Scalable, Self-Healing, and Self-Optimizing Routing Overlays

Olivier Brun, Hassan Hassan, Josselin Vallet

Scalable, Self-Healing, and Self-Optimizing Routing Overlays

Olivier Brun, Hassan Hassan and Josselin Vallet
LAAS-CNRS
Université de Toulouse
7 Avenue du Colonel Roche
31077 Toulouse, France.
email: {brun, hhassan, jvallet}@laas.fr

December 7, 2015

Abstract

After Internet routing was shown in a number of classic measurement papers to result in paths that are sub-optimal with respect to a number of metrics, routing overlays were proposed as a method for improving performance, without the need to re-engineer the underlying network. In this paper, we present SMART, a self-healing, self-optimizing and highly scalable routing overlay, which has a number of advantages with respect to existing solutions. First, SMART can run with off-the-shelf applications and does not require any kernel modification. In addition, SMART can be widely deployed over a sizable population of routers, because it can quickly learn and efficiently track the optimal path with a limited monitoring effort. We describe the design objectives, the architecture and the implementation of SMART, as well as the online decision methods used for learning the optimal routes. Experimental results demonstrate significant improvements over native IP routing, both in terms of latency and throughput.

1 Introduction

Current Internet routing protocols may work reasonably well when only "best effort" delivery is required, but the requirements of modern distributed services are typically far more stringent, demanding greater performance and availability of end-to-end routes than these protocols can natively deliver.

These services often require continuous operation over time, always maintaining the response time below an acceptable threshold, and even small degradations in their performance can have a considerable business impact, in terms of slowed-down service adoption, lost revenue or even damage to brand reputation.

A number of classic measurement studies (see, e.g., [1, 2]) have revealed that the performance of flows could be significantly improved by choosing alternate paths to the ones proposed by IP (Internet Protocol) routing protocols. In addition, it was also shown that path outages are routine events in the Internet, and that the inter-domain routing protocol BGP (Border Gateway Protocol) reacts and recovers slowly from link/node failures [3, 4, 5], causing path outages that can last for several tens of minutes [6, 2, 6, 7].

The ideal solution would be a complete rethink of the Internet routing infrastructure, doing away with the existing architecture and redesigning it with the benefit of hind-sight about its deficiencies. Unfortunately, the so-called ossification of the Internet prevents even changes that are unanimously recognized as necessary to take place.

Routing overlays have been proposed as an alternative solution that can potentially provide the desirable flexibility and control over the routing infrastructure, without the need to re-engineer the Internet [8, 9, 10, 11]. A routing overlay is formed of end hosts, which are deployed in different spots over the Internet. These nodes monitor the quality of the IP routes between themselves and use this information to decide whether to route packets directly over the IP route or by way of other overlay nodes. A routing overlay therefore enables controlling the path of data through the network without modifying the underlying IP mechanism for computing routes, but just by adding intermediate routing hops into the path taken by packets. 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 data traverse may be quite different.

There are several advantages to the use of routing overlays. Firstly, they can be used to quickly recover from path outages. Indeed, they can exploit the inherent redundancy of the Internet to find an alternate path when an IP route becomes unavailable, even if Internet routing protocols cannot. In addition, routing overlays can also be used to improve the quality of service of data flows by overriding the routes determined by Internet protocols and routing traffic based on metrics directly related to application performances.

In this paper, we present SMART[1], a self-healing, self-optimizing and

[1] Self-MAnaging Routing overlay

highly scalable routing overlay that we developed. SMART is self-healing because it is able to quickly detect and recover from path outages. It is self-optimizing because it can discover the optimal routes within the overlay network for service-specific routing metrics. It is highly scalable because it was designed to learn the optimal routes in large overlays with a minimum monitoring effort. Last but not least, SMART was designed to control the path of data of an application through the overlay without the application even being aware that its data flows are routed over the overlay, so that it can work with off-the-shelf applications.

In the following, we describe the design objectives, the architecture and the implementation of SMART. Since one of our essential design goals was to build a routing overlay that can be widely deployed over a sizable population of routers, we elaborate on the methods implemented in SMART for discovering optimal routes in large overlay networks with a minimum probing effort. Finally, we also present experimental results obtained with real-world experiments over the Internet. These results demonstrate that it is possible to significantly improve over native IP routing with a modest monitoring effort. Due to the lack of space, we do not present the methods used for assessing the quality of overlay links, but interested readers may refer to [12].

The rest of this paper is organized as follows. In Section 2, we discuss the similarities and differences of our routing overlay with existing solutions. Section 3 is devoted to the description of the architecture and components of our system. In Section 4, we describe the technical mechanisms used for forwarding a packet from its source to its destination. Section 5 presents the methods used for discovering optimal routes in the overlay with a minimum monitoring effort, whereas experimental results are presented in Section 6. Finally we conclude in Section 7 with a brief summary and a description of future work.

2 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 [13, 14, 15], to implement application-layer multicast [16, 17, 18, 19], and even to provide countermeasures to DDoS attacks [20, 21]. Overlay network technologies are also used by Akamai Inc. for dynamic content delivery [22, 23, 24, 25, 26]. Comprising

more than 61,000 servers located over 1,000 networks in 70 countries, 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, such as Virtual Extensible LAN (VXLAN) [27] and Distributed Overlay Virtual Ethernet (DOVE) [28]. The main difference between SMART and these technologies is that they 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 (Resilient Overlay Network) projects, which have clearly demonstrated the benefits of moving some of the control over routing into the hands of end-systems. The Detour framework [29] 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. In contrast, the developers of RON have opted for a tighter integration of the application and the overlay network since RON is a software library that programs link against [30]. This approach permits "pure application" overlays with no kernel modifications, and allows the use of application-defined routing metrics. Although the objectives of SMART are similar to those of Detour and RON, SMART has the advantage that it can work with off-the-shelf applications an on standard operating systems. Another major difference is that Detour and RON do not scale very well: as the number of overlay nodes n increases, their costly $O(n^2)$ probing overhead becomes a limiting factor. In practice, a reasonable RON overlay can support only about 50 routers before the probing overhead becomes overwhelming [30].

