All flow entries are updated and maintained in the control element of the switch, and the forward element of the switch is only responsible for lookup and data forwarding. Through the function division, two independent logical entities are formed to ensure the fast forwarding of the underlying data flow.

The main difference between the two mechanisms is that in the traditional one by one mechanism, there is no need to distinguish between the Table-Mod messages and the Flow-Mod messages, and they are processed one by one according to the distribution order. In the batch update mechanism, the Table-Mod messages and the Flow-Mod messages are processed separately. After receiving the Table-Mod messages, they are processed directly to create empty flow tables for FRT and wait for the flow entries to be updated. The

Flow-Mod messages are first cached in SSTs in the form of a list, then integrated into FRT according to the messages fields when the batch update condition is met. Compared with the one by one mechanism, which needs to rebuild the LT structure for each Flow-Mod message, the batch update mechanism directly merges the corresponding operations in the FRT. After all the Flow-Mod messages in this batch are processed, only one centralized rebuild operation is needed to generate the LT structure.

The main contribution of this paper is the batch processing of Flow-Mod messages: including the batch cache in SSTs waiting for update, then completing the batch integration of each flow entry into FRT directly, and finally rebuilding the LT of multiple flow tables in a batch at one time. The significance of batch update is to achieve multiple Flow-Mod messages processing only by rebuilding the LT once. The update performance is improved by reducing the rebuilding burden during the update process.

The above is the principle of batch update mechanism. In order to more specifically introduce how to use the mechanism for practical application, we give a typical example to illustrate the mechanism.

Example 3.1. *We suppose that the SDN switch receives 4 Table-Mod messages and 8 Flow-Mod messages from the controller within 1 ms. The Table-Mod messages are used to create 4 flow tables, Table1∼Table4. The contents of Flow-Mod messages are shown in Table 1. Among them, we specially send two special types of messages: No. 5 and No. 8, which mean to delete and modify the existing flow entries in the switch respectively. Meanwhile, we set the condition of batch update with a quantity threshold of 10 and a time threshold of 1 ms. The following shows how we use the batch update mechanism to process them.*

Table 1. Content of Flow-Mod messages in Example 3.1

No.	*tableId*	*index*	*opType*
1	*1*	*1*	*Add*
2	*2*	*1*	*Add*
3	*2*	*2*	*Add*
4	*2*	*3*	*Add*
5	*2*	*1*	*Del*
6	*4*	*1*	*Add*
7	*4*	*2*	*Add*
8	*4*	*1*	*Mod*

Figure 2 illustrates the implementation of the batch update mechanism, which is divided into five steps to introduce the whole batch process.

Step 1: Parse the contents of 4 Table-Mod messages, and create 4 empty flow tables in FRT according to tableId.

Step 2: Cache Flow-Mod messages in SST continuously until batch update condition is met. Set the batch flag of this SST as 0, the number of messages in SST and the update time is counted in real time, waiting for batch update conditions to be met.

Step 3: Due to the fact that the set quantity threshold is 10, 8 messages cannot meet the QC, but after the switch is more than 1 ms away from the last batch update, it can be batch updated according to the TC.

Step 4: When the batch update condition is met, remove all the flow entries cached in SST from the list header one by one, and each flow entry is integrated into the corresponding flow table according to the tableId, index, and opType. The operations of flow entries

