Implementation of Content-oriented Networking Architecture (CONA): A Focus on DDoS Countermeasure

Junho Suh*, Hoon-gyu Choi*, Wonjun Yoon*, Taewan You*, Ted "Taekyoung" Kwon**, Yanghee Choi**

School of Computer Science and Engineering
Seoul National University
Seoul, Korea
{jhsuh, hgchoi, wjyoon, twyou}@mmlab.snu.ac.kr*, {tkkwon, yhchoi}@snu.ac.kr**

ABSTRACT

In this paper we introduce a Content-oriented Networking Architecture (CONA) and describe its implementation on the NetFPGA-OpenFlow platform. CONA seeks to substantiate a content-centric communication model to address accountability. In CONA, an access router is extended to figure out what contents are requested, and hence is called an agent. A CONA agent receives a content request from an attached host and delivers the requested content. In this way, CONA achieves the accountability and can take a countermeasure against resource-exhaustive attacks like DDoS. In the CONA implementation, we seek to support legacy hosts by redirecting HTTP requests (to the agent), capturing HTTP GET messages, and extracting content names. We implement CONA on a testbed consisting of NetFPGA-OpenFlow platforms. We also carry out experiments to illustrate a scenario in which CONA limits the behavior of a malicious host that generates a resource-exhaustive attack.

1. INTRODUCTION

Over the 40 years, the current TCP/IP Internet architecture has been evolving. However, there is a consensus that we need to redesign a new network architecture to address many technical issues of the Internet such as security, mobility, routing scalability, accountability (e.g. [1, 2]). The most controversial part of the original TCP/IP architecture is the end-to-end communication principle, which is focused on the endpoints. Recently, a novel approach to revolutionize the networking paradigm is gaining momentum, so called content-centric or data-oriented networking (e.g. [3, 4, 5]). The rationale behind the content-oriented networking approach is that users are oblivious to where the contents are located. That is, when a user downloads files from peer-to-peer or CDNs, the address of the endpoint (or the server) does not matter; the user is concerned with the content only.

We believe the content-oriented networking approach can be exploited to bolster the security from the following reasons. First, the current Internet service allows any host to send any packets to any destination. This restriction-free connectivity prevents an Internet service provider (ISP) from effectively blocking the malicious packets (e.g. DDos). Second, it is not easy to check the trustworthiness of the content that we download. For instance, by subverting a DNS entry, an HTTP GET request is forwarded to a wrong site, which leads to phishing and pharming. Third, when a flash crowd to a popular content takes place, the corresponding server may crash or respond very slowly. The main culprit of all the above problems is that the network forwards the packets of a content without knowing the context of the content. In this paper, we take a content-oriented approach to address the security and accountability issues in the current Internet. We first propose a content-oriented networking architecture (CONA) in which a user (or its host) requests a content file from its ISP, and the ISP returns the requested content file. Then we discuss how CONA can solve or mitigate the security and accountability problems. Finally, we will explain how to design and implement CONA by using the NetFPGA prototype.

2. CONTENT-ORIENTED NETWORKING ARCHITECTURE (CONA)

2.1 Assumptions

2.1.1 Host Identifier

A host does not necessarily have its own public IP address. It just needs to communicate with its access router. So the MAC address of the host or a private address assigned to the host will suffice in CONA. The access router will manage the hosts attached to its subnet. Henceforth, an access router is called “an agent.” The reason is that an agent will solicit the contents on behalf of its hosts instead of blindly forwarding IP packets from the hosts. An agent has a globally-routable (or public) IP address on its egress link. If a host is a publisher or a server with a content file (e.g. web page) to publish, it may have a domain name. Then its agent’s public IP address is registered with the publisher’s domain name in a mapping infrastructure (i.e. the DNS). To summarize, the endpoints like hosts and servers cannot directly communicate with each other.

2.1.2 Content Identifiers

A content is named (or specified) by an HTTP URI[1]. Its publisher (actually, the agent of the publisher) can be located by sending a DNS query[2]. Then the DNS will reply with the corresponding entry that contains the public IP address of the agent to which the publisher is attached. If the publisher wishes to ensure the authenticity of the content file, she can attach her digital signature and her certificate.

