# Protego: Cloud-Scale Multitenant IPsec Gateway

Jeongseok Son, *KAIST, Microsoft Research;* Yongqiang Xiong, *Microsoft Research;*
Kun Tan, *Huawei;* Paul Wang and Ze Gan, *Microsoft Research;* Sue Moon, *KAIST*

https://www.usenix.org/conference/atc17/technical-sessions/presentation/son

This paper is included in the Proceedings of the
2017 USENIX Annual Technical Conference (USENIX ATC ’17).

July 12–14, 2017 • Santa Clara, CA, USA

ISBN 978-1-931971-38-6

# Protego: Cloud-Scale Multitenant IPsec Gateway

Jeongseok Son[†⋆], Yongqiang Xiong[⋆], Kun Tan[‡], Paul Wang[⋆], Ze Gan[⋆], Sue Moon[†]
[⋆]*Microsoft Research,* [†]*KAIST,* [‡]*Huawei*

## Abstract

Virtual cloud network services let users have their own private networks in the public cloud. IPsec gateways are growing in importance accordingly as they provide VPN connections for customers to remotely access these private networks. Major cloud providers offer IPsec gateway functions to tenants using virtual machines (VMs) running a software IPsec gateway inside. However, dedicating individual IPsec gateway VMs to each tenant results in significant resource waste due to the strong isolation mechanism of VMs.

In this paper, we design *Protego*, a distributed IPsec gateway service designed for multitenancy. By separating the control plane and the data plane of an IPsec gateway, Protego achieves high availability with active redundancy. Furthermore, Protego elastically scales in and out by seamlessly migrating IPsec tunnels between the data nodes without compromising their throughput. Our evaluation and simulation based on production data show that Protego together with a simple resource provisioning algorithm saves more than 80% of the resources compared with allocating independent VMs.

## 1 Introduction

Major cloud providers offer virtual networks as a service to customers so that they can setup their own private network topology in the cloud [1, 8, 4]. Tenants create virtual networks and connect applications running inside virtual machines (VMs) to operate their own distributed services. The ease of management, flexibility and elasticity of a virtual network has driven enterprise customers to extend their existing networks using cloud service in lieu of physical network [29].

To seamlessly incorporate remote virtual networks into existing on-premises networks, tenants establish site-to-site VPN connections between the gateways. For site-to-site VPN connections, IPsec is typically used to have secure communication between on-premises and cloud networks. Hence, cloud providers provide tenants with IPsec gateways in addition to the virtual network service. IPsec gateways in on-premise networks peer with them to initiate IPsec tunnels [9].

It is thus crucial for cloud providers to have a flexible and scalable way to provide IPsec gateway functionality to tenants. The current state of the art is shipping software IPsec gateway to tenants using VMs following the trend of Network Function Virtualization (NFV) [29]. Once a tenant makes a request to create an IPsec gateway, an IPsec gateway VM is dedicated to the tenant. It is a natural approach as VMs are basic resource allocation blocks in cloud environments and provide inherent isolation mechanism.

However, dedicating IPsec gateway VMs to tenants results in significant waste of resource for two reasons. First, VMs exclusively occupy a fixed amount of resource. Hence, cloud providers should over-provision the VMs for peak VPN traffic demand. If a tenant does not utilize all the allocated resource of VMs, the unused portion of it is just wasted. Second, each independent gateway VM needs a high availability (HA) setup, which requires additional redundancy. Since VM startup takes several minutes in the cloud due to resource allocation and data copy [38], a passive standby node is typically introduced for fast failover [6]. If every IPsec gateway requires HA, they capture twice as much resource as they actually need.

These limitations have led us to devise a new IPsec gateway architecture to serve multiple tenants with shared resources. To this end, we propose *Protego*, a cloud-scale software IPsec gateway. We design Protego with the following properties: (1) *multitenancy* to serve multiple tenants without violating the bandwidth requirement of each tenant, (2) *elasticity* to seamlessly scale in and out according to the aggregated traffic demand across tenants, and (3) *high availability* to provide reliable service to users without reserving a passive standby for every active VM.

To achieve both high availability and elasticity, Protego separates the control plane from the data plane. For high availability, the relatively long-lived control plane states are saved to a centralized control node. On the other hand, the data plane state is costly to preserve in the same way since it changes every packet sent and received. Hence, Protego saves it locally in data nodes and quickly reconstruct it via the alive control node in case of failure. Protego migrates tunnels between the data nodes without tearing down an old tunnel through rekeying process. This enables Protego to elastically allocate and de-allocate VMs according to varying IPsec traffic of tenants.

Our evaluation using the prototype implementation presents that Protego can migrate IPsec tunnels even without a transient bandwidth degradation. Based on this seamless tunnel migration, we design a provisioning algorithm to autonomously adjust the amount of resource it subscribes. We show that it is possible to save more than 80% of the resources compared with allocating independent VMs to tenants while meeting the bandwidth guarantee to tenants.

To summarize, we make the following contributions: (1) We present a new architecture of distributed IPsec gateway for the cloud which enables high availability with active redundancy. (2) We devise an IPsec tunnel migration scheme that does not compromise the bandwidth of a tunnel during the migration for elastic resource provisioning. (3) We demonstrate Protego with a simple provisioning algorithm indeed saves significant resources through our evaluation and simulation based on production data.

## 2 Background and Motivation

We first describe why and how IPsec gateways are deployed in cloud environments. Then we identify the necessity of cloud-scale IPsec gateway by showing the resource usage of the IPsec gateways deployed in our data centers. We finally enumerate the requirements of an IPsec gateway for the cloud and challenges of accomplishing it.

### 2.1 Virtual network and site-to-site VPN

The majority of users who purchase the virtual cloud networks are enterprise customers [29]. They use the virtual network services to extend their on-premises network into the cloud. Since virtual networks provide customers with private IP address space, they can seamlessly move their corporate network to the cloud to take advantage of the flexibility of cloud environments.

To connect a virtual network in the cloud to an existing on-premises network, site-to-site VPN is typically used. Site-to-site VPN remotely connects the entire networks from one another over the public Internet. The VPN connection is established between two VPN gateways. Then they encapsulate outbound traffic and decapsulate inbound traffic rather than individual hosts do so.

### 2.2 IPsec gateway

IPsec is a de-facto standard for site-to-site VPN connections. IPsec ensures secure communication between the peers by authenticating and encrypting IP packets. For site-to-site VPN, an IPsec gateway encapsulates the entire packet to create a virtual hop, an *IPsec tunnel*, between the peer gateways.

IPsec primarily consists of two protocols: Internet Key Exchange (IKE) and Encapsulating Security Payload (ESP)[1]. The main purpose of IKE is to authenticate the peer and setup the shared attributes between the peers for secure communication. A set of those attributes is called a security association (SA). IKE protocol is used to settle those SAs. ESP protocol encrypts packets to provide confidentiality, integrity and data origin authenticity using negotiated symmetric keys.

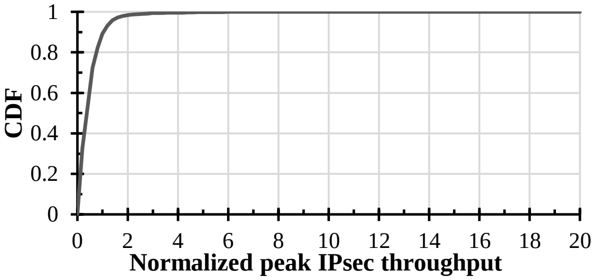

**Figure 1:** CDF of the peak IPsec throughput of data centers

When an IPsec tunnel is established, initial message exchanges first generate an IKE SA for the peers, which contains a shared key and a cipher suite used to encrypt bidirectional IKE traffic. The shared attributes for ESP encryption and decryption, called CHILD SA, are negotiated securely via further IKE message exchanges. CHILD SAs are unidirectional so the inbound and outbound ESP traffic are encrypted with a different SA.

