Grant Agreement No.: 610764
Instrument: Collaborative Project
Call Identifier: FP7-ICT-2013-10

PANACEA

Proactive Autonomic Management of Cloud Resources

D2.3: Autonomic Communication Overlay

Version 1.0

Work package	WP2
Task	Task 2.2
Due date	30/03/2015
Submission date	30/03/2015
Deliverable lead	Olivier Brun (CNRS)
Version	1.0
Authors	Urtzi Ayesta , Olivier Brun, Hassan Hassan, Balakrishna Prabhu
Reviewers	Michel Diaz, Dimiter Avresky, Eduardo Huedo

Abstract	This document describes the proposed approach for shielding distributed services deployed in multiple clouds from path outages and performance degradations of the Internet. This approach relies on a self-healing, self-optimizing and highly scalable routing overlay that is able to monitor the quality of Internet paths between its nodes and to adapt its routing scheme according to application-specific metrics and to what is observed from the underlying network.
Keywords	Overlay network, inter-cloud communications, self-recovery, self-optimization, network metrology, optimal path discovery

Document Revision History

Version	Date	Description of change	List of contributor(s)
v1.2	15.03. 2015	Draft document for review	U. Ayesta, O Brun, H. Hassan, B. Prabhu (Editors)
v1.3	26.03.2015	Second draft with changes	D. Avresky, M. Diaz, E. Huedo, O. Brun

Disclaimer

Project co-funded by the European Commission in the 7th Framework Programme (2007-2013)		
Nature of the deliverable:	R	
Dissemination Level		
PU	Public	**X**
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to bodies determined by the PANACEA project	
CO	Confidential to PANACEA project and Commission Services	

EXECUTIVE SUMMARY

The primary goal of PANACEA is to significantly increase the availability and performance of services deployed in the cloud. This will be achieved by designing and developing innovative solutions for a proactive and autonomic management scheme of cloud services. PANACEA-enabled services will be able to manage themselves without direct human intervention, recovering from many inevitable anomalies and autonomously optimizing their performance in changing conditions.

In this document, we describe the proposed approach for shielding distributed services deployed in multiple clouds from path outages and performance degradations of the Internet. This approach relies on a self-healing, self-optimizing and highly scalable routing overlay formed by software routers deployed in the multiple clouds, and possibly in other locations. The routing overlay is able to monitor the quality of Internet paths between its nodes. It can therefore adapt its routing scheme according to application-specific metrics and to what is observed from the underlying network, so that it becomes possible to quickly detour packets along an alternate route when the primary route becomes unavailable or suboptimal.

We describe in depth the design objectives, the architecture and the implementation of the routing overlay. We present as well the methods used for assessing the quality of overlay links according to various metrics, and discuss several approaches for discovering optimal routes in large overlay networks with a minimum probing effort. Finally, we also give some preliminary experimental results obtained either on a network emulation platform or with real-world experiments in the Internet.

Table of Contents

LIST OF FIGURES

ABBREVIATIONS

ACM	**Autonomic Cloud Manager**
API	**Application Programming Interface**
BGP	**Border Gateway Protocol**
CDN	**Content Delivery Network**
CPN	**Cognitive Packet Network**
DDoS	**Distributed Denial of Service**
DOVE	**Distributed Overlay Virtual Ethernet**
EXP3	**Exponential-weight algorithm for Exploration and Exploitation**
GUI	**Graphical User Interface**
IaaS	**Infrastructure as a Service**
IP	**Internet Protocol**
IT	**Information Technology**
ML	**Machine Learning**
NFV	**Network Functions Virtualization**
O&M	**Operations and Maintenance**
OPD	**Optimal Path Discovery**
OS	**Operating System**
OSPF	**Open Shortest Path First**
PaaS	**Platform as a Service**
QoS	**Quality of Service**
RA	**Reception Agent**
REST	**Representational State Transfer**
RON	**Resilient Overlay Network**
SLA	**Service Level Agreement**
TA	**Transmission Agent**
TCP	**Transmission Control Protocol**
UDP	**User datagram Protocol**
VLAN	**Virtual Local Area Network**
VM	**Virtual Machine**
VXLAN	**Virtual Extensible Local Area Network**
WAN	**Wide Area Network**
WP	**Work Package**

1 INTRODUCTION

Even small degradations in the performance and availability of modern services deployed over the Internet can have a considerable business impact. These services usually require continuous operation over time, always maintaining the response time below an acceptable threshold, in order to avoid lost revenue and reduced productivity. In the case of e-commerce sites for instance, degradations of response time can have a dramatic impact on service adoption and on the number of site visitors that are converted into paying customers. Similarly, unplanned outages usually represent substantial loss in revenue and can cause significant damage to brand reputation.

In view of that, and in spite of the increased adoption of clouds, many companies are still reluctant to deploy sensitive business applications in the cloud. The main reason for this reluctance is that, currently, cloud-computing platforms do not provide the availability and performance levels required by these applications.

The primary goal of PANACEA is to significantly increase the availability and performance of services deployed in the cloud. We will achieve this by designing and developing innovative solutions for a proactive and autonomic management scheme of cloud services. We address both the failures and performance degradations that can occur at the node level and at the network level.

Regarding the node level, our approach is to enable cloud services to manage themselves without direct human intervention [2]. Thanks to the introduction of new mechanisms in the cloud manager, PANACEA-enabled services will be both *self-aware* and *self-configurable*, in that they will know themselves as well as the state of their environment and will have the effector mechanisms to trigger a reconfiguration when an anomaly is detected. PANACEA-enabled services will be also *self-healing* in that, by continuously monitoring system and application metrics [3,4], and using advanced machine learning algorithms [6,7], they will be able to predict anomalies and to proactively reconfigure themselves. Finally, PANACEA-enabled services will be *self-optimizing* thank to an autonomic job allocation mechanism providing the ability to dispatch incoming jobs to the best available resources in order to maintain and improve the response time of jobs.

Regarding the network level, our approach is to design a communication system shielding inter-cloud communications from path outages and performance degradations of the Internet. This communication system consists in a *self-healing*, *self-optimizing* and *highly scalable* routing overlay that accepts customized routing policies formulated by distributed applications according to their own needs. The software agents composing the overlay monitor the quality of Internet paths (latency, bandwidth, loss rate) between themselves and can quickly detect path failures and QoS degradations. They are able to detour packets along an alternate path when the given primary path becomes unavailable, and to discover the optimal paths between themselves with a minimum monitoring effort.

In this document, we describe the design objectives, the architecture and the implementation of the routing overlay. We cover in depth the methods used for

assessing the quality of overlay links according to various metrics (delay, loss rate, available bandwidth). One of our design goals is to build a routing overlay that can be widely deployed over a sizable population of routers. We discuss the implications of this requirement and present methods for discovering optimal routes in large overlay networks with a minimum probing effort. Finally, we also present some preliminary experimental results obtained either on a network emulation platform or with real-world experiments in the Internet.

The rest of this document is organized as follows. We first provide in Section 2 experimental evidences of the limitations of the Internet routing infrastructure, which show that it is difficult to achieve desired levels of availability and performance natively on the Internet. In Section 3, we present the design objectives of our routing overlay and discuss the similarities and differences with existing solutions. Section 4 is devoted to the description of the architecture and components of our system. In Section 5, we describe the technical mechanisms used for forwarding a packet from its source to its destination. Section 6 is devoted to the metering of the quality of the overlay links, whereas Section 7 presents the methods used for discovering optimal routes in the overlay without using a non scalable all-pair probing approach. We describe the test and validation platforms in Section 8, before presenting some experimental results in Section 9. Finally we conclude in Section 10 with a brief summary and a description of future work.

2 LIMITATIONS OF THE INTERNET ROUTING INFRASTRUCTURE

It is now broadly admitted that the performances of individual Internet flows could be improved by selecting alternate paths to those provided by IP routing protocols [46,51]. There are several reasons for the suboptimal performance of Internet routing. One of these reasons is that Internet routing protocols are not able to perform load-sensitive routing to dynamically shed load away from congested paths to less-loaded ones. Another reason is related to the way the inter-domain routing protocol BGP (Border Gateway Protocol) is designed. BGP performs a distributed computation to determine the "best" path between a source and a destination, taking into account routing policies expressed by ISPs using a variety of control knobs. Individual ISPs may, in making their routing decisions, choose to optimize a wide variety of properties. Because each ISP chooses its own objectives, different ISPs may choose to optimize different quantities, leading to an overall path that captures no simple notion of "best", and rarely if ever is best for the user.

Besides, the routing scalability of the Internet comes at the expense of reduced fault-tolerance of end-to-end communications between Internet hosts. Indeed, BGP hides many topological details in the interest of scalability and policy enforcement, and damps routing updates when potential problems arise to prevent large-scale oscillations. As a consequence, BGP reacts and recovers slowly from link/node failures, causing path outages that can last for several tens of minutes [24,36,40].

Studies done over the last fifteen years have provided numerous evidences that Internet routing suffers from many flaws in terms of availability and performance. We describe below the result of an experiment showing that the situation has not significantly evolved since these studies were performed. To perform this experiment, we selected 20 nodes of the NLNog ring[1] as depicted in **Figure 1**. We measured the latency and loss rates between all pairs of nodes every two minutes for a period of one week using the ICMP-based ping utility. When five consecutive packets are lost, we consider that the source and destination are disconnected.

The main observations from this Internet-scale experiment were the following:

- There was a path outage at least once in the week for 65% of origin/destination pairs, and 21% of these path outages lasted more than 4 minutes (and more than 14 minutes for 11% of them).
- The latency of the IP routes exhibits strong and unpredictable variations (see, e.g., Figure 2) that can be as high as 500%. The IP route is not the minimum latency path in 38% of the cases. There is always at least one origin/destination pair whose latency can be reduced by more than 76% by selecting an alternate path to the IP route.
- Similarly, more than 11% of the IP routes have a loss rate greater than 1%.

[1] The NLNog ring is a network of 293 nodes distributed over 46 countries (see https://ring.nlnog.net).

By selecting alternate paths to those proposed by IP routing protocols, it would have been possible to have no loss at all.

Figure 1 Geographical location of the 20 nodes selected in the NLNog ring.

These results show that path outages are routine events in the Internet, and that the paths selected by IP routing protocols are strongly suboptimal. This Internet-scale experiment thus confirms the observations already made in previous studies. Besides, the data collected during the experiment are used in Section 9 for validation purposes.

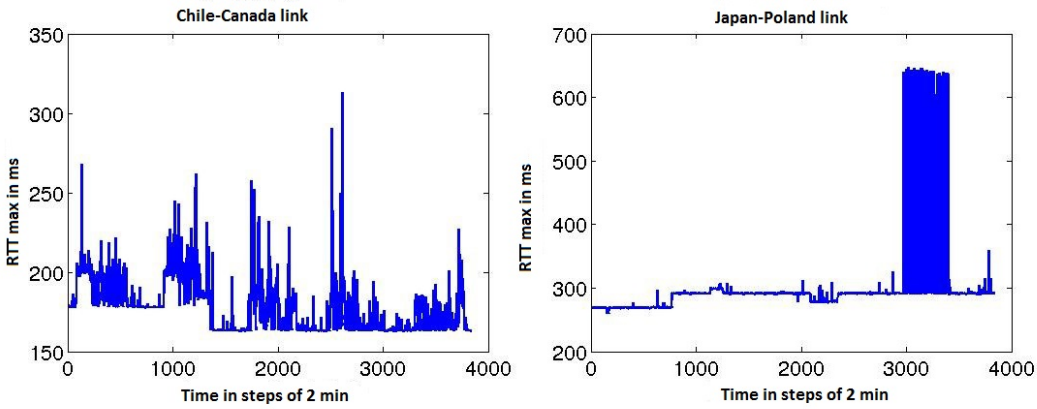

Figure 2: RTT of the IP routes Chile-Canada and Japan-Poland with respect to time.

3 A SELF-HEALING AND SELF-OPTIMIZING ROUTING OVERLAY

Unfortunately, the routing infrastructure of the Internet has become resistant to major changes, preventing even necessary changes to take place and hindering the development of new network functionalities (e.g., multicast, QoS). Current routing protocols may work reasonably well when only "best effort" delivery is required. The requirements for PANACEA-enabled distributed services are typically far more stringent, demanding greater performance and availability of end-to-end routes than these protocols can deliver.

The solution that we propose consists in a self-healing, self-optimizing and highly scalable routing overlay that accepts customized routing policies formulated by distributed applications according to their own needs. The overlay network is formed by software routers deployed over the Internet, and is able to monitor the quality of Internet paths (latency, bandwidth, loss rate) between overlay nodes and detour packets along an alternate path when the given primary path becomes unavailable or suffers from congestion. This approach is expected to provide applications with the desired flexibility and control of the routing infrastructure, allowing adapting the routes to application-specific requirements and to the observed quality of the underlying Internet paths.

We first provide background information on routing overlays in Section 3.1. We then describe the functional requirements of our communication system in Section 3.2. Finally, we discuss similarities and differences with existing solutions in Section 3.3.

3.1 Overlay Networks

Instead of relying on IP routing protocols that are slow in reacting and recovering from link failures or congestion, an alternative approach is to use a routing overlay [19,26,48,58]. The overlay nodes correspond to end hosts, which are deployed over the Internet (see **Figure 3**). The nodes of the overlay cooperate with each other to forward data on behalf of any pair of communicating nodes.

A routing overlay therefore enables controlling the path of data through the network. In a routing overlay, the endpoints of the information exchange are unchanged from what they would have been in the absence of the overlay, but the route through the network that the packets traverse may be different.