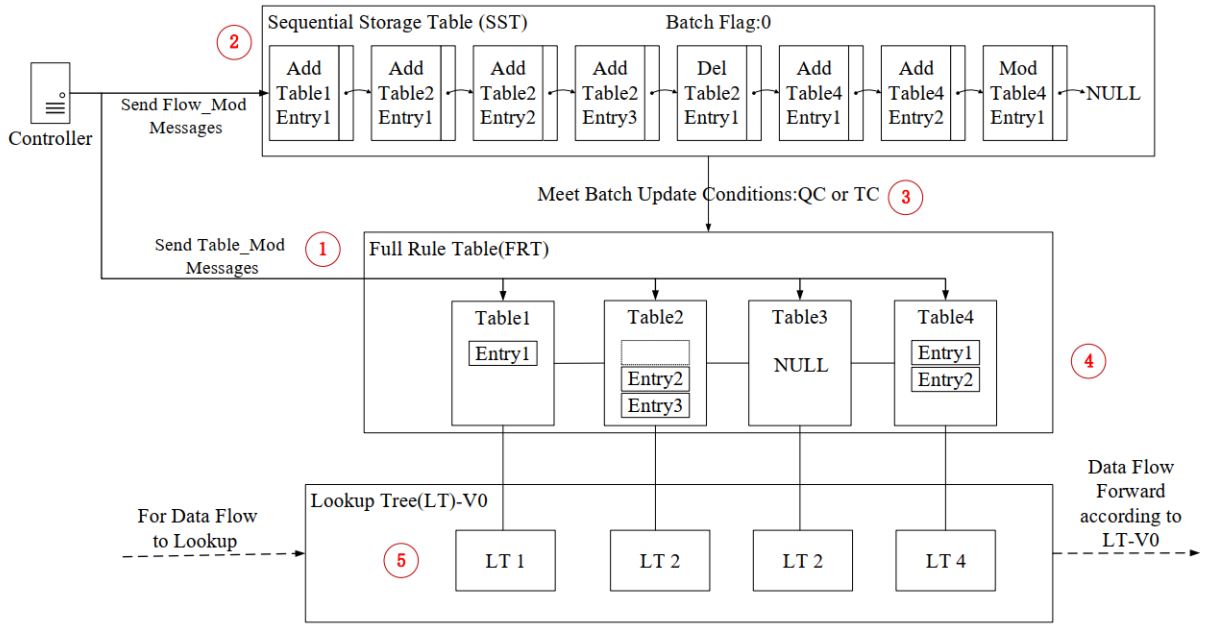


FIGURE 2. An example for the implementation of batch update mechanism

are directly executed in FRT at this time, when all messages are processed, the latest FRT in the switch has been formed, which does not contain the deleted flow entry table2entry1. Then table4entry1 is already modified.

Step 5: Finally, the LT structure is built based on the flow table, all the LT structures (LT1-LT4) made up the complete version of LT-V0, and the data flows entering the switch are forwarded according to the new rules in LT-V0.

4. **Performance Evaluation and Discussion.** Software verification and validation is an important process to ensure the normal implementation of the system and test whether it meets the development requirements [27]. In this paper, we use the deductive research approach to test in a specific scenario. First, we verify the update performance of the current switch, find the time-consuming factors, and then through the processing of batch update mechanism, software validation is used to judge whether the mechanism improves the update performance.

In this experiment, we compare and evaluate the performance of one by one and batch update mechanism in processing Flow-Mod messages. Among the numerous full update tree structures, the commonly used ACL algorithm is adopted to verify and validate the batch update mechanism. ACL is a collection of one or more flow rules, which is essentially a packet filter. We use Classbench [28] to generate flow rules with dependencies and flow update instruction sets. The flow rules refer to the judgment statement that describe the packets matching conditions, which can be the source address, destination address, port number, etc. The device matches packets based on these flow rules and can filter out specific packets. We install the OpenFlow software switch and implement the batch update mechanism on a general purpose processor.

The Flow-Mod and Table-Mod messages are issued by ONOS [29], an OpenFlow network controller. The data flows used for testing are generated by the Spirent Test Center instrument, which arrive at the SDN switch in consistent with the Markovian queuing model [30]. To eliminate the interference of other irrelevant factors, the experimental environment and the method for time interval measurements are the same. Our main

evaluation goals are to find out: (i) What is the most time consuming factor in one by one mechanism, (ii) What is the average time taken to update different number of flow entries, and (iii) The impact of multiple flow tables on updating performance.

4.1. **Analyzing time consuming factors.** The overhead of updating flow entries comes from two processes: preparing data and building ACL lookup tree. For building ACL lookup tree, the two update mechanisms are the same, but for preparing data process, the mechanism of one by one is used to realize the messages parsing, while batch update is mainly used to build SST and FRT.

We choose different number of flow entries (e.g, 10, 100, 1,000, 5,000) for testing, all flow entries are set in the same flow table, and one lookup tree is needed to build. To make the data more accurate, each experiment was tested 10 times and averaged. By the mechanism of one by one and batch update, the percentage of time to build ACL tree to the total time is shown in Figure 3.