2.1.3 Internet connectivity

When a host first tries to set up Internet connectivity, it first receives an agent advertisement message (similar to a router advertisement message) from an agent. An agent advertisement message may include the domain name of the agent and its certificate as well as the network configuration information. Hence, a host can verify the identities (and their authenticity) of agents[3]. To deliver content files, we assume that the host can configure some locally unique IP address for communications with the agent (e.g. using DHCP), which could be a private address in IPv4 or a link local address in IPv6. After the host and its agent establishes its association, the host can request a content file from the agent.

2.2 CONA operations

2.2.1 User-ISP interaction

After the user's host sets up local connectivity with the agent, the host will send an HTTP GET message to the agent to request a content. The agent will first check its cache to find out whether the requested content is stored locally. (In some sense, it also serves as a web cache.) If not, it will contact the DNS to find out the public IP address of the agent of the publisher. The host's agent will contact the the publisher's agent to download the content. The publisher's agent will get the content from the publisher. At this moment, let us assume both agents can communicate with the current TCP/IP architecture.

When the agent (of the host) finishes downloading the content, it forwards the content to the host[4]. Depending on its policy, it may cache the content or not. In this way, if there is a flash crowd for a particular HTTP URI, the agent of the requesting host can directly return the content efficiently.

2.2.2 ISP-ISP interaction

Let us explain how CONA works when the agent of the content requesting host and the agent of the publisher belong to different ISPs. Now the border routers that connect the two ISPs are called gateways due to their CONA-related functionalities other than packet forwarding. As similar to User-ISP interactions, the gateways will deliver the contents between each other.

[1]As more and more internet traffic is delivered over HTTP [6], we consider web contents only in this study.

[2]Note that an agent will send the DNS query on behalf of its host; in this way, a host cannot contact the DNS infrastructure, which helps fortify the security against DNS poisoning.

[3]Especially in wireless environments, there can be multiple agents.

[4]To expedite the content delivery to the host, the host's agent can relay the packets from the publisher's agent while downloading from the publisher's agent.

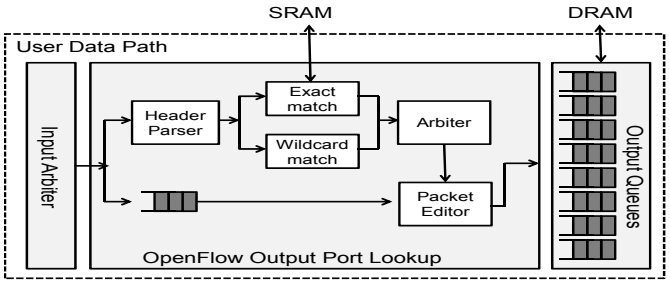

Figure 1: A reference OpenFlow switch pipeline model

On receipt of the content request for a publisher outside the ISP, the agent of a host can forward the content request message to the border router by current IP routing. Or it may look up to a routing server. In this paper, we take the latter approach. Then the routing server (e.g. a BGP router with some extension) looks at the IP address of the publisher's agent, and returns the the IP address of the gateway within the ISP toward the publisher[5].

Let us denote the gateway selected by the routing server in the given ISP and the next hop gateway in the neighbor ISP by G_s and G_n, respectively. So G_s will send the content request message to G_n. If the publisher belongs to the same ISP as G_n, G_n will contact the agent of the publisher to download the content, which in turn is relayed to G_s. If the publisher does not belong to G_n's ISP, G_s will contact its own routing server and perform the same procedure.

There will be a lot of contents downloaded in parallel over the link between two gateways (of the adjacent ISPs). For the purposes of fast processing, we propose to prepend a short-term flow label (similar to that of MPLS) in front of the IP packets. There is a one-to-one correspondence between the content (or its flow) and the label. In this way, each ISP can keep track of which contents are delivered and how large is each content and so forth.

Like agents, gateways can also store the contents by their own policy[6]. Thus, if the content request message results in a cache hit, the gateway can directly send the content without further forwarding the request.

3. CONA IMPLEMENTATION

To implement and test CONA, we adopt the OpenFlow architecture to manage flows in the testbed. OpenFlow takes a minimalist approach to control flows by managing the flow tables of the network entities (e.g. switches). This OpenFlow's principle of flow management fits well with CONA in the sense that an agent in CONA will establish a flow when it receives a content request from a host.