Routing overlays can be used to quickly recover from path outages. Indeed, the nodes of the overlay usually reside in multiple autonomous systems (AS). Because each AS is independently administrated and configured, they generally fail independently of each other. As a result, if the underlying topology has physical path redundancy, it is often possible for the routing overlay to find paths between its nodes, even if Internet routing protocols cannot. As illustrated in **Figure 4** the basic idea is to leverage the inherent redundancy of the underlying topology to detour packets along an alternate path, when the given primary path between two

software routers becomes unavailable.

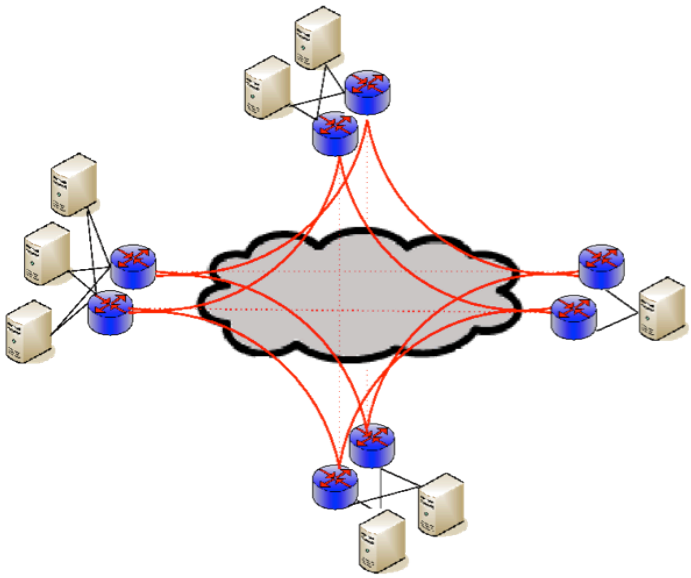

Figure 3: Structure of an overlay network.

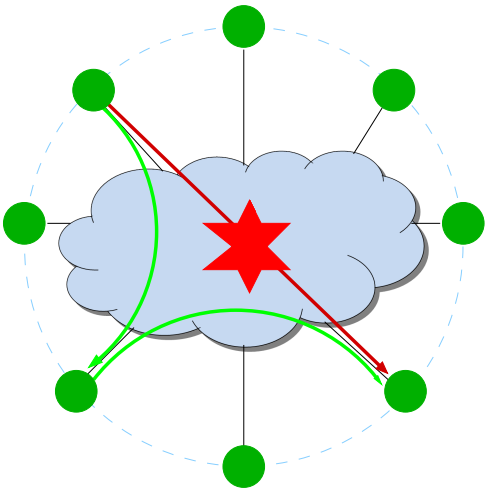

Figure 4: An alternate path is used when the primary one is faulty.

In addition, routing overlays can also be used to improve the quality of service of data flows. Indeed, a routing overlay overrides the routes determined by Internet protocols, routing traffic based on metrics directly related to application performances. In this way, a routing overlay enables to adapt routes to application-specific requirements and to the observed quality of the underlying Internet paths.

3.2 Goals of our overlay network

Our goal is to design a routing overlay shielding distributed applications from path outages and performance degradations of the Internet. The communication should comply with the following functional requirements:

- The overlay has to be *self-healing* as it has to quickly detect and recover from path outages and performance failures,
- The overlay has to be *self-optimizing* as it has to adapt routes within the overlay network so as to optimize throughput, loss, and latency,
- The proposed solution has to be *highly scalable*, meaning that it should be possible to discover the optimal paths in a large overlay network with a *limited monitoring effort,*
- *Service-specific routing metrics* have to be available to decide how to forward packets in the overlay, so as to cope with the different communication requirements of different distributed applications,
- The proposed solution has to be *independent of applications* (it is not a software communication library that programs link against) and of operating systems (no kernel modification).

3.3 Similarities and differences with respect to existing solutions

Researchers have successfully used overlay networks to solve problems in various areas. To name but a few of the applications, overlays have been used for self-organization in peer-to-peer networks, in particular in systems such as Chord [54], Pastry [50] and Tapestry [60], to implement application-layer multicast [18,22,42,47], to tolerate multiple node and link failures in high-speed networks with arbitrary topology [67], and even to provide countermeasures to DDoS attacks [55,59,]. Overlay network technologies are also used by Akamai Inc. for dynamic content delivery [15,41,44,49,53]. Comprising more than 61,000 servers located over 1,000 networks in 70 countries worldwide, the Akamai platform delivers 15-20% of all Web traffic worldwide.

More recently, several frameworks have been proposed for overlaying virtualized Layer-2 networks over Layer-3 networks. Virtual Extensible LAN (VXLAN) is a proposed VLAN-like encapsulation protocol to encapsulate MAC-based OSI Layer 2 Ethernet frames within layer 4 UDP packets [43]. VXLAN will make it easier for network engineers to scale out a cloud computing environment while logically isolating cloud apps and tenants. It increases the number of available VLANs in a subnet from 4,000 to more than 16 million, and allows for Layer-2 adjacency across geographically distant IP networks. While VXLANs have certainly enabled a whole new level of scalability for virtual networks, one of the challenges in deploying VXLAN is its use of IP Multicast to implement the L2 over L3 network capability. IBM's virtual network overlay technology, called Distributed Overlay Virtual Ethernet (DOVE), differs from VXLAN in its ability to create an overlay without requiring the physical infrastructure to operate in multicast [9]. The main difference between the PANACEA routing overlay and the above mentioned technologies is the self-healing and self-optimizing properties of our system, whereas these technologies rely on the routes provided by Internet routing protocols, without seeking to control how

data flows are routed between end hosts.

In that respect, our system is much more closer to the solutions developed by the Detour and RON projects, which have clearly demonstrated the benefits of moving some of the control over routing into the hands of end-systems. The Detour framework [23] is an in-kernel packet encapsulation and routing architecture designed to support alternate-hop routing, with an emphasis on high performance packet classification and routing. It uses IP-in-IP encapsulation for sending packets along the alternate routes, and provides its flow classification and flow database as an in-kernel data structure for high performance. In contrast to the application-independent Detour Framework, the authors of RON believe that tighter integration of the application and the overlay network is important for functionality and deployment: it permits "pure application" overlays with no kernel modifications, and it allows the use of application-defined quality metrics and routing decisions [14]. RON is a software library that programs link against. RON nodes cooperate with each other to forward data on behalf of any pair of communicating nodes, forming an overlay network. To find and use alternate paths, RON monitors the health of the underlying Internet paths between nodes, dynamically selecting paths that avoid faulty areas. The RON architectures demonstrate that the methods for failure detection and recovery work well for discovering alternative paths in the Internet infrastructure. Although the objectives of our system are the same as those of Detour and RON, one advantage of our solution is that it can work with off-the-shelf applications, without requiring any modification of the application or the operating system. In addition, Detour and RON regularly evaluate the quality of all overlay links, which is clearly not scalable and can be only done for small overlay topologies, whereas our solution is designed to discover optimal paths with a limited monitoring effort.

The latter design objective is shared with a self-aware routing protocol called the *Cognitive Packet Network* (CPN) [29,30]. CPN provides QoS-driven routing, and performs self-improvement in a distributed manner by learning from the experience of special packets, which gather on-line QoS measurements and discover new routes. The routing decisions are made at each node of the network, and they are based on adaptive learning techniques using random neural networks. CPN has been shown to be effective for a variety of uses[2], and the pervasive monitoring system developed within PANACEA uses some of the fundamental concepts from CPN, as discussed in D2.1 [3] and D2.2 [4]. There are also many similarities between CPN and the autonomic communication overlay; both approaches provide a scalable solution for self-healing and self-optimization in communication networks. However, the innovation developed in PANACEA is specifically designed for overlay networks while CPN typically operates at the networking layer, although the application of CPN techniques to peer-to-peer overlay networks has been considered in [31]. Another crucial difference between the two approaches is that PANACEA's overlay network provides provable worst-case time guarantees for

[2] See http://san.ee.ic.ac.uk/publications.shtml for a complete list of references.

finding optimal paths in an unknown network since it addresses the generalized adversarial multi-armed bandit problem in a network setting. Finally, CPN is a patented technology [32], whereas we will make our overlay solution available to the public as an open-source software. Nevertheless, we intend to perform experiments with both solutions in order to better understand the benefits of each one in practical settings.

4 ARCHITECTURE OF THE ROUTING OVERLAY

In this Chapter, we describe the architecture proposed to achieve the objectives presented in Section 3.2. We first provide a global view of the architecture in Section 4.1 and then describe each of its components in Section 4.2.

4.1 Global view of the architecture

As shown in **Figure 6**, the overlay network is formed of software routers deployed at the cloud sites and possibly in other locations in the Internet.

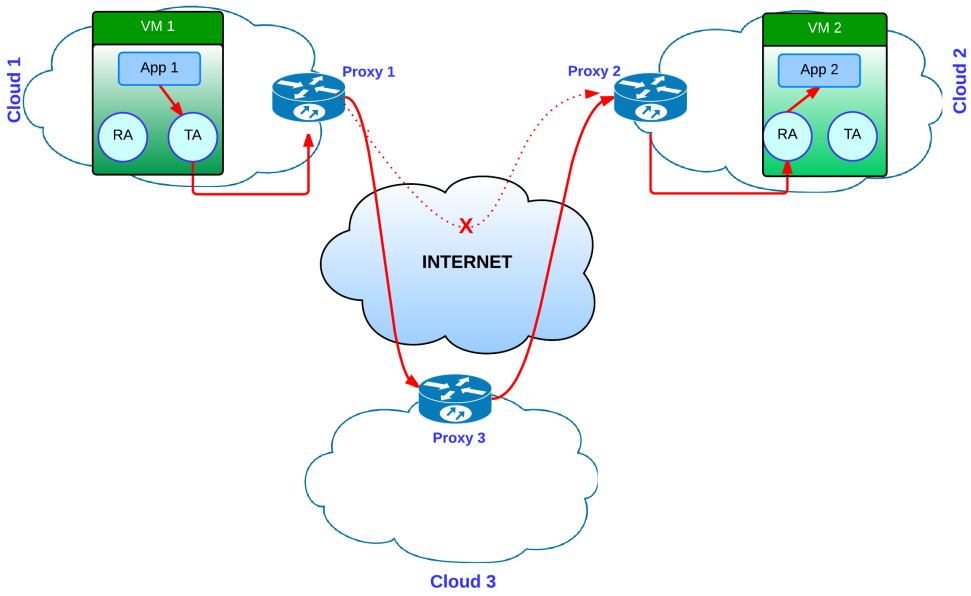

Figure 6: Architecture of the Autonomic Communication Overlay.

In this architecture, two types of agents are used. Transmission (TA) and Reception (RA) agents are run on each VM running a task of the distributed application, and represent the entry and exit points of the overlay network respectively. Each cloud also runs a software router called a Proxy. The Proxy is in charge of monitoring the quality of the overlay paths towards certain destinations, selecting the best paths and forwarding the packets emitted towards them.

As described in Figure 6, the traffic flows are routed over the overlay network, enabling us to avoid congested or failed parts of the Internet when a Proxy detects that the primary Internet path is subject to anomalies. The routing overlay is thus able to adapt its routing scheme according to what is observed from the underlying network. In addition, this approach enables to integrate routing and path selection with distributed applications more tightly, that is, it gives the ability to consult application-specific metrics when selecting paths.

4.2 Components of the system

4.2.1 Transmission and reception agents

Let us recall that one of our design objectives is to control the path of data of an application through the network, without the application even being aware that its data flows are routed over the overlay. To this end, we use packet interception and encapsulation mechanisms operating in a transparent way for the application. These mechanisms are implemented by two software agents, which are activated automatically at start-up of their respective VM:

- **Transmission agent**: the role of the Transmission Agent (TA) is to intercept the packets sent by the application running in the same VM and to forward them to the local Proxy using IP-in-IP encapsulation. The Proxy will then handle the forwarding of the packet towards the destination cloud. On the data-forwarding path, the TA is therefore the entry node in the overlay network.
- **Reception agent**: the role of the Reception Agent (RA) is to receive the packets sent by the local Proxy and to deliver the original packets to the local application running in the same VM. On the data-forwarding path, the RA is therefore the exit node of the overlay network.

4.2.2 Proxy

An agent, called a Proxy, is executed in each cloud and acts as an intermediary for communications with other clouds.

As described in Figure 7, the Proxy is an entity constituted of three different software agents:

- **Monitoring agent**: it monitors the quality of the Internet paths between the local cloud and the other clouds in terms of latency, bandwidth, and loss rate. The monitoring agent can be queried by the routing agent in order to discover the quality of a given path according to a certain metric. It can also be configured to monitor the availability of a path at regular time intervals. We describe in more depth the methods used to estimate the quality of overlay links in Section 6.
- **Routing agent**: This agent is configured to optimize a service-specific routing metric towards certain destinations. To this end, it drives the monitoring agent so as to discover an optimal path (e.g., low-latency, high-throughput, etc.) with a minimum monitoring effort. In Section 7, we describe in details the methods used to learn the optimal paths in the overlay. For each destination, the optimal path towards that destination discovered by the routing agent is written in the routing table of the forwarding agent.

- **Forwarding agent**: this agent[3] is in charge of forwarding each incoming packet to its destination on the path it was instructed to use by the routing agent. We use source routing, that is, the routing table of the source Proxy describes the complete path to be followed by a packet to reach its destination. Each subsequent Proxy simply determines the next forwarding hop from the information contained in the PANACEA header. The final Proxy forwards the packet to the appropriate RA that then delivers the original packet to the destination application. Section 5 describes in more details the packet forwarding process.