### 2.3 Motivation: Inefficient resource usage of IPsec gateway VMs

A prevalent way for cloud providers to deploy IPsec gateways is using VMs running the software implementation of it inside [29]. VMs let them make the best use of their existing commodity server resources and VM management system without installing additional hardware middleboxes. VMs also provide isolated performance for each tenant and can easily scale by dynamically creating or destroying instances.

However, we found that VMs allocated per tunnel underutilizes resources significantly for the following two reasons.

**Exclusive resource allocation.** Once a VM is allocated to a tenant, the resources of the VM becomes exclusively dedicated to the tenant. Thus, even when a tenant does not fully utilize the capacity of an IPsec gateway, the remaining resources of it cannot be used for serving other tenants' demand.

Figure 1 shows the cumulative distribution of the peak aggregated throughput of all IPsec gateways in each of our data centers. The actual bandwidth values are normalized by the maximum bandwidth that a single IPsec gateway supports. In each data center, there are as many

[1] Authentication Header (AH) is an alternative protocol, but ESP is dominantly used for VPN because only ESP provides confidentiality.

IPsec gateway VMs as there are IPsec tunnels established by tenants. However, the daily peak IPsec bandwidth is less than a single gateway VM capacity in approximately 90% of the data centers. It indicates that most IPsec gateways handle far less traffic than its maximum capacity most of the time.

Even though IPsec gateway VMs have considerable amount of idle resources, there is no easy way to take away the unused resources of VMs for other use. Oversubscribing physical machines with VM consolidation and live migration has been studied as a solution [47, 16, 51]. However, live migration consumes high network bandwidth and easily takes tens of seconds since the whole memory of a VM is iteratively transferred via network [19]. These drawbacks prevent cloud providers from using live migration frequently for flexible resource reallocation.

**Passive standby for high availability.** IPsec gateways should be highly available since the failure directly results in the downtime of the entire virtual network service. High availability (HA) is generally achieved by using more than one nodes to form a cluster. When one node fails, another node in the cluster quickly takes the role of the failed one. Existing hardware and software IPsec gateways form an active/passive cluster, or 1 + 1 redundancy for HA [12, 2]. The cluster synchronizes the IKE state of an active node with a passive node so that the passive node can keep doing stateful processing after failover.

Although adding a passive standby is a straightforward way to achieve HA, passive backups do not participate in processing IPsec traffic. The resources allocated to passive backups are thus just wasted for HA. In the worse case, 50% of resources is devoted for high availability if every gateway VM has a redundant passive standby.

## 2.4 Requirements

To overcome the limitations brought up above, Protego should have the following features:

**Elastic and scalable capacity adjustment.** Protego should be able to save resources without compromising the quality of service of IPsec traffic. It should adjust its capacity by dynamically capturing and releasing resources according to the varying demand of tenants.

**High availability with active redundancy.** High availability is an essential characteristic to meet service level agreement (SLA). For better resource utilization, Protego should achieve HA with active nodes which process the online traffic rather than with passive standby nodes.

**Tunnel performance isolation and guarantee.** To make tenants share a single IPsec gateway service, Protego needs to isolate the performance of each IPsec tunnel of tenants so that aggressive users cannot affect the other ones.

## 2.5 Challenges

A straightforward approach for elasticity and active redundancy is to form a cluster of nodes. Instead of dedicating individual gateway to a tenant, a cloud provider may install the cluster which consists of software or hardware IPsec gateways behind a load balancer and let it process IPsec traffic of multiple tenants. However, the stateful processing of IPsec gateways raises challenges of meeting the requirements in § 2.4 using existing IPsec gateways.

**Migrating tunnels without throughput degradation.** To elastically adjust the cluster size, a cloud provider should have a means to move the workload between the gateways. A strawman approach is to simply tearing down an existing IPsec tunnel and establish a new one in another gateway. However, this approach leads to significant throughput degradation since the gateways cannot process traffic during the tunnel setup, which requires several sequential round trips of packets. To avoid or alleviate this issue, we should determine how to migrate or share state associated with a tunnel between gateways and when to redirect the packets belong to a tunnel.

**Deciding on the right amount of resources to reserve.** We need to carefully decide on the amount of resources to reserve due to the latency of spinning up new VMs, which takes several minutes in the major cloud services [38]. Protego would easily violate the performance guarantee until new VMs are added, if it reserves too little resources to save them. On the other hand, it would waste resources if it subscribes too much. Therefore, we should devise a way to determine the proper amount of resources to subscribe in order to save resources while meeting the bandwidth requirement of tenants.

**Optimizing the packet processing performance.** IPsec packet processing is computationally intensive since it involves encryption and decryption of the payload. To maximize the throughput of Protego, it is crucial to parallelize packet processing using multiple cores. However, IPsec gateways maintain ESP packet counters to include a sequence number in ESP packets for the anti-replay feature [33]. In order to ensure that the sequence number is not reused, a simple method is to make packet processing threads share a global packet counter for each tunnel and update it every packet sent. This approach requires locking, however, which decreases parallelism in packet processing significantly. Hence, it is unsuitable to achieve multiple Gbps per-tunnel throughput we aim to offer.

# 3 Protego Core Ideas

Protego meets the requirements of a cloud-scale IPsec gateway described in the previous section based on the following key ideas.

### 3.1 Separation of control and data planes

In traditional software IPsec gateway implementations, IKE, ESP modules and pertinent state are consolidated into a single node. Each member of IPsec gateway cluster thus has separate IKE module and state.

We propose a separate control node which incorporates the signaling plane of gateways into a single node. The control node deals with traffic steering and dynamic provisioning of the data plane. The data plane of Protego consists of a cluster of VMs and focus on packet encryption and decryption process. By this separation, each plane manages its state to deal with different access patterns and focuses on ensuring different properties.

**Keeping control plane state in a central node.** Recovering IKE state is costly since re-negotiating an IKE SA takes several sequential round trip of messages [32]. On the other hand, it is updated infrequently, every tens of seconds or every couple of minutes, when it receives heartbeat messages from a peer gateway.

Protego saves control plane state to a centralized control node exploiting this relaxed update frequency. By saving the state to a centralized store every time it is updated, Protego achieves tunnel migration without stopping processing traffic.

**Quick recovery of data plane state.** ESP data nodes handle data packets to encrypt and decrypt them. An ESP packet counter is updated per-packet basis, which makes it infeasible to store ESP state separately as Protego does for IKE state. However, ESP SA can be initiated in 1 RTT if the IKE SA is alive. Hence, The Protego control plane just re-negotiates the ESP state during failover.

### 3.2 Seamless tunnel migration by rekeying

The key enabler of elasticity is seamless migration of workload. Protego is able to migrate existing IPsec tunnels from one ESP node to another one leveraging rekeying process of IPsec [32] without impairing their throughput.

IPsec gateways use keys for a limited amount of time. Before a SA expires, a gateway negotiates a new key with its peer. This process is referred to as rekeying. Rekeying is done in parallel without collapsing the old SA. Because Protego has a global signaling plane, it can insert a new key to any data node which will receive a migrated tunnel. Protego seamlessly steers the traffic using software load balancers tailored to IPsec protocol.