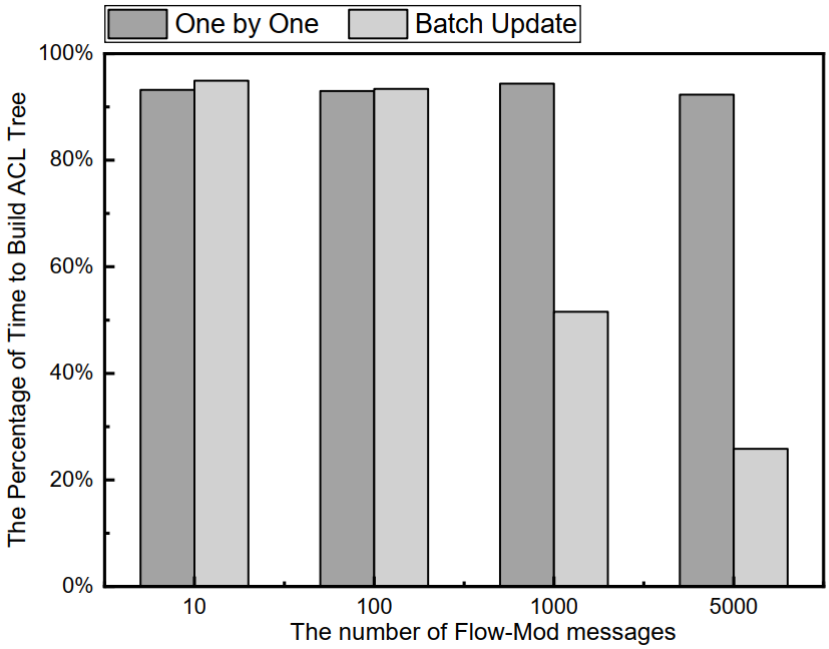


FIGURE 3. The percentage of time to build ACL tree to the total time in two update mechanisms

As can be seen from Figure 3, the time to build an ACL lookup tree accounts for most of the total time (about 93%) in the one by one mechanism, and the percentage is basically unchanged for different number of flow entries, indicating that in any frequent update scenarios, the updating performance is being affected by the build of full update lookup tree.

For the mechanism of batch update, as the number of flow entries increases, the percentage of time to build the ACL tree to the total time decreases. By reducing the time of rebuilding ACL tree, the update cost will be greatly reduced, showing batch update mechanism eliminates the performance bottleneck associated with time consuming factors.

4.2. **Comparing the performance of updating different number flow entries.** In this subsection, in order to evaluate the performance of updating flow entries, we need to calculate the time interval for a switch to successfully update a flow entry, from the switch receives a Flow-Mod message to the message actually takes effect.

4.2.1. *One by one mechanism.* In the mechanism of one by one, we set the flow entry to be tested with forwarding to a specific port, the rest does not contain the action of forwarding to this port, then send data flow matching the flow entry. When the message being tested arrives at the switch, record the time T_{t1}, after this message comes into force, there will be some data packets forward from the specific port, then record the time of the first packet to be forwarded, named T_{r1}.

When updating N flow entries, record the average update time T_{AO}, and the calculation formula of T_{AO} is

$$T_{AO} = \frac{\sum_{i=1}^{N} T_{r1}(i) - T_{t1}(i)}{N} \tag{1}$$

10,000 flow entries were updated one by one in the experiment, and experimental results of the 1st, 50th, 100th, 1,000th, 5,000th and 10,000th message update time are in Table 2. With the increase of messages, the update time of each flow entry increases. Considering the ACL lookup tree needs to be rebuilt whenever flow entry is updated, the more messages, the longer it takes to build the tree. The update time of No. 10,000 is 984 times as much as No. 1! Sufficient to prove that update process has a lot of room for optimization.

TABLE 2. Update time for special flow entries in one by one mechanism (total update number: 10,000)

Number of flow entries (No.)	Average update time: T_{AO} (ms)
1	0.10
50	0.50
100	0.75
1,000	9.18
5,000	47.22
10,000	98.44

4.2.2. *Batch update mechanism.* We set the number of messages to N in the batch update mechanism, and only the last message is forwarded to a specific port. When the first message arrives at the switch, record the time T_{t2}, after the last message of this batch comes into force, record the time of the first packet to be forwarded T_{r2}. Record the average total update time T_{AB}, and the calculation formula of T_{AB} is

$$T_{AB} = \frac{T_{r2} - T_{t2}}{N} \tag{2}$$

4.2.3. *Experimental design under different flow environments.* According to the experimental results in Table 2, the update time increases exponentially when the number of flow entries is more than 1,000. Therefore, the experiment takes it as the benchmark and divides into two update environments: low-flow ($N < 1{,}000$) and high-flow ($N > 5{,}000$). In the low-flow environment, each batch interval is 100, and interval of 5,000 is in the high-flow environment. We design the experiment in different environments with different update mechanism and different number of Flow-Mod messages, statistically the average time of each flow entry for performance comparison.

Considering the conditions for executing batch update are QC and TC (in Subsection 3.1), in this section, different quantity thresholds and time thresholds are selected to test