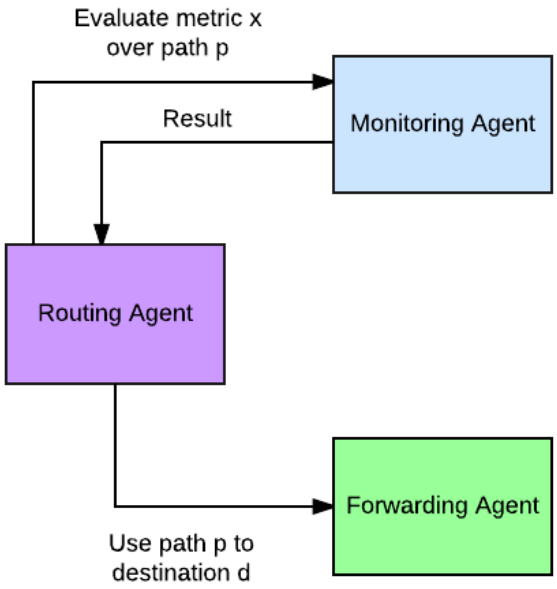

Figure 7: Interactions between the entities constituting the proxy.

4.3 Integration in the PANACEA architecture

The autonomic communication overlay is designed as a stand-alone component protecting distributed services deployed in multiple clouds from performance failures of the Internet. However, it is an integral part of the PANACEA cloud management solution. The interaction between the service and cloud managers of the PANACEA solution and the routing overlay occurs through the Routing Agent of the Proxy, as shown in **Figure 8**. In this architecture, the autonomic service manager (ASM) handles the proactive and autonomic management of services and applications through the interface provided by the cloud manager.

[3]We emphasize that, although a single forwarding agent was used in our experiments, it is possible to use multiple forwarding agents for load-balancing purposes.

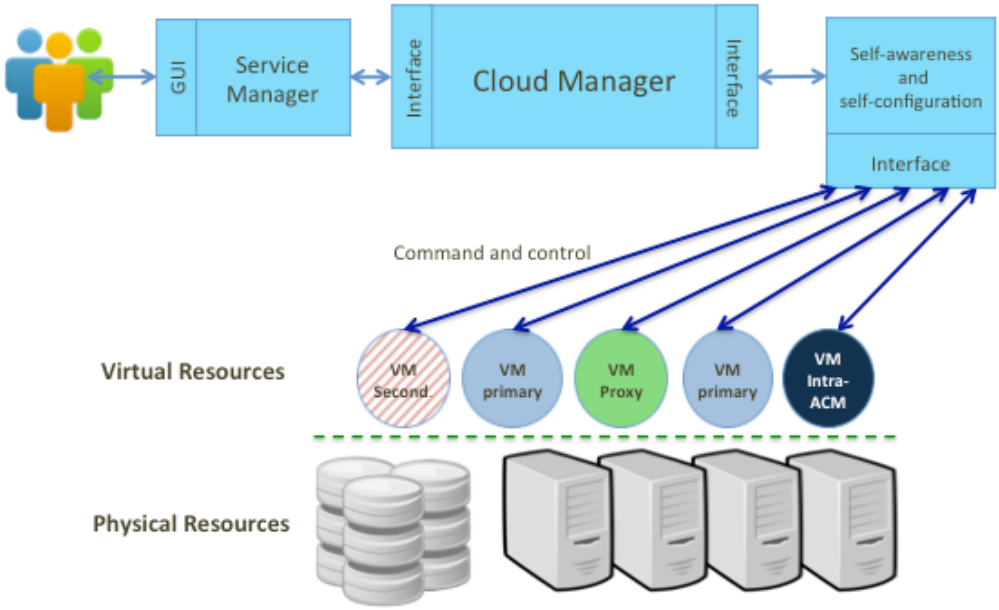

Figure 8: Integration in the PANACEA architecture.

The overlay architecture is generic enough to fit any cloud architecture. We mainly need to deploy the proxies, and this can be done statically or dynamically as a virtual appliance, that is as a pre-configured VM image, ready to run on a hypervisor. The new mechanisms introduced in the cloud manager for self-awareness and self-configuration will be used to inform the Proxy of the IP addresses to which messages may be sent by local tasks.

The RA and TA will be activated automatically at start-up of each VM running a component of a PANACEA-enabled service. A direct communication channel will be established between each TA and its local Proxy using connection-oriented sockets, thereby allowing the Proxy to activate or deactivate the interception of IP packets sent towards specific destinations.

5 PACKET INTERCEPTION, ENCAPSULATION AND FORWARDING

In our architecture, the forwarding of a packet from its source to its destination proceeds as shown in Figure 9.

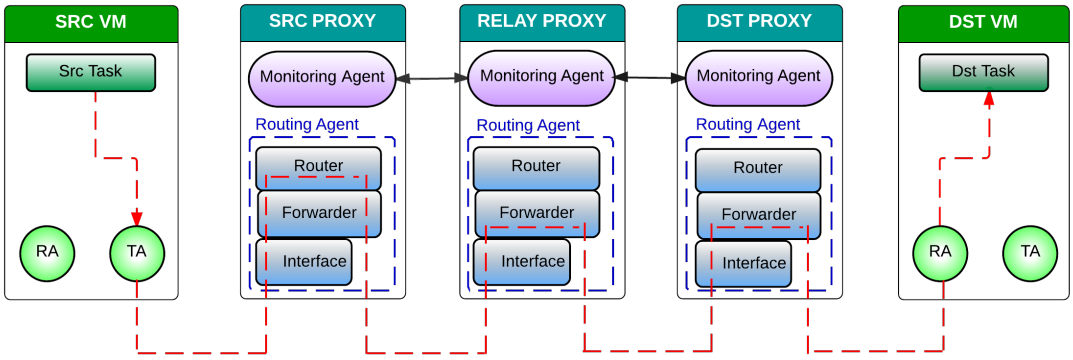

Figure 9: Forwarding process.

When a packet is sent by a source task to a destination task located in a different cloud, it is first intercepted and forwarded to the TA. The TA uses IP-in-IP encapsulation to forward an altered packet to the local Proxy. The payload of the altered packet is that of the original packet along with an additional Panacea header. Upon reception of the PANACEA packet, the routing agent of the Proxy looks-up its routing table in order to determine the path to the destination. The choice of source routing is dictated by scalability considerations (see Section 7). The sequence of intermediate proxies is written in the PANACEA header, and then the PANACEA packet is forwarded to the first one of these proxies. Each intermediate proxy then forwards the packet to the next hop on the path, until the final proxy is reached. When this occurs, the packet is forwarded to the RA of the destination VM. The RA decapsulates the PANACEA packet and forwards the original IP packet to the destination task using a raw socket. We present below the technical details of each of these operations.

5.1 Packet interception

The TA needs a mechanism so as to be able to intercept the packets sent by the application running in the same VM and to forward them to the local Proxy. We emphasize that the TA does not intercept all packets, but only packets towards specific destinations located in a different cloud. The list of destination IP addresses for which packet interception has to be done is configured at start-up of the agent and can be changed dynamically. Packet interception is realized using a filtering mechanism known as NetFilter NFQUEUE.

Netfilter is a framework inside the Linux kernel [12], which offers flexibility for various networking-related operations (e.g. packet filtering or network address translation) to be implemented in form of customized handlers. Netfilter represents a set of hooks inside the Linux kernel, thus it allows specific kernel modules to

register callback functions with the kernel's network stack. These functions, usually applied to the traffic in form of filtering and modification rules, are called back for every packet that traverses the respective hook within the network stack.

NFQUEUE is an iptables target, which delegates the decision on packets to user-space software (the TA in our case). Figure 10 shows the network filter queue for packet interception under Linux. We note that this is the only part of the code that is non portable, and that similar packet interception mechanisms exist in other operating systems.

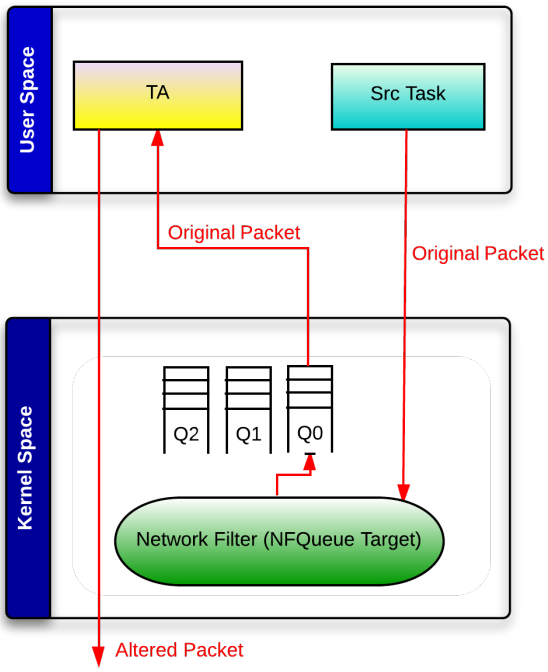

Figure 10: The network filter queue for packet interception under Linux.

5.2 Packet encapsulation

Upon receipt of the packet sent by the local application, the TA takes the entire content of the packet received and encapsulates it into its own message format, adding a PANACEA header that contains control information. The PANACEA packet is then sent to the local proxy using UDP. The encapsulation process is shown in Figure 11.

The PANACEA header follows a specific format. It contains in particular the IP address of the destination Proxy (which differs from that of the local Proxy, in the outer IP header) as well as the complete path to reach it (list of intermediate Proxies[4]). The TA leaves the latter field blank, since the path to the destination Proxy will be determined by the forwarding agent of the source Proxy.

[4]In our implementation, the length of a path is at most L_{max} (the value of this parameter is set to 4 by default).

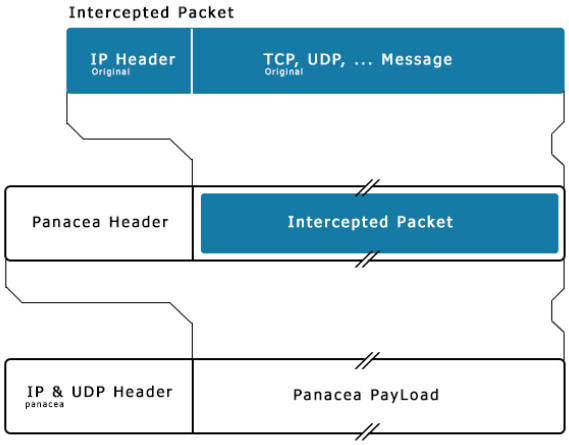

Figure 11: Packet encapsulation.

5.3 Processing by the forwarding agent

This agent is in charge of forwarding packets to their final destination. Upon receipt of a packet, the forwarding agent inspects its PANACEA header to determine its precise role. There are three cases:

- **The packet is at the source Proxy**: this is the case if the Proxy is not the final destination and if the field describing the end-to-end path is blank. In that case, the forwarding agent looks up for the path to the destination Proxy in its routing table, writes this path in the PANACEA header of the packet, and then forwards it to the next hop on the path.
- **The packet is at an intermediate Proxy**: the forwarding agent then just forwards the incoming packet to the next hop on the path, after having updated its destination IP address.
- **The packet has reached the destination Proxy**: the forwarding agent then forwards the packet to the RA on the destination VM.

5.4 Decapsulation and transmission to the destination

The RA decapsulates the Panacea packet and forwards the original data packet to the destination task using a raw socket, that is, an internet socket that allows the direct sending and receiving of IP packets without any protocol-specific transport layer formatting. We note that the packet is directly delivered to the recipient application because the destination IP address is that of the destination VM.

We also note that there is an additional difficulty when the public IP address of the destination VM is different from its private IP address. Since hosts located in different IP networks communicate via their public IP addresses, the original packet sent by the source task contains a public IP address. The automatic remapping of

public IP addresses into private IP addresses by the Network Address Translation (NAT) mechanism is therefore only possible for the PANACEA packet, and not for the encapsulated packet. As a consequence, the original packet cannot be send as it is to the destination application. To overcome this difficulty, the RA uses a configuration file containing translation table entries to convert the public IP address of the packet into a private address. In addition, IP header checksum and any higher-level checksums that include the IP address are also changed by the RA.

6 METERING THE QUALITY OF OVERLAY LINKS

The probe agent implements the methods to measure latency, loss rate or observed throughput on a path in the overlay. We describe below the methods used to estimate each of these metrics.

6.1 Latency and loss rate

The principle of latency and loss measurement is quite simple. We send probe packets along specific paths. At each node of the path, a timestamp is added to the packet. When the packet returns to its source, it contains all the timestamps, in both directions. Thus at the end of the operation we can easily deduce latency on each network segment. We send 5 consecutive probe packets to estimate the average transmission delay on each link of the path. A packet that does not return to its destination before a timeout (set by default to 10 seconds, but configurable by the user) is considered lost. It is then possible to determine an imprecise but sufficient percentage of losses to detect a path failure. The structure of the probe packet is shown in *Figure 12*.

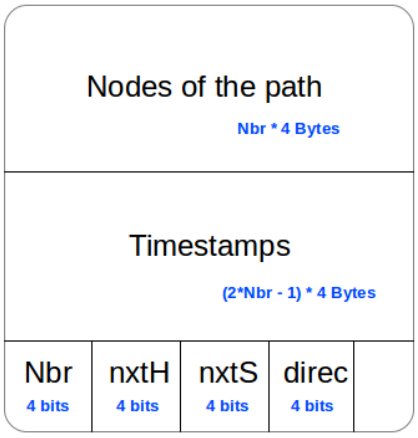

Figure 12: Structure of latency probe packets.

