Accessing Cloud with Disaggregated Software-Defined Router

Hua Shao, *Tsinghua University;* Xiaoliang Wang, *Tencent and Nanjing University;*
Yuanwei Lu, Yanbo Yu, and Shengli Zheng, *Tencent;*
Youjian Zhao, *Tsinghua University*

https://www.usenix.org/conference/nsdi21/presentation/shao

This paper is included in the
Proceedings of the 18th USENIX Symposium on
Networked Systems Design and Implementation.

April 12–14, 2021

978-1-939133-21-2

Open access to the Proceedings of the
18th USENIX Symposium on Networked
Systems Design and Implementation
is sponsored by

Accessing Cloud with Disaggregated Software-Defined Router

Hua Shao[1]*, Xiaoliang Wang[2,3]*, Yuanwei Lu[2], Yanbo Yu[2], Shengli Zheng[2], Youjian Zhao[1]
[1] *Tsinghua University,* [2] *Tencent,* [3] *Nanjing University*

Abstract

The last decade has witnessed a rapid growth of public clouds. More and more enterprises are deploying their applications on the cloud platform. As one of the largest public cloud providers, Tencent cloud serves tens of Tbps inbound-/outbound traffic via cloud gateways for customers with diverse cloud access requirements. Traditionally, cloud gateways were built with proprietary routers. From years of experience operating cloud network, we found that commodity router based cloud gateways are hard to scale, lack of extensibility and are difficult to inter-operate with the SDN-based cloud networks. To this end, we build our own Disaggregated Software-defined Router (DSR) to serve cloud access traffic. We architecturally split cloud router functionalities into several disjoint modules: 1) an access module built out of off-the-shelf commodity switches; 2) a software-based fast and scalable forwarding module; 3) a robust and scalable routing module built with commodity servers; 4) an SDN control module for traffic management and devices configuration. All the components can be independently scaled and maintained. DSR can deliver new network features at high velocity and has sustained the rapid growth of the cloud access traffic. In this paper, we present the design, implementation and our years of operational experiences of DSR.

1 Introduction

With customers increasingly deploying their computation and storage services in the cloud, public cloud services have achieved rapid growth in recent years [11, 16, 18, 35, 37]. As one of the largest cloud providers, Tencent cloud has provided a wide range of services including computing, storage and CDN to support a diverse set of customizable solutions across multiple industries such as e-commerce, online education and mobile games. To satisfy the cloud access requirements of worldwide customers and their end-users, it is critical for the cloud provider to set up fast, secure, stable and low-latency connection among cloud customer's on-premise datacenters, distributed sites, Internet end-users and the cloud resources.

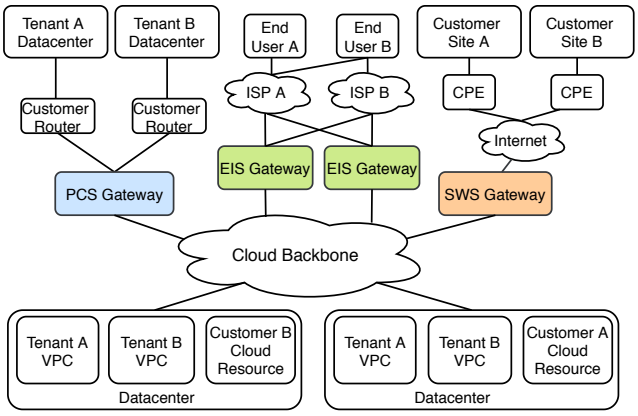


Figure 1: Various types of cloud access scenarios

Tencent cloud supports various types of solutions to facilitate customers' cloud access requirements, namely Private Connect Service (PCS), Elastic Internet Service (EIS) and Software Defined WAN (*i.e.* SD-WAN) Service (SWS).

Private Connect Service. PCS establishes a direct connection between customers' on-premise datacenters and public cloud. It bypasses the Internet and offers higher reliability, faster speed and lower latency than public Internet connections. As demonstrated in Figure 1, customers access to their virtual private clouds (VPCs) hosted in the public cloud through PCS gateways. In our network, each PCS gateway supports ∼10K tenants, one of which has up to 20K routes. PCS provides guaranteed network quality, like consistent network performance on latency and bandwidth with low cost.

Elastic Internet Service. Many customers, *e.g.* content providers and online education institutes, run their computing and storage services in our cloud datacenters. EIS allows global end users to access these online services. As shown in Figure 1, EIS gateways are widely deployed at the Points of Presence (PoPs), interconnecting with multiple Autonomous Systems (AS). EIS gateway needs to fulfill several important

*Joint first authors

tasks, *e.g.* BGP peering with different ISPs, management for inbound/outbound traffic across optimized network routes and forming demilitarized zone (DMZ). As the main entrance of a cloud region, EIS gateway needs to support a large forwarding table. In our settings, EIS gateway needs to support up to 10M longest prefix match (LPM) entries.

SD-WAN Service. Software-defined WAN Service (SWS) allows enterprise's branch offices to access their resources on the cloud through encrypted connections over any mixed transport services, such as LTE, Internet broadband, and MPLS, as illustrated in Figure 1. By leveraging IPSec VPN, WAN optimization (*e.g.* application-aware QoS, robust redundant packet transmission and data compression) and software-defined management/orchestration technologies, SWS simplifies IT infrastructure deployment and network management to connect users at distributed sites to applications on the cloud securely.

Cloud gateway is the main component in the access sites to meet these access requirements. Traditionally, the cloud gateway applies proprietary commodity routers to provide large-scale port extension, high speed traffic routing and forwarding, as well as access control against illegal connections or DDoS attack. However, with the rapid increase of customers, the traditional solution can not sustain anymore. First, we can not deploy millions of forwarding table entries in the commodity routers to meet our customers' requirements. Second, the lack of programmability of the proprietary devices slows down the feature roll-out velocity. We have to count on vendors' development plan which can be months or even years. Third, it is hard to inter-operate between the cloud SDN network and the edge BGP network. To meet these operational challenges, we provide ingenious solution for diverse types of customers to access the cloud (see § 2.2 for more details).

We set out to build a Disaggregated Software-defined Router (DSR) to address the above challenges. It enables cost-effective scaling to keep up with the fast growth of traffic volume. It leverages software feature velocity to provide fast response to user requirements. To be specific, DSR consists of a high performance data plane forwarding module, which is responsible for operations like routing table lookup, tunnel traffic encapsulation/decapsulation, IPSec encryption/decryption and packets forwarding. A standalone BGP routing module is built to exchange routing information between cloud network and external peers. SDN controllers are used to realize configuration, state management and orchestration of different components. Last but not the least, it performs routing decisions in a centralized fashion, thus yields better traffic steering. We leverage a disaggregated architecture, *i.e.* each component can be scaled independently and released on demand.

DSR serves tens of Tbps traffic, which has been deployed in Tencent cloud for over 3 years. In this paper, we introduce the design and implementation of the scalable and flexible

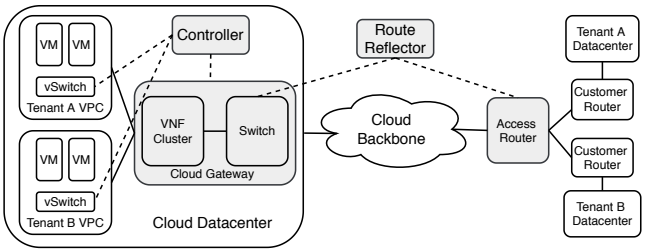


Figure 2: Commodity router based PCS

system to access the cloud. We first use an example of the PCS connection to demonstrate the limitation of commodity routers based gateways (§ 2). Referring to the structure of commodity devices, we explain the architecture of DSR (§ 3). We present the system design details (§ 4, § 5, § 6) and share our operational experiences to the community (§ 8).

2 Background and Motivation

For ease of understanding, we first demonstrate the original solution of PCS using commodity devices. Then, we introduce the limitation of commodity router-based architecture which motivates the design of DSR.

