An Adaptive Broadcast and Multicast Traffic Cutting Framework to Improve Ethernet Efficiency by SDN

You-Chiun Wang and Han Hu

Abstract—In local area networks (LANs), Ethernet is a widely used layer-2 networking technology due to the low cost and self-configuring ability. It allows computers and switches to form a broadcast domain to exchange data, which means that many protocols built on Ethernet rely on sending data to every node. However, as the network grows, the efficiency of Ethernet degrades since the network is flooded with spam packets caused by broadcast. Even worse, traditional layer-2 switches do not well support multicast protocols but realize them by also broadcasting packets. To conquer these problems, the paper develops an *Adaptive Broadcast and multicast traffic Cutting (ABC)* framework based on software-defined networking (SDN). By taking two Ethernet protocols, ARP and IGMP, as examples, we show how to exploit SDN to restrain unnecessary traffic to improve Ethernet efficiency via our framework. With Mininet simulations, we verify that the ABC framework not only greatly reduces spam packets than legacy Ethernet but also saves controller overhead comparing with other SDN-based solutions. Moreover, we also implement the ABC framework on the campus network to demonstrate its practicability.

Index Terms—broadcast, Ethernet, multicast, OpenFlow, SDN.

1 Introduction

Ethernet achieves a complete triumph in the competition of layer-2 networking technologies, and it has been widely used in many wired LAN systems such as campus networks, enterprise networks, and data centers. The great success of Ethernet comes from its low cost and convenience. In particular, computers can effortlessly join an Ethernet network almost without manual configuration. Today, the Ethernet interface card is an essential component of every computer for communication.

In Ethernet, a set of computers are connected together by a switch and form one broadcast domain. Many protocols built on Ethernet thus rely on broadcast traffic for service or resource discovery [1]. The *address resolution protocol (ARP)* is one representative, where a computer uses broadcast to find out the MAC address corresponding to the IP address of another computer in the same domain. On the other hand, since a switch deals with only layer-2 tasks, it cannot well support *Internet group management protocol (IGMP)*, which is a layer-3 multicast protocol. Without IGMP snooping, multicast packets will be also sent to all computers in the broadcast domain.

The broadcast mechanism of Ethernet functions smoothly in a small LAN. However, when the number of computers substantially grows, switches have to be organized hierarchically to form a single, huge broadcast domain. In this case, the whole network will be congested with numerous broadcast packets. We take the campus network in our department as an example, which has nine class-C subnets ranged from 140.117.168.0 to 140.117.176.255. Table 1 gives the statistics of data received by each computer during 24 hours. In particular, both broadcast and multicast traffic occupies more than 95% and 92% of the number of packets and bytes received by the computer, respectively. Nevertheless, most data of such traffic are irrelevant to the capturing computer. The phenomenon shows that Ethernet becomes inefficient in a large-scale LAN, as each computer actually requires a very small portion of its receiving data.

TABLE 1: Statistics of data received by one computer during 24 hours.

type	traffic	packets	ratio	bytes	ratio
broadcast	ARP	5,076,350	66.77%	306,079,820	40.26%
	control	843,772	11.10%	113,727,980	14.96%
	data	172,724	2.27%	10,303,440	1.36%
	sum	6,115,867	80.44%	431,816,038	56.80%
multicast	SSDP	592,380	7.79%	203,116,302	26.72%
	LLMNR	408,650	5.38%	34,502,867	4.54%
	data	80,813	1.06%	31,326,354	4.12%
	sum	1,141,121	15.00%	272,715,458	35.87%
unicast	data	345,570	4.56%	55,643,023	7.33%

Simple service discovery protocol [2].
Link-local multicast name resolution [3].

Conventional solutions to the broadcast problem are to partition the network into multiple broadcast domains either physically by layer-3 routers or logically by virtual LANs. In this way, broadcast and multicast packets will be confined to each small broadcast domain. However, these solutions have some drawbacks. First, they usually incur a high cost, since routers are expensive than switches while it involves plenty of manual configuration in virtual LANs. Second, some protocols like NetBIOS are not routable at layer 3. Thus, such protocols may not well operate. Third, without mobile IP, mobility of computers and migration of virtual machines (e.g., in data centers) between different broadcast domains is complicated. We will further discuss this issue in Section 3.1.

Therefore, this paper seeks to conquer the broadcast problem in a large-scale LAN without partitioning it into multiple broadcast domains. The idea is to exploit the emerging *software-defined networking (SDN)* technique [4]. In particular, SDN logically separates the LAN into control and data planes. A centralized SDN controller deals with the control plane and

The authors are with the Department of Computer Science and Engineering, National Sun Yat-sen University, Kaohsiung, 80424, Taiwan.
E-mail: ycwang@cse.nsysu.edu.tw; m033040090@student.nsysu.edu.tw

makes decisions such as how to interpret packet headers and where to forward them. On the other hand, the data plane is distributed among switches to take charge of packet transmission. Through this architecture, network administrators can manage traffic flows in the LAN by setting rules in the controller.