The latter design objective is shared with a self-aware routing protocol known as the Cognitive Packet Network (CPN) [31, 32]. 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. The application of CPN techniques to peer-to-peer overlay networks has been considered in [33]. More recently, the use of CPN-inspired learning techniques in SMART was investigated in [34]. In the present paper, we describe in much more details the architecture and implementation of SMART, and investigate the relevance of a different approach for learning the optimal overlay routes. In addition, whereas only results related to the round trip delay were reported in [34], we present here some experimental results on bandwidth optimization.

3 Architecture of the Routing Overlay

The overlay network is formed of software routers scattered over the Internet. In our experiment, these routers were executed in Virtual Machines (VM) running in cloud computing platforms, but they can be ran on physical hosts as well.

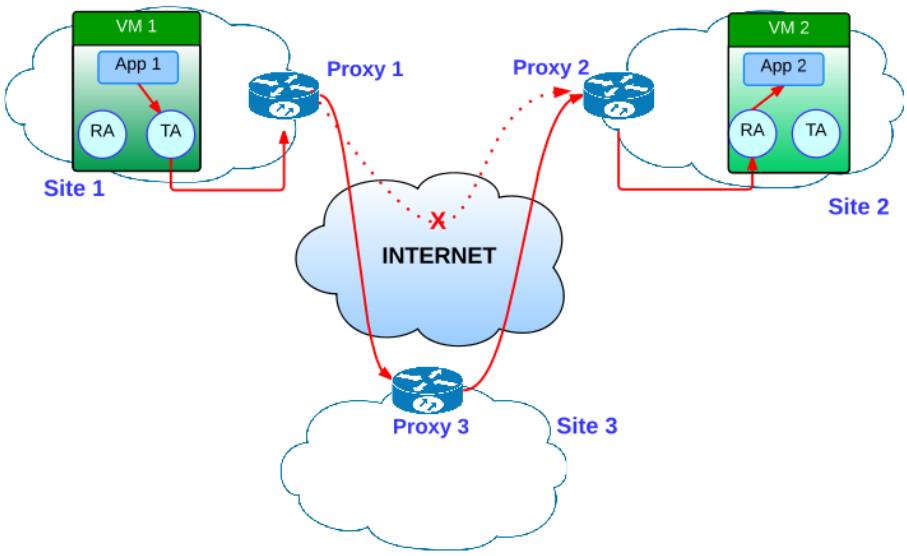

Figure 1: Architecture of the Autonomic Communication Overlay.

Two types of agents are used. Transmission (TA) and Reception (RA) agents are local agents that are executed on each VM running a task of the distributed application. They represent the entry and exit points of the overlay network, respectively. Each site also runs a single 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 of the application over them. As shown in Figure 1, this enables to avoid congested or failed parts of the Internet when a Proxy detects that the IP route is subject to anomalies.

3.1 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.
- **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.

3.2 Proxy

An agent, called a Proxy, is executed in each site and acts as an intermediary for communications with other sites. The Proxy is in fact an entity constituted of three different software agents:

- **Monitoring agent**: it monitors the quality of the Internet paths between the local site and the other sites 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.
- **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 (cf. Section 5). 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 is in charge of forwarding each incoming packet to its destination on the path it was instructed to use by the routing agent.

4 Packet interception, encapsulation and forwarding

In our architecture, the forwarding of a packet from its source to its destination proceeds as shown in Figure 2.

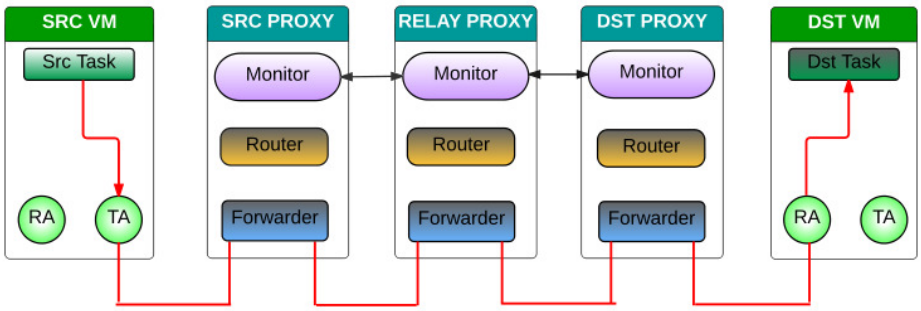

Figure 2: Forwarding process.

When a packet is sent by a source task to a destination task located in a different site, 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, referred to as the SMART packet in the following, is that of the original packet along with an additional header. Upon reception of the SMART packet, the forwarding 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 5). The sequence of intermediate Proxies is written in the SMART header, and then the SMART 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 SMART 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.

4.1 Packet interception

The TA intercepts the packets sent by the application running in the same VM and forwards 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 site. 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.

As shown in Figure 3, packet interception is realized using a filtering mechanism known as NetFilter NFQUEUE. Netfilter represents a set of hooks inside the Linux kernel [35]. It allows specific kernel modules to register functions that 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).

