
| | |
|---|---|
| Project acronym: | **PrEstoCloud** |
| Project full name: | **Proactive Cloud Resources Management at the Edge for efficient Real-Time Big Data Processing** |
| Grant agreement number: | **732339** |

# D3.3 – Inter-site Network Virtualization Orchestrator

| | |
|---|---|
| Deliverable Editor: | **Quentin Jacquemart and Guillaume Urvoy-Keller (CNRS)** |
| Other contributors: | **Quentin Jacquemart and Guillaume Urvoy-Keller (CNRS)** |
| Deliverable Reviewers: | **Salman Taherizadeh (JSI) and Noam Amram (LiveU)** |
| Deliverable due date: | **30/03/2018** |
| Submission date: | **03/05/2018** |
| Distribution level: | **Public** |
| Version: | **1.0** |

This document is part of a research project funded by the Horizon 2020 Framework Programme of the European Union

## Change Log

| Version | Date | Amended by | Changes |
|---|---|---|---|
| 0.1 | Mar. 26, 2018 | Quentin Jacquemart (CNRS), Guillaume Urvoy-Keller (CNRS) | Extended ToC |
| 0.2 | Mar. 29, 2018 | Quentin Jacquemart (CNRS), Guillaume Urvoy-Keller (CNRS) | Content for Chapter 2 |
| 0.4 | Apr. 03, 2018 | Guillaume Urvoy-Keller (CNRS) | Content for Chapter 4 |
| 0.5 | Apr. 06, 2018 | Quentin Jacquemart (CNRS) | Content for Chapter 2, Chapter 3 |
| 0.5a | Apr. 13, 2018 | Quentin Jacquemart (CNRS) | Content for Chapter 3 |
| 0.5b | Apr. 17, 2018 | Guillaume Urvoy-Keller (CNRS) | Content for Chapter 5 |
| 0.6 | Apr. 19, 2018 | Guillaume Urvoy-Keller (CNRS) | Content for Chapter 1 |
| 0.7 | Apr. 20, 2018 | Quentin Jacquemart (CNRS) | Revised Chapter 2, Chapter 3 |
| 0.8 | Apr. 23, 2018 | Quentin Jacquemart (CNRS) | Revised Chapter 4 |
| 0.9 | Apr. 24, 2018 | Quentin Jacquemart (CNRS) | Revised Chapter 1, Chapter 5 |
| 0.10 | Apr. 25, 2018 | Guillaume Urvoy-Keller (CNRS) | Revised Chapter 2, Chapter 3 |
| 0.11 | Apr. 25, 2018 | Quentin Jacquemart (CNRS) | Executive Summary |
| 0.12 | Apr. 26, 2018 | Salman Taherizadeh (JSI) | Internal review |
| 0.13 | May 01, 2018 | Noam Amram (LiveU) | Internal review |
| 0.14 | May 03, 2018 | Quentin Jacquemart (CNRS), Guillaume Urvoy-Keller (CNRS) | Revisions following internal reviews |
| 1.0 | May 03, 2018 | Quentin Jacquemart (CNRS), Guillaume Urvoy-Keller (CNRS) | Ready for submission |

# Contents

# List of Figures

# List of Tables

# List of Abbreviations

| Abbreviation/Acronym | Description |
|---|---|
| AES | Advanced Encryption Standard |
| ABW | Available Bandwidth |
| ADR | Asymptotic Dispersion Rate |
| API | Application Programming Interface |
| AS | Autonomous System |
| AWS | Amazon Web Services |
| AZ | Availability Zone |
| BW | Bandwidth |
| CA | Certificate Authority |
| CLI | Command Line Interface |
| CNRS | Centre National de la Recherche Scientifique |
| CTR | Counter Mode |
| CPU | Central Processing Unit |
| DMZ | Demilitarized Zone |
| Eth (or ETH) | Ethernet |
| Geneve | Generic Network Virtualization Encapsulation |
| GW | Gateway |
| GRE | Generic Routing Encapsulation |
| IaaS | Infrastructure as a Service |
| IETF | Internet Engineering Task Force |
| IKE | Internet Key Exchange |
| I3S | Laboratoire d'Informatique, Signaux et Systèmes de Sophia Antipolis |
| IP | Internet Protocol |
| IPsec | Internet Protocol Security |
| ISP | Internet Service Provider |
| L2TP | Layer 2 Tunneling Protocol |
| LAN | Local Area Network |
| NAT | Network Address Translation |
| NIC | Network Interface |
| NVGRE | Network Virtualization using GRE |
| OvS | Open vSwitch |
| OWD | One-Way Delay |
| PGM | Probe Gap Model |
| PKI | Public Key Infrastructure |
| PPTP | Point-to-Point Tunneling Protocol |
| PRM | Probe Rate Model |
| RAM | Random Access Memory |
| RC4 | Ron's Cipher 4 |

| | |
|---|---|
| RFC | Request For Comments |
| RSA | Rivest–Shamir–Adleman |
| RTT | Round-Trip Time |
| SDN | Software-Defined Network/Networking |
| SSH | Secure Shell |
| SSL | Secure Socket Layer |
| SST | Stateless Transport Tunneling |
| TCP | Transmission Control Protocol |
| TLS | Transport Layer Security |
| UDP | User Datagram Protocol |
| VLAN | Virtual LAN |
| VM | Virtual Machine |
| VNet | Virtual Network |
| VNIC | Virtual Network Interface |
| VPC | Virtual Private Cloud |
| VPN | Virtual Private Network |
| VXLAN | Virtual eXtensible LAN |

# Executive Summary

D3.3: Inter-site Network Virtualization Orchestrator, iteration 1
T3.2: Inter-site Network Virtualization Orchestrator
WP3: Mobile-based Real-Time Processing Network Based on a Cloud-Edge Communication Layer

In this Deliverable, we present the *architecture* behind the PrEstoCloud **overlay network**. This overlay network enables the *software components* of distributed applications running on the PrEstoCloud platform to **communicate** in a *seamless manner*, regardless of the actual physical location of the software component (i.e. private cloud, public cloud, or edge). Due to the heterogeneity of deployment scenario envisioned by the PrEstoCloud platform, it is necessary for the network overlay to meet the constraint of **genericity**. In other words, the proposed overlay solution should not be bound to a proprietary vendor or technology, that would hinder its applicability when considering application deployments.

To this end, we start by a broad review of all applicable technologies for multi/hybrid cloud overlays. We first investigate the offerings specific to each IaaS platform and consider their applicability, first, on their own, and then, as part of the PrEstoCloud framework. We also look at the various opened Internet standards that are applicable for the PrEstoCloud overlay. Based on this extensive review, conducted in Chapter 2, we select the fittest technologies, TLS and IPsec, as the base for the PrEstoCloud overlay.

In Chapter 3, we discuss implementation issues for the PrEstoCloud overlay, and detail the inner workings of the current PrEstoCloud overlay prototype. Following the specifications, we have developed a prototype which is able to establish an overlay between hybrid clouds composed of a local private instance of OpenStack, Amazon Web Services, and Microsoft Azure. Furthermore, we illustrate that our architecture is indeed generic enough to be implemented through the functions available on all IaaS platforms. We emphasize and put into perspective the implementation differences between the two selected technologies. We also discuss the challenges resulting from bootstrapping the overlay on the different IaaS platforms, and provide the corresponding architectural solutions.

In Chapter 4, we deploy our prototype in the wild in order to conduct network measurements related to the stability, resiliency, and performance of the PrEstoCloud overlay. We show that the overlay network is highly stable, regardless of the choice of the underlying technology (TLS or IPsec). We further show that the latency overhead due to the overlay network is negligible, regardless of the implementation. Finally, we focus on estimating the available bandwidth inside the overlay network.

Finally, in Chapter 5, we conclude our discussion by summarizing the status at the current stage, and list the next milestones towards the completion of the second iteration of this deliverable and task.

# Chapter 1

# Introduction

PrEstoCloud aims at enabling the deployment of distributed streaming applications over compute nodes that might be located in private clouds, public clouds, and up to the edge (IoT/mobile devices or small data centers). Each application can be seen, from a networking perspective, as a set of software components deployed in physical or virtual machines, possibly embedded in containers.

The different *components* of the application should be able to **communicate** in a *seamless manner*, as if they all were deployed in the same local area network. This means that the hosting physical or virtual devices should be configured with an IP address in a specific IP subnet.

The **inter-site network virtualization orchestrator** should enable a *fast* deployment of a **network overlay** emulating this flat IP space from the application perspective. Overlay network techniques [17] allow the creation of a virtual network topology on top of a physical one. This enables to specify which node of the network is connected to another – even if, in the underlying physical network, they are not directly connected to one another – thereby creating a logical topology. To put it differently, if the path between two nodes is composed of several links, in an overlay network, the two nodes can be considered as being directly connected through one single virtual link.

In the context of PrEstoCloud, the overlay will be materialized on each device by a specific network interface featuring an IP address in a dedicated (and large enough) private IP address subnet. The network orchestrator will further take care of adding specific routing rules and possibly dedicated machines to build the overlay.

The network orchestrator should *provide* the following **services** to the PrEstoCloud platform:

- *interconnect* compute resources with one another, irrespectively of their actual physical location (private cloud, public cloud, edge);
- *deploy* an **overlay**, i.e. a full virtual infrastructure, to build the virtual IP network needed by PrEstoCloud components;
- *secure* this global infrastructure, to, in turn, enable a secure communication between the different application components over an untrusted channel such as the Internet;
- *report* on low-level information, including the status of the global overlay components, and other Quality of Service metrics of interest for the application, such as round-trip time (RTT), or the available bandwidth.

We further impose two additional *constraints*:

- *genericity*: the solution proposed should easily support any kind, any size and any combination of clouds that are going to be connected to each other (i.e. not be bound to a proprietary solution);

- *performance*: data streaming applications might be bandwidth hungry. Thus, our solution should permit to achieve an optimal throughput and a low delay.

This Deliverable is organized as follows. In Chapter 2, we survey the technologies available to build the overlay we need in PrEstoCloud. These technologies consist of tools offered under public license or solutions offered by cloud providers (e.g. Amazon Web Services). While we want to have generic solutions, as stated in our list of constraints, we do not rule out a priori solutions provided by cloud providers as they might be inter-operable with other solutions and might be fast and easy to configure. While reviewing publicly available tools and cloud provider specific tools, we will discover that those tools often feature only a part of the solution we need for our Network Orchestrator, but not a complete one.

The study of the state of the art performed in Chapter 2 leads to presentation of the architecture of the PrEstoCloud overlay network solution, fulfilling the constraints stated previously, in Chapter 3. The design phase has been complemented by the creation of a prototype. This prototype has been evaluated carefully in Chapter 4 for a scenario encompassing a private OpenStack infrastructure, Amazon Web Services data centers, and Microsoft Azure data centers.

We performed measurements in the wild comparing the performance of the different options featured by our prototype with each other and also with a native solution where the machines, using their public addresses or the NAT (Network Address Translation) connect directly to each other. While the latter (native) option does not respect our design requirements, it can be seen as a reference point when assessing the performance of any overlay solution that will, by definition, add additional protocol layer to interconnect the machines.

We further present results related to the measurement of available bandwidth[1]. The key idea here is to be able to measure the available bandwidth with a minimal network footprint as the latter is directly converted into financial cost for the end user. While preliminary, those results shed light on the available bandwidth that can be observed in between the different data centers of a provider and raise interesting measurement problems related to the complexity of the network set-up used by cloud providers.

We end this Deliverable by summarizing the current stage completed so far in the design and implementation of the PrEstoCloud overlay network. We also list the next milestones towards the completion of the second iteration of this Deliverable due at the end of M30.

[1]The *available bandwidth* will be formally defined in Chapter 4. For now, it can be seen as the capacity left over when cross traffic occupation as been discounted for a given link or path.

# Chapter 2

# Existing Technologies

In Chapter 1, we introduced the concept of *network overlay*, and the need for such technologies as a support for the PrEstoCloud platform, particularly in the context of *hybrid* and *federated* clouds. In this Chapter, we review the existing technologies and ready-to-use tools related to the two following questions:

1. how to connect a *private* network and/or infrastructure with the cloud?
2. how to connect *multiple public* clouds together?

The first question is a main topic when considering hybrid clouds, giving the ability to connect a private cloud with a public cloud, e.g. for elasticity purposes. The second question can be further sub-divided into two separate topics. The first subquestion relates to the ability to connect two (physical) parts of a public cloud owned by the same provider together, i.e. the ability and means to connect multiple data centers. This need might arise due to the geographic dispersion of the underlying physical public cloud infrastructure. The second is the ability to connect the virtual infrastructures of public clouds *from multiple independent providers*, e.g. connect AWS's virtual infrastructure with Azure's. This ability is essential in order to seamlessly be able to switch between multiple cloud providers, depending on selection criteria, such as instance price, latency, privacy concerns, etc.

The first part of this Chapter provides a detailed review of the networking features available within public clouds. We will see that, while numerous, these features (i) are non-standard and proprietary, (ii) may rely on third-party privileged business partners with first-hand access to infrastructure/technology (iii) are incompatible among themselves, and, therefore, lead to a situation of *vendor lock-in*, which is undesirable as a customer.

The second part of this Chapter focuses on the standardized technologies available for creating an overlay for the PrEstoCloud platform, while meeting the requirements of openness and genericity that are necessary for the platform to run, interchangeably, over the multiple public and private clouds.

## 2.1 Networking Features in Cloud Platforms

In this Section, we analyze the networking features of the main public and private cloud platforms, namely: Amazon Web Services, Microsoft Azure, VMWare vCloud Air, OpenStack, and CloudStack. We also predominantly emphasize our focus on the two main players: AWS and Azure [37].

As a reminder, these platforms offer various solutions and services to manage an *Infrastructure as a Service* (IaaS). The IaaS model empowers the user with the ability to control the infrastructure starting from the application layer down to the control of the nodes instances themselves (Figure 2.1). Even if the lower layers,

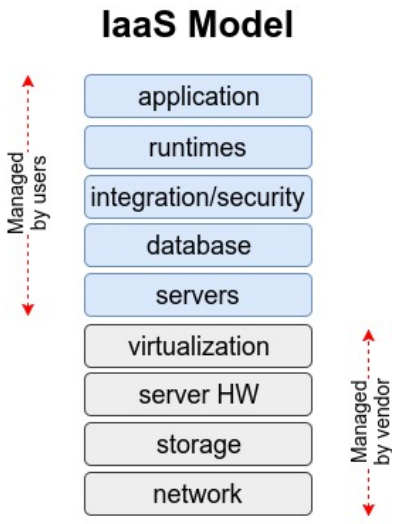


**Figure 2.1:** IaaS management model

e.g. the network configuration, are not fully under the control of the user, the cloud provider usually makes available a set of services and tools to bring limited customizations at these levels. Overall the IaaS platforms propose solutions for network management, giving users access to this set of services:

1. creation and management of virtual networks, defining IP addresses ranges and subnetworks,
2. NAT management and routing tables definition,
3. connection to the Internet (public addresses) or private connections through VPNs.

We now describe all the different services and products offered by the various providers, so as to understand their capabilities and limitations.

### 2.1.1 Amazon Web Services

The underlying physical infrastructure behind Amazon Web Services (AWS) is geographically distributed over several isolated zones, called **regions** [8]. By default, resources instantiated in different regions cannot communicate with each other. However, there are different ways to enable cross region communication: either over the public Internet, or using the AWS built-in VPN service. Each region is, in turn, divided into several **availability zones** (AZs), designed to ensure fault tolerance between machines instantiated into different AZs.

If more freedom on placing virtual machines is needed, e.g. to obtain specific delay or throughput constraints, Amazon provides two different possibilities:

- the possibility to require a *Dedicated Host* [5], a physical machine in which we can instantiate our virtual machines;
- alternatively, the VMs can be grouped into Placement Groups [7], a logical aggregation of VMs in the same availability zone.

From a networking point of view, Amazon divides its datacenter network into **Virtual Private Clouds** (*VPCs*) [9], which are dedicated virtual networks inside an AWS region. A VPC is *isolated* from other virtual networks, and can be split into different subnets, within which AWS resources (e.g. EC2 instances) can be instantiated. These instances will be assigned an IP address within the VPC subnet. The different subnets can be configured

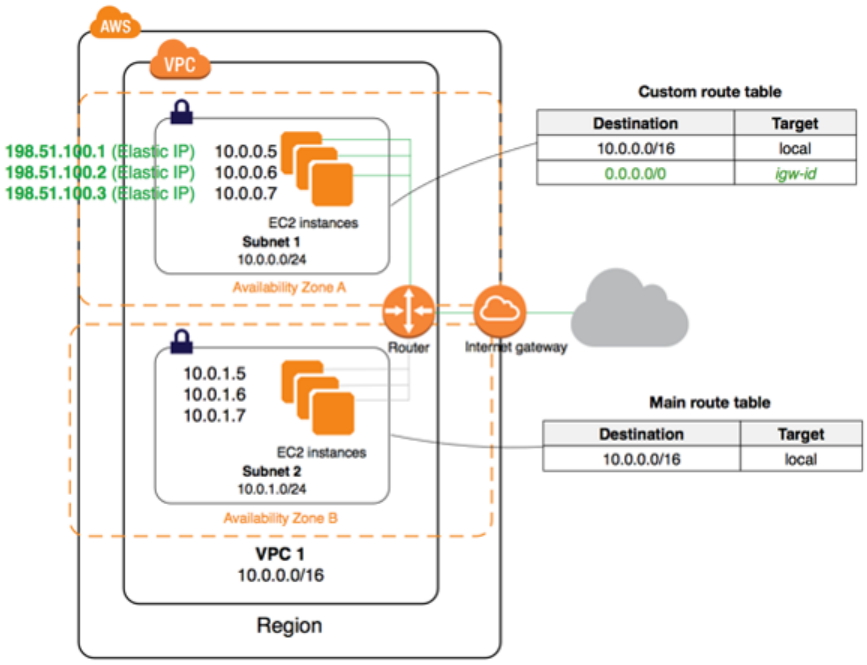


**Figure 2.2:** Use of an *Internet gateway* to connect a VPC to the Internet

to be connected to the Internet, through an *Internet Gateway* (Figure 2.2), or to remain private networks. Of course, two subnets inside the same VPC can have different connections to the gateways. Moreover, other than connecting a VPC to the Internet, a VPN gateway can be used to connect it to the user personal on-premises infrastructure (e.g. user's corporate network) (Figure 2.3).

All virtual machines instantiated within a VPC can communicate directly with each other. If this communication needs to be expended across VPCs, the VPN gateway can be used to connect two different VPCs inside the same AWS region. In this regard, Amazon proposes several solutions [6]. All AWS instances that need to be connected to the Internet have their own public IP address, called **Elastic IP**. It can be also used to improve fault tolerance. Indeed, the public IP address can be assigned dynamically to one instance, thus if the assigned instance fails we can easily re-assign this public address to another one. In this case, if a user connects to this IP address, even using the Amazon DNS server, it will reach the backup service.