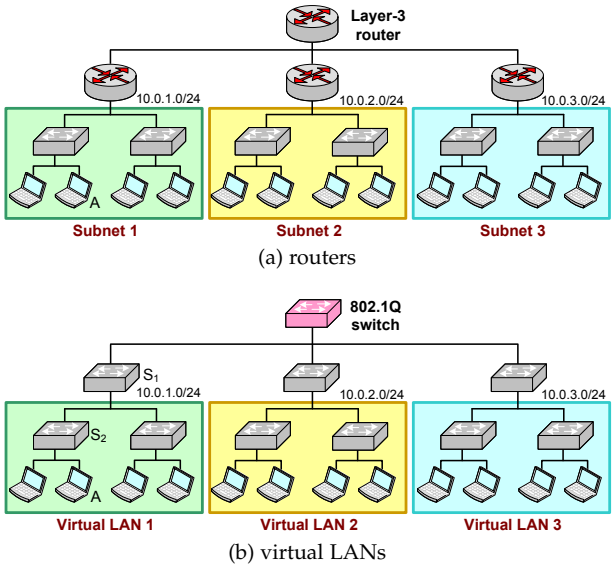
SDN gives a flexible manner to adjust the LAN's transmission behavior on the fly. Based on SDN, we propose an *Adaptive Broadcast and multicast traffic Cutting (ABC)* framework to eliminate spam packets. Specifically, the controller analyzes each incoming packet and learns the status of the ongoing protocol. Then, it spontaneously sets transmission rules to prevent switches from forwarding unnecessary broadcast or multicast packets that will be generated by that protocol. We use both ARP and IGMP to demonstrate how the ABC framework operates, which are two fundamental protocols in Ethernet. Our contributions are threefold. First, unlike other SDN-based approaches, the ABC framework helps the controller intelligently convert ARP addresses instead of making the controller act as a proxy to process every ARP packet. Therefore, it can greatly alleviate the burden of the controller. Second, the design of our framework considers the issues of backward compatibility and multicast, which are rarely discussed in the literature. Simulation results show that the ABC framework can significantly reduce ARP broadcast comparing with existing work in a LAN where traditional and OpenFlow switches coexist. Finally, the ABC framework is implemented on our campus network, and the experimental results verify its outstanding performance than legacy Ethernet.

We organize the rest of this paper as follows. Section 2 gives an overview of SDN. Section 3 surveys related work. In Section 4, we discuss our ABC framework. Section 5 evaluates system performance. Finally, we draw a conclusion in Section 6.

2 SDN Overview

OpenFlow [5] is a popular implementation for SDN. It replaces traditional switches by OpenFlow switches that can receive and execute commands from the controller. Specifically, OpenFlow defines the communication interface between an OpenFlow switch and the controller, and also the operations of OpenFlow switches. Each OpenFlow switch maintains a *flow table* to deal with every incoming packet, which contains flow entries that specify matching rules and actions. Once receiving a packet, the OpenFlow switch finds a flow entry whose matching rule is satisfied and then performs the entry's action. However, if no entry can be found, the OpenFlow switch triggers an event of *table miss*, which sends a *Packet_In* message with that packet's information to the controller. Then, the controller will return a new flow entry to tell the OpenFlow switch how to handle the packet by sending a *Packet_Out* message. Thus, the controller can easily manage OpenFlow switches and regulate the transmission of packets in the LAN.

On the other hand, Ryu [6] is a popular open-source framework to carry out the controller. It can well support OpenFlow and provides a set of application program interfaces (APIs) in Python to help users develop their own SDN applications. In particular, a user can implement his/her application by registering related input events along with the handling function. Then, Ryu employs an event queue to dispatch these events to the corresponding functions in a first-in-first-out (FIFO) manner. Moreover, Ryu allows users to acquire packet headers and compose packets through a packet-handling mechanism. Therefore, we will use Ryu to realize our ABC framework in the controller.



(a) routers

(b) virtual LANs

Fig. 1: Solutions to the broadcast problem in legacy Ethernet.

3 Related Work

3.1 Solutions in Legacy Ethernet

Traditional solutions to the broadcast problem in a large-scale LAN are to intuitively divide it into many small broadcast domains. One approach is to replace some switches with more sophisticated routers, where each router is responsible for one subnet (i.e., broadcast domain), as shown in Fig. 1(a). However, this approach puts restrictions on the allocation of IP addresses, as a router will check the legality of a computer's IP address in its subnet by doing the AND operation with the subnet mask. Moreover, it is infeasible to support mobility of computers among different subnets. Fig. 1(a) gives an example, where a computer A with IP address 10.0.1.3 cannot move into subnet 2 with network segment 10.0.2.0/24, because the router in subnet 2 will drop all of its packets.

Another solution uses virtual LANs defined by IEEE 802.1Q [7]. It adds a 4-byte label in the Ethernet header to let an 802.1Q switch know which port should be used to relay the packet. Thus, the 802.1Q switch can logically divide its child switches into different virtual LANs, each corresponding to a broadcast domain, as shown in Fig. 1(b). Comparing with the solution by routers, virtual LANs can support mobility of computers. Fig. 1(b) gives an example, where computer A wants to move to virtual LAN 2. In this case, switches S_1 and S_2 have to support IEEE 802.1Q and be reconfigured to make computer A become a member of virtual LAN 2. Obviously, it requires the administrator to manually configure multiple switches, which is not efficient and flexible.

A number of research efforts aim at dealing with the broadcast problem in legacy Ethernet. Myers et al. [8] use special hardware to let switches support high-layer protocols, and convert broadcast traffic into unicast traffic. However, each switch has to record the information of all computers in the LAN. Thus, Kim et al. [9] adopt a hash table to conquer this problem by distributing computers' information among different switches. EtherProxy [10] deploys a gateway on the

entrance to each broadcast domain to record all passing packets. Then, it tries to change the broadcast address to unicast address(es) to avoid unnecessary broadcast traffic. By borrowing the idea of network address translation (NAT), EtherAgent [11] cuts a broadcast domain into internal and exterior parts, so that the amount of broadcast traffic can be reduced. Nevertheless, the effect of EtherAgent is similar to the traditional solution by routers.

3.2 SDN-based Solutions

A few studies employ SDN to solve the broadcast problem. Cho et al. [12] try to diminish ARP traffic in a large-scale data center network by implementing the SDN controller as an ARP proxy. Network administrators set up both IP and MAC addresses of data centers to the controller in advance. Then, all ARP requests are sent to the controller and the controller unicasts ARP replies to the destination computers. Except for the ARP proxy, both studies [13], [14] also make the controller become a DHCP (dynamic host configuration protocol) server [15]. Whenever a new computer joins the network, the corresponding OpenFlow switch will relay its DHCP discovery packet to the controller in order to find an unused IP address. Jehan et al. [16] use an independent DHCP server to deal with the IP address assignment, while the controller solely serves as the ARP proxy. However, all DHCP packets are still forwarded to the controller. Obviously, the above studies ask the controller to play the role of ARP proxy or DHCP server to deal with such broadcast traffic. However, they do not take advantage of SDN's property to adaptively determine the paths to route packets. Instead, these studies simply send all broadcast packets to the controller, which imposes a heavy burden on the controller.

