DEFINE: A Service-Oriented Dynamically Enabling Function Model

Wei-Yi Tan [1,a], Zhi-Gang Sun [1], Wei Quan [1]

[1] *National University of Defense Technology, Changsha, China*
[a] *tanweiyi15@163.com*

Abstract. With the rapid expansion of network scale and the continuous evolution of network applications, the Internet becomes much more open and uncontrollable, and inevitably faces a variety of security threats. In order to satisfy the requirements of the current Internet security and transmission performance, the general solution is increasing specific devices (e.g., firewalls, network monitors) to detect and defend against attacks. However, these devices typically use a dedicated hardware-based or software-based architecture which is closed, leading to limited scalability, management complexity. And deploying a new application needs to develop new equipment, which take a long development cycle and costs a lot.

In this paper, we introduce an innovative Dynamically Enable Function In Network Equipment (DEFINE) to allow tenant get the network service quickly. First, DEFINE decouples an application into different functional components, and connects these function components in a reconfigurable method. Second, DEFINE provides a programmable interface to the third party, who can develop their own processing modules according to their own needs. To verify the effectiveness of this model, we set up an evaluating network with a FPGA-based OpenFlow switch prototype, and deployed several applications on it. Our results show that DEFINE has excellent flexibility and performance.

1 Introduction

With the rapid expansion of network scale and the emergence of new network technologies, network traffic and the number of users growing exponentially, the Internet faces much more challenges on security, scalability, robustness, and mobility [3]. In order to overcome these difficulties, more and more specific middleboxes [7] (e.g., firewall, network monitor, DPI detector) have been added in the network to guarantee the performance and resist the external attacks.

While these specific network devices guarantee high performance, they also bring some problems, such as limited flexibility caused by a closed architecture, and complicated to manage due to the differences in device standards between the vendors. Furthermore, it is hard to exchange information (e.g., switch states) between devices from distinct vectors, leading to difficulties in deploying network services from a network-wide vision. A typical solution is deploying additional equipment in the network to meet new needs, which costs a lot. For example, an enterprise usually installs various devices to meet network service (e.g., QoS) or security needs from different departments, which is not only complicated to configuration and management, but also expensive to deploy so many devices. This limitation becomes much more serious in a cloud network, as a cloud has much more virtual networks requiring a logic-dependent set of devices for each virtual network.

From the recent trend of network technology, the future network will be much more flexible, scalable, and easy to manage [2]. An ideal network will be that network functions/services can be deployed on any device and removed at any time without deploying or designing new hardware/software-based devices [1], which greatly reduces the cost of network operation and maintenance, and improve the flexibility of the network at the same time.

DEFINE has two important features. The first one is service-oriented, that is, a service-oriented network device is considered as a "services/functions pool", i.e., it integrates many services/functions the user wanted. And, only a simple instruction from a high-level controller can start any one service on this device, which greatly reduces the time to deploy new services/ functions, such as firewall, QoS, load balance (LB) etc.

Another feature is dynamic assignment. Dynamic assignment means that the remote controller can dynamically load new functions transferred from a center code library, a bit like the app store, or terminates any running functions according to the change of the user's demand or exception, such as memory overflow. Therefore, DEFINE not only enables the device perform

any functions pre-stored on network device (service pool), but also any other specific functions (such as deep packet detection, TCP handshake agent) developed by third part by a remote transferring function.

Furthermore, DEFINE also provides the controller interface to read/write state in corresponding modules in data-path pipeline. DEFINE offloads some common packet processing functions on the hardware data path to accelerate network functions.

In summary, the contributions of this paper are:

We present a dynamic assignment model, which decouples the application into several functional components, and provides API to third party, which can create and load new functions on device directly.

We proposal a dynamic management algorithm on processes, which automatically adjust the resource threshold of each process related to the number of processes running on device currently. When the resource occupancy of one function exceeds its threshold, one optimal process will be chosen and terminated by taking into count the function dependent relationship and the importance of high-level application.