2.1 Commodity Routers based PCS

Previously, PCS service is built by leveraging the MPLS VPN technique [7]. Figure 2 shows the architecture of the original version of PCS, which consists of the following components:

- **Access Router (AR):** It is a commodity router supporting various types of electrical or optical interfaces with different rates, *e.g.* 1Gbps/10Gbps/100Gbps. AR sets up BGP sessions for routing between customers' on-premise datacenters and public cloud. Tenants are isolated based on Virtual Routing and Forwarding (VRF) tables. AR works as an MPLS provider edge (PE) router of the cloud MPLS backbone network.

- **Cloud Gateway (CG):** CG is deployed in cloud datacenters as another MPLS PE node of the backbone network. It is built by leveraging a commodity switch and a server-based Virtual Network Function (VNF) cluster. The commodity switch is the gateway between datacenter network and MPLS backbone network. Tenants are isolated through the switch build-in VRF. Due to the limitation of routing entries in switch, the routing table from the on-premise datacenters to tenant Virtual Machines (VMs) are stored in the VNF cluster.

- **Route Reflector (RR):** RR is a commodity router. RR establishes MP-BGP sessions with AR and CG respectively. It conveys customer's on-premise IP routes to the cloud switch and vice versa.

- **Controller:** Controller is introduced to manage and configure network components. For example, controller installs a default route to CG at virtual switch (vSwitch) of each server for traffic destined for on-premise datacenters.

Now we explain how a packet is forwarded from a tenant virtual machine to its on-premise datacenter. For the outbound traffic, the vSwitch first encapsulates packets with GRE [13] header (The VPC id assigned to the tenant is encoded in the GRE header) and forwards the packet to VNF deployed in a server cluster. The VNF decapsulates the GRE tunnel and encapsulates the packet with a VLAN header which embeds a certain VLAN id according to the VPC id. Then the packet is forwarded to the cloud switch, which forwards the packet to the MPLS backbone based on the forwarding table. Whenever AR receives the packet, it forwards the packet to the destination.

For the inbound traffic, the main difference is at the cloud gateway. The cloud switch is configured with a default route to forward all traffic to the VNF cluster, which stores all the routing information for VMs in the cloud datacenter. The gateway supports the access control through firewall and safeguards applications through Distributed Denial of Service (DDoS) protection service running on dedicated servers. Finally, the VNF cluster encapsulates the packet with GRE tunnels and forwards it to certain physical machines according to its routing decision.

2.2 Motivation

Based on production experiences of operating the network, we found many limitations of the commodity router to meet the rapidly growing customers and their rising demands.

For AR, though the commodity devices can forward a large volume of traffic, *e.g.* Cisco ASR 9000 series routers [5] support tens of Tbps throughput, it cannot scale to support a large number of tenants, *i.e.*, the number of VRFs and LPM table entries are rather limited. The commodity access router only supports less than 1K of VRF elements and 1M FIB entries which cannot satisfy the requirements of PCS which needs to support $\sim$10K tenants and EIS which requires $\sim$10M forwarding tables. We have to expand network by using additional routers. It is not cost-effective and substantially increases the management and control complexity.

We usually need to roll out new network features in few weeks to meet our customers' requirements, *e.g.*, supporting jumbo frames or 4B-length AS-path attribute. In addition, the cloud providers usually deploy new functions to improve operation and maintenance capabilities, *e.g.* measurement of top-*N* largest flows. However, due to the lack of programmability and limited device management API of proprietary commodity routers, it can take several months or years if we need vendors to make change to device software or hardware [35, 41]. The slow feature velocity significantly affects the user experiences and the reputation of the cloud provider.

BGP protocol is widely used to interconnect autonomous networks in the Internet [35, 41]. For instance, MPBGP protocol is used to exchange routing information between access router and cloud switch. On the other hand, the cloud networks have increasingly adopted the SDN technique for efficient traffic steering and fast network failure convergence. We found many problems for inter-operation between the cloud SDN network and the external BGP network. For example, in commodity router based PCS, the controller should update the state of the direct connection in real time, such that if the direct connection fails, vSwitch can carry out corresponding routing updates and quickly switch to the backup path. To achieve that, the controller has to periodically pull the routing tables from CG. However, due to the slow NETCONF messages processing, the long pulling interval leads to slow routing convergence.

3 Design

We target at building a general platform, *i.e.* a disaggregated software-defined router (DSR), to meet various types of cloud access requirements.

3.1 Design Rationale

DSR should meet the following requirements:

- **Simplicity**. The commodity router needs to support all kinds of network standards and a variety of customized protocols for cloud network, enterprise network or campus network, etc. In cloud access scenarios, many features integrated in the commodity routers are not required. We can simplify the design by supporting a minimum set of functionalities, *e.g.*, BGP and static routing for inter-operation with external network, Bidirectional Forwarding Detection (BFD) [23], Internet Protocol Service Level Agreement (IPSLA) [10] for fast convergence, and VXLAN [27] and GRE [19] for tunneling.

- **Scalability**. To meet various cloud access requirements of Internet users, customer on-premise datacenters and enterprise branch-sites, the key components of the cloud gateway, *i.e.* data plane, routing and control plane should be able to scale independently without affecting each other.

- **Reliability**. All the key components are built in the cluster with redundancy such that no single point failure can cause performance degradation. Specifically, the key components can be deployed cross available zones for remote disaster recovery.

- **Elasticity**. The system should support high feature velocity and realize efficient inter-operation between the external

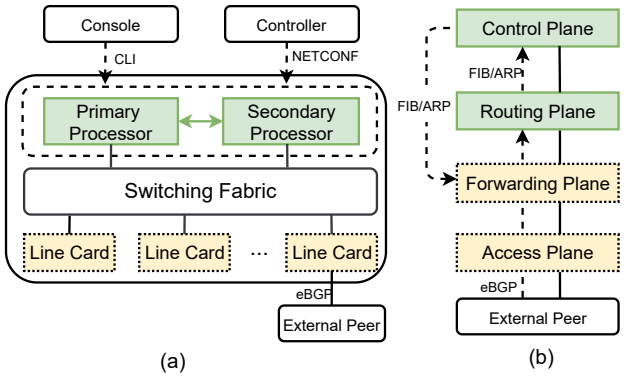


Figure 3: Sketch of (a) a typical commodity router and (b) the disaggregated software-defined router architecture

BGP network and the internal software defined cloud network in real time.

3.2 System Overview

3.2.1 The disaggregated design

We review the architecture of a typical commodity router. As shown in Figure 3 (a), it consists of three major components:

- **Processor**: Processor acts as the control plane, which runs network protocols like BGP, OSPF and BFD. Usually, there is a secondary processor for high availability.
- **Switching Fabric**: Switching fabric is responsible for high speed internal inter-connection.
- **Line Card**: Line card is intended to connect many users with different types of interfaces. It forwards and filters packets based on routing tables and access control lists (ACLs) respectively.

The three components are tightly coupled in the commodity router. The processor capability, routing table sizes and bandwidth are configured with a fixed ratio. We can not independently scale any component on demand.

Consequently, we turn the router into a disaggregated architecture, as illustrated in Figure 3 (b):

(i) The functions of line card are divided into two components: the access plane and the forwarding plane. The access plane provides various types of interfaces and supports layer 2 forwarding. Based on the operational experiences, the access plane is stable, *i.e.* we do not need to frequently update the access plane. As a result, we build the access system with a group of small-scale commodity switches. The forwarding plane deals with layer 3 packets processing with large scale forwarding tables (FIB, IPSec, QoS, etc). We introduce the software-defined forwarding module to achieve high feature velocity and scalability.