Also, we implement CONA using NetFPGA hardware platforms to expedite packet forwarding. A NetFPGA platform is characterized by three parts: (1) the Xilinx Virtex-II FPGA with 4 Ethernet cards and memories, (2) the Gate-

[5]This functionality is somewhat similar to the NOX server in OpenFlow platform. If there are multiple candidate neighbor ISPs for the publisher, some policy (similar to the BGP policy) is needed to select a gateway connected to the best neighbor ISP, which is out of the scope.

[6]We believe that an ISP can orchestrate the caching policy over its agents and gateways (e.g. [10]).

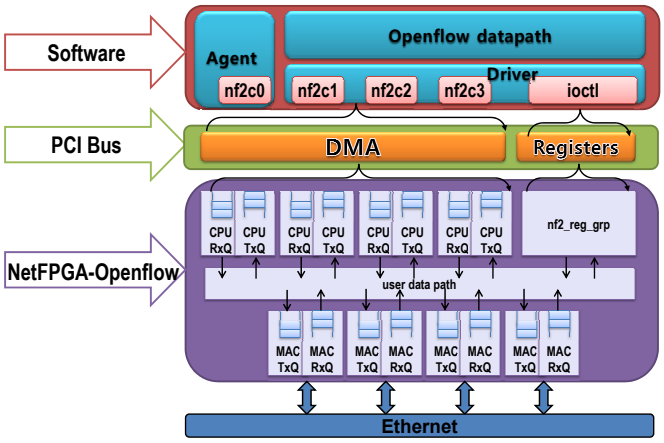

Figure 2: Architecture of the CONA agent

ware that deals with packet processing (e.g. to which output port to forward a packet), and (3) the software including a controller package that populates the hardware forwarding table (in the NetFPGA board) that mirrors the Linux's routing table in user space. These modules allow us to build a full line rate router/switch (i.e. 4 X 1Gbps Ethernet forwarding).

3.1 NetFPGA-OpenFlow Platform

Figure 1 [15] shows the reference design (also called the "reference pipeline" due to parallelism in packet forwarding) of a NetFPGA platform that supports the OpenFlow architecture, which we call a *NetFPGA-OpenFlow* platform henceforth. The key functionality is to which port an incoming packet to forward, which is the job of the Output_Port_Lookup part in the reference design. It contains the matching module (i.e., exact_matching and wildcard_matching) to match the corresponding flow entry by the control information of an incoming packet (i.e., MAC addresses, IP addresses, port numbers).

The NetFPGA-Openflow platform is controlled by the OpenFlow management software, which consists of two parts: a user program and a kernel module. The user program communicates with the OpenFlow controller and processes the Openflow protocol messages. The kernel module keeps track of the flow tables, processes the packets (if needed), and updates the statistics.

To implement the CONA agent, we add a hardware component in the NetFPGA-Openflow platform by Verilog programming. Also we develop a software component which interacts with both the NetFPGA-Openflow platform and the Openflow management software. The hardware component consists of (1) a URL interceptor to capture the packets containing the content name, and (2) a flow limiter to block or mitigate the DDoS attack. The software component monitors the content request traffic which arrives at the agent. Also it blocks the DDoS attack when it receives the DDoS attack notification from the OpenFlow controller. The overall architecture of the agent is shown in Figure 2.

3.2 Hardware Implementation

To insert both the URL interceptor and the rate limiter into the reference design in the NetFPGA-OpenFlow platform, we slightly modified both the Output_Port_Lookup module in such a way that the URL-extractor and the rate limiter operate compatible with the NetFPGA-OpenFlow switch.

3.2.1 *URL interceptor*

The URL interceptor is a preprocessing module that identifies the packets containing the content name (e.g. HTTP URI). Then it intercepts and forwards them to the agent software.