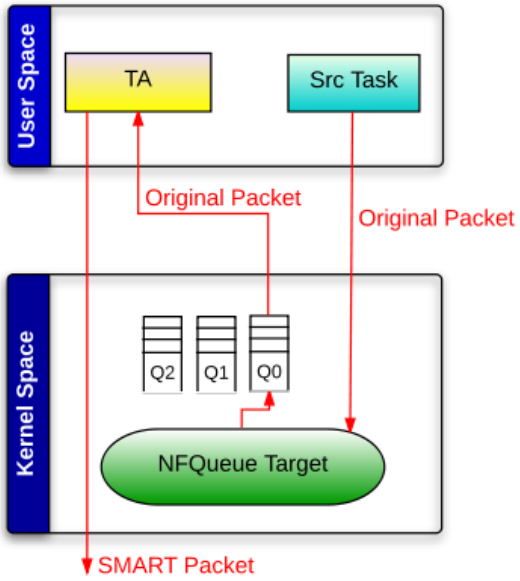

Figure 3: Forwarding process.

4.2 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 SMART header that contains control information. 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, that is, the list of intermediate Proxies. 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. Once the header added, the SMART packet is sent to the local proxy using UDP.

4.3 Processing by the forwarding agent

Upon receipt of a SMART packet, the forwarding agent inspects its header to determine its precise role. There are three cases:

1. 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 header of the SMART packet, and then forwards it to the next hop on the path.

2. 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.

3. The packet has reached the destination Proxy: the forwarding agent then forwards the packet to the RA on the destination VM.

4.4 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. The packet is directly delivered to the recipient application because the destination IP address is that of the destination VM. We note that there is an additional difficulty when the public IP address of the destination VM is different from its private IP address. In that case, the automatic remapping of public IP addresses into private IP addresses by the Network Address Translation (NAT) mechanism is only possible for the SMART packet, and not for the inner original packet. 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.

4.5 Packet forwarding overhead

In order to evaluate the time overhead introduced by SMART with respect to native IP routing in a controlled environment, we have used the Common Open Research Emulator (CORE). CORE is an open-source network emulator developed by Boeings Research and Technology division and supported, in part, by the US Naval Research Laboratory [36]. It consists of a GUI for drawing topologies of lightweight virtual machines, emulating end hosts or networking devices (e.g. routers, switches, etc.) running Internet protocols. We have used CORE to emulate linear topologies of different sizes $n = 2, \ldots, 5$ as shown in Figure 4.

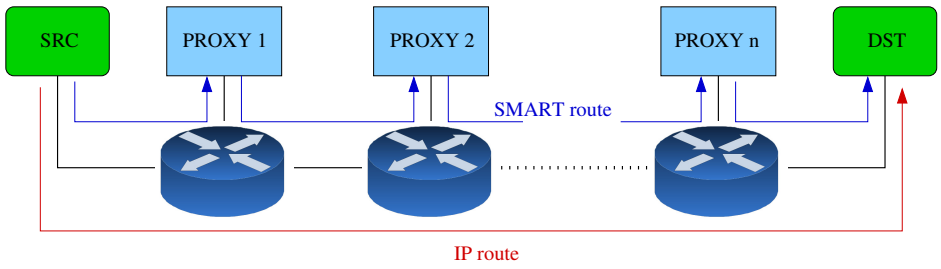

Figure 4: Experiment to evaluate SMART forwarding overhead.

For each size, we have measured the end-to-end RTT with and without SMART. When SMART is activated, it routes all packets through all available proxies before reaching the destination. We observed an additional end-to-end latency of about 3ms with respect to native IP routing, regardless of the size n of the topology (indicating that most of the overhead is due to the processing done by the TA and RA).

5 Discovering the Optimal Routes

In this section, we assume for simplicity that there is a single origin/destination (OD) pair and describe the algorithm implemented by the source Proxy for learning an optimal route to the destination Proxy. This algorithm is implemented by the Routing Agent of the source Proxy. We assume that at discrete time steps (say, every minute) the routing algorithm measures the quality of some links and uses this information to decide how to route packets between the source and destination nodes. We define the monitoring effort of the routing algorithm as the number of probed links per time slot. As mentioned in Section 2, existing routing overlays use all-pairs probing,

which has the advantage that it is guaranteed that an optimal path is discovered; but the downside is that this approach does not scale very well due to its costly $O(n^2)$ monitoring effort in an overlay of n nodes. Since we wish to build a routing overlay that can be widely deployed over a sizable population of routers, instead of requiring an optimal path to be found 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 performance as the best path. The idea is to design an algorithm that exploits past observations so as to quickly learn and efficiently track the optimal path.

We formulate this problem as a multi-armed bandit problem [37] in which decisions correspond to paths between the source and the destination, and consider it in the adversarial setting where path costs can change arbitrarily from one time step to the other. In this setting, no probabilistic assumption is made regarding the costs of overlay paths, and in particular there is no independence assumption made on these costs. To solve this adversarial bandit problem, we use an algorithm directly inspired from the well-known EXP3 algorithm [38]. At each successive time slot, it chooses a subset of paths to probe, and measures the quality of these paths (e.g., by summing the edge delays if the metric to be optimized is the latency). The algorithm then sends its packet over the minimum-cost path among those it has probed. In other words, probing does not cover *all* possible paths but only a few paths which have been observed in previous probing steps to provide the best performance. However, we have to widen our probing at random over other paths, so that we do not miss out on paths whose quality has substantially improved over recent history. We first give some background information below on the adversarial multi-armed bandit problem, and then present the routing algorithm implemented in SMART.

