Invariant Preserving Middlebox Traversal

Ahmed Abujoda and Panagiotis Papadimitriou

Institute of Communications Technology, Leibniz Universität Hannover, Germany
{first.last}@ikt.uni-hannover.de

Abstract. Middleboxes, such as firewalls, NATs, proxies, and application accelerators are known for their undesirable implications on traffic (mainly due to packet headers or paylod modifications) and for hindering connection establishment when certain protocols are in use (e.g., UDP, SCTP).
Since many of these implications occur in middleboxes within ISPs or cellular networks, we present a software-defined network (SDN) architecture that can foster the collaboration between end-hosts and ISPs. In particular, an end-host can express a desirable behavior from the network, specified as an invariant (*e.g.,* no IP header or payload modification), and the ISP, in turn, can establish a connection through middleboxes that preserve this invariant. We discuss the proposed architecture and the requirements for invariant preserving middlebox traversal. We further propose an algorithm for the selection of the best path through a sequence of invariant-preserving middleboxes. We use simulations to assess the efficiency of our approach.

1 Introduction

The increasing demand for security and access control along with the need for better application support has led to the deployment of network appliances, known as middleboxes, by enterprises, Internet Service Providers (ISP), and cellular network operators. The proliferation of middleboxes, such as firewalls, proxies, network address translators (NAT), intrusion detection systems (IDS), and redundancy elimination boxes, has been reported in recent studies [11, 9].

Unfortunately, the additional functionality that middleboxes embed comes at a cost: middleboxes introduce various undesirable implications on traffic. For example, NATs rewrite IP addresses and ports, proxies break end-to-end semantics, firewalls may block UDP traffic or cache out-of-order-packets introducing varying delays, while application optimizers can modify the packet payload [10, 9]. Furthermore, the deployment of firewalls and NATs along most Internet paths may hinder connection establishment with protocols such as Stream Control Transmission Protocol (SCTP) or Multi-Path TCP [8]. To mitigate these problems, most applications resort to tunneling, *e.g.,* non-HTTP traffic may be tunnelled over HTTP to traverse firewalls, SCTP usually has to be tunnelled over TCP (or over UDP in case it is not blocked). Furthermore, traffic may be encrypted at the client device (*e.g.,* using HTTPS) to inhibit payload modifications by application optimizers [10]. However, this increases power consumption in mobile devices.

Most of these implications stem from the middleboxes deployed by access ISPs and cellular networks. To obviate the need for tunneling or traffic encryption for middlebox traversal, we consider fostering the collaboration between end-hosts and ISPs. More precisely, an end-host can express requirements for the establishment of a certain type of connection, *e.g.,* do not modify packet fields or paylod, permit UDP traffic or access to public DNS servers. Such requirements can be specified in the form of invariants (*e.g.,* using the API in [10]). Upon the submission of such a request, the ISP may be willing to redirect the traffic through a set of middleboxes (*e.g.,* NAT and firewall) that comply with his security policy and, at the same time, preserve the invariant expressed by the end-host. This can be offered to ISP clients as a value-added service, which may be appealing to a wide range of users (*e.g.,* home network users, mobile users, enterprises) that currently experience limitations in the applications or services they can run.

Establishing connections through a sequence of invariant-preserving middleboxes raises several requirements: (i) the collection of middlebox configurations, (ii) parsing and checking middlebox configurations against requested invariants, (iii) the selection of invariant-preserving middleboxes and shortest paths, and (iv) the insertion of forwarding entries in the ISP's routers to route the traffic through the assigned path. To this end, we present a software-defined network (SDN) architecture for invariant preserving middlebox traversal. Following the trend for (logically) centralized control, we rely on a centralized controller deployed by the ISP, which retrieves middlebox configurations, selects middleboxes and paths that preserve the specified invariant, and configures packet forwarding along the selected path. Middlebox checking against invariants can be performed using recent advances on static analysis, such as Header Space Analysis (HSA) [6] or SymNet [7]. For the installation of flow entries to routers, we employ OpenFlow [5]. Our work is mainly focused on middlebox and path selection. To this end, we present and evaluate an algorithm for the selection of a path through a set of invariant-preserving middleboxes. Our simulation results show that our approach increases substantially the number of established connections, especially under low and moderate levels of network utilization.