The work of [17] considers alleviating ARP and DHCP traffic in a hybrid LAN consisting of wired and wireless networks. Rather than implement the ARP proxy and DHCP server on the controller, it asks each OpenFlow switch to relay ARP and DHCP packets to two destinations. One is the controller and the other may be the DHCP server, a computer, or flood (i.e., broadcast). Thus, the controller has the information of all computers and the DHCP server, so it can command OpenFlow switches to forward packets on the designate ports accordingly. However, since an OpenFlow switch has to send a copy of each broadcast packet to the controller, this work also puts a heavy load on the controller. Also, [17] does not consider backward compatibility. When there are traditional switches in the LAN, they still use broadcast to deal with ARP and DHCP traffic.

Comparing with existing SDN-based solutions, our ABC framework not only significantly diminishes the load of the controller by forwarding only required packets to it, but also addresses how to deal with multicast traffic. Moreover, simulation results in Section 5 will show that the ABC framework substantially reduces ARP broadcast in a LAN containing traditional switches, which demonstrates that it can support good backward compatibility with legacy Ethernet.

4 The Proposed ABC Framework

Fig. 2 illustrates the system architecture of our ABC framework based on SDN. We aim at the design of control plane (i.e., the controller) to make OpenFlow switches reduce broadcast and multicast traffic in the data plane. Each OpenFlow switch has a

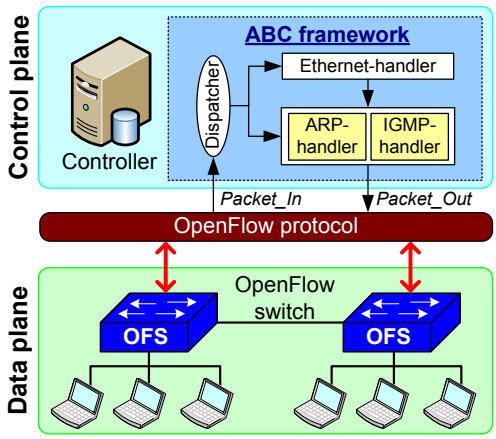


Fig. 2: System architecture of the ABC framework.

flow table to decide how to forward incoming packets. In case that it cannot find any entry from the table whose rule satisfies the incoming packet, the OpenFlow switch sends a Packet_In message with the packet's information to the controller. Then, the controller specifies the new rule and action to deal with that packet by replying a Packet_Out message. In particular, the ABC framework contains four major components to learn network status and generate necessary rules/actions:

- Dispatcher: It takes charge of discovering and interacting with OpenFlow switches. When capturing a Packet_In message, the dispatcher will relay the message to the corresponding handler.
- Ethernet-handler: This handler gives a preliminary analysis of Packet_In messages and records the information of computers in the LAN.
- ARP-handler: We develop the handler to cope with ARP traffic, whose packets occupy more than 60% of all broadcast packets (referring to Table 1).
- IGMP-handler: This handler allows OpenFlow switches to well support multicast, so as to avoid sending multicast packets to those irrelevant computers.

The ABC framework is modularized, so it is easy to support other types of traffic by adding corresponding handlers. Below, we detail our design of each component.

4.1 Design of Dispatcher

The dispatcher has two major missions. One is to discover new OpenFlow switches in the LAN. To do so, when an OpenFlow switch starts to operate, it builds a TLS (transport layer security) connection with the controller to trigger an initial handshake, as shown in Fig. 3. This connection makes both the OpenFlow switch and the controller enter the "HELLO_WAIT" state and exchange a Hello message with each other, which indicates the supported OpenFlow version. Then, the OpenFlow switch and the controller will agree to use the lowest version and finish the initial handshake procedure.

After the initial handshake procedure comes the feature discovery procedure. The OpenFlow switch turns to the "FEATURE_WAIT" state and sends a Hello message to the controller again to make it enter the same state. In this case, the controller will send a feature request to the OpenFlow switch to query its parameters. After returning the feature reply, both the OpenFlow switch and the controller enters the

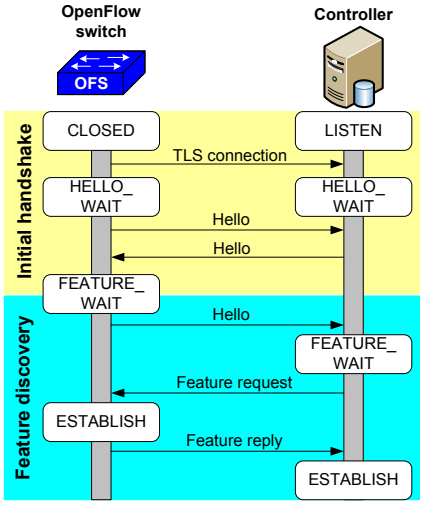


Fig. 3: The procedures of initial handshake and feature discovery.

TABLE 2: Default flow entries installed in each OpenFlow switch.

Matching rules	Actions
dl_dst=FF:FF:FF:FF:FF:FF, arp, arp_op=1	actions=CONTROLLER:6633
igmp, nw_dst=224.0.0.1	actions=CONTROLLER:6633, FLOOD
igmp, nw_dst=224.0.0.0/3	actions=CONTROLLER:6633

"ESTABLISH" state and finish the feature discovery procedure. Through the above procedures in Fig. 3, the controller can easily acquire the information of all OpenFlow switches in the LAN.

The other mission of the dispatcher is to receive Packet_In messages from OpenFlow switches and assign them to the corresponding handlers. To let OpenFlow switches transmit correct Packet_In messages, the controller will set up some default flow entries in each switch's flow table, as listed in Table 2. Specifically, the first entry indicates that every new ARP request with a broadcast address (i.e., FF:FF:FF:FF:FF:FF) must be sent to the controller with port 6633. In this way, the controller can learn the IP and MAC addresses of the corresponding computer. The second entry deals with a membership-query packet defined in IGMP, whose IP address must be 224.0.0.1. We use this entry to let the controller get the information of a multicast group. However, computers and routers also require the membership-query packet to obtain the same information. That is why we add the term "FLOOD" in the corresponding action. Finally, the last entry is responsible for processing other packets in IGMP, such as membership report and group leave. (We will further discuss these packets in Section 4.4.) Through the entries defined in Table 2, the controller can acquire necessary packets to cope with both ARP and IGMP traffic in the LAN.