5.1 Adversarial Multi-armed Bandit Problem

We represent the overlay network by a complete graph G of n nodes, and we let s and d be the source and destination nodes, respectively. A decision algorithm $\mathcal{A}$ for the multi-armed bandit problem is given as input N paths in G from s to d, indexed from 1 to N. For example, these paths may correspond to the paths of at most two hops between s and d (that is, the direct link and all paths with exactly one intermediate node), in which case $N = n - 1$. The cost of a path i (e.g., its latency, or the inverse of its throughput) may vary arbitrarily over time, but it is assumed to be upper bounded by some constant $\Delta > 0$. At round $t = 1, 2, \ldots$, a cost $\ell_i(t) \in [0, \Delta]$

is assigned to each path i, but it is not revealed to the algorithm. Then, the algorithm chooses a path $i(t) \in \{1, 2, \ldots, N\}$, sends a message over this path and observes its cost $\ell_{i(t)}(t)$. The cumulative cost of the algorithm over T rounds is defined as

$$L_T(\mathcal{A}) = \sum_{t=1}^{T} \ell_{i(t)}(t), \tag{1}$$

whereas the cumulative cost of path i over the T rounds is $L_T(i) = \sum_{t=1}^{T} \ell_i(t)$. The normalized regret of the algorithm $\mathcal{A}$ with respect to the best path is then

$$R_T(\mathcal{A}) = \frac{1}{T}\left(L_T(\mathcal{A}) - \min_{i=1,\ldots,N} L_T(i)\right). \tag{2}$$

The goal is then to design an algorithm $\mathcal{A}$ that perform asymptotically as well as the best path, i.e., such that $R_T(\mathcal{A})$ converges to 0 as T grows to infinity, uniformly over all outcomes sequences.

5.1.1 Online Decision Algorithm

In [38], Auer et al. gave a randomized algorithm to solve the adversarial multi-armed bandit problem. This algorithm is known as EXP3 and it is based on exponential weighting with a biased estimate of the gains (defined, in our case, as $g_i(t) = \Delta - \ell_i(t)$ for path i), combined with uniform exploration. The regret of this algorithm can be upper-bounded, for any $0 < \delta < 1$, and a fixed time horizon T, with probability at least $1 - \delta$, by

$$R_T(EXP3) \leq \frac{11\Delta}{2}\sqrt{\frac{N\log(N/\delta)}{T}} + \frac{K\log(N)}{2T}. \tag{3}$$

Note that the regret of this algorithm decreases in time according to $1/\sqrt{T}$. We have implemented in the routing agent a slightly modified version of the EXP3 algorithm, which is inspired from the "power of two choices" technique in randomized load-balancing [39]. In this version, precisely described in Algorithm 1, the routing algorithm chooses a subset $I(t) = \{i_1(t), \ldots, i_K(t)\}$ of paths to probe at each round. The path $i_1(t)$ is the IP route from s to d, and the other paths are chosen randomly according to a probability distribution $\mathbf{p}(t)$ that depends on the weights $w_1(t), \ldots, w_N(t)$ of the paths. This distribution is a mixture of the uniform distribution and a distribution which assigns to each path a probability mass exponential in the estimated cumulative gain for that path. Once the

paths probed by the Monitoring Agent, the algorithm selects the path $i^*(t)$ with the best performance among those in $I(t)$, and informs the Forwarding Agent that it has to use this path if $i^*(t) \neq i^*(t-1)$. Finally, the algorithm updates the weights of the paths.

Algorithm 1 Learning optimal paths with the EXP3 algorithm.

Parameters: integer $K \geq 1$; real $\gamma \in (0, 1]$.
Initialization: $w_i(1) = 1$, $i = 1, \ldots, N$.
for $t = 1,\ 2, \ldots$ **do**
    Compute the probability of each path:

$$p_i(t) = (1-\gamma)\frac{w_i(t)}{\sum_{j=1}^{N} w_j(t)} + \frac{\gamma}{N}$$

    Set $i_1(t)$ to the IP route and choose randomly paths $i_2(t), \ldots, i_K(t)$ according to $\mathbf{p}(t)$.
    Probe the paths $i \in I(t) = \{i_1(t), i_2(t), \ldots, i_K(t)\}$.
    Compute the gains $g_i(t) = \Delta - \ell_i(t)$ for $i \in I(t)$.
    Select the best path $i^*(t) = \arg\max_{i \in I(t)} g_i(t)$.
    Update the weights:

$$w_i(t+1) = \begin{cases} w_i(t)\, exp\left(\gamma\, \frac{g_i(t)}{N p_i(t)}\right) & i \in I(t), \\ w_i(t) & \text{otherwise.} \end{cases}$$

end for

It is easy to show that when $K > 1$ this algorithms performs at least as well as EXP3, so that its regret decreases at least as fast as $1/\sqrt{t}$. In practice, with $K = 3$, we often obtain negative values of the regret, indicating that the algorithm performs even better than the best fixed path. If we restrict ourselves to the $N = n - 1$ paths of at most two hops, then the monitoring effort of the algorithm is $2K - 1$, independently of the size of the overlay network. More generally, with m OD pairs the monitoring effort is $m(2K - 1)$, which is less than in the all-pairs probing approach as long as $m < \frac{n(n-1)}{2K-1}$.

6 Experimental Results

6.1 Latency minimization

We now describe the results that were obtained with the proposed algorithm during an Internet-scale experiment done in spring 2014, where we used 19 nodes of the *NLNog* ring[2] shown in Figure 5. Note that these overlay nodes are interconnected by literally hundreds of Internet nodes which are unknown to us or the overlay, and which support the overlay itself.