To extract what content is requested (i.e., the content name) from the received packets, it is necessary to parse the packets' header. In the NetFPGA-OpenFlow switch, all packets pass through the Header_Parser in Figure 1. The Header_Parser module looks at the 5-tuples (i.e. IP addresses, port numbers, protocol) of a packet header and concatenates them as *the flow information* in order to match against the flow table inside the hardware. In this way, we can identify application-level protocol (i.e., the port number of DNS or HTTP) of the packet. Therefore, a packet corresponding to the content request (e.g. web page) is intercepted by this module. Then the packet is directed to the agent software via a dedicated interface (i.e., nf2c0 in Figure 2). On the other hand, all the other normal packets pass through the normal pipeline of the NetFPGA-Openflow switch.

3.2.2 *Flow limiter*

The flow limiter is the module to block/mitigate the DDoS attack flows when the switch receives the DDoS alert message from the controller. On receipt of the message, the switch limits the rate of content request messages to a victim server. The NetFPGA base package provides the rate limiter module. Although the module blocks a dedicated interface (e.g., nf2cx), we believe that this provides similar functionality of what we expect. Later, we will implement the flow limiter to distinguish the flows passing through the same interface.

3.3 Software

The CONA agent software extends the URL-Extractor software implementation in [13]. This extracts 10-tuple (i.e., the flow information) from the IP packet delivered through the particular interface (i.e., nf2c0) by using a raw socket. One difference from the original URL-Extractor software [13] is that the CONA agent interacts with the Openflow management software by using the dpctl utility[7].

To facilitate the URL extraction process while supporting legacy hosts (that are not CONA-compliant), we use a trick to make the agent respond to DNS queries instead of the DNS server. Before a user sends a content request (e.g., HTTP GET message), she sends a DNS query for the domain name of the corresponding content server. The DNS query is captured by the CONA agent, who sends the DNS reply with its own IP address on behalf of the DNS server. In this way, the user will send the content request to the agent. Note that this mechanism is to support non CONA-compliant legacy hosts, who think of the agent as merely a router. CONA-compliant hosts will send content-request messages to its agent and hence the DNS redirect is not

[7]The dpctl is an administrative utility that monitors and configures the Openflow datapath, and updates the routing table entries. It is used to add, delete, modify, and monitor the datapaths.

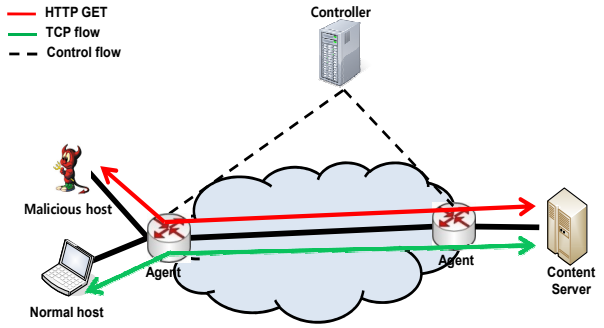

Figure 3: The network setting to test the CONA countermeasure for a DDoS attack. While a legitimate normal user sends a content request message and downloads a file over TCP, the attack host sends the content request messages to the content server. To adjust the rate of content request messages effectively, we use UDP for sending the content request messages.

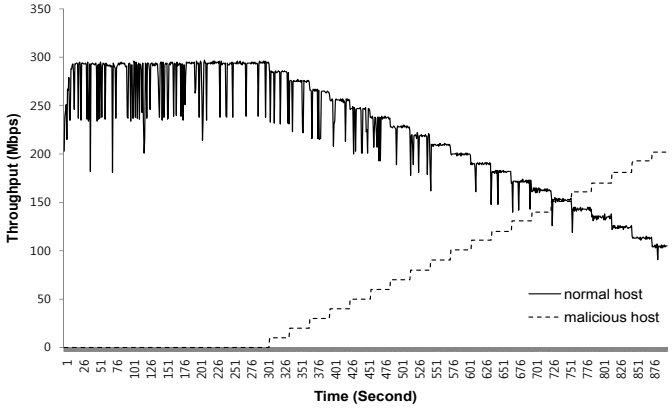

Figure 4: How a victim server reacts when there is a gradual increase of content request messages from a malicious host while a normal hosts is downloading a large file.

needed.

The agent keeps track of all the incoming (and outgoing) content requests (i.e., the request rate for each content server), and monitors whether a particular server is under a DDoS attack. When the agent detects the onslaught of the DDoS attack (say, the incoming content request rate for a server is higher than a threshold), it notifies the controller of the incident. The controller will alert other agents to block/reduce the content requests to the victim server.