4.2 Design of Ethernet-handler

The Ethernet-handler takes charge of recording the information of computers in the LAN. Therefore, whenever the dispatcher obtains a Packet_In message, it has to forward one copy to the Ethernet-handler. On the other hand, the Ethernet-handler maintains an *Ethernet table* to take down the data path identification (DPID), MAC address, and port number of each receiving Packet_In message. In OpenFlow, each device is associated with one unique DPID to help the controller distinguish different devices in the LAN. Notice that every computer has at most one record in the Ethernet table.

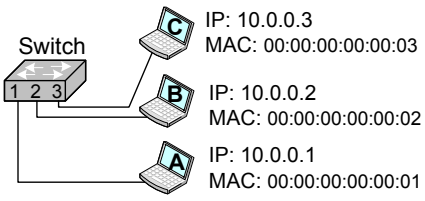


Fig. 4: An example of ARP.

Afterwards, the Ethernet-handler transmits a Packet_Out message to the corresponding OpenFlow switch to make the switch add a flow entry containing the computer's information and indicating how to deal with its packets. The following flow entry gives an example: ("dl_dst=00:00:00:00:00:01", "actions=OUTPUT:1"). It will ask the OpenFlow switch to relay the packets with destination MAC address equal to 00:00:00:00:00:01 to its port 1. In this way, the computer can have the privilege to access the Ethernet, as the OpenFlow switch can know how to process its packets.

4.3 Design of ARP-handler

ARP helps a computer learn the association between an IP address and a MAC address. To do so, each ARP packet contains four address fields: *sender hardware address (SHA)*, *sender protocol address (SPA)*, *target hardware address (THA)*, and *target protocol address (TPA)*, which indicate the MAC address of the source computer, the IP address of the source computer, the MAC address of the destination computer, and the IP address of the destination computer, respectively.

We take an example in Fig. 4 to explain how ARP works, where three computers A, B, and C connect together by a switch via its ports 1, 2, and 3, respectively. Suppose that computer A has a packet to be sent to computer C. It determines that computer C has the IP address of 10.0.0.3 (e.g., through the domain name server). To transmit the packet, computer A should know computer C's MAC address. Thus, computer A first adopts a cached ARP table to search 10.0.0.3 for any existing entry of computer C's MAC address (i.e., 00:00:00:00:00:03). If the table returns no result, computer A sends an ARP request with the destination MAC address of FF:FF:FF:FF:FF:FF (i.e., broadcast address) and the source MAC address of 00:00:00:00:00:01. When the switch receives this ARP request, it broadcasts the request to all its ports. On the other hand, when computer C gets the ARP request, it sends an ARP reply with the destination MAC address of 00:00:00:00:00:01 and the source MAC address of 00:00:00:00:00:03. In this case, the switch will relay the ARP reply to computer A through its port 1. Afterwards, computer A will cache this information in its ARP table. Next time when it wants to send a packet to computer C again, it broadcasts an Ethernet frame with the destination MAC address of 00:00:00:00:00:03, containing the IP packet to the LAN.

ARP is easy to use, but it will generate a lot of ARP requests when many computers query their target MAC addresses. Even when a computer knows the MAC address of its target (from the ARP table), it still uses broadcast to send the packet. Thus, the LAN will be inevitably congested by spam packets caused by ARP. To conquer this problem, our idea is to allow the controller to receive ARP packets, so as to

TABLE 3: The flow entries generated by the ARP-handler based on the example in Fig. 4.

Matching rules	Actions
arp, tpa=10.0.0.1	actions=set_field:00::01->the_dst, OUTPUT:1
arp, op=2, spa=10.0.0.3, tpa=10.0.0.1	actions=CONTROLLER:6633, OUTPUT:1
	actions=CONTROLLER:6633

learn the information of computers (e.g., IP/MAC addresses and ports). Then, it installs corresponding flow entries in OpenFlow switches to convert ARP broadcast packets into unicast packets.

In practical implementation, when an OpenFlow switch receives an ARP request, it first checks whether the ARP request can be sent to the right computer via unicast by referring to its flow table. If there is no flow entry relevant to the IP address indicated in the ARP request, the OpenFlow switch still broadcasts the ARP request and also sends one copy to the controller. On the other hand, the controller can learn a computer's IP/MAC addresses through its first ARP request and also the target computer's IP/MAC addresses through the corresponding ARP reply. This can be done by extracting both the SPA and SHA fields from each ARP packet. In this way, the controller can quickly learn the IP and MAC addresses of computers in the LAN.

Let us use the example in Fig. 4 again to show how to generate the corresponding flow entries by the ARP-handler, as presented in Table 3. Suppose that the controller has received the ARP request from computer A that queries the MAC address of computer C. In this case, the controller can learn both IP and MAC addresses of computer A. Thus, the first flow entry indicates that all (ARP) packets with IP address 10.0.0.1 should be transmitted to computer A by unicast through port 1 of the OpenFlow switch. Then, since it is expected that computer C will send an ARP reply to computer A, the second flow entry will ask the OpenFlow switch to also forward the ARP reply to the controller, so as to help it acquire the MAC address of computer C. Notice that this flow entry is temporary, for example, with lifetime of five seconds. When the controller receives the ARP reply later, it can install a new flow entry below: ("arp, tpa=10.0.0.3", "actions=set field:00::03->eth_dst, OUTPUT:3") This flow entry asks the OpenFlow switch to convert all ARP broadcast packets to computer C into unicast packets. Finally, all other ARP packets whose IP addresses do not appear in the flow table (e.g., computer B) will be sent to the controller for processing.