The remainder of the paper is organized as follows. In Section 2, we review and discuss the different implications of middleboxes on traffic and connection establishment. Section 3 provides an overview of our SDN architecture. In Section 4 we discuss our algorithm for invariant preserving path selection. In Section 5, we present our simulation environment and results. Finally, in Section 6, we highlight our conclusions.

2 Middleboxes implications

In this section, we discuss the implications of widely used middleboxes on traffic and connection establishment:

- **NATs:** due to the limited size of IPv4 address space, NATs have become one of most popular middleboxes on ISP networks, especially on cellular networks [9]. They enable the sharing of a public IP address among multiple hosts with private IP addresses by mapping the private IP address and the source port number of a host connection (TCP or UDP) to the public IP address and a selected port number. As a result, hosts sitting behind NATs are not visible to the outside world, *i.e.,* establishing connections with NATed hosts (*e.g.,* a VoIP, P2P applications) requires complicated NAT traversal techniques [14–16] and might need the participation of a third party (*e.g.* a relay [17]). However, even with NAT traversal techniques, connection establishment could fail due to the ISPs policies and configurations. As shown in [9], to perform load balancing cellular network operators may assign multiple NATs to a single device. Subsequently, this hinders NAT traversal techniques that depend on learning the NAT's public IP address by establishing multiple connections with the NAT (since different connections are handled by different NATs). Furthermore, operators might configure NATs to assign random port numbers to the mapped connections which hampers applications (*e.g.,* P2P) performing NAT traversal by trying to infer the mapped NAT port number [9].

- **Firewalls:** they are essential to today's network functionality by providing protection against malicious traffic as well as untrusted and policy-violating accesses. However, despite their importance, firewalls have become an obstacle hindering not only the deployment of new protocols and extensions (SCTP, ECN), but also restricting the connectivity of the traditional protocols. More particular, recent studies have shown that applications and protocols are being forced to tunnel over HTTP/HTTPs to bypass firewalls [10] [9]. Even when connections are successfully established, firewalls introduce further implications such as buffering out-of-order packets which impairs the functionality and performance of TCP connections, and terminating long-lived flows due to short timeouts on firewalls, leading to increased power consumption and service disruption [9].

- **Proxies:** they perform several functions to optimize the performance of particular applications or protocols such as caching contents, data compression and TCP connections splitting. Proxies are usually implemented with a specific application in mind which impairs the functionality of new applications passing through them. For example, mobile devices have to tunnel over HTTPS to avoid HTTP optimizers breaking their protocol semantics by modifying their packets payload or by sending a cached reply instead of forwarding the packet to the end server[10].

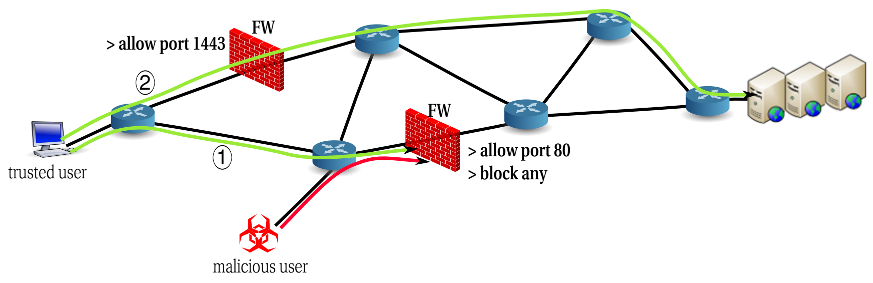

Fig. 1. Example of invariant preserving middleboxes traversal.

3 Architecture overview

In this section, we discuss the requirements of invariant preserving middlebox traversal and present the components of our SDN architecture.

Consider the example in Fig. 1, where a trusted user (*e.g.,* an enterprise with well-established relation/contract with the ISP) is trying to access a server through port 1443. Since the firewall on its default path (path 1) blocks any traffic on ports other than port 80, the user fails to establish a connection with the server. A straightforward solution is to request the ISP to reconfigure the firewall on the default path. However this will allow the malicious user's traffic to traverse the network. An alternative solution, which we consider in this paper, is to allow the user to express her requirement as an invariant to the ISP (in this case allow port 1443), and in turn the ISP identifies a path which preserve the invariant and does not violate the the ISP policy (path 2).