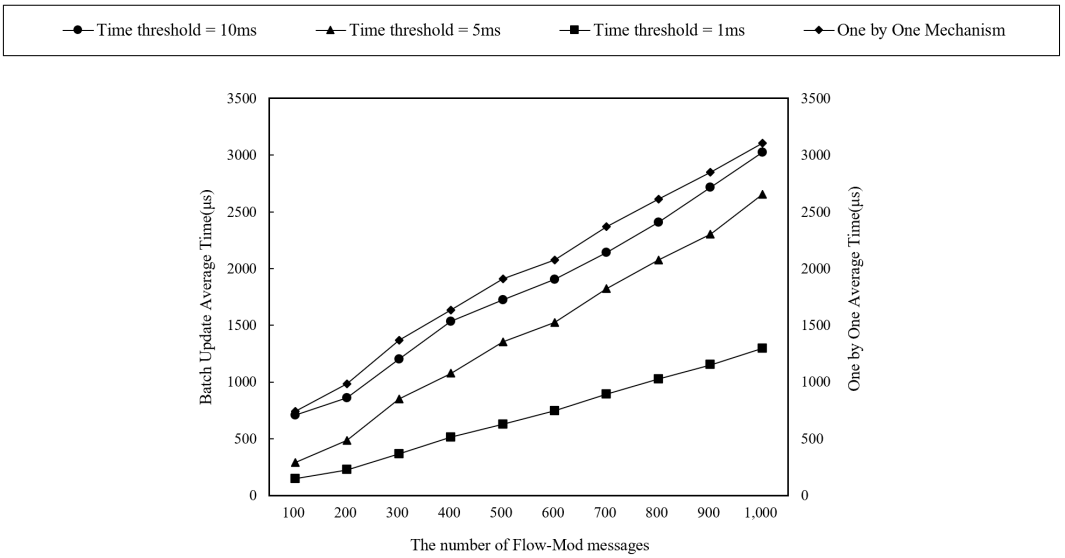
the update performance. Quantity threshold is defined as the minimum number of Flow-Mod messages processed in each batch, and time threshold is defined as the maximum time for each batch of SSTs to cache Flow-Mod messages, when one of these thresholds is met, the batch update mechanism can be executed.

Figure 4 shows the comparison of average time taken to process the different number of Flow-Mod messages in a low-flow environment (the number of update flow entries is less than 1,000). Each figure represents the updating performance of different time thresholds in the same quantity threshold. 1%, 10%, and 100% of the maximum number of update Flow-Mod messages are selected as the quantity threshold. From Table 2, we know that the time to update 1 message and 1,000 messages is 0.1 ms and 9.18 ms respectively in one by one mechanism, when the minimum quantity threshold is 10, it represents a batch will process 10 messages, so we design the minimum time threshold as $0.1 * 10 = 1$ ms for comparison. The maximum time threshold is set to be slightly greater than the time required in one by one mechanism, to ensure the maximum number of Flow-Mod messages can be updated in a batch, so 10 ms is set. Besides, 5 ms is set as an intermediate parameter, so quantity thresholds are selected as {10, 100, and 1,000} and time thresholds in {1 ms, 5 ms, and 10 ms} are picked representative parameters to verify the mechanism. Obviously, when quantity threshold = 1, time threshold = 0, it means updating one by one.

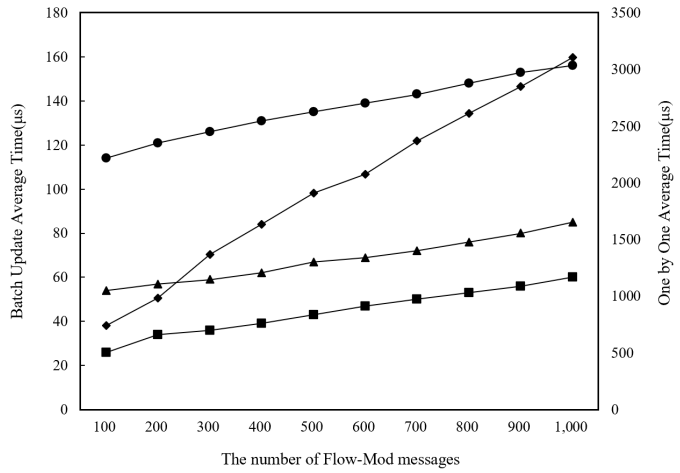
As we can see from Figures 4(a) and 4(b), when the quantity threshold is constant, increase time threshold, the larger number of update flow entries are, the longer it takes to update each flow entry, indicating that in a low-flow environment, when the quantity threshold is small, the updating performance is better by shortening the update time interval. Comparing Figures 4(a) and 4(b) horizontally, under the same time threshold, the larger quantity threshold is, the shorter the average batch update time. For example, we set the number of Flow-Mod messages as 1,000, when time threshold = 1 ms and quantity threshold changes from 10 to 100, the updating time is shortened from 1,300 μs to 60 μs, which means setting a large quantity threshold, the number of flow entries in each batch increases, and then reducing the frequency of rebuilding the LT, thus improving the updating performance significantly.

