Performing Software Defined Route-Based IP Spoofing Filtering with SEFA

Guang Yao¹,²,³, *Jun Bi¹,²,³, Tao Feng¹,²,³, Peiyao Xiao¹,²,³, Duanqi Zhou¹,²,³
¹Institute for Network Sciences and Cyberspace, Tsinghua University
²Department of Computer Science, Tsinghua University
³Tsinghua National Laboratory for Information Science and Technology (TNList)
Beijing, China
{yaoguang@cernet, junbi@tsinghua, fengt09@mails.tsinghua, xpy10@mails.tsinghua, zdq12@mails.tsinghua}.edu.cn

Abstract—IP spoofing is a well-known security threat on the Internet. Though there have been a number of spoofing prevention mechanisms, due the diversity of networks and management objectives, the operators may prefer a framework which enables easy installation and modification of the IP spoofing prevention solution, rather than a single mechanism.

In this article, a lightweight and efficient framework for route-based IP spoofing filtering, named SEFA, is proposed. Through providing a collective view of the network and decoupling the filtering rule generation from network devices, SEFA enables easily installation of spoofing filtering application. SEFA mainly resolves the challenge that how to build network abstraction without taking full controllability. SEFA has been implemented based on slightly modifying commercial routers and an open source controller. Based on experiments, SEFA is found to be able to reduce the overhead and the latency of filtering rule generation and installation, while keeping off the complexity and latency of generating forwarding rules by the controller.

Index Terms—IP Spoofing, Software Defined Networking, filtering

I. INTRODUCTION

IP spoofing[1], which means attackers make use of forged source IP addresses to launch attacks, has been long recognized as a serious security problem on the Internet. An attacker can modify the source address of attacking traffic to address assigned to others or even unassigned to hide its actual location, or circumvent access control rules on source, or utilize the flaw of the victim system, or gain the capability of launching reflection based attack. There are a number of notorious attacks rely on IP spoofing, including SYN flooding [2], Smurf [3]. DNS amplification [4] etc. In 2006, a DNS amplification attack is launched against a Top Level Domain (TLD) name server and severely degraded the capacity of the TLD name server for several days [5]. According to the observation of CAIDA, there are a large number of spoofing activities [6]. In the NANOG 50th meeting, a presentation from ARBOR shows spoofing is quite significant despite of large size of zombie networks [7].

There have been a number of proposed spoofing filtering mechanisms, which are generally classified into filtering-on-path [8-11] and end-to-end authentication [12, 13]. However, there is not a silver-bullet for all the networks. It is due to networks are different in size, complexity and potential risk may not be affordable by the operator.

If taking into account the evolution of networks, even for a single network, nor is there a solution always applicable. In practice, operators may want to choose the solution exactly suitable for their network and demand. However, it is hard to get support from vendors for the requirements are too diversified. These problems prompted us to design a framework for spoofing filtering instead of another solution. This framework should allow operators to easily install the chosen spoofing prevention solution in their network, and modify the solution if the requirement changes.

In this article, we propose a novel architecture to support spoofing filtering, named SEFA (Software dEfined Filtering Architecture). SEFA provides a lightweight and efficient framework for route-based IP spoofing filtering, which is a special but the most prevalent type in filtering-on-path category. SEFA is composed by three layers: OpenRouter in the data plane, SEFOS in the control plane, and SEF APP in the application plane. Spoofing filtering mechanisms, which run as applications, are loosely coupled with SEFA; thus they can be easily installed and modified. SEFA provides a collective view of the network to make the applications be able to generate more accurate rules. SEFA has been implemented based on slightly updating commodity routers and an open source

* Jun Bi is the corresponding author.
controller. Based on the experiments, SEFA is found to be able to increase the filtering capacity, reduce the overhead and latency of filtering rule generation and installation, while keeping off the complexity and latency of generating forwarding rules by the controller.

Particularly, we must note that SEFA is not another controller on OpenFlow. Actually, the original model of OpenFlow is not suitable for this scenario, as we illustrated in section IV. We slightly modified the model of OpenFlow in SEFA. We consider such a modification may be of more or less generality for other scenarios. We hope our experience may be valuable for researches in other areas of OpenFlow.