This structure is quite similar to the header of a Panacea data packet. It mainly consists of a field containing the nodes of the path and another for timestamps storage. The other fields are counters used to determine the next node and the packet propagation direction (forward or reverse). In our implementation we arbitrarily fix the maximum number of hops for a path with the parameter L_{max} (set to 4 by default).

6.2 Available bandwidth

The available bandwidth is a key metric for some applications. We evaluate below two estimation techniques: a passive method for TCP based on the Square Root formula (SQRT) [61] and an active method called Train Of Packet Pair (TOPP) [62,63]. These techniques have not yet been implemented in the routing overlay.

6.2.1 Passive measurements

We evaluate here a method for predicting TCP throughput from non-invasive data. The method is based on a simplified model of TCP, which assumes a persistent TCP connection in the Congestion Avoidance phase and independent random packet losses according to a Bernoulli process. In this method, the available bandwidth is estimated as

$$A = \frac{1}{\mathrm{RTT}}\sqrt{\frac{3}{2p}} \text{ packets/second,}$$

where p is the packet loss probability. We compare the estimation provided by the SQRT method with the bandwidth measured with the iperf utility on the CORE emulator (see Section 8.1). We consider a simple case with two nodes connected by a single link with a fixed RTT of 100ms and a variable loss rate. We perform measurements for a period of 30 seconds with iperf, using a TCP window of size 64 KB and packets of size 1.5KB. Each measurement is repeated 10 times. The results are shown in Figure 13. The relative error is often greater than 100 %. The poor quality of the results leads us to consider a more sophisticated active method like TOPP.

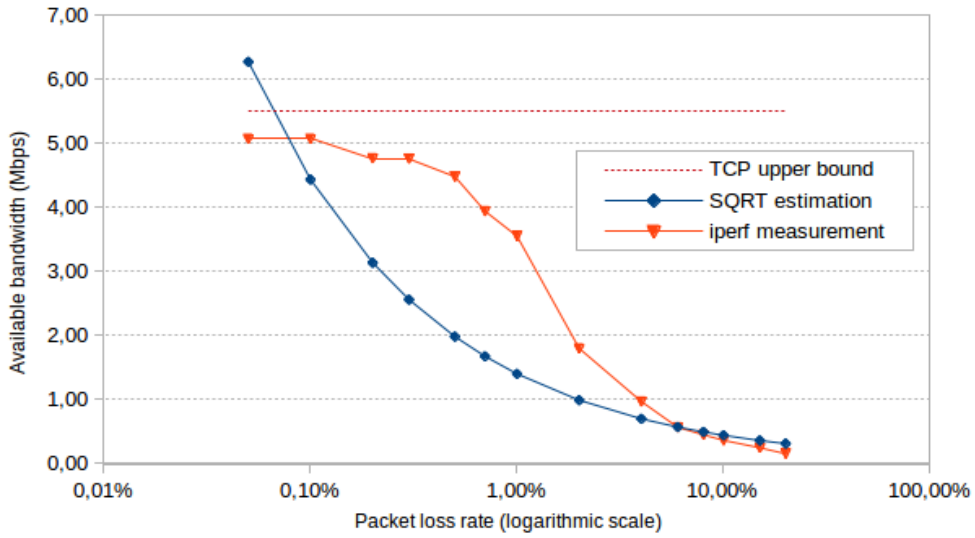

Figure 13: SQRT available bandwidth estimation, for a 100ms RTT,

6.2.2 Active measurements

Melander et al. proposed a measurement methodology called *Trains of Packet Pairs* (TOPP) to estimate the available bandwidth of a network path [62,63]. TOPP sends many packet pairs at gradually increasing rates from the source to the sink. Suppose that a packet pair is sent from the source with initial dispersion Δ_s. The probing packets have a size of L bytes and thus the offered rate of the packet pair is $R_o = L/\Delta_s$. If R_o is more than the end-to-end available bandwidth A, the second

probing packet will be queued behind the first probing packet, and the measured rate at the receiver will be $R_m=L/\Delta_t<R_o$. On the other hand, if $R_o<A$, TOPP assumes that the packet pair will arrive at the receiver with the same rate it had at the sender, i.e., $R_m = R_o$. TOPP estimates the available bandwidth A to be the maximum offered rate such that R_o is approximately R_m.

Validation with the CORE network emulator

To evaluate the quality of the estimation done by TOPP, we emulate a simple network within the CORE emulator, with 2 nodes connected by a single link of fixed capacity C. Delays and losses on the link are set to 0. We introduce a UDP background traffic using 50 % of the link capacity.

Figure 14 shows the ratio Δ_t/Δ_s as a function of the sending rate Δ_s for four different link capacities C (250 kbps, 500 kbps, 1000 kbps and 5000 kbps). For the three smaller capacities, we observe the expected trend with a constant value approximately equal to 1 for $R_o<A$ and an increase for $R_o>A$. We note that the inflection point of the curve is approximately located at the available bandwidth A. For C = 5000 kbps, it is impossible to observe any trend: sending only two packets is insufficient for large bandwidths.

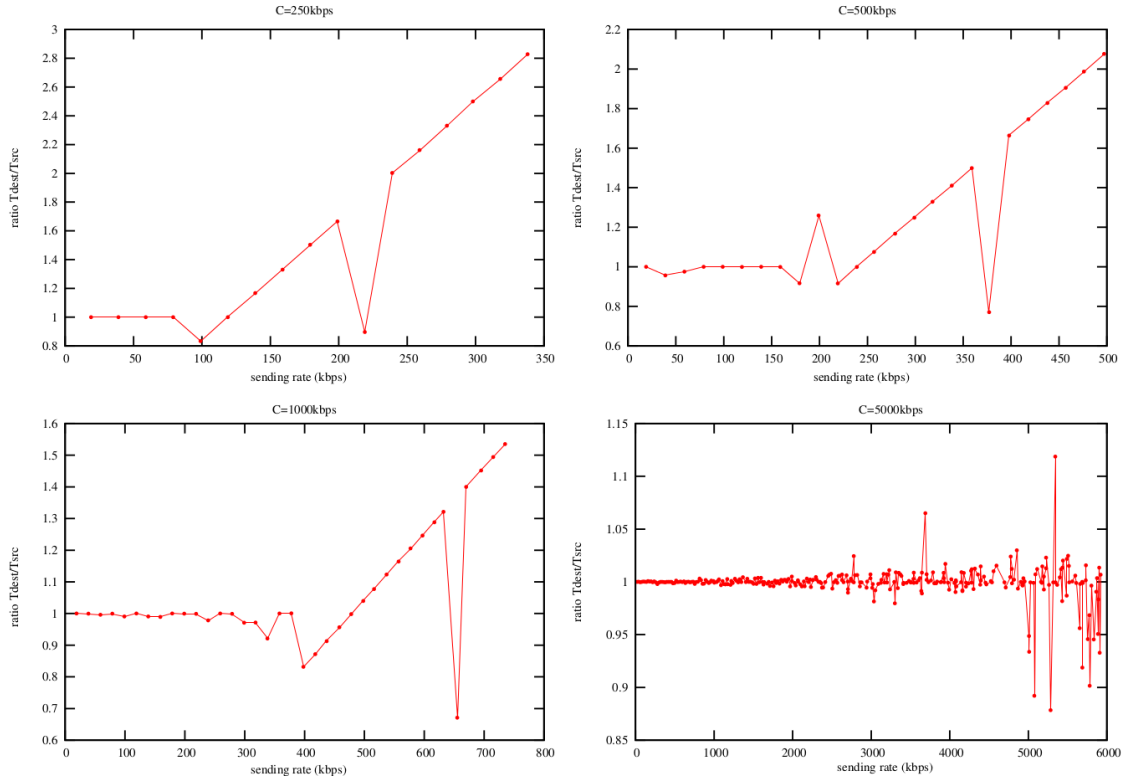

Figure 14: TOPP behaviour on a 50% congested link

Note that some noise is observed for the first three configurations. In practice, the TOPP method requires a complex statistical processing of the measurements. We therefore consider in the following the implementation of TOPP provided by the

tool IGI [64]. We test the results obtained with this tool on the previous 4 configurations, with and without background traffic. The results (mean of 10 measurements) are presented in Table1. According to these results, IGI seems to provide a good estimation of the available bandwidth. The total time needed to perform a measurement range from 1.6 seconds to over 50 seconds, depending on the available bandwidth. An experiment for a 400ms RTT has shown that the RTT of the link does not significantly impact the measurement time (the maximum measurement time is then around 1 minute).

C (kbps)	No background trafic			50 % background trafic		
	A (kbps)	IGI (kbps)	t(s)	A (kbps)	IGI (kbps)	t(s)
250	250	246	15,9	125	117	52,2
500	500	499	8,3	250	234	27,1
1000	1000	973	4,6	500	467	14
5000	5000	5015	1,6	2500	2302	4,5

Table1: Available bandwidth estimation with IGI (RTT = 40ms).

Validation in the Internet

We used the IGI tool to estimate the bandwidth available on a real Internet path between two remote machines. We compare the estimation results with those obtained using the iperf measurement tool (version 2.0.5), gradually increasing the UDP background traffic. Bandwidth estimations are summarized in *Figure 15*. Each point corresponds to the average of 10 successive measurements. Some error is observed for the lowest background traffic, but accuracy improves when the available capacity becomes smaller. These results demonstrate that the TOPP method can be reliable.

It is also important to quantify the measurement effort required to obtain the estimations. *Figure 16* shows the total amount of data sent for each measure, as well as the average sending rate of probe packets. We note that a few hundred KBytes are necessary to obtain a measurement, and that the sending rate decreases with the available capacity.

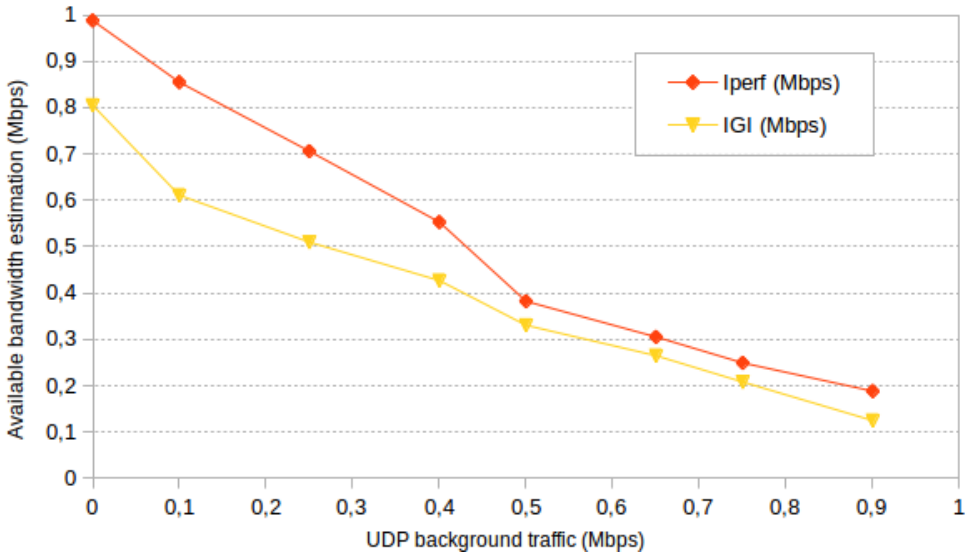

Figure 15: Available bandwidth estimation on a real Internet path with IGI.

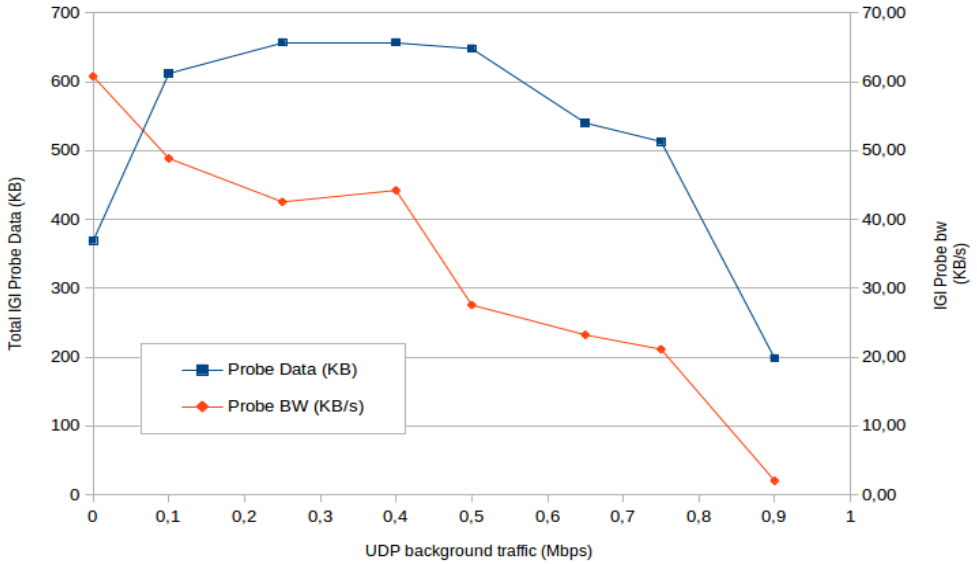

Figure 16: IGI probing effort on a real Internet path.

Quality of path selection

Sometimes, the estimations provided by IGI suffer from significant variations. We study here how the TOPP method allows us to choose the path with the greatest available bandwidth. To this end, we define two parallel paths in the CORE emulator (see *Figure 17*). This topology has a source node S and two destinations nodes t_1 and t_2. The two links have the same capacity C = 10 Mbps. We introduce a UDP background traffic between nodes S and t_1, with a demand d_1 = 5 Mbps, so that the upper link always offers an available bandwidth of 5Mbps. We also inject background traffic on the bottom link at a variable rate d_2.

