Using a P4 Hardware Switch to Block Trackers and Ads for All Devices on an Edge Network

Shie-Yuan Wang and Jin-Ting Li
Department of Computer Science
National Yang Ming Chiao Tung University, Taiwan
shieyuan@cs.nycu.edu.tw

***Abstract*—Nowadays, when a user downloads a web page, many targeted or untargeted advertisements are embedded into the downloaded web page. These advertisements slow down the page download and display and occupy a very large portion of the screen. Besides, they unnecessarily consume more network bandwidth. To solve these problems, in this paper we design and implement a method inside a P4 hardware switch to block trackers and advertisements. Our method successfully blocks their operations by blocking the DNS requests issued by them. Our method is to be deployed at a switch that connects an edge network to the Internet. Using such a configuration, all devices on the edge network will be automatically protected by our method without the need to install a per-device protection software. Experimental results show that our method can block trackers and advertisements for all devices on an edge network, save the network bandwidth consumed by them, and enable a web browser to download and display web pages faster.**

***Index Terms*—privacy protection, tracker, advertisement blocking, programmable switches, P4**

I. Introduction

Recently, web trackers and advertisements are everywhere when a person browses web pages on the Internet. Web trackers, which are scripts and mostly written in JavaScript, are purposely embedded into a web page by the web page developer (for commercial benefits) and they work underground. These trackers collect the user's information such as the settings of the used device and the user's behavior such as what items on a page the user clicks. Then, they send the collected data to remote analytics websites, which collaboratively analyze and identify the user and his interests. On these analytics websites, a user profile is created for each identified user to store information about him. Over time, as a user visits more and more web pages that have trackers embedded into them (note that these web pages may be hosted at different web sites), more and more information about the user will be added to the profile of the user. As a result, the privacy of the user is gradually infringed. Currently, the Safari web browser running on Apple Inc.'s devices can list the visited websites that embedded trackers into their web pages and the analytics websites that these trackers sent their data to in the past 30 days. From the large numbers of these tracker-embedded websites and analytics websites, one can see that this privacy-infringing data collection task happens every day and poses a great threat to user privacy.

In addition to the above tracker-embedded websites and analytics websites, many websites nowadays place advertisements on their web pages to make money. Showing advertisements on a web page actually is purposely performed by the web page developer (again for commercial benefits). The web page developer purposely embeds scripts into the web page. When these web pages are downloaded to the user's web browser and their embedded scripts get executed, these scripts will connect to some advertisement websites with the fingerprint of the user (i.e., the information that can identify the user). These advertisement websites then contact the above analytics websites and use the fingerprint as the key to get the profile of the user. After getting the preference of the user, the advertisement websites then return selected advertisements to the web browser. As a result, many advertisements targeted to the user are shown on the downloaded web page. In this paper, we call such a script "advertisement downloader" as its purpose is to download advertisements.

From the above explanations, one can see that the tracker-embedded websites, analytics websites, and advertisement downloader-embedded websites collaborate together to carry out precision marketing. In such an environment, the privacy of the user is lost. Besides, the inserted advertisements unnecessarily increase the loading time and displaying time of a web page and consume more network bandwidth. These advertisements also occupy a large portion of the screen hindering the user from reading the contents of the original web page. Such problems are especially serious for mobile devices that are equipped with a less powerful CPU and a small screen and the available bandwidth between them and their accessed websites is small.

At present, the most common way to defend web trackers and advertisement downloaders is by installing an "ad blocker" extension to the web browser. (Note that "ad" is commonly used as a short name for advertisement in the advertisement placing/blocking industry. Thus, in the rest of the paper, we may use "ad" to replace "advertisement" when there is no ambiguity.) Web trackers and ad downloaders work similarly. Most of them are written in JavaScript and embedded in the web pages. Furthermore, they both use HyperText Transfer Protocol (HTTP) to send requests to and get replies from third-party websites. Thus, most ad blockers block trackers and ad downloaders by establishing and using a filter list. When an HTTP request is made by a tracker or ad downloader, they will check whether the domain name in the HTTP header of the request appears on the filter list. If yes, the request will be

blocked. Otherwise, the request is allowed to be sent to the destination website.

Although the above method works well for web browsers, since it is implemented as an extension of the web browser, it only works for web browsers and cannot block trackers used in other applications. For example, it cannot block the trackers embedded in Facebook pages when one uses the Facebook app on a mobile device like iPad. As another example, it cannot block the trackers embedded in YouTube pages when one uses the YouTube app on a mobile phone like iPhone. Although some ad blockers can be installed on a mobile device as an app, due to the strict security-protection measures imposed by the mobile device, they are not allowed to interfere with other apps and thus they cannot block trackers or ad downloaders embedded in other apps.

To solve the above problems, we want a method that can block trackers and ad downloaders for different kinds of devices and applications. Besides, to be able to automatically protect all devices on a network, this method should be deployed in a hardware switch that connects an edge network to the Internet. In this way, when a tracker or ad downloader running on the edge network sends its HTTP request to a third-party website on the Internet, this method can intercept, inspect, and block the request if the domain name carried in the HTTP header of the request appears on the filter list. Although this method works for the HTTP protocol; however, we observed that nowadays many websites have started to use the HyperText Transfer Protocol Secure (HTTPS) protocol to encrypt and send their data. For such websites, the above method may not work.