We first measured the latency between all pairs of nodes every two minutes, communicating through the Internet, for a period of one week using the ICMP-based ping utility. Furthermore, when five consecutive packets were lost between a specific pair of nodes, we considered that the particular source was disconnected from that destination. We thus collected some 1.7×10^6 measurement data over the week, from which we can compute the weighted adjacency matrix of the overlay graph at each measurement epoch, and hence compare the round trip delay of the IP route with that of the optimal overlay route.

Figure 5: Geographical location of the 20 nodes selected in the NLNog ring.

The analysis of collected data confirmed the deficiencies of Internet routing observed in previous studies. There was an outage of the IP route at least once in the week for 65% of OD pairs, and 21% of these outages lasted more than 4 minutes (and more than 14 minutes for 11% of them). This analysis also revealed that, as shown in Figure 6, in 50% of the cases it

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

is possible to improve over the latency of the IP route by adding one or more intermediate overlay nodes to the path. Surprisingly enough, in 30% of the cases, the minimum latency path is a path with only one intermediate overlay node, that is, a two-hop path. This shows that a limited deviation from IP actually produces much better QoS than IP itself. Interestingly, even though in 20% of the cases the optimal path is a 3 or 4 overlay-hop path, there is on the average no significant gain (only 5.4%) in considering overlay paths of more than two hops. This suggests that we can restrict ourselves to paths with at most one intermediate overlay node (this is true only on average, since, for instance, the RTT between Narita/Paris can be more than halved if we use two intermediate nodes instead of at most one).

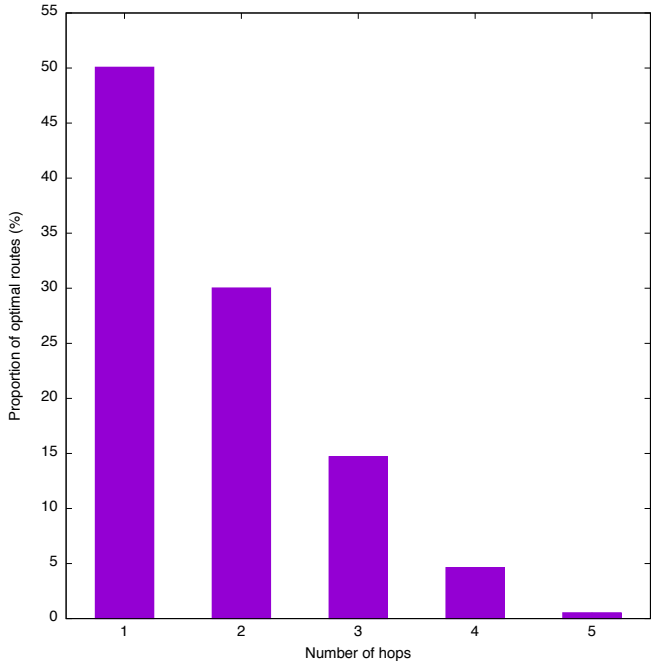

Figure 6: Percentage of instances when the optimal path includes 1, 2, 3 or 4 hops.

As we will now show, SMART allows a significant decrease in round-trip delay, with a very modest monitoring and computational effort. We consider a fixed OD pair, the other overlay nodes serving just as relays. We restrict ourselves to the $N = 18$ overlay paths of at most two hops and assume that the routing algorithm probes $K = 3$ paths (including the direct IP route) at each time slot, that is, every two minutes. The algorithm therefore

Table 1: Performance of native IP and SMART routings on the whole set of NLNOG traces compared to optimal two-hop routing.

	IP route	SMART
Non optimal instants (%)	44.5	3.8
Gap to optimal latency(%)	14.4	0.39

Table 2: Average RTT (ms) for some pathological OD pairs.

	IP route	SMART	OPT 2-hops
Melbourne/Gibraltar	390	274.7	273.5
Narita/Santiago	406.7	254.5	253.0
Moscow/Dublin	179.9	81.9	80.8
Honk Kong/Calgary	267.1	131.8	130.0
Singapore/Paris	322.3	154.9	153.2
Tokyo/Haifa	322.6	180.8	180.1

measures 5 links per measurement and decision round (to be compared to the 342 links monitored in the all-pairs probing approach). Our results are summarized in Table 1, which shows the average relative gap to the minimum latency that can be achieved with two-hop routing (the averaging is over time and over the 342 OD pairs). These results demonstrate that SMART uses the optimal two-hop route in 96% of the cases, and that it provides near-optimal latencies, with a clear improvement over native IP routing (13.8% on average). However, these average values do not truly measure the gains obtained in the pathological routing situations we seek to improve. In Table 2, we present the results for some OD pairs, for which our system allows a huge decrease in round-trip delay.

On the other hand, Figure 7 shows the RTT between Narita (Japan) and Santiago (Chile) over 5 successive days. The RTT of the direct IP route is about 400 ms, whereas the RTT of the minimum latency path is about 250 ms. As can be seen, SMART learns quickly which is the minimum latency path and tracks this path until the end of the 5 days. Figure 8 shows the same results over the first 3 hours. We notice that it takes only 25 measurement epochs (50 minutes) for SMART to learn the optimal route.

