SDN-based Traffic Control: Implementation of Forwarding and Redirection Mechanisms

1st Jiahao.Qi
School of Advanced Technology
Xi'an Jiaotong-Liverpool University
Suzhou,China
2362068

2nd Shuobai.Chen
School of Advanced Technology
Xi'an Jiaotong-Liverpool University
Suzhou,China
2362870

3rd Zhenxi.Chen
School of Advanced Technology
Xi'an Jiaotong-Liverpool University
Suzhou,China
2363276

4th Antian.Sun
School of Advanced Technology
Xi'an Jiaotong-Liverpool University
Suzhou,China
2362202

***Abstract*—This research is based on the SDN architecture and realizes the forwarding and redirection control of network traffic by using Mininet and Ryu. The system enhances processing efficiency by optimizing the flow table installation mechanism. Experiments have verified its stability and scalability, providing a feasible solution for teaching and actual network scheduling.**

I. Introduction

A. Task Specification

SDN separates the network control plane from the data plane, enabling the network to have greater flexibility and programmability. This project is based on the SDN architecture, using the Mininet platform and Ryu controller to build a simple network topology that includes a client, two servers, and an SDN switch. The main objective of the project is to implement two different traffic control strategies: traffic forwarding and traffic redirection, and on this basis, analyze network performance.

B. Challenge

- Configure the flow table of the SDN switch correctly so that it triggers the controller decision when the first TCP SYN packet is received
- Implement traffic redirection in the controller and modify the IP and MAC addresses simultaneously to maintain communication consistency
- Ensure that the flow table entries can be cleared in a timely manner after the timeout to avoid affecting subsequent experiments

C. Practice Relevance

SDN technology also has high practical value in real network environments. Firstly, it can intelligently achieve load balancing because the controller can monitor in real time the broadband utilization rate and CPU/ memory load of each link in the network. This feature is very suitable for scenarios where multiple users operate centrally and have high requirements for traffic and service stability, such as school examination platforms, avoiding the crash of a single server due to overload. Subsequent requests from students can also be intelligently directed to more idle servers.

Moreover, SDN technology can achieve secure traffic control. Compared with the passive defense of traditional firewalls against attacks, it can realize strategic traffic diversion. This feature is highly suitable for government departments and others with high information confidentiality requirements, and can effectively protect business servers.

D. Contribution

- An SDN experimental topology with precise IP and MAC address configuration was constructed based on Mininet, providing a reliable simulation environment for traffic control research
- A forwarding controller with MAC self-learning function and a redirection controller capable of deeply parsing and modifying IP/TCP packets have been developed, achieving a complete technical solution from basic forwarding to transparent traffic control
- Key optimizations have been made in the flow table installation mechanism - when the data packet has been cached by the switch, the buffer_id is directly associated with the flow table entry, avoiding the repeated transmission of data packet content and effectively improving the efficiency of flow table distribution and network performance.

II. Related Work

The rise of SDN stems from the urgent need for network programmability and flexibility. As one of the core implementations of SDN, the OpenFlow protocol innovatively separates the control plane from the data plane of network devices [1]. This study demonstrates the feasibility of programming flow tables for underlying network switches through a centralized controller using standard protocols like OpenFlow. This infrastructure is fundamental to the realization of our project.

In the field of SDN traffic engineering, Google's B4 network has demonstrated the breakthrough application of SDN in industrial-level scenarios [2]. The research team achieved extremely high traffic scheduling accuracy and resource utilization by deploying a global SDN wide area network. The core lies in using a central controller to dynamically optimize and load balance the traffic between data centers. This concept is concretely reflected in our project: although the scales vary, we still rely on a centralized decision-making mechanism to dynamically guide the traffic destined for specific targets to the standby nodes, verifying the fundamental capabilities of SDN in intelligent traffic scheduling.

At the forefront of network security, the Avant-Guard framework proposed by Seungwon Shin et al. [3] has elevated the traffic control capability of SDN to a new height. This research has significantly enhanced the real-time response and mitigation capabilities of SDN to complex network attacks through an innovative "connection migration" mechanism and "triggered instructions". The core idea is to strategically manipulate the flow path to actively direct potential malicious traffic to the security detection nodes. The transparent traffic redirection function implemented in our project in ryu_redirect.py can be regarded as the initial practice of such advanced security applications. By deeply analyzing data packets and dynamically modifying network addresses, we have confirmed that SDN can achieve precise traffic diversion, providing a technical foundation for building an intelligent defense system.