To overcome this difficulty, we improved the above method and used a DNS-based approach in our method. Before a tracker or ad downloader sends HTTP requests to a collaborating website, it must first resolve the domain name of the website into the corresponding IP address. This name resolution is performed by the tracker or ad downloader sending a DNS request packet to the DNS server. Unlike HTTPS packets, DNS packets are not encrypted. Furthermore, unlike an HTTP packet, which contains many variable-length HTTP header fields and thus cannot be easily parsed by a switch, the format of a DNS request packet can be more easily parsed by the switch. Taking advantage of these properties, our method uses a Programming Protocol-independent Packet Processors (P4) [1] hardware switch to parse all DNS request packets passing through it. If the domain name carried in a DNS request packet is found to be on the filter list, our method will return 0.0.0.0 in the DNS reply packet as the resolved IP address for the queried domain name. This 0.0.0.0 indicates an invalid IP address, which will cause the tracker or ad downloader to abandon connecting to its collaborating website. As a result, the downloaded web page can be displayed cleanly without ads and in less time.

In this paper, we make the following contributions. Firstly, we design a method that can block trackers and ad downloaders running on all types of devices residing on an edge network. The trackers and ad downloaders that are blocked by our method are not limited to those used in the web application. Instead, the trackers and ad downloaders used in other apps running on a mobile device such as the FaceBook or YouTube apps can be blocked by our method as well. Secondly, using our method, if a new device is attached to the edge network (either wired or wireless), our method will automatically protect the new device and the user of the device need not configure anything. Thirdly, we have successfully implemented our method in a P4 hardware switch and evaluated its performance in the real life. Experimental results show that our method is effective in blocking trackers and ad downloaders. In addition, our method operates at a higher speed than an ad-block scheme named "Pi-hole" and is more secure than it.

II. Related Work

Several papers have studied trackers and their relationships with ads. The paper [2] reviewed how trackers work. It pointed out some security and privacy issues raised by trackers and surveyed several policies and tools to deal with them. The authors in [3] analyzed different behavior of trackers and listed several countermeasures against them. They also discussed different ways to place ads without infringing user privacy.

Nowadays, several software-based methods exist to guard against ads and trackers (e.g., "Adblock Plus" [4] and "Adguard" [5]) and several papers have studied them. In [6], the authors focused on how to block trackers and analyzed several web browser extensions for blocking different types of trackers. The authors in [7] tested various configurations in ad blockers and compared their performance in blocking ads. In [8], the authors measured the amount of bandwidth saved with ad blockers by instructing volunteers to browse websites randomly in the real life. In [9], the authors tried different filter lists for ad blockers and studied their impact on performance. In [10], the authors performed a large-scale study on the actual performance improvements made by using content-blockers.

Building a filter list is the most common way to guard against trackers. However, this mechanism can be easily bypassed. Many filter lists (e.g., EasyList [11]) that ad blockers use are open source. Once the tracker developers have noticed that their domain names have been included in the list, they can rename their domain names to pass through the deny list. Thus, the filter list must be updated constantly to stop trackers. Some papers [12], [13], [14] tried to solve this problem by machine learning. In [12], the authors proposed to learn and detect trackers by analyzing the DOM (Document Object Model) of web pages. The authors in [13] utilized machine learning to detect ads based on their HTML structure and JavaScript behavior. The authors in [14] instead studied various algorithms to learn the ads behavior of HTTP requests. Although these studies show that using machine learning to detect ads is feasible, it is hard to implement these schemes on a mobile device due to the high complexities of their machine learning models.

"Pi-hole" [15], like our method, can provide network-wide protection for devices residing on a network. However,

unlike our method, "Pi-hole" is a software-based method and functions as a DNS server. "Pi-hole" needs to be run on a device to act as the default DNS server for the devices that want to use it to guard against trackers and ads. The users of these devices need to manually set the IP addresses of their default DNS servers to the IP address used by "Pi-hole." Like our method, "Pi-hole" downloads a filter list from the Internet and checks whether the domain name carried in a DNS request appears on the filter list. If yes, like our method, it also returns 0.0.0.0 as the resolved IP address in the DNS reply packet. Otherwise, it will forward the DNS request to a (real) DNS server in the Internet DNS hierarchy to continue the name resolution process.

Compared with "Pi-hole," our method differs from it in several aspects. Firstly, our method can automatically protect all devices on a network. In contrast, in "Pi-hole" only those devices that set the "Pi-hole" server as their default DNS server can be protected by "Pi-hole." Secondly, our method is more secure than "Pi-hole." Our method is executed in the hardware switch, which requires the network manager to deploy and manage, thus our method is more secure. In contrast, "Pi-hole" is an open source software DNS server that can be run up by any person and on any device. If a malicious user implants a piece of malware into his "Pi-hole" server and advertises his "Pi-hole" server as the default DNS server to all devices on a network, in addition to blocking trackers and ads, the malicious "Pi-hole" server can perform many underground attacks such as DNS Spoofing and DNS Flooding. Thirdly, since our method is a hardware-based method, our method operates at higher speeds than "Pi-hole," which will be shown later in Section V.

III. Design and Implementation

A P4 [1] switch is composed of several main components. The first component is the packet parser, which is needed to parse the headers of a packet. The packet parser of our method can recognize DNS packets among all packets that pass through the switch. Because our method only focuses on the DNS packets generated by trackers or ad downloaders, our method is designed to recognize only the DNS packets that use User Datagram Protocol (UDP) and contain only one type A or AAAA query. This format is the most common format used by DNS packets on the Internet. For all other kinds of DNS packets, our parser accepts and forwards them out of the switch normally.