However, when the quantity threshold far exceeds the number of update flow entries, as shown in Figures 4(c), the quantity threshold = 1,000, in this case, the TC is the only condition that determines the update. When the number of Flow-Mod messages is less than 500, if the time threshold is set to be large, it needs to wait for a longer time to cache, making the time of updating longer with the time threshold increase. When the number of Flow-Mod messages is 100-500, an update interval of 1 ms is more appropriate, while for 500-700, time threshold = 5 ms is selected to optimize the updating performance. Therefore, it is very important to reasonably select the update conditions according to different environments and different number of flow entries. In general, no matter what update condition is set, it can be seen that batch update mechanism takes less time and has better update performance than one by one update mechanism, which proves the effectiveness of the proposed mechanism.

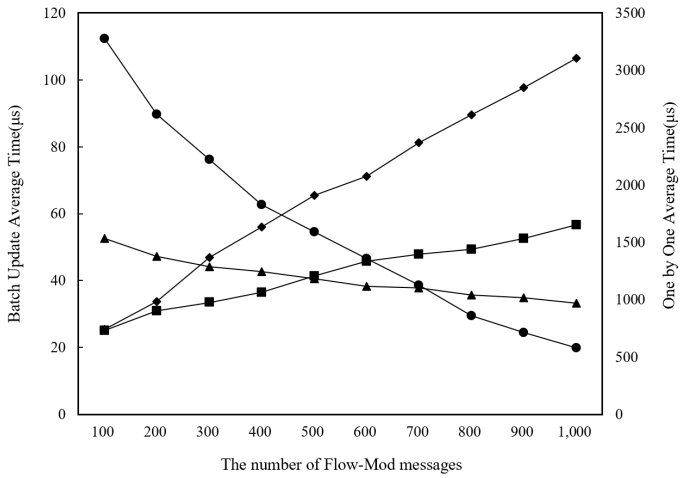
Figure 5 shows the performance comparison between the two update mechanisms in a high-flow environment (the number of update flow entries is more than 5,000). The number of Flow-Mod messages is tested from 5,000 to 50,000. The selection of quantity threshold and time threshold is consistent with the low-flow environment, but when we update 1,000 Flow-Mod messages using batch update mechanism, it can be seen from Figure 3 that the percentage of time to build ACL tree to the total time is 51.56%, which indicates that the time for preparing data and building the lookup tree is basically balanced, and the update performance is optimal at this time. Therefore, we choose minimum quantity