Another solution to interconnect a private infrastructure/cloud to AWS is *Direct Connect* [47]. Direct Connect provides dedicated and isolated connections to the AWS services. By means of *Direct Connect Partners* (i.e. privileged Internet Service Providers), a direct connection between the VPC and the user's private network can be established, using private and prioritized channels. These lines are also dedicated, i.e. bandwidth will not be shared with the rest of the Internet traffic. A software (LOA-CFA) must be downloaded by the user in order to complete the interconnection [52]. Amazon also provides several locations to which a direct connection can be established in case the private infrastructure equipment is hosted in the same facility as the AWS Direct Connect.

### 2.1.2 Microsoft Azure

Similarly to Amazon Web Services, Microsoft Azure is divided into **regions** [53], to provide service *availability* and *redundancy*. Each region is paired with another region in the same geographic zone to replicate the resources, except for South America, which only hosts one single Azure region.

The services offered by Azure are similar to the ones proposed by AWS, its direct competitor. The **place-**

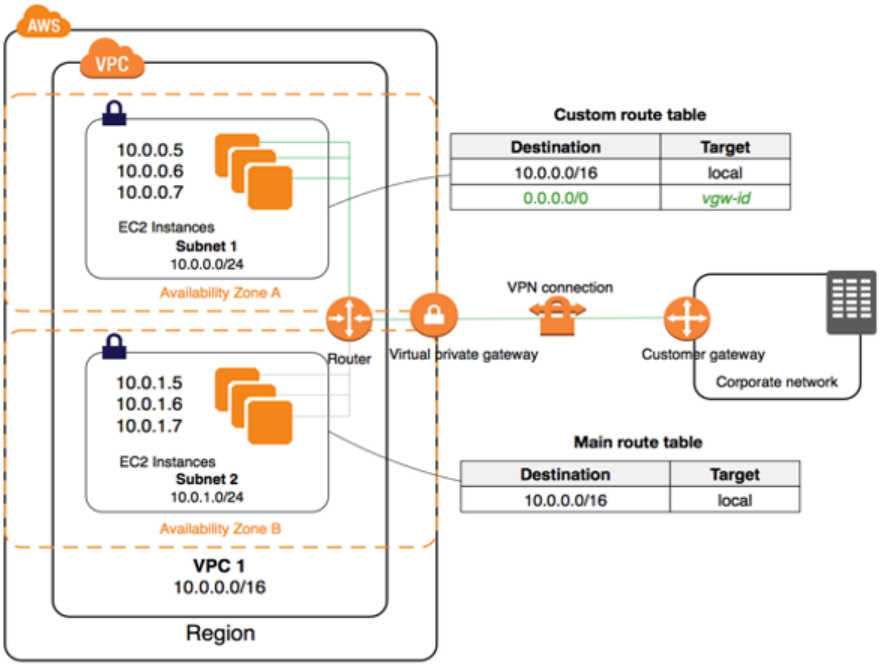


**Figure 2.3:** Use of a *VPN gateway* to connect a VPC to a personal/private infrastructure

**ment groups** [50] are used to map instances to *affinity groups*. They allow the placement of virtual machines in servers located close to each other, in the same region. Similarly, an **availability set** [46] logically ensures redundancy and fault tolerance. Furthermore, **scale sets** [54] can be used to manage a group of VMs with the same configuration, in order to help automatically scale computation resources.

Azure allows the creation of **virtual networks** (*VNets*) [57], comparable to AWS VPCs, in which virtual machines can be instantiated. To assign a VM to a VNet a *Network Interface* (NIC) shall be created. A single VM can be composed of multiple NICs. A NIC supports both public and private IP addresses. With a public IP address, the VM is able to communicate with other VMs that are not connected to the same VNet directly, through the Internet. Private IP addresses are used for communications inside the a single VNet. A VNet can be divided into subnets [49] to organize the network.

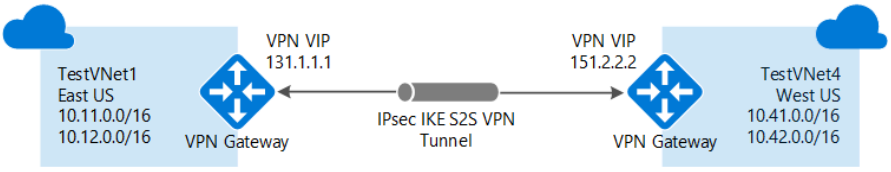


**Figure 2.4:** Point-to-Point connection model [58]

To make two or more VNets communicate, even if in different regions, a *VNet-to-VNet* connection can be configured [58]. Through a VPN connection, relying on an IPsec tunnel, VNets are able to reach each other (Figure 2.4).

The same method can be used to connect a VNet to a user's private infrastructure, known as a *Site-to-Site* connection [45] (Figure 2.5). Like AWS, Azure provides an ISP-based private connection called *Express Route* [48], which allows the user to directly connect to the VNet without going through the Internet, thereby reducing delays and increasing throughput. Express Route partners are however needed to enable this connection.

Finally, it is possible to create a Point-to-Site (P2S) connection [51], in case a single machine needs to be connected to a VNet. The underlying technology is a point-to-site SSTP tunnel. A VNet can be connected to

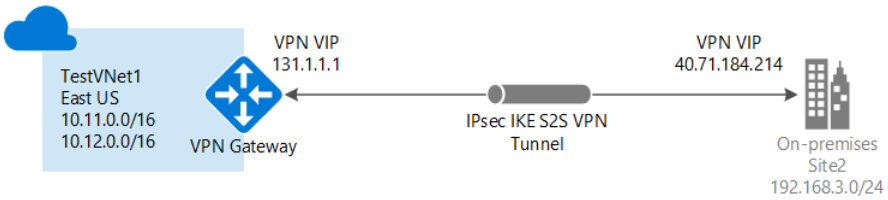


**Figure 2.5:** Site-to-Site connection model [45]

multiple "points" through different tunnels.

### 2.1.3 VMware vCloud Air

Unlike AWS and Azure, VMware vCloud Air does *not* have a region subdivision. It supports the creation of virtual private networks in its vCloud Air platform through its **virtual private cloud** [83].

The vCloud Air **hybrid DMZ** (Demilitarized Zone) [82] enables isolated virtual networks in the cloud, allowing users to connect their private networks to the cloud. Hybrid DMZs also allow extending security policies and network functions to the cloud. For example, if a load balancer or a firewall are hosted in a private datacenter, vCloud Air can rely on these locally-hosted services instead of the default ones.

vCloud Air **Advanced Networking Services** [84] provides *network segmentation*, *dynamic routing*, and *network interconnection* in two different ways:

1. point-to-site SSL connection, similar to the one proposed by Azure;
2. site-to-site connection through IPsec tunnel.

Meanwhile, vCloudAir **Direct Connect** [81] occupies the same scope as AWS Direct Connect and Azure Express Route.

### 2.1.4 OpenStack and CloudStack

Both of these platforms are open source projects that serve as a framework to build a public or private cloud service for third-party companies or private users. They provide modules that can provide the same networking services as the public cloud offerings.

OpenStack provides the **Neutron** module for network management [64], enabling virtual networks definitions, floating IPs (same as AWS Elastic IP), and various networking plug-ins to support several technologies (e.g. OvS, CiscoUCS, Linux Bridge, …).

Similarly, CloudStack aims to build a Network as a Service [19], using the modules **Nicira NVP** and **Nuage VSP**, to manage virtual networks, and create shared networks. Elastic IPs provides the same function as the AWS namesake service.

### 2.1.5 Discussion

Table 2.1 summarizes the available networking features for each IaaS cloud platform, as described in the previous sections. It shows that the range of achievable features for each vendor is similar, with only negligible differences in functionality, and different nomenclature. This can be explained by the fact that these are commercial platforms, that, due to market competition, need to offer the same set of features to their customers, for fear of losing them to the competition.

| | Amazon Web Services | Microsoft Azure | VMWare vCloud Air | OpenStack |
|---|---|---|---|---|
| **Choose IP range** | Virtual Private Cloud (VPC) | Virtual Network | Advanced Networking Services | Virtual/Self-service Networks |
| **Reach cloud from Internet** | Elastic IP | (Dynamic) public IP assigned to resource | Direct Connect | Floating IP |
| **Interconnect zones and/or data centers** | VPC peering | • VNet peering (same location)<br>• VNet-to-VNet tunneling (across sites) | DMZ | BGP+VPN routing |
| **Bridge with personal infrastructure** | Direct Connect | • VNet-to-VNet tunneling<br>• Express Route<br>• SSTP tunnel | Direct Connect VPN | |

**Table 2.1:** Networking features in ubiquitous IaaS platforms

However, let's consider the last row in Table 2.1, detailing the ability to bridge a personal infrastructure with a public IaaS platform. As we can see, all three major platforms offer at least one way to achieve it. However, AWS's and VMWare's Direct Connect services, as well as Azure's Express Route, are all based on having a third-party ISP, who is a partner with the targeted cloud provider, dedicating infrastructure to the connection. In other words, this is a static service setup which cannot be activated and deactivated on demand. Moreover, the involvement of a privileged partner ISP implies further restrictions on the availability of the service, usually requiring that the private infrastructure is connected to the Internet via that same ISP. Furthermore Azure's VNet-to-VNet has restricted availability as it is embedded in the Azure platform. The possibility, here, is to connect a private infrastructure running with the Azure stack to the public Azure infrastructure. In this situation, the restriction is the choice of the cloud management platform for the private infrastructure. Finally, Azure's SSTP option is of limited availability, due to the underlying SSTP protocol, which is a Microsoft-specific (but documented) protocol, mostly implemented within Windows products. While there are some open-source implementation for further compatibility, they are not from Microsoft themselves, potentially limiting the compatibility, especially regarding new features. One option proposed by [75] is to create a specific Windows Server VM in AWS, and install a VPN software supported by Azure to enable the connection. This obviously imposes a strong constraint on the cloud architecture. Following these observations, it prevails that, while available, these solutions are incompatible with one another. Relying on either means vendor locking the platform in a way that is inoperable with other available solutions.

One of the objectives of the PrEstoCloud project is to make the best possible use out of (private and public) cloud resources, depending on the evolving needs of an application deployed over a hybrid infrastructure. Therefore, relying on these solutions means sticking to a specific cloud architecture, which may *not* be the best at a particular moment, due to specific constraints (e.g. cost, latency, privacy issues, …), as further elaborated in WP7 (Use-case Specification and Validation) deliverables D7.1 – As-Is and To-Be Scenarios, and D7.2 – Planning and Validation Plan.

## 2.2 Overlay Protocols

Following the observations described in Section 2.1.5, the PrEstoCloud consortium needs to design its own solution for creating the PrEstoCloud overlay network. We here outline a few requirements of this design:

1. genericity: the final design needs to be applicable to a variety of private and public clouds, and over all types of (guest) software environments;
2. transparency: the final design needs to be transparent to the deployed application, i.e. the design should not require any modification to the application itself;
3. minimal overhead: the final design should not impede too heavily on the capacity (e.g. bandwidth) of network transfers.

In this Section, we review the existing applicable standardized protocols that *could* be used as the basis for our network architecture design.

### 2.2.1 Virtual eXtensible LAN

*Virtual eXtensible LAN* (**VXLAN**) is one of the most used techniques for implementing virtual LANs [42]. Its core concept is deeply rooted in the legacy *VLAN* mechanism. VXLAN encapsulates Ethernet frames (Layer 2) in UDP datagrams (Layer 4). It associates a 24-bit unique VXLAN ID to every machine in the overlay. Consequently, VXLAN is much more scalable than the legacy VLAN (IEEE 802.1Q), which only supported a 12-bit identifier.

VXLAN relies on so-called *VXLAN tunnel endpoints* (*VTEPs*) located at each tunnel extremity for encapsulating and decapsulating the packets. A VTEP is assigned an IP address, and maintains a local table mapping the destination IP address of the connected nodes with the corresponding VXLAN ID. The encapsulation process consists in adding to the original Ethernet frame a VXLAN header which contains the VXLAN ID, the IPs of the source and destination, and the MAC address of the destination. The VXLAN frame is then sent to the VTEP of the destination IP address. This specific header format allows VXLAN to be a stateless. However, its reliance on UDP means it does not take into account eventual packet losses or out-of-order deliveries.

The VXLAN effort was originally started by VMWare and Cisco. It is now backed by several large cloud players such as Broadcom, Dell, Huawei, Juniper, Pica8, Open vSwitch, Red Hat, and more.

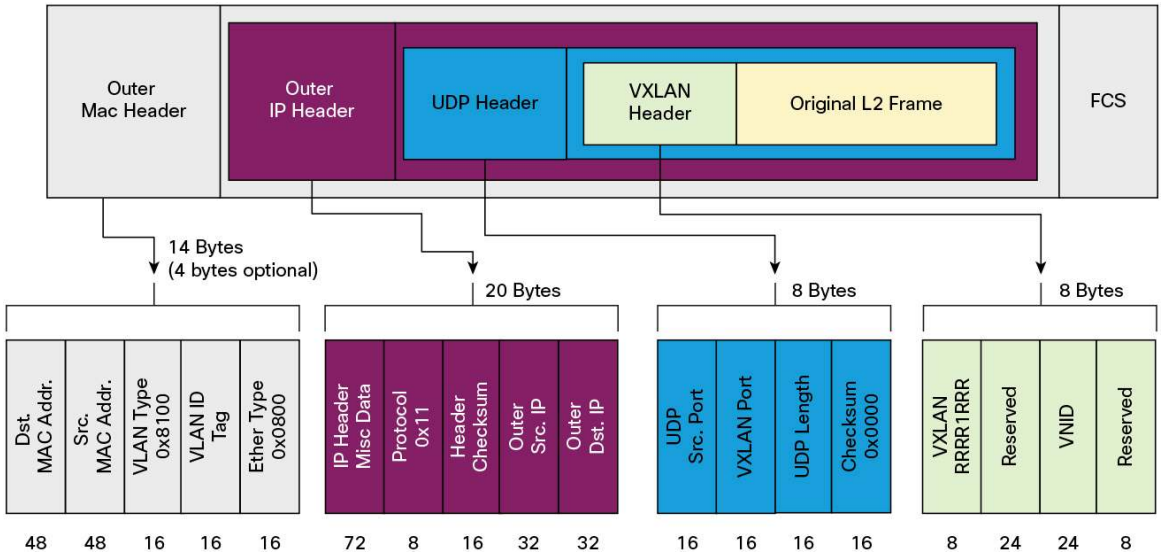


**Figure 2.6:** VXLAN frame

### 2.2.2 Network Virtualization using Generic Routing Encapsulation

*Network Virtualization* using *Generic Routing Encapsulation* (**NVGRE**) is an extension of GRE, discussed in [24]. GRE is an encapsulation protocol originally developed by Cisco in the mid-1990s to encapsualte a variety of protocols within an IP packet. Typical examples of use include IP-in-IP encapsulation, and non-IP datagrams across IP networks (e.g. ATM or MPLS frames across IP networks).

Microsoft recently proposed an extension of GRE to overcome inner scalability issues faced in large cloud computing environments, such as IaaS platforms. As such, it was designed to solve the same problems as VXLAN, and can thus be considered as a competitor to VXLAN. Actually, NVGRE's design is similar to VXLAN's, and ports another legacy concept (GRE instead of VLAN) to the cloud-networking age. It extends GRE to allow the encapsulation of an Ethernet (layer 2) frame in an IP (layer 3) packet, alongside a NVGRE header. NVGRE also relies on endpoints to perform the encapsulation and decapsulation of the frames. Unlike VXLAN, NVGRE comprises in its header a flow ID field, which can be exploited in multipath networks.

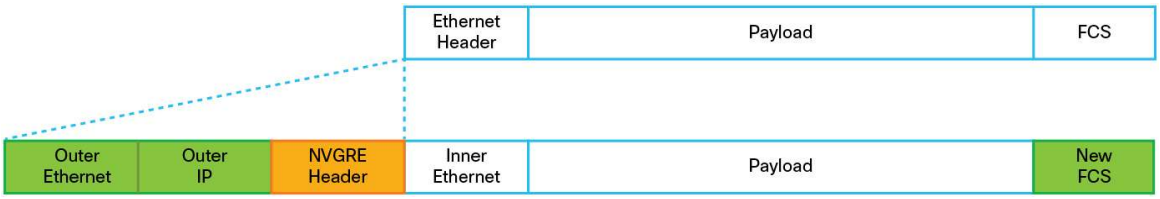


**Figure 2.7:** NVGRE frame

### 2.2.3 Stateless Transport Tunneling

The *Stateless Transport Tunneling* (**STT**) encapsulation scheme is also a Layer 2 over Layer 3 encapsulation mechanism. It was originally designed by VMWare to overlay two virtual switches over the physical infrastructure. It relies on a TCP-like header in order to take advantage of hardware acceleration found in modern NICs, particularly regarding the fragmentation and reassembly of jumbo packets. However, STT lacks diffusion among vendors, making it impractical to implement in some specific scenarios. Nevertheless, STT appears to perform better than its competitors in terms of CPU utilization and bandwidth throughput [33], [34]. For instance, it was observed that, using the Open vSwitch's implementation of STT and GRE, over a 10-Gbps Ethernet link, GRE was consuming about 50% more CPU resources, while, simultaneously, only achieving a throughput of 2.4 Gb/s, compared to 9.3 Gb/s for STT [34].

### 2.2.4 Generic Network Virtualization Encapsulation

The *Generic Network Virtualization Encapsulation* (**Geneve**) [20] is a new encapsulation protocol proposal, presented as a functional convergence for the encapsulation protocols presented above [18]. The proposal includes characteristics from the other encapsulation protocols, e.g. a 64-bit metadata field from STT. While promising, Geneve is still in gestation, and the current drafts exhibit some weak aspects, e.g. the variable options length field in the header, which, on the one hand gives flexibility to the protocol, and, on the other hand, causes ambiguity and implementation problems.

### 2.2.5 Overlays and Software-Defined Networking

A number of works have combined overlay network solutions with *Software Defined Networking* (**SDN**), in hope to gain more dynamism and flexibility. This kind of approaches give full control of the network and to perform flow orchestration.

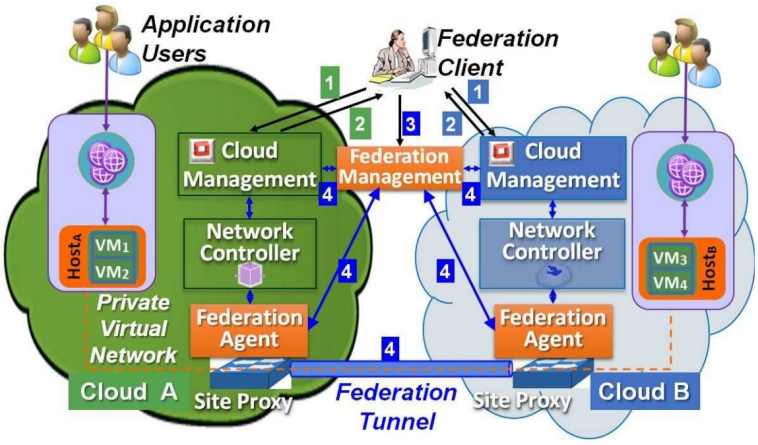


**Figure 2.8:** An overlay with SDN technologies (source: [38])

In this context, recent works like BEACON [59] and Seamless Cloud Interoperability [38] define three main components to enable the federation:

- The *federation management* (FM) is the central entity connected to every cloud. It provides a global view of the network and manages the identities and the cross-cloud connectivity. It further provides an interface and a virtualized view of the network to the user.
- The *federation agent* (FA) offers a control plane and management functions to the local cloud network, and communicates with other FAs in other clouds.
- The *federated data path* (FDP), also named *site proxy*, is the gateway of the cloud that connects the local network to the inter-cloud through a so-called *federation tunnel*. The FDP includes data plane functionalities, such as packet forwarding to the local network and to the federation.