The P4 hardware switch used in this work is an Inventec D10056 switch [16], which uses the Intel/Barefoot Tofino chip as its P4 switching ASIC. The P4C compiler that comes with this switch can only parse packet headers composed of fixed-length fields. This limitation poses a problem for our method as the length of the domain name in a DNS packet is variable. To handle this problem, our method uses several parser states to parse the fields of a DNS packet. We found that because there is only one domain name carried in the DNS packet and this part is the only part whose length is not fixed, we can first compute the length of the domain name from the length of the UDP packet, whose payload carries the DNS request. Then, we can use a P4 switch counter to repeatedly parse the domain name one byte at each time.

Fig. 1 illustrates how our P4 packet parser parses a DNS packet. The first state is the initial state. In the second state, the parser parses the IP/UDP headers and gets the length from the UDP header. Then, the parser continues to parse all DNS fields before the domain name field in the third state. Note that in the DNS packet, the domain name field is named the QueryName field. In the following state, the parser computes the length of the domain name by subtracting the lengths of all fixed fields (including the QueryType and QueryClass fields that follow the QueryName field, each of which occupies 2 bytes) from the length in the UDP header and then sets the counter to this value. In the fifth state, the parser parses the domain name one byte at each time and also decreases the counter's value by one until it reaches 0. Finally, in the final state, the parser continues to parse the rest fields of the packet and completes the parsing. Note that the rest fields here mean the QueryType and QueryClass fields. Our P4 program needs to extract their values to correctly process and form the DNS reply packet. Our current packet parser design allows our scheme to parse a domain name with up to 255 characters. This maximum length setting can be increased if needed and is large enough for our current uses.

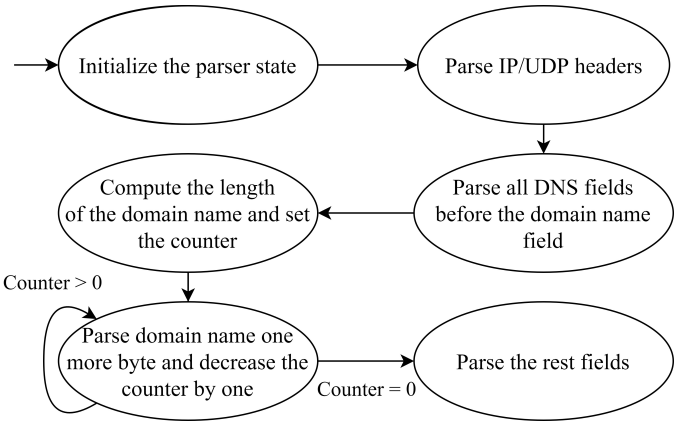

Fig. 1: The state diagram of our packet parser

After the domain name is extracted from the DNS packet, based on whether it appears on the filter list or not, our P4 program determines whether to block or accept it. Currently, the domain name filter list [17] used in our work is imported from "Pi-hole." This list includes about 100,000 domain names related to trackers and ad downloaders. Note that other filter lists can also be used and downloaded from the Internet. Actually, different ad blockers maintain their own filter lists.

The operations of a P4 switch are based on match-action tables. One limitation of the P4 match-action table is that the match field of a table must to be fixed-length. However, the domain name extracted from a DNS packet is variable length. To deal with this problem, our method hashes all bytes of the domain name into a 64-bit value and uses this value as the

match key to look up the match-action table.

To support high-speed packet switching, the match-action tables in a P4 switch share a block of high-speed memory. Because there are many domain names on the filter list, pre-installing an entry for every domain name on the filter list does not make the best use of this high-speed memory. In our method, initially the match-action table is empty without any entry. When our method "sees" a domain name that it did not see before, it will send the domain name to our controller asking it to check whether the domain name is on the filter list. Based on the result, the controller then installs an entry for this domain name into the table.

Our controller is written in Python and runs locally on the P4 switch. Thus, the elapsed time between the time when our method sends the original DNS packet to the controller and the time when the controller installs an entry for it is very tiny. During such a tiny period, our method replicates the DNS packet and lets the replicated DNS packet recirculate the pipeline until it matches the entry installed by the controller.

As the controller installs more and more entries into the table, the number of entries in the table will grow and may reach the limit of the table. Our method currently uses the first-in-first-out (FIFO) replacement strategy to deal with the situation when the table becomes full. Other replacement strategies like least-recently-used (LRU) can be used as an alternative. Our method currently uses 65535 as the maximum number of entries for the table. We think that this number is large enough for an edge network. If needed, this number can be increased.

In our P4 program, we define two types of actions named ***Accept*** and ***Block***, respectively. Action ***Accept*** will not modify anything in the packet and it will forward the DNS packet normally like forwarding other packets. On the other hand, action ***Block*** will first modify the DNS request packet and transform it into a DNS reply packet with the answer IP address being 0.0.0.0. Then, the formed DNS reply packet will be sent back to the host that issued this DNS request (i.e., the tracker or ad downloader that issued this DNS request). The reason for fabricating a DNS reply packet with 0.0.0.0 and sending it back to the issuing host is that the issuing host will abandon its DNS name resolution after receiving such a DNS reply. Our method could simply drop the DNS request packet without sending back such a reply. However, doing so will cause the issuing host to keep sending more DNS requests until timeout. This will consume more network bandwidth and waste the CPU cycles of the user's device.

From Fig. 2, one can see that when a DNS packet enters the pipeline, our P4 program will hash its domain name and use the hash value to match the entries of the match-action table. If a match is made, our P4 program will perform the ***Block*** or ***Accept*** action based on the action of the matched entry. If no match is made (i.e., the hash value does not exist in the table yet), our P4 program will check whether the current packet is a replication of a previous packet and is recirculating. If so, this means that the controller is processing the original packet of this packet and thus we just recirculate the packet again. Otherwise, our P4 program will replicate the packet, send the replicated packet back to the pipeline to recirculate it, and forward the original packet to the controller.