(a) Quantity threshold = 10

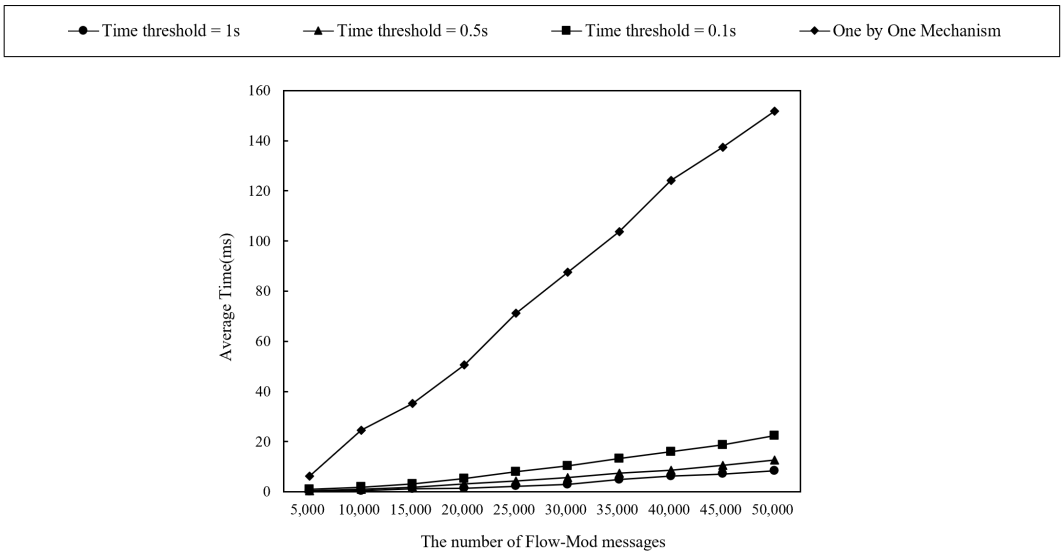


(b) Quantity threshold = 100

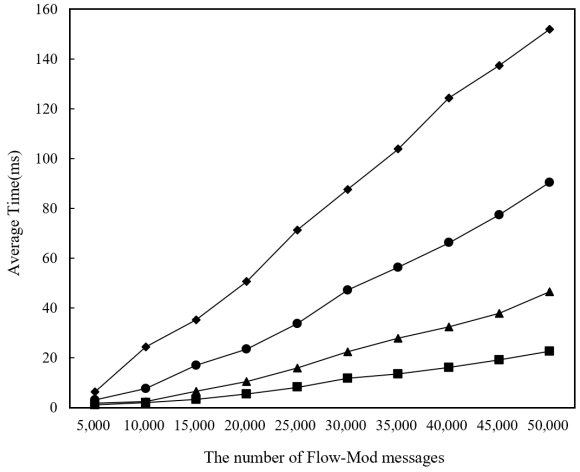


(c) Quantity threshold = 1,000

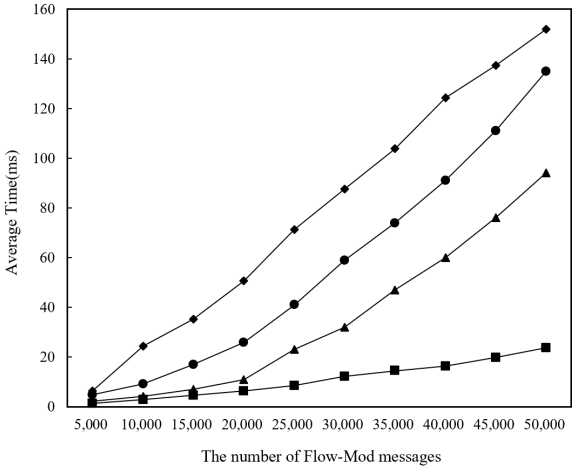
FIGURE 4. The average update time of two update mechanisms in a low-flow environment



(a) Quantity threshold = 1,000



(b) Quantity threshold = 5,000



(c) Quantity threshold = 50,000

FIGURE 5. The average update time of two update mechanisms in a high-flow environment

threshold as 1,000. Finally, quantity thresholds are selected as {1,000, 5,000, and 50,000}, time thresholds as {0.1 s, 0.5 s, and 1 s}.

According to the experimental results, it can be seen from Figure 5(a) that the update performance improvement is more obvious in a high-flow environment. The number of Flow-Mod messages increases linearly, the average update time of each entry is also showing a linear growth trend. When we update 50,000 Flow-Mod messages with one by one mechanism, the average time to update a Flow-Mod message is as high as 151.78 ms, which means one by one mechanism is difficult to meet the performance requirements under the high-flow environment.

When quantity threshold is larger than 1,000, as shown in Figures 5(b) and 5(c), by shortening time threshold, the average update time can be shortened. In this case, the larger the time threshold selection is, the closer to 1,000 Flow-Mod messages in each batch of update caches, and the shorter the average update time is. Considering the time to build lookup trees in a high-flow environment no longer occupies most of the update time, reducing the number of Flow-Mod messages in each batch, the time-consuming process of data preparation can be reduced and the update speed can be improved. When the total update number of Flow-Mod messages is 50,000, we set the quantity threshold = 1,000, time threshold = 0.1 s, it only takes 8.4 ms on average to update a Flow-Mod message. Compared with the one by one update mechanism, it is 18 times faster, showing batch update mechanism can be used to implement the flow entries fast updating, and is more applicable for the frequent update scenarios in the SDN.

The mechanism mentioned in this paper looks at the update performance in frequent update scenarios from a macro perspective. However, when considering the update time of a certain flow entry, the batch update mechanism generates additional data preparation time, and the update time of individual entry increases compared to the original one by one mechanism. This is the cost of the batch update mechanism, and we can balance the impact by designing reasonable batch update conditions based on the actual update environment.

4.3. **Evaluating the impact of multiple flow tables on updating performance.** Since the build of lookup tree is based on the flow table, the multiple flow tables need to be set up as the same number lookup trees. Therefore, the multi-level flow tables will have an important impact on updating performance.

By using batch update mechanism, the flow entries are evenly distributed in the multi-level flow tables to test the performance of updating. Since the switch to be tested can store up to 64 flow tables, 640 flow entries are selected in the low-flow environment and 64,000 flow entries are selected in the high-flow environment for testing.

For low-flow environment, set up the quantity threshold = 100, time threshold = 1 ms, Figure 6 shows that with the increase of flow table levels, the average update time (T_{AB}) is increasing. Considering the performance bottleneck of low-flow environment is to build the lookup tree, therefore, the more levels of the flow table, the worse performance.

For high-flow environment, set up the quantity threshold = 1,000, time threshold = 1 s, as can be seen from Figure 6, with the increase of flow table levels, the average update time (T_{AB}) is decreased. As the number of flow entries updated in each flow table is more than 1,000, it can be seen from Figure 3 that the performance bottleneck in this environment is the time for data preparation, which is the time for SST and FRT establishment, so it is better to reduce the number of flow entries in each flow table and increase the flow table levels for improving update performance.