Our ARP-handler has two special designs. First, unlike most of existing SDN-based solutions discussed in Section 3.2, we do not make the controller act as an ARP proxy to deal with all ARP packets in the LAN. Instead, the ARP-handler only processes one pair of ARP request and reply for each unknown computer. Therefore, the load of the controller can be significantly reduced. Second, once the ARP-handler knows the IP and MAC address of a computer, it will install a flow entry to allow the OpenFlow switch to convert the broadcast address into a unicast address. This design considers the backward compatibility to traditional switches. In particular, since the broadcast address has been converted to the unicast address, such a switch will not send the ARP packet to all its ports but only forward the packet to the port linked to the target computer, thereby reducing spam packets generated by ARP.

TABLE 4: An example of flow entries generated by the IGMP-handler.

Matching rules	Actions
igmp, nw_dst=224.0.0.0/3	actions=CONTROLLER:6633, OUTPUT:10
ip, nw_dst=233.0.0.1	actions=GROUP:3909091329

4.4 Design of IGMP-handler

IGMP allows computers to manage their multicast group membership. Routers also use IGMP to discover group members. In IGMP, two roles are defined: *querier* and *host*. A querier periodically sends a *membership-query* packet to get the information of multicast group. Then, four cases may occur: 1) If it receives a membership-query packet from another with a smaller IP address, this querier becomes a host. Thus, each multicast group will have just one querier. 2) If the querier cannot hear any *membership-report* packet after a predefined timeout, it will not forward multicast data packets as there are no members in the multicast group. 3) If the querier receives membership-report packets, it starts forwarding multicast data packets to the group's members. 4) If the querier receives a *group-leave* packet, it stops forwarding multicast data packets.

On the other hand, a host will conduct the following operations: 1) When receiving a membership-query packet, the host has to reply a membership-report packet to the querier. However, if the host has heard membership-report packets sent from other members in the same multicast group, it need not reply its membership-report packet. 2) If a computer wants to join the multicast group, it actively sends a membership-report packet to notify the querier. 3) A host can directly leave the multicast group without sending any packet to the querier. However, if the host is the last one that sent the membership-report packet, it has to send a group-leave packet to notify the querier.

A traditional layer-2 switch will broadcast IGMP and multicast data packets to all its ports. Thus, such packets may result in unnecessary load on those computers that do not join the multicast group, as they have to also process these irrelevant packets in layer 3. IGMP snooping allows the switch to use a table to map between ports and multicast traffic, so as to filter out irrelevant multicast data packets. However, it relies on the prerequisite that there exists a multicast router to keep generating membership-query packets, and these packets must be forwarded by all switches. Besides, IGMP snooping is a layer-2 optimization for layer-3 IGMP. Not all switches can fully support IGMP snooping. In IGMP snooping, each switch has to find out the relationship between its ports and the received multicast traffic. In contrast to this mechanism, our IGMP-handler provides a more efficient solution to let the controller install flow entries learned from IGMP packets in OpenFlow switches to facilitate filtering multicast traffic. Below, we discuss how the IGMP-handler deals with different types of IGMP packets.

Membership query: This packet is sent by the querier, so the IGMP-handler can acquire its IP and MAC addresses, and also the port of the OpenFlow switch that connects with the querier. Consequently, the IGMP-handler can add a flow entry to the OpenFlow switch to ask it to forward following IGMP packets to both the controller and the querier. The first flow entry in Table 4 gives an example. The flow entry commands the OpenFlow switch to forward all IGMP packets (with a multicast IP address of 224.0.0.0) to not only the controller (with port 6633) but also the querier that connects to the

TABLE 5: An example of the group table.

Group ID	Bucket
group_id =3909091329	bucket=actions=set_field:00:01->eth_dst, OUTPUT:1 bucket=actions=set_field:00:05->eth_dst, OUTPUT:3 bucket=actions=set_field:00:07->eth_dst, OUTPUT:4

switch's 10th port. In this way, the controller can acquire the information of the multicast group based on the following IGMP packets. Besides, the OpenFlow switch can relay IGMP packets (e.g., membership reports) merely to the querier, which helps reduce the amount of unnecessary IGMP traffic.

Membership report: Due to the above flow entry, the controller can also receive the membership-report packet sent from a member in the multicast group. To let the controller learn all members, each member has to reply the membership-report packet for the first time that it receives the membership-query packet. Then, the controller maintains a *group table* to record every member that it has learned. Each group is associated with one unique group ID, which is converted from the corresponding multicast IP address. Then, the IGMP-handler adds a flow entry to direct multicast data packets to these members. Table 5 shows an example of the group table, where the corresponding multicast IP address is 233.0.0.1. We can convert the IP address to an integer of 3909091329 to be the group ID. The controller learns three group members with MAC addresses of 00:00:00:00:00:01, 00:00:00:00:00:05, and 00:00:00:00:00:07, which connect to the OpenFlow switch via its ports 1, 3, and 4, respectively. In addition, the IGMP-handler adds the second flow entry in Table 4 to ask the OpenFlow switch to refer to the group table to forward multicast data packets. In this way, we can make sure that multicast data packets will be transmitted to only the members in the multicast group.

Group leave: As mentioned earlier, the group-leave packet is sent by the last member in the multicast group. Consequently, once the controller receives this packet, the IGMP-handler will remove the corresponding record in the group table.

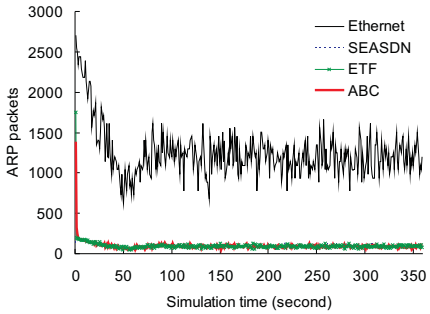
In the IGMP-handler, since a multicast address will be translated to the unicast address of each member in the multicast group, a switch can forward multicast data packets only to those ports that connect with member computers. In this way, we can avoid unnecessary multicast traffic and provide backward compatibility with legacy Ethernet.