The contribution of our work is of three folds:

1. We propose the diversity of network and operator requirements asks for a solution allowing enable diversified spoofing prevention mechanisms;
2. We propose SEFA, which is a light-weight and efficient software-defined framework for route based spoofing filtering;
3. We implemented SEFA with moderate modifications on commercial routers and an open source controller. Besides, we evaluated SEFA based on experiments using the implemented prototypes.

The following of this paper is organized as follows. At first we introduce the related researches. Then we illustrate the constraints, choices, and challenges in SEFA design. In the following section, we give a detailed description of SEFA. Then we introduce the implementation of SEFA and the evaluation results. At last, we conclude our work with future works.

II. RELATED WORK

A. IP Spoofing Prevention Mechanisms

IP spoofing is recognized as a serious security threat for long. The most practical IP spoofing solutions are Ingress Filtering and IPsec [14]. Ingress Filtering filters packets if their incoming interface does not match the corresponding network of their source address. The filtering is usually performed by manually configured ACL or uRPF check. There have been numerous Best Current Practices (BCPs) (BCP38, BCP84) and IETF RFCs about Ingress Filtering. Ingress Filtering is also a well-supported function on commodity routers. Standardization and universally support make Ingress Filtering widely deployed. Ingress Filtering is generally enabled at edge devices. In most cases, it relies on the uRPF feature to build the correct filtering table. On core routers, because there is no explicit binding between the prefix and interface, only traffic with bogus prefixes is filtered.

IPsec provides the ability of authenticating the origin of the packet. The major problem of IPsec is that global distribution of the key is still a challenge. Besides, IPsec is vulnerable to DDoS attack because the attacking traffic has aggregated at the victim host, and the computation cost of authentication is generally not low enough.

There also have been a number of research works, which can be classified into 2 categories.

1: Filtering-on-path. Such mechanisms filter packets on the path, generally before the packet aggregated at victim network. DPF [8] is a direction based filtering mechanism, which filters packets based on the binding between prefix/flow and interface. However, the article does not propose a universal mechanism to build the filtering table. Passports [11] proposes checking cryptographic signature hop-by-hop at AS granularity. It can filter spoofing traffic before they aggregate, and the filtering can be performed at AS granularity, which is better than general filtering-on-path mechanisms. However, the cost is also heavy: it requires the cooperation of ISPs, update of routing device, the update of client and modification of the packet. SAVE [9] is a new protocol on routing devices, which announce the local route selection choice and help others build the mapping table from incoming prefix to the interface. iDPF [10] builds inter-domain filtering rules based on the valley-free feature of inter-domain routing and the BGP announcement filtering rules.

2: End-to-end authentication. Such mechanisms validate the source address at the other end of communication (at different granularity). [12] proposed a mechanism which inserts AS tag into data packet and check the binding between address prefix and AS tag at the destination AS. This mechanism requires the association of ASs and update of routing devices. HCF [13] checks the validation of source prefix based on the binding between prefix and hop count value which is inferred from remaining TTL value. HCF is light-weighted, and of no deployment requirement, but it has significant false negative because the forged network and the attacker can have non-trivial probability to have the same hops away from the destination, and it can be bypassed by smart attackers who modify the initial TTL value. Also, it can result in false positive when the hop count value is inconsistent, e.g., in case of route change.

B. OpenFlow based IP Spoofing Prevention Mechanisms

After the birth of OpenFlow, there have been a number of security mechanisms based on OpenFlow. Ethane [15] offloads all the tasks to the controller. The remaining works [16-18] are typically working on edge devices, and these mechanisms do not face (and solve) the complexity of routing. Though it is possible to remove routing protocols when OpenFlow is enabled on all the routers, there are still a number of challenges, e.g., scalability, reliability, etc.

III. BASIC MODEL OF ROUTE-BASED FILTERING MECHANISMS

Route-based spoofing filtering mechanisms check each source IP address with the expected incoming interface [19]. Current, mostly due to the limitation of packet check method supported by network devices, such mechanisms are the most prevalent class in the world composed by various IP spoofing prevention mechanisms, e.g., uRPF, ACL, DPF [8], IDPF [10], etc.

Though it may be arguable, most of the existing route-based filtering mechanisms represent the following model with three steps:

1. Gets a view of address assignment (typically just part of addresses and with different granularities, e.g., host,
network, autonomous system) and routing (the real route entries, or only empirical “laws” of routing, e.g., valley-free, symmetric).