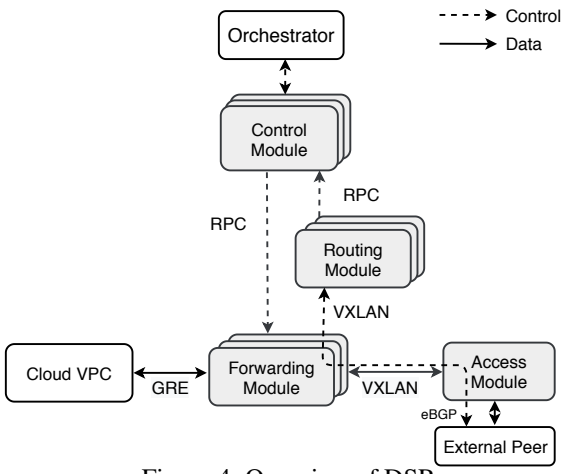


Figure 4: Overview of DSR

(ii) The functions of the processors are divided into two components: the routing plane and the control plane. They are implemented with software over common servers. The routing plane takes charge of the protocol and routing management, like BGP, BFD, IP-SLA and static routing protocols. The routing information is delivered to the control plane through RPC messaging. The control plane stores the routes, generates the forwarding tables, and then installs the FIB/ARP tables to the forwarding plane. This design is based on the observation that the routing plane and control plane require different resource settings. For example, the routing plane requires high computational capacity for complex protocol processing. Meanwhile, the control plane deals with performant management and ensures consistent forwarding processing during upgrade. We use different software suites (DPDK, ONOS&ODL) there. As a result, the development and release cycles are different for these two planes.

(iii) We use the standard VXLAN protocol for the interconnection of different planes. Tenants are isolated based on VXLAN VNI. All the components are decoupled and deployed in different fault domains which allows independent scaling. Generally, the access plane is deployed at the access sites, *e.g.* PoPs or remote user sites. The others are deployed in the cloud datacenter.

(iv) The forwarding, routing and control planes are built using multiple servers in a cluster. No single server failure can cause the break down of the entire system.

3.2.2 System architecture

The architecture of DSR is shown in Figure 4. It includes the following components.

Access module. We apply commodity off-the-shelf switches for the inter-connection between the internal cloud network and the external peers. It supports protocols of BG-

P/OSPF/BFD/LACP (Link Aggregation Control Protocol) at underlay network. It provides various types of physical ports with different rates, *e.g.* 1GE/10GE/100GE. For the overlay network, it realizes layer 2 isolation for tenants using the switch built-in VLAN functionality. VXLAN tunnels are established for communication between the access module and the forwarding module.

Forwarding module. The forwarding module is responsible for high performance packet processing and routing with large forwarding tables. It sends BGP/BFD messages to the routing module via layer 2 VXLAN tunnel and forwards data path traffic VPCs via self-defined GRE tunnel in the cloud datacenter. We build the forwarding module using a group of servers. With the server built-in large memory, it can support a large number of VRFs and large LPM tables for tenant isolation and packet forwarding. Forwarding module also supports IPSec processing and WAN optimization functionalities.

Routing module. The key purposes of the routing module are: (i) inter-connecting with the external peers (commodity devices) through dynamic routing using BGP; (ii) supporting a large amount of customers and maintaining the neighbors information; (iii) realizing fast convergence and failover when routing information updates and network failure happens. The core of the routing module is a high performance home-made BGP speaker. We optimize the BGP speaker performance and customize it for different scenarios. For example, in PCS, we embed the bandwidth allocation ratio for customers' traffic in the BGP header. While, in ISP peering scenario, we select the routing path based on the network state through SDN controller. It supports Non-Stop Routing & Forwarding (NSR& NSF) for high availability. We optimize its ability to handle a large number of BGP updates. For outbound traffic, the control module feeds VPC routes into the routing module, which in turn conveys the routes to the external peer. As the BGP messages are carried over VXLAN tunnels, they are able to establish BGP sessions directly between the external peer and routing module.

Control module. Control module acts as a local controller. It stores the routing information and is responsible for optimal traffic steering computation. Apart from that, control module provides distributed message queues in order to efficiently synchronize dynamic forwarding rules among forwarding module clusters and routing module clusters.

Orchestrator. The orchestrator acts as a global controller. It is responsible for distributing operator's configuration requests to corresponding control modules. The Orchestrator collects particular traffic scheduling requests from external management system and synchronizes them to the control module. It is a centralized routing computation platform by taking into consideration of various kinds of metrics, *e.g.* network latency, bandwidth capacity and monetary costs, to achieve consistent performance while reduce costs for customers [35, 41].

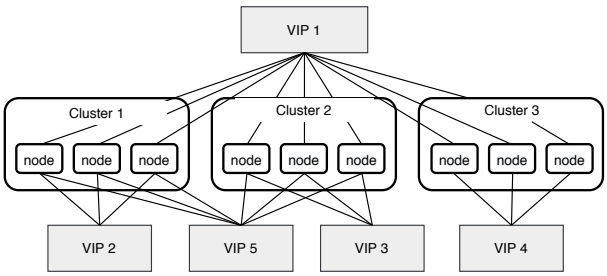


Figure 5: Scaling out forwarding plane through multiple VIPs

3.2.3 Challenges

Since DSR components are loosely coupled with each other, we can easily add or remove one instance in any component independently for scaling or upgrading purposes without affecting the whole system. For the design and implementation of each specific module, we meet the following challenges:

- We need to ensure high forwarding and routing capacity in software given a large number of tenants and highly frequent routing updates. To be specific, DSR serves 10Ks of tenants and maintains 10M entries routing tables at line rate. We have optimized the forwarding module and the BGP speaker of the routing module to address this challenge. Meanwhile, we provide low latency packet forwarding by minimizing the impact of system call, CPU scheduling and packet losses.

- Originally, the commodity router is a single device to serve the arrival traffic. However, the DSR is a distributed architecture consisting of multiple components. The instances of each components are deployed in the cluster, which are connected through multiple-path Clos network. The disaggregated design makes the management more complex than managing an individual commodity router. We need to carefully schedule and route the arrival traffic and control messaging inside DSR, which can easily affect the performance and stability of the whole system.

4 Scalability

We explain the optimization on forwarding module and the routing module to address the scalability issues.

4.1 Scalable Forwarding Plane

In order to provide a scalable forwarding module to meet the packet processing requirements, our efforts go along two directions: (i) Scaling out the packet processing components by adding new cluster of servers; (ii) Optimizing the program of packet processing and LPM lookup to ensure fast forwarding at large scale.

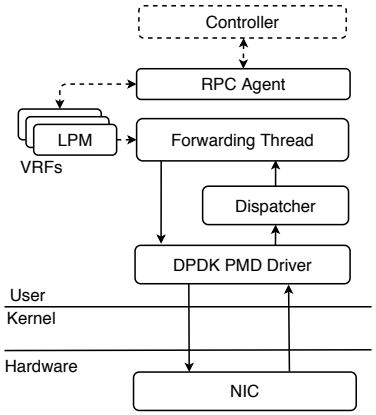


Figure 6: Architecture of the dataplane forwarding module

4.1.1 Scale out with multi-VIP

For building a scalable forwarding module based on common servers, a straightforward approach is to expose a single virtual IP (VIP) for all servers in the cluster. We announce the VIP to all tenants. All servers are then configured with the same forwarding tables. By doing so, arrival traffic are balanced to servers using ECMP hashing at a per flow basis. With this stateless load balancing, the system's processing ability can be easily scaled out/in by adding/removing servers. However, with more and more VRFs are configured, the forwarding capability of the cluster is limited since each server has to support the entire routing tables of VPCs in the region.

In fact, in our production network, the tenants' requirements on bandwidth are diverse. Most tenants require traffic bandwidth less than 10Gbps. Only a few tenants have traffic around 100Gbps. We seldom see tenants have traffic more than 500Gbps or over the forwarding capability of a single cluster. To this end, we introduce a flexible multi-VIP structure. As shown in Figure 5, each cluster is configured with multiple VIPs shared by tenants. For example, VIP 2 belongs to cluster 1, which serves multiple tenants with little traffic. Meanwhile, VIP 5 is applied to cluster 1 and cluster 2, which are assigned to users with large amount of traffic, *i.e.*, these users can deploy forwarding tables in both cluster 1 and cluster 2. Similarly, a tenant with an extremely large amount of traffic can be fulfilled by using VIP 1. Consequently, there is no needs for a single server to store routing tables of all tenants. We can improve the scalability of the system while retaining the benefits of the stateless load balance.