In our design, to reduce the load of the controller, all information that the controller does not need is removed from the original packet and only the domain name field and our custom header remain. Then, depending on whether the domain name is on the filter list, the controller will install the hash value of the domain name with the corresponding action into the table. As for the recirculating packet, after the new entry is installed by the controller, it will match that entry and be processed by the action of that entry. With this design, after an entry for a specific domain name has been installed into the table, any future DNS packet carrying the same domain name will match this entry and need not be sent to the controller.

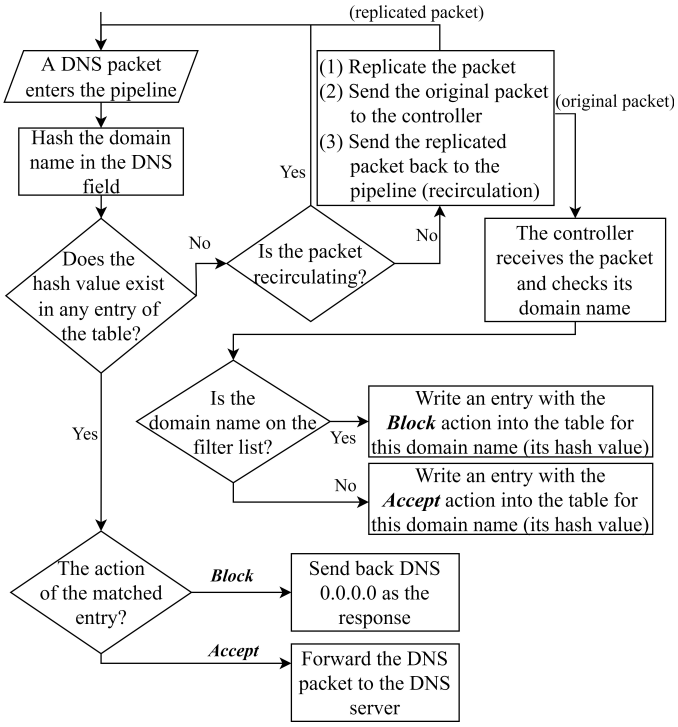

Fig. 2: The pipeline flowchart of our design

IV. Experimental Setup

Fig. 3 shows the network topology of our testbed. Our method runs inside the P4 switch, which connects the devices shown at the bottom to the Internet shown at the top. Each of these devices is equipped with a 3.20 GHz Intel i7-7800 CPU and runs Ubuntu 18.04 as its operating system. The length of each optical link connecting the switch with a device is 5 meters and the bandwidth of each link in each direction is 10 Gbps. The normal hosts are the devices on which we run a Chrome web browser to fetch web pages from the Internet. To compare the performance of our method and "Pi-hole," we selected one of the devices and installed the "Pi-hole" server on it.

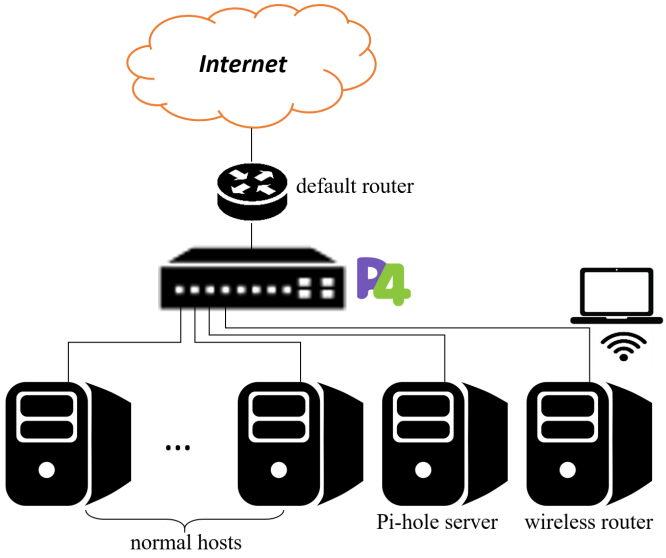

Fig. 3: The network topology of our testbed

To also compare the performance of our method and "Pi-hole" on a lossy wireless network, which may suffer from higher packet loss rates due to weak signal strength and packet collisions, we let the rightmost device act as a wireless router by using the NetworkManager [18] command to provide Wi-Fi accesses for mobile devices. Then, we run web browsers or different kinds of apps (e.g., Facebook and YouTube) on these mobile devices to fetch web pages from the Internet.

The default routers of these normal hosts and the wireless router are all set to the "default router" sitting between the P4 switch and the Internet. By this setting, the traffic between any of these devices and the Internet will pass through the P4 switch. In our method, since these normal hosts and the used mobile devices are all configured to use a public DNS server on the Internet (which is the Google's DNS server [19] using 8.8.8.8 as its IP address), the DNS packets issued by the trackers and ad downloaders running on them will be intercepted and processed by our method.

V. Performance Evaluation

A. Functionality Test

We activated our method and observed its blocking effects on many commercial websites in this test. To clearly show its effectiveness, we selected Yahoo's home page because it always displays many ads. Fig. 4 shows the difference between a web page with ads and without ads, respectively. This page was downloaded successfully without losing its original contents. The differences between the top and bottom pages are visible to naked eyes. Our method blocked the ads and thus caused the blank space at the top and bottom-right corner of the page. We checked the console log of the page and found that the failed HTTP requests gave the error message "ERR_CONNECTION_REFUSED" due to the failure to find the destination IP address. We conducted the same tests on different kinds of mobile devices (e.g., laptop, iPad, and iPhone, etc.) and used different kinds of apps running on them to browse many commercial websites. Experimental results confirmed that our method worked as expected and could block ads and trackers for all devices residing on an edge network.