We implemented DEFIEN on an open source platform (FAST), and evaluated the feasibility and performance of DEFINE by loading a DDoS detection application.

The rest of the paper is organized as follows: Section 2 introduces the motivation and challenges of our work. Define is proposed in section 3, and process resource management strategy on DEFINE is shown in section 4. Section 5 provides the experimentation and evaluation of DEFINE on FAST. Finally, section 6 draws conclusions from the results.

2 Motivation and Challenge

2.1 Motivation of Creating Service in Network

In recent years, cloud computing has gained great success by sharing computing resources. Enterprises and tenants can rent computing resources in cloud to reduce their costs and management investment in infrastructure. Inspired by this, network service provider can also learn from the idea of cloud services, exchange business model from only providing users with interconnection services to providing a variety of specific network internal services defined by users, such as firewall, load balance, DPI. Nowadays, the typical network internal service needs from the following aspects:

User-defined routing: users want differentiated routing capabilities, such as low-latency routing, low-cost links, routing without passing through some specific areas. The current inter-domain routing protocol (BGP) obviously cannot meet this need.

Cloud computing environment: cloud computing has been explosive growth in recent years, while the cloud only gives users the access to cloud server without configuring their own network services on the cloud devices, such as switches and routers.

Games and video live service: users of this type of service are scattered around, but applications require shorter application access latency, such as video transcoding, aggregation of game updates. Implementing these services within the network can reduce processing latency and improve the user satisfaction.

Network detection: many applications need to detect real-time network conditions (such as accessibility, congestion, link delay, etc.) to check the fault or path selection. However, due to the distributed nature of the Internet, these detection servers must be distributed around the world, and if detected within the network, will be more accurate and complete network state information than detecting at the network edge.

There are also some technologies presented to overcome these difficulties, e.g., network function virtualization [5], to build separate virtual networks for different users on the physical network, including virtual node resources and link resources. Users can deploy a variety of services on their virtual networks. This approach is similar to IaaS (Infrastructure as a service) in cloud computing, but this has the obvious drawback that users still need to manage virtual networks like physical networks. And through the support DEFINE network equipment deployed to the network, you can provide users with a variety of services.

2.2 Challenges

At present, the deployment of new network services in the network faces some problems. Developing a new network service is a long process, especially when the equipment vendor needs to add a third-party technology to the device. Since vendors take into consideration the property and security of the devices, these devices systems and hardware architecture are usually closed. While third parties can gain access to the code base through a number of commercial means, this process requires a lot of money and time overhead, which is obviously not allowed for deploying a new technology/service.

The problem is exactly what DEFINE is trying to solve. DEFINE, by decoupling the application into different functional components and reinterpreting the interaction between components with a common perspective, provides a simple application deployment model, as well as a set of platform interaction interfaces that ensure fast application development and deploy. In addition, in order to allow third-party applications to coexist peacefully, DEFINE proposed a process management algorithm. Finally, DEFINE has also proposed a viable path for collaborative applications for different applications.

2.3 Related Work

Network Function Virtualization (NFV) [5] is used to solve the existing limitation number and inflexibility of current network equipment. NFV uses a standard virtualization technology, which removes the traditional functions on the specific devices to a union and high-performance server. NFV runs on a standard server, and the network function can be performed on this server without redesign a new network device.

However, the DEFINE presented in this paper is quite different from the NFV. First, the inter-operability

between NFV systems is weak. In an actual network, the resources between distinct heterogeneous systems cannot be shared to satisfy the need of collaborative services. In addition, the interaction and switching process between systems is complex, and cannot meet some real-time requirements of service (e.g., computer games). Second, the performance of NFV cannot be guaranteed. Since NFV is based on software, which has additional cost to send packets from the network card to CPU, and this I/O cost can become a bottleneck. However, DEFINE is a cooperation method based on hardware and software, i.e., the general packet processing part is offloaded to the hardware-based data path, and software data path is used to deal with the complex part, e.g., state processing. In this way, the performance has been greatly improved. Finally, different form NFV which has both a specific physical part and virtualization resources that requires complicated management, Define is based on the existing network architecture, which can load different application functions to the device with simple configuration. That is, DEFINE does not involve any complex network management, which greatly reduces difficulties of operation and maintenance.