III. Design

A. Network System Design

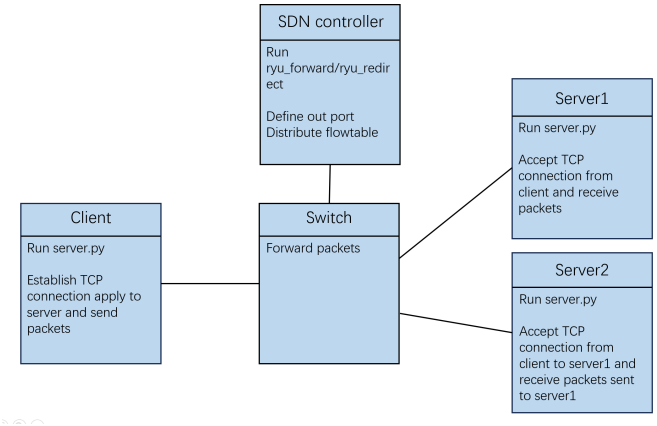

Fig. 1. Enter Caption

The above diagram illustrates the design of the network system. Running networkTopo.py on Linux virtual machine will set 5 nodes, which are the SDN controller, the switch and three hosts that act as two servers and a client separately. The servers and the client is connected through the switch and the SDN controller provide service to the whole topology which is defining the output pout and installing the flowtable to the switch for each new packet_in event. By defining the output port, the SDN controller may distribute a flowtable to the switch that let it distribute the packet to the exact host, or change the port before distributing the flowtable to redirect the packet according to the application currently running on it(ryu_forward/ryu_redirect).

B. Workflow

The following sequence diagram represents the interactions between the SDN controller, the switch and the three hosts. As a starter, the client establishes TCP connection with the server through the switch. For the first time that the packet arrives at the switch, the switch will install the flowtable from the SDN controller. The controller sets the flowtable so that the switch can flood the packet if the receiving host has not been reached, or set the output port according to the application currently running on the controller. Then the controller sends the flowtable to the switch and the switch flood or forward the packet. The flowtable will also be inferred in the later packet transfer.

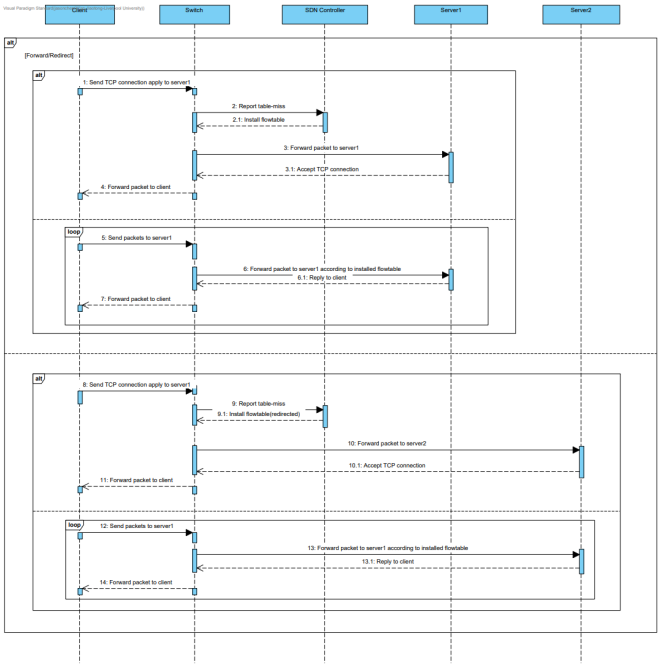

Fig. 2. Workflow

C. Algorithm

The following algorism represents the redirection process in detail. The algorithm is triggered whenever the controller receives a PACKET_IN message from the switch. Its input includes the packet, the switch identifier, the input port, and the parsed Ethernet, IPv4, and TCP headers. The output is a set of corresponding flow entries installed on the switch and a PACKET_OUT message to forward the current packet.