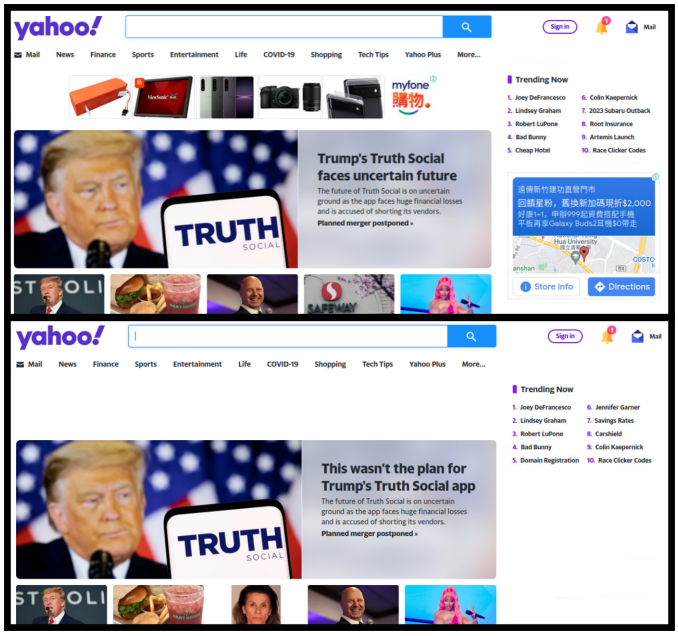

Fig. 4: The home page of Yahoo. The top image shows the original page (with ads). The bottom image shows the page (without ads) after activating our method.

We compared the ad-block effects of our method with those of "AdBlock Plus" [4] and "Pi-hole" and the results are summarized below. Using a similar DNS-blocking design, "Pi-hole" provided the same effects as our method on a page. Although our method and "Pi-hole" could block most ads on a page, some ads are still displayed on the page. We found that these ads purposely used some websites that are also used by other important services. Thus, web page downloads might fail if our method or "Pi-hole" blocked the domain names of these websites. In contrast, we found that "AdBlock Plus" can display a clearer web page. This is because it is a web browser extension and thus can see the contents of HTTP requests. As a result, it can block ads more precisely. However, as we explained previously, since "AdBlock Plus" is a web browser extension, it only works for web browsers and cannot work successfully for other kinds of apps such as Facebook and YouTube on mobile devices.

B. Performance Test

1) DNS packet processing: The nslookup [20] command is a common tool for looking up DNS records. Thus, we used it to generate DNS requests in our experiments and measured the average DNS response time under our method and "Pi-hole," respectively. In this test, we measured the execution time of 16,000 nslookup command invocations. A half of the domain names in these nslookup commands are selected from

the "Pi-hole" filter list [17] while the other half are domain names not on the filter list. Since the maximum number of entries for the P4 match-action table was set to 65535 in our method, which is larger than 16,000, the table was never full in our experiments.

We tested four blocking methods and compared their performance. The first method (denoted "None" in Fig. 5) was used as a baseline, in which no blocking method was used. In the second method (denoted "P4 ad blocker (First execution)"), we "cold start" our method without any entry pre-installed in the P4 switch's table. In the third method (denoted "P4 ad blocker"), we ran our method twice. That is, we ran our method the second time with the entries installed during the first run of our method. This can be said to "warm start" our method. The fourth method, which is denoted "Pi-hole," used "Pi-hole" as the blocking method. In this method, our method is disabled and the normal hosts and mobile devices shown in Fig. 3 set the IP address of their default DNS servers to the IP address of the "Pi-hole" server. When receiving a DNS packet with a domain name not on the filter list, the "P-hole" server will send it to the Google's DNS server to perform the lookup.

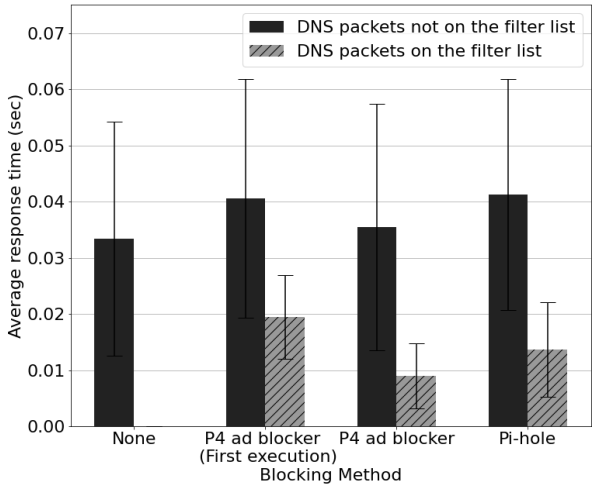

Fig. 5: The average DNS response time under different blocking methods

Fig. 5 shows the average DNS response time under these blocking methods. For each of these methods (except for the "None" method), one can see that the average response time of the DNS packets not on the filter list is longer than that of the DNS packets on the filter list. There are two reasons to explain these results. Firstly, the former DNS packets need to travel to a public DNS server on the Internet to complete their lookups; however, the latter DNS packets will be given a fake DNS response quickly by a local P4 switch in our method or by a local DNS server in "Pi-hole." Secondly, the former DNS packets need to be compared against all domain names on the filter list before they are forwarded to the Google DNS server. In contrast, on average the latter DNS packets only need to be compared to a half of the domain names on the filter list before a match is made.