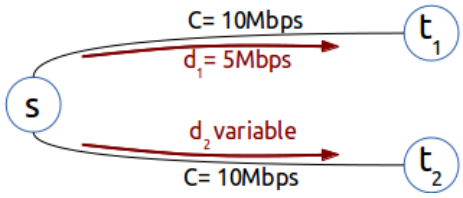

Figure 17: Topology used in the CORE emulator to validate the chosen path.

For each value of d_2 we estimate the bandwidth with IGI between S and t_1, and S and t_2, and store the estimated values for the two links. We repeat each measurement 100 times, and use a moving average over the last N steps to choose the path with the largest capacity (N=1 disables the moving average). For each value of d_2 and N, we calculate the percentage p of times the best path has been chosen.

A wrong choice of path when the bandwidth of the two paths is close does not have the same impact as a wrong choice when the two paths offer very different bandwidths. We thus use the following ratio to assess the quality of the path selection obtained with TOPP:

$$r = \begin{cases} \dfrac{p(C-d_2)+(1-p)(C-d_1)}{C-d_2} & \text{if } d_2 < d_1 \\ \dfrac{p(C-d_1)+(1-p)(C-d_2)}{C-d_1} & \text{if } d_2 > d_1 \end{cases}$$

Figure 18 shows the values of this ratio for several values of N as a function of the rate d_2. Note that even when a moving average is not used, the bandwidth obtained is greater than 90 % of the optimal bandwidth. When a moving average is used, the results are further improved and become higher than 95 % of the optimal bandwidth.

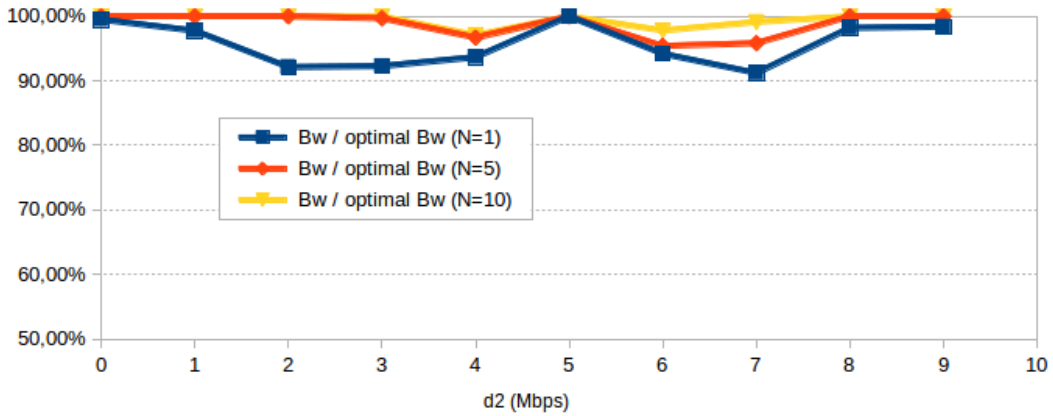

Figure 18: Ratio of obtained bandwidth divided by the optimal bandwidth.

6.2.3 Chosen solution

As the SQRT method is only valid for the TCP protocol and in view of the poor quality of the estimations provided by this method, we opt for the active measurement method TOPP. The results obtained with IGI show that a good implementation of this method can provide reliable estimations of the available bandwidth.

7 DISCOVERING THE OPTIMAL ROUTES

As mentioned in Section 3.3, most routing overlays use all-pairs probing, resulting in a costly $O(n^2)$ probing overhead as the number of participating nodes n increases. Evidence indicates that such an $O(n^2)$ approach is able to scale to approximately 50 overlay nodes. One of our design goals is to build a routing overlay that can be widely deployed over a sizable population of routers. In this Section, we present the methods that our solution uses for discovering optimal routes in large overlay networks with a minimum probing effort.

We first present an algorithm that enables to measure the latency of all overlay links from probe packets sent by the source node. This algorithm is not scalable and is implemented in our system only for comparison purposes. We then introduce the *Optimal Path Discovery Problem*, where the goal is to minimize the number of uncovered edges in order to discover an optimal path between two given nodes in an unknown complete graph. Our results on this problem show that it cannot be guarantee that an optimal path will be discovered with a subquadratic probing effort. Finally, we present an algorithm for progressively learning the optimal routes in the overlay topology with a constant probing effort.

7.1 Covering the graph with an Eulerian cycle

How to measure the latency of all overlay links by sending probe packets from the source? To this end, we propose an algorithm using the time stamps stored in probe packets. We view the overlay network as a complete graph. The algorithm uses a Eulerian cycle in this complete graph in order to measure the metric of all edges as efficiently as possible.

In graph theory, an Eulerian cycle is a path, which starts and ends on the same vertex and visits every edge exactly once. Euler proved that a necessary condition for the existence of Eulerian circuits is that all vertices in the graph have an even degree [25]. For a complete graph, this implies that the number n of nodes has to be odd, which is assumed in the following[5]. In that case, an Eulerian cycle is easily computed using Fleury's algorithm [28]. Let us denote by s the source node. The Eulerian cycle is necessarily composed of K=(n-1)/2 sub-cycles:

$$C=s, a_1, a_2,\cdots, a_{l1}, s, a_{l1+1},\cdots, a_{l1+l2}, s,\cdots, s, a_{l1+l2+\cdots+lK}, s$$

We exploit the specific structure of the Eulerian cycle to obtain a set of (n-1)/2 edge-disjoint paths enabling to cover the full graph. Each of these paths is then split again in path of length at most L_{max}, on which probe packets are sent.

Once collected the metrics of all overlay links, a shortest path algorithm is used to find an optimal path from the source to the destination. Note that Dijkstra's algorithm cannot be used due to the constraint on path length. We instead use an adaptation of Bellman-Ford algorithm described in [45].

[5]The case of an even number of nodes is dealt with a slight adaptation of the described technique.

7.2 The optimal path discovery problem

Shortest path problems are arguably among the most fundamental problems studied in computer science and have a variety of applications in as diverse areas as computational geometry, geographical information systems, network optimization and robotics, to mention just a few. This is reflected by the large body of literature devoted to shortest path problems and by the many algorithms that have been proposed to solve them (see, e.g., [65] for an introduction). Nevertheless, in some situations the difficulty lies more in collecting the information on the graph than in the shortest path computation itself because the cost of obtaining the length of the edges dominates the total cost of computing a shortest path. This is clearly the case for routing overlays where the cost of collecting the metrics of the links by sending probe packets is far much higher than that of computing a shortest path.

In such situations, it makes sense to seek to minimize the amount of information required to compute a shortest path. To study such situations, we introduced in [21] the *Optimal Path Discovery* (OPD) problem in graphs. In the OPD problem the costs of the edges are initially unknown but can be discovered by *querying* an oracle. The goal is to find an optimal path between two given nodes querying the minimum number of edges, where the cost of a path is a function of the costs of the edges it is comprised of.

In [21], we investigate the OPD problem for a broad class of cost functions, including, but not restricted to, additive functions for which the cost of a path is the sum of the costs of its edges. We handle both the cases when the performance metric has to be minimized (e.g., to find a minimum latency path) or maximized (e.g., to find a path with maximum probability of successful transmission). We consider as well the extension obtained by relaxing the constraint that an optimal path has to be discovered and allowing the cost of a feasible path to be at most αtimes that of an optimal path for some $\alpha \geq 1$. Of course, we fall back on the original problem when $\alpha=1$. We note that for additive performance metrics and $\alpha=1$ the OPD problem coincides with the so-called *Shortest Path Discovery* problem introduced in [66]. As detailed in [66], this problem finds applications in speech recognition, systems based on segmentation lattices, hierarchical planning, or exploration of unsafe environments.

Since our particular motivation for studying the OPD problem comes from routing overlays, we focus the analysis on the case of complete graphs. We formally state the problem in Section 7.2.1, and then present our main results in Section 7.2.2.

7.2.1 Problem Statement

Definitions

Let $G=(V,E)$ be a complete undirected graph with n nodes and consider two nodes $s, t \in V$. It is assumed that each edge of G has an *unknown positive value*. We denote by $f(e)>0$ the unknown value of edge $e \in E$. We assume that a function $F:2^E \rightarrow [0,\infty)$ determines the value of a set of edges. That is, for any set of edges $H \subseteq E$, $F(H)$ is the value of H. According to the above definition, $F(P)$ is the value of the (s,t)-path P. Depending on the context, an *optimal path* between nodes s and t is either an path P minimizing F(P) over all (s,t)-paths, or an (s,t)-path P maximizing F(P) over all (s,t)-paths. In the following, we denote by δ the value of an optimal (s,t)-path.

In some situations, rather than requiring an optimal path between nodes s and t, we might be less ambitious and be satisfied with a path providing some performance guarantee. To account for such situations, we introduce the concept of an α*-approximation* of an optimal path, which is formally defined as an (s,t)-path P such that $F(P) \leq \alpha\delta$ in the minimization case, and $F(P) \geq \alpha\delta$ in the maximization case. Note that an optimal path is a 1-approximation, so that searching for an α-optimization is equivalent to searching for an optimal path in the case $\alpha = 1$.

Given $\alpha \geq 1$, in order to discover an α-approximation of an optimal path between s and t, and to be able to *guarantee* that the discovered path is indeed an α-approximation, an algorithm has to uncover the unknown values of some of the edges. We use the abstraction of an oracle for that purpose. When an algorithm queries the oracle for the values of a set of edges $H \subset E$, the oracle reveals to the algorithm the value f(e) of all edges $e \in H$. For short, we say that the algorithm *uncovers a set of edges* when we refer to this process.

We say that a set of edges $C \subseteq E$ is an α*-certificate* of a (s,t)-path in G if and only if:

- The value of the edges in C is known, whereas edges in $E \setminus C$ have an unknown value,
- C contains a path from s to t, the so-called *proposed path*, and
- There are enough uncovered edges in C to guarantee that the proposed path is an α-approximation.

We remark that the proposed path belongs to the α-certificate, implying that its value is fully known. Note also that, for any $\alpha \geq 1$, there always exists at least one α-certificate since the set E that contains all the edges of the graph is an α-certificate.

Mathematical formulation of the OPD Problem

The goal of the OPD problem is to minimize the number of queries made to the oracle in order to obtain an α-certificate of a (s,t)-path. Formally, the problem is defined as follows

Minimize $|C|$

subject to

C is an α-certificate of a (s,t)-path.

In other words, the goal is to find a set of fully uncovered edges of minimum cardinality enabling to guarantee that an α-approximation of an optimal path has been discovered. A natural performance measure for the efficiency of an algorithm solving the OPD problem is the worst-case number of queries it does for instances of size[6] n. Let us denote by $U(A(I))$ the set of edges whose value has been uncovered by the algorithm A when solving the OPD problem on the instance I. Formally, the *number of queries* for instances of size n of algorithm A is β_n if and only if $|U(A(I))| \leq \beta_n$ for all instances of size n.

As we shall see, the *number of queries* does not allow to correctly differentiate algorithms according to their performance. Therefore, we introduce a new measure of the performance of an algorithm, the *query ratio*, which is defined as

$$\boldsymbol{max_I \frac{|U(A(I))|}{OPT(I)}},$$

where OPT(I) is the size of the smallest (in cardinality) α-certificate of the instance I.

7.2.2 Summary of Main Results

Under some assumptions on how the value F(H) of a set $H \subset E$ is computed from the values f(e) of the edges $e \in H$ (see [21] for details), we were able to obtain lower and upper bounds on the *number of queries* and on the *query ratio* of an algorithm. These bounds are presented below.

Lower Bounds on the Number of Queries β_n

It can easily be proved that, for any $\alpha \geq 1$, all α-certificates contain a cut-set separating node s from node t. Since the smallest cut-set in a complete graph of size n has size n-1, we obtain the following result.

Lemma 1. For any algorithm that solves the OPD problem for a finite approximation factor $\alpha \geq 1$, it holds that $\beta_n \geq$ n-1.

Nevertheless, the previous lower bound on βn is optimistic since there exist cases in which any algorithm needs to uncover strictly more than n-1 edges. Our precise result is stated below.

Lemma 2. For any algorithm A that solves the OPD problem for a finite approximation factor $\alpha \geq 1$, there exists a bad instance for which A requires at least $n^2/4$ uncovered edges in order to provide an α-approximation.

[6] We say that an instance I is of size n if the associated complete graph has n nodes.

For routing overlays, these results show that in order to provide a performance guarantee on the path used, we will need a number of probed links that is at least linear in the number of nodes, and, in some cases, even quadratic in the number of nodes. Interestingly, any algorithm may still have to do a number of queries of order n^2 even if we are ready to accept a significant performance degradation by requiring only an α-approximation for some large value of α. In that respect, the number of queries is not a fair measure of the performance of algorithms solving the OPD problem, and we therefore focus on the study of the query ratio.

Lower and Upper Bounds on the Query Ratio

Our first bound on the query ratio is obtained by considering an algorithm A and designing a malicious adversary[7] of this algorithm that plays the role of the oracle.

Theorem 1. For any algorithm *A* that solves the OPD problem for a finite approximation factor $\alpha \geq 1$, there exists an instance I for which the following inequality holds

$$\frac{|\boldsymbol{U}(\boldsymbol{A}(\boldsymbol{I}))|}{\boldsymbol{OPT}(\boldsymbol{I})} \gg \mathbf{1} + \frac{\mathbf{4}}{\boldsymbol{n}} - \frac{\mathbf{8}}{\boldsymbol{n^2}}.$$

Lemma 2 implies that it is not possible to design an algorithm that will make the minimum number of queries for any instance. In [21], we propose an algorithm whose query ratio is strictly lower than 2, as formally stated in the following theorem.