In general, the update performance decreases with the increase of flow table level in low-flow environment. However, in high-flow environment, the update performance is

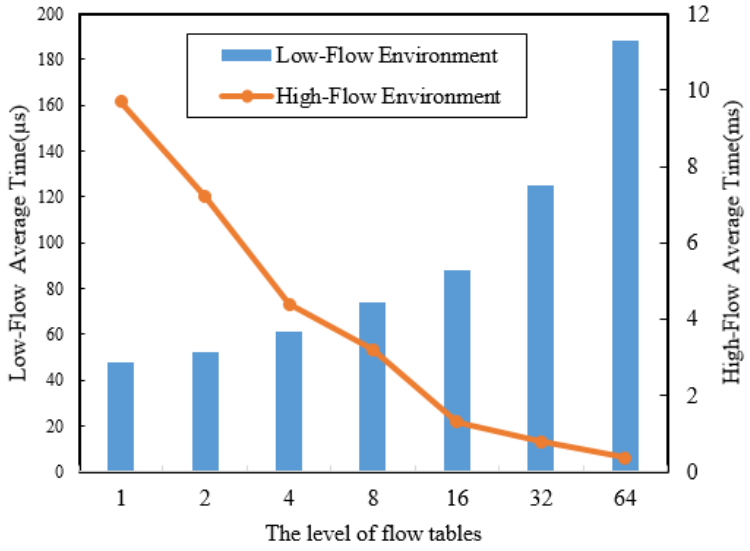


FIGURE 6. The average update time of multiple flow tables using batch update mechanism in different environments

improved when each flow table is in low-flow environment by increasing the flow table level.

5. **Conclusion.** For modern lookup engine design in SDN switch, besides the pursuit of lookup speed, efficient update mechanism is also an indispensable feature to maintain high forward performance. Through the analysis of the actual Flow-Mod messages update process, this paper puts forward the batch update mechanism, using the ACL algorithm to analyze the performance bottleneck in low-flow environment and high-flow environment respectively, proves the effectiveness of the proposed update mechanism. The batch update mechanism applies not only to exchanging the flow entries, but also to scenarios with the high update frequency, such as the multi-table update aggregation of the virtualized router.

In the future, we will consider flow entries as file data and make the switch as a storage node, embed common rules in the FRT of the switch in advance, to realize the update at an even higher speed. Up to now, the maximum number of flow tables tested is 64, and the maximum number of flow entries in each flow table is 64,000. Although the data selection is limited, it is sufficient to illustrate the usefulness of this mechanism. The test scope will be expanded to prove its universality in the frequent update scenarios in the future work.

Acknowledgment. This work is partially supported by Strategic Leadership Project of Chinese Academy of Sciences: SEANET Technology Standardization Research System Development (Project No. XDC02070100). The authors also gratefully acknowledge the helpful comments and suggestions of the reviewers, which have improved the presentation. Besides, the authors would also like to express thanks to Linan Jing, Dengyu Ran and Chi Zhang at the Chinese Academy of Sciences.

REFERENCES

[1] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang and A. Vahdat, Hedera: Dynamic flow scheduling for data center networks, *Proc. of the 7th USENIX Symposium on Networked Systems Design and Implementation*, San Jose, CA, USA, pp.89-92, 2010.

[2] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou and M. Zhu, B4: Experience with a globally-deployed software defined WAN, *Computer Communication Review*, vol.43, no.4, pp.3-14, 2013.

[3] S. Kandula, S. Sengupta, A. G. Greenberg, P. Patel and R. Chaiken, The nature of datacenter traffic: Measurements & analysis, *ACM SIGCOMM Conference on Internet Measurement Conference*, Chicago, IL, USA, pp.202-208, 2009.

[4] T. Benson, A. Akella and D. A. Maltz, Network traffic characteristics of data centers in the wild, *ACM SIGCOMM Conference on Internet Measurement Conference*, Melbourne, 2010.

[5] D. Y. Huang, K. Yocum and A. C. Snoeren, High-fidelity switch models for software-defined network emulation, *Proc. of the 2nd ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, Hong Kong, China, 2013.

[6] M. Kuźniar, P. Perešíni and D. Kostić, What you need to know about SDN flow tables, *International Conference on Passive and Active Network Measurement*, New York, vol.8995, pp.347-359, 2015.

[7] M. Kuźniar, P. Perešíni, D. Kostić and M. Canini, Methodology, measurement and analysis of flow table update characteristics in hardware OpenFlow switches, *Computer Networks*, vol.136, pp.22-36, 2018.

[8] X. Jin, J. Gossels, J. Rexford and D. Walker, CoVisor: A compositional hypervisor for software-defined networks, *The 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI15)*, Oakland, CA, USA, pp.87-101, 2015.

[9] K. Kannan and S. Banerjee, Compact TCAM: Flow entry compaction in TCAM for power aware SDN, *International Conference on Distributed Computing and Networking*, pp.439-444, 2013.