3. Define Architecture

3.1 Design Idea

As the control logic and forwarding logic of traditional network devices are tightly coupled, the control plane expansion and customization is so complicated that deploying new application on existing network directly is difficult. DEFINE inherits the idea of separation and separation from SDN, decoupling the application into different functional components, and the components running in the controller are responsible for deploying the policy. The components in the network device are forwarded under this policy. Reduce the complex functions of network equipment bearer, and improve the flexibility of network function loading and deployment. However, in SDN, most control operations and decision are processed in the control plane, which causes the controller to respond frequently to the data plane that greatly increases the load of the southbound interface to become a system bottleneck. Therefore, DEFINE offloads some of general functions of the control plane into the data plane and gives sufficient computation resources to allow the application to customize its own data plane functionality so that it can perform operations that can only be handled in the controller before.

DEFINE overall structure is shown in Figure 1. DEFINE design a set of communication instructions to achieve that data plane application can get specific packets from hardware-based data path (match some pre-defined rules) by registering a callback function, and sends back the processed packets to the hardware-based data path. For example, in Figure 1, there are functions A, B, and C represent the message processing logic for applications A, B, and C, respectively. They can receive the destination packets from the forwarding pipeline through the API provided by DEDINE. Some special applications, e.g., firewalls that implement interception, may drop their packets after they are viewed, protecting the target network or the terminal from malicious traffic attacks.

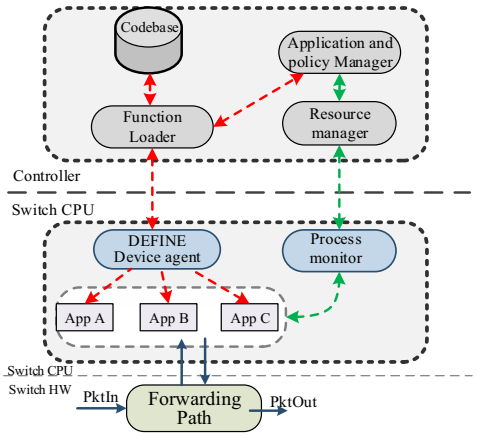

Figure 1: Architecture of DEFINE Framework

3.2 Key Modules in DEFINE

DEFINE defines a complete process from third-party application development, application registration to user application loading.

In DEFINE, controller consists of a code library, application and policy management module, function load module, resource manager. In the software-based data plane, DEFINE has designed a DEFINE agent to receive functions from the controller to load instructions and application code and recompile and run locally on the device.

Furthermore, we design a process detector in the data plane that polls the resource occupancy rate of each application process at regular intervals, passes the passive or proactive way to the resource manager in the controller. The resource manager then makes scheduling decisions according to the consideration of function priority and dependent relationship.

Application and policy manager: It is charge of two main functions. First, it receive the function code from a network administrators, and it store that code into codebase and information of the function into a function table in DEFINE for further usage. Second, this module also receives service requests from each network tenant, and it translates them into service policies and stores the policies into a policy table.