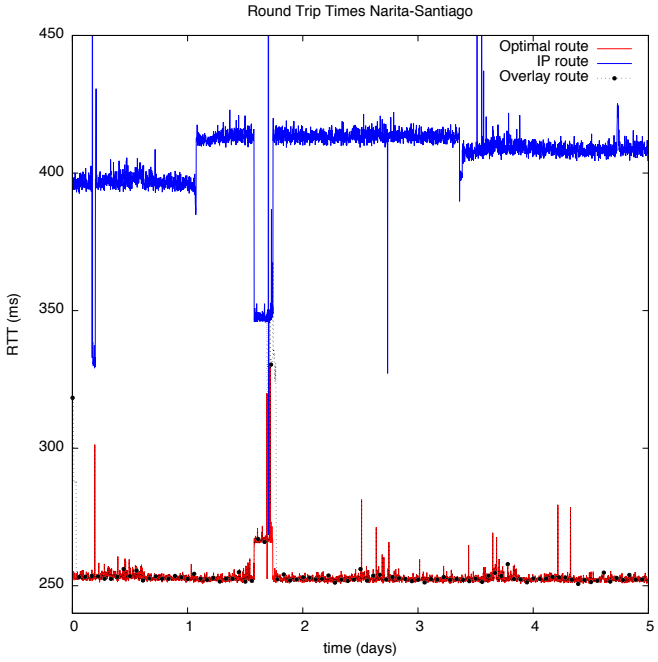

Figure 7: RTT (ms) measured for the Narita(Japan)-Santiago(Chile) connection in an experiment lasting 5 consecutive days.

6.2 Throughput maximization

We now describe the results obtained in an experiment involving 9 AWS (Amazon Web Services) data centres located as shown in Figure 9. In summer 2015, we measured the available throughput between all pairs of data centres every five minutes, communicating through the Internet, for a period of four days. We thus collected some 8.3×10^4 measurement data over the 4 days period. Assuming that the available throughput over a path is the minimum of the throughputs of its constituent links, the analysis of these data revealed that the IP route is the maximum throughput route only in 23% of the cases, and that most of the time, the maximum throughput overlay route passes through 1 or 2 intermediate nodes (see Figure 10).

As in Section 6.1, we consider only the $N = 8$ overlay paths of at most two hops and take $K = 3$. The monitoring effort is therefore limited to 5 links, whereas the all-pair probing measures the throughput of 72 links at each measurement epoch. Our results are summarized in Table 3. As for the RTT, we observe a clear improvement over native IP routing. Here again,

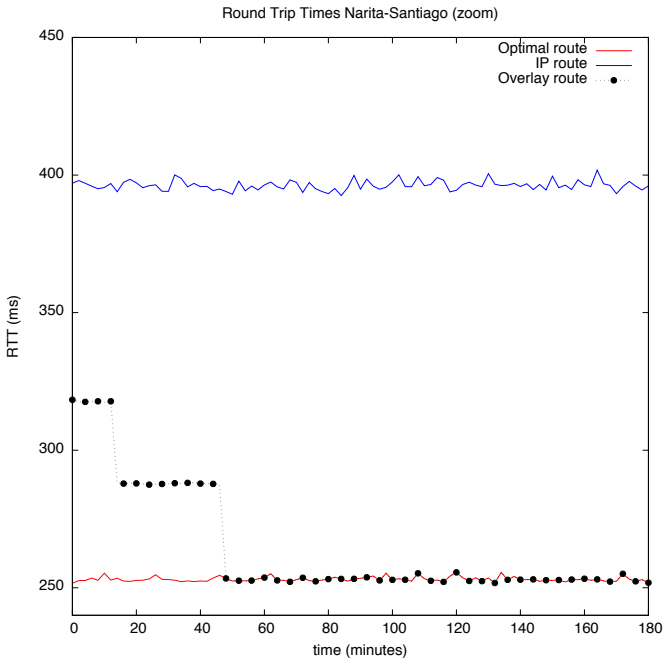

Figure 8: RTT ms for the Narita(Japan)-Santiago(Chile) connection over the first 3 hours of the experiment reported in Figure 7.

Figure 9: Geographical location of the 9 AWS data centres.

we present in Table 4 the results obtained for some pathological OD pairs, for which the available throughput is at least doubled.

On the other hand, Figure 11 shows the available throughput between Sydney (Australia) and Virginia (USA) over the 4 successive days. The

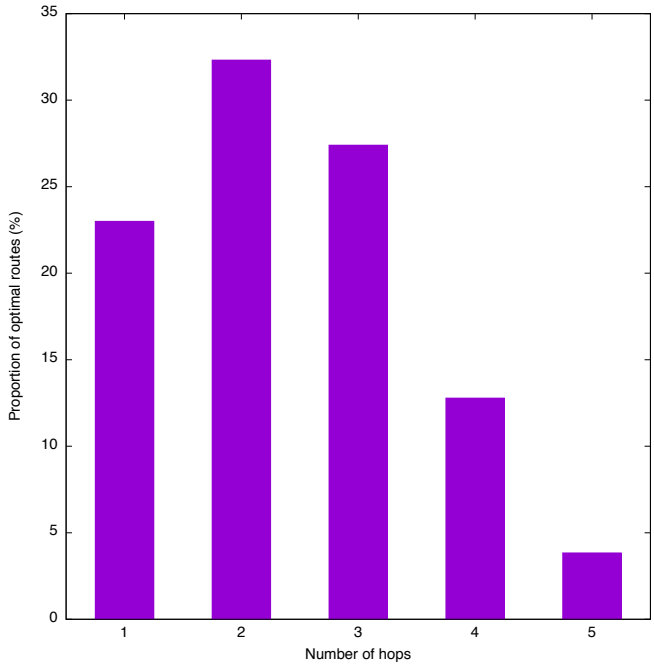

Figure 10: Percentage of instances when the optimal path includes 1, 2, 3, 4 or 5 hops.

average throughput of the direct IP route is 8.5 Mbps, whereas the average throughput of the optimal path is 55.3 Mbps. Figure 12 shows the same results over the first 3 hours. We notice that SMART discover an optimal routes almost immediately, but that it is less effective at tracking it than it was the case for the RTT.