2. Based on the view, binds address with the expected incoming interface.

3. Filters packet if the packet does not match the binding.

Considering the nature that addresses are assigned distributively and the forwarding paths are joint result of decisions from all the routers, a collective view of address assignment and routing is always valuable for spoofing prevention mechanisms. Even for uRPF, which generates bindings just based on the local view of each router, the collective view can be used to verify whether the bindings are valid or not. Note that to build a collective view is not a new idea: SAVE has proposed a mechanism to enable each router to have a collective view. However, from today's point of view, such a mechanism may introduce too much complexity to each router.

The general model and the general requirement for a collective view inspire the design of SEFA. We think the common functions of the filtering mechanisms can be provided by the SEFA, and then the filtering mechanisms can run just as applications on SEFA. Thus, SEFA is designed to process 2 main tasks: 1. it collects and builds the collective view of address assignments and routing of the network, and presents it to the applications; 2. it handles filtering rules insert/remove requests from the applications. A schema of SEFA is illustrated in Fig. 1.

Note that even based on the same view of network, mechanisms are not necessary to generate the identical rules. For example, the mechanisms will determine where the filtering should be performed (e.g., at the border router [20] or the access router), which type of filtering rule should be installed (exact match rules or wildcard rules), whether the filtering should be performed passively [21] or proactively), etc.

IV. DESIGN OF SEFA

A. The design constraints

It should be noted that SEFA provides no more functions than a SDN (Software Defined Network) controller. Actually, if there are no constraints on the design space of SEFA, any SDN controller can replace SEFA. However, it is not the case we met in reality. From the operators in our campus network, we collect the following constraints:

1. SEFA should keep the router mostly unchanged. Operators do not want to offload all the functions on routers to the SEFA. For example, the routing and QoS, which are very mature in routers, should not be offload to some unproven applications. Only the spoofing filtering should be offloaded to SEFA

2. SEFA should only provide the minimal function set. They do not want to manage a full-stack SDN controller. SEFA should only support the necessary functions for spoofing filtering, for ease of management.

3. SEFA should provide a general abstraction of the network. Through providing a general abstraction, the implementation of new spoofing filtering will be easier. Besides, the visualization and management will benefit from a general abstraction.

4. SEFA should be based a general and lightweight southbound interface. SEFA should try to use the interfaces which the vendors are willing to support. Besides, the interface should not cost too much resource because the routers have been of heavy load.

It can be found an OpenFlow based SDN controller can perfectly satisfy 3 and 4, but obviously violate 1 and 2. Thus, a new architecture should be designed.

B. The design choices and challenges

Based on the design constraints, we made the following design choices:

- Use OpenFlow. OpenFlow is a lightweight and efficient interface that vendors are willing to support.
- Use flow table as the device abstraction. Flow table is the abstraction of the data path of an OpenFlow switch. It has been proven to be a successful abstraction.
- Keep functions other than filtering in routers.

Note that there are two contradictions here. First, currently using OpenFlow means the controller should take full control of the data plane, while in our design only the filtering is offloaded to the controller. Second, currently the flow table on the controller is built based on the full controllability on the data plane, while in our design the controller only has a very limited controllability. The contradictions result in two challenges correspondingly:

- How to use OpenFlow without taking full control of the data plane?
- How to build flow table on controller without full controllability?
Essentially, the first challenge originates from the coupling between OpenFlow and full controllability, and the second challenge originates from the coupling between flow table building and the full controllability. We think the couplings are unnecessary at least in our scenario. Thus, we try to decouple OpenFlow, flow table building on the controller with full controllability: OpenFlow is only used to manage part of functions on the router, and the flow table on the controller is built by the controller and the data plane devices jointly.

C. Overview of SEFA

Fig. 2 illustrates the framework of SEFA. SEFA has three components: OpenRouter(data plane), SEFOS(control plane) and filtering applications (applications). The interface between OpenRouter and SEFOS is a slightly modified OpenFlow, named OpenFlow+, and the interface between SEFOS and filtering applications is SEF-NBI. The components will be introduced in the following sections.

D. SEFA Data Plane: OpenRouter

The data plane of SEFA is named OpenRouter, which works in a hybrid model. OpenRouter has its own separated data plane and control plane. The data plane of OpenRouter works in legacy way. The packets are processed by xFlow(sFlow or NetFlow), ACL and FIB in sequence just as normal routers.