Function loader: This module is mainly responsible for receiving and analyzing command from the application and policy management module, and obtain code file from the code-base. Then, send the Function_ Loading message to the DEFINE device agent, which contain a {Function ID, Function Type, Path},

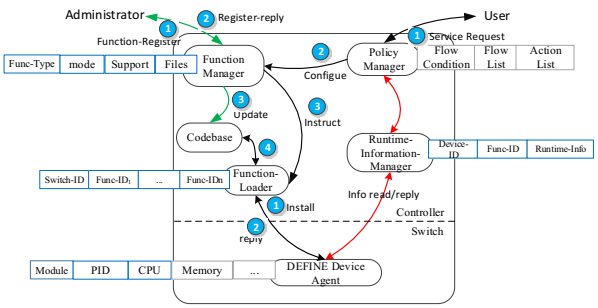

Figure 2: DEFINE workflow

Where Function ID and Path are both used to load a function code in codebase

Table 1. Parameters in the 3-tuple

Parameter(s)	Explanation
Function ID	The unique identifier of the function
Function Type	Indicating the function types.
Path	The path of code file

Resource Manager: The resource manager in the controller is primarily responsible for the unified management of the functions that are running on the device. Through the process management algorithm, dynamically calculate the threshold of the resource occupancy rate of each process. Once an application resource occupancy rate exceeds the limit value, the function limit message will be sent to the device agent, then the agent will limit the rate that the process receives packets.

DEFINE Device Agent: It is a daemon on the data plane. It proxies the connection between the switch's data plane and the controller, which allows it to intercept messages between them. In addition, it is responsible for loading the application, managing the forwarding table of the device, and forwarding the data packets sent from the forwarding path to the specific application processing component.

Process Monitor: Collects the resource occupancy rate of all application processes on the device, including CPU usage and memory usage, and sends the information to the resource manager in either active or passive manner.

3.3 Define

A typical operation of DEFINE works as follows. A network administrator registers network service to DEFINE. After registration, tenants need to create their service requests and submit them into DEFINE. Then, DEFINE parses the submitted service requests to understand the intention of tenants and writes the corresponding service policies to policy table. Next, DEFINE inform Function Loader that 3-tuple mentioned above and the latter obtain function code from codebase. When the DEFINE device agent receives a message from the Function Loader that instructs it to load a function, it:

- Writes the function's code to a local temporary file.

Registers the function information in a local table so that it gets called whenever the agent receives a message from controller with the funcion's ID in its type field.

Compiles the function's code, and dynamically links it to the DEFINE data path agent.

Starting function.

Table 2: The API of DEFINE

API	Description
Message2Switch（Message SwitchID）	The interface that application's controller component use to sends the message to the switch component
Message2Controller（Message）	The interface that application's switch component use to sends the message to the controller component
AddCaptureRule（MatchCondition，Module ID）	Register a packet handler to receive packet that match this condition
AddCopyRule（MatchCondition）	Register a packet handler to receive a copy of the packet that matches this condition.
DeleteRule（RuleID）	According to the rule ID, delete the corresponding flow table rules
GetFlowStatics（MatchCondition）	application can obtain statistics for a specific stream with this interface.
FunctionDestroy（Module Id）	When the service is stopped, the controller deletes the corresponding information in the tables and the rule table entries assigned to it.

DEFINE opens up some API for the application, summarized in Table 2. The application's components can use these interfaces to interact with each other and select packets that it want to receive. The details of each interface are as follows:

Message2Switch (Message, SwitchID): This interface allows an application to send messages from controller component (application code running in the controller) to the switch component (application code running in the switch). Where Message is the message content for the internal communication, and SwitchId is the switch ID that identifies the destination device.

Message2 controller (Message): The device agent sends a message to the controller through this interface,

and Message is the internal communication content of the application.

AddCaptureRule（MatchCondition, Module ID）: With this interface, the function can register its paceket handler to receive specific packet when it through the data plane. The characteristics of the data stream are specified by the MatchCondition , which is Generally the five tuple of packet's header. ModuleId is the function id that device agent allocate, the forwarding path uses it to decide which module to forward the packet to. AddCaptureFlowRule can convert the application's custom packet match condition to a Rule Table Entry and write it to the Rule Table and then get the returned Rule ID. The Rule Table contains a set of flow rule entry, which the data plane can use to select specific data flow to send to a functional module. In DEFINE, before the packet pass through data plane, it will first match rules in Rule Table, and then match the Standard forwarding table to decide the next destination in the network where the packet should forward to.