4. EVALUATION

In this section we describe and evaluate the prototype of the CONA agent implementation to substantiate a DDoS countermeasure on the testbed made up of NetFPGA-OpenFlow switches. The agents will react to DDoS attacks in a cooperative fashion by regulating the content request rate toward a victim server. Recall that, for the purpose of accountability, each agent keep track of what contents are requested (over the incoming link) and delivered (over the outgoing link).

4.1 Testbed Configuration

We set up a testbed whose topology has two CONA agents: one for two hosts and the other for the publisher (or the server). Two desktop computers are attached to the agent on the left in Figure 3, while the server is attached to the agent on the right in Figure 3.

In the testbed in Figure 3, we will illustrate a DDoS attack scenario and show how CONA can take a reactive action. The agent in the left has two hosts; so it aggregates all the content requests from the two hosts toward the publisher. The publisher of the content running a HTTP web server has a high-definition video content.

4.2 A DDoS Attack Scenario

Suppose a large number of bots across multiple ISPs are activated to launch a DDoS attack to a particular publisher. If every agent allows the request of contents of well known services only, bots might not be activated. However, we conservatively assume that there are vulnerable points in distributed environments; for example, commands to bots can still be spread by mimicking a legitimate procedure in CONA.

Once the DDoS attack starts, the agent of the publisher will first detect the storming incoming requests toward the target publisher because the content requests messages of the DDoS attack traffic arrives at the point. We assume that the requested contents are all different and hence content caching is not effective in the scenario. The agent concludes the DDoS attack if the arrival rate of content requests for a particular server exceeds a threshold or if the content request pattern is not normal (e.g. see [16]). On the conclusion of a DDoS attack, the server's agent notifies the controller that which server is under DDoS attack. On receipt of the DDoS notification, the controller performs two tasks: (1) it starts checking incoming flows onto the attacked publisher from the other agents in the testbed[8], and (2) it sends a rate limit message to each of the relevant agents (who relay malicious content requests) to limit the content request rate onto the publisher[9]. This rate limiting in terms of the content request rate can be easily checked at the agent of the publisher and propagated across ISP's boundaries in a daisy-chain manner. In this way, the DDoS attack traffic will be contained by a backpressure mechanism.

To emulate the DDoS attack scenario, the normal host requests and receives a large file (700 Mbytes) over TCP by using iperf. Since TCP has the congestion control mechanism, its bandwidth approaches the maximum bandwidth of the link pipe. Meanwhile, the attacker can start the DDoS attack by sending content request messages (emulated by HTTP GET messages here) at a high rate.

[8]In real ISP's network case, the controller will check the incoming content requests from the adjacent ISPs and its own agents.

[9]As for (2), we can think of a passive approach (e.g. [7]) that simply discards the DDoS attack traffic locally (at the agent of the victim or some local clearing center [9]) without explicit signaling; however, we believe that some active approach (i.e. sending rate limit messages) across multiple ISPs can effectively solve these resource-exhaustive attacks.

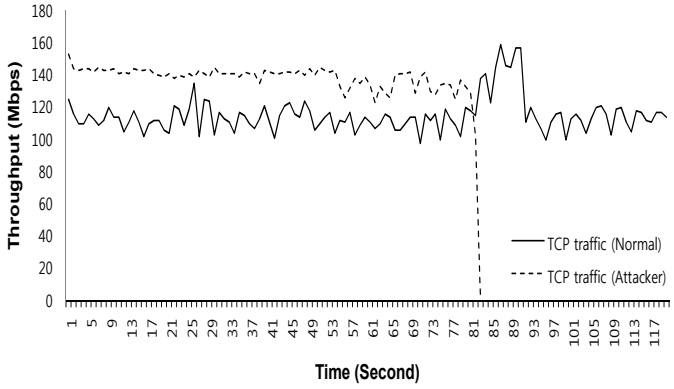

Figure 5: This experiment shows how the rate limiting mechanism in CONA reacts to a resource-exhaustive attack scenario with enabling the rate limiter. Here, the throughput of a malicious host is limited to zero.