To this end, we envision an SDN architecture where a centralized control plane provides invariant preserving routing and redirection through the ISP network. Accordingly, we assume the deployment of OpenFlow switches which serve as the data plane of the network. Furthermore, as in today's ISP network, a set of middleboxes are deployed in the network at different locations to provide services such as protection against malicious traffic (e.g. firewalls), enable the sharing of IPv4 addresses (NAT) and caching of frequently used content (proxies). To provide invariant preserving connection establishment, our SDN architecture needs to fulfil a set of requirements:

- **efficient resource management:** we consider two objectives for resource management in ISP networks: (i) delay minimization, where an ISP aims at routing traffic through the path with the shortest delay and (ii) load balancing, where an ISP aims at balancing the traffic load across the network.

- **correctness:** traffic should traverse paths that preserve the invariant while not violating the ISP policy, *e.g.,* a video flow with a particular port number might be redirected through a firewall which grants access to it but still needs to keep an upper bound on the BW consumed by this flow. This requires correct and efficient parsing and checking of the state and configuration of the middleboxes deployed in the network.

- **traffic redirection:** the controller should be able to install forwarding entries in the ISP's switches to reroute traffic through the selected path. This should also take into account middleboxes which modify the packets routing header fields such as the IPs addresses (*e.g.,* NATs, load balancer).

To fulfil these requirements, we design a control plane which consists of four components (Fig. 2):

- **MBs configuration and state collection**: this component collects and stores the state and configurations (*e.g.,* firewall rules) of each middlebox deployed in the network. It accesses middleboxes through interfaces exposed to the controllers by the vendors. These interfaces could be vendor-specific (*e.g.,* CISCO CLI) or standard interface such as netconf [3] or SIMCO [4].
- **static checking**: this component implements tools such as SymNet [7] or HSA [6] to parse and analyse the state and configurations of middleboxes against the requested invariants. It basically identifies the implications the middleboxes have on the flow and hence, specifies the middleboxes which do not violate the flow invariant.
- **network monitoring:** this component keeps track of the network topology as well as the network links and middleboxes utilization. It reads the counters of the network switches deployed on the network using OpenFlow [5]. For monitoring middleboxes utilization, it uses again the interface exposed by the vendors.
- **path selection:** based on the output provided by the static checking and the network monitoring component, this component selects a path which fulfils the invariant of the connection while considering the utilization of the network and the middleboxes. It implements our path selection algorithm presented in Section 4.
- **switch configuration:** it installs the required flow entries in OpenFlow switches to redirect the flow through the path selected by path selection component. For middleboxes (e.g NAT) which modify some of the flow's 5 tuples (source and destination IP addresses, source and destination port numbers, protocol), the flow can be identified by adding tags to each packet such as in [12] and [13].

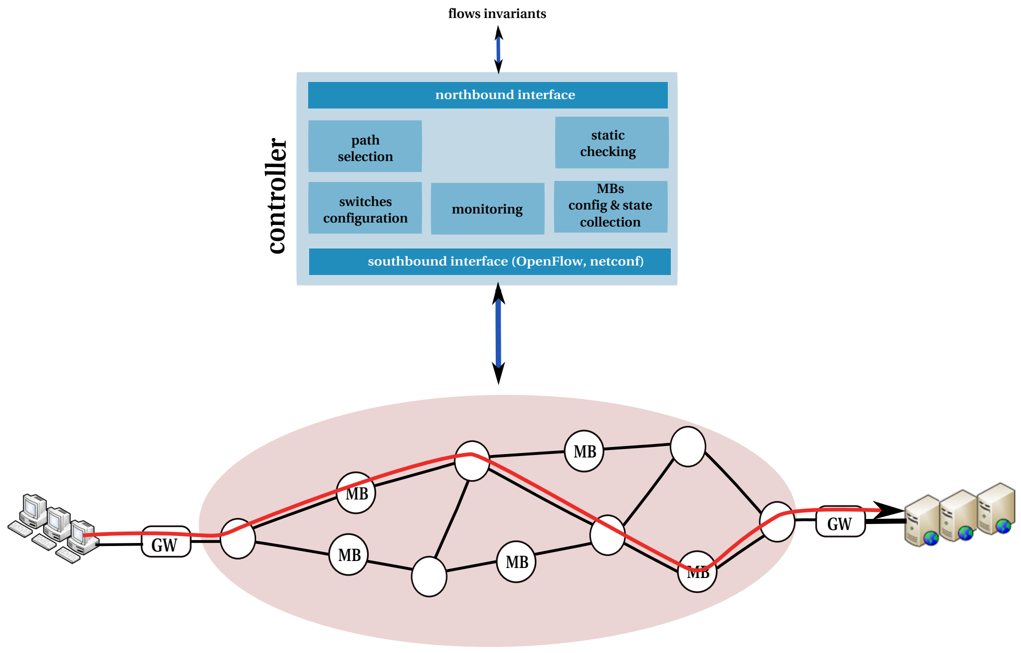

Fig. 2. Architecture components.