### 3.3 Elastic provisioning algorithm

VM live migration requires operators to apply complex modeling and prediction techniques [36, 15, 50, 17, 49] to minimize the high overhead of live migration. In contrast, we devise a straightforward resource provisioning algorithm leveraging the light-weight and instant migration scheme of Protego. We model the IPsec tunnel placement as a one-dimensional bin packing problem. Solving this problem is not sufficient, however, since we still have to consider the long latency of spinning up VMs. To precisely estimate the amount of resources to subscribe in advance, Protego keeps track of the resource usage distribution of IPsec tunnels and calculate the convolution of these distributions. We will describe those algorithms together in detail in § 5.

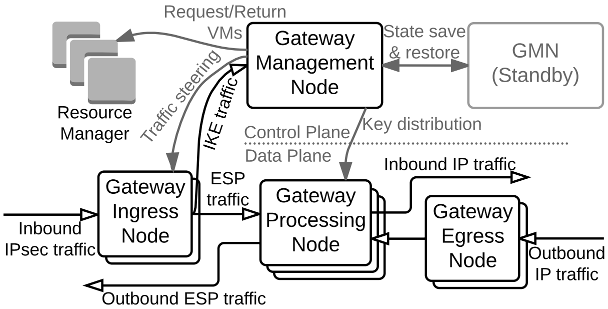

**Figure 2:** Protego architecture and data flow overview

## 4 System Design

We present how we design Protego with the core ideas in § 3 to satisfy the requirements enumerated in § 2.4

### 4.1 Architecture Overview

Protego has separate control plane and data plane. The control plane consists of Gateway Management Node (GMN), a controller which handles IKE traffic and decides the amount of resources to reserve. It also steers ESP traffic by inserting forwarding rules. The data plane consists of a set of Gateway Processing Node (GPN) which processes ESP packets. Gateway Ingress Node (GIN) or Gateway Egress Node (GEN), which are software load balancers tailored for Protego, exposes external virtual IP addresses (VIP) and forward the traffic destined to VIPs to an appropriate node. It also limits the bandwidth of each tunnel for performance isolation. Figure 2 shows the overall architecture of Protego.

### 4.2 Control Plane: Gateway Management Node

**IKE packet processing.** GMN processes the IKE traffic of IPsec tunnels. As we discussed above, the main role of IKE is to negotiate SAs that include a cipher suite, and materials to generate symmetric keys with its peer gateway. We do not elaborate on the protocol details, which can be found at RFC5996 [32].

Once a shared symmetric key for ESP encryption is created, GMN distributes this key to one of the nodes in the data plane. Then it adds a rewrite rule to a GIN(Gateway Ingress Node) and GEN(Gateway Egress

Node) to steer the corresponding ESP traffic to a GPN.

Whenever GMN processes a packet, it saves updated IKE SAs to the standby GMN. In case of failure, the standby node takes over the role of the active GMN node. GIN is responsible for detecting the failure of GMN by monitoring heartbeat messages and steering IKE packets to the standby node after a failover.

**Resource management.** Another important role of GMN is adjusting the number of GPNs in the data plane. When the traffic increases, GMN adds more VMs to the ESP node pool and move some existing tunnels to the new ESP node and vice versa. GMN monitors the CPU utilization of every GPN periodically. When a GPN sends a tunnel migration request to balance the load, GMN selects an appropriate node which can receive the tunnels. If there is no nodes that can receive the tunnels, GMN requests additional VMs to the resource manager of a cloud provider. All request and response packets for the resource management are sent and received using TCP for reliable transmission.

**Traffic steering.** When an IPsec tunnel is migrated, GMN inserts appropriate forwarding rules to a GEN and a GIN. This process includes the selection of a GPN which receives an IPsec tunnel to be migrated.

## 4.3 Gateway Ingress and Egress Node

GIN and GEN are analogous to software load balancers, but provide additional features necessary to Protego. We added the following functionalities to Ananta [42], which is a scalable software load balancer with high availability.

**Traffic forwarding.** The major role of GIN and GEN in Protego is directing packets. They rewrite the destination address to the address of a GPN which is selected to process the traffic. GIN exposes an external VIP which the inbound traffic is destined to. For the inbound traffic, GIN should be able to distinguish different tunnel traffic destined to the same IP in order to distribute ESP packets across different GPNs. GIN matches the Security Parameter Index (SPI) of ESP packets for this. For the outbound traffic, GEN simply uses the traffic selector, an ACL (Access Control List)-like filter exchanged when GMN negotiates CHILD SAs.

**Rate limiting.** Another important role of GINs and GENs is limiting the bandwidth of tunnels. One of the requirements of Protego is enforcing per-tunnel performance isolation. Protego achieves this by limiting the rate of tunnels to the maximum bandwidth that cloud provider promise to support to tenants.

**GPN failure detection.** As long as GMN is alive, a peer gateway cannot detect the failure of GPN since GMN keeps transmitting IKE heartbeat messages. GIN and GEN are responsible for detecting the failures. Introducing heartbeat messages is a common technique for this. However, the heartbeat messages with a tiny interval overload the internal network and the detector as the number of nodes grow. Instead of the fixed interval, we want the heartbeat interval of GPNs with higher throughput to be shorter to detect failure more quickly. To do so, GIN/GEN uniformly sample and tag packets to trigger heartbeat messages from GPNs. We describe this process in more detail in § 6.

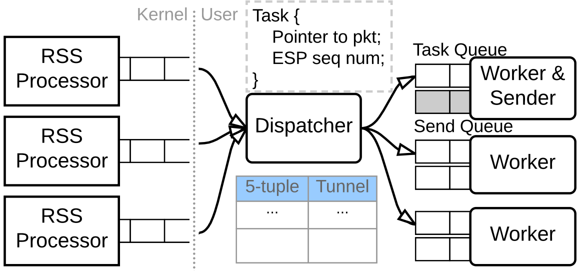

**Figure 3:** GPN design and packet processing flow

## 4.4 Data Plane: Gateway Processing Node

GPNs handle encryption and decryption of all tunnels. GMN decides on the mapping between tunnels and GPNs, and inserts forwarding rules to GIN/GEN accordingly. Each GPN also monitors and reports its resource utilization (CPU, bandwidth, etc.) to GMN periodically. When the utilization exceeds a certain threshold, it sends a tunnel migration request to GMN to change the mapping for load balancing.

To optimize the performance of Protego while guaranteeing the uniqueness of sequence number, we avoid using locks with the design depicted in Figure 3. We pin a *worker* thread to each core for all packet processing tasks and make those worker threads run independently from one another. One special worker thread, *dispatcher*, enforces packet ordering within a tunnel and distributes packet processing tasks across multiple cores. Another special type of worker thread, *sender*, is responsible for sending processed packets in batch.

Note that the dispatcher and sender are also worker threads. They are not completely dedicated to the task dispatching and sending. When all the task queue of other workers are full, the dispatcher puts the task to its own queue and performs encryption or decryption. A worker thread becomes a sender only when its send queue has some enqueued send requests. This design choice is for maximizing encryption and decryption performance by fully utilizing CPU cores under heavy workloads.