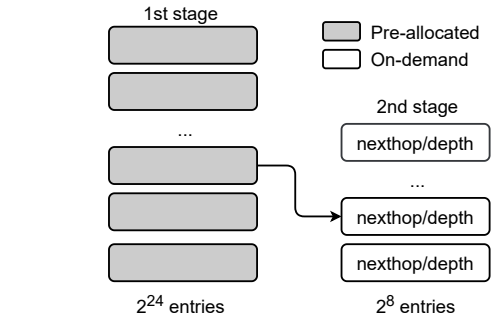
4.1.2 Fast datapath route lookup

We have leveraged the DPDK suites [12] to develop the high performance dataplane forwarding module. As illustrated in Figure 6, the arrival packets are forwarded directly from the NIC to user space bypassing kernel overhead. To efficiently utilize the modern multi-core CPU architecture, traffic need to be balanced to multiple cores, we apply the NIC built-in RSS functionality to uniformly distribute the traffic to different dispatchers which run on dedicated CPU cores. To avoid packet out-of-order, packets belong to the same overlay flow should be processed on the same core. To achieve that, the dispatchers decapsulate packets' outer tunnels and distribute packets to different forwarding threads based on the overlay 5-tuple. The forwarding threads encapsulate packets with another tunnel according to packets' destination. Packets are then forwarded to external WAN network. We plan to accelerate this procedure with advanced NICs which support overlay packet header hashing.

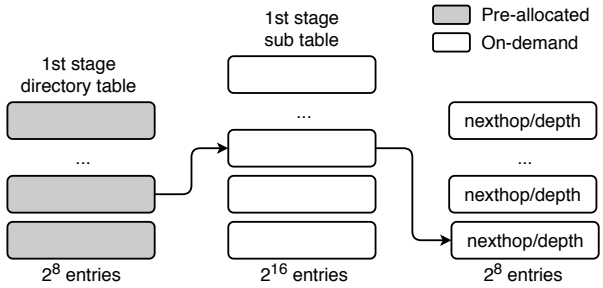
Short packet processing pipeline. In high performance packet processing, a long pipeline will likely lead to more cache misses and causes performance degradation. The packet processing cost mainly stems from the LPM lookups. Generally, a packet would require two LPM lookups, *i.e.* the packet first does LPM lookup for underlay encapsulation header based on the overlay IP, then the encapsulated packet does LPM look up for the output physical port based on the underlay IP. We have shortened this pipeline by saving the second LPM lookup. To achieve that, we combine the second LPM lookup results *i.e.* the output physical port and the the first LPM lookup results *i.e.* the encapsulation header into one unified action. The action is pre-programmed into the action field of the first LPM lookup entry. After the first LPM lookup, the packet is encapsulated and forwarded to the corresponding physical port following the pre-programmed rules. With the shorter pipeline, the packet processing performance is largely improved.

Fast route lookup at large scale. When dealing with 10M routing entries, we need to take care of the storage and the lookup speed. At first, we leverage the LPM library in DPDK suites [1] to implement the forwarding function. Unfortunately, the original DPDK LPM library is not efficient to store routing entries. We encountered performance issues with the increasing amount of tenants.

To illustrate the problem, we first briefly introduce how the DPDK LPM library works. The library uses a classic trie-tree structure to store the routing entries. As shown in Figure 7(a), for IPv4 lookup, it uses two stages. The first stage is an array with 2^{24} entries. The higher 24-bit of an IPv4 address is used as the array index. Each entry stores the base address of a secondary stage array. The lower 8-bit of an IPv4 address acts as the index into the secondary stage array. Based on the address in the first array, a routing result can be identified. Each IPv4 lookup requires at most two memory lookups in this design. However, to achieve this, DPDK LPM library needs to pre-allocate a large memory to store the entire 2^{24} entries no matter whether there exists an IP route or not. This leads to severe memory waste and limits the ability to support a large number of VRFs. For example, to store 64K IPv4 routes, the first stage would require $2^{24} \cdot 4B = 64MB$ memory for each VRF. In the worst case, to store 64K routes in the second stage, it requires $64K \cdot 2^8 \cdot 4B = 64MB$, *i.e.* 128MB



(a) DPDK LPM table design



(b) Optimized LPM table design

Figure 7: LPM table data structure

memory in total. To support 10K VRFs, 1.28TB memory is required. Moreover, the first stage requires a large continuous memory, the memory fragmentation makes it hard to fulfill the need especially when the system has been up for a long time.

The root cause of this problem is the static memory allocation mechanism. In fact, only few tenants would require a large routing table. Most of our customers has a small routing table. Based on this observation, we design a dynamic memory allocation mechanism for LPM tables (Figure 7(b)).

First, we turn the DPDK LPM first stage array into one directory table (2^8 entries) and many sub-tables (each with 2^{16} entries). Only the directory table is pre-allocated. Thus the minimum memory required for a VRF is only $2^8 \cdot 8B = 2KB$. With the same amount of memory, the number of VRFs we can support increases by 3 orders of magnitude. Furthermore, with the same amount of memory, the maximum table size that can be used by a VRF is also enlarged.

Second, in the original DPDK LPM design, the memory of both the first and the second stages are pre-allocated. In contrast, in our design, the first stage sub-tables and the seconds stage tables are both allocated on demand. This significantly improves the memory efficiency. However, this design leads to problem when considering the route modification and deletion. When a route is deleted, we can not free the memory taken by this route because there may be arrival packets at the same time. A lock mechanism can help solve the problem but with a large overhead. Instead, we have designed an user-space RCU (Read-Copy Update) mechanism to solve this problem without using any lock. To be specific, when updating a forwarding entry, we first do a copy of the original data and make changes on the new copy. Then we modify the corresponding pointer to point at the new copy. The old entry is deleted only when there is no other threads accessing it. We implemented the RCU as a Quiescent-State-Based Reclamation (QSBR) flavor one [20]. Our evaluation shows that we can provide high performance packet processing and high route updates speed at 64K entries/s simultaneously.

Hardware acceleration With the rapid growth of traffic and slowing down of Moore's law, the CPU becomes the bottleneck for packet processing in software-based routing modules. To address this problem, prior studies have applied the cost-efficient hardware acceleration technique [16,21,28,29]. With regard to the limited forwarding table, *e.g.* Barefoot Tofino chips have ~60MB on-chip memory [6, 16, 28, 32], which can only support 100K of LPM entries according to our evaluation, we have designed a large flow offloading scheme for the forwarding module. Based on the historical information of the cloud traffic, we found that the top 1% largest flows contribute 70% of overall traffic. We maintain a small group of prefixes in switch hardware for those large flows and forwards them with the hardware. By pushing the flexible and complex logic to software while leveraging the stable hardware offloading features, this approach saves more than 50% CPUs and greatly reduces the cost[1].

4.2 Highly Scalable Routing Module

Traditionally, in commodity router based PCS, standard MPLS-VPN technique [42] is used (§ 2). To differentiate BGP routes of different tenants, BGP messages from each tenant is configured to export m distinct BGP Routing Target (RT) attributes. At the cloud side, different tenants are separated via VRFs, say n VRFs. A tenant VRF only learns BGP routes from the corresponding remote on-premise datacenter, which is achieved by importing certain RTs. The process of route insertion is rather slow for traditional routers. Once receiving a BGP route update, the corresponding VRF is located by comparing each RT attribute with all VRF's RT values. m RTs (m is around 10) and n VRFs (n is around 10K) result in an insertion complexity of $O(mn)$. The performance can not sustain highly frequent BGP routes insertion.

In DSR, with the help of the connect module switch, BGP import/export RT features are no longer required. Specifically, DSR leverages the connect module switch to tag routes from different tenants based on VXLAN tunnels. The VNI in the VXLAN tunnel is pre-determined for each tenant. Then the BGP module can insert the route to the corresponding VRF using the VNI number. This design reduces the route insertion complexity to $O(n)$.

Some PCS customers have multiple datacenters. The newly built datacenters use the latest DSR based PCS while others

[1]The detail of the implementation is beyond the scope of this paper.

may still use commodity router based PCS. For availability purpose, the routing module of DSR needs to learn the routes to certain datacenters from commodity routers. This induces a requirement of inserting BGP routes into VRFs based on the RT attributes. As aforementioned, this would take $O(mn)$ time complexity for m RT attributes and n VRFs, leading to scalability issue.