Add Copy Rule (Match Condition): This interface will copy the packet that match the conditions, and then forwarded this copy to the packet handler, and then the original packet will enter to the standard forwarding flow table.

Get Flow Statics (Match Condition): The device agent can send a request message to the data plane through this interface, and can obtain the relevant flow statistics including the total number of bytes, the sending rate and so on. And then the relevant information to obtain the message and then delivered to the application.

Delete Rule (Rule ID): When the application no longer needs to receive the specified packet, call this function to delete the corresponding rule.

4 Process Management

4.1 Lightweight Process Managerment Algorithm

In this section, we present a lightweight process management algorithm is utilized in Resource Manager of DEFINE to manager the application process running on the device. Network equipment deployed in the network must ensure reliability and robustness. Due to the limited computing resources and storage resources of the network equipment, the resources of each process must be allocated reasonably so as to ensure that the resources are not occupied excessively by the individual processes, which will impact other processes, and then damage to the overall performance of the device.

This algorithm mainly consider two factors, the end time and value of the process. The process with shorter running time, or the process is more important, will get more resources. The algorithm is shown in Formula (1):

$$r_i = \frac{v_i + \frac{1}{P_i} \times k}{\sum_{n=1}^{m} v_n + \frac{1}{P_n} \times k} \qquad (1)$$

The formula (1) detail is as follows:

r_i: The maximum percentage of equipment resources that the process can occupy；

v_i: The value of process；

k: Scale factor；

P: The estimated running time of the process；

This algorithm takes full account of the value of the process and the time factor. The value of the process is the primary consideration of the allocation of resources, applications process can be allocated to the more resources if it is more important than the others. For example, firewalls, intrusion detection systems, etc. should give them high priority to ensure their normal operation. When the value of the process is almost the same, we add time factors as a secondary consideration, if a process's estimated run time is relatively short, it will be allocated more resources. Because they can release the resources in a shorter time.

4.2 A Method to Deal with Process Dependencies

With the process management algorithm mentioned above, we can calculate each process's threshold of resource. In general, when the resource occupancy rate of a process exceeds the threshold, the Resource Manager sends the message to the DEDINE device agent to limit the rate that the process receive packet.

However, in some cases, there may be dependency between the applications. As shown in Figure 3, each node in the diagram represents a process, and a parent node has a child node, indicating that the application represented by this child node depends on the application represented by this parent node.

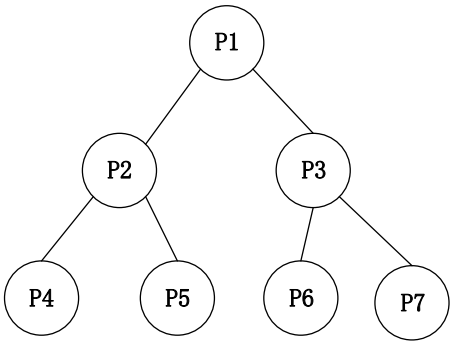

Figure 3: Diagram of process dependency

For example, we assume there is a quite simple Scenario that only after the packet pass through the intrusion detection system detection (P2), and then can be sent to the web application (P4). When the resource occupancy rate of P2 exceeds the threshold, if we choose to turn it off, it will cause all processes that depend on it not working properly. In this case, DEFINE prefers to close the application that depends on it, P4 or P5. It needs to traverse the left and right sub-trees of P2, find all of its leaf nodes, and compare the value of the leaf nodes. Finally, DEFINE limit the resource of the process that has the least value. This mechanism minimizes the impact of the process to be limited on other applications on the device, ensuring system performance.

5. Performent Evalueation

In this section, to verify the feasibility and evaluate the efficiency of DEFINE, we set up an evaluating network. To demonstrate the Rapid and simple development of our framework , we also implement a DDoS attack detection application that can capture the key features of DDoS attack traffics on the data plane by polling the values of counters in OpenFlow switches.