One can see that for the DNS packets with a domain name on the filter list, "P4 ad blocker" (i.e., the "warm-start" P4 method) can reduce 0.01 seconds delay as compared to "P4 ad blocker (First execution)" (i.e., the "cold-start" P4 method). This improvement is due to the entries that were learned and installed during the first run of our method. Comparing "P4 ad blocker" with "Pi-hole," one can see that for the DNS packets with a domain name on the filter list, the average DNS response time of "P4 ad blocker" is shorter than that of "Pi-hole." This improvement again is due to the entries learned and installed in the P4 match-action table. In contrast, every time when the "Pi-hole" server receives a DNS packet, it needs to compare the domain name in the DNS packet to all of the domain names on the filter list. Thus, its average response time is higher than that of "P4 ad blocker."

For the DNS packets with a domain name not on the filter list, one can see that the average response time of "P4 ad blocker" is also shorter than that of "Pi-hole." This improvement is also due to the entries learned and installed in the P4 match-action table. Note that the entries learned and installed in the P4 match-action table are not only for the domain names on the filter list. Instead, if a domain name is found by the controller of our method to be not on the filter list, our controller will also install an entry for it with the ***Accept*** action. Thus, when a DNS packet with the same domain name enters the P4 switch in the future, it will be matched by this entry and forwarded to a public DNS server immediately. In contrast, every time when the "Pi-hole" server receives a DNS packet, even if the DNS packet contains a domain name that is not on the filter list and has been seen before, the "Pi-hole" server still needs to compare the domain name in the DNS packet to all of the domain names on the filter list, Due to this reason, for such DNS packets, their average response time in "Pi-hole" is higher than that in "P4 ad blocker."

One can see that the average response time of the DNS packets with a domain name on the filter list in "P4 ad blocker (First execution)" is higher than that in "Pi-hole." This is due to the communication latency between the dataplane of the P4 switch and the controller and the latency caused by the controller when installing entries into the match-action table. However, since our method is executed in a P4 hardware switch that connects an edge network to the Internet and normally a hardware switch will operate for a long period of time to provide reliable services to all devices on the network, we expect that our method will deliver high performance as more and more entries have been installed into the table.

2) Network traffic reduction: In this test, we compared how much ads traffic can be blocked by "P4 ad blocker" and "Pi-hole" respectively when compared against the "None" method, which does not block ads at all. We collected 34 commercial websites for this test and only those with more than 10 ads and trackers are shown in Table I to save space. We selected these websites due to their popularity and the large number of ads

and trackers used in their web pages. Most of these websites are news or shopping sites, which attract many people to visit them. The numbers of ads and trackers used on these websites are reported by "AdBlock Plus." To measure the download performance of these websites, we used the sitespeed.io [21] tool in our experiments. This tool can run locally on a device to generate many statistics about a web page download.

TABLE I: Some websites that are used in our experiments

Website	Ads and trackers
www.huffpost.com	10
www.reuters.com	10
utn.com	11
www.appledaily.com	11
www.ltn.com.tw	11
www.momoshop.com.tw	14
www.163.com	16
www.ettoday.net	19
tw.news.yahoo.com	20
www.msn.com	22
www.yahoo.com.tw	23
www.bloomberg.com	28
www.gamestar.de	32
www.sohu.com	34
www.dailymail.co.uk	190

In this test, we wrote a Python script and used the Selenium [22] package to control the operations of the Chrome web browser running on one normal host (shown in Fig. 3). Under the control of our script, the web browser will automatically browse the pages of specified websites. After starting loading a web page, we let the web browser wait at least 5 seconds on each page so that the trackers and ad downloaders embedded in it can be fully downloaded and have enough time to "do their jobs." We found that some websites might send HTTP requests to many third-party websites. Because some pages of these third-party websites might take too much time to download, we set 30 seconds as the maximum waiting time for each website. If within 30 seconds the download of the contents of a website (including all of its ads) cannot be finished, we say that this website is a "failed" website in our experiments. During each run of the experiment, our script let the web browser visit these 34 websites one by one and we used the tcpdump [23] tool to capture all packets transferred from these websites to the web browser. Then, we measured the amount of transferred traffic (in the units of packets and bytes) under the "None," "P4 ad blocker," and "Pi-hole" methods, respectively.

We carried out 20 runs of this test and computed the averages of the numbers of packets and bytes transferred in these runs. Fig. 6 shows the average number of packets transferred in the "None," "P4 ad blocker," and "Pi-hole" methods, respectively. The black bars (denoted by "All protocols") show the average numbers of all of the transferred packets regardless of their transport-layer protocol types. The gray bars instead show the average numbers of TCP-only packets. Traditionally, most websites use the application-layer HTTP(s) 1.1 [24] protocol to transfer data and HTTP(s) 1.1 uses TCP as its default transport-layer protocol. However, recently more and more websites have started to use the QUIC [25] protocol and HTTP/2 [26] to transfer data such as GIF images or videos. Since QUIC uses UDP as its transport-layer protocol and several large companies such as Google and Facebook are using QUIC with HTTP/2 for their websites, nowadays not only TCP is used to transfer web traffic, but UDP is also used. Therefore, in the figures we separate the TCP-only traffic from the "All protocols" traffic, which includes both TCP and UDP traffic, to gain insights into the results.

From this figure, one can see that because "P4 ad blocker" and "Pi-hole" both block ads by blocking the DNS packets issued by trackers and ad downloaders, their performances are about the same. Another finding is that enabling either "P4 ad blocker" or "Pi-hole" can reduce the number of "All protocols" packets by 40.92% and reduce the number of TCP-only packets by 40.33%. These savings are quite large.