To optimize the BGP insertion performance in this scenario, we use a hash function based solution to speed up the process of VRFs lookup. The key idea is that we store the RT-to-VRF information in a hash table where the RT value serves as the key. The value of each item is organized as a linked list of VRFs based on the RT. With this optimization, the capability of route insertion is significantly improved.

The routing plane needs to exchange keep-alive messages with a huge number of BGP neighbors, *e.g.* 10K. Traditionally, the keep-alive function is implemented in the same thread with the routing message processing function. When the thread is heavily loaded with routing updates, the keep-alive messages can not be processed in real time, leading to BGP timeout at remote side. We separate the keep-alive message handling and BGP message processing into different threads to solve the problem. As a result, we can maintain stable BGP neighbor relations even under the most heavy BGP message processing workload.

In summary, with all the above optimizations, DSR can improve the routing module BGP performance by ∼20 times. The BGP module can perform 500K routes insertions per second and support 10Ks of VRFs (see § 7.2 for more details).

5 Reliability

We have deliberately designed several techniques to enhance the system reliability.

5.1 Forwarding Path Failure Detection

Conventionally, when peering with commodity routers, we enable BFD protocol [23] to provide fast and consistent forwarding path failure detection. However, with the disaggregated design, components of DSR are deployed in different availability zones. There are multiple paths available between the peering devices and the routing module. The BFD messages will transmit along only one single path. If one link of the path failed, the BFD message will be lost. Then the BGP module would regard that the remote peer is not reachable and delete the corresponding routes for fast convergence. In fact, because there are many paths available from the access module to the routing module, the data path is not disconnected.

To address this problem, we should distribute the BFD messages into multiple paths. We have modified the hash function of ECMP in the access module. In detail, for an arriving BFD packet, instead of using the fixed five-tuple of the message to generate the source UDP port value in the VXLAN header, we apply the IP ID field to compute the ECMP hashing. The hash key is then encoded into the UDP source port of the VXLAN header. As a result, BFD packets will traverse different paths along the network. We apply the similar idea at the forwarding module and routing module for BFD ECHO packets in the reverse route. When a link fails, there are still enough BFD packets available to report the healthy state of the remote peer.

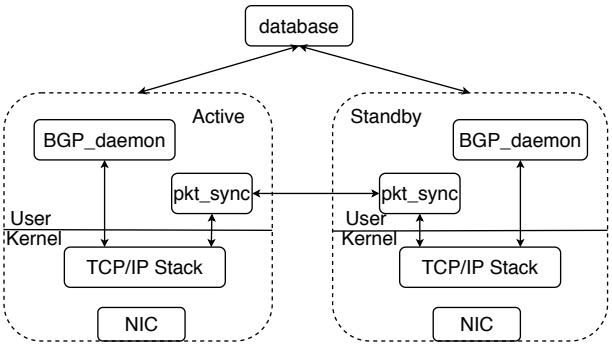


Figure 8: The NSR mechanism of the BGP module

5.2 BGP Non-Stop Routing

We developed a Non-Stop Routing (NSR) [42] mechanism against single BGP module failure. NSR mechanism requires that the failure of the BGP speaker to be transparent to the peering devices. This allows the failed BGP speaker immediately switch-over to a backup BGP speaker, and the backup device have all the information required to take over.

Traditionally, the commodity routers use a backup processor to implement the NSR functionality, as shown in Figure 3. They customize the protocol processors in the Linux kernel stack to enable this feature. However, in our case, the cloud resource manager requires that the kernel software should be homogeneous for all cloud servers to avoid operational issues.

We come up with a solution without kernel modification. As shown in Figure 8, the key idea behind our NSR design is to make the active BGP module synchronise its TCP states, *e.g.* sequence number and window size, with the standby BGP module before the active BGP speaker sends acknowledgements to the remote peer. We introduce a reliable database in the control plane to synchronize the neighbors information. Once the backup BGP speaker receives the TCP states, it initiates a socket system-call with TCP repair option, which keeps the TCP states in correct accordance without sending or receiving any TCP packet. Thus the standby module can have the same TCP states with its active counterpart.

Moreover, during BGP software upgrade, we need to make sure the data path traffic is not affected. This is accomplished by employing the NSR mechanism to proactively stop the active one and hand over BGP processing to the standby module. When the software upgrade is done, the active module is restored. It is notable that this process allows us to scale out/in the routing module easily.

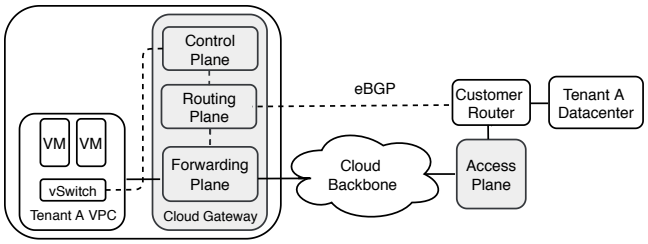


Figure 9: DSR based private line service

6 Fast Development of New Features

Public cloud users mainly rely on their providers to provide an elastic, reliable, scalable and safe networking environment. Customers have diverse requirements in terms of latency, bandwidth, self-defined protocol, etc. DSR is proposed to achieve fast feature velocity to satisfy customers' requirements. Through a few examples we demonstrate the benefits of using DSR for the operation of cloud network.

DSR based PCS gateway. We revisit the example introduced in § 2.1 and show that a more scalable and elastic PCS can be built with DSR compared with previous commodity router based design. As shown in Figure 9, once the tenant network is connected to the cloud network via PCS, the customer router can directly set up BGP sessions with the routing module of the cloud gateway. The routing module announces the routes of tenant on-premise datacenters to its VPC network through the control module. Similarly through the BGP sessions, the tenant on-premise datacenter can learn the VPC routes and VM routes from the routing module. We are able to provide real-time routing updates between the cloud SDN network and the external BGP network.

Customizing egress routing. A PCS customer may set up multiple PCS channels with the cloud through multiple DSR instances deployed at different sites. We can provide network quality aware traffic steering for our customers in this scenario. In detail, beside propagating the peering routes to the control module, DSR measures and informs the control module the quality of each path, including bandwidth, delay and packet loss rate. With both the routing information and network quality information, the control module can choose the optimal egress route for the traffic. Moreover, customers can customize their own routing policy to choose a specific path for certain traffic.

Fast failure recovery. Beside a PCS channel, the on-premise datacenter can configure another IPSec VPN channel over Internet as a backup. By default, traffic are forwarded through the PCS channel. When network failure is detected, *e.g.* a broken cable, the BGP module quickly reports the failure to the control module and withdraws the route. The control module then modifies the next hop of the routing on the forwarding module to the IPSec channel. The whole procedure can be done within 10ms, which is two orders of magnitude lower than traditional solution with commodity routers which requires periodically checking the routing updates from the commodity routers through the slow NETCONF interface.

On-premise to on-premise datacenters traffic acceleration. Through PCS, customers can transfer data among multiple on-premise datacenters via a hub-spoke model, where the cloud gateway acts as the hub. Previously, this is enabled by the route reflector, which reflects BGP routes among multiple cloud access routers. The scalability to support a large amount of customers are limited by the route reflector's BGP processing ability. With the home-made BGP module, the constraint is relaxed as the routing module can be horizontally scaled.

Protection against DDoS attack. DDoS traffic are short-bursts with high volume. When DDoS traffic are identified by the DDoS detection engine, those traffic need to be redirected to a DDoS traffic cleaning center. Notice that DSR is built on commodity servers, it can be potentially affected by DDoS traffic. We aim to redirect the suspicious DDoS traffic at the access module before they come to the software-based forwarding module. Unfortunately, the attack traffic are identified based on layer 3 routing information, but the current access switch works as a layer 2 switch for simplicity. To enable layer 3 packets forwarding for the DDoS traffic only, we introduce an ARP-spoofing scheme. To be specific, we first install the suspicious DDoS traffic routes as advertised by the DDoS detection engine in the access switch. And whenever an ARP request message of the DDoS traffic arrives, DSR can send an ARP reply with the MAC of the access module instead of the MAC of the DSR routing module. By so doing, all the inbound DDoS traffic will be encoded with the MAC of the access module as the destination MAC. Then the access switch can work as a layer 3 switch, and forwards suspicious DDoS traffic based on layer 3 routing table. Notice that as we only do ARP spoofing for DDoS traffic, the normal inbound traffic and outbound traffic are still directly forwarded to the forwarding module of DSR.