Table 3: Performance of native IP and SMART routings on the whole set of AWS traces compared to optimal two-hop routing.

	IP route	SMART
Non optimal instants (%)	73.9	30.1
Gap to optimal latency(%)	31.3	6.6

Table 4: Average throughputs (Mbps) for some pathological OD pairs.

	IP route	SMART	OPT 2-hops
Dublin/Sydney	11.5	35.5	40.5
Singapore/Sao Paulo	12.8	39.5	43.6
Sydney/Virginia	8.5	50.7	55.3
Virginia/Singapore	7.4	31.2	36.1
Virginia/Sydney	6.9	32.2	36.7
Virginia/Tokyo	10.3	37.5	43.4

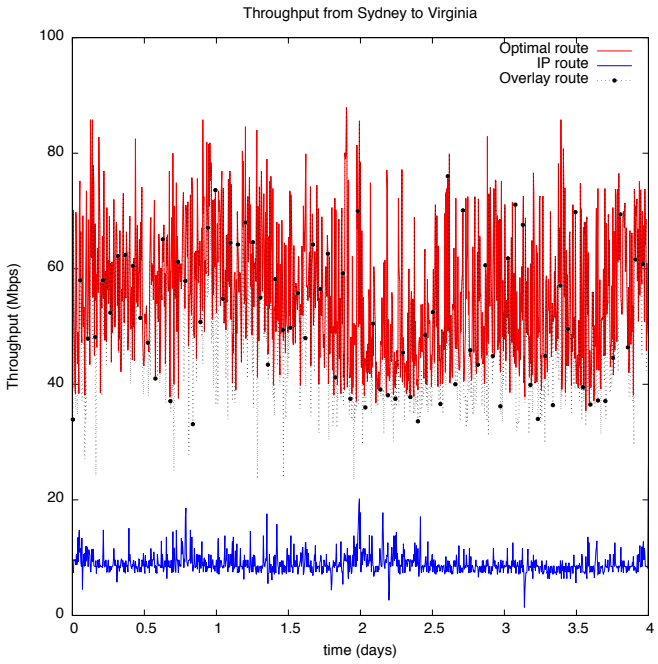

Figure 11: Throughput (Mbps) measured from Sydney (Australia) to Virginia (USA) over 4 consecutive days.

7 Conclusion

Internet routing works reasonably well most of the times. Yet, our experimental results show that a routing overlay that make measurement-based online routing decisions can yield spectacular improvements over native IP routing in some cases. The issue is that it is not possible to measure the qual-

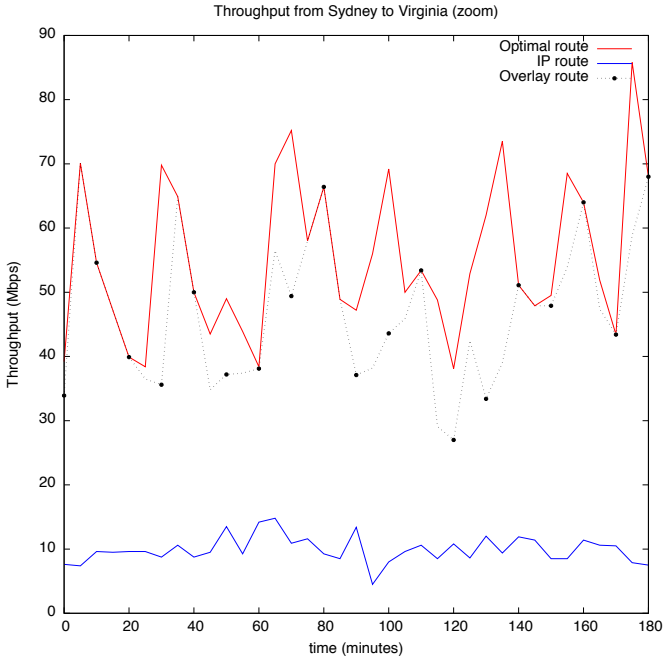

Figure 12: Throughput (Mbps) measured from Sydney (Australia) to Virginia (USA) over the first 3 hours of the experiment reported in Figure 11.

ity of all overlay links in large overlays, implying that a tradeoff between the quality of the routes discovered and the monitoring effort to discover them is required. To the extend of our knowledge, SMART is the first routing overlay to address this issue.

The results we have obtained have essentially considered paths of at most two overlay hops. Although considerable improvements over native IP routing have been demonstrated, this may not be sufficient for some source to destination pairs. In order to increase the number of potential overlay paths without impairing the convergence time of the learning algorithm, we plan to investigate a different approach based on the so-called *Online Shortest Path Problem* [40]. This approach makes use of the following crucial observation: when the latencies of the edges of some paths are measured, then this also provides some information about the latency of each path sharing common edges with probed paths. As future work, we intend to study experimentally the performance of this approach, as well as to investigate its generalization to non-additive metrics for the *Online Widest Path Problem*.

Acknowledgement

The research leading to these results has received funding from the European Community's Seventh Framework Programme [FP7/2007-2013] under the PANACEA Project (www.panacea-cloud.eu), grant agreement no 610764.