4 Path Selection

We develop an algorithm which selects a network path traversing middleboxes that preserve a connection invariant. In addition to the configuration and the state of each middlebox, our algorithm takes into account the available bandwidth on each network link as well as the available processing capacity of each middlebox. The algorithm is executed by an SDN controller which has the knowledge of the network topology and utilization, the middleboxes utilization and state, and the connection invariant. We present two variants of our algorithm: the first aims at minimizing end-to-end delay, whereas the second strives to achieve load balancing across the network.

We represent the ISP network as a weighted undirected graph $G = (N, L)$, where N is the set of nodes and L is the set of links between nodes of the set N. Nodes are classified into set of routers R and a set of middleboxes M such that $N = R \cup M$. Each m_i has a processing capacity which is denoted by $CP(m_i)$ and a state $S(m_i)$. Each link $l_{ij} \in L$ between two nodes n_i and n_j is associated with the available bandwidth $C(l_{ij})$. Let P_{ij} represents the set of paths in the network G, between the pair of nodes n_i and n_j. The available bandwidth $C(p)$ of a path $p \in P_{ij}$ is given by the minimum residual bandwidth of the links along the path:

$$C(p) = \min_{l_{ij} \in p} C(l_{ij}) \tag{1}$$

We further represent a connection demand with a vector $d = \{n_{src}, n_{dst}, r, cmp, v\}$, where $n_{src}, n_{dst} \in N$ denote the connection source and destination nodes, r represents the traffic rate, cmp is the required computing capacity to process the traffic, and v is the connection invariant.

Algorithm 1 Path selection

```
Inputs: $G = (N, L), d$

for each $l \in L$ do
    if $C(l) < r$ then
        $delete\ l\ from\ G$
    end if
end for

$P_{ij} \leftarrow$ FIND_ALL_PATHS($n_{src}, n_{dst}$) // all paths between source and destination
SORT($Pi, j$) // sort paths based on their length or available BW

for each $p \in P_{i,j}$ do
    $found \leftarrow true$
    for each $m \in P$
        if $cmp > CP(m)$ or $v \cap S(m) = \emptyset$ then
            $Found \leftarrow false$
            break
        end if
    end for
    if $found$ then
        return $p$
    end if
end for
return $\emptyset$ // no path was found
```

The algorithm selects a path between the source and destination of a connection. It starts by removing all the links with insufficient available bandwidth to fulfil the rate of the connection. This step reduces the size of the graph that FIND_ALL_PATHS function has to process to calculate all paths between the source and the destination. This function implements the algorithm in [2] which has a complexity of $O(N + L)$. Reducing the size of N and L results in lower runtime. After identifying all the paths, we sort them in increasing order based on the number of hops or on a decreasing order based on the available bandwidth $C(p)$. Sorting based on number of hops results in delay minimization, whereas sorting based on the available bandwidth achieves load balancing. We term the two algorithm variants as *invariant preserving SP algorithm* and *invariant preserving LB algorithm*, respectively. Finally, the algorithm goes through

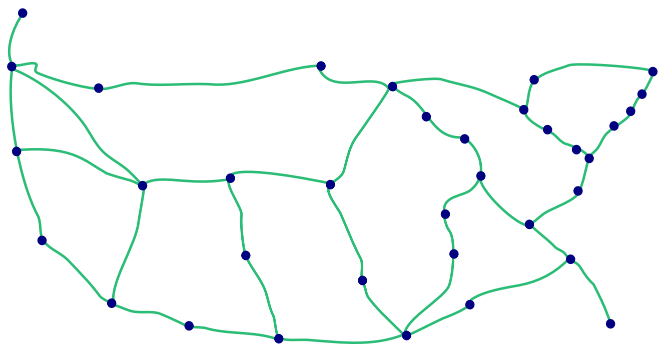

Fig. 3. Simulation OpenFlow switches topology.

the sorted paths to find the first one which fulfils the invariant as well as the connection processing demand.