7 Evaluation

We demonstrate the capability of packet processing and BGP convergence time of DSR. The servers we used for evaluation are Dell PowerEdge R740 with 192GB memory and 80 cores (Intel(R) Xeon(R) Gold 6133 CPU@2.50GHz with hyper threading enabled). Each server is equipped with a dual 40Gbps port NIC.

7.1 Forwarding Module Evaluation

The host-based DSR forwarding module is able to process packets at line rate with low latency. In the experiment, we generate VXLAN packets and the forwarding module decapsulates a total of 68B from the header of the arrival packet.

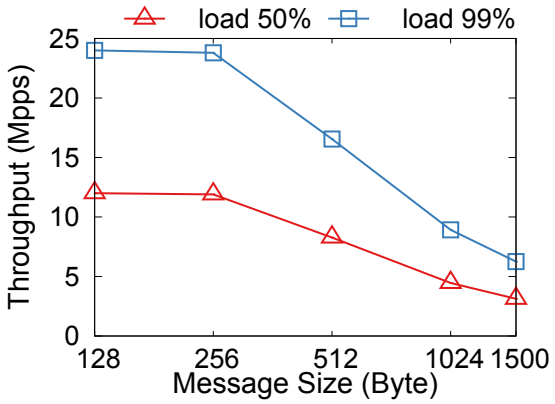


Figure 10: Dataplane throughput under different loads

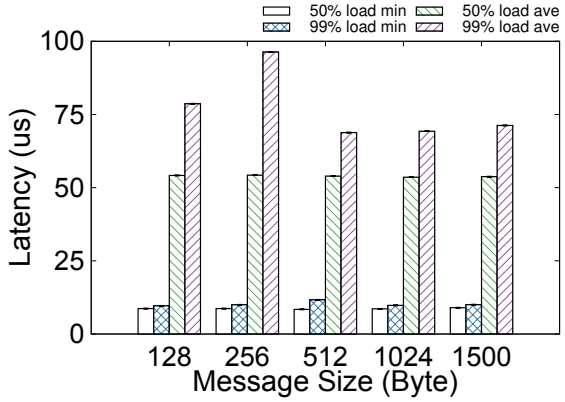


Figure 11: Dataplane latency under different loads

To maintain low latency and high throughput, we have introduced many software optimization techniques in the forwarding plane, *e.g.* kernel bypassing, minimizing cache miss and short pipeline. We measure the CPU consumption of the forwarding plane under moderate 50% and high 99% workloads with various packet sizes ranging from 128B to 1500B. For the load of 50% workload, the generated traffic consume 50% of the CPU while for 99% workload, the traffic consume 99% of the CPU. We can effectively utilize the CPU resource to provide high rate packet processing.

Dataplane throughput under different workloads. As shown in Figure 10, at 99% load, the datapath can process traffic at high rate under different packet sizes ranging from 128B to 1500B. It can achieve 24Mpps for packets of 128B. The throughput decreases proportionally with the workloads which shows graceful scaling property. The good performance stems from our optimizations, *i.e.* short pipeline design and FIB lookup optimization, as introduced in § 3.

Dataplane latency under different workloads. We measure the per-packet processing latency of DSR. As shown in Figure 11, the data plane can provide low and stable latency of 50-70μs for both 50% and 99% workloads. The minimum latency identifies the packet processing delay without ring-buffer queueing delay. It is 10μs. For average latency, it is stable across different packet sizes. With the increase of workloads, the software ring-buffer queue size also slightly increases which leads to a moderate increase in latency.

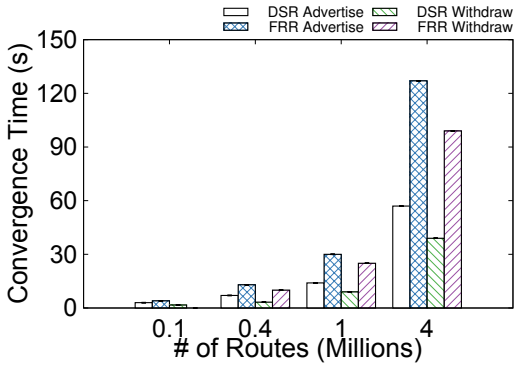


Figure 12: DSR BGP performance compared with FRR

Dataplane throughput during FIB update. To demonstrate the effectiveness of the datapath lockless design, we compare datapath throughput with and without FIB updates. The FIB update rate is 64Kbps. We generate VXLAN traffic for 30s and records the packet processing rate. The throughput can achieve 20Mpps with or without FIB updates, *i.e.* supporting lin-rate packet processing. This demonstrates that the lock-free design enables fast routing updates without impacting the datapath forwarding performance. We omit the figure due to space limit.

7.2 Routing Module Evaluation

We compare the routing module with the open source FRR [3] solution under 300 RTs, and demonstrate the performance of our home-made BGP speaker under a large number of VRFs and neighbors.

BGP advertising and withdrawal convergence time is the key metric to be reported. As shown in Figure 12, the DSR routing module outperforms the FRR under different number of routes, *e.g.* 0.1M, 0.4M, 1M and 4M. For BGP advertisement, when there are 0.1M routes, the convergence time of DSR is 33.3% lower than FRR. When the number of routes is 4M, DSR converges ~2.2× faster than FRR. For BGP withdraw, DSR has 1.5-2× lower convergence time than FRR. The performance gain is achieved based on the optimization introduced in § 4.2 which reduces the complexity for route insertion from $O(mn)$ to $O(n)$.

We then evaluate the performance of DSR routing module when there are 1K VRFs and each VRF has 8 neighbors. It is notable that the traditional commodity router deployed in our system can not support 1K VRFs. We record the time to advertise, withdraw and report different amounts of routes to the controller. As shown in Figure 13, the time increases moderately when the amount of routes increases, which demonstrates excellent scalability of the home-made BGP speaker. This performance gain mainly stems from the BGP optimization techniques introduced in § 4.2.

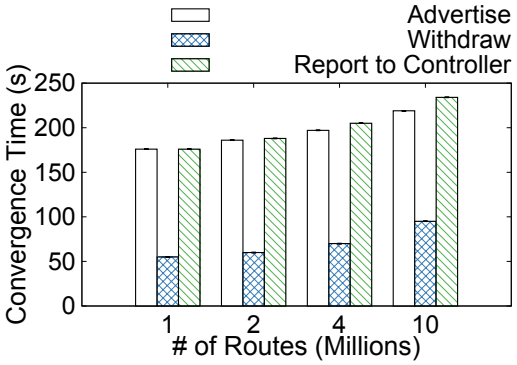
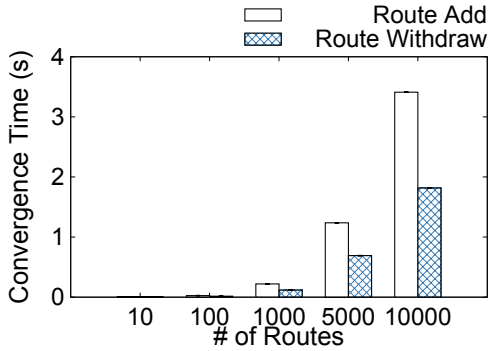
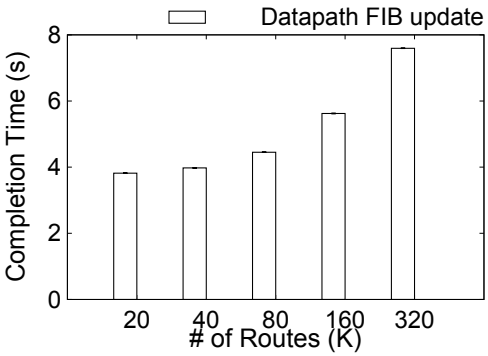


Figure 13: BGP performanec under many VRFs



(a) Performance for handling reported BGP routes



(b) Performance for routing configuration to data path

Figure 14: SDN controller performance for BGP processing

7.3 Control Module Evaluation

The control module is responsible for computing FIB based on the routes collected from the routing module and installs the FIB entries to the datapath forwarding plane. We evaluate the effectiveness of control module for BGP routing configuration. As shown in Figure 14(a), it takes less than 200ms to add or withdraw 1K routes. For adding and withdrawing 10K routes, it takes about 3.4 seconds and 1.8 seconds respectively. More VRFs will add a little more overhead during the VRF insertion process. We then evaluate the control module's performance for updating the FIB entries of the underlying forwarding module. As shown in Figure 14(b), it takes ∼7.5 seconds to configure 320K routes, which meets our needs in most cases.