[10] A. Lazaris, D. Tahara, X. Huang, E. Li and M. Yu, Tango: Simplifying SDN control with automatic switch property inference, abstraction, and optimization, *The 10th ACM International Conference on Emerging Networking Experiments and Technologies (CoNEXT'14)*, Sydney, Australia, pp.199-212, 2014.

[11] X. Wen, C. Diao, X. Zhao, Y. Chen and K. Bu, Compiling minimum incremental update for modular SDN languages, *ACM*, Chicago, USA, pp.193-198, 2014.

[12] B. Zhao, J. Zhao, X. Wang and T. Wolf, RuleTailor: Optimizing flow table updates in OpenFlow switches with rule transformations, *IEEE Transactions on Network and Service Management*, vol.16, no.4, pp.1581-1594, 2019.

[13] R. M. Ramos, M. Martinello and C. E. Rothenberg, SlickFlow: Resilient source routing in data center networks unlocked by OpenFlow, *Local Computer Networks*, Sydney, Australia, pp.379-386, 2013.

[14] M. Yu, J. Rexford, J. M. Freedman and J. Wang, Scalable flow-based networking with difane, *ACM SIGCOMM Computer Communication Review*, vol.40, no.4, pp.351-362, 2010.

[15] L. Xie, Z. Zhao, Y. Zhou, G. Wang and H. Zhang, An adaptive scheme for data forwarding in software defined network, *International Conference on Wireless Communications & Signal Processing*, Hefei, China, pp.1-5, 2014.

[16] H. Chen and T. Benson, The case for making tight control plane latency guarantees in SDN switches, *ACM Symposium on SDN Research*, Santa Clara, CA, USA, pp.150-156, 2017.

[17] Z. Cao, X. Chen and H. Ni, A survey of flow caching for packet forwarding, *Journal of Network New Media*, vol.8, no.6, pp.11-17, 2019.

[18] F. Syed, Z. Ullah and M. K. Jaiswal, Fast content updating algorithm for an SRAM-based TCAM on FPGA, *IEEE Embedded Systems Letters*, vol.10, no.3, pp.73-76, 2018.

[19] E. Kim, Y. Choi, S. Lee and H. Kim, Enhanced flow table management scheme with an LRU-based caching algorithm for SDN, *IEEE Access*, vol.5, pp.25555-25564, 2017.

[20] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma and S. Banerjee, Devoflow: Scaling flow management for high-performance networks, *ACM SIGCOMM Computer Communication Review*, vol.41, no.4, pp.254-265, 2011.

[21] H. Ma, Y. Yang and Z. Mi, A distributed storage framework of FlowTable in software defined network, *Computers Electrical Engineering*, vol.43, no.4, pp.155-168, 2015.

[22] X. Jin, H. Liu, R. Gandhi, S. Kandula, R. Mahajan, M. Zhang, J. Rexford and R. Wattenhofer, Dynamic scheduling of network updates, *Computer Communication Review*, vol.44, no.4, pp.539-550, 2014.

[23] V. Stefano and C. Luca, Safe, efficient, and robust SDN updates by combining rule replacements and additions, *IEEE/ACM Transactions on Networking*, vol.25, no.5, pp.3102-3115, 2017.

[24] H. Shimonishi, H. Ochiai, N. Enomoto and A. Iwata, Building hierarchical switch network using OpenFlow, *International Conference on Intelligent Networking Collaborative Systems*, Barcelona, Spain, pp.391-394, 2009.

[25] N. Mckeown, T. Anderson, H. Balakrishnan, M. Guru and S. Jonatha, OpenFlow: Enabling innovation in campus networks, *ACM SIGCOMM Computer Communication Review*, vol.38, no.2, pp.69-74, 2008.

[26] K. Amphawan, SST: An efficient Suffix-Sharing Trie structure for dictionary lookup, *AMS Annual Meeting*, Austin, TX, USA, pp.179-184, 2013.

[27] Y. Ghadi, M. Sh. Daoud, F. Kharbat and T. Elamsy, Evaluation of the difference between verification and validation of software and analyzing the significance among both, *ICIC Express Letters, Part B: Applications*, vol.10, no.10, pp.885-893, 2019.

[28] D. E. Taylor and J. S. Turner, ClassBench: A packet classification benchmark, *Proc. of IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies*, vol.15, no.3, pp.499-511, 2005.

[29] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow and G. Parulkar, ONOS: Towards an open, distributed SDN OS, *ACM*, Chicago, USA, pp.1-6, 2014.

[30] Z. Tayachi, M. Escheikh and K. Barkaoui, Performance evaluation of virtual switch with batch arrival using quasi-birth-death process, *2019 International Conference on Industrial Engineering and Systems Management (IESM)*, Shanghai, China, pp.1-6, 2019.