Theorem 2. In the case α=1, Algorithm 2 in [21] has a query ratio which is at most 2-1/(n-1) for instances of size n.

We refer to [21] for the details of the algorithm, as well as the experimental evaluation of its performance.

[7] That is, an adversary whose goal is to maximize the number of queries made by the algorithm.

7.3 Learning algorithms

We wish to build a routing overlay that can be widely deployed over a sizable population of routers, implying that the monitoring effort (i.e., the number of probed links) should grow at most linearly with the number of nodes of the overlay. Unfortunately, our results on the OPD problem (in particular Lemmas 1 and 2) prove that there is no hope to obtain even a modest performance guarantee with such a monitoring effort. Therefore, instead of requiring a performance guarantee at each time step, we look for an online decision algorithm that uses a limited monitoring effort but achieves asymptotically the same average (per round) end-to-end latency as the best path. The idea is to design an algorithm that exploits past observations so as to quickly learn link parameters and efficiently track the optimal path.

We formulate this problem as an online shortest path problem [35], and consider it in the adversarial setting where link delays can change arbitrarily from one time step to the other. In this setting, no probabilistic assumption is made regarding the delays on the links, and in particular there is no independence assumption made on the delays of the links. Furthermore, we assume that at each time slot, the algorithm chooses a subset of paths to probe, and that it only observes the latencies of edges in the probed paths [16,17,20]. The algorithm then sends its packet over the minimum latency path among those it has chosen to probe. In this model, the delays may change from one time slot to the next one in an arbitrary way, and our goal is to find a way of choosing the paths to probe at each time slot such that the sum of the total delays incurred by packets over time is not significantly more than that of the best fixed route in the network. We first describe the mathematical model we are considering, and then present the algorithm we have implemented so far.

Mathematical Model

Consider a network represented by a graph G=(V,E), where V is the set of vertices and E the set of edges, and assume that we have to send packets from a distinguished vertex s, called source, to another distinguished vertex t, called destination. At each time slot a packet is sent along a chosen route connecting s and t. In each round t=1, . . ., n of the decision game, the algorithm chooses a subset K_t of paths (the probed paths) among all paths from s to t. Then a loss $\ell_{e,t} \in [0, 1]$ is assigned to each edge $e \in E$, and the algorithm observes the latency $\ell_{e,t}$ of edge e that belongs to a path in K_t. At round t, the packet is then sent over the path in K_t with the minimum latency, yielding the latency

$$\boldsymbol{\ell_t = min_{P \in K_t} \sum_{e \in P} \ell_{e,t}}$$

The *cumulative latency* of the algorithm over n rounds is the sum of the latencies incurred by all packets sent, that is

$$\boldsymbol{L_n = \sum_{t=1}^{n} \ell_t}$$

whereas the cumulative latency of a path P over the n rounds is

$$\boldsymbol{L_{P,n} = \sum_{t=1}^{n} \sum_{e \in P} \ell_{e,t}}$$

The *normalized regret* of the algorithm with respect to the best path is then defined as

$$R_n = \frac{1}{n}\left(L_n - min_P L_{P,n}\right)$$

where the minimum is taken over all paths P from s to t. The goal is to design an algorithm that chooses the set K_t of paths to be monitored at time t so as to perform asymptotically as well as the best path, i.e., such that R_n converges to 0 as n grows to infinity, uniformly over all outcomes sequences.

Online Decision Algorithms for the Shortest Path Problem

For the basic multi-armed bandit problem[8], Auer et al. [17] gave a randomized algorithm, based on exponential weighting with a biased estimate of the gains (defined, in our case, as $g_{P,t}=K-\ell_{P,t}$ for path P), combined with uniform exploration. Their algorithm is called EXP3. Applying EXP3 to the on-line shortest path problem in the bandit setting results in a performance that can be bounded, for any $0<\delta<1$, and fixed time horizon n, with probability at least 1-δ, by

$$R_n \leq \frac{11\,K}{2}\sqrt{\frac{N\,ln\left(\frac{N}{\delta}\right)}{n}} + \frac{K\,ln(N)}{2n}$$

where N is the number of possible paths and K is an upper bound on the length of a path. Note that the regret of this algorithm decreases in time according to $1/\sqrt{n}$. As a first step towards the use of a learning algorithm in our routing overlay, we have implemented in the routing agent a slight extension of the EXP3 algorithm. In our extension, the algorithm chooses a subset of paths to probe, observes the latencies of these paths, and then use the minimum latency one. The direct Internet path is always included in the list of paths to be probed. More precisely, the algorithm is as follows:

1: $\omega_i(1) = 1, i = 1, \ldots, N$

2: **For** t=1,2,⋯,n **do**

3: Compute the probability of each path i=1,⋯,N

$$p_i(t) = (1-\gamma)\frac{\omega_i(t)}{\sum_j \omega_j(t)} + \frac{\gamma}{N}$$

4: Choose randomly the set K_t of paths to probe according to p(t)

5: Select the best path $i = argmin_{P\in K_t} \sum_{e\in P} \ell_{e,t}$

6: Update the weight of the best path: $\omega_i(t+1)=\omega_i(t).\exp\left(\gamma\frac{g_{i,t}}{Kp_i(t)}\right)$

7 **end for**

The main reason for the implementation of this algorithm is its simplicity. However, for the shortest path problem the bound on R_n is unacceptably large because the number of paths N is exponentially large in the size of the graph. This dependence on the number of paths is

[8] In which there is no special structure of the set of actions that can be exploited.

probably the main reason for the slow convergence of the algorithm observed in our experiments.

In order to achieve a bound that does not grow exponentially with the number of edges of the graph, it is imperative to make use of the following fact: when the latencies of the edges of the chosen paths are revealed, then this also provides some information about the latency of each path sharing common edges with chosen paths. This fact was exploited in [35] in order to design an algorithm that achieves a per-round regret roughly of the order $K\sqrt{|E|lnN/n}$, where |E| is the number of edges of the graph, K is an upper bound on the lengths of the path, and N is the total number of paths. As future work, we intend to implement this algorithm in our routing overlay and study experimentally its performance.

8 TEST AND VALIDATION PLATFORMS

We consider two complementary approaches for the test and validation of our solution: a virtualized platform and a real-world platform. These platforms are presented below.

8.1 Virtualized platform: network emulation with CORE

Compared to a real-world platform, a virtualized network platform can greatly simplify the test and validation of the system as it enables to study its behaviour in a controlled environment. Two major approaches can be considered: discrete-event simulations and emulation. Since the simulation of the routing overlay will be performed as part of Task 4.2, we have opted for an emulation-based approach.

The emulation-based approach allows running existing programs, as if they were working on a real machine and a real network. A network emulator can be used as a platform for the development, the test and validation of system implementation in almost real conditions. In our experiments, we have used the Common Open Research Emulator (CORE), an open-source network emulator developed by Boeing' s Research and Technology division and supported, in part, by the US Naval Research Laboratory [13].

CORE consists of a GUI for drawing topologies of lightweight virtual machines (see Figure 19), emulating end hosts or networking devices (e.g. routers, switches, etc.) running Internet protocols (e.g. OSPF, RIP, BGP). CORE also provides Python modules for scripting network emulation, and useful functions for inspecting the status of virtual network elements and for running applications and creating traffic in the network. The user can save the network configuration to a file and open saved configuration files. Also, the simulation can be connected to a physical Ethernet port to be connected to other computers running CORE simulations or to live equipment. This is called distributed CORE.

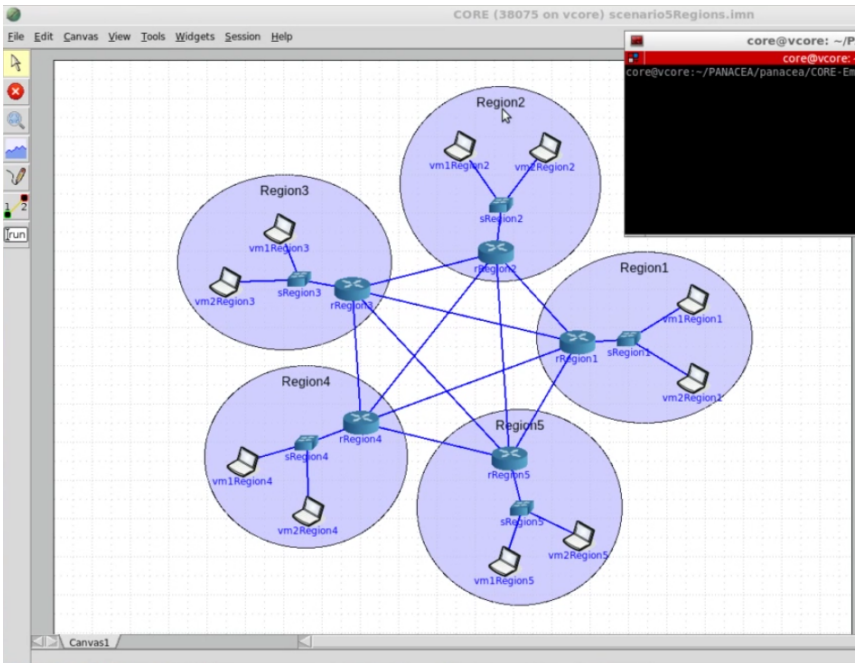

Figure 19:CORE Emulator GUI.

CORE uses Linux Containers (LXC) as its virtualization method [11]. LXC provide a light isolation of the emulated machines. Each machine operates in a separate container, which provides a more or less complete virtualization of the execution environment (processor, memory, network), but does not emulate the machine itself. CORE also handles the creation of the network between emulated machines, and allows real time modifications of link characteristics (delay, loss, capacity, packets duplication, etc.).

Sometimes, functions close to the system require modifying the host operating system kernel to run properly on virtualized machines. CORE developers provide patches to solve this problem. We had to use one of these patches to make packet interception functional on the emulated machines.

8.2 Real-world platform: Amazon EC2

Amazon Elastic Compute Cloud (EC2) [10] is a service allowing customers to rent virtual computers on which they can run their own applications. EC2 allows an easy deployment with a web interface and an API through which a customer can create, start, and stop server instances, on demand. The user can choose several hardware configurations and multiple operating systems. Users pay according to the time they use rented resources. The EC2 servers are located around the world in 8 data centres (see Figure 20). We have used these 8 data centres to deploy our routing overlay network and perform some experiments, as described in Section 9.3.2.

Figure 20: Amazon EC2 data centres location.

9 EXPERIMENTAL RESULTS

In this section, we focus on the functional validation of our implementation. We focus on the latency, which is main metric already implemented in our solution. We first start by introducing the client application used for our observations.

9.1 UDP-based ping

It is well-known that ICMP Echo, more familiar as" ping", is not a valid tool for measuring the performance and behaviour of the Internet. Ping is a tool of value mainly for determining whether connectivity exists or not, but it is a weak tool for measuring delay, variation in delay (jitter), or losses because ICMP is a lower-priority protocol. To validate that our routing overlay is able to discover minimum latency path in the overlay topology, we have to use an application measuring the delay from the user's point of view. We therefore implemented a measurement tool similar to the ping command, but using the UDP protocol instead of the ICMP protocol. We will use this application for our various experiments.

9.2 Overhead of the system

We present here the results concerning the time overhead introduced by the Panacea system compared to normal routing. For this matter, we measure the RTT using the application UDP ping that we developed, with and without activating the Panacea system. We use the CORE emulation platform with linear topologies of different sizes N = 2, ..., 5, as shown in *Figure 21*.

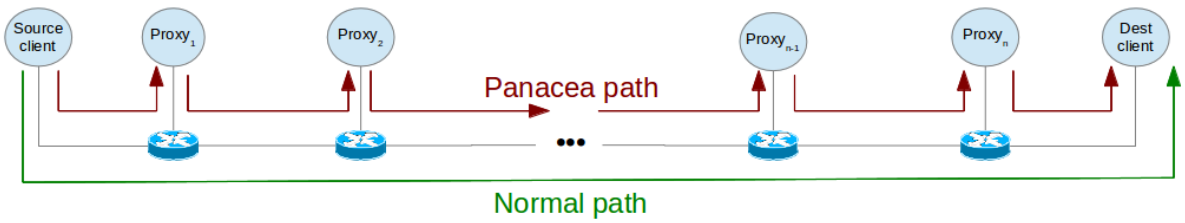

Figure 21: Topologies used for overhead estimation

We fix a zero delay on all the links in the topology. For each size, we measure the RTT with and without Panacea interception. When the interception is activated, we force the Panacea system to route the packet through all available proxies before reaching the destination. With these measures, we observed an additional latency of about 3ms (6ms per RTT) end-to-end from the normal route, regardless of the size of the topology (implying that the largest part of the time is spent in the interception/encapsulation phase).

Remark

To estimate the overall impact of the system, we must also take into account the latencies observed between the client machine and its local proxy. The algorithms that we propose

take into account the total overhead introduced by the system (time overhead as estimated above + client / proxy latency) to determine the best route.

9.3 Latency optimization

This section is devoted to the validation of the architecture and the implementation proposed to optimize the latency (which is the only metric now fully implemented in the system). We start with experiments with the CORE emulation platform and we then consider an experiment of 4.5 days on the Amazon EC2 platform. The results show that the proposed architecture and implementation are functional and effective in reducing the latency experienced by the user.

To observe the behaviour on a larger network, we finish this section with an observation of what can be achieved with the EXP3 algorithm on the NLNOG traces, via an offline simulation.