5 Performance Evaluation

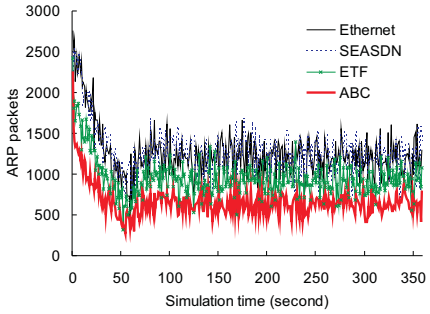
This section evaluates system performance of our ABC framework by simulations and practical deployment. We adopt the Mininet simulator [18] with version 2.2.1, which supports OpenFlow with version 1.3 and RYU with version 3.29 (for the controller). Simulation results of both ARP and IGMP traffic will be investigated. Afterwards, we implement the ABC framework on our campus network and measure its performance.

5.1 ARP Experiment by Mininet

In the first experiment, we consider a LAN with 50 computers. Each computer has an ARP table to cache the mapping of IP and MAC addresses that it learns from ARP packets. Following the default setting in Linux, each cached record will become overdue and be removed from the ARP table every 60 seconds. In addition, two network scenarios are addressed. In the scenario of *one large switch*, all computers are connected by a single switch (with 50 ports). For legacy Ethernet, the switch is a traditional switch. For an SDN-based method, the switch is an OpenFlow switch. We use the scenario to simulate large switches, for example, used in a data center network. In the scenario of *three hybrid switches*, the switches are organized hierarchically, where a root switch links to two traditional switches, each further connecting with 25 computers. For legacy Ethernet, the root is a traditional switch. For an SDN-based method, the root is an OpenFlow switch. This scenario considers an application where the network administrator wants to add some OpenFlow switches in a LAN originally consisted of traditional switches. It also helps evaluate the degree of backward compatibility of each SDN-based method.



(a) one large switch



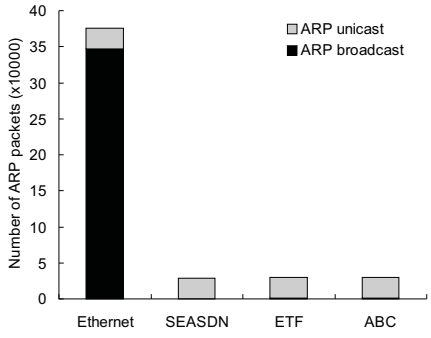
(b) three hybrid switches

Fig. 5: Comparison on the number of ARP packets generated per second.

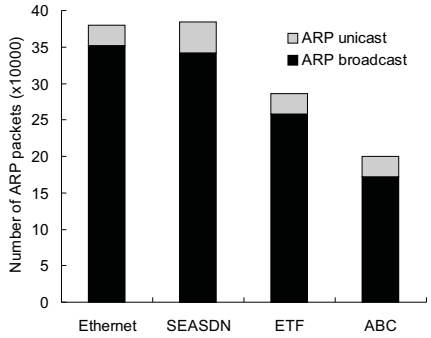
We compare the ABC framework with two SDN-based methods discussed in Section 3.2: SEASDN (scalable Ethernet architecture using software defined networking) [16] and ETF (extensible transparent filter) [17], whose objectives are also to decrease ARP broadcast traffic in the LAN.

Fig. 5(a) shows the number of ARP packets generated every second in the scenario of one large switch. Since the cache timeout is 60 seconds, the number of ARP packets in legacy Ethernet will gradually decrease before the first 60 seconds and then become relatively stable. For SDN-based methods (i.e., SEASDN, ETF, and ABC), there is an impulse in the beginning, because the controller has no information of computers initially. After the controller knows every computer in the LAN (around 1–2 seconds), it can ask the OpenFlow switch to cut out ARP broadcast packets and replace them by unicast packets. Thus, SDN-based methods can greatly reduce ARP traffic.

Fig. 5(b) gives the number of ARP packets generated per second in the scenario of three hybrid switches. Since the two traditional switches do not understand the SDN rules set by the controller, they still use broadcast to send out

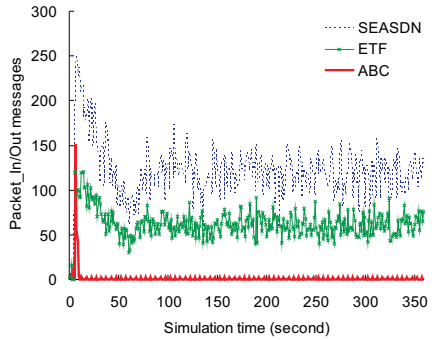


(a) one large switch

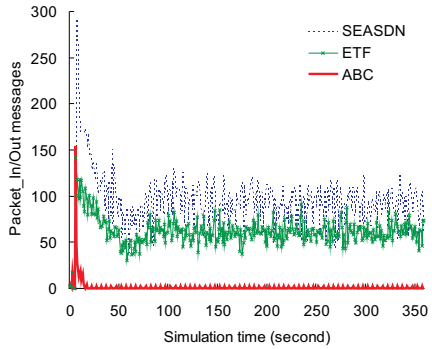


(b) three hybrid switches

Fig. 6: Comparison on the aggregate number of ARP packets transmitted.



(a) one large switch



(b) three hybrid switches

scenario	SEASDN	ETF	ABC
one large switch	43,004	22,373	490
three hybrid switches	33,109	22,222	539

(c) aggregate number of Packet_In/Out messages

Fig. 7: Comparison on the number of Packet_In/Out messages.

ARP packets. Thus, the number of ARP packets by SDN-based methods increase accordingly. As mentioned earlier in Section 3.2, SEASDN forces the controller to act as the ARP proxy, so it results in almost the same performance with legacy Ethernet in this scenario. ETF, on the other hand, asks switches to forward ARP packets to the corresponding ports but the MAC addresses of these packets are still the broadcast address. In this case, the traditional switches will broadcast these ARP packets as usual. Comparing with both SEASDN and ETF, our ABC framework can adaptively translate the broadcast address of an ARP packet to the unicast address of the receiving computer. Consequently, the traditional switches will forward ARP packets through unicast rather than broadcast, thereby significantly reducing unnecessary ARP broadcast traffic.