Algorithm 1: Kernel Pseudo Code of Network Traffic Redirection — SDN Controller

```
ipv4_pkt ← pkt.get_protocol(ipv4.ipv4)
tcp_pkt ← pkt.get_protocol(tcp.tcp)
if out_port ≠ OFPP_FLOOD and ipv4_pkt ≠ ∅ and tcp_pkt ≠ ∅ then
    ipv4_src ← ipv4_pkt.src
    ipv4_dst ← ipv4_pkt.dst
    if src = Client['mac'] and dst = Server1['mac'] then
        if Server2['mac'] ∈ mac_to_port[dpid] then
            out_port ← mac_to_port[dpid][Server2['mac']]
        else
            out_port ← OFPP_FLOOD
        end
        match ← {eth_type : IP, ipv4_src : ipv4_src, ipv4_dst : ipv4_dst}
        actions ← {set_field(eth_dst = Server2['mac']),
                   set_field(ipv4_dst = Server2['ip']), output(out_port)}
    else if src = Server2['mac'] and dst = Client['mac'] then
        if Client['mac'] ∈ mac_to_port[dpid] then
            out_port ← mac_to_port[dpid][Client['mac']]
        else
            out_port ← OFPP_FLOOD
        end
        match ← {eth_type : IP, ipv4_src : ipv4_src, ipv4_dst : ipv4_dst}
        actions ← {set_field(eth_src = Server1['mac']),
                   set_field(ipv4_src = Server1['ip']), output(out_port)}
    else
        match ← {in_port : in_port, eth_src : src, eth_dst : dst}
        actions ← {output(out_port)}
    end
else
    match ← {in_port : in_port, eth_src : src, eth_dst : dst}
    actions ← {output(out_port)}
end
install_flow(match, actions, priority, idle_timeout)
send_packet_out(msg, actions)
```

IV. Implementation

A. Development Environment

TABLE I
Development Environment

Field	Description
Operating System	Microsoft Windows 11
CPU	Intel(R) Core(TM) i7-14650HX
RAM	32G
IDE	JetBrains PyCharm Professional
Python Version	Python 3.14
SDN controller software	Ryu
Python Library	'Mininet', 'OVSKernelSwitch', 'RemoteController', 'app_manager', 'ofp_event', 'CONFIG_DISPATCHER', 'MAIN_DISPATCHER', 'ofproto_v1_3', 'ether_types', ' IPv4', 'TCP'

B. Steps of Implementation

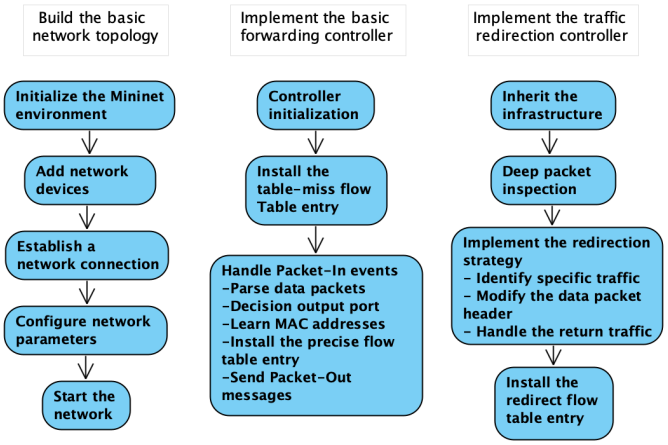

Fig. 3. Steps of Implementation

C. Programming Skills

- **Object-oriented programming**: This project adopted object-oriented programming and achieved the standardized integration of controllers by constructing two classes, Forward and Redirect, which inherit the RyuApp base class of the Ryu framework. The independent state is encapsulated within a class through the __init__ method, and the event handling logic is encapsulated as a method of the class. Specific OpenFlow events are bound using decorators. All these reflect the inheritance, encapsulation, and polymorphism characteristics of OOP.
- **Program modular design**: The system architecture follows the modularization principle and is functionally divided into three independent modules: topology construction, forwarding control, and redirection control. Internally, logical reuse is achieved through abstract general methods (such as the add_flow flow table installation interface), decoupling the parsing, learning, decision-making, and execution steps in the process. This ensures

that each module has a single responsibility and clear interfaces, enhancing the maintainability and scalability of the code.