5 Evaluation

In this section, we evaluate the efficiency of our path selection algorithms for invariant preserving middlebox traversal. In particular, we use simulation to measure the connection acceptance rate and the network and middleboxes utilization. Furthermore, we compare our algorithms in terms of load balancing level and path hops counts per connection.

5.1 Evaluation environment

We have developed a Python flow-level simulator to establish invariant preserving connections through a ISP network. To model ISP network, we use internet2 topology [1] which consists of 34 nodes (Fig. 3). Each node in this figure represents an OpenFlow switch, whereas each edge is a network link with 1 Gbit/second bandwidth. At different locations of the topology, we deploy 12 Middleboxes. Each middlebox has 10 GHz CPU capacity and performs access control using a randomly generated list of destination port numbers (each Middlebox works as a stateless firewall). We generate non-expiring connections with destination port numbers, rates, and processing demands sampled out of a uniform distribution. For each generated connection, we randomly select a source and a destination switch. Using the algorithm in section 4, a connection is established if a network path which preserve its invariant , and fulfils its rate and processing demand is found, otherwise it is rejected. For each successfully established connection, the bandwidth of the links and the processing capacity of middleboxes on the path are updated accordingly.

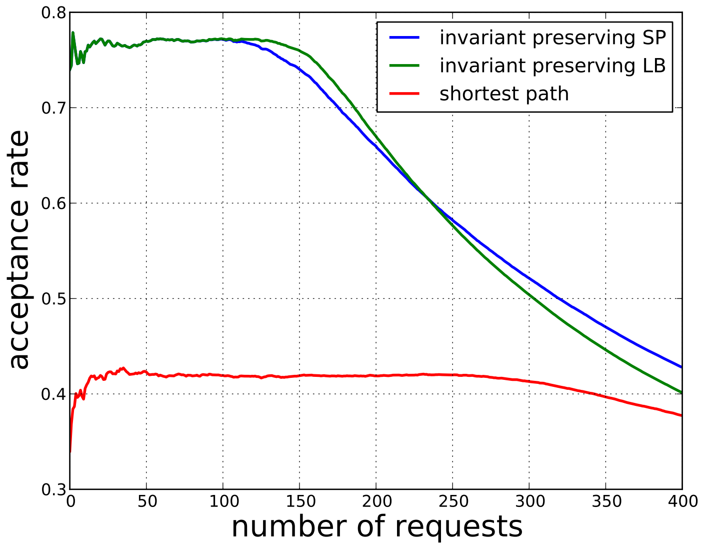

Fig. 4. Connection establishment rate vs. number of arriving requests.

We compare the efficiency of our approach against the traditional shortest path selection. In particular, for each new connection we calculate the shortest path between the connection source and destination, if the shortest path fulfils the connection demand and invariant, the connection is established, otherwise it is rejected.

We conducted our simulation on a machine with Intel Core i5 quad-core CPU at 3.20 GHz and 16 GB of RAM. We repeat each experiment 100 times and report the average.

5.2 Evaluation results

We start by measuring the *connection acceptance rate*. This represents the percentage of connections' sizes for which invariant-preserving paths were selected. As Fig. 4 shows, our algorithms (invariant-preserving SP and invariant-preserving LB) establish almost 40% more connections than the traditional shortest path selection. This is because our algorithms select alternative paths when either the invariant or the connection demand are not fulfilled, whereas the traditional shortest path rejects the connection when the shortest path does not meet the connection requirements. This can be also seen through the evolution of network and middleboxes utilization (Fig. 6 and Fig. 5). It also illustrates that the invariant-preserving LB algorithm achieves better utilization and reaches saturation faster than the invariant-preserving SP. This is because that invariant-preserving LB accepts more requests at the beginning when the network resources are underutilized.