Other proposals exploit a similar overlay network approach, but with an increased focus security, using virtual network functions (VNFs) [44] and VNF chains [43]. In both cases, security policies are enforced over the federation through the ad-hoc placement of VNFs.

RESERVOIR [74] exploits virtual networks to create isolated slices, aiming to achieve a system that allows different providers connect to each other dynamically. Buyya et al. [16], concentrate on the economic point of view, using a component named *cloud exchange*, which is in charge of evaluating all vendors' proposals and find the best offer, to help customer choose the cheapest cloud provider for given services.

### 2.2.6 Discussion

Some consider that the technologies reviewed in this Section are not to be considered competitors but different solutions for different situations [21]. For instance, VXLAN is widely supported by most vendors and can be optimally used in multi-vendor environments. This is the case when data traffic has to move to an external network service, and shall end up in different physical switches from different vendors. Meanwhile, in more local scenarios, like in a datacenter, i.e. with a single vendor, VXLAN causes overhead in terms of CPU and throughput. STT has been developed to deal with this situation and to provide better performances.

From this first analysis, potential protocol candidates for the PrEstoCloud overlay network are VXLAN and STT, given their popularity and performance optimization, respectively. Geneve is still too young to be implemented in a critical real-world scenario. NVGRE shows low performance compared to its competitors, which is an important pitfall, particularly in the context of PrEstoCloud, where we consider high-throughput applications.

However, neither VXLAN, nor STT provide security features. For this reason, we now focus (in Section 2.3) on the study of the protocols that enable secure communications.

## 2.3 Virtual Private Networks

*Virtual Private Networks* (**VPNs**) originally arose from the need to interconnect multiple geographic sites of multinational corporations (e.g. branch offices to the central headquarter) over the Internet. VPNs are thus a specific case of network overlays, that extend private networks across public networks. To ensure the private connectivity, VPNs usually provide encrypted tunnels to provide secured communications. VPNs technologies have evolved to be multipoint-to-multipoint overlays, often lacking neighbour discovery capabilities included in the protocols described in Section 2.2. Instead, VPNs rely on ad-hoc nodes rules and routes to reach the other machines and sites.

### 2.3.1 Point-to-Point Tunneling Protocol

*Point-to-Point Tunneling Protocol* (**PPTP**) is an extension of the legacy protocol PPP (Point-to-Point Protocol). PPP is a layer-2 protocol whose derivatives – PPP over Ethernet (PPPoE) and PPP over ATM (PPPoA) – are ubiquitous as underlying access technology to Internet, e.g. by DSL modems. PPP is able to connect two nodes without the need for any underlying networking service or device.

PPTP enables the creation of a tunnel between an *Access Concentrator* (PAC) and a *Network Server* (PNS), working directly at Level 2 [36]. The encapsulation is performed by an extended version of the GRE protocol. In addition, PPTP establishes a PPP connection and uses TCP to exchange control messages [14]. Unlike PPP, PPTP does not provide encryption, and must be combined with other tools to provide secure communications.

### 2.3.2 Layer 2 Tunneling Protocol

The *Layer 2 Tunneling Protocol* (**L2TP**) works directly at Layer 2, and encapsulates the data-link (e.g. Ethernet) frame in a UDP packet. L2TP adjuncts its header to the UDP packet in order to be able to rely control flags/registers to provide networking serivces, e.g. besides packet characteristics as length and port number. L2TP encapsulates a Layer 2 frame without providing security services. It is usually combined with IPSec (see Section 2.3.3) to this end.

### 2.3.3 IPSec

IPSec is originally an effort by the IETF to extend the TCP/IP stack with security features. It was originally designed for IPv6, but the specifications made it compatible with IPv4.

It works at Layer 3, and creates a tunnel able to transmit IP packets transparently to applications, i.e. without the need to modify the application source code [2]. IPsec can be used for net-to-net connections – i.e. connecting two networks together –, host-to-net – i.e. a remote device connects to a network –, and host-to-host – i.e. two devices connect together across the network. Moreover, it can be used in two modes:

- *transport mode*, which provides encryption and/or authentication, but only on the payload of the IP packet. It is limited to host-to-host situations, and inserts an IPsec header between the the original IP header and its payload.
- *tunneling mode*, which provides encryption and authentication on the *whole* IP packet. This is, by far, the most popular use of IPsec, in combination with a net-to-net configuration. In this mode, IPsec encrypts

the whole original IP packet, and creates a new IP packet, adjunct with an IPsec header.

IPsec is one of the most complex protocols in use over the Internet. It is actually composed of a series of protocols, each with a specific function.

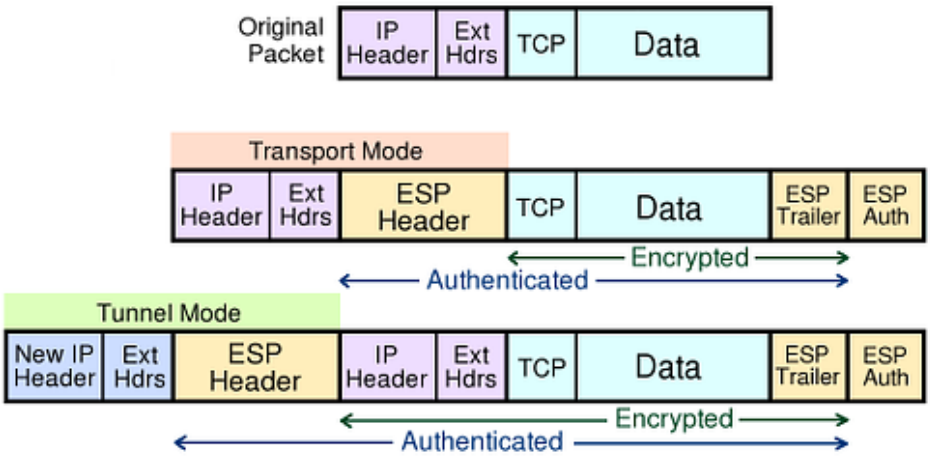


**Figure 2.9:** IPSec tunneling methods

The *authentication header* (AH) protocol provides source authentication and/or data integrity. The *encapsulation security protocol* (ESP) provides source authentication, and/or data integrity, and/or integrity. Finally, the *security associations* protocol provides the necessary algorithms and parameters to carry out AH and ESP operations. It is, in a way, the "control protocol" for IPsec, able to mutually authenticate the communicating parties, negotiate and exchange security parameters, and create the IPsec connection between the entities. To establish an IPSec connection, it requires two phases:

1. performs mutual authentication of client and server, and key exchange;
2. negotiates the cypher and authentication algorithm.

IPsec is thus a connection-oriented protocol, even though it works at layer 3 (whereas IP is stateless and connectionless).

### 2.3.4 Transport Layer Security

*Transport Layer Security* (**TLS**) is a protocol derived from *Secure Socket Layer* (SSL). SSL was created in the early 1990's by Netscape, in a bid to provide security to e-commerce. By design, SSL fully relied on TCP (thus, working at/above Layer 4). This original design choice made SSL much simpler, in conception, than IPsec, because it can rely on TCP's well-known-and-established strengths (e.g. in-order-packet-delivery, retransmits). In the late 1990's, the IETF standardized Netscape's original design, leading to the family of TLS protocols that we rely on today.

This protocol is mainly used in web browsers to provide security through encryption. It is usually preferred to secure applications since it is implemented directly at the application layer. However, this means that an application should be modified, at code-level, in order to take advantage of TLS. Nowadays, this usually simply entails calling the appropriate API to secure sockets, and not simply to simple TCP sockets.

The most complex part of TLS is the TLS handshake, which occurs after the TCP three-way handshake, i.e. right after a TCP connection is established. The goal of the TLS handshake is for communicating parties to authenticate each other, and negotiate security parameters, such as cipher suites, MAC algorithms, etc.

While originally designed for host-to-host communications (originally described as application-to-application), the OpenVPN consortium has extended TLS to be able to provide net-to-net connections, or host-to-net connections, by relying on a custom layer/protocol on top of TLS. TLS-based VPNs are popular for their ease-of-use, particularly in terms of configuration, when compared to IPsec.

### 2.3.5 Discussion

The biggest added value of VPNs compared to overlay networks, is the security VPNs bring to the communications. For this reason, we decided to exclude PPTP and L2TP from being contenders as PrEstoCloud network overlay protocols. According to recent research studies, [60] and [36], IPSec and TLS – the two remaining tools from our list – offer similar performance.

Hence we decided to keep for further evaluation IPSec and TLS. The first one because of its wide distribution and support, even if it is notoriously considered hard to configure. Regarding TLS, we decided to keep it in our list of candidate building blocks because of its famous implementation in the OpenVPN suite [65].

## 2.4 Conclusion

In this Chapter, we reviewed the solutions available to connect a private infrastructure, or private cloud, to a public cloud, and those available to connect multiple public clouds from different vendors together. In the first part of the Chapter, we first reviewed the networking features available in private and public IaaS platforms. Then we reviewed and discussed standardized protocols, as well as state-of-the-art research efforts to establish overlay networks in the context of multi-hybrid cloud computing.

We showed how the built-in features of public clouds are incompatible with one another, leading to a vendor lock-in effect. We then reviewed network overlay protocols and elected VXLAN and STT as best fit within their category. SDN-based overlays are still deeply rooted in research territory. Moreover, in the context of PrEstoCloud, the network overlay will be predominantly deployed between the data center sites elected to run a distributed big-data application. As such, the established overlay should allow VMs located within any data center to establish connections and exchange data with any other VM, in any other cloud. The number of data centers involved in the application deployment may, of course, vary through time, as new sites may be elected (or dismissed) for (or from) deployment for optimisation reasons. Still, the number of selected data centers will be order of magnitude lower than the number of virtual machines. Therefore, even though the PrEstoCloud infrastructure will exhibit dynamicity in terms of application placement and VM instantiation, the underlying network will be of a more *static* nature. Consequently, it appears unnecessary to rely on SDN-based overlays, as the main advantage of SDN solutions is the higher granularity available for creating traffic rules.

Based on the analysis carried out in this chapter, we advocate to base the PrEstoCloud overlay network on VPN tunneling techniques. VPNs strike a healthy balance between complexity and flexibility. A VPN-based inter-cloud model provides an optimal architectural solution to the problem, while minimizing performance overhead. Moreover, it allows to build a general solution, independent of any cloud provider.

While VPNs still seem to feature a series of obstacles regarding configuration and implementation in the cloud environment, VPNs offer key characteristics, as encryption and possibly throughput performances, that give them some advantages compared to the others solutions. Considering the inapplicability of cloud-specific solutions, by means of vendors services and tools incompatibility, we deflected to an external approach. Aiming for something more lightweight and compatible with our conditions, we discarded the overlay network approach from our set of solutions, leaving VPNs as the preferred option to focus on.

We further mention at this stage that the literature outlays unsettled opinions about the implementability of VPN techniques to the cloud. On one side, Moreno et al. [59] mention that the public clouds use of NATs,

to translate the public addresses to the VMs in their network, can cause problems to VPN creation. On the other side, Dayananda et al. [22] propose an architecture that uses IPsec to create an inter-cloud environment. The idea is to have a central *hub-gateway* that maintains fixed tunnels to all the connected clouds, and when needed, dynamically creates tunnels between the clouds. This avoids an overhead of tunnels, using just the ones needed. The hub-gateway works as a central manager of the inter-cloud. Unfortunately, the proposed architecture has not been evaluated nor implemented in any way. So this solution doesn't discredit the NAT obstacle previously mentioned.

# Chapter 3

# Architecture and Implementation

In the previous chapter, we reviewed the set of technologies available for connecting multiple hybrid clouds together so that instantiated virtual machines could communicate with one another. We settled upon Virtual Private Networks (VPNs) as the underlying component for the PrEstoCloud overlay network. In this Chapter, we describe our proposed architecture. First, we explain the core choices behind our architecture, which we then describe as a whole. Finally, we present the implementation challenges related to the architecture, and the solutions we rely on.

## 3.1 Architecture

In this Section, we detail the reasons behind the core choices for the architecture of the PrEstoCloud overlay network. We review and consider the different possible methods and topologies to connect the clouds together. Doing so, we introduce the PrEstoCloud VPN Gateway (Section 3.1.1), detail its form and functions, and how the VPN gateways will be connected together.

### 3.1.1 VPN Gateway

The **VPN Gateway** has two main functions in the PrEstoCloud overlay network:

1. VPN endpoint,
2. Internet gateway.

The gateway will work as the VPN endpoint in charge of establishing the VPN tunnels between the PrEstoCloud sites, and of encapsulating/decapsulating the packets destined to remote PrEstoCloud machines. Furthermore, to provide an additional level of security (i.e. isolation from untrusted networks/the Internet), we set each deployment cloud network (e.g. an AWS VPC) as private. This entails that any instantiated virtual machine cannot access the Internet, or be accessed from the Internet. We setup the VPN gateway as the *only* VM instance with a *public IP address* in its deployment zone (e.g. an AWS VPC). Consequently, only the VPN gateway is able to be accessed *from* other networks. We further connect the VPN gateway to the IaaS platform's Internet gateway, so that it is also able to connect *to* remote machines. With this setup, in each cloud, the VPN Gateway is the single virtual machine connected to the local network, being able to communicate with every node in it, *and* to outside networks. It is therefore an natural place to limit the inbound connections just to the ones needed by the deployed application. In this way, all the traffic directed to and coming from the public Internet will pass through the VPN gateway, which is capable of packet forwarding. Undesired traffic can be denied entry to the VPC network.

### 3.1.2 Connection Mode

There are traditionally two ways to connect VPNs, as detailed in Section 2.3.3. These are:

- remote access, also known as host-to-net, and
- site-to-site

A remote access VPN connects *one* client to a remote network. The main use case for this connection mode are remote employees that need to access the private network of a corporation from a remote location, often with a trusted device (e.g. a corporate laptop). Site-to-site VPNs connect two entire networks together accross the Internet. It can be used to connect two remote networks of the same company (e.g. a branch office with the central headquarters), or between business partners. In site-to-site mode, from the user point of view, the networks can be accessed as if there were only one.

Site-to-site VPNs rely on two endpoints, usually in a client/server configuration, to create a tunnel. As detailed in Section 3.1.1, the VPN gateway will be used as the VPN endpoint in each cloud, and will also the only instanciated VM in each deployment zone reachable from outside of the IaSS platform. The VPN gateways are then ideal to create a VPN tunnel in site-to-site mode. This implies that no overlay-specific configuration is needed within the end-application VMs. Moreover, every application VM will be part of the extended LAN, and, therefore, any requried connection among these VMs, within this extended LAN, will be done transparently to the application, without the need for modifying the application itself.

### 3.1.3 Overlay Topology

There are multiple possible topologies when connecting multiple sites togetether. In this Section, we review the possible topologies for the PrEstoCloud overlay network.

Moreno-Vozmediano et al. [59] evaluate three different topologies for a hybrid cloud environment. These topologies are:

- the star topology (Figure 3.1),
- the tree topology (Figure 3.2),
- the mesh topology (Figure 3.3).

The evaluation relies on a *Federation Agent* (FA), which, for the purpose of this discussion, we can assimilate to the VPN gateway presented in Section 3.1.1. The FA is used to create the tunnels that connect the different locations endpoints.

The star topology, depicted in Figure 3.1, uses a unique FA, placed in one of the clouds – usually in the private cloud or private infrastructure of the user. The FA creates a tunnel directly to every other instance in all other clouds (i.e. to all remote VMs). The FA is connected to the local machines through the physical infrastructure.

The tree topology, depicted in Figure 3.2, envisions one FA per cloud. The clouds are then connected to each other using a star topology. The local FA creates a tunnel to the remote clouds FAs. Within each cloud, the instances create a tunnel to the cloud's FA, giving the tree shape.

Finally, the mesh topology, depicted in Figure 3.3, extends the tree topology, by adding a direct tunnel between all the different FAs, meaning that the remote clouds are effectively connected together.

The evaluation results provided by [59] underline that the communication performance *from local to public clouds* doesn't present significant differences between these three topologies. However, communication in within a public cloud is not efficient in the star topology, as traffic needs to be transferred back to the central

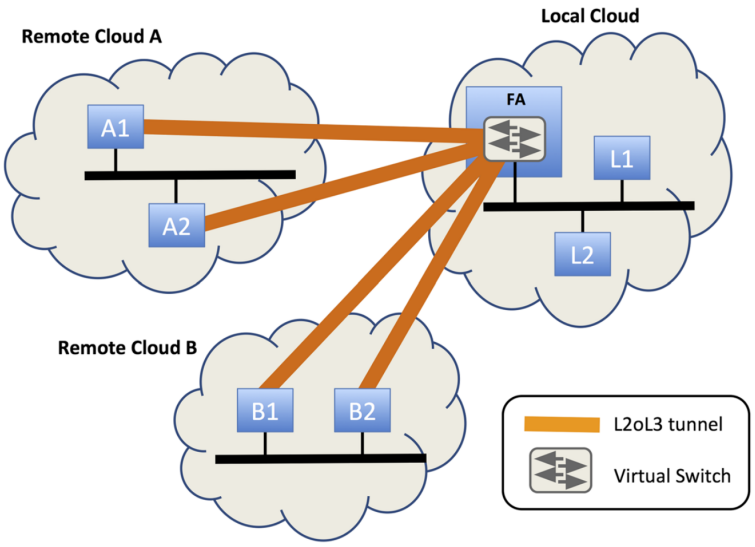


**Figure 3.1:** Star Topology

FA to reach any virtual machine, including those located in the same cloud. Similarly, the communication between different public clouds obtain important advantages from the mesh topology.

To put it differently: the star topology exhibits the worst results. The tree topology places itself in the middle: it provides good performance in local communication, but exhibits the same deficiencies as the star topology when considering inter-datacenter transfers. Meanwhile, the mesh topology is the optimal one, as it benefits from the optimization of the tree topology in the local network, and, at the same time, solves the efficiency problems in the inter-datacenter transfers by adding one tunnel between each couple of FAs.

Grozev et al. [25] provide a more general approach to inter-cloud architecture, rather than a survey of network-level topologies. The presented approaches are summarized below.

- Centralized topology: a central entity maintains a repository that stores all the cloud resources. Based on that, it performs resource allocation. This can be associated to the star topology seen above.
- Peer-to-peer topology: the clouds are directly connected to each other. This is comparable to a mesh topology.
- Multi-cloud services: similar to a centralized topology, but with the difference that the clients use services (e.g. RESTful services [15]) to contact the clouds instead of network-level tunnels.
- Multi-cloud libraries: similarly, but where each client has to develop their own broker using a set of API given by the different clouds.

These last two topologies go out of our scope since we aim to a more network-level architecture.

Consequently, for the PrEstoCloud overlay network, we also opt for the mesh topology, so as to optimize inter-datacenter operations. This is particularly important for several reasons. First, to avoid additional delays in inter-datacenter transfers; second, since public clouds charge network transfers depending on usage, depending on the location of the centralized FA, cost-per-network-transfer would be charged multiple times.