We wish to thank the administrators of the NLNog ring for providing us access to this platform.

References

[1] S. Savage, A. Collins, E. Hoffman, J. Snell, and T. Anderson, "The end-to-end effects of internet path selection," *SIGCOMM Comput. Commun. Rev.*, vol. 29, no. 4, pp. 289–299, Aug. 1999. [Online]. Available: http://doi.acm.org/10.1145/316194.316233

[2] V. Paxson, "End-to-end routing behavior in the internet," in *in Proc. ACM SIGCOMM'96*, Stanford, CA, USA, August 1996, pp. 25–38.

[3] C. Labovitz, A. Ahuja, A. Bose, and F. Jahanian, "Delayed internet routing convergence," *SIGCOMM Comput. Commun. Rev.*, vol. 30, no. 4, pp. 175–187, Aug. 2000. [Online]. Available: http://doi.acm.org/10.1145/347057.347428

[4] M. Dahlin, B. Chandra, L. Gao, and A. Nayate, "End-to-end wan service availability," in *In Proc. 3rd USITS*, 2001, pp. 97–108.

[5] 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.

[6] C. Labovitz, R. Malan, and F. Jahanian, "Internet routing instability," *IEEE/ACM Transactions on Networking*, vol. 6, no. 5, pp. 515–526, 1998.

[7] A. Feldmann, O. Maennel, Z. M. Mao, A. Berger, and B. Maggs, "Locating internet routing instabilities," in *Proceedings of the ACM SIGCOMM 2004 Conference (SIGCOMM), Portland, Oregon, USA*, August 2004.

[8] 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.

[9] J. Touch, Y. Wang, L. Eggert, and G. Finn, "A virtual internet architecture," ISI, Tech. Rep. ISI-TR-2003-570, March 2003.

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

[11] M. Beck, T. Moore, and J. Plank, "An end-to-end approach to globally scalable programmable networking," in *in Proceedings of the ACM SIGCOMM workshop on Future directions in network architecture*, A. Press, Ed., 2003.

[12] U. Ayesta, O. Brun, H. Hassan, and B. Prabhu, "D2.3 - autonomic communication overlay," Deliverable of the FP7 PANACEA project (www.panacea-cloud.eu), Tech. Rep., 2015.

[13] I. Stoica, R. Morris, D. Karger, M. 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.

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

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

[16] Y. Chu, S. Rao, and H. Zhang, "A case for end system multicast," in *ACM SIGMETRICS 2000*, ACM, Ed., Santa Clara, CA, June 2000, pp. 1–12.

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

[18] 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.

[19] 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*. Springer-Verlag, 1999, pp. 72–89.

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

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

[22] 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.

[23] 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.

[24] T. Leighton, "Improving performance on the internet," *Communications of the ACM*, vol. 52, no. 2, February 2009.

[25] E. Nygren, R. K. Sitaraman, and J. Sun., "The akamai network: A platform for high-performance internet applications," *ACM SIGOPS Operating Systems Review*, vol. 44, no. 3, July 2010.

[26] R. K. Sitaraman, M. Kasbekar, W. Lichtenstein, and M. Jain, *Overlay Networks: An Akamai Perspective*, ser. In Advanced Content Delivery, Streaming, and Cloud Services, E. Pathan, Sitaraman, and Robinson, Eds. John Wiley & Sons, 2014.

[27] J. Moy, "RFC 7348: Virtual eXtensible Local Area Network (VXLAN): A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks," Tech. Rep., 2014. [Online]. Available: https://tools.ietf.org/html/rfc7348

[28] R. Moats, "Open dove," https://wiki.opendaylight.org/view/Open_DOVE:Main, 2013.

[29] A. Collins, "The detour framework for packet rerouting," Tech. Rep., 1998.

[30] D. Andersen, H. Balakrishnan, F. Kaashoek, and R. Morris, "Resilient overlay networks," in *Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles*, ser. SOSP '01. New York, NY, USA: ACM, 2001, pp. 131–145. [Online]. Available: http://doi.acm.org/10.1145/502034.502048

[31] 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*, August 29-September 1 2000, pp. pp 3–12.

[32] E. Gelenbe and Z. Kazhmaganbetova, "Cognitive packet network for bilateral asymmetric connections," *IEEE Trans. Industrial Informatics*, vol. 10, no. 3, pp. 1717–1725, 2014.

[33] M. Gellman, "Qos routing for real-time traffic," Ph.D. dissertation, Imperial College London, 2007.

[34] O. Brun, L. Wang, and E. Gelenbe, "Data driven smart intercontinental overlay networks," 2015, submitted to IEEE Jour. on Selected Areas in Communications.

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

[36] J. Ahrenholz, C. Danilov, T. Henderson, and J. Kim, "Core: A real-time network emulator," in *In IEEE Military Communications Conference (MILCOM 2008)*, November 2008, pp. 1–7.

[37] N. Cesa-Bianchi and G. Lugosi, *Prediction, Learning and Games*. Cambridge University Press, 2006.

[38] P. Auer, N. Cesa-Bianchi, Y. Freund, and R. Schapire, "The non-stochastic multi-armed bandit problem," *SIAM Journal on Computing*, vol. 32, no. 1, pp. 48–77, 2002.

[39] M. D. Mitzenmacher, "The power of two choices in randomized load balancing," Ph.D. dissertation, University of California at Berkeley, 1991.

[40] A. Gyorgy, T. Linder, G. Lugosi, and G. Ottucsak, "The on-line shortest path problem under partial monitoring," *Journal of Machine Learning Research*, vol. 8, pp. 2369–2403, 2007.