D. Actual Implementation

To begin with, the controller checks whether the packet is both IPv4 and TCP, if the result is valid, the controller keeps on checking, or the controller will not redirect the packet. Then, the SDN controller checks the MAC address of the source and the destination of the packet, if both of them matches the record in the controller, it looks for the port for the arranged redirected destination. If the record of the port if found, it will be set as the output port and the match condition and the action will be updated. If the port is not arranged, that is, the address of the arranged destination is not recorded in the controller, the action will be set as flood the packet. At last, the flowtable is installed in the switch and instruct the matching and the action of certain packets,

E. Difficulties and Solutions

During the project implementation process, we overcame the following three technical difficulties:

- **Understanding and Application of Ryu Event-driven Architecture:**
  Difficulty: The Ryu framework is based on an event-driven model, and its core mechanism is to implicitly register event handlers through the decorator @set_ev_cls. The principle of decorators is difficult to understand - how the controller automatically captures OpenFlow events and routes them to the corresponding methods. Especially the concept of the MAIN_DISPATCHER state machine.
  Solution: We visualize this process by drawing a sequence diagram of event flow. Specifically, when the switch is connected to the controller, it first enters the CONFIG_DISPATCHER state to install the initial flow table; After the handshake is completed, it switches to the MAIN_DISPATCHER state to start processing data packets. Through this visual analysis, we accurately bound switch_features_handler to the configuration stage and packet_in_handler to the runtime stage, ensuring the correct timing of event handling.
- **Optimal management of flow table buffers:**
  Difficulty: The buffer mechanism of OpenFlow involves complex lifecycle management. The difficulty lies in determining when the data packet has been cached by the switch (buffer_id is valid) and how to utilize this mechanism to avoid duplicate data transmission. It is very important to understand the meaning of OFP_NO_BUFFER and the timing of buffer release here.
  Solution: We have implemented an intelligent buffer detection and usage strategy. When msg.buffer_id is valid, pass this parameter into add_flow. After the switch installs the flow table, it will automatically apply the new rule to the cached data packets. When the buffer is invalid, roll back to the traditional mode and resend the data Packet through the packet-out message. This adaptive mechanism not only enhances the installation efficiency of flow tables but also ensures reliability in various situations.

V. Testing and Results

A series of tests are used to evaluate the performance of the program. The subsequent sections delineate the testing environment, the testing steps, and the testing results.

A. Testing Environment

The Linux virtualization environment, facilitated by Oracle VM VirtualBox, functions as a platform for software development, testing, and deployment. Detailed specifications are provided in the following table:

TABLE II
TESTING ENVIRONMENT

Category	Specification
Laptop	Legion Y9000X IRH8
CPU	intel core i9
Opreating System	Ubuntu (64-bit)
Python version	Python 3.8.10
Mininet version	2.2.2
Ryu Controller Version	ryu 4.34
Packet Capture Tool	Wireshark

B. Testing Steps

1) Forwarding Case:

a) Mininet network topology set up: The network topology was generated using our networkTopo.py script, which consists of a client host, two server hosts (Server1 and Server2), and an OVS switch. A Ryu SDN controller was executed externally on the host machine. By entering the command 'sudo python2 networkTopo.py' in the terminal, the mininet network topology is created and five Xterm control panes are generated, shown in Figure2

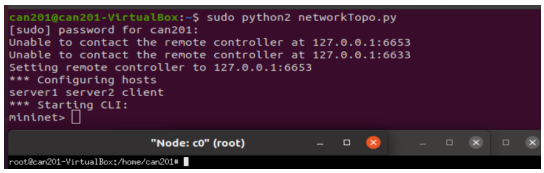

Fig. 4. setting up mininet network topology

b) Check Host's IP & Mac address: Following the deployment of the network topology, each host's IP and MAC addresses are validated through the execution of the 'ifconfig' command in the terminal, as illustrated in Figures 3, 4, and 5.

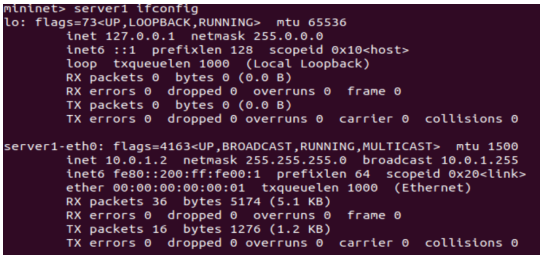

Fig. 5. server1

c) Connectivity test: Before conducting TCP tests, the client was required to ping both Server1 and Server2 to ensure basic connectivity, as required by the specification.