## 4.5 Tunnel migration

IPsec tunnel migration is an essential operation for elasticity. Protego leverages rekeying process of IKE to mi-

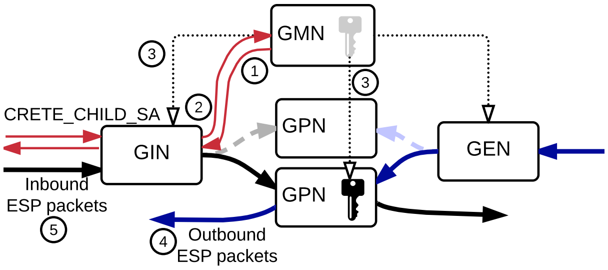

**Figure 4:** Tunnel migration process

grate a tunnel from one GPN to another one. Following is the detailed tunnel migration steps depicted in Figure 4.

1. GMN sends the CREATE_CHILD_SA request with a new Diffie-Hellman (DH) value[2] and a nonce.
2. GMN receives the CREATE_CHILD_SA response which include the DH value and the nonce of a responder. GPN generates two new child SAs using those information for the inbound and outbound tunnels.
3. GMN hands the new SAs over to a GPN which would receive the tunnel to migrate. GMN also adds a corresponding steering rule to GIN and GEN using the SPI and the traffic selector of new SAs known by the CREATE_CHILD_SA exchange.
4. GPN starts to use the new outbound SA. Once the peer gateway receives this traffic of new inbound SA, it starts to use its new outbound SA.
5. GIN steers the ESP packets destined to new inbound SA of Protego to the new GPN. The old inbound SA is no longer used.

The old SAs are not destructed during the migration process, so Protego can seamlessly migrate tunnels without affecting the performance.

## 5 Elastic Resource Provisioning

We present an algorithm to dynamically provision and de-provision the data plane.

### 5.1 Objectives

Our algorithm has two conflicting goals. One is to minimize the resource usage for better efficiency, and the other one is satisfying the throughput requirement of tenants.

Therefore, it is critical for Protego to gauge the minimum amount of resources, or the number of VMs needed to reserve to ensure the per-tunnel performance to tenants. We precisely model the resource requirements and use a bin packing algorithm to figure it out.

| Notation | Explanation |
|---|---|
| $\sigma_i$ | Maximum CPU usage of a tunnel $i$ |
| $\alpha_i$ | Current CPU usage of a tunnel $i$ |
| $\beta_j$ | Current CPU utilization of a node $j$ |
| $U_i$ | Probability distribution of the CPU utilization of a tunnel $i$ |
| $\varepsilon$ | Throughput guarantee violation tolerance |
| $Y$ | Probability distribution of aggregated tunnel CPU utilization |
| $C$ | Number of VMs reserved for Protego |
| $T_H$ | CPU utilization threshold for hotspot detection |

**Table 1:** Variables used in the algorithm description

### 5.2 Model

**Hierarchy of virtual machine.** VM states are classified into roughly three categories in the cloud. *Active VMs* are booted VMs actively used by a service. *Shutdown VMs* are not yet booted and not under control of any one. In addition to those typical states, *Inactive VMs* [45] are booted and under control of a service but reserved for scaling out the service capacity. By introducing the inactive state, cloud providers are allowed to reduce the resources allocated for those inactive VMs.

Each VM group has a different latency to be added to the ESP node pool. Normally, active and inactive VMs are added almost instantly since they are controlled by a service, but shutdown VMs take at least several minutes to be active. If the service does not reserve enough active and inactive VMs, tenants may experience severe performance issue.

**Node capacity.** In IPsec gateways, the CPU resource of nodes is the bottleneck that determines the throughput of IPsec tunnels. We assume that every node has the same CPU resource, and regard all nodes have normalized CPU capacity 1.

**Maximum resource usage.** The maximum CPU usage of tunnels is bounded in our case since we limit the bandwidth of tunnels. We express the maximum limit of the tunnel CPU utilization as a real number $\sigma_i$. $(0 < \sigma_i < 1)$

**Current resource usage of a tunnel and utilization of a node.** The CPU usage of a tunnel is periodically calculated with the interval of $\tau$. Let $\alpha_i$ is the current CPU usage of a tunnel at specific times. Then the CPU utilization of a node is defined as $\beta_j = \sum_{i=1}^{k} \alpha_i$, where $k$ is the number of tunnels in the node.

**Resource usage distribution of a tunnel.** The CPU usage of a tunnel varies over time. The usage distribution of a tunnel $U_i$ takes this into account. $U_i(x)(0 < x < \alpha_i)$ is the probability density function of the CPU usage, which

[2] The Diffie-Hellman value can be excluded complying with the IKEv2 specification. We added it just for stronger guarantees of forward secrecy.

shows the likelihood of how much CPU resource a tunnel would consume at a certain time.

**Violation tolerance.** The violation tolerance $\varepsilon$ expresses how tolerable the system is on the throughput guarantee violation. If the traffic of a tunnel during a certain time interval is not fully served due to the insufficient resources of a GPN, the tunnel fails to achieve demanded throughput of a tenant as packets get dropped. The sum of the time intervals of such time should account for less than $\varepsilon$ of the total available time of Protego.

## 5.3 Minimum number of VMs for per-tunnel throughput guarantee

Based on the model described in § 5.2, we figure out the minimum number of VMs that Protego should reserve to satisfy the IPsec tunnel throughput guarantee to tenants.

**Aggregated traffic distribution.** Let $Y = \sum_{i=1}^{n} U_i$, where $n$ is the number of all tunnels. $Y$ denotes the probability distribution of aggregated CPU usage of all tunnels in the system. Since $U$ is a discrete probability distribution, we can calculate the convolution of any two resource usage distributions using the following formula.

$$Y(z) = \sum_{k=0}^{1} U(k)U(z-k)$$

We use the formula to sum up $n$ resource usage distributions inductively to the $Y$. We assume that the tunnel resource usage distributions are independent from one another.

**Minimum number of VMs for the throughput guarantee.** The throughput guarantee constraint is formally expressed with $Y$ and $\varepsilon$:

$$Pr(Y > C) \leq \varepsilon$$

where $C$ is the total resource of the system. In our case, $C$ is the number of active and inactive VMs because all VMs have the normalized CPU capacity 1. $\varepsilon$ is a given constant and $Y$ is derived from $U_i$. Hence, we can figure out $C$, the number of VMs that Protego needs to reserve to guarantee the throughput.

Protego should keep its number of active and inactive VMs above $C$ so that the probability of the violation is maintained below $\varepsilon$. In a real deployment, however, we need to take the $T_H$ into account since it incurs a small resource waste. Thus, the number of VMs it reserves should be higher than $C/T_H$. We assume the degree of external fragmentation of the capacity of GPNs is negligible here.

## 5.4 Load balancing and tunnel consolidation

Protego detects nodes which the demand of assigned tunnels exceeds its capacity and balance the workload by migrating the tunnels to other relatively idle nodes. At the same time, Protego periodically consolidates tunnels to minimize the number of active VMs.

**Hotspot node detection.** GMN should detect nodes of which the demand of tunnels exceeds its capacity. We set a CPU utilization threshold $T_H > max\ \alpha_i$ and regard a node as hotspot if $\beta_j > T_H$. $T_H$ should be large enough to ensure high utilization of nodes.

**Tunnel migration.** Once the hotspot node is detected, a subset of the tunnels in the node should be migrated to lower $\beta_j$ below $T_H$. To minimize the number of migration, the tunnels are sorted in decreasing order of $\alpha_i$, and largest $k$ tunnels where $\sum_{i=1}^{k} \alpha_i > \beta_j - T_H$ are chosen and migrated in that order. The same Best Fit algorithm is used to choose a node to place each tunnel. The system adds an inactive VM to the active pool if none of the nodes are not able to receive the tunnel.