Fig. 7 instead compares the average number of "All protocols" bytes and TCP-only bytes under these methods. As shown in Fig. 6, "P4 ad blocker" and "Pi-hole" have about the same performance. One can see that after enabling either "P4 ad blocker" or "Pi-hole," the number of "All protocols" bytes can be reduced by 20.73% while the number of TCP-only bytes can be reduced by 20.55%. These savings are quite important when the available bandwidth between a device and its accessed website is small.

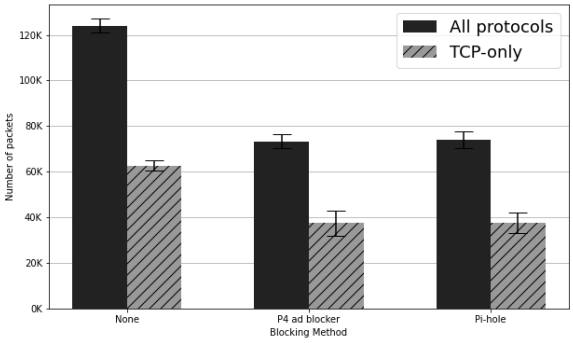

Fig. 6: The average number of "All protocols" packets and TCP-only packets transferred during visiting 34 websites

3) Web page download under different packet loss rates: An edge network can have many wireless routers (which function as layer-3 routers) or wireless access points (which function as layer-2 switches) to provide Wi-Fi accesses to mobile devices. The provided Wi-Fi wireless networks may suffer from high packet loss rates due to weak signal strength or packet collisions. Such problems can easily happen when a device is far away from the wireless router/access point or does not have a line-of-sight communication path between it and the wireless router/access point. To evaluate the performances of "P4 ad blocker" and "Pi-hole" on such an unreliable wireless network, we used the tc [27] command to configure the packet loss rates of the links that connect the normal hosts, "Pi-hole" server, and wireless router to the P4 switch (shown in Fig. 3). In our experiments, we used 0%, 5%, and 10% packet loss

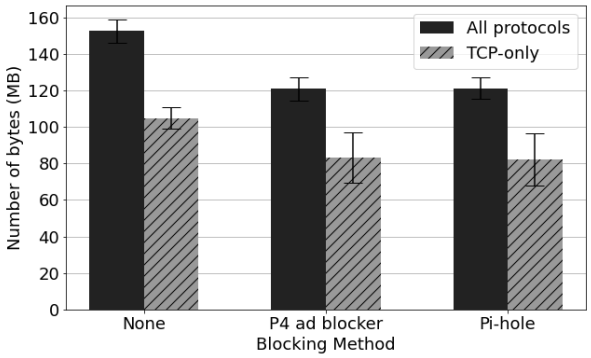

Fig. 7: The average number of "All protocols" bytes and TCP-only bytes transferred during visiting 34 websites

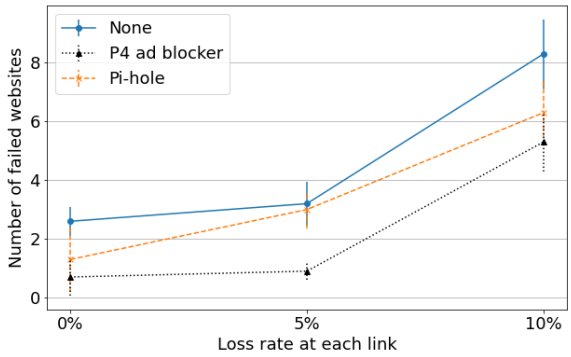

Fig. 8: The number of visited websites that the web browser failed to download completely within 30 seconds

rates respectively for each direction of these links to simulate different packet loss rates on a wireless network.

The test performed here was the same as that performed in Section V-B2. The web browser running on one normal host visited all of the 34 websites one by one. If it could not finish downloading all of the contents of a website in 30 seconds, we call such a website a "failed" website.

Fig. 8 shows the number of failed websites in different blocking methods under different packet loss rates. As can be seen, under all of these tested packet loss rates, the number of failed websites in "P4 ad blocker" is smaller than that in "Pi-hole." This is because when a tracker or ad downloader issues a DNS request packet, regardless of whether the domain name in the DNS packet is on the filter list or not, the number of lossy links that the DNS request and reply packets need to traverse in "Pi-hole" is higher than that in "P4 ad blocker." Thus, the chance that the DNS request/reply procedure may fail in "Pi-hole" is higher than that in "P4 ad blocker." Although after a timeout period, the DNS request packet will be resent by the tracker or ad downloader, if a web page has many trackers and ad downloaders, the chance that the web browser cannot finish downloading all of the contents of the web page (including ads) will be higher in "Pi-hole." The results in Fig. 8 confirm the explanations.

Fig. 9 shows the execution time required to visit all of these 34 websites under different packet loss rates in these blocking methods. One can see that the execution time of "P4 ad blocker" is shorter than that of "Pi-hole" and the differences are about 50 seconds under the packet loss rates of 5% and 10%, respectively. The reason why "P4 ad blocker" outperforms 'Pi-hole" can be explained by the above reasons as well.