Executing the command 'sudo ryumanager ryu_forward.py' within the terminal initiates the deployment of the Ryu SDN controller application.

And executing the commands 'ping 10.0.1.2' and 'ping 10.0.1.3' within the terminal interface, as illustrated in Figure 6.

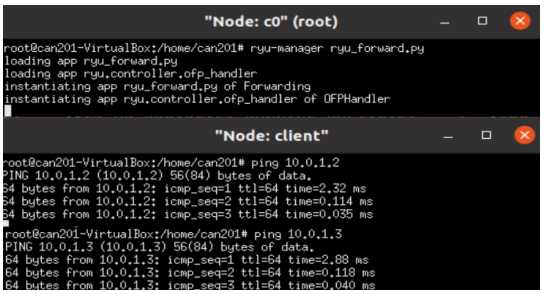

Fig. 6. ryu forward and ping

Both ping tests are successful, confirming that the Ryu controller (in ryu_forward.py) gets the correct IPv4 forwarding rules.

d) TCP Communication Tests: Following the completion of ping connectivity verification, the server application will be deployed on Hosts 1 and 2 to handle TCP segment reception via execution of the command 'python3 server.py'. Subsequently, the client application will be executed on the client host after a delay of five seconds by executing 'python3 client.py' in the terminal. It can be observed that only Server1 successfully received the TCP packet, confirming correct forwarding behavior. The configured flow table entry is validated through execution of the command 'sudo ovs-ofctl dump-flows s1 -O OpenFlow13' demonstrated in figure 7.

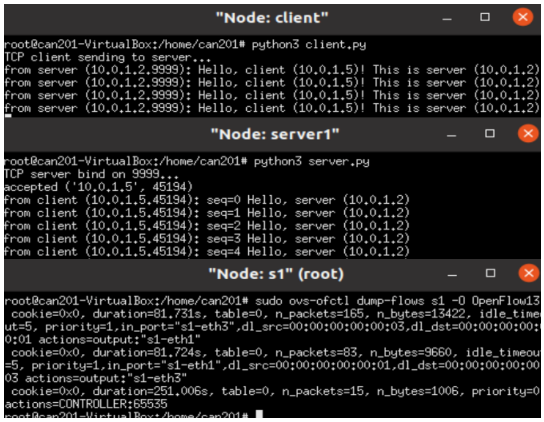

Fig. 7. forwarding testing

2) Redirection Case: Exactly the same as the forward case, the mininet topology was established, and then the host's IP and MAC addresses are checked.

a) Connectivity test: A new controller (ryu_redirect.py) was launched, followed by the same connectivity tests, shown in figure 8.

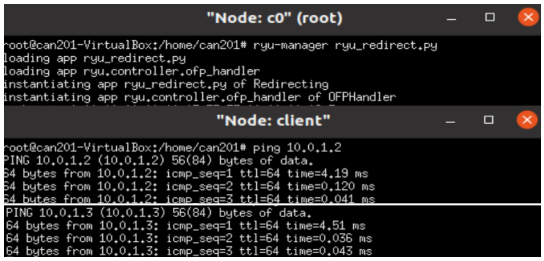

Fig. 8. ryu redirect and ping

b) TCP Redirection Test: Similar to the forward case, figure 9 demonstrate that the TCP segment transmitted by the client is rerouted to the server2.

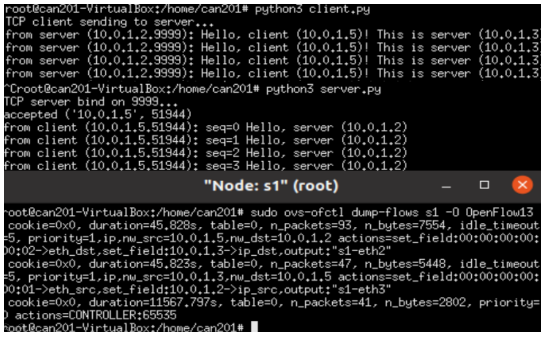

Fig. 9. redirect testing

C. Testing Results

This project selected Wireshark as the primary analytical instrument to caputer TCP SYN and ACK segments for network traffic inspection, and latency assessment within computer network analysis. Utilizing Wireshark on client-eth0, the

timestamp data for the subsequent packets—SYN, SYN+ACK, and ACK are documented, As illustrated in the subsequent images.