**Tunnel consolidation.** Protego periodically decides new tunnel allocation based on Best Fit Decreasing (BFD) algorithm, which guarantees to use no more than $11/9$ bins of the optimal solution [25]. Protego periodically sorts all tunnels in decreasing order of $\alpha_i$, and use BFD to figure out a new placement of the tunnels. After every consolidation, Protego makes empty active VMs inactive to minimize the number of active VMs.

# 6 Implementation

## 6.1 GIN & GEN

GIN and GEN are both based on our packet filtering driver based on Windows NDIS Lightweight filter (LWS) driver. The main task of GIN and GEN are modifying the destination IP address of the packets to forward them to a right GPN which possesses the shared keys for the inbound and outbound traffic of a tunnel that packets belong to. For this purpose, GIN and GEN maintain the mappings between SPI and GPN IP addresses, and traffic selectors and GPN IP addresses.

Another important role of GIN and GEN is detecting the failure of GPNs. GIN and GEN manipulate the last bit of TOS field in outer IP header of ESP packets for tagging. They sample a part of packets and set the last bit of TOS to 1. Once a GPN detects the bit is set, it mirrors the packet with the reversed source and destination addresses and empty payload back to GIN/GEN as a heartbeat. When there is no reply within a certain period, the packet is regarded as dropped. After three consecutive drops, GIN or GEN concludes that a corresponding GPN fails.

## 6.2 GMN

We implement GMN based on the existing IPsec service module in the Routing and Remote Access Service

(RRAS) [14]. We add the state backup and recovery logic to the implementation of Remote Access service in Windows Server 2012 R2. Our modified RRAS captures state modification by wrapping global variables with setter functions. Also, public interface is added to expose states and save the changed ones to an external IKE module of the passive GMN. These interfaces are implemented based on asynchronous RPC (Remote Procedure Call) already implemented in Windows Server [10].

## 6.3 GPN

We implement our own filter driver to catch packet receive notifications from NIC and return the address of a free buffer in the Free Buffer Queue. NIC copies received packet data to the buffer, then the filter driver pushes the pointer to the receive queue exposed to user space.

Dispatcher thread maintains an array of buffers to hold the encrypted or decrypted packets to be sent in a batch. It pushes the request to one of the Task Queue of worker threads and a worker encrypts or decrypts the packet data in turn. The worker writes back the process packet to the array of buffer at the assigned index, and increases the processing counter. Once the processing counter reaches the total size of the array size, the sender thread starts to send out the whole buffer. Upon receiving the send completion notification, the buffers are returned to Free Buffer Queue maintained by our filter driver so that it can be reused.

Note that GIN, GEN, GMN and GPN can be implemented independently on top of different platforms although we implemented all of them in Windows servers in our local test bed. They can be built and combined on public clouds by third-party as well if an enterprise tenant wants to deploy their own VPN service.

# 7 Evaluation

The test bed has the same networking and configuration as our real production IPsec gateway environment. The experimental setup consists of 32 servers with 16-core Intel Xeon E5-2650 v2 CPU working at 2.6Ghz and Mellanox Connect-3 Pro 40Gbps NIC. We use Windows Server 2012 R2 and Hyper-V.

Figure 5 shows the topology of the experiment. We use a WAN emulator to emulate latency and packet loss.

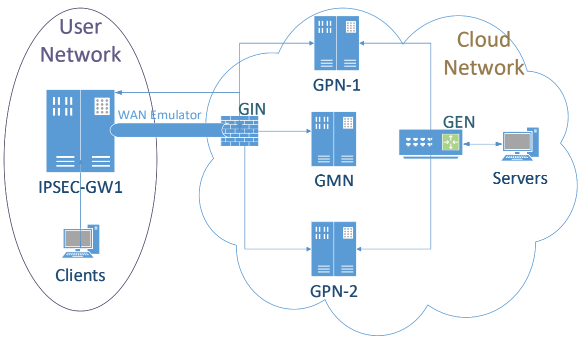

**Figure 5:** Experiment topology

## 7.1 Failover

To evaluate the impact of failures in Protego, we establish an IPsec tunnel between Protego and the IPsec gateway in the user network in our experimental topology. The client sends 300 Mbps of TCP traffic to the server machine. While the client is sending the traffic, we power off GPN and GMN one by one and monitor the throughput at the server side. We set the sampling rate of GIN and GPN for failure detection to 1/1000 and the minimum sampling interval to 10 ms.

We powered off GPN at around 18 second. In Figure 6, the throughput drops slightly as some packets are dropped during the failover period. Once a new ESP key is negotiated and inserted to a new GPN, the throughput recovers to the original value after the TCP slow-start phase. In the GMN failure case, the throughput of the tunnel is not degraded as shown in Figure 6, since CHILD SAs are alive and used for ESP packet processing, and GMN is restored almost instantly.

Figure 8 shows the latency of failover and IKE state update, which we measured running the operations 20 times. It takes 0.28 seconds in total for the failover. The round trip time between peer gateways for re-negotiating a new CHILD SA accounts for 68% of the total failover time. The latency of updating an IKE SA in a passive GMN is 89 ms, which is quick enough to handle IKE heartbeat messages sent every few seconds.

## 7.2 Tunnel migration overhead

We created two GPNs as described in Figure 5 to see the throughput change of an IPsec tunnel during migration. We measured the throughput of a TCP stream in the server.

Figure 7 shows the throughput of the IPsec tunnel over time. We exposed the tunnel migration API to manually initiate the process via command line of GMN. The migration process is started at approximately 18 seconds. The tunnel performance is maintained during the migration process according to the figure. The time it takes to migrate a tunnel is the same as the sum of the rekey and ESP state insertion time mentioned in the failover section.

## 7.3 GPN performance

**Multi-core throughput.** In order to measure the performance and multi-core scalability of GPN, we establish a single IPsec tunnel between the IPsec gateway in the user network and one of the GPNs of Protego. To measure the encapsulation performance, the server sends