4.3 Experimental Results

In this section we carry out two kinds of experiments: (i) to see how the server behaves with the mix of legitimate and malicious incoming content requests and (ii) to validate the effectiveness of the rate limiting mechanism when a DDoS attack is activated.

Figure 4 shows the throughput result when there is no rate limiting mechanism (e.g. no counter measure for DDoS attacks). A normal legitimate host is emulated by receiving a large video file. While the normal host keeps downloading the video file at almost 300 Mbps throughput, a malicious host gradually increase the content request rate onto the publisher starting at 300 seconds. Here, each content request from the malicious host makes the server deliver the requested small file; the average of the files requested by the malicious hosts is about 100Kbytes. We can observe that the resource-exhaustive attack from the malicious host affects the throughput of the normal host, which is decreased as the rate of the content requests (HTTP GET messages per second) from the malicious host is increased.

Figure 5 shows how the throughput of the malicious host is blocked by the rate limiter. Until 80 seconds, two hosts keep receiving video streaming data from the content server. At 80 seconds, however, a malicious host generates a surge of content request messages onto the server. And all the traffic is blocked by the agent of the malicious host because all the flows from the malicious host are classified into the resource-exhaustive attack by the agent of the publisher. Even if the normal host is attached to the same agent as the malicious host, the throughput of the normal host is hardly affected. We believe the rate limiting mechanism illustrated in this experiment can be a groundwork to substantiate a more sophisticated countermeasure for DDoS attacks.

5. CONCLUSION

In this paper, we proposed a novel content-oriented networking architecture (CONA) in which hosts request contents and its agent deliver the requested contents. In CONA, a host cannot send any arbitrary packets to any destination, which enhances the security. Moreover, the agent can keep track of which contents are requested by which hosts, which achieves the accountability. Due to the accountability and content-aware supervision, CONA can react to resource-exhaustive attacks like DDoS effectively. We implement CONA on a testbed that consists of NetFPGA switches with OpenFlow modules. We carry out experiments to demonstrate how CONA can take a countermeasure on a malicious host who generates a resource-exhaustive attack. We also discuss how CONA supports the legacy hosts by redirecting and capturing HTTP messages.

6. REFERENCES

[1] NSF FIND Project, http://www.nets-find.net/
[2] EU FIRE Project, http://cordis.europa.eu/fp7/ict/fire/
[3] Teemu Koponen et al., "A Data-Oriented (and Beyond) Network Architecture," ACM SIGCOMM 2007.
[4] Van Jacobson et al., "Networking named content," ACM CONEXT 2009.
[5] Sanjoy Paul et al., "The Cache-And-Forward Network Architecture for Efficient Mobile Content Delivery Services in the Future Internet," Proceedings of the First ITU-T Kaleidoscope Academic Conference on Innovations in NGN: Future Network and Services, 2008
[6] Craig Labovitz et al., NANOG 47, "2009 Internet Observatory Report," Oct. 2009.
[7] Zhenhai Duan et al., "Constructing Inter-Domain Packet Filters to Control IP Spoofing Based on BGP Updates," IEEE INFOCOM 2006.
[8] John Kubiatowicz et al., "OceanStore: An Architecture for Global-Scale Persistent Storage," ASPLOS 2000.
[9] Sharad Agarwal et al., "DDoS Mitigation via Regional Cleaning Centers," Sprint Technical Report RR04-ATL-013177, 2004.
[10] Jeffrey Erman et al., "Network-Aware Forward Caching," WWW 2009.
[11] Diana Smetters and Van Jacobson, "Securing Network Content," PARC Technical Report, 2009.
[12] Natasha Gude et al., "NOX: Towards an Opearting System for Networks," ACM CCR, July 2008.
[13] Michael Ciesla et al., "URL Extraction on the NetFPGA Reference Router," NetFPGA Developers Workshop, 2009.
[14] Ericsson Research, Project OpenFlow-MPLS, http://www.openflowswitch.org/wk/index.php/OpenFlowMPLS
[15] Jad Naous et al., "Implementing an OpenFlow Switch on the NetFPGA platform," ANCS'08, San Jose, CA, USA, November 6-7, 2008.
[16] Yi Zie and Shun-Zheng Yu, "Monitoring the application-layer DDoS attacks for popular websites," ACM/IEEE ToN, Vol. 17, No. 1, 2009.