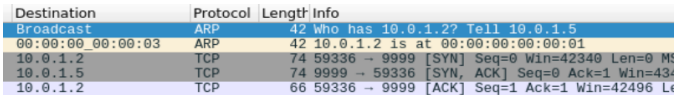

Fig. 10. wireshark

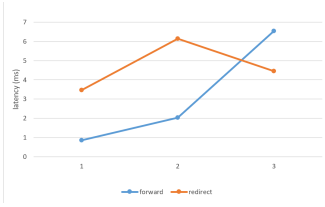

Fig. 11. Result of testing

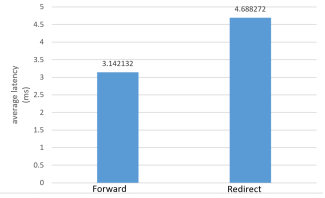

Fig. 12. Result of average testing

Based on the data demonstrated above,the forwarding case exhibits lower latency, given that after the initial flow table entry is established, no subsequent rule alterations are required. Conversely, the redirection process involves supplementary processing steps, including Packet-In events directed to the controller, IP and MAC address rewriting, and the deployment of new flow entries. Consequently, redirection latency surpasses forwarding latency.

VI. Conclusion

The network project has constructed the topology of the system and achieved the applications of forward and redirect. A notable feature of this system is that, when a packet reaches the switch, the switch does not send the whole packet to the SDN controller, instead, the switch saves the packet in its buffer and sends only the header and the buffer id to the controller. Such implementation remarkably reduces the burden of the traffic and significantly saves the time spent on requiring the flowtable. Also, the switch can fillter the LLDP and STP datagrams, which are confusion for date transferring among hosts. This results in the reduction of space spent on storing the match-action pairs and more efficient flowtable look up.

Though the current network system is already quite complete, and is sufficient for daily scenes, there is still some problems that obstruct the further development network system. For instance, if a certain link or host shuts down, The flow table on the switch still directs the packet to the port where the failure occurs. Additionally, without changing the configuration of the client or the IP of the server, directing a certain traffic from one server to another is not likely to be achieved. However, in this system, since the SDN controller has the global topology and real-time port status, it can re-calculates the new path in milliseconds, therefor, timely recover from the failure. Also, the system achieves packet redirection by decorating the header of the datagram.

This system is based on Mininet and Ryu controller, hencce, it is suitable for the teaching of SDN as experimenting platform. In developing scenes, it helps application upgrade and canary release. In practical, firstly, the development team can deploy a new version of the service on Server2, then, redirect a part of client requests to Server2 and observe the results, if everything works well, gradually increase the proportion.

To put this system into production, there are still some aspects of the system that could be improved. To start with, the portion of packet redirection and folwtable installing is currently hard coded, but it could be dynamic and have multiple prioritized match-action pairs in real implementation. Moreover, the system mainly focuses on forwarding and redirecting but lack of access control or abnormal traffic detection. Such absence leads to weak performance against port scanning or excessive connection attempts. Last but not least, to improve flexibility and manageability, a northbound interface on top of the controller could be adopted. Through REST-based API. the redirection part can be abstracted, therefor, external applications could alter the traffic redirection rules via HTTP requests.

Acknowledgment

TABLE III
INDIVIDUAL CONTRIBUTION PERCENTAGE

Name	Percentagen
Jiahao.Qi	25%
Shuobai.Chen	25%
Zhenxi.Chen	25%
Antian.Sun	25%

References

[1] N. McKeown *et al.*,"OpenFlow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*,Vol. 38, No. 2O, pp. 69-74, Apr. 2008.doi:10.1145/1355734.1355746

[2] S. Jain *et al.*,"B4: experience with a globally-deployed software defined wan," presented at the SIGCOMM '13: Proceedings of the ACM SIGCOMM, New York, NY, USA, 2013. doi:10.1145/2486001.2486019

[3] S. Shin, V. Yegneswaran, P. Porras and G. Gu,"AVANT-GUARD: scalable and vigilant switch flow management in software-defined networks," *Association for Computing Machinery*, pp.413-424, 2013. doi:10.1145/2508859.2516684