8 Operational Experiences

The DSR system has been in production for over three years and deployed at over 30 sites replacing the commodity routers there. We has delivered tens of new features with a release cycles of 2∼4 weeks. In contrast, the typical commodity router based solution releases new feature in several months or years. As a result, DSR can save an order of magnitude of monetary costs compared with the traditional solution. Now we introduce the efforts we have taken to operate DSR.

8.1 vNetVerifier System

DSR consists of multiple components built on x86 servers. During the operation, we found that x86 servers are not as reliable as commodity router. Despite the techniques we have used to improve reliability, we need a health monitoring system to discover and locate the failure. We have proposed a virtual network verification system, *i.e.* vNetVerifier, to solve the problem. We discuss some useful features of the system.

Transparent monitoring plane. We aim to achieve active monitoring without interfering with the customer traffic. For this purpose, we have created multiple virtual testing tenants and added monitoring rules in corresponding components. They act like real tenants using the same control and data paths as normal cloud tenants. We generate probing traffic continuously to monitor system faults.

Fine-grained probing. We carry out fine-grained datapath probing at single server granularity and single processing core granularity. As the datapath servers of the same forwarding fleets share the same Virtual IP using ECMP hashing, to monitor a targeted server, we craft probing packets with the underlay IP of the targeted server as the destination IP. For single core monitoring, we encode the processing core ID into the probing packets. Then the dispatchers on the datapath server examine the core ID and forward the packet to the targeted processing core. In this way, we achieved fine grained probing at server-level and core-level. As the decoupled components of DSR are connected via multiple paths. A single-path failure may lead to a wrong reaction as discussed in § 5.1. As a result, we intentionally eliminate the effect of single-path failure through multi-path probing similar to the operation introduced in § 5.1.

Online testing. Conventionally, before we update network components, *e.g.* software upgrade or hardware replacement, we test the components intensively in the small-scale testing environment. However, it is hard to mimic the realistic environment to find all the problems. To this end, we carry out online testing by leveraging the testing tenant in our production environment. To be specific, we convert different testing cases into corresponding network configurations and customized probing packets. Then we carry out online testing before and after the network updates. When abnormal traffic

are detected, the ongoing network upgrading will be stopped and reverted. We are also investigating at high fidelity network emulation approaches similar as CrystalNet [26].

8.2 Fault Isolation and Localization

Fault domain isolation. An important principle during our operation is to guarantee that the datapath forwards packets correctly and continuously even when the control module and routing module fails. To achieve that, we need to do fault domain isolation for different components. A fault domain is defined as the "blast range" of a certain system failure, *e.g.* the failure of one BGP module node may affect the routing decision of a datapath forwarding node which may drop user traffic. The disaggregated design and deployment of control module, routing module and forwarding module makes it possible to isolate different fault domains. Since the control module stores all the routing information in a persistent database, if the BGP modules fail, the datapath can forwards traffic using the control module's routing information.

BGP events and traffic paths analysis. When fault happens, we need sufficient information to identify the root cause. The BGP events and traffic paths are two most important types of information. We have recorded all the BGP events in the database for post-mortem analysis. Since traffic paths can be dynamically changed, we use traceroute like tools to identify the path.

9 Related Work

To overcome the limitation of the traditional BGP protocol, Facebook [35] and Google [41] have developed SDN-based systems to control egress traffic routing for peering edges. Facebook EDGE FABRIC relies on vendor routers, which overrides the routers' normal BGP selection in each individual Points of Presence (PoP). Google Espresso removes the need for commodity BGP routers by employing centralized traffic engineering, which selects egress at distant PoPs. It is notable that DSR faces more complicated scenarios, including not only the peering routing for Internet users, which is the focus of EDGE FABRIC and Espresso, but also inter-connection between public cloud and customers' on-premise datacenters as well as customers' branch offices. Though we share the same idea with Espresso that separating the logic of the control plane from the data plane, DSR targets a different set of challenges on scalability, flexibility and reliability. In order to satisfy the requirements of diverse cloud access scenarios, the proposed disaggregated software router architecture is a general platform that is capable of customizing and scaling each module independently. DSR meets more stringent scalability requirements to support 10Ks VRF tables for PCS and 10M FIB entries for EIS. Therefore, we demonstrated the optimization for scaling the forwarding and routing plane, which is not addressed in Espresso. For flexibility, Espresso introduce the application-aware routing to serve Internet users. In contrast, DSR focuses on fast feature delivery to satisfy the demands of cloud customers, like self-defined routing policy and GRE encapsulation, etc. Furthermore, we solved multiple reliability issues when introducing the disaggregated software-defined router whose components are implemented in a distributed environment, which is not addressed in Espresso either.

NFV technologies are widely used to replace traditional proprietary middle boxes and switching devices. NFV technique reduces cost and provides high feature velocity [17, 24, 25, 30–34, 36, 40]. Previous works focus on several different aspects of the NFV, *e.g.* elastic scaling [22, 34, 40], NFV management [30, 39], performance optimization and implementation [9, 30], etc. DSR is an integration of multiple NFV techniques at cloud gateway. DSR applies DPDK [12] and VPP [8, 14] for the high performance data module. For the BGP speaker, we have evaluated the open source candidates like Quagga [2], GoBGP [4], BIRD [15] and FRR [3]. We developed our high performance BGP speaker based on FRR, which supports multiple functions like IPv6, BFD, etc. For IPSec gateway in the SWS scenario, both the control and data path of the IPSec protocol are coupled on the forwarding fleet. We plan to investigate similar approach as in [38] to separate the control and data plane of the IPSec protocol in order to achieve high scalability.

10 Conclusion

Cloud gateways play a significant role for enterprise customers and Internet users to access the resources in the cloud. With the rapid growth of users, we introduce DSR to replace the commodity routers based cloud gateways in Tencent Cloud. It employs a disaggregated software-defined architecture which provides high scalability and accelerates the features rollout velocity. Meanwhile, integration with the SDN technique achieves smart traffic steering and fast failure convergence. DSR has been deployed in production for over 3 years and has served tens of Tbps traffic for our customers.

Acknowledgments

The authors thank the anonymous reviewers and our shepherd, Nandita Dukkipati, for providing valuable feedback. We thank the teams at Tencent for their contributions to the work, including but not limited to, Pan Feng, Quan He, Haiyang Tan, Qingyan Yu, Bo Wei, Changling Tang, Da Hu, Jianchao Lv, Jiangbo Wang, Jiexuan Cui, Kejin Zuo, Lingyu Zhang, Mengfan Dai, Ruiqiang Dai, Xiaolong Si, Yanxiong Chen, Zhengbin Wang, Zhengzhong Yu, Zhi Tan, Zhichao Hou, Zhifang Wu, Zhiyi Qiu, Zicheng Yuan and Yachen Wang.

References

[1] DPDK documentation, LPM library. https://doc.dpdk.org/guides/prog_guide/lpm_lib.html.

[2] GNU Quagga Project. http://www.nongnu.org/quagga/, 2010.

[3] FRRouting (FRR). https://frrouting.org, 2017.

[4] GoBGP: BGP implementation in Go. https://github.com/osrg/gobgp, 2019.

[5] Cisco ASR 9000 series aggregation services routers, https://www.cisco.com/c/en/us/products/routers/asr-9000-series-aggregation-services-routers/index.html, Mar 2020.

[6] Trident4 / BCM56880 series, high-capacity strataxgs trident 4 Ethernet switch series, 2020.