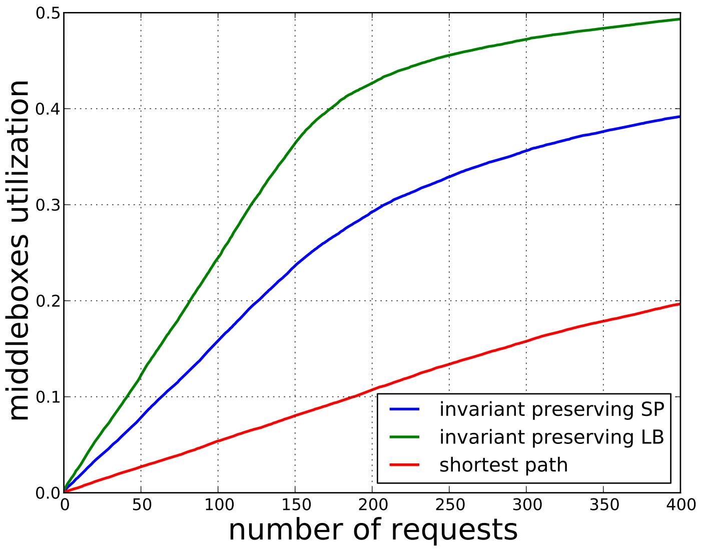

Fig. 5. Total utilization of middleboxes deployed on the network.

We also measure the network *load balancing level* which we define as the maximum link utilization over the average link utilization across ISP network. Lower values of the load balancing level represent better load balancing, whereas a value of 1 designates optimal load balancing. As we can see in Fig. 7, the invariant-preserving LB algorithm outperforms both our invariant-preserving SP algorithm and the traditional shortest path.

We further look at the path length per selected path in terms of the number of hops. As we expect, the invariant-preserving SP outperforms the invariant-preserving LB algorithm (Fig. 8), however, as the network resources become more utilized the difference between both algorithms diminishes. This is because the number of alternative paths with sufficient capacity decreases which limits the solution search space for both algorithms.

6 Conclusions

In this paper, we presented a SDN architecture for establishing invariant preserving connections traversing middleboxes and fostering the collaboration between end-hosts and ISPs. In particular, an end-host can express a desirable behavior from the network, specified as an invariant (*e.g.,* no IP header or payload modification), and the ISP, in turn, can establish a connection through middleboxes that preserve this invariant. To this end, we developed an algorithm to select redirection paths through a sequence of invariant-preserving middleboxes while considering network and middlebox utilization. Our algorithm can be adapted

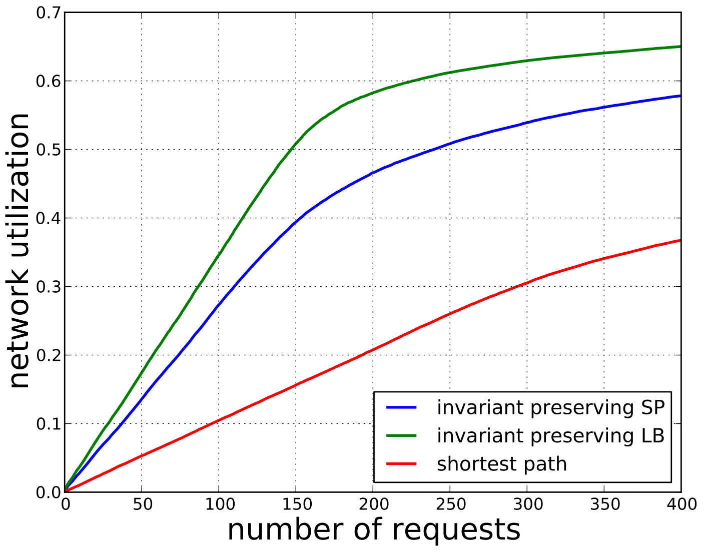

Fig. 6. Total utilization of all network links.

to fulfil different objectives: load balancing or delay minimization. Using simulations, we showed that our algorithm increases substantially the number (more than 40%) of established connections with invariant preservation and achieves a network-wide load balance as well as higher network and middleboxes utilization.

We believe that our work indicates the feasibility of invariant preserving middleboxes traversal and takes a step towards providing more flexibility at the core of the network for new service and protocol deployment. As part of future work, we plan to implement and experimentally evaluate the efficiency of our SDN architecture using our Emulab-based network testbed.

7 Acknowledgments

This work was partially supported by the EU FP7 T-NOVA Project (619520).

References

1. Internet2, http://www.internet2.edu/.
2. R. Sedgewick, Algorithms in C, Part 5: Graph Algorithms, Addison Wesley Professional, 3rd ed., 2001.
3. R. Enns, NETCONF Configuration Protocol, RFC 4741, IETF, Dec. 2006.
4. M. Stiemerling, J. Quittek, and C. Cadar, NEC's Simple Middlebox Configuration (SIMCO), RFC 4540, http://tools.ietf.org/html/rfc4540
5. N. McKeown et al., OpenFlow: Enabling Innovation in Campus Networks, ACM SIGCOMM CCR, 38(2), 2008.