9.3.1 Validation with CORE

We consider a topology with 5 different sites, each with a proxy and a client machine running the agents in our system. We want to measure the RTT between the client machine in region 1 and the client machine in region 2.

We use our UDP ping application to measure the RTT by installing a server on the machine 2 and a client on the machine 1. In parallel, we are also executing a second client/server UDP ping couple, but running on a UDP port that is not intercepted by the transmission agents. We can thus compare the performance obtained with and without the overlay routing. Measurements are started automatically every minute on the two couples client / server. Latency measurements by proxies are also conducted every minute.

The experimental scenario is described in *Figure 22*. In the starting topology (step 1), all links have a latency of 25ms: the direct path is the shortest to reach the node 2 from the node 1. Five minutes later (step 2), the latency of the link 1-2 reaches 125ms, and an overlay routing with two hops is used. Steps 3 and 4 occur 5 and 10 minutes later, respectively, and forces the system to use a path of length 3 and then 4.

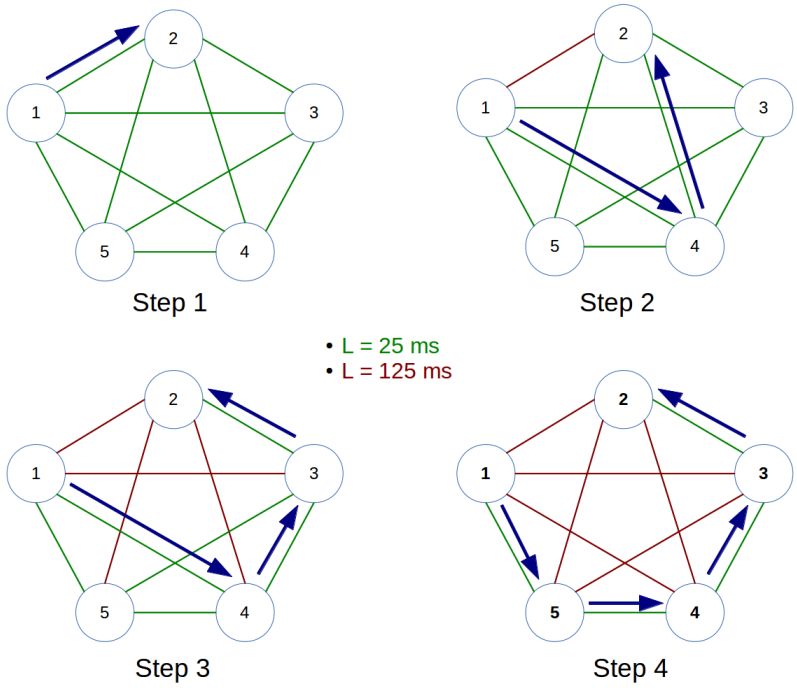

Figure 22: Test scenario used on CORE

The RTT measurements obtained are presented in *Figure 23*. We can observe the 4 levels corresponding to the 4 steps of the scenario. After the first latency increase, we see that our system redirect the traffic on a lower latency path compared to the direct IP route. On this scenario, the learning algorithm EXP3 does not always manage to find the optimal path due to the limited number of steps between each event. Nonetheless, we see that this algorithm can significantly reduce latency compared to direct IP routes.

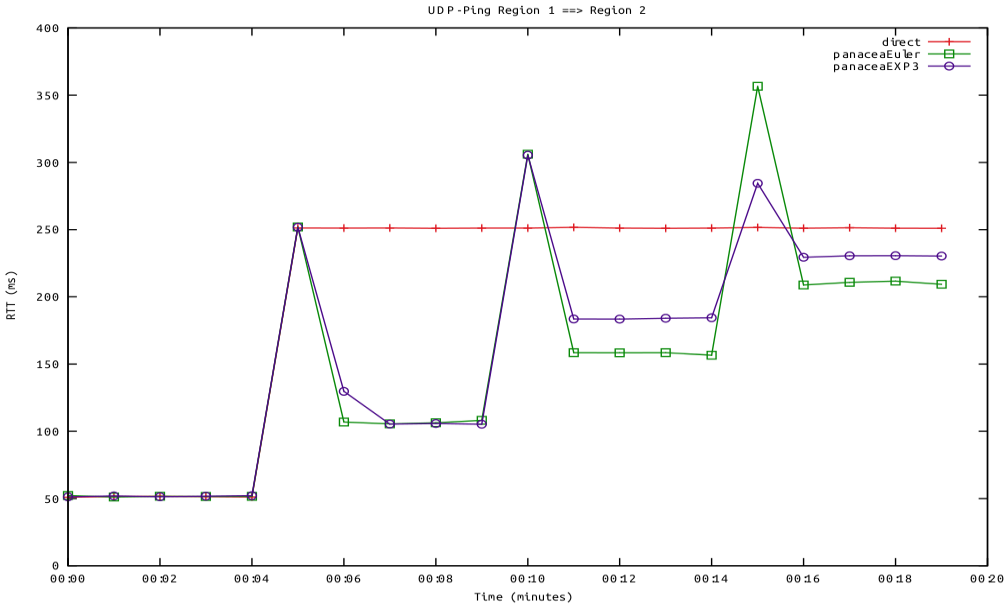

Figure 23: Latency optimization on CORE for the proposed scenario

These results validate the overall system, from packet interception to packet routing, but the scenario remains limited. Therefore we will now introduce tests in real situations made on the Amazon EC2 platform.

9.3.2 Validation with Amazon EC2

We present below the experimental results obtained on the 8 sites offered by the Amazon platform. Like the CORE experiment, we use two pairs of client/server UDP ping. In this experiment, we measure the RTT for each of the 56 origin / destination pairs, to observe the overall behaviour across the whole overlay network. Of course, we do not control here the events that can occur in the network. The RTT is measured every two minutes, for a total period of 4.5 days. On this platform, we have only tested the exhaustive algorithm (Euler). For this topology size, the number of links to be measured remains reasonable (56 links).

We present in *Figure 24* a plot representative of the results that can be obtained when the Internet latency is particularly suboptimal. We observe that the overlay network has significantly reduced the delay experienced between the cities of Tokyo and SaoPaulo[9]. On the contrary, when it is not possible to improve the latency, we could verify that our system did not introduce any additional cost thanks to the adaptive packet interception.

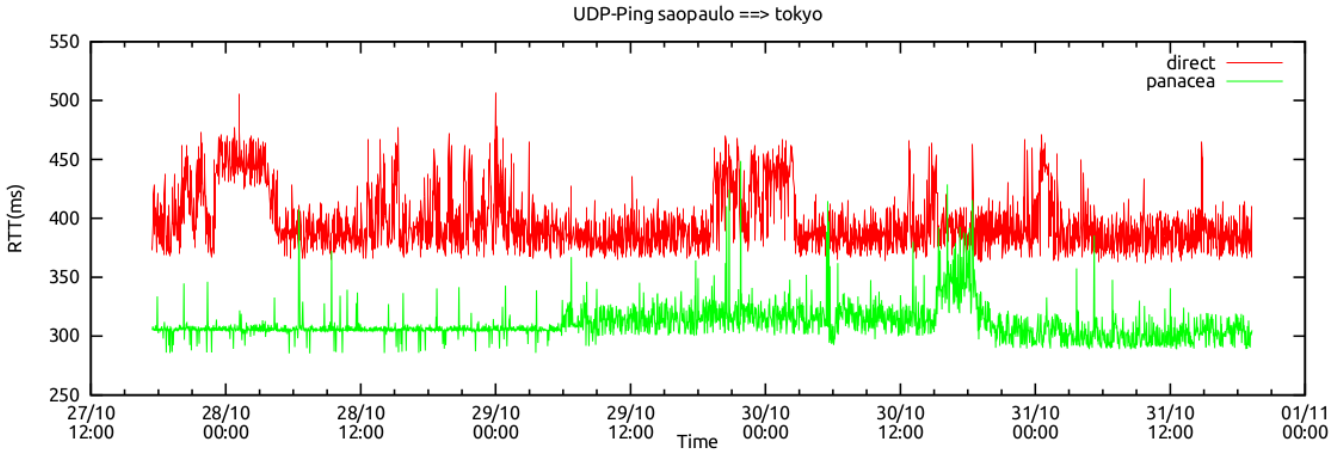

Figure 24: Latency optimization between SaoPaulo and Tokyo in the Amazon EC2 platform.

9.3.3 Validation with the NLNog traces

In this section, we observe the behaviour of the system on a larger topology. We use the traces obtained on the NLNOG network, which consists of 20 nodes (see **Figure 1**). The

9We noticed that the latencies of path passing through the Pacific Ocean could be significantly reduced with our system.

number of links to be measured here becomes larger (190 bidirectional links), and justifies the use of an algorithm minimizing the measurement effort.

We conduct offline simulations reproducing the system and its algorithms, using NLNOG network latency data as input. The simulations take into account the additional one-way delay introduced by the overlay, that we overestimated here to 5ms. We seek to study the EXP3 algorithm performance compared to the direct Internet routing and to the exhaustive algorithm (Euler). The average results obtained with the EXP3 algorithm are summarized in Table 2. We considered the case of 2, 5 and 10 randomly chosen paths at each round. As a path consists of at most L_{max}=4 links, we know that the EXP3 algorithm will measure at most 8, 20 or 40 links per round for these three configurations (to be compared to the 190 links).

	Direct	2 draws	5 draws	10 draws
Non optimal instants (%)	22.27	22.11	18.45	15.2
Average gap to optimal latency(%)	9.7	8.98	7.97	6.62

Table 2:Average behaviour of the EXP3 algorithm on the whole NLNOG traces compared to the exhaustive algorithm (Euler)

We first observe that for 22.27 % of cases[10], the direct path is less interesting than the overlay path provided by the Eulerian algorithm. We can then observe the influence of the number of draws of the EXP3 algorithm: we clearly see that increasing the number of draws enables to get closer to the optimal routing.

These average values do not truly measure the gains obtained in the pathological routing situations we seek to improve. We present in Table 3 some examples for which the system provides an interesting gain, despite the partial coverage of the graph.

	Direct	2 draws	5 draws	10 draws
Singapor-Israel	19.58	15.84	11.08	4.25
Japan-Chile	48.96	39.20	8.97	14.66
Australia-Chile	20.86	13.73	7.28	4.18
Norway-Singapour	17.41	9.85	5.56	1.89
Poland-Brazil	12.05	11.67	6.63	2.90
Ireland-Moscow	77.78	59.30	32.49	16.05
Israel-Moscow	37.69	29.60	13.76	5.27

Table 3: Gap (%) w.r.t.the optimal latency (Euler) for some suboptimal Internet cases.

10 This value is lower than the 38% mentioned in the introduction, because the implemented routing algorithms take into account the total overhead introduced by the system (10 ms per RTT: 2 * 3 = 6 ms for overhead + an arbitrary value of 4 * 1 ms for the client / proxy latencies) to choose the best route.

These examples show the influence of the number of random draws on the obtained results. For this topology of N=20 nodes, we can see a clear improvement compared to the direct Internet routing when the number of random draws is greater than or equal to 5 (N/4 for this topology).

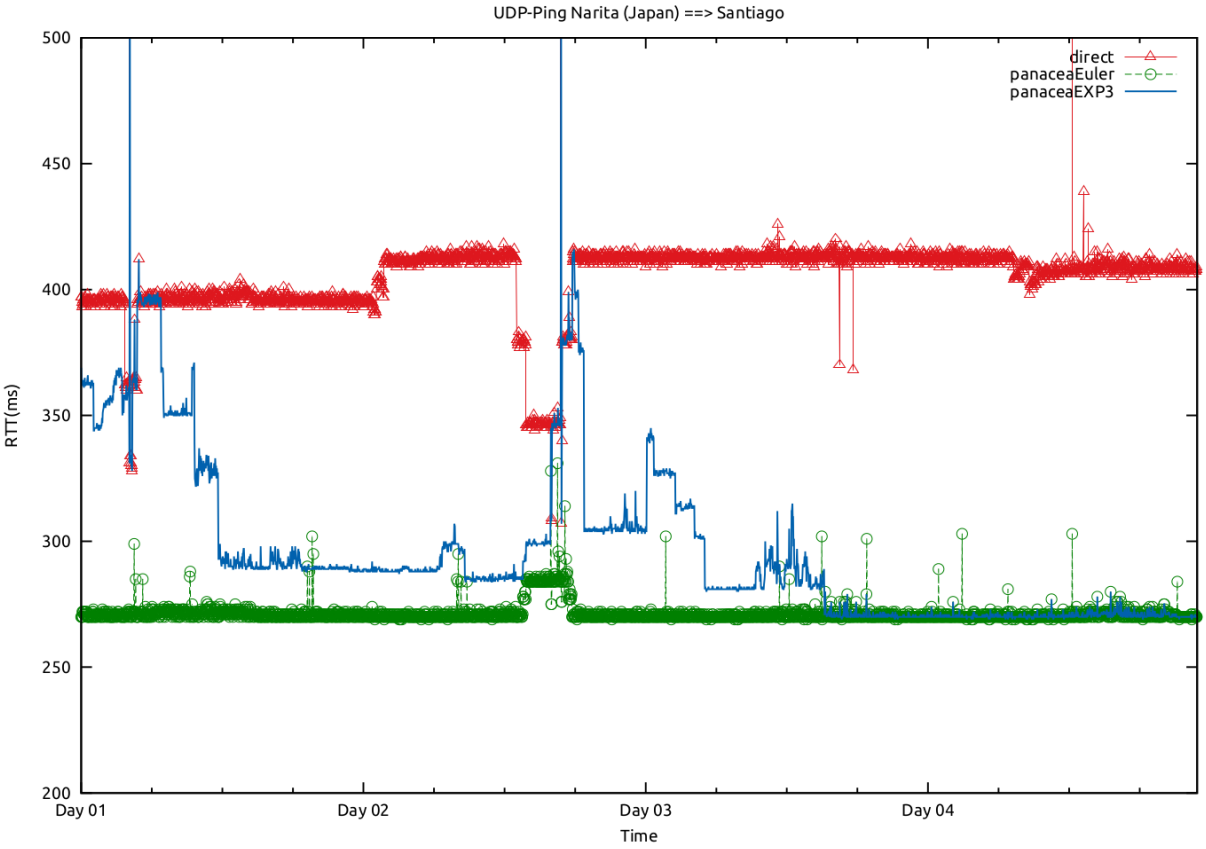

Figure 25: NLNOG link Japan-Chile, 5 draws for EXP3 algorithm

Figure 25 presents the example of the Japan-Chile link when using 5 random draws in our extension of the EXP3 algorithm. We see that the algorithm converges slowly to the optimal route, even when some network events occur in the middle of the observed week.