Fig. 6 presents the aggregate number of ARP packets transmitted during 360-second simulation time. We can observe that broadcast packets occupy a very large portion of ARP traffic in legacy Ethernet. On the contrary, SDN-based methods result in almost no ARP broadcast in the scenario of one large switch. However, when traditional and OpenFlow switches coexist (i.e., the scenario of three hybrid switches), SEASDN performs as worse as legacy Ethernet, which indicates that it cannot well support backward compatibility. On the other hand, by allowing the controller to adaptively convert the ARP broadcast address to unicast address(es), our ABC framework can reduce around 51.2% of ARP broadcast packets even when there are some traditional switches in the LAN, which demonstrates its effectiveness and flexibility.

We then evaluate the controller's overhead by measuring the number of Packet_In and Packet_Out messages received by and sent from the controller, respectively, as shown in Fig. 7. SEASDN makes the controller serve as an ARP proxy, so it uses Packet_In messages to forward ARP requests to the controller and Packet_Out messages to forward ARP replies to the destination computers. Therefore, one ARP procedure will generate two Packet_In messages and two Packet_Out messages in SEASDN. On the other hand, ETF also uses Packet_In messages to forward ARP packets to the controller, and it generates two Packet_In messages for each ARP procedure. Therefore, ETF suffers from lower overhead than SEASDN. In the ABC framework, the controller requires Packet_In messages to learn the computers in the LAN and uses Packet_Out messages to set transmission rules in OpenFlow switches initially, so there exists an impulse in the beginning (around the 6–9th seconds) in both Fig. 7(a) and (b). Afterwards, the ABC framework generates very few Packet_In and Packet_Out messages as the controller has obtained the IP and MAC addresses of all computers. Thus, the ABC framework greatly reduces the overhead comparing with both SEASDN and ETF, as shown in Fig. 7(c).

5.2 IGMP Experiment by Mininet

We then study the performance of our ABC framework on IGMP multicast traffic. Because SEASDN and ETF cannot cope with multicast traffic, we do not compare them with the ABC framework as their performance will be the same with legacy Ethernet. In this experiment, we consider one multicast server (i.e., IGMP querier) and six computers connected together by a switch. The switch will be a traditional and OpenFlow switch in legacy Ethernet and the ABC framework, respectively. The server keeps sending IGMP and multicast data packets to a multicast group with IP address of 233.0.0.1 every second. On the other hand, each computer has different behavior. For computers 1 and 4, they join the multicast group with address

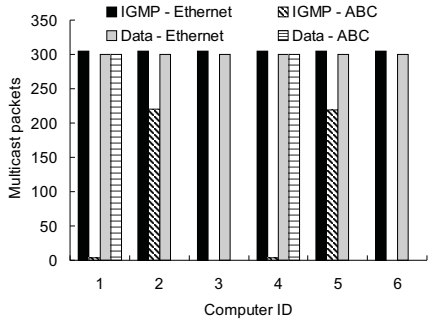


Fig. 8: Comparison on multicast packets received by different computers.

of 233.0.0.1 in the beginning and do not leave the group. Computers 2 and 5 arbitrarily join a multicast group for two seconds, leave the group, and then stay idle for one second. The above procedure is repeated. For computers 3 and 6, they do not join any multicast group.

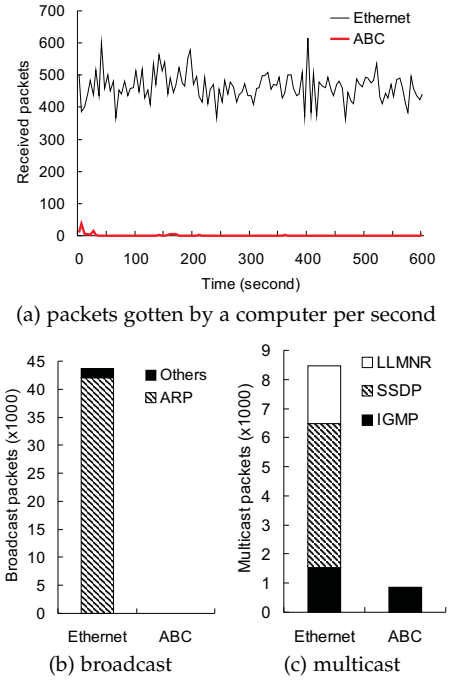
Fig. 8 gives the number of multicast packets received by different computers during 300-second simulation time. Since the traditional switch does not support IGMP snooping, it simply broadcasts IGMP and multicast data packets to all computers. In this case, even though a computer does not join any multicast group (e.g., computers 3 and 6), it still has to receive and process such irrelevant packets. On the contrary, our ABC framework can assist the controller in learning the group members and converting the multicast address accordingly. Thus, the six computers will have different results. Specifically, for computers 1 and 4, because they stay in the same multicast group with the server, the OpenFlow switch only sends them few IGMP membership queries in the beginning and keeps relaying multicast data packets from the server. These packets are transmitted using unicast. For computers 2 and 5, since they arbitrarily join a multicast group in each short period, they will receive more IGMP packets. However, as they do not join the multicast group with IP address of 233.0.0.1, the OpenFlow switch will not relay multicast data packets from the server to them. Finally, as computers 3 and 6 do not join any multicast group, they will not receive IGMP or multicast data packets.

To sum up, the ABC framework will not send multicast data packets to those computers that join other multicast groups. In addition, when a computer does not participate in any multicast group, it will not be bothered by IGMP or multicast data packets. Therefore, our ABC framework can diminish unnecessary multicast traffic and save the network bandwidth, which helps improve Ethernet efficiency.

5.3 Practical Deployment

We also implement the ABC framework on our campus network mentioned in Section 1 to verify its practicability. To do so, we use a TP-LINK WR1043NR switch and update its firmware by OpenWrt [19] to make the switch support OpenFlow. In addition, we adopt TShark [20] to dump and analyze network traffic.

Fig. 9 gives the analysis of packets received by two computers during 600 seconds, where one computer connects with a traditional switch while the other computer links to our OpenFlow switch. From Fig. 9(a), we observe that each computer receives around 400–600 packets every second in legacy Ethernet. However, a lot of packets are spam in terms



(a) packets gotten by a computer per second

(b) broadcast (c) multicast

Fig. 9: Experimental result by practical deployment.

of the receiving computer. In contrast, our ABC framework eliminates most of such spam packets, so the computer can efficiently receive only its necessary packets.