### 3.1.4 Complete Architecture

The final architecture is the combination of the elements outlined in the previous sections. Figure 3.4 depicts the PrEstoCloud overlay architecture in a multi-cloud scenario. In the particular case of Figure 3.4, an applica-

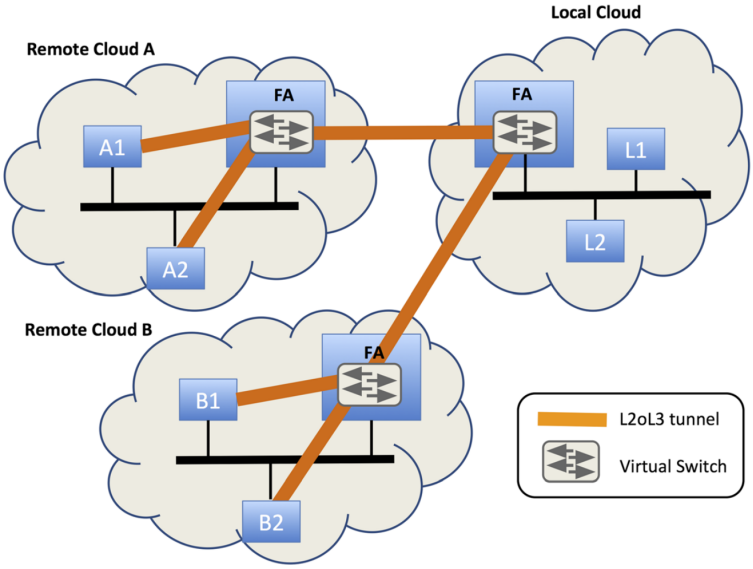


**Figure 3.2:** Tree Topology

tion is deployed over multiple VMs, spanning across two public clouds (Amazon Web Services, and Microsoft Azure), and one private cloud (running on OpenStack).

As shown in Figure 3.4, we place one VPN Gateway in each cloud network: one in AWS, one in Azure, and one in the private Open Stack. For the sake of simplicity of the description, we abstract the fact that each public cloud can be further subdivided in multiple zones, as detailed in Chapter 2. Although we don't rely directly on cloud-specific services to enable our overlay (see Section 2.1.5), we still need to rely on cloud specific features to integrate our architecture.

First, we use the IaaS platform's network subdivision (e.g. VPC and subnets), to define the pool of IPs to be used in the private networks, thereby leaving to the IaaS the task of IP assignment. Second, we rely on the IaaS's virtual switch to enable communications among instances in the same cloud, without the need of creating a VPN tunnel at that level. Compared to the topologies presented in Section 3.1.3, we do not require a local VPN connection between the VM instances and the VPN gateway. Isolation of the VM instances is already achieved through cloud infrastructure isolation, such as the virtual private network. Finally, an elastic IP (or, more generally, a static public IP address) is assigned to the VPN Gateway, to make it reachable from the outside. It is also needed to establish the VPN connection between client and server.

There are two crucial advantages to retain this particular design for the PrEstoCloud overlay network:

1. The combination of VPN tunnel encryption, and of placing the computing units in a private network ensure privacy and security to the deployed PrEstoCloud application.

2. We minimize the overhead by just connecting the gateway machines to each other. In this way any traffic moving within a single cloud will neither have encapsulation overhead, nor encryption overhead, thereby optimizing the throughput within a cloud.

The proposed architecture is also a general solution to the hybrid cloud networking problem. The VPN gateway is a machine that can be deployed in every cloud environment, regardless of the vendor (more details in Section 3.2). The VPN connection is then configured in an ad-hoc manner in the gateways. In this way, we are simply installing an application from the cloud provider's point of view. Hence, applying the correct configurations, we bypass all the infrastructural incompatibilities that would exist if we had used the IaaS platform's VPN service.

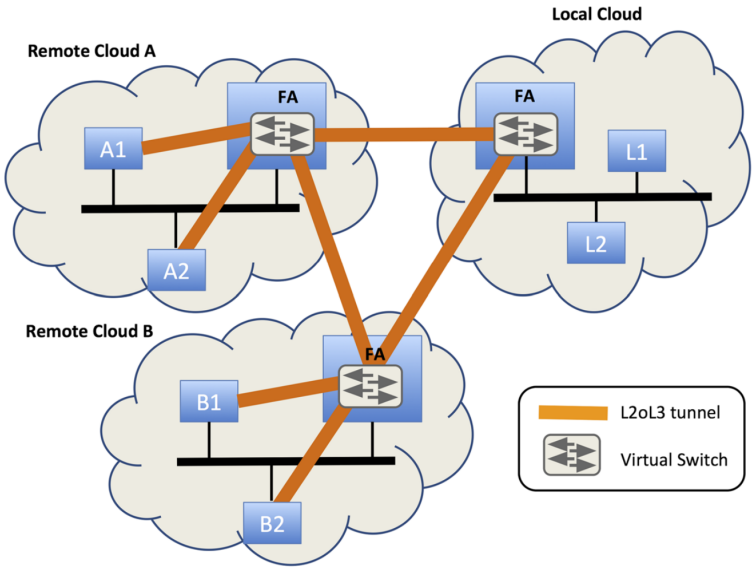


**Figure 3.3:** Mesh Topology

## 3.2 Implementation and Challenges

In this Section, we provide the implementation details behind the first iteration of the PrEstoCloud overlay network. This initial version will be gradually improved and integrated into the wider PrEstoCloud platform through the work carried out within Work Package 4 and Work Package 6.

### 3.2.1 VPN Support Implementation

IPSec is natively supported by Linux. However, a wrapper software is required to manage the IPSec system services, the configuration, and all the protocols supported, like the key exchange service. There are several packages that implement IPSec, all of which perform the same actions, and can be considered equivalent. The differentiator is the supporting community. These packages are LibreSwan [41] and StrongSwan [80]. We opted to rely on StrongSwan, as it is officially supported on more systems than LibreSwan. Moreover, the latest release is more recent.

As described in Chapter 2, TLS was not born as a VPN technology. It has been subsequently implemented as such by several different software packages. The main implementations of TLS as VPN are OpenVPN [65], SoftEther [77], and OpenConnect[63]. We opted to use OpenVPN, which, out of these, is the most popular package due to its ease of configuration.

### 3.2.2 Initial Overlay Implementation

Considering the need for future integration of the network overlay component within the overall PrEstoCloud platform, we developed an automated software that could be easily integrated within the platform. In particular, our approach to the development of the prototype for the setup of the PrEstoCloud overlay network has been deeply influenced by the design philosophy behind ProActive, the autonomic manager used within PrEstoCloud (see deliverables D4.1 – Autonomic Resource Manager, and D4.3 – Autonomic Data-Intensive Application Manager). The prototype is thus designed to work in a full GNU/Linux environment (see Section 3.2.3), with support of bash [12] and Python [71]. The implementation comprises the full workflow: from cloud provisioning of the VMs, to the configuration and establishment of the VPN connections. While some of these functions will eventually be taken over by ProActive (particularly the provisioning part), designing the software

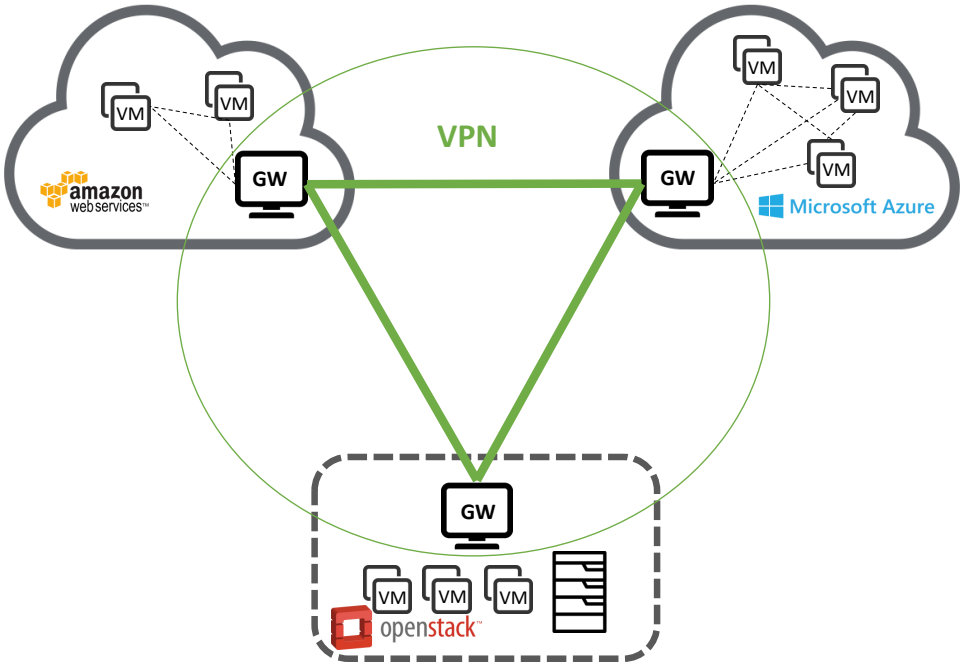


**Figure 3.4:** VPN-based Inter-Cloud Architecture

from the ground up provides a deeper understanding of the elements at play in the overlay establishment, as well as tailors the list of features later required from ProActive. The prototype is envisioned for establishing an overlay between AWS, Azure, and a local OpenStack instance.

The operation performed by the initial overlay implementation can be decomposed as the three following phases.

- Phase 1: provision of the AWS and the Azure clouds (detailted Section 3.2.3). Both clouds provide sets of APIs for a wide range of languages. We rely on the custom command line interface (CLI) APIs [3], [4] directly executable through bash.
- Phase 2: generation of the cryptographic keys for the authentication and encryption to be used by the VPN back end. These keys are different for all gateways, and pre-shared for mutual authentication. The keys distribution will be performed in the last phase respectively for VPN server and client.
- Phase 3: set-up and configuration of the VPN servers, and VPN client machines.

The context of operation for the prototype is to consider an up-scaling operation from the local OpenStack cloud. Namely, we are able to configure the local (OpenStack) gateway, but we need to remote configure public clouds, and VMs instantiated in these clouds. For this reason, we also rely on Ansible [10] to help software deployment on remote machines. Through an SSH connection, Ansible is able to run commands, copy files to and from remote machines. Eventually, the functions provided by Ansible will also be taken over by ProActive and its framework.

### 3.2.3 Public Cloud Configuration

To federate a public cloud into the overlay, the first thing to do is to provision the VPN gateway VM and configure it as needed. First, we create a virtual network (or VPC) in each cloud, composed of one chosen subnetwork, that provides the pool of IP addresses that will be assigned to the instances. The challenges within this phase are:

- the IPs over the different clouds cannot overlap;
- the traffic directed to other federated clouds must be forwarded to the VPN Gateway;
- only the ports necessary for VPN establishment and gateway configuration shall be opened;
- adapt the cloud system, particularly the default behavior of its network, to the one that we require from the overlay.

In the remainder of this Section, we detail the steps necessary to overcome these challenges.

#### 3.2.3.1 IP Numbering Plan

To make the architecture transparent to the application, IP addresses should be unique across the whole overlay. For this reason, it is necessary to adhere to a specific numbering plan, so as to avoid that two VM instances on two different clouds rely on the same internal IP address.

We established a private IP numbering scheme to separate the different clouds and networks in the PrEstoCloud architecture. This plan extends the 24-bit block defined in RFC1918 [72] for private networks: 10.0.0.0/8. It is the largest private IP address block available.

More precisely, we use the following numbering plan. A machine is allocated the following IP address:

`10.A.B.C`

where To better illustrate with examples, in our prototype, `A=1` for AWS, and `A=2` for Azure. Furthermore, the

| | |
|---|---|
| A | Identifies the cloud provider |
| B | Identifies multiple deployments in the same cloud provider |
| C | Identifies the virtual machine |

**Table 3.1:** IP numbering scheme definition

two different VPCs in AWS will have `B=1` and `B=2`, and similarly for Azure's availability zones. Therefore, the machine `10.1.2.4` is the fourth VM in the second VPC of AWS.

In this manner, we give a global scheme that will not overlap, per construction, despite the number of clouds we have in the system. The conventions in this scheme are:

1. one different identifier for each cloud provider, as shown in Table 3.1;
2. the deployments identifier will be sequential starting from 0;
3. the host number will be automatically assigned by the provider's DHCP server;
4. the VPN Gateway uses the last IP address in range, i.e. `C=254` for the VPN gateway.

As an example: the AWS VPN Gateway will be `10.1.0.254`; the Azure VPN Gateway will be `10.2.0.254`. If we have two deployments in AWS, the second gateway IP will be `10.1.1.254`. A node inside the Azure cloud can have an IP in the range `10.2.0.1`-`10.2.0.253`; and so on.

Relying on the `10.0.0.0/24` range in the way we just described leads to the following possible number of simultaneous instances:

- 255 distinct cloud providers;
- up to 255 distinct availability zone within each cloud provider, with up to 255 distinct VMs in theses zones;
- equivalently, up to 65,535 VMs instantiated within each IaaS platform

Please note that, since we are in charge of the numbering plan, we can alter the IP assignment scheme. For example, if the need for more than 255 VMs within a single VPC arises, we can consider that `B` is a 4-bit value, therefore, that `C` is a 12-bit value, leading a total number of 4096 machines per VPC. (The VPC's subnetwork will then be a /20 instead of a /24.)

#### 3.2.3.2 Routing Tables

As previously described, the VPN gateway acts as the endpoint for the VPN tunnel. As such, all the traffic directed *from* the cloud and *to* other clouds has to be directed to the gateway machine. In order to do that, and bypass the IaaS's own virtual switch, we need to modify its routing table.

We instruct the IaaS switch to forward all traffic originated from any of the nodes, and whose destination is not within the local network's range, to be redirected to the VPN Gateway address.

In the particular case of AWS, the IaaS switch contains two mandatory rules, which cannot be removed. They are

1. the default traffic is directed to the Internet gateway;
2. all the traffic directed to the VPC (`/16`) is forwarded to the switch.

This last rule is an obstacle as it prevents us from creating a more specific rule (e.g. a `/24` rule). Nevertheless defining a less specific `/8` rule solves the issue. In other words, we instruct the AWS switch to redirect to the VPN gateway all traffic related to the overlay network address (i.e. `10.0.0.0/8`). Then, the VPN gateway will know how to orchestrate the traffic.

#### 3.2.3.3 Security Groups

By default, both AWS and Azure allow all outgoing traffic, and deny any kind of inbound traffic. To customize this behavior, we rely on **Security Groups** (SGs). An SG defines a set of inbound and outbound rules that are fully configurable by the user. By default, an SG is assigned to a VPC. But, if necessary, it can be assigned to multiple VPCs.

We rely on a generic SG for the overlay, which contains all the rules to allow the traffic of the in. Table 3.2 lists these rules.

#### 3.2.3.4 Provider-Specific Configurations

##### Overriding Default NAT Behavior

Even if IP forwarding is enabled on the VPN gateway, the traffic does not flow across clouds without some

| Port | Protocol | Service | Reason |
|---|---|---|---|
| 22 | TCP | SSH | to connect to the machine from the outside |
| 443 | TCP | OpenVPN | TCP tunnneling |
| 1194 | UDP | OpenVPN | UDP tunneling |
| 500 | UDP | IPSec | IKE key exchange and authentication |
| 4500 | UDP | IPSec | IPSec NAT Traversal |

**Table 3.2:** Security Group inbound rules in the cloud network

further configurations.

Cloud providers rely on NATs to translate the addresses of the machines, and to send the packets to the nodes within a VPC. This NAT can block the overlay traffic, meaning the nodes are not able to send or receive traffic to/from other clouds. For this reason, it is crucial to alter the two following parameters to allow the traffic to be forwarded by the VPN Gateway:

- For AWS: source/destination check [55]. In AWS each instance performs traffic checks, whereby an instance accepts traffic only if it is the source or the destination of it. This option needs to be disabled on the VPN gateways to allow them to manage the traffic and fully work as a relay.
- For Azure: IP Forwarding [56] flag. In Azure, the behavior is similar to AWS, and the IP Forwarding flag needs to be turned to true for the cloud instances.

#### VPN Gateway and OpenStack

In OpenStack, when a network is declared as private, an Internet router cannot be added to the network. This means that we cannot route the PrEstoCloud VPN gateway to the Internet. To overcome this limitation, we create an intermediate auxiliary network in which we insert an OpenStack router, which is connected to the Internet.

We then create the VPN gateway VM in this auxiliary network. Once it has booted, we configure its network interface to use the OpenStack router. Then, we create an additional virtual interface and attach it to the virtual machine. Finally, we attach this interface in the private network. Consequently, the VPN gateway has an interface in the deployment network, but also in the auxiliary network, which is routed to the Internet via the OpenStack router.

This design is effectively a demilitarized zone.

### 3.2.4 VPN Configuration

OpenVPN and IPSec both require different configurations and supporting toolsets. For example, they use two different services for the key generation: X.509 certificates for OpenVPN, and IKE for IPsec. Moreover, they both rely on a static configuration file. This file needs to be adapted each time, depending on how many clouds need to be interconnected, depending on the the subnetwork addresses, and the role of the gateway (i.e. VPN client or server).

#### 3.2.4.1 Client/Server Configuration

VPNs establish tunnels following the client/server paradigm: the client connects the connection to the server, and after mutual authentication, establishes the tunnel. This means that the tunnel is created just between the server and the clients, not between the clients themselves. Therefore, we cannot simply put a VPN endpoint where tunnels are to be connected, each VPN endpoint has to be tailored to the correct client/server configuration.

In a basic configuration with three clouds, given one VPN server, it is easy to create a tunnel to the other two, which contain VPN clients. However, in that fashion, the two VPN clients are not connected to each other, thereby failing to achieve the envisioned full-mesh topology. For this reason, it is necessary to add another VPN server in one of the other two clouds, in order to create a tunnel between the two. Please note that a single VPN client can easily connect to multiple servers.

We propose to install inside each VPN gateway *both*, a VPN client and a VPN server. In this way, clouds will be able to connect to one another to form the full-mesh topology. Moreover, this way, we also ensure fault tolerance for VPN tunnels. For example, if the initial connection from a client to a server does not work (e.g. due to NAT traversals), the connection can reverted, and try to connect it in the opposite direction. This architecture is depicted in Figure 3.5.