[7] Loa Andersson and George Swallow. RFC3468: The multiprotocol label switching (MPLS) working group decision on MPLS signaling protocols, 2003.

[8] David Richard Barach and Eliot Dresselhaus. Vectorized software packet forwarding, June 14 2011. US Patent 7,961,636.

[9] Anat Bremler-Barr, Yotam Harchol, and David Hay. OpenBox: a software-defined framework for developing, deploying, and managing network functions. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM)*, pages 511–524, 2016.

[10] CISCO. IPSLA. http://www.cisco.com/en/US/products/ps6602/products_ios_protocol_group_home.html.

[11] Michael Dalton, David Schultz, Jacob Adriaens, Ahsan Arefin, Anshuman Gupta, Brian Fahs, Dima Rubinstein, Enrique Cauich Zermeno, Erik Rubow, James Alexander Docauer, et al. Andromeda: Performance, isolation, and velocity at scale in cloud network virtualization. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 373–387, 2018.

[12] Intel DPDK. Data plane development kit. https://www.dpdk.org/, 2014.

[13] Dino Farinacci, Tony Li, Stan Hanks, David Meyer, and Paul Traina. Generic routing encapsulation (GRE). *RFC*, 2784, 2000.

[14] FD.io. VPP. https://wiki.fd.io/view/VPP.

[15] Ondrej Filip, L Forst, P Machek, M Mares, and O Zajicek. BIRD internet routing daemon. *NANOG-48, Austin, TX*, 2010.

[16] Daniel Firestone, Andrew Putnam, Sambhrama Mundkur, Derek Chiou, Alireza Dabagh, Mike Andrewartha, Hari Angepat, Vivek Bhanu, Adrian Caulfield, Eric Chung, et al. Azure accelerated networking: Smartnics in the public cloud. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 51–66, 2018.

[17] Aaron Gember-Jacobson, Raajay Viswanathan, Chaithan Prakash, Robert Grandl, Junaid Khalid, Sourav Das, and Aditya Akella. OpenNF: Enabling innovation in network function control. *ACM SIGCOMM Computer Communication Review*, 44(4):163–174, 2014.

[18] Albert Greenberg, James R. Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A. Maltz, Parveen Patel, and Sudipta Sengupta. Vl2: A scalable and flexible data center network. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM)*, SIGCOMM '09, page 51–62, New York, NY, USA, 2009. Association for Computing Machinery.

[19] Stan Hanks, David Meyer, Dino Farinacci, and Paul Traina. Generic routing encapsulation (gre). 2000.

[20] Thomas E Hart, Paul E McKenney, Angela Demke Brown, and Jonathan Walpole. Performance of memory reclamation for lockless synchronization. *Journal of Parallel and Distributed Computing*, 67(12):1270–1285, 2007.

[21] Stephen Ibanez, Gordon Brebner, Nick McKeown, and Noa Zilberman. The P4->NetFPGA workflow for line-rate packet processing. In *Proceedings of ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 1–9, 2019.

[22] Murad Kablan, Azzam Alsudais, Eric Keller, and Franck Le. Stateless network functions: Breaking the tight coupling of state and processing. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 97–112, 2017.

[23] Dave Katz, Dave Ward, et al. Bidirectional forwarding detection (BFD). RFC 5880, June, 2010.

[24] Sameer G Kulkarni, Wei Zhang, Jinho Hwang, Shriram Rajagopalan, KK Ramakrishnan, Timothy Wood, Mayutan Arumaithurai, and Xiaoming Fu. Nfvnice: Dynamic backpressure and scheduling for nfv service chains. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM)*, pages 71–84, 2017.

[25] Bojie Li, Kun Tan, Layong Luo, Yanqing Peng, Renqian Luo, Ningyi Xu, Yongqiang Xiong, Peng Cheng, and Enhong Chen. ClickNP: Highly flexible and high performance network processing with reconfigurable hardware. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM)*, pages 1–14, 2016.

[26] Hongqiang Harry Liu, Yibo Zhu, Jitu Padhye, Jiaxin Cao, Sri Tallapragada, Nuno P Lopes, Andrey Rybalchenko, Guohan Lu, and Lihua Yuan. Crystalnet: Faithfully emulating large production networks. In *Proceedings of Symposium on Operating Systems Principles (SOSP)*, pages 599–613, 2017.

[27] Mallik Mahalingam, Dinesh G Dutt, Kenneth Duda, Puneet Agarwal, Lawrence Kreeger, T Sridhar, Mike Bursell, and Chris Wright. Virtual extensible local area network (VXLAN): A framework for overlaying virtualized layer 2 networks over layer 3 networks. *RFC*, 7348:1–22, 2014.

[28] Rui Miao, Hongyi Zeng, Changhoon Kim, Jeongkeun Lee, and Minlan Yu. Silkroad: Making stateful layer-4 load balancing fast and cheap using switching ASICs. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM)*, pages 15–28, 2017.

[29] Vladimir Olteanu, Alexandru Agache, Andrei Voinescu, and Costin Raiciu. Stateless datacenter load-balancing with beamer. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 125–139, 2018.

[30] Shoumik Palkar, Chang Lan, Sangjin Han, Keon Jang, Aurojit Panda, Sylvia Ratnasamy, Luigi Rizzo, and Scott Shenker. E2: a framework for nfv applications. In *Proceedings of Symposium on Operating Systems Principles (SOSP)*, pages 121–136, 2015.

[31] Aurojit Panda, Sangjin Han, Keon Jang, Melvin Walls, Sylvia Ratnasamy, and Scott Shenker. Netbricks: Taking the v out of NFV. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 203–216, 2016.

[32] Ben Pfaff, Justin Pettit, Teemu Koponen, Ethan Jackson, Andy Zhou, Jarno Rajahalme, Jesse Gross, Alex Wang, Joe Stringer, Pravin Shelar, et al. The design and implementation of Open vSwitch. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 117–130, 2015.

[33] Shriram Rajagopalan, Dan Williams, and Hani Jamjoom. Pico replication: A high availability framework for middleboxes. In *Proceedings of Annual Symposium on Cloud Computing*, pages 1–15, 2013.

[34] Shriram Rajagopalan, Dan Williams, Hani Jamjoom, and Andrew Warfield. Split/merge: System support for elastic execution in virtual middleboxes. In *10th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 227–240, 2013.

[35] Brandon Schlinker, Hyojeong Kim, Timothy Cui, Ethan Katz-Bassett, Harsha V Madhyastha, Italo Cunha, James Quinn, Saif Hasan, Petr Lapukhov, and Hongyi Zeng. Engineering egress with edge fabric: Steering oceans of content to the world. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM)*, pages 418–431. ACM, 2017.

[36] Vyas Sekar, Norbert Egi, Sylvia Ratnasamy, Michael K Reiter, and Guangyu Shi. Design and implementation of a consolidated middlebox architecture. In *9th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 323–336, 2012.

[37] Upendra Bhalchandra Shevade, Kyle Benjamin Schultheiss, and Gregory Rustin Rogers. Scalable routing service, June 4 2019. US Patent App. 14/274,534.

[38] Jeongseok Son, Yongqiang Xiong, Kun Tan, Paul Wang, Ze Gan, and Sue Moon. Protego: Cloud-scale multitenant ipsec gateway. In *USENIX Annual Technical Conference (ATC)*, pages 473–485, 2017.

[39] Chen Sun, Jun Bi, Zhilong Zheng, Heng Yu, and Hongxin Hu. Nfp: Enabling network function parallelism in nfv. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM)*, pages 43–56, 2017.

[40] Shinae Woo, Justine Sherry, Sangjin Han, Sue Moon, Sylvia Ratnasamy, and Scott Shenker. Elastic scaling of stateful network functions. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 299–312, 2018.

[41] Kok-Kiong Yap, Murtaza Motiwala, Jeremy Rahe, Steve Padgett, Matthew Holliman, Gary Baldus, Marcus Hines, Taeeun Kim, Ashok Narayanan, Ankur Jain, et al. Taking the edge off with espresso: Scale, reliability and programmability for global internet peering. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM)*, pages 432–445. ACM, 2017.

[42] Randy Zhang and Micah Bartell. *BGP design and implementation*. Cisco Press, 2003.