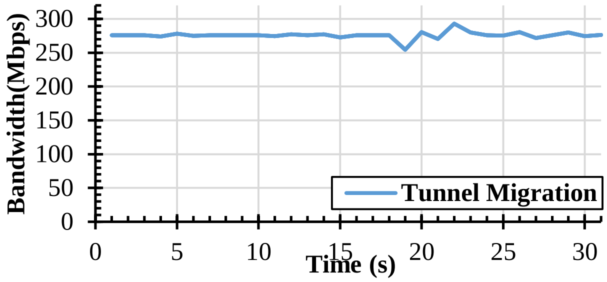

**Figure 6:** Impact of failure on tunnel throughput

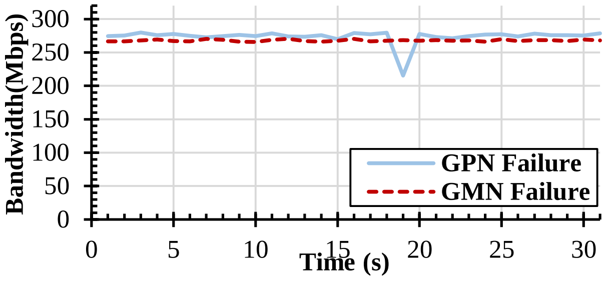

**Figure 7:** Impact of migration on tunnel throughput

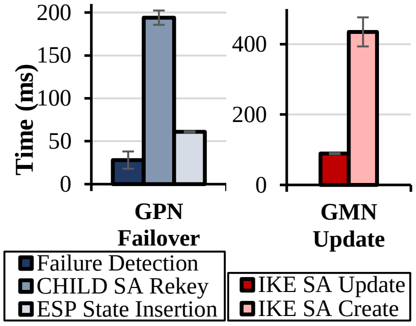

**Figure 8:** Failover and state update latency breakdown

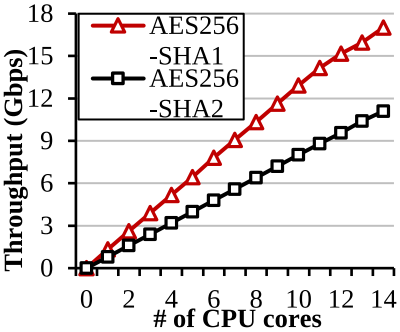

**Figure 9:** Single node throughput

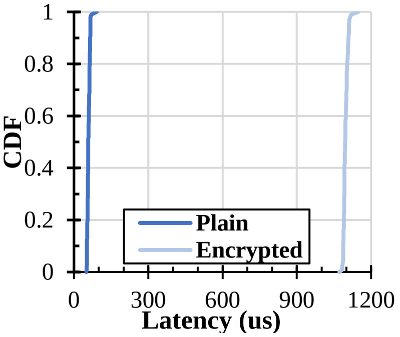

**Figure 10:** CDF of latency per packet

TCP traffic to a client. The TCP packet length is 1400 bytes. We used a number of TCP connections to fully saturate the CPU resource of the GPN.

Figure 9 shows the throughput of an IPsec tunnel measured in the server using the aggregated TCP throughput. As the number of CPU cores increases, the throughput of a single tunnel performance of a single GPN increases linearly. Protego can provide 10 Gbps of the throughput with 8 cores when AES256-CBC is used for encryption and SHA1 is used for integrity. When SHA2 is used for integrity, more than 12 cores is required to achieve 10 Gbps in our evaluation setup.

**Packet processing latency.** We also measured latency added by GPN node. To quantify the latency incurred by a GPN node, we measure the latency of packets which only pass through GIN and skip GPN, and then that of packets processed by a GPN to encrypt them with AES256CBC-SHA1. A client sends 1400 bytes TCP packets of which payload contains timestamp value. A server which receives the packet prints out the latency based on the embedded timestamp. We turned off WAN emulator in this evaluation and place all VMs in the same rack.

We sampled 1,000 packets to draw CDF graph in Figure 10. The deviation of latency distribution is quite small as they are connected by a single ToR. The median value of the case when GPN is not involved is 61 us, and is 1094 us when GPN is involved. The latency overhead of Protego is around 1 ms. It is negligible compared with RTT of WAN, which is tens or hundreds of ms in general.

## 7.4 Resource provisioning simulation

We evaluate the algorithm elaborated in § 5 by doing a large-scale simulation. We use the throughput data of IPsec tunnels in our data centers to figure out how much resource is saved by Protego compared with the existing VM allocation based system. We collected the hourly average throughput of IPsec tunnels for 24 hours. We divide the actual tunnel throughput values by the maximum capacity of deployed IPsec gateways. Then we multiply the resulting ratio by an arbitrary maximum tunnel throughput we choose for simulation.

**Resource saving.** We collected the 1-day throughput data of IPsec gateways in one of our data centers. The average throughput of the tunnels is measured every minute. We assume that all GPNs have the same processing capacity, and all ESP packets with the same size consume the same amount of CPU resource when processed.

The throughput trace of 170 tunnels was collected and used in our simulation. We normalized the maximum tunnel throughput to 1.5 Gbps, which is the maximum tunnel throughput supported by major cloud providers [3, 5]. The GPN capacity is set to 5 Gbps. ($\sigma_i = 0.3$) The hotspot threshold $T_H = 0.90$ and the throughput measurement interval is 1 minute. Also, the violation tolerance $\varepsilon = 0.95$.

Figure 11 displays illustrates the aggregated IPsec throughput of all tunnels and the total capacity of active

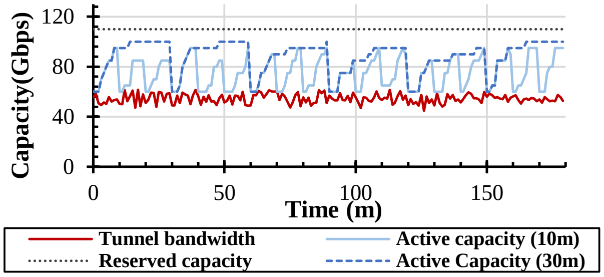

**Figure 11:** Resource provisioning efficiency

VMs used as GPNs. The reserved capacity represents the total resource of all VMs that Protego reserves by figuring out the minimum number of VMs it needs for bandwidth guarantee based on the formula explained in § 5.3. In this simulation, the reserved capacity is 110 Gbps since Protego subscribes 22 VMs.

The consolidation interval is set to 10 minutes and 30 minutes respectively. The number of active VMs grows between the consolidation points since Protego balances the IPsec workload by migrating the tunnels as their throughput are fluctuating. The number of active VMs shrinks every consolidation interval.

It is trivial from the figure that the smaller the consolidation interval is, the less active VMs the Protego utilizes. The average provisioned capacity of active VMs is 65.38, 74.75 Gbps, and 88.17 Gbps for the 5-minute, 10-minute, and 30-minute consolidation intervals respectively. The average total throughput of the tunnels is 57.49 Gbps. The trade off of finer consolidation interval is investigated using the result in the next subsection.

**Throughput guarantee.** Another important requirement of a resource provisioning algorithm is to meet the throughput guarantee. We introduce *daily bandwidth guarantee* to measure how much time Protego actually provisions enough resources in a similar way as availability SLAs are defined.

$$DailyBandwidthGuarantee(\%) = \frac{TotalAvailableMinutes - MinutesOfViolation}{TotalAvailableMinutes}$$

The violation happens when the sum of the demand bandwidths of IPsec tunnels, which are rate limited, exceeds the capacity of a GPN. We assume that the packet scheduler of GPNs is completely fair so the bandwidth guarantee is violated only when its bandwidth demand is larger than its fair share. In public clouds, only the availability of VPN services are guaranteed [7, 13]. Cloud providers seldom guarantee the bandwidth in the SLA in any form [39]. We suggest the bandwidth guarantee to briefly show the trade-off between the utilization and the QoS with different consolidation intervals. We do not determine the optimal parameters of our algorithm here, which will be different depending on the internal performance indicators of each cloud provider.

| Consolidation Intervals | 3 min | 5 min | 10 min | 30 min | 60 min |
|---|---|---|---|---|---|
| Active VM Capacity (Gbps) | 61.23 | 66.17 | 73.97 | 88.34 | 93.22 |
| 99th-percentile Guarantee (%) | 90.21 | 93.07 | 96.84 | 98.24 | 98.63 |
| Resource Saving (%) | 88.00 | 87.03 | 85.50 | 82.68 | 81.72 |

**Table 2:** Bandwidth guarantee and resource saving achieved with different consolidation intervals

Table 2 contains the detailed numbers we get from the simulation. The resource saving is calculated by dividing the capacity of active VMs by the total capacity of VM assuming that one VM is dedicated to each tunnel. Since there are 170 tunnels of which maximum bandwidth is 1.5 Gbps, the total capacity is 255 Gbps to provision for peak demands. Moreover, the high availability requirement doubles the number of necessary VMs. Therefore we figure out the total capacity required for the old system is 510 Gbps. When the consolidation interval is 10 minutes, Protego can save around 85.50% of VM resource while meeting the bandwidth guarantee of 99 % of the tunnels for 96.84 % of the total available time of Protego.