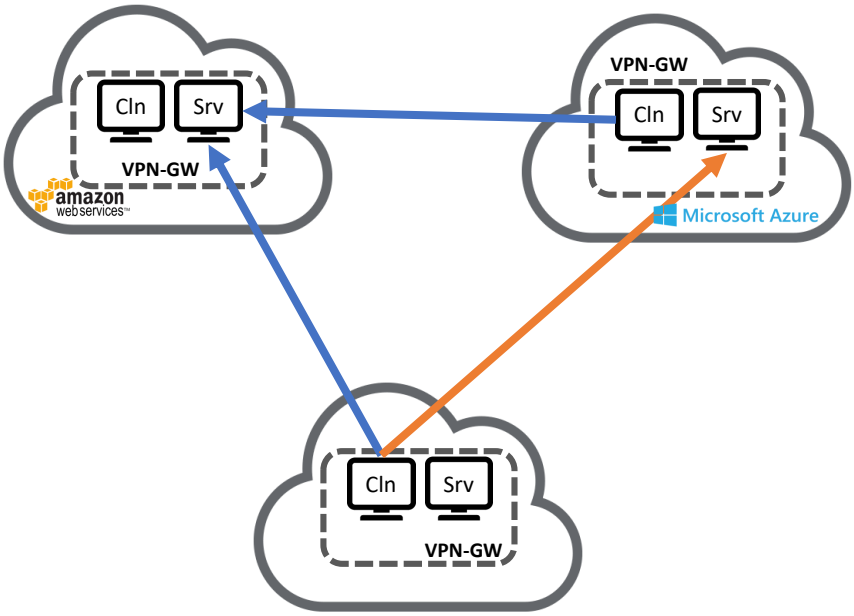


**Figure 3.5:** VPN Client/Server configuration in the Inter-Cloud prototype

#### 3.2.4.2 Key Generation

OpenVPN uses the RSA encryption scheme [30] to generate the certificates. The program Easy-RSA can be used to create the certificate authority (CA) and key generation. IPSec includes its own sub-protocol, the Internet Key Exchange protocol (IKE [28]), to perform the same actions.

Both approaches follow roughly the same path to create the required keys, even if small differences occur:

1. first, set-up the certificate authority and generate its certificate. The CA is is the digital notary in charge

of signing public keys, thereby attesting the association between a key and an entity such as the VPN gateway. OpenVPN just saves the information about the certificate authority, and automatically encloses them at the creation of a new certificate. IPsec requires to manually create the certificate for the certificate authority, which will be then used to sign the public keys.

2. second, generate the keys for the servers and the clients. In both cases, the procedure is the same. Easy-RSA creates keys and certificates with one command. With IPSec, we first need to create the key for the node, and then sign it with the CA's certificate to generate its own certificate.

The key distribution is performed through Ansible, copying the files to the specific folders in the remote machines through a secure SSH connection.

#### 3.2.4.3 Sites Connection

Besides the basic configurations, such as the IP of the server, the pool of IPs to be assigned to the clients, the encryption algorithms, etc, the site-to-site configuration needs to be appropriate on both VPN endpoints.

To better understand why each need to be configured differently, we first specify how the routing mechanism works for each of the VPN technologies:

- OpenVPN creates a virtual tunnel interface in the machine, for each tunnel it creates. The traffic routing is performed by adding rules in the Linux route table, and redirecting the traffic to the virtual interface.
- IPSec manages the traffic using Linux `iptables`, without creating any virtual interface. Every time a new IPsec connection is created, a set of network policies are derived from the configuration files, and automatically add them to `iptables`. Consequently, the traffic directed to the VPN's IP range will be correctly redirected to the IPSec tunnel.

It is therefore essential for both ends of the VPN tunnel to have the symmetric configuration, in order to be able to create the correct rules.

The configuration file for StrongSwan (IPSec) is divided into two zones – right and left –, both of which contain a set of variables. The left variables correspond to the machine in which the IPSec service is running; the right variables correspond to the other side of the tunnel. In the case of the VPN server, we specify:

- right subnet: the subnet behind the VPN server. This lets the clients know that they have to use the VPN tunnel to reach this set of IPs;
- left subnet: the subnet behind the VPN client.

On the client, the configuration is similar but mirrored.

For OpenVPN, the situation is slightly more complex. While in IPSec, we just have to define the subnetwork of the site to which we are connecting, in OpenVPN we have to specify which routes have to be added in the routing tables through the `push` command.

1. In the server configuration file, we have to specify the needed routes, which will be sent to the client. In OpenVPN this operation is called `push` (e.g. `push "route <server_network_address> <server_network_mask>"`).
2. Inside the Client Configuration Folder (`ccd`), we have to create a file of the name of the client that is going to connect. In the server configuration file, we have to specify to check for specific clients configurations in the correct folder (`ccd`).

3. In the file `ccd/<client_name>`, we have to specify the subnetwork of the client, through an `iroute` rule. This mechanism allows the client subnetwork to have access to the VPN and vice-versa (e.g. `iroute <client_network_address> <client_network_mask>`)

Contrary to StrongSwan, with OpenVPN, the configuration is fully done on the server side.

## 3.3 Summary

In the first part of this Chapter, we focused on the architecture of the PrEstoCloud overlay network. The architecture provides minimal impact on the application, and on the VMs that run the application fragments. Instead, we deploy a dedicated **VPN gateway** per data center (e.g. an AWS site, or a private OpenStack infrastructure). The compute nodes, that run the application fragments, are configured to use this *VPN gateway* to access other sites, and the Internet. As such, the compute nodes are completely isolated from the Internet. We connect the *VPN gateways* together in a full-mesh topology, thereby guaranteeing efficient communication channels between the sites where application fragments are deployed.

In the second part of this Chapter, we delved into the implementation details. We looked at the supporting packages for VPN technologies: StrongSwan and OpenVPN. We provided the similarities and differences for their configuration, and presented how to correctly tune them to meet our needs. Furthermore, we detailed the deployment of the *VPN gateway* on the main IaaS cloud platforms (AWS, Azure, vCloud, and OpenStack), and saw how to integrate the functionality of the VPN gateway within the platform itself, which is platform dependant, and the compute nodes.

# Chapter 4

# Evaluation

We introduced the architecture and design of PrEstoCloud network overlay in the previous chapter. We report in the present chapter preliminary tests that aim at evaluating the performance of the prototype we implemented based on this specification. More precisely, we want to assess:

- the stability of the system over time: if we have drops of connection or strange behaviors due to the VPN implementation;
- the overhead of the tools: if the tunneling and encryption mechanisms we chose degrade network performance as compared to native IP routing;
- the practical differences between the two VPN tools: if we have a winner in terms of the previous two points.

To check the first point we tested the latency, for the second we tested the network bandwidth. Meanwhile, the third point will be derived from the results of the two evaluations.

## 4.1 Network Measurement Metrics

Different metrics exist to evaluate a network path. The **bandwidth** that represents the maximum amount of data that can be transferred over a link over a specific period of time. The **throughput** is the actual speed that can be obtained on a network path. Finally, the **latency** (or delay) is the time required to transfer a single empty message from a source to a destination. Knowing the delay, we can compute the **round-trip-time** (*RTT*) taking into account also the time for a packet to come back from the destination to the source.

For several years, an important area of research focused on bandwidth estimation. Three main metrics related to bandwidth estimation have been defined rom these studies:

- **Link Capacity**: is the link's maximum achievable transmission rate; the smallest link capacity over the entire path defines the end-to-end capacity, this link is called *narrow link*;
- **Available Bandwidth** (*ABW*): is the actual free capacity in the link. In a link different flows compete to get a portion of the capacity. The traffic they generate is called *cross traffic* and the remaining part is the available bandwidth. The lowest available bandwidth in the path represent entire end-to-end available bandwidth. We refer to this link as *tight link*;
- **Bulk Transfer Capacity**: is the maximum throughput achievable by a single TCP connection. It is commonly computed by stressing the network with high transfer rates and can overcome the ABW, e.g., when the cross traffic consists of TCP transfers. Indeed, in the latter case, the congestion control algorithm of TCP will seek to a new equilibrium corresponding to a fair share of the available bandwidth between the

competing TCP flows. An extreme example is the case when a single TCP flow is able to fully utilize a link. In this case, the available bandwidth is equal to zero but the Bulk Transfer Capacity should be equal to 50% of the link capacity if the two TCP connexions are homogeneous (same RTT, same flavor of the congestion control algorithm). In practice, however, TCP connexions are often limited by the application on top [27].

Obviously, when we try to inject new traffic to the network, we are are not limited by the capacities of the links, but rather we have to consider all the traffic that already circulates in the network, making the available bandwidth the true bottleneck for our application.

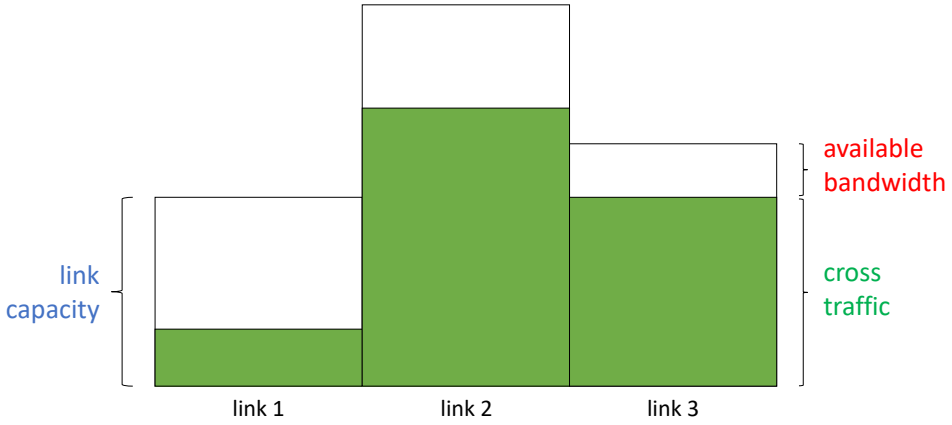


**Figure 4.1:** Network metrics representation: the link capacity is represented by the upper black line, the green part represents the cross traffic, the white part is the available bandwidth

## 4.2 Latency

### 4.2.1 Methodology

Our measurement set-up is the following. We connected a physical machine (Table 4.1) in our lab (I3S/CNRS), that we use as VPN gateway. It is connected to two different cloud deployments: one in AWS and one in Azure (Table 4.2).

| | |
|---|---|
| **OS** | Linux Ubuntu 16.04 64bit |
| **CPU** | Intel Core 2 Duo E7300 @ 2.66GHz |
| **RAM** | 4GB |

**Table 4.1:** Specification of the local machine at I3S (CNRS)

We deployed the free tier instances in AWS and the smallest one in Azure. To simultaneously test IPSec and OpenVPN, we had to add a second network interface (NIC) in each of the instances. Indeed, if we assign the same NIC address to both the VPNs, there will be concurrency over traffic forwarding between the tools and they won't work.

| | AWS | Azure |
|---|---|---|
| Region | Frankfurt (EU Central 1) | Netherland (West Europe) |
| Instance Type | t2.micro | Standard_DS1 |
| Instance Image | Ubuntu 16.04 | |
| IPSec Interface | 10.1.0.0/24 | 10.2.0.0/24 |
| OpenVPN Interface | 10.1.1.0/24 | 10.2.1.0/24 |

**Table 4.2:** Cloud testing deployment configuration

The final configuration of our testbed results in three VPN gateways in three different clouds, connected to each other through OpenVPN, IPSec, and public Internet. We performed a 24h-long ping test in which we send one ICMP packet every 1 second. In order to have comparable results, we simultaneously tested the three different paths: native IP (i.e. no tunneling), IPSec tunnel, and OpenVPN tunnel. This is illustrated in Figure 4.2, where the solid lines correspond to the native IP routing, while the dashed ones correspond to the VPN tunnels.

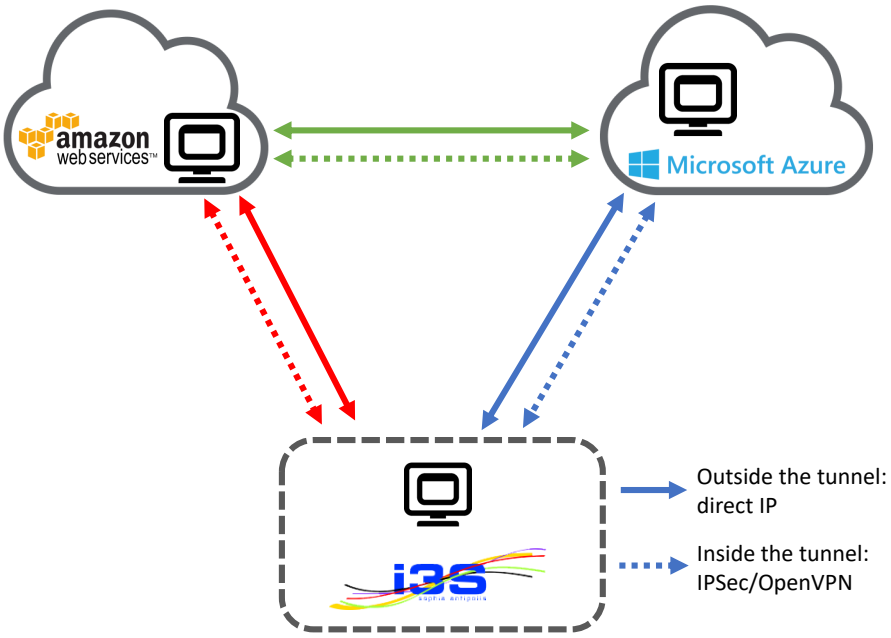


**Figure 4.2:** Testbed configuration for the latency test

### 4.2.2 Results

#### 4.2.2.1 Native IP Routing

In Figure 4.3 and Figure 4.4, we report the results obtained for the three paths. The time series view (Figure 4.3) allows to observe possible trends and correlations between the measurements, while the cumulative distribution function (CDF) in Figure 4.4 allows to better infer the relative performance of the tools. The path between

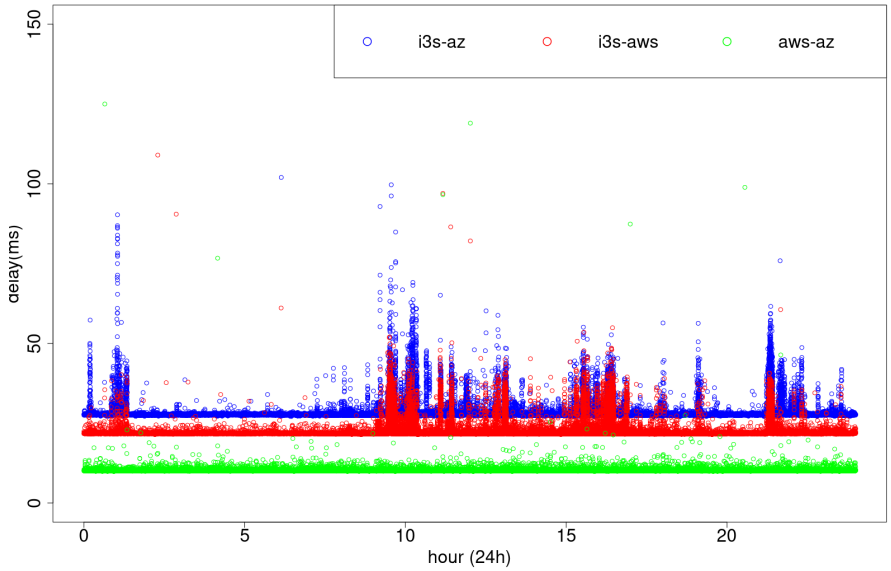


**Figure 4.3:** Time series (24h-long ping test with a packet interval of 1 second) of the three paths performed through public Internet: I3S/CNRS to Azure (blue), I3S/CNRS to AWS (red), and AWS to Azure (green)

the I3S laboratory and the VM in the Azure Cloud plotted in blue, the path from I3S to Amazon AWS in red, and the AWS-Azure path in green.

The CDF shows us that the AWS-Azure path is really stable and considerably shorter than the two others. It has a 95th percentile value around 10ms, meanwhile, in the path between I3S and the two clouds we see that we have 10% of outliers, 5% more than the AWS-Azure path. In fact, looking at the time series of Figure 4.3, we have various latency spikes during the day that affect the path.

These delays indicate the lower-bound that can be expected from the performance of the PrEstoCloud

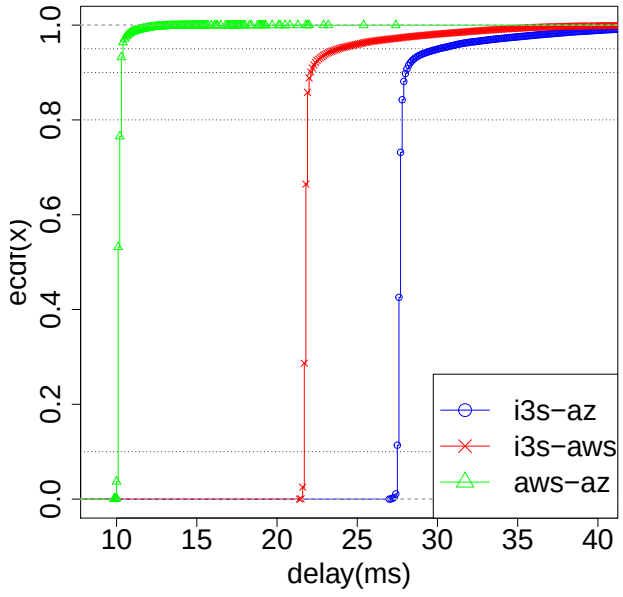


**Figure 4.4:** CDF of the three paths through the public Internet: I3S/CNRS to Azure (blue), I3S/CNRS to AWS (red), and AWS to Azure (green)

overlay network, as they reflect the latency that exists between the PrEstoCloud VPN gateways on the three different sites, on the IP level. The overlay network will be built on upper levels.

#### 4.2.2.2 Tunnels

We noticed the very same spikes observed for the native IP case and for the tunneled results, proving that the path stability is not due to IPSec or OpenVPN. Instead, the latency variability is due to network infrastructure instability whose root cause is apparently close enough to I3S Labs to affect both the path leading to the Azure and AWS data centers we consider.

We next focus on the the I3S-Azure path as the other path is qualitatively similar. In Figure 4.5 the two tools have the same cumulative distribution as the not encapsulated one (public), and the overhead impact on the latency is minimal. This is better visible in the boxplot representation shown in Figure 4.6. In a boxplot, the middle bar inside a box is the median, while the upper and lower edges are the 75-th and 25-th quantiles respectively. Extreme values, that are far from the core of the distribution, are represented as crosses. Figure 4.6 shows that we have less than one additional millisecond when using OpenVPN, and even less with IPSec. Similar results are obtained in the Amazon-Azure connection (Figure 4.7). Furthermore, the results suggest that the shorter is the link, the smaller the tool overhead.