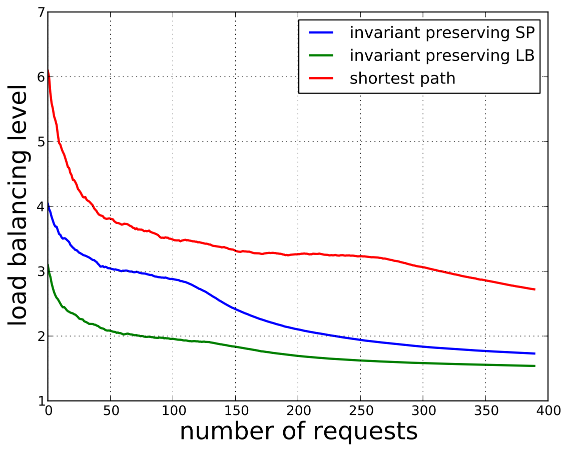

Fig. 7. The network load balancing level.

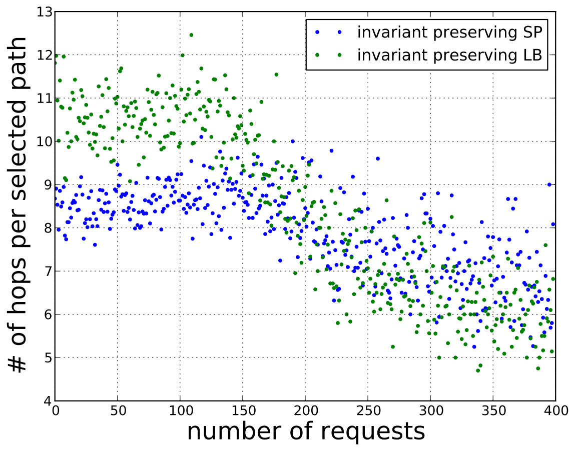

Fig. 8. The length of each selected path for each connection.

6. P. Kazemian , G. Varghese, and N. McKeown, Header space analysis: static checking for networks, USENIX NSDI, San Jose, CA, April 2012.
7. R. Stoenescu, M. Popovici, L. Negreanu, and C. Raiciu, Symnet: Static checking for stateful networks, ACM HotMiddlebox, 2013.
8. D. Wischik, C. Raiciu, A. Greenhalgh, and M. Handley, Design, implementation and evaluation of congestion control for multipath tcp, USENIX NSDI, 2011.
9. Z. Wang, Z. Qian, Q. Xu, Z. Mao, and M. Zhang, An untold story of middleboxes in cellular networks, ACM SIGCOMM, Toronto, Canada, August 2011.
10. C. Raiciu, V. Olteanu, R. Stoenescu, Good Cop, Bad Cop: Forcing Middleboxes to Cooperate, IAB 2015.
11. J. Sherry et al., Making Middleboxes Someone Elses Problem: Network Processing as a Cloud Service, ACM SIGCOMM, Helsinki, Finland, August 2012.
12. S. Fayazbakhsh , V. Sekar , M. Yu , and J. Mogul, FlowTags: enforcing network-wide policies in the presence of dynamic middlebox actions, ACM SIGCOMM HotSDN, Hong Kong, China, August 2013.
13. A. Gember et al., Stratos: Virtual Middleboxes as First-Class Entities.
14. S. Guha, Y. Takeda, and P. Francis, NUTSS: A SIP-based Approach to UDP and TCP Network Connectivity, ACM SIGCOMM FDNA, 2004.
15. J. L. Eppinger, TCP Connections for P2P Apps: A Software Approach to Solving the NAT Problem, http://reports-archive.adm.cs.cmu.edu/anon/isri2005/CMU-ISRI-05-104.pdf.
16. A. Biggadike, D. Ferullo, G. Wilson, and A. Perrig, NATBLASTER: Establishing TCP Connections Between Hosts Behind NATs, ACM SIGCOMM ASIA, 2005.
17. W. Kho, S.A. Baset, and H. Schulzrinne, Skype relay calls: Measurements and experiments, IEEE Global Internet Symposium, 2008.