## 8 Discussion

**Security implication.** One may argue that the security of the overall system is weakened due to the risk of placing secret keys in a shared VM, GMN. However, the VMs of Protego are not leased to tenants but are under control of cloud providers. They can block external network access to those control nodes as they normally do for their internal servers. Note that Protego performs complete IPsec protocol as it is. Rekeying process for migration may incur some overhead but does not compromise security.

**Keeping occasionally changed state in a centralized node.** We make Protego keep the IKE state in a centralized node. Likewise, the same approach could be applied to other NFs to make the data plane stateless. For example, asset monitoring systems such as PRADS [11] employ fingerprints to identify clients. Since they are rarely changed, storing them in a centralized node would be a good way to build a scalable monitoring system.

## 9 Related Work

**Software NFs for the cloud.** Flexible and easy to manage software NFs are becoming more prevalent in data centers these days [42, 21, 23, 24, 22]. Especially, software load balancers are deployed and replacing hardware ones. Ananta [42] is the first software load balancer specially designed for cloud environments. Ananta has a separate control plane and data plane. Yoda [23] decouples the flow state from load balancers and stores it in a persistent storage for high availability. Ananta and Yoda have influenced the design of Protego. Maglev [21] is a software load balancer, further optimized for the throughput of a single machine. Maglev employs a forwarder thread which calculates the 5-tuple hash of the packets and put them into the receiving queue of a dedicated packet rewriter thread. The dispatcher thread of Protego plays a similar role to the forwarder and steering thread. However, the data plane design of Protego is different from Maglev and other packet processing frameworks [20, 30, 34, 35, 28] in that it is specially designed for IPsec. Protego takes the state dependency between ESP packets into account and enables even the packets belong to the same tunnel distributed across multiple worker threads.

**NFV Frameworks for scalability and availability.** OpenNF [27] controller manages both the forwarding rules of SDN controller and the internal state of NFs to migrate flows from one NF instance to another. OpenNF controller buffers the packets of the flow in migration until the corresponding per-flow state is moved, which adds hundreds of milliseconds of per-packet latency. U-HAUL [37] selectively apply the OpenNF migration scheme to elephant flows to optimize the migration performance. Unlike these controllers, Protego achieves loss-free migration without migrating the per-tunnel state by leveraging the rekeying feature of IKE protocol.

E2 [40], Stratos [26], and OpenBox [18] are frameworks that provide high-level means of developing, placing and scaling NFs by introducing an NF-agnostic controller. We want to point out that employing an NF-specific controller is often necessary and efficient as we have shown. StatelessNF [31] uses a low-latency data store to make NFs stateless for scalability and high availability. Protego has a similarity to StatelessNF in that it stores state in a centralized node, but Protego maintains the frequently changing state locally. NetBricks [41] is a framework built on Rust to ensure safe memory isolation in user-level for NFs without VMs. Unlike NetBricks, we still use VMs for cloud providers to leverage existing resource allocation and management system based on virtualization platforms. FTMB [44] and Pico Replication [43] leverage VM checkpoint or snapshot to ensure high availability of middleboxes. We avoid checkpoint-based approaches since VM restore time in the cloud easily takes several minutes [38], which is too long to meet the tight availability SLA of cloud providers.

We want to emphasize that the efficiency improvement has been achieved by taking multitenancy into account when designing our system. The NFs for cloud environments should have a means to seamlessly and quickly migrate workloads and resource allocation/deallocation policy to elastically adapt to varying demand of tenants.

**Resource provisioning in shared environment.** Deciding the right amount of resources to provision in a shared environment has been a long-standing problem. Urgaonkar et al. [46] shows the gain of oversubscribing the resource in a shared hosting platform. They profile the resource usage of applications offline with realistic workloads. Based on those resource usage distributions, the system decides the capacity of resources. We have adopted the resource usage model of them to decide the minimum number of VMs to reserve.

In the context of cloud computing, VM placement and migration is one of the most widely studied areas. Sandpiper [49] leverages VM live migration to balance the workload of overloaded physical machines. Bobroff et al. [17] takes SLA into account and design a forecasting technique to minimize SLA violation. Verma et al. [47] considers dynamic VM resizing to efficiently consolidate VMs with less frequent migration. Though there has been a large volume of literature along this line, the high network bandwidth consumption of live migration and long migration time of hotspot VMs [48] hinder the wide deployment of it. Protego provides a migration scheme which is far more lightweight and quicker than VM live migration. In addition, it provides the finer granularity of load balancing by migrating tunnels instead of an entire VM. Therefore, it has a better potential for real production use.

## 10 Conclusion

We have described Protego, a software IPsec gateway specifically designed for cloud environments. Protego serves multiple tenants using shared resources for statistical multiplexing. It separates the control plane from existing IPsec gateways and preserves its state for high availability. We leverage IKE rekeying feature to seamlessly migrate tunnels without impairing their throughput. We devise a resource provisioning algorithm and demonstrate that Protego can save more than 80% of the resources comparing with existing approach, while guaranteeing the IPsec throughput for higher than 90% of uptime.

## Acknowledgements

We thank our shepherd Ittay Eyal and the anonymous reviewers for their helpful feedback. We are also grateful to Shinae Woo and Bojie Li for their thoughtful comments on the drafts, and Joongi Kim for providing his figure templates. This work was supported in part by the MISP (Ministry of Science, ICT & Future Planning), Korea, under the National Program for Excellence in SW (2016-0-00018) supervised by the IITP (Institute for Information & communications Technology Promotion) (2016-0-00018).

## References

[1] Amazon Web Service - Virtual Private Cloud (VPC). https://aws.amazon.com/vpc/.

[2] Cisco - IPSec Stateful Failover (VPN High Availability) Feature Module. http://www.cisco.com/c/en/us/td/docs/ios/12_2/12_2y/12_2yx11/feature/guide/ft_vpnha.html.

[3] Google - VPN throughput. https://cloud.google.com/compute/docs/vpn/advanced.

[4] Google Cloud Platform - Virtual Network. https://cloud.google.com/virtual-network/.

[5] Microsoft Azure - About VPN Gateway. https://docs.microsoft.com/en-us/azure/vpn-gateway/vpn-gateway-about-vpngateways.

[6] Microsoft Azure - Planning and design for VPN Gateway. https://azure.microsoft.com/en-us/documentation/articles/vpn-gateway-plan-design/.

[7] Microsoft Azure - SLA for VPN Gateway. https://azure.microsoft.com/en-us/support/legal/sla/vpn-gateway/.

[8] Microsoft Azure - Virtual Network. https://azure.microsoft.com/en-us/services/virtual-network.

[9] Microsoft Azure - VPN Gateway. https://azure.microsoft.com/en-us/services/vpn-gateway/.

[10] Microsoft Remote Procedure Call. https://msdn.microsoft.com/en-us/library/windows/desktop/aa378651(v=vs.85).aspx.

[11] PRADS - Passive Real-time Asset Detection System.

[12] strongSwan - High Availability. https://wiki.strongswan.org/projects/strongswan/wiki/HighAvailability.

[13] VPN Service Level Agreement (SLA). https://cloud.google.com/vpn/sla.

[14] Windows Remote Access Service. https://technet.microsoft.com/en-us/library/dn636119(v=ws.11).aspx.

[15] S. Akoush, R. Sohan, A. Rice, A. W. Moore, and A. Hopper. Predicting the Performance of Virtual Machine Migration. In *MASCOTS*, 2010. IEEE.