### 4.2.3 Traceroute

We further investigated the difference of latency between the two I3S-cloud paths. We can see a higher delay to reach the Azure datacenter. The AWS and Azure datacenter are placed respectively in Frankfurt and Netherlands. Even when accounting for the geographic distance, we shouldn't observe a difference as high as 5ms to reach Azure. Performing a traceroute to the two IPs we discovered that the packets follow the same path until the Paris exchange point for both the connection, then it needs 7 hops more to reach the Azure datacenter than the Amazon one. Despite this is not a strong motivation for the latency difference, we can still suppose that the 7 hops have an important impact on the latency, considering that the intra-datacenter traffic should

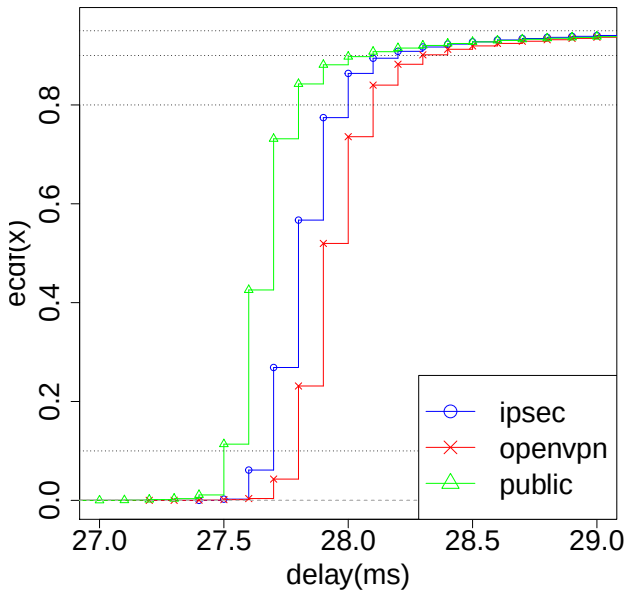


**Figure 4.5:** CDF of the I3S-azure path through the three channels: IPsec (blue), OpenVPN (red), and public Internet (green)

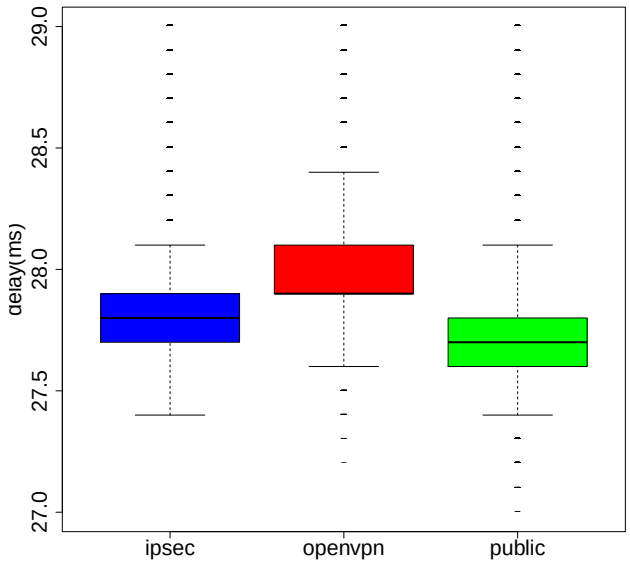


**Figure 4.6:** Boxplot of the I3S-Azure path through the three channels: IPsec (blue), OpenVPN (red), and public Internet (green)

have a similar delay in both clouds.

Furthermore, Having the two I3S-cloud flows following the same path we suspect that the network instability problem source is near the I3S laboratory, particularly in the path that connects i3s to the Paris exchange point.

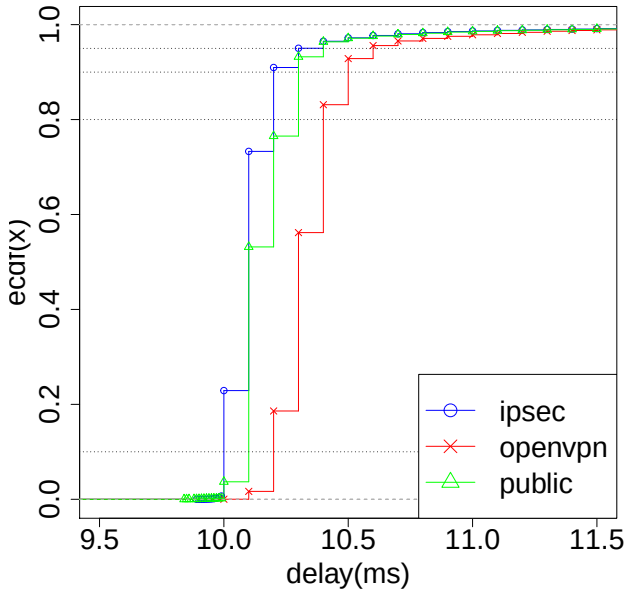


**Figure 4.7:** CDF of the cross-cloud (AWS-Azure) path through the three channels: IPsec (blue), OpenVPN (red) and public Internet (green)

## 4.3 Throughput

Data is the most valuable asset in nowadays industry, and this is true also for cloud services. The data transfer cost can become a prominent factor on the final bill of the user. Note that in general, incoming bytes are free of charge and only outgoing bytes are charged. For the sake of illustration, Table 4.3 reports the cost of inter-region data transfer.

| Region | Amazon (€/GB) | Azure (€/GB) |
|---|---|---|
| **EU** | 0.02 | 0.0734-0.0422 |
| **US** | 0.02 | 0.0734-0.0422 |
| **SA** | 0.16 | 0.1527-0.1350 |
| **AP** | 0.09 | 0.1164-0.1012 |

**Table 4.3:** Inter-Datacenter data transferring cost, as of Sept. 15, 2016. For Azure, data transfer costs scale with volume. [70]

Table 4.3 shows how expensive it would be to perform bulk transfer tests in a cloud network. For that reason we decided to follow a less intrusive methodology, trying to understand the available bandwidth of the paths, using some state-of-the-art available bandwidth measurement tools developed in the mid-2000.

We foresee some challenges with this methodology: the available bandwidth tools are old and designed for a physical environment. In contrast, we are working in a virtualized scenario that could alter the behaviour of these tools. In addition, our environment presents much faster links than was available over a decade ago.

### 4.3.1 Network Measurement in the Cloud

The literature concerning bandwidth measurements in cloud environments is relatively limited. The closest work that we can find is the series of publications by Persico et al. [66]–[70] where the authors study the intra-datacenter and inter-datacenter throughput in Amazon and Azure.

In [68], the authors measure the intra-datacenter throughput between different machines in Amazon AWS. They test different sizes and different types of machines, discovering that these two factors influence the achievable throughput.

In [69], the authors analyze the Azure intra-datacenter throughput, showing that the network throughput of an Azure VM has a strong variability factor. Even if like AWS the performance is directly related to the instance characteristics, in some cases even identical machines were giving different throughput values in different deployments, leaving the user unable to control this factor.

In [67], the authors propose a platform *CloudStuf* to monitor a public cloud network. It allows to easily retrieve information about network performances, which are not usually specified by the cloud providers. However, CloudSurf still doesn't support inter-cloud monitoring. Additionally, it uses invasive tools to measure the network.

Finally, in [66] and [70], the authors study the performances of the inter-datacenter connection between different regions of the same provider, comparing AWS and Azure. They come out with some interesting results regarding the providers' behavior in this wider scenario, in particular:

- the network throughput is considerably influenced by the regions in communication. In fact, connections to Asia Pacific and South America are less performing.
- there is a performance asymmetry between the regions: we can encounter different throughput values

and have stability issues depending on which direction we are testing the two regions.

- they observe an on-the-fly network variability on Azure: after 120 seconds of continuous flow the provider applies a logical limitation to the throughput, resulting in an important drop of network performances.
- noticing the previous points, they performed traceroute tests in the Amazon cloud to discover how the infrastructure is really connected. In essence, the Amazon paths between the different regions are not always internal to the Amazon network (Figure 4.8): to bypass the infrastructural obstacles in some regions, Amazon can rely on third-party Autonomous System (AS).

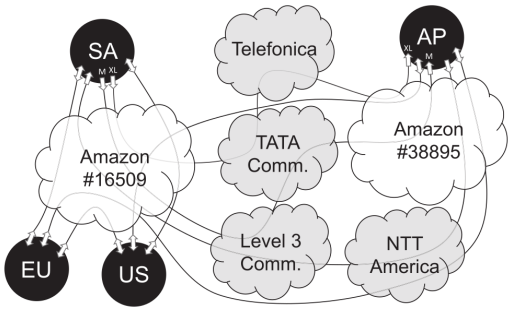


**Figure 4.8:** Amazon AS level graph of the inter-datacenter paths. Incoming and outgoing paths can be different on the basis of the directional arrows.

Despite these interesting results, all the measurements are obtained with `nuttcp` [62], which similarly to `iperf` [31], computes the bulk transfer metric. Also, as they point out in their work, data transfer between regions can be expensive. So for the scope of our study, we decided not to rely on their methodology.

Another work tries to test the cloud network: Li et al. [39] propose CloudCmp as a tool to evaluate cloud providers under different aspects: computation, storage and network. Like the previous work, for the network measurements they use `iperf`, measuring the bulk transfer capacity.

We decided to follow our network evaluation considering some available bandwidth measurement tools, whose estimation techniques are usually less intrusive (data consuming) than bulk transfer tools.

### 4.3.2 Available Bandwidth Measurement

Available Bandwidth (ABW) measurement techniques are usually classified into two groups.

1. Probe Gap Model (PGM): the idea of this technique is based on the dispersion of the packets in a path. It sends a pair of packets at a predefined gap (time interval), assuming that the gap with which the two packets arrive at the receiver has a strong correlation with the cross-traffic in the tight link. Comparing the initial rate with the received rate, it determines the amount of cross-traffic, with that it is able to derive the available bandwidth subtracting the cross-traffic from the capacity of the tight link. Examples of ABW estimation tools that fall in this category are: Initial Gap Increasing (IGI) [29], Spruce [79], and Abing [61]. We also mention Pathrate [23], that differently from the others, estimates the path capacity using the same technique and assuming that the tight link and the narrow link are the same.
2. Probe Rate Model (PRM): differently from the previous model, PRM tries to estimate the ABW using the induced congestion principle: changing iteratively the probing rate of the packet trains. When the probing rate exceeds the available bandwidth the probe packets will be queued at the tight link, resulting in an increased on-way delay or in dispersion between packets. Conversely, when the probe rate is below

the ABW, there should be no noticeable trend. These tools try to find a sweet spot where the probe rates is just below the ABW. Tools under this category are: Pathload [32] and Pathchirp [73].

An implicit assumption of the first group (PGM) is that one must know the tight link capacity to perform the ABW estimation. For this reason and also for the fact that several studies have reported the good performance of PRM tools (see below), we decided to focus on this group of tools.

Recently, some studies [35], [40], have proposed new techniques to estimate the available bandwidth in modern networks. However, both of them are just proposal and not implemented tools was made available.

#### 4.3.2.1 Evaluation of ABW Estimation Tools

Several works try to evaluate all the available estimation tools, trying to define their characteristics such as accuracy, intrusiveness and response time. The first one, accuracy, regards the estimation given by the tool and how much it is close to the real values. The intrusiveness defines the amount of data and probe packets that are used by the tool. The response time is simply the time needed by the tool to obtain the results.

Shriram et al. [76] compare several tools, including `iperf`. They set-up a fully-controlled testbed in which they induce synthetic cross-traffic in order to test the behavior of the tools in different scenarios. To check the validity of their results, they also perform some real world experiments on third-party testbed networks. They concluded that Pathload and PathChirp are the most accurate tools in their environment.

Batista et al. [13] compare Pathload with abget [11] in a grid network scenario, concluding that Pathload performs better and can be considered the preferable tool to estimate available bandwidth in a grid network.

A complete work regarding available bandwidth tools comparison was made by Guerrero et al. [26]. They evaluate the tools under the previously mentioned metrics, adding the tool reliability. This metric corresponds to the percentage of times that the tool converges on correct results. They observe that the four metrics are influenced by path capacity, network delay, packet loss rate, percentage of cross-traffic and cross-traffic packet size. In the following, we will refer to these five characteristics as a whole as the *network state*. Their main conclusions are:

- Pathload estimation time, is highly affected by the network state;
- Abing, Pathchirp and IGI have an high error rate when the network state presents some anomalies;
- The packet loss rate impacts Spruce and Abing reliability.

The authors of [40] make an analysis of current networks from the assumptions of the previous paper, showing that legacy available bandwidth estimators can encounter problems. In particular, they point out that these legacy tools are developed for a fluid cross-traffic model, meanwhile, modern networks present a bursty model, where we do not have a constant flow, such that two probed packets can happen to not capture cross traffic at all, impacting the estimation accuracy.

In conclusion, from the literature comparisons, Pathload seems to behave better than the others in most situations. Of course not without any problems, like estimation time or runtime errors, but the trade-off seems sustainable. We will thus rely on Pathload in the cloud, aiming at two objectives:

1. test the usability, accuracy and intrusiveness of Pathload in a modern network environment;
2. perform a initial comparison between the two VPN tools.

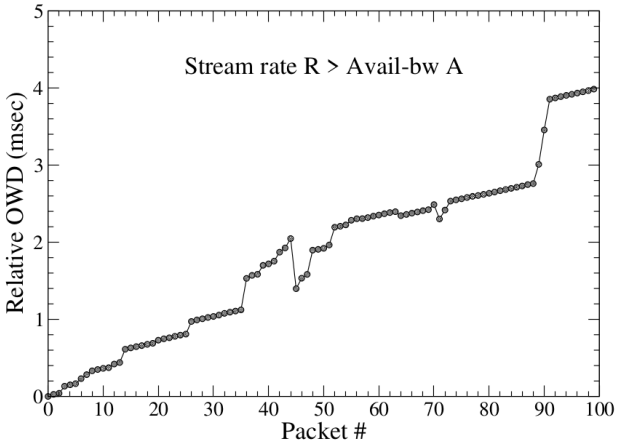


**Figure 4.9:** One Way delay behavior in Pathload [32]: OWD variation for a periodic stream when $R > A$ (increasing trend) (source: [32])

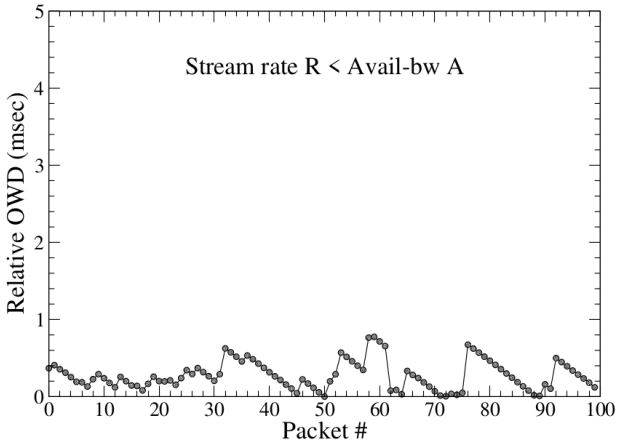


**Figure 4.10:** One way delay behavior in Pathload [32]: OWD variation for a periodic stream when $R < A$ (no trend) (source: [32])

### 4.3.3 Pathload

Pathload implements the self-loading periodic stream (SLoPS) technique [32]. It works in a client/server mode where they are respectively called sender and receiver. The sender sends a stream of K ($K = 100$) packets of size L within an interval T, to generate a certain rate R (where $R = \frac{L}{T}$). Pathload relies on the fact that when the rate R goes beyond the available bandwidth A, the fleet will cause congestion on the tight link increasing the one-way delay (OWD) of the packets (Figure 4.9), on the other side (if $R < A$) no queuing will happen (Figure 4.10).

To define a starting rate, Pathload relies on the ADR (Asymptotic Dispersion Rate): a metric measured by dispersion of long packet trains. Even if for a long time, it was believed as a technique to measure the link capacity, it has been proved in [32] that the ADR lies in between the available bandwidth and the actual link capacity.

At each iteration, the sender dispatches a fleet composed of multiple streams, the receiver labels the receiving fleet with a type: no trend, if the rate is below the available bandwidth; increasing, if we are above ABW; grey, if we are near to the actual available bandwidth value (Figure 4.11). After this classification, if it still didn't converge to a result, it computes the rate of the next fleet, adapting it to the trend: if $R > ABW$ it

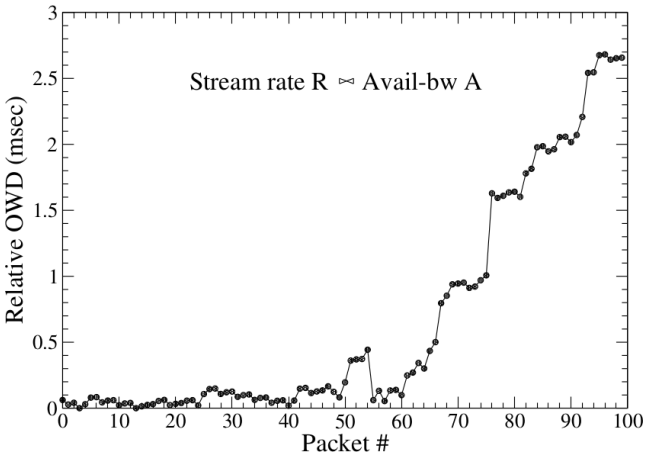


**Figure 4.11:** One way delay behavior in Pathload [32]: OWD variation for a periodic stream when $R \bowtie A$ (grey zone trend) (source: [32])

| | |
|---|---|
| **Europe (EU)** | Frankfurt |
| **United States (US)** | North Virginia |
| **South America (SA)** | Sao Paulo |
| **Asia Pacific (AP)** | Tokyo |

**Table 4.4:** Regions deployed for Pathload tests in Amazon AWS

decreases the rate, if $R < ABW$ it increases the rate. If it is in the grey area, it tries to reduce the gap between the minimum and maximum values of the ABW range found until now, setting the average between the two as the new rate R. The min and max values are respectively the maximum value found below the available bandwidth and the minimum value higher than the available bandwidth.

Pathload goes on with this dichotomous search until it converges on an available bandwidth range. In fact, Pathload provides a range in which the available bandwidth should be located and not a specific value. This is because the ABW is a floating value changing over time.

### 4.3.4 Available Bandwidth in Amazon Web Services

#### 4.3.4.1 Methodology

For the present study, we decided to focus on the Amazon inter-datacenter network. Amazon actually features 14 regions: 4 in the United States, 1 in Canada, 3 in Europe 5 in Asia Pacific and 1 in South America. We want to test the available bandwidth between different regions. To do so we deployed one EC2 instance in 4 different regions, listed in Table 4.4. We selected one region per continent in order to get a global view of the Amazon AWS infrastructure and to compare the VPN tools in a wide geographically-distributed environment.

We performed 80 Pathload tests between each pair of regions. Following the hint from Persico et al. [70], to test the link asymmetry, 40 of these tests was performed in one direction and 40 on the other one (e.g. between the regions EU and US we performed 40 tests from EU to US and 40 tests from US to EU, inverting the position of Pathload's sender and receiver). This gave us 6 paths to test in both ways, for a total of 12 tested paths.

To check the validity of the results, we performed 5 `iperf` tests, of 10 seconds duration, for each direction and for both TCP and UDP settings. For UDP, we specified a target throughput of 1Gbps.

The above methodology was performed three different times, one for each of the technologies to compare: directly through IP, through IPSec and OpenVPN, to compare the eventual tunneling overhead. We performed three times (IP, IPSec and OpenVPN), each of these 80 tests for the 6 paths, for a total of $3 \times 80 \times 6$ Pathload tests and $3 \times 5 \times 6$ iperf tests.

#### 4.3.4.2 Results

From a global view of the results (Figure 4.12) we can directly notice the apparent asymmetry among some regions depending on the direction. The smallest ABW seems to be between Europe and South America, probably due to the infrastructure limitations of the southern region. The other paths float around the same values, between 400 and 600Mbps.