5.1 Evaluation Environment

We create a testbed presented in figure 4. It consists of a FPGA-based OpenFlow switch prototype [] with eight Gigabit ports, a 1.99GHz Intel Celeron J1900 CPU and a 2GB memory that runs Ubuntu 14.04 , a control platform, a quad-core Intel i5 CPU with 8GB of RAM.

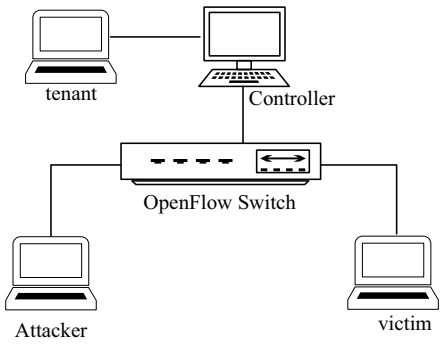

Figure 4 : Diagram of our testbed network.

5.2 Startup Time and Network Cost Measurement

We measure two metrics to estimate the performance overhead of DEFINE. First,we measure the startup time of the DDoS attack detection application. It is an important feature to estimate the performance of DEFINE, because it determines how long the tenants can get the service. Second, we estimate the network cost that represents total cost of delivering and parsing a message between a tenant and an OpenFlow switch.

Fig. 5 presents the test results of the starting the application in the Open Flow switch time (Represented by red) and the network cost (Represented by green).

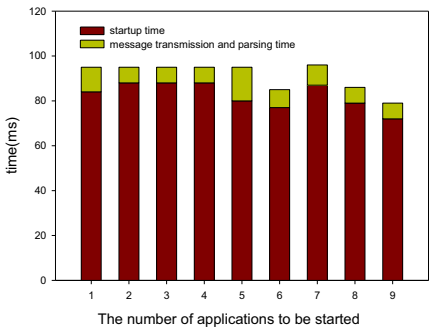

Figure 5 : The number of time takes to transmit message and start different applications.

In this case, we started nine different types of applications to understand the overhead more clearly. In the case of the starting application time, starting an application probably takes about 100ms, which is an expected result from the real network environment case. Another important thing

From the result is that the time for DEFINE to transmit and parse message is less than 10ms, which means that the framework of is quite efficient.

6. Conclusion

Current approaches to create services on IP network is a lengthy process. Because the network equipment customarily have been closed. The DEFINE framework provides a better approach that overcomes these challenges. It allows applications to improve performance by extending devices with custom functionality. We implement a DDOS attack detection application and run it on an OpenFlow switch, to demonstrate the deployment of DEFINE. We have evaluated DEFINE in real networks. With promising results, we believe that DEFINE is a meaningful step towards creating services on IP network.

References

[1] James Kelly, Wladimir Araujo Kallol Banerjee. Rapid Service Creation using the JUNOS SDK. PRESTO'09, August 21, 2009, Barcelona, Spain.

[2] Eric Keller, Jennifer Rexford. The "Platform as a Service" Model for Networking.

[3] D. Taylor and J. Turner, "Diversifying the Internet," in IEEE GLOBECOM, November 2005.

[4] Chen Li, Xiaodong Duan, Wei Chen, Weijiang Cheng [D]. Thinking and Practice of SDN and NFV.

[5] M.B.Anwer, M.Motiwala, M. b. Tariq, and N. Feamster, "SwitchBlade: A Platform for Rapid Deployment of Network Protocols on Programmable Hardware," in Proc. ACM SIGCOMM, 2010

[6] Seugwon Shin, Phillip Porras, Vinod Yegneswaran, Martin Fong, Guofei Gu, FRESCO: Modular Composable Security Services for Software-Defined Networks.

[7] Aaron Gember, Robert Grandl, Junaid Khalid, and Aditya Akella. Design and implementation of a framework for software-defined middlebox networking. In ACM SIGCOMM Computer Communication Review, volume 43, pages 467–468. ACM, 2013.