Traffic Optimization at the Application Level

Proof of concept, development and usefulness evaluation of the ALTO solution

Gonçalo Camaz, Paulo Caldas
Department of Informatics
University of Minho
Braga, Portugal
a76861@alunos.uminho.pt, a79089@alunos.uminho.pt

Pedro Sousa
Centro Algoritmi, Department of Informatics
University of Minho
Braga, Portugal
pns@di.uminho.pt

***Abstract* — This paper presents an overview, proof of concept, and a preliminary demonstrative benchmarking study of the Application-Layer Traffic Optimization (ALTO) architecture proposed by the IETF ALTO Working Group. The main ALTO system purpose is to allow applications to get a more complete view of the underlying network infrastructure, allowing for well-reasoned connection decisions in situations of service redundancy. This paper first begins with a technical description of the ALTO project, and afterwards evaluates how P2P applications, guided by our proposed prototype implementation of the ALTO architecture, perform in comparison to traditional peer selection algorithms on the task of downloading a file in a typical file-sharing P2P network environment. The obtained results from our developed ALTO prototype system state that an application guided by the ALTO solution reduced overall network usage by around 40% with no significant impact in the application performance.**

Keywords – Network optimization, P2P, ALTO, Traffic Engineering, CORE emulator.

I. INTRODUCTION

A Peer-to-Peer (P2P) network is a way of connecting one or more nodes in such a way that computing tasks and resources are scattered throughout those nodes, called "peers", where each one has the ability to act both as a client and as a server. Whilst the P2P network architecture isn't a complete replacement to the classic server-client counterpart, it is an attractive alternative because the nature of its design allows for better scalability and resilience, as well as removing a single logical point of control. As such, P2P networks are now a useful architectural design for many current use cases, for instance file sharing, multimedia streaming, or unregulated social networking. In fact, according to TeleGeography, a telecommunications research firm, the demand for bandwidth is constantly growing and, according to researchers, P2P applications are the fastest growing consumers of it, rivaling the traffic used in web surfing - most shockingly, Cachelogic estimated that between 60% and 80% of capacity on consumer ISP networks is used by P2P traffic [1]. With more focus in the present, a Cisco forecast shows that by 2022, 82% of all IP traffic will be for video, and states that media streaming with P2P as a distribution mechanism is a good tool for low-cost content delivery [2]. The usage and dependency on these kinds of networks makes it a priority that these are correctly implemented and optimized. As such, a point of interest in this work is the way in which, in cases when a service or resource is redundantly deployed throughout the network, a peer chooses another to connect to, e.g., how does a BitTorrent client decide what peer to retrieve a file chunk from when a tracker gives him plenty of options to choose from. For many