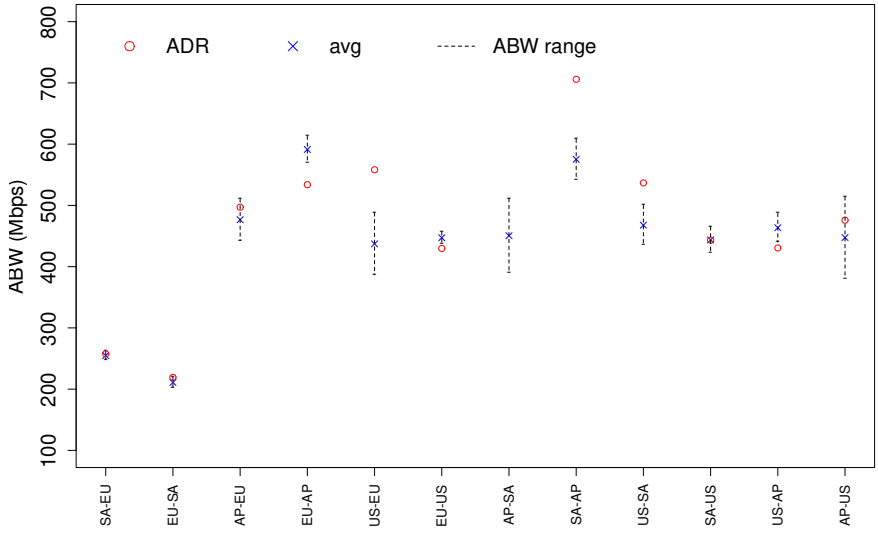


**Figure 4.12:** Global view of the Pathload results in direct IP connection in the 12 Amazon AWS paths. Missing ADR values means that they are above the y-axis limit of 800Mbps.

Moreover, we noticed that the different bandwidths can be related to the load in the different regions. All the tests were performed during French working hours, meaning that Asian and American regions were in the nighttime.

In the following paragraphs we are going to analyze Pathload accuracy and overhead, compare the behavior of IPSec and OpenVPN on the AWS infrastructure and finally, do some considerations about the Amazon network that we noticed during our tests. We are going to report the results that we found more significant, as they are representative of the whole set.

#### 4.3.4.3 Pathload Accuracy

To check the accuracy of the tool, we compared the Pathload results with the values reported by `iperf`. For this evaluation, we considered just the tests with the IP protocol, we treat the VPN tools evaluation in the last paragraph.

We consider the `iperf` UDP throughput as the upper bound of the achievable speed, foreseeing the Pathload range to be around this value. In fact, most of the results show consistent values with this assumption, see Figure 4.13 and Figure 4.14. Yet there are some outliers where we have a low UDP throughput and the

Pathload results are significantly above this value (Figure 4.15). In most cases, these outliers are connected to the South American region, suggesting that the cause of these outliers is the physical infrastructure of the region. We also observe, using traceroute, that several paths were existing between each data centers in some cases, with per (transport level) connexion load balancer. We leave for future work (Deliverable D3.4, the second iteration of this document) a more in depth study of these features.

In all cases, we notice that the Pathload results are always stable and have small variability. This suggests that Pathload should provide accurate results in the majority of the cases.

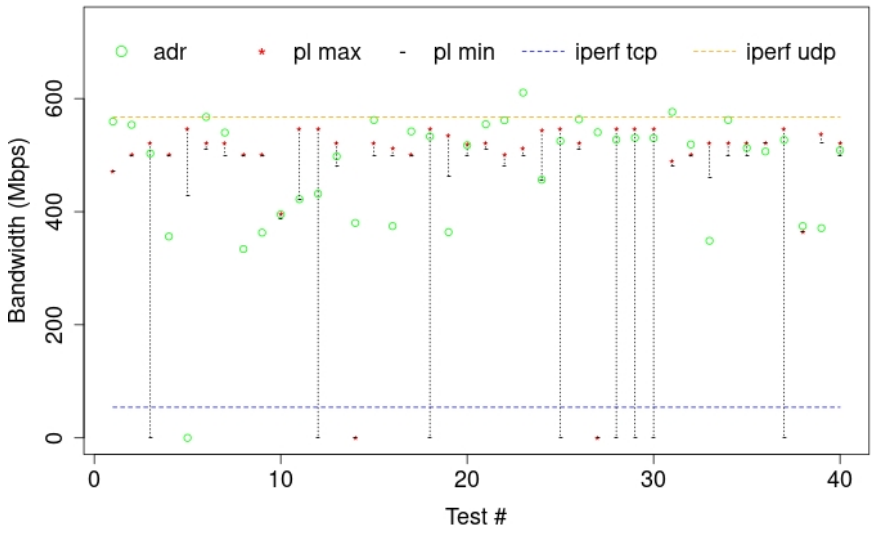


**Figure 4.13:** Pathload available bandwidth estimation in the Amazon AWS inter-datacenter network through IP protocol. The results are compared with the `iperf` results (UDP and TCP): AP-US path, the consistent values are below but still near to the UDP throughput

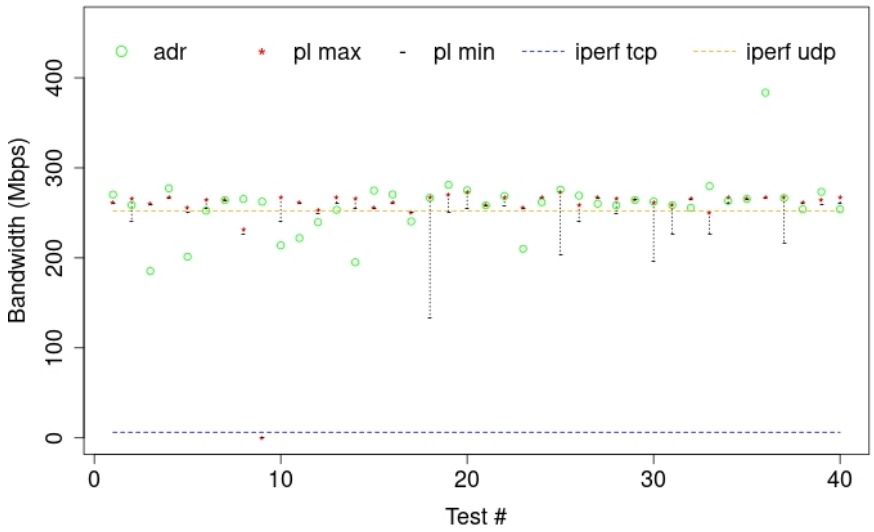


**Figure 4.14:** Pathload available bandwidth estimation in the Amazon AWS inter-datacenter network through IP protocol. The results are compared with the `iperf` results (UDP and TCP): SA-EU path, the consistent values are slightly above the UDP throughput

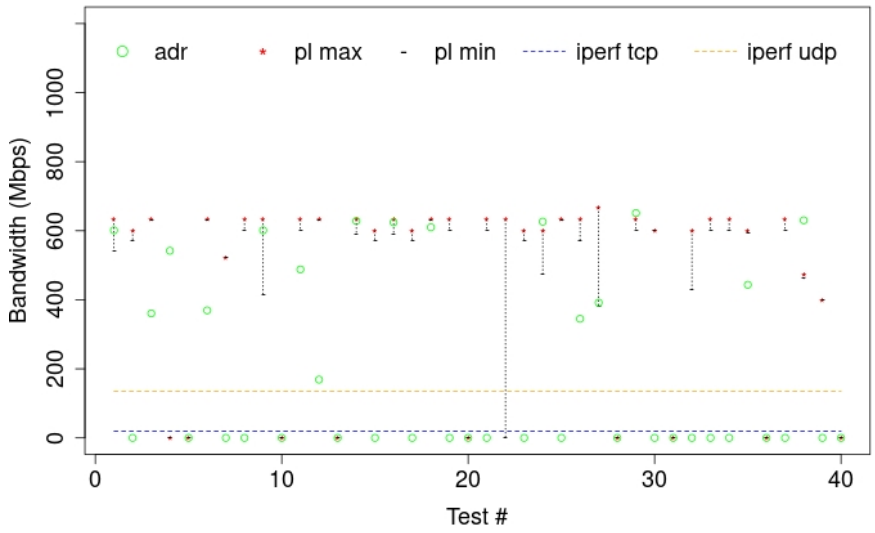


**Figure 4.15:** Pathload available bandwidth estimation in the Amazon AWS inter-datacenter network through IP protocol. The results are compared with the `iperf` results (UDP and TCP): SA-AP path, the outlier result goes beyond the UDP throughput

#### 4.3.4.4 Pathload Intrusiveness

To understand how intrusive Pathload is in our tests, we considered the total amount of data used per test. We found out a directly proportional relation between estimation time and used data (Figure 4.16 and Figure 4.17). However, the same estimation time doesn't always correspond to the same amount of data, as we can see in the figures. This is dependent upon various factors, as explained in [26]. As we are considering different regions with different infrastructural characteristic and we assume high variability in network characteristics, the Pathload behavior is proving consistent with the previous works.

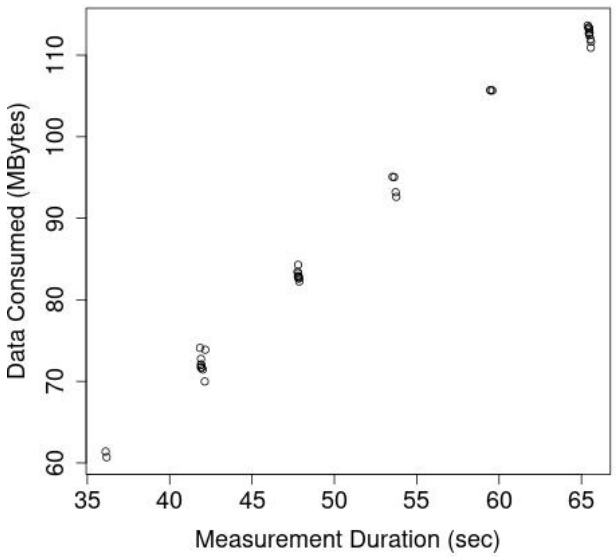


**Figure 4.16:** Estimation Time / Used Data relation in Pathload. Values at 0 are not converged tests, after 60 seconds of run the tool stop with no convergence: EU-SA path

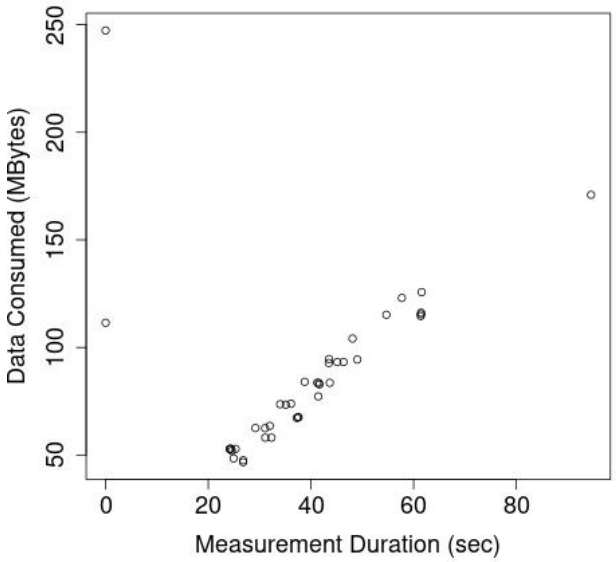


**Figure 4.17:** Estimation Time / Used Data relation in Pathload. Values at 0 are not converged tests, after 60 seconds of run the tool stop with no convergence: AP-US path

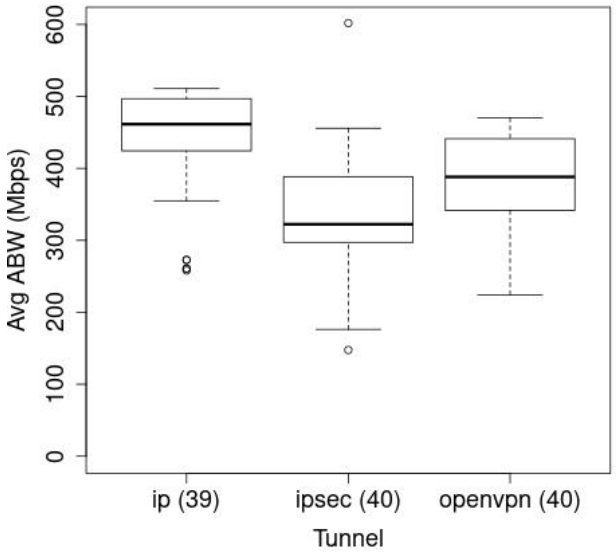


**Figure 4.18:** Average of the available bandwidth estimation compared among the three different packet encapsulation: IP, IPSec, OpenVPN. The results are filtered for converged Pathload tests, the number in the x-axis shows the number of considered tests - US-EU path: some variation between the encapsulations but not significant

In addition, considering that a 10-second `iperf` test that reaches a throughput of 400Mbps uses in average more than 400 MBytes, we can assert that Pathload fulfills our purpose of adopting a less intrusive technique.

### 4.3.5 ABW Measurements within VPN Tunnels

When it comes to comparing the performance of IPSec and OpenVPN we obtained similar results for every path. Indeed, in most cases, e.g. Figure 4.18 and Figure 4.19, we observed that the estimated available bandwidth is comparable between the three encapsulation techniques: the basic IP link and the encrypted IPSec and OpenVPN. There was also a few exceptions, e.g. Figure 4.20, involving the South American data centers that already led to problems during the latency tests.

As a last note, we can observe that the slight difference in terms of latency between the two tools observed during the ping tests did not lead to a significant difference in terms of throughput. It could have been the case if the latency difference was accumulative over all subsequent packets. However, a pipe effect seems to occur, leading to similar results between IPSEc and OpenVPN.

## 4.4 Conclusion

Our aim in this chapter was to evaluate the following aspects of our prototype:

- the stability of the system over time;
- the overhead of the tunneling/encryption tools;
- the practical differences between IPSec and OpenVPN.

Though preliminary, the tests conducted led us to the following clear conclusion: the performance, of the implementations of IPSec and OpenVPN we used are almost identical. The stability is related to the quality of the implementations in the Linux distributions we considered. They have been extensively tested and improved over several years by the users and developers respectively.

As for the overhead, the almost equal score obtained by the two tools is less obvious than it seems at first

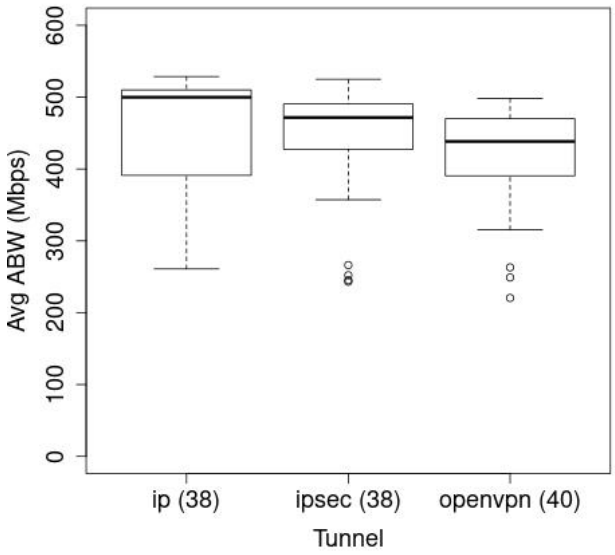


**Figure 4.19:** Average of the available bandwidth estimation compared among the three different packet encapsulation: IP, IPSec, OpenVPN. The results are filtered for converged Pathload tests, the number in the x-axis shows the number of considered tests - AP-US path: the three encapsulations are equivalent

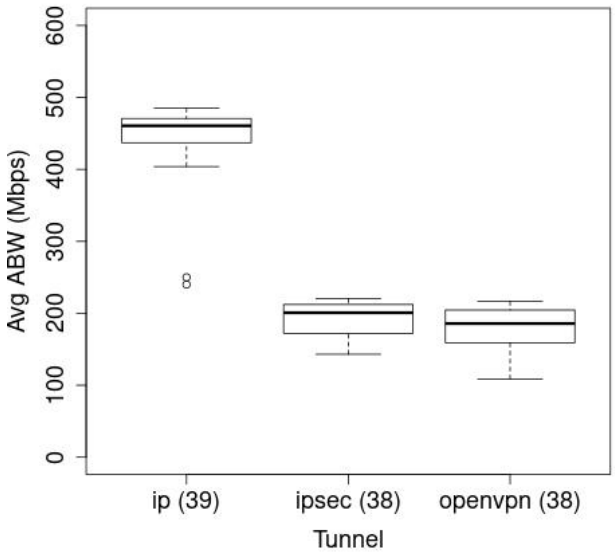


**Figure 4.20:** Average of the available bandwidth estimation compared among the three different packet encapsulation: IP, IPSec, OpenVPN. The results are filtered for converged Pathload tests, the number in the x-axis shows the number of considered tests - US-EU path: this outliers shows a significant difference between the VPNs and the IP link

sight. Indeed, IPSec is deeply integrated in the kernel while OpenVPN remains a user-land application with a heavier encapsulation. As OpenVPN is often considered as easier to configure and less likely to be filtered by any middlebox in the network, we will tend to favor its usage over IPSec in practice. This means that OpenVPN will be the default choice while IPSec will be offered as an alternative.

As our tests were performed in the wild, more precisely in a typical usage scenario for PrEstoCloud, this led us to a series of interesting additional conclusions. First, the stability of the latency and throughput measurements we observed is due for one part to the stability of the tools we use but also to the quality of the paths to and in between the data centers. Furthermore, the inter data centers paths, between the same or between different cloud providers, apparently feature a higher level of stability than a typical path from the edge of the network (an academic lab in our test). While this small scale test would need be confirmed by a wider measurement campaign, they already suggest that inter data centers paths should not constitute a bottleneck while deploying distributed applications over several data centers, as we envisage in PrEstoCloud.

Last but not least, the measurement tests made with available bandwidth tools and especially Pathload raise a number of interesting research issues. Indeed, the observed stability of the paths is the result of the somewhat complex network architecture with multiple paths and connexion level load balancers. We will further investigate this topic and how a relevant bandwidth measurements can be reported to the application in the near future, and hope to report advances in the domain in the second iteration of this deliverable.

# Chapter 5

# Conclusion and Future Work

In this Deliverable, we have laid the foundations of the PrEstoCloud overlay network. After a complete review of the landscape of solutions offered by public cloud providers or in terms of publicly available tools, we have selected a set of basic building blocks to build our network overlay. We next presented the overall design principles we adopted to have a complete solution fulfilling the set of constraints we had, namely:

- interconnect compute resources with one another, irrespectively of their actual physical location (private cloud, public cloud, edge);
- deploy an overlay solution to build the virtual IP network inter-connecting the application components;
- enable a secure communication between the different application components;
- continuously monitor the status of the global overlay components (overlay gateways) and report Quality of Service metrics of interest for the application, e.g., Round Trip Time or available bandwidth;
- genericity: the solution proposed should not be bound to a proprietary platform or framework;
- performance: the solution should permit to achieve an optimal throughput and a low delay.

We opted for a distributed design relying on well-established Virtual Private Network solutions to build a full mesh between all the data centers (up to the edge) where the application needs to be deployed.

We have demonstrated, using the prototype we implemented and tested using some Amazon Web Services and Windows Azure data centers, that the overhead of the solution is minimal in terms both of latency and throughput.