The control plane of the OpenRouter is the key to resolve the first design challenge. It is composed by an OpenFlow agent and a RTM (Routing Table Management). The RTM inserts the forwarding entries generated by routing protocols into the FIB. In another word, the forwarding entries are still generated by routing protocols running distributedly, and the controller does not have to generate forwarding rules.

The OpenFlow agent has 3 critical tasks: 1. Reporting new flows to the controller, then the controller can detect spoofing flows; 2. Installing filtering rules generated by the controller into the ACL to enforce spoofing packet filtering; 3. Representing the forwarding states and interface states to the controller to help generate filtering rules.

As illustrated in Fig. 3, the OpenFlow agent translates sampled packets into PACKET_IN events. Then flows can be sensed by the controller. The FLOW_MOD events are translated into ACL modification actions. As applications in SEFA are only expected to generate Permits or Denies, only modifying the ACL is enough.

There is currently no switch-to-controller OpenFlow messages which contain flow entries generated by the routers. To represent the forwarding rules to the controllers, three OF-extension messages are designed: NETSTATE_CHANGES, NETSTATE_REQUEST, and NETSTATE_DATA. Since the new messages extended OF messages, we name the new protocol by OpenFlow+(OF+). NETSTATE_CHANGES is an asynchronous message used to push changes in the forwarding table/interface table/route table to the controller. The controller can also pull states from the tables with sending a NETSTATE_REQUEST, the router will send NETSTATE_DATA which contains the full tables. To make the OpenFlow agent aware of the updates on the tables, the RTM also sent updates to the OpenFlow agent. This mechanism resolves the second design challenge. The network state can be collected by the controller with low latency and low overhead through using these messages.

The entries in these messages are all formatted using the nested TLV. For example, the Route entry, which can be contained in NETSTATE_CHANGES and NETSTATE_DATA, is as illustrated in Fig. 4. Based on the cooperation of the OpenFlow agent and the RTM, the filtering can be performed efficiently without involving the complexity of forwarding table generation.
E. SEFA Control Plane: SEFOS

The SEFA control plane is named SEFOS. SEFOS generates the abstraction of the data plane and communicates with the applications and OpenRouters.

Note that currently the data plane of SEFA, i.e., OpenRouter, is not standard OpenFlow device. Thus, to form a data plane device abstraction, SEFOS translates the non-OpenFlow elements into OpenFlow elements. Each OpenRouter is abstracted as a pipeline of two flow tables. Flow table 0 contains the ACL rules, i.e., spoofing filtering rules. Flow table 1 contains the translated interface table, FIB and RIB in sequence. Each entry in the interface table is translated into a flow entry whose match is the local prefix and the action is outputting to the corresponding interface. FIB and RIB are translated into corresponding forwarding flow entries. Based on the flow tables, a NIB (Network Information Base) is built similarly as ONIX [22], with an additional ACL table.

The main task of SEFOS is maintaining the abstraction, i.e., syncing the abstraction with the data plane and the applications. In detail, the SEFOS receives filtering rule setup/remove requests from applications, updates the flow table, and installs/removes the rules on the OpenRouters; on the other hand, SEFOS collects routers, links, routes, interface information from the OpenRouters, update the flow tables and NIB, and notify the updates to the applications.

To install flow entries generated by applications on the routers, simply OpenFlow is enough. In SEFA, the applications and SEFOS can be decoupled. SEFOS provides a set of network API (named SEF NBI, as Fig. 6.) to allow access from remote hosts. For flow table update, the applications can simply use a POST interface to add or remove a number of flow entries.

However, applications may require syncing with the NIB (or part of the NIB). If applications repeatedly fetch data from NIB, large communication overhead will be introduced. Simply pushing changes to the applications is not cost-effective either. An application may only care about changes on some of the elements. For example, an uRPF on SEFA only cares the routes of edge routers. We use a Subscribe-Publish model to accomplish the syncing. The applications can make a REGISTER interface to subscribe the interested elements in the NIB. Whenever the value of some of the elements is changed, or the element is removed, the SEFOS uses the NOTIFY interface to announce the changes to the applications. Then the changes can be PULLed by the applications based on the ID of the changed elements from the NIB. A mapping table from network elements to listening applications is maintained by the SEFOS to find the applications which should be notified whenever a change happens. Whenever new elements are added into the NIB, the elements will be announced to all the applications.