10 CONCLUSIONS

We presented in this document the design objectives, the architecture and the implementation of the PANACEA communication system. This system is a self-healing, self-optimizing and highly scalable routing overlay that accepts customized routing policies formulated by distributed applications according to their own needs. Overlay nodes monitor the quality of Internet paths (latency, bandwidth, loss rate) between themselves and are able to quickly detect and recover from path outages. They use learning techniques to discover the optimal paths between themselves with a minimum monitoring effort.

At this stage, we already have an alpha implementation of the routing overlay and some preliminary results have been obtained. Future work include the integration of the TOPP method for estimating the available bandwidth over a path in the overlay, as well as the implementation of a learning algorithm achieving a lower regret bound than EXP3. It is expected than in a few months our system will be ready to be used for the whole validation of the PANACEA technologies through the implementation of use cases 1 and 2.

11 REFERENCES

[1] PANACEA D5.1: Market analysis, business models and value chain. Feb. 2015

[2] A. G. Ganek, T. A. Corbi: "The dawning of the autonomic computing era", IBM Systems Journal 42 (1): 5-18, 2003. http://dx.doi.org/10.1147/sj.421.0005

[3] PANACEA D2.1: Principles of pervasive monitoring. Apr. 2014

[4] PANACEA D2.2: Design of a pervasive monitor. Dec. 2014

[5] ETSI: "Network Functions Virtualization – Introductory White Paper," Oct. 2012

[6] PANACEA D3.1: Implementation of a virtualization framework at a node level of the overlay, based on open source software and developed in the project tools, for realizing the machine learning framework. Oct. 2014

[7] PANACEA D3.2: Machine learning framework and global architecture for proactive management. Dec. 2015

[8] Software-defined networking: The new norm for networks. White paper. Open Networking Foundation, April 13 2012.

[9] Open dove. https://wiki.opendaylight.org/view/Open_DOVE:Main, 2013.

[10] Amazon ec2. http://aws.amazon.com/fr/ec2/, 2014.

[11] Lxc containers. https://linuxcontainers.org/lxc/introduction/, 2014.

[12] Netfilter/iptables. http://www.netfilter.org/, 2014.

[13] J. Ahrenholz, C. Danilov, T.R. Henderson, and J.H. Kim. Core: A real-time network emulator. In Military Communications Conference, 2008. MILCOM 2008. IEEE, pages 1-7, Nov 2008.

[14] David Andersen, Hari Balakrishnan, Frans Kaashoek, and Robert Morris. Resilient overlay networks. In Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles, SOSP '01, pages 131-145, New York, NY, USA, 2001. ACM.

[15] K. Andreev, B. M. Maggs, A. Meyerson, and R. Sitaraman. Designing overlay multicast networks for streaming. In Proceedings of the Fifteenth Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA), San Diego, CA, USA, June 2003.

[16] J-Y Audibert and S. Bubeck. Regrets bounds and minimax policies under partial monitoring. Journal of Machine Learning Research, 11:2785-2836, 2010.

[17] P. Auer, N. Cesa-Bianchi, Y. Freund, and R. Schapire. The non-stochastic multi-armed bandit problem. SIAM Journal on Computing, 32(1):48-77, 2002.

[18] S. Banerjee, B. Bhattacharjee, C. Kommareddy, and G. Varghese. Scalable application layer multicast. In Proc. of the ACM SIGCOMM, New York, USA, 2002.

[19] M. Beck, T. Moore, and J.S. Plank. An end-to-end approach to globally scalable programmable networking. In ACM Press, editor, in Proceedings of the ACM SIGCOMM workshop on Future directions in network architecture, 2003.

[20] D.A. Berry and B. Fristedt. Bandit problems: Sequential allocation of experiments.

Monographs on Statistics and Applied Probability. Chapman & Hal, London, 1985.

[21] Olivier Brun, Josu Doncel, and Christopher Thraves Caro. Shortest path discovery problem, revisited (query ratio, upper and lower bounds). Research report, LAAS-CNRS, October 2014.

[22] Y.H. Chu, S.G. Rao, and H. Zhang. A case for end system multicast. In ACM, editor, ACM SIGMETRICS 2000, pages 1-12, Santa Clara, CA, June 2000.

[23] Andy Collins. The detour framework for packet rerouting. Technical report, 1998.

[24] Mike Dahlin, Bharat Chandra, Let Gao, and Amol Nayate. End-to-end wan service availability. In In Proc. 3rd USITS, pages 97-108, 2001.

[25] L. Euler. Solutio problematis ad geometriam situs pertinentis. Commentarii academiae scientiarum Petropolitanae 8, pages 128-140, 1741.

[26] N. Feamster, H. Balakrishnan, J. Rexford, A. Shaikh, and J. van der Merwe. The case for separating routing from routers. In ACM Press, editor, Proceedings of the ACM SIGCOMM workshop on Future directions in network architecture, 2004.

[27] A. Feldmann, O. Maennel, Z. Morley Mao, A. Berger, and B. Maggs. Locating internet routing instabilities. In Proceedings of the ACM SIGCOMM 2004 Conference (SIGCOMM), Portland, Oregon, USA, August 2004.

[28] Fleury. Deux problèmes de géometrie de situation. Journal de Mathématiques élémentaires, pages 257-261, 1883.

[29] E. Gelenbe and Z. Kazhmaganbetova. Cognitive packet network for bilateral asymmetric connections. IEEE Trans. Industrial Informatics, 10(3):1717-1725, 2014.

[30] E. Gelenbe, R. Lent, A. Montuori, and Z. Xu. Towards networks with cognitive packets. In Proc. 8th Int. Symp. Modeling, Analysis and Simulation of Computer and Telecommunication Systems (IEEE MASCOTS), San Francisco, CA, USA, pages pp 3-12, August 29-September 1 2000.

[31] M. Gellman. QoS Routing for Real-time Traffic. PhD thesis, Imperial College London, 2007.

[32] E. Gelenbe, Cognitive Packet Networks, US Patent 6804201 B1, 2004

[33] J.C. Gittins. Multi-armed bandit allocation indices. Wiley-Interscience Series in Systems and Optimization. John Wiley & Sons, Chichester, 1989.

[34] J. Gross. Programmable networking with open vswitch. LinuxCon - http://events.linuxfoundation.org/sites/events/files/slides/OVS-LinuxCon%202013.pdf, September 2013.

[35] A. Gyorgy, T. Linder, G. Lugosi, and G. Ottucsak. The on-line shortest path problem under partial monitoring. Journal of Machine Learning Research, 8:2369-2403, 2007.

[36] J. Han and F. Jahanian. Impact of path diversity on multi-homed and overlay networks. In In Proceedings of IEEE International Conference on Dependable Systems and Networks, 2004.

[37] Ningning Hu, Student Member, Peter Steenkiste, and Senior Member. Evaluation and characterization of available bandwidth probing techniques. IEEE Journal on Selected

Areas in Communications, 21:879-894, 2003.

[38] Teerawat Issariyakul and Ekram Hossain. Introduction to Network Simulator NS2. Springer Publishing Company, Incorporated, 1 edition, 2008.

[39] C. Labovitz, R. Malan, and F. Jahanian. Internet routing instability. IEEE/ACM Transactions on Networking, 6(5):515-526, 1998.

[40] Craig Labovitz, Abha Ahuja, Abhijit Bose, and Farnam Jahanian. Delayed internet routing convergence. SIGCOMM Comput. Commun. Rev., 30(4):175-187, August 2000.

[41] T. Leighton. Improving performance on the internet. Communications of the ACM, 52(2), February 2009.

[42] J. Liebeherr and T. K. Beam. Hypercast: A protocol for maintaining multicast group members in a logical hypercube topology. In Proceedings of the First International COST264 Workshop on Networked Group Communication, pages 72-89. Springer-Verlag, 1999.

[43] J. Moy. RFC 7348: Virtual eXtensible Local Area Network (VXLAN): A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks. Technical report, 2014.

[44] E. Nygren, R. K. Sitaraman, and J. Sun. The akamai network: A platform for high-performance internet applications. ACM SIGOPS Operating Systems Review, 44(3), July 2010.

[45] Jay Painter and Ethan A. Merritt. Optimal description of a protein structure in terms of multiple groups undergoing TLS motion. Acta Crystallographica Section D, 62(4):439-450, Apr 2006.

[46] V. Paxson. End-to-end routing behavior in the internet. In in Proc. ACM SIGCOMM'96, pages 25-38, Stanford, CA, USA, August 1996.

[47] D. Pendarakis, S. Shi, D. Verma, and M. Waldvogel. Almi: An application level multicast infrastructure. In Proc of the 3rd USNIX Symposium on Internet Technologies and Systems (USITS), San Francisco, CA, USA, March 2001.

[48] L. Peterson, S. Shenker, and J. Turner. Overcoming the internet impasse through virtualization. In in Proceedings of the 3rd ACM Workshop on Hot Topics in Networks (HotNets-III), November 2004.

[49] H. Rahul, M. Kasbekar, R. Sitaraman, and A. Berger. Towards realizing the performance and availability benefits of a global overlay network. In Passive and Active Measurement Conference, Adelaide, Australia, March 2006.

[50] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In the Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001), 2001.

[51] Stefan Savage, Andy Collins, Eric Hoffman, John Snell, and Thomas Anderson. The end-to-end effects of internet path selection. SIGCOMM Comput. Commun. Rev., 29(4):289-299, August 1999.

[52] Yevgeny Seldin, Csaba Szepesvári, Peter Auer, Montanuniversität Leoben, Yasin Abbasi-yadkori, Peter Deisenroth, Csaba Szepesvári, and Jan Peters. Evaluation and analysis of the performance of the exp3 algorithm in stochastic environments.

[53] R. K. Sitaraman, M. Kasbekar, W. Lichtenstein, and M. Jain. Overlay Networks: An Akamai Perspective. In Advanced Content Delivery, Streaming, and Cloud Services. John Wiley & Sons, 2014.

[54] I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In SIGCOMM'01, San Diego, California, USA., August 27-31 2001.

[55] R. Stone. Centertrack: An ip overlay network for tracking dos floods. In in Proc. USENIX Security Symposium '00, August 2000.

[56] Csaba Szepesvári. Shortest path discovery problems: A framework, algorithms and experimental results. In AAAI, pages 550-555, 2004.

[57] Geoff Huston Tony Bates, Philip Smith. Cidr report - http://www.cidr-report.org/as2.0/. Technical report, CIDR, 2015.

[58] J. Touch, Y. Wang, L. Eggert, and G. Finn. A virtual Internet architecture. Technical Report ISI-TR-2003-570, ISI, March 2003.

[59] J. Wang, L. Lu, and A.A. Chien. Tolerating denial-of-service attacks using overlay networks - impact of overlay network topology. In Proc. First ACM Workshop on Survivable and Self-Regenerative Systems, 2003.

[60] B. Y. Zhao, L. Huang, J. Stribling, S.C. Rhea, A.D. Joseph, and J.D. Kubiatowicz. Tapestry: A resilient global-scale overlay for service deployment. IEEE Journal on Selected Areas in Communications, 2003.

[61]Matthew Mathis, Jeffrey Semke, JamshidMahdavi, and TeunisOtt. The macroscopic behavior of the tcp congestion avoidance algorithm. SIGCOMM Comput. Commun. Rev., 27(3) :67–82, July 1997.

[62] Bob Mel, Mats Björkman, and Per Gunningberg. Regression-based available bandwidth measurements. In In International Symposium on Performance Evaluation of Computer and Telecommunications Systems, 2002.

[63]B. Melander, M. Bjorkman, and P. Gunningberg. A new end-to-end probing and analysis method for estimating bandwidth bottlenecks.In Global Telecommunications Conference, 2000.GLOBECOM ' 00. IEEE, volume 1, pages 415–420 vol.1,2000.

[64] Ningning Hu, Student Member, Peter Steenkiste, and Senior Member. Evaluation and characterization of available bandwidth probing techniques. IEEE Journal on Selected Areas in Communications, 21 :879–894, 2003.

[65] R. K. Ahuja, T. Magnanti and J. B. Orlin, Network Flows, Prentice Hall, 1993.

[66] Csaba Szepesvari, Shortest Path Discovery Problem: A Framework, Algorithms and Experimental results, AAAI ,2004.

[67] D. Avresky and N. Natchev, Dynamic Reconfiguration in Computer Clusters with Irregular Topologies in the Presence of Multiple Node and Link Failures, IEEE Transactions on Computers, vol.54, no. 5, pp. 603-615, May 2005, doi:10.1109/TC.2005.76