In relation with the monitoring task described above, we have focused on the problem of monitoring the bandwidth available for the application under cost constraints. Using tools that have been deployed before the spreading of virtualization and the cloud era, we have carried out a preliminary study highlighting the stability of the inter data center paths. We further uncovered the complexity of the physical/virtual set-up used by cloud providers within their data centers and especially load balancing over multiple paths. This means that providing meaningful figures for a running application transferring traffic in between data centers will require to adapt the tools initially used to measure a fixed physical path. We will further report on this issue and the integration of a solution that will enable to report to a PrEstoCloud application accurate bandwidth measurement in the next version of this deliverable.

The next steps of our agenda are as follows.

- Integration of the solution within the ProActive framework. This work is already ongoing in the context of the preparation of the first PrEstoCloud prototype that will be available in the second semester of 2018.

- Increase the resiliency of the network overlay. This might entail using a cluster of gateway nodes monitoring each other and ready to take over the role of a failed sibling. Researchers in [78] addressed this issue from the cloud provider viewpoint by decoupling the control path (tunnel establishment) and the data path (containing actual sessions' parameters) and we could inspire from some of their ideas.

- Increase the throughput of the network overlay. The cryptographic operations required for VPN traffic may lead to performance degradation if the set of cryptographic parameters is not properly chosen. Stream ciphers are known for their faster real-time operations, which is why they are usually favored for encrypting communication channels. However, recently, the most ubiquitous stream cipher, RC4, has been heavily under attack and is now considered insecure. The ECRYPT eSTREAM project identified 7 possible strong stream ciphers to take over in the domain. Moreover, mode of operations for block ciphers (e.g. CTR) enable the precomputation of pseudo-random values through a block cipher (e.g. AES) in order to convert them into stream ciphers. We intend to test the cryptographic suites available with StrongSwan and OpenVPN in order to select the most fit for large-volume real-time data transfers, such as found in the context of PrEstoCloud.

- Improve the monitoring of the overlay at runtime. Monitoring is on one side related to resilience as one needs to know when and where a connectivity issue occurs. The data collected on the current state of the application, e.g. the latency or throughput on the overlay links connecting various PrEstoCloud premises (data centers, edge) might also be used to take informed decisions concerning the adaptivity of resources within PrEstoCloud.

- Scalability of the VPN gateway. Indeed, the scalability of the service might also be an issue if a host of edge nodes need to be connected to the overlay. We will have to assess the resource consumption (RAM, CPU) consumed by the different solutions when the number of connections increases. A related issue is the stability of the VPN client in the edge node [1]. The exact solution to use might depend on the application requirements in terms of rate.

# Bibliography

[1] A. Alshalan, S. Pisharody, and D. Huang, "A survey of mobile vpn technologies", *IEEE Communications Surveys & Tutorials*, vol. 18, no. 2, pp. 1177–1196, 2016.

[2] A. Alshamsi and T. Saito, "A technical comparison of ipsec and ssl", in *Advanced Information Networking and Applications, 2005. AINA 2005. 19th International Conference on*, IEEE, vol. 2, 2005, pp. 395–398.

[3] Amazon, *AWS CLI*, https://www.amazon.com/cli/, [Online; accessed 30-August-2017].

[4] ——, *Azure CLI*, https://docs.microsoft.com/en-us/cli/azure/install-azure-cli, [Online; accessed 30-August-2017].

[5] ——, *Dedicated Hosts*, https://aws.amazon.com/ec2/dedicated-hosts/?nc1=h_ls, [Online; accessed 30-August-2017].

[6] ——, *Multi VPC connectivity*, https://aws.amazon.com/answers/networking/aws-multiple-region-multi-vpc-connectivity, [Online; accessed 30-August-2017].

[7] ——, *Placement Groups*, http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/placement-groups.html, [Online; accessed 30-August-2017].

[8] ——, *Using regions and availability zones*, http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-regions-availability-zones.html, [Online; accessed 30-August-2017].

[9] ——, *Virtual Private Cloud*, http://docs.aws.amazon.com/AmazonVPC/latest/UserGuide/VPC_Introduction.html, [Online; accessed 30-August-2017].

[10] *Ansible*, https://www.ansible.com/, [Online; accessed 30-August-2017].

[11] D. Antoniades, M. Athanatos, A. Papadogiannakis, E. P. Markatos, and C. Dovrolis, "Available bandwidth measurement as simple as running wget", in *Proc. of Passive and Active Measurement Conference (PAM 2006)*, 2006, pp. 61–70.

[12] *Bash*, https://www.gnu.org/software/bash, [Online; accessed 30-August-2017].

[13] D. M. Batista, L. J. Chaves, N. L. da Fonseca, and A. Ziviani, "Performance analysis of available bandwidth estimation tools for grid networks", *The Journal of Supercomputing*, vol. 53, no. 1, pp. 103–121, 2010.

[14] T. Berger, "Analysis of current vpn technologies", in *Availability, Reliability and Security, 2006. ARES 2006. The First International Conference on*, IEEE, 2006, 8–pp.

[15] D. Booth, H. Haas, F. McCabe, E. Newcomer, M. Champion, C. Ferris, and D. Orchard, "Web services architecture", W3C, W3C Working Group Note, 2004, https://www.w3.org/TR/ws-arch/.

[16] R. Buyya, R. Ranjan, and R. N. Calheiros, "Intercloud: Utility-oriented federation of cloud computing environments for scaling of application services", in *International Conference on Algorithms and Architectures for Parallel Processing*, Springer, 2010, pp. 13–31.

[17] Cisco, *Data Center Overlay Technologies*, https://www.cisco.com/c/en/us/products/collateral/switches/nexus-9000-series-switches/white-paper-c11-730116.html, [Online; accessed 30-August-2017], 2013.

[18] ——, *Encapsulation Techniques: Generic Network Virtualization Encapsulation, VXLAN Generic Protocol Extension, and Network Service Header*, https://www.cisco.com/c/en/us/solutions/collateral/data-center-virtualization/application-centric-infrastructure/white-paper-c11-733127.pdf, [Online; accessed 30-August-2017], 2014.

[19] CloudStack, *Cloud Stack Networking*, http://docs.cloudstack.apache.org/en/latest/#advanced-networking-guides, [Online; accessed 30-August-2017].

[20] B. Davie, *Geneve, VXLAN, and Network Virtualization Encapsulations*, https://octo.vmware.com/geneve-vxlan-network-virtualization-encapsulations/, [Online; accessed 30-August-2017], 2014.

[21] ——, *VXLAN, STT, and Looking Beyond the Wire*, https://octo.vmware.com/vxlan-stt-and-looking-beyond-the-wire-format/, [Online; accessed 30-August-2017], 2013.

[22] M. Dayananda and A. Kumar, "Architecture for inter-cloud services using ipsec vpn", in *Advanced Computing & Communication Technologies (ACCT), 2012 Second International Conference on*, IEEE, 2012, pp. 463–467.

[23] C. Dovrolis, P. Ramanathan, and D. Moore, "Packet-dispersion techniques and a capacity-estimation methodology", *IEEE/ACM Transactions On Networking*, vol. 12, no. 6, pp. 963–977, 2004.

[24] P. Garg and Y.-S. Wang, *NVGRE: Network Virtualization Using Generic Routing Encapsulation*, RFC 7637, Sep. 2015. DOI: 10.17487/RFC7637. [Online]. Available: https://rfc-editor.org/rfc/rfc7637.txt.

[25] N. Grozev and R. Buyya, "Inter-cloud architectures and application brokering: Taxonomy and survey", *Software: Practice and Experience*, vol. 44, no. 3, pp. 369–390, 2014.

[26] C. D. Guerrero and M. A. Labrador, "On the applicability of available bandwidth estimation techniques and tools", *Computer Communications*, vol. 33, no. 1, pp. 11–22, 2010.

[27] A. Hafsaoui, A. Dandoush, G. Urvoy-Keller, M. Siekkinen, and D. Collange, "A fine-grained response time analysis technique in heterogeneous environments", *Computer Networks*, vol. 130, pp. 16–33, 2018.

[28] D. Harkins and D. Carrel, "The internet key exchange (ike)", RFC Editor, RFC 2409, 1998, http://www.rfc-editor.org/rfc/rfc2409.txt. [Online]. Available: http://www.rfc-editor.org/rfc/rfc2409.txt.

[29] N. Hu and P. Steenkiste, "Evaluation and characterization of available bandwidth probing techniques", *IEEE journal on Selected Areas in Communications*, vol. 21, no. 6, pp. 879–894, 2003.

[30] IEEE, "Standard p1363 – standard specifications for public-key cryptography", Tech. Rep., 2009.

[31] *iperf*, https://www.iperf.fr/, [Online; accessed 30-August-2017].

[32] M. Jain and C. Dovrolis, "End-to-end available bandwidth: Measurement methodology, dynamics, and relation with tcp throughput", *ACM SIGCOMM Computer Communication Review*, vol. 32, no. 4, pp. 295–308, 2002.

[33] R. Kawashima and H. Matsuo, "Implementation and performance analysis of stt tunneling using vnic offloading framework (cvsw)", in *Cloud Computing Technology and Science (CloudCom), 2014 IEEE 6th International Conference on*, IEEE, 2014, pp. 929–934.

[34] T. Koponen, K. Amidon, P. Balland, M. Casado, A. Chanda, B. Fulton, I. Ganichev, J. Gross, P. Ingram, E. J. Jackson, *et al.*, "Network virtualization in multi-tenant datacenters.", in *NSDI*, vol. 14, 2014, pp. 203–216.

[35] K. LaCurts, S. Deng, A. Goyal, and H. Balakrishnan, "Choreo: Network-aware task placement for cloud applications", in *Proceedings of the 2013 conference on Internet measurement conference*, ACM, 2013, pp. 191–204.

[36] J. B. R. Lawas, A. C. Vivero, and A. Sharma, "Network performance evaluation of vpn protocols (sstp and ikev2)", in *Wireless and Optical Communications Networks (WOCN), 2016 Thirteenth International Conference on*, IEEE, 2016, pp. 1–5.

[37] L. Leong, D. Toombs, B. Gill, G. Petri, and T. Haynes, "Magic quadrant for cloud infrastructure as a service", *Gartner*, 2014.

[38] A. Levin, K. Barabash, Y. Ben-Itzhak, S. Guenender, and L. Schour, "Networking architecture for seamless cloud interoperability", in *Cloud Computing (CLOUD), 2015 IEEE 8th International Conference on*, IEEE, 2015, pp. 1021–1024.

[39] A. Li, X. Yang, S. Kandula, and M. Zhang, "Cloudcmp: Comparing public cloud providers", in *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, ACM, 2010, pp. 1–14.

[40] M. Li, Y.-L. Wu, and C.-R. Chang, "Available bandwidth estimation for the network paths with multiple tight links and bursty traffic", *Journal of Network and Computer Applications*, vol. 36, no. 1, pp. 353–367, 2013.

[41] *LibreSwan*, https://libreswan.org/, [Online; accessed 30-August-2017].

[42] M. Mahalingam, D. Dutt, K. Duda, P. Agarwal, L. Kreeger, T. Sridhar, M. Bursell, and C. Wright, *Virtual eXtensible Local Area Network (VXLAN): A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks*, RFC 7348, Aug. 2014. DOI: 10.17487/RFC7348. [Online]. Available: https://rfc-editor.org/rfc/rfc7348.txt.

[43] P. Massonet, S. Dupont, A. Michot, A. Levin, and M. Villari, "An architecture for securing federated cloud networks with service function chaining", in *Computers and Communication (ISCC), 2016 IEEE Symposium on*, IEEE, 2016, pp. 38–43.

[44] ——, "Enforcement of global security policies in federated cloud networks with virtual network functions", in *Network Computing and Applications (NCA), 2016 IEEE 15th International Symposium on*, IEEE, 2016, pp. 81–84.

[45] Microsoft, *Site to Site connection*, https://docs.microsoft.com/en-us/azure/vpn-gateway/vpn-gateway-howto-site-to-site-resource-manager-portal, [Online; accessed 30-August-2017].

[46] ——, *Availability in Azure*, https://docs.microsoft.com/en-us/azure/virtual-machines/windows/manage-availability, [Online; accessed 30-August-2017].

[47] ——, *Direct Connect*, https://aws.amazon.com/directconnect, [Online; accessed 30-August-2017].

[48] ——, *Express Route*, https://docs.microsoft.com/en-us/azure/expressroute/expressroute-introduction, [Online; accessed 30-August-2017].

[49] ——, *Network and subnetworks*, ttps://docs.microsoft.com/en-us/azure/virtual-machines/windows/network-overview#virtual-network-and-subnets, [Online; accessed 30-August-2017].

[50] ——, *Placement Groups*, https://docs.microsoft.com/en-us/azure/virtual-machine-scale-sets/virtual-machine-scale-sets-placement-groups, [Online; accessed 30-August-2017].

[51] ——, *Point to Point connection*, https://docs.microsoft.com/en-us/azure/vpn-gateway/vpn-gateway-howto-point-to-site-resource-manager-portal, [Online; accessed 30-August-2017].

[52] ——, *RDedicated Connection*, http://docs.aws.amazon.com/directconnect/latest/UserGuide/getstarted.html#DedicatedConnection, [Online; accessed 30-August-2017].

[53] ——, *Regions and Availability*, https://docs.microsoft.com/en-us/azure/virtual-machines/windows/regions-and-availability, [Online; accessed 30-August-2017].

[54] ——, *Scale Sets*, https://docs.microsoft.com/en-us/azure/virtual-machine-scale-sets/virtual-machine-scale-sets-overview, [Online; accessed 30-August-2017].

[55] ——, *Source Destination Check*, http://docs.aws.amazon.com/AmazonVPC/latest/UserGuide/VPC_NAT_Instance.html#EIP_Disable_SrcDestCheck, [Online; accessed 30-August-2017].

[56] ——, *Source Destination Check*, https://docs.microsoft.com/en-us/azure/virtual-network/virtual-networks-udr-overview, [Online; accessed 30-August-2017].

[57] ——, *Virtual Networks*, https://docs.microsoft.com/en-us/azure/virtual-machines/windows/network-overview, [Online; accessed 30-August-2017].

[58] ——, *VNet to VNet connection*, https://docs.microsoft.com/en-us/azure/vpn-gateway/virtual-networks-configure-vnet-to-vnet-connection, [Online; accessed 30-August-2017].

[59] R. Moreno-Vozmediano, R. S. Montero, E. Huedo, and I. M. Llorente, "Implementation and provisioning of federated networks in hybrid clouds", *Journal of Grid Computing*, pp. 1–20,

[60] S. Narayan, C. J. Williams, D. K. Hart, and M. W. Qualtrough, "Network performance comparison of vpn protocols on wired and wireless networks", in *Computer Communication and Informatics (ICCCI), 2015 International Conference on*, IEEE, 2015, pp. 1–7.

[61] J. Navratil and R Les Cottrell, "Abwe: A practical approach to available bandwidth estimation", in *in Passive and Active Measurement (PAM) Workshop 2003 Proceedings, La Jolla*, Citeseer, 2003.

[62] *nuttcp*, https://www.nuttcp.net/, [Online; accessed 30-August-2017].

[63] *OpenConnect*, https://www.infradead.org.org/openconnect/, [Online; accessed 30-August-2017].

[64] OpenStack, *Neutron*, https://wiki.openstack.org/wiki/Neutron, [Online; accessed 30-August-2017].

[65] *OpenVPN*, https://openvpn.net/, [Online; accessed 30-August-2017].

[66] V. Persico, A. Botta, A. Montieri, and A. Pescape, "A first look at public-cloud inter-datacenter network performance", in *Global Communications Conference (GLOBECOM), 2016 IEEE*, IEEE, 2016, pp. 1–7.

[67] V. Persico, A. Montieri, and A. Pescapé, "Cloudsurf: A platform for monitoring public-cloud networks", in *Research and Technologies for Society and Industry Leveraging a better tomorrow (RTSI), 2016 IEEE 2nd International Forum on*, IEEE, 2016, pp. 1–6.

[68] V. Persico, P. Marchetta, A. Botta, and A. Pescapé, "Measuring network throughput in the cloud: The case of amazon ec2", *Computer Networks*, vol. 93, pp. 408–422, 2015.

[69] ——, "On network throughput variability in microsoft azure cloud", in *Global Communications Conference (GLOBECOM), 2015 IEEE*, IEEE, 2015, pp. 1–6.

[70] V. Persico, A. Botta, P. Marchetta, A. Montieri, and A. Pescapé, "On the performance of the wide-area networks interconnecting public-cloud datacenters around the globe", *Computer Networks*, vol. 112, pp. 67–83, 2017.

[71] *Python*, https://www.python.org/, [Online; accessed 30-August-2017].

[72] Y. Rekhter, B. Moskowitz, D. Karrenberg, G. J. de Groot, and E. Lear, "Address Allocation for Private Internets", Internet Engineering Task Force, RFC 1918, Feb. 1996.

[73] V. J. Ribeiro, R. H. Riedi, R. G. Baraniuk, J. Navratil, and L. Cottrell, "Pathchirp: Efficient available bandwidth estimation for network paths", in *Passive and active measurement workshop*, 2003.

[74] B. Rochwerger, D. Breitgand, E. Levy, A. Galis, K. Nagin, I. M. Llorente, R. Montero, Y. Wolfsthal, E. Elmroth, J. Caceres, *et al.*, "The reservoir model and architecture for open federated cloud computing", *IBM Journal of Research and Development*, vol. 53, no. 4, pp. 4–1, 2009.

[75] P. Roman, *Step-by-Step: Connect your AWS and Azure environments with a VPN tunnel*, https://blogs.technet.microsoft.com/canitpro/2016/01/11/step-by-step-connect-your-aws-and-azure-environments-with-a-vpn-tunnel/, [Online; accessed 30-August-2017], 2016.

[76] A. Shriram, M. Murray, Y. Hyun, N. Brownlee, A. Broido, M. Fomenkov, and K. C. Claffy, "Comparison of public end-to-end bandwidth estimation tools on high-speed links.", in *PAM*, Springer, vol. 3431, 2005, pp. 306–320.

[77] *SoftEther*, https://www.softether.org/, [Online; accessed 30-August-2017].

[78] J. Son, Y. Xiong, K. Tan, P. Wang, Z. Gan, and S. Moon, "Protego: Cloud-scale multitenant ipsec gateway", in *2017 USENIX Annual Technical Conference (USENIX ATC 17)*, Santa Clara, CA: USENIX Association, 2017, pp. 473–485, ISBN: 978-1-931971-38-6. [Online]. Available: https://www.usenix.org/conference/atc17/technical-sessions/presentation/son.

[79] J. Strauss, D. Katabi, and F. Kaashoek, "A measurement study of available bandwidth estimation tools", in *Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement*, ACM, 2003, pp. 39–44.

[80] *StrongSwan*, https://strongswan.org/, [Online; accessed 30-August-2017].

[81] VMWare, *VMWare Direct Connect*, http://www.vmware.com/cloud-services/networking/direct-connect.html, [Online; accessed 30-August-2017].

[82] ——, *Hybrid DMX*, http://www.vmware.com/cloud-services/infrastructure/vcloud-air-hybrid-dmz.html, [Online; accessed 30-August-2017].

[83] ——, *Virtual Private Cloud*, http://www.vmware.com/cloud-services/infrastructure.html, [Online; accessed 30-August-2017].

[84] ——, *VMWare Advanced Networking*, http://www.vmware.com/cloud-services/infrastructure/vcloud-air-advanced-networking-services.html, [Online; accessed 30-August-2017].