Another task of the SEFOS is maintaining the PACKET_IN message queue. It should be noted that the flows reported in PACKET_IN messages are not unmatched flows as OpenFlow. They are actually flows sampled by the xFlow mechanism. In the data plane, they are forwarded based on the FIB. Thus, there is no need to send the events to the applications. However, there can be filtering applications which install filters passively, i.e., filtering is only performed whenever attacking flows are discovered. For such applications, they can register for the message queue, and the SEFOS will then push the events to the applications. Such applications can also pull the first event from the queue directly.

F. SEFA Applications

Applications can make use of interface provided by SEFOS to generate and install rules. In this section, we take IDPF and VASE on SEFA as examples, as illustrated in Fig. 7 and Fig. 8.

IDPF is a spoofing filtering mechanism based on inferring the incoming interface of packets. Based on the Valley-free property of Internet routing, whenever a packet arrives at the interface from which the route to the source of the packet has been learned, it should be regarded as valid. In another word, the IDPF generate filtering rules based on information in RIB. The IDPF application based on SEFA can simply register on the route table in the NIB, and install rules generated based on the route table.

VASE is an intra-domain spoofing filtering mechanism based on calculating the forwarding path of flows. It makes use of the forwarding table and interface table to calculate the path of a part of flows and generates filtering rules to protect such flows. Thus, the VASE application gets forwarding table entries and interface entries from the NIB and generates the filtering entries. VASE performs filtering passively. Thus, the VASE application registers for the message queue. Whenever spoofing is detected, the corresponding filtering rules will be installed.
There are spoofing filtering mechanisms not based on routing information but on other protocols or even learning data packets. For example, SAVI is based on address assignment protocols, including DHCP, NDP, etc. In such cases, the applications can install Redirect rules on OpenRouters to redirect the concerned messages to the controller. Then the applications can learn necessary information from the redirected messages.

Currently, SEFA can only support route-based filtering. This limitation is introduced by the data plane. If there are routers supporting the functions required by these mechanisms, SEFA can be modified slightly to adopt such mechanisms. HCF is a very direct example. HCF learns the valid TTL scope from data packets and filters spoofing packets based on TTL value. If the ACL can check the scope of the TTL value, HCF can be supported by SEFA. Take SPM as another example. If the routers have a specialized module to insert, remove and check tags in packets, based on the translation of the OpenFlow agent, the module can also be accessed by the controller in an OpenFlow-like style. Then SPM can be implemented as an application generating rules in the module.

V. IMPLEMENTATION AND EXPERIMENTS

A. Implementation

The data plane of SEFA, i.e., OpenRouter, is implemented through updating the software of a commodity router (DigitalChina DCRS5980). It is worth noting that it is much easier to update a switch to support SEFA than to support OpenFlow, because the OpenFlow agent only need to manage the ACL. The SEFOS is currently implemented based on extending NOX. The main changes include adding new OF messages, constructing the NIB, and providing northbound interfaces. We have implemented VASE based on SEFA.

The main modification on the router is adding the module to process OpenFlow/ OpenFlow+ messages. A trick implementation detail is that NETSTATE_DATA will be split in sending to allow the SEFOS process the content without waiting for the whole message transmitted.

In the header file of OpenFlow for NOX, the new messages types are added. The corresponding events and event handlers are registered in NOX main body. For NOX uses an event-driven model, whenever the new messages are received, the corresponding handler processes will be called. Flow tables and NIB are generated based on the messages. The elements in NIB are stored in database (MySql) as tables.

For the northbound API, the messages are also formatted with TLV. This is designed considering the scalability. New types of data can be easily translated through defining new TLVs. SEFOS plays as a TCP server for SEF applications. It accepts connections from applications, processes the data requests, and answers the requests.
B. Experiments

We built a network with 6 OpenRouters (DCRS 5980). The topology is illustrated in Fig. 9, and the photo of the network is in Fig. 10. We run VASE on SEFA (named VASE-SEFA) and VASE on traditional interfaces (named VASE-Tradition) in the network. The topology is a simplified mirror of our campus network. The switches are numbered the same as they are numbered in the campus network. Thus, they are not numbered from 1 to 6.

C. Evaluation Results

1) Comparisons with Traditional Interfaces

In this section, we describe the main advantages of SEFA against traditional interfaces.

a) Rule Installation Latency

Fig. 11 shows the rule installation latency of VASE-SEFA and VASE-Tradition. VASE-Tradition makes use of Expect script to install filtering rules. The average latency of VASE-Tradition is more than 100ms. VASE-SEFA uses OpenFlow to install rules. The average latency is around 30ms. It can be found SEFA can significantly reduce the rule installation latency.