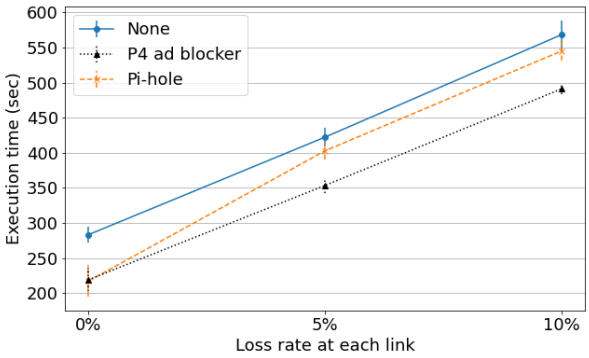

Fig. 9: The execution time required to visit all of the tested 34 websites

VI. Conclusion

Nowadays, collaborating trackers are causing serious threats to user privacy. They work underground and most users do not see them. As for web advertisements, they delay web page downloads and displays and consume more network bandwidth than needed. In this paper, we design and implement a P4 hardware switch-based method to block trackers and advertisements for all devices residing on an edge network. We compared the performance of our method named "P4 ad blocker" with that of other methods named "None" and "Pi-hole," respectively. When compared with the "None" method, which does not block any tracker or advertisement, our method can save the network bandwidth by about 20%. When compared with "Pi-hole," which is a software-based method using a similar DNS-blocking design, our method can reduce the DNS processing time and the execution time required to download web pages over lossy links.

Although "Pi-hole" can be easily deployed by advertising the "Pi-hole" server as the default DNS server for all devices on an edge network to provide "network-wide" protection, actually such a method poses serious security threats to all devices on an edge network as they can be easily attacked by the common DNS Spoofing and DNS Flooding attacks. This is very possible as the "Pi-hole" server is an open-source software that can be easily implanted into malicious code. In contrast, since our method is executed inside a hardware switch, which needs to be deployed and managed by the network manager, our method is a more secure solution.

REFERENCES

[1] Programming Protocol-independent Packet Processors (P4), available : https://p4.org/

[2] J. R. Mayer and J. C. Mitchell, "Third-Party Web Tracking: Policy and Technology," 2012 IEEE Symposium on Security and Privacy, May 20-23, 2012, San Francisco, USA

[3] Iskander Sanchez-Rola, Xabier Ugarte-Pedrero, Igor Santos, and Pablo G. Bringas, "The Web Is Watching You: A Comprehensive Review of Web-tracking Techniques and Countermeasures," Logic Journal of the IGPL, Volume 25, Issue 1, February 2017, Pages 18–29

[4] AdBlock Plus, available : http://adblockplus.org/

[5] Adguard, available : http://adguard.com/

[6] Muhammad Muzamil, Akmal Khan, Shabir Hussain, M.Zeeshan Jhandir, and Rafaqat Kazmi, "Analysis of Tracker-Blockers Performance," Vol 4 No 1 (2021): Pakistan Journal of Engineering and Technology

[7] C. E. Wills and D. C. Uzunoglu, "What Ad Blockers Are (and Are Not) Doing," 2016 Fourth IEEE Workshop on Hot Topics in Web Systems and Technologies (HotWeb), 24-25 October 2016, Washington, DC, USA

[8] Parmar A, Dedegikas C, and Toms M, Dickert C (2015) "Adblock Plus Efficacy Study." Technical Report, Simon Fraser University. available : http://www.sfu.ca/content/dam/sfu/snfchs/pdfs/Adblock.Plus.Study.pdf.

[9] J. Huang and W. Cheng, "Filtering Performance Analysis and Application Study of Advertising Filtering Tools," 2017 12th International Conference on Intelligent Systems and Knowledge Engineering (ISKE), 24-26 November 2017, Nanjing, China

[10] I. Castell-Uroz, J. Solé-Pareta, and P. Barlet-Ros, "Demystifying Content-blockers: A Large-scale Study of Actual Performance Gains," 2020 16th International Conference on Network and Service Management (CNSM), 02-06 November 2020, Izmir, Turkey

[11] EasyList - Overview, available : https://easylist.to/

[12] H. P. Jason Bau, Jonathan Mayer, and J. C. Mitchell, "A Promising Direction for Web Tracking Countermeasures," In Proceedings of Web 2.0 Security and Privacy (W2SP). IEEE Computer Society, 2013.

[13] U. Iqbal, P. Snyder, S. Zhu, B. Livshits, Z. Qian, and Z. Shafiq, "AdGraph: A Graph-Based Approach to Ad and Tracker Blocking," 2020 IEEE Symposium on Security and Privacy (SP), 18-21 May 2020, San Francisco, USA

[14] T. Vo and C. Jaiswal, "Adremover: The Improved Machine Learning Approach for Blocking Ads," 2019 IEEE 10th Annual Ubiquitous Computing, Electronics and Mobile Communication Conference (UEMCON), 10-12 October 2019, New York, USA

[15] Pi-hole: Network-wide Protection, available : https://pi-hole.net/

[16] Inventec D10056 Datacenter Switch, available : https://productline.inventec.com/Switch/Download/D10056.pdf

[17] Pi-hole block list, available : https://raw.githubusercontent.com/ StevenBlack/hosts/master/hosts

[18] Network Manger, available : https://ubuntu.com/core/docs/networkmanager

[19] Google Public DNS, available : https://developers.google.com/speed/publicdns

[20] nslookup(1) - Linux man page, available : https://linux.die.net/man/1/nslookup

[21] Sitespeed.io, available : https://www.sitespeed.io/

[22] Selenium with Python, available : https://selenium-python.readthedocs.io/

[23] TCPDUMP & LIBCAP, available : https://www.tcpdump.org/

[24] Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing, Request for Comments: 7230, Internet Engineering Task Force (IETF) .

[25] QUIC: A UDP-Based Multiplexed and Secure Transport, Request for Comments: 9000, Internet Engineering Task Force (IETF) .

[26] Hypertext Transfer Protocol Version 2 (HTTP/2), Request for Comments: 9113, Internet Engineering Task Force (IETF) .

[27] tc(8) - Linux man page, available : https://man7.org/linux/man-pages/man8/tc.8.html