Fig. 9(b) shows the number of broadcast packets received by a computer, where ARP broadcast packets dominate all broadcast packets. In legacy Ethernet, the traditional switch sends more than 42,000 broadcast packets to the computer in 600 seconds. On the contrary, the OpenFlow switch cuts them out and transmits fewer than 30 ARP broadcast packets to the computer, which saves more than 99.93% of ARP broadcast traffic. In addition, since the controller can adaptively convert a broadcast address to the unicast address of the receiving computer, the OpenFlow switch will not send irrelevant broadcast packets to the computer.

Fig. 9(c) presents the number of multicast packets received by a computer. Since the computer joins the IGMP multicast group, it will receive a few membership queries and multicast data packets in the ABC framework. On the other hand, because the traditional switch cannot support multicast protocols such as LLMNR and SSDP, the switch uses broadcast to send out their packets. In the experiment, since the computer is not a destination of LLMNR and SSDP senders, the OpenFlow switch will eliminate these irrelevant packets. That is why there are no LLMNR and SSDP packets in the ABC framework. Through the experiment, we demonstrate that our ABC framework can efficiently reduce unnecessary broadcast and multicast traffic, which significantly improves the transmission efficiency in a practical LAN.

6 Conclusion

Ethernet is the basic technique used in layer 2 but it unavoidably generates a lot of spam packets due to the broadcast nature. Based on SDN, we propose the ABC framework to reduce unnecessary broadcast and multicast packets and improve Ethernet efficiency. Through Mininet simulations, we show that the proposed ABC framework not only reduces ARP broadcast traffic, especially when there exist traditional

switches in the LAN, but also saves the controller's overhead, as compared with other SDN-based approaches such as SEASDN and ETF. In addition, the ABC framework can efficiently decrease IGMP traffic than legacy Ethernet. By implementing the ABC framework on our campus network, we also demonstrate its effectiveness and practicability. For future work, we will investigate how to apply the SDN technique to improve performance of a DVB-H (digital video broadcasting-handheld) network, which also relies on broadcast to provide mobile TV services [21], [22].

REFERENCES

[1] Y.D. Lin, R.H. Hwang, and F. Baker, *Computer Networks: An Open Source Approach*.New York: McGraw-Hill Education, 2012.
[2] IETF, "Simple service discovery protocol/1.0," 1999.
[3] IETF, "Link-local multicast name resolution (LLMNR)," RFC 4795, 2007.
[4] W. Xia, Y. Wen, C. H. Foh, D. Niyato, and H. Xie, "A survey on software-defined networking," *IEEE Comm. Surveys & Tutorials*, vol. 17, no. 1, pp. 27–51, 2015.
[5] I.F. Akyildiza, A. Leea, P. Wang, M. Luoc, and W. Chou, "A roadmap for traffic engineering in SDN-OpenFlow networks," *Computer Networks*, vol. 71, pp. 1–30, 2014.
[6] Ryu SDN framework. [Online]. Available: http://osrg.github.io/ryu/
[7] IEEE Standard 802.1Q, "IEEE standard for local and metropolitan area networks – virtual bridged local area networks," 2005.
[8] A. Myers, E. Ng, and H. Zhang, "Rethinking the service model: scaling Ethernet to a million nodes," *Proc. ACM SIGCOMM Workshop on Hot Topics in Networking*, 2004, pp. 1–6.
[9] C. Kim, M. Caesar, and J. Rexford, "SEATTLE: a scalable Ethernet architecture for large enterprises," *ACM Trans. Computer Systems*, vol. 29, no. 1, pp. 1:1–1:35, 2011.
[10] K. Elmeleegy and A. L. Cox, "EtherProxy: scaling Ethernet by suppressing broadcast traffic," *Proc. IEEE INFOCOM*, 2009, pp. 1584–1592.
[11] C.H. Chiu and C.L. Lei, "Etheragent: scaling Ethernet for enterprise and campus networks," *Int'l J. Innovative Computing, Information and Control*, vol. 9, no. 6, pp. 2465–2483, 2013.
[12] H. Cho, S. Kang, and Y. Lee, "Centralized ARP proxy server over SDN controller to cut down ARP broadcast in large-scale data center networks," *Proc. IEEE Int'l Conf. Information Networking*, 2015, pp. 301–306.
[13] J. Wang, T. Huang, J. Liu, and Y. Liu, "A novel floodless service discovery mechanism designed for software-defined networking," *China Comm.*, vol. 11, no. 2, pp. 12–25, 2014.
[14] P.W. Chi, Y.C. Huang, J.W. Guo, and C.L. Lei, "Give me a broadcast-free network," *Proc. IEEE Global Comm. Conf.*, 2014, pp. 1968–1973.
[15] IETF, "Dynamic host configuration protocol," RFC 2131, 1997.
[16] N. Jehan and A.M. Haneef, "Scalable Ethernet architecture using SDN by suppressing broadcast traffic," *Proc. Int'l Conf. Advances in Computing and Comm.*, 2015, pp. 24–27.
[17] K. Kataoka, N. Agarwal, and A.V. Kamath, "Scaling a broadcast domain of Ethernet: extensible transparent filter using SDN," *Proc. IEEE Int'l Conf. Computer Comm. and Networks*, 2014, pp. 1–8.
[18] Mininet. [Online]. Available: http://mininet.org/
[19] OpenWrt. [Online]. Available: https://openwrt.org/
[20] TShark. [Online]. Available: https://www.wireshark.org/docs/man-pages/tshark.html
[21] W.H. Yang, Y.C. Wang, Y.C. Tseng, and B.S.P. Lin, "A request control scheme for data recovery in DVB-IPDC systems with spatial and temporal packet loss," *Wireless Comm. and Mobile Computing*, vol. 13, no. 10, pp. 935–950, 2013.
[22] Y.C. Wang, "Profit-based exclusive-or coding algorithm for data retransmission in DVB-H with a recovery network," *Int'l J. Comm. Systems*, vol. 28, no. 9, pp. 1580–1597, 2015.