Another discovery is the latency to install only one rule. VASE-Tradition requires more than 1s even when just installing 1 rule, because it must set up connection with the switch and interact with the switch for several steps to finish initialization. However, for VASE-SEFA, there is no need to set up a connection as the channel between the switch and the controller is always kept, and only one message is required to set up a rule.

b) Route Sync Latency

VASE generates rules based on the forwarding table. Thus, VASE should tightly sync with the state of switches to avoid generating wrong rules. In this experiment, we disconnect the link between sw9 and sw3, and measure the latency since then until the route table used by the application re-sync with the switches. The result is illustrated in Fig. 12.

Whenever the route converges (10s~30s), the OpenRoute will notify the SEFOS the new routes. In this way, the VASE-SEFA can get the new routes with little latency whenever they are generated. Thus, VASE-SEFA just uses around 20s to re-sync the route state after the disconnection.

For VASE-Tradition, because there is no mechanism to get notified the route changes, it polls the route state of the switches to get updates. Due to the limitation of CPU capacity and control plane bandwidth of the switches, the polling can be only performed every 10 minutes. Besides, VASE-Tradition requires to collect the whole FIB from the switches, which would take much more time than only collecting route changes. Thus, the route sync latency ranges from around 30s to 600s.

Based on the comparison, it can be found the SEFA can significantly increase the agility of filtering mechanisms through providing PUSH interfaces when network changes.

c) CPU Usage

For VASE, it is very costly to generate filters, as it must calculate the path of each flow. Especially for VASE-Tradition, as there is no mechanism to know the changed routes directly, whenever route changes, it must compare the new routes with old routes, and re-calculate the paths for the affected flows. However, for VASE-SEFA, as new routes are provided by the
SEFOS, it can directly generate filters based on the new routes. Fig. 13 shows the CPU usage of VASE-SEFA and VASE-Tradition. On average, the CPU usages of both mechanisms are low. However, whenever generating filters on route changes, VASE-Tradition requires around 100% CPU resource, whereas VASE-SEFA only requires 18%. This result shows SEFA can significantly reduce the overhead of filter generation through providing efficient interfaces whenever route changes.

2) Comparisons with OpenFlow

We also compared SEFA with OpenFlow. DCRS 5980 does not support pure OpenFlow mode. We use an OpenFlow switch from DELL with approximate capacity instead. As we emphasized above, SEFA does not introduce the cost and complexity of routing to the controller and applications, compared with OpenFlow.

We performed a very simple experiment: using both switches to forward packets. The average latency of DCRS 5980 is just around 25μs, though this value will increase with the size of the ACL. However, the latency of DELL is significantly large than this value. We choose FloodLight as the controller and use a simple forwarding application. The latency is around 120ms for the first packet, though the latency of forwarding the following packets is only around 25 μs.

We do not declare SEFA is always better than OpenFlow. Actually, we regard SEFA as an OpenFlow tailored for the filtering (especially spoofing filtering) scenario, where is no need to involving the controller on making forwarding decisions.

VI. CONCLUSION AND FUTURE WORKS

In this article, we proposed SEFA, which is a lightweight and efficient software-defined framework for IP spoofing filtering mechanisms. SEFA is a hybrid architecture: filtering is managed by OpenFlow, whereas forwarding is still handled by routing protocols. Based on the experiments, SEFA is found to be able to significantly reduce the overhead and latency of filtering rule generation and installation, while keeping off the complexity and latency of generating forwarding rules by the controller.

SEFA is a trial for hybrid OpenFlow: we try to open the filtering system while keeping the routing system only visible. In another word, we are seeking a balance between the traditional routers and the OpenFlow. Considering there are a number of mature systems in routers, we will try to explore whether it is possible to partly open some of the systems, while keeping the other systems closed, or if there is a framework for such hybrid modes of OpenFlow.

ACKNOWLEDGEMENTS

This work is supported by the National High-tech R&D Program ("863" Program) of China(No.2013AA010605), the National Science Foundation of China (No.61161140454), National Science & Technology Pillar Program of China (No.2012BAH01B01), the National Science Foundation of China (No. 61303194), and the Post-Doctoral Funding of China(2013M530047).

REFERENCES