[16] S. A. Baset, L. Wang, and C. Tang. Towards an Understanding of Oversubscription in Cloud. In *HotICE*, 2012. USENIX.

[17] N. Bobroff, A. Kochut, and K. Beaty. Dynamic Placement of Virtual Machines for Managing SLA Violations. In *INM*, 2007. IEEE.

[18] A. Bremler-Barr, Y. Harchol, and D. Hay. OpenBox: A Software-Defined Framework for Developing, Deploying, and Managing Network Functions. In *SIGCOMM*, 2016. ACM.

[19] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live Migration of Virtual Machines. In *NSDI*, 2005. USENIX.

[20] M. Dobrescu, N. Egi, K. Argyraki, B.-G. Chun, K. Fall, G. Iannaccone, A. Knies, M. Manesh, and S. Ratnasamy. RouteBricks: Exploiting Parallelism to Scale Software Routers. In *SOSP*, 2009. ACM.

[21] D. E. Eisenbud, C. Yi, C. Contavalli, C. Smith, R. Kononov, E. Mann-Hielscher, A. Cilingiroglu, B. Cheyney, W. Shang, and J. D. Hosein. Maglev: A Fast and Reliable Software Network Load Balancer. In *NSDI*, 2016. USENIX.

[22] S. K. Fayaz, Y. Tobioka, V. Sekar, and M. Bailey. Bohatei: Flexible and Elastic DDoS Defense. In *USENIX Security*, 2015. USENIX.

[23] R. Gandhi, Y. C. Hu, and M. Zhang. Yoda: A Highly Available Layer-7 Load Balancer. In *EuroSys*, 2016. ACM.

[24] R. Gandhi, H. H. Liu, Y. C. Hu, G. Lu, J. Padhye, L. Yuan, and M. Zhang. Duet: Cloud Scale Load Balancing with Hardware and Software. In *SIGCOMM*, 2014. ACM.

[25] M. R. Garey, R. L. Graham, and J. D. Ullman. Worst-case Analysis of Memory Allocation Algorithms. In *STOC*, 1972. ACM.

[26] A. Gember, A. Krishnamurthy, S. S. John, R. Grandl, X. Gao, A. Anand, T. Benson, A. Akella, and V. Sekar. Stratos: A Network-Aware Orchestration Layer for Middleboxes in the Cloud. In *CoRR*, 2013.

[27] A. Gember-Jacobson, R. Viswanathan, C. Prakash, R. Grandl, J. Khalid, S. Das, and A. Akella. OpenNF: Enabling Innovation in Network Function Control. In *SIGCOMM*, 2014. ACM.

[28] Y. Go, M. A. Jamshed, Y. Moon, C. Hwang, and K. Park. APUNet: Revitalizing GPU as Packet Processing Accelerator. In *NSDI*, 2017. USENIX.

[29] A. Greenberg. Windows Azure: Scaling SDN in the Public Cloud. Open Networking Summit (ONS) 2014.

[30] S. Han, K. Jang, K. Park, and S. Moon. PacketShader: A GPU-accelerated Software Router. In *SIGCOMM*, 2010. ACM.

[31] M. Kablan, A. Alsudais, E. Keller, and F. Le. Stateless network functions: Breaking the tight coupling of state and processing. In *NSDI*, 2017. USENIX.

[32] C. Kaufman, P. Hoffman, Y. Nir, and P. Eronen. Internet Key Exchange Protocol Version 2 (IKEv2). RFC 5996, RFC Editor, 2010.

[33] S. Kent. IP Encapsulating Security Payload (ESP). RFC 4303, RFC Editor, 2005. `http://www.rfc-editor.org/rfc/rfc4303.txt`.

[34] J. Kim, K. Jang, K. Lee, S. Ma, J. Shim, and S. Moon. NBA (Network Balancing Act): A High-performance Packet Processing Framework for Heterogeneous Processors. In *EuroSys*, 2015. ACM.

[35] B. Li, K. Tan, L. L. Luo, Y. Peng, R. Luo, N. Xu, Y. Xiong, P. Cheng, and E. Chen. ClickNP: Highly Flexible and High Performance Network Processing with Reconfigurable Hardware. In *SIGCOMM*, 2016. ACM.

[36] H. Liu, C.-Z. Xu, H. Jin, J. Gong, and X. Liao. Performance and Energy Modeling for Live Migration of Virtual Machines. In *HPDC*, 2011. ACM.

[37] L. Liu, H. Xu, Z. Niu, P. Wang, and D. Han. U-HAUL: Efficient State Migration in NFV. In *AP-Sys*, 2016. ACM.

[38] M. Mao and M. Humphrey. A Performance Study on the VM Startup Time in the Cloud. In *CLOUD*, 2012. IEEE.

[39] J. C. Mogul and L. Popa. What We Talk About when We Talk About Cloud Network Performance. In *SIGCOMM CCR*, 2012. ACM.

[40] S. Palkar, C. Lan, S. Han, K. Jang, A. Panda, S. Ratnasamy, L. Rizzo, and S. Shenker. E2: A Framework for NFV Applications. In *SOSP*, 2015. ACM.

[41] A. Panda, S. Han, K. Jang, M. Walls, S. Ratnasamy, and S. Shenker. NetBricks: Taking the V out of NFV. In *OSDI*, 2016. USENIX.

[42] P. Patel, D. Bansal, L. Yuan, A. Murthy, A. Greenberg, D. A. Maltz, R. Kern, H. Kumar, M. Zikos, H. Wu, C. Kim, and N. Karri. Ananta: Cloud Scale Load Balancing. In *SIGCOMM*, 2013. ACM.

[43] S. Rajagopalan, D. Williams, and H. Jamjoom. Pico Replication: A High Availability Framework for Middleboxes. In *SoCC*, 2013. ACM.

[44] J. Sherry, P. X. Gao, S. Basu, A. Panda, A. Krishnamurthy, C. Maciocco, M. Manesh, J. a. Martins, S. Ratnasamy, L. Rizzo, and S. Shenker. Rollback-Recovery for Middleboxes. In *SIGCOMM*, 2015. ACM.

[45] R. P. Singh, T. Brecht, and S. Keshav. Towards VM Consolidation Using a Hierarchy of Idle States. In *VEE*, 2015. ACM.

[46] B. Urgaonkar, P. Shenoy, and T. Roscoe. Resource Overbooking and Application Profiling in Shared Hosting Platforms. In *OSDI*, 2002. USENIX.

[47] A. Verma, J. Bagrodia, and V. Jaiswal. Virtual Machine Consolidation in the Wild. In *Middleware*, 2014. ACM.

[48] W. Voorsluys, J. Broberg, S. Venugopal, and R. Buyya. Cost of Virtual Machine Live Migration in Clouds: A Performance Evaluation. In *CloudCom*, 2009. Springer-Verlag.

[49] T. Wood, P. J. Shenoy, A. Venkataramani, and M. S. Yousif. Black-box and Gray-box Strategies for Virtual Machine Migration. In *NSDI*, 2007. USENIX.

[50] K. Ye, X. Jiang, D. Huang, J. Chen, and B. Wang. Live Migration of Multiple Virtual Machines with Resource Reservation in Cloud Computing Environments. In *CLOUD*, 2011. IEEE.

[51] X. Zhang, Z.-Y. Shae, S. Zheng, and H. Jamjoom. Virtual Machine Migration in an Over-committed Cloud. In *NOMS*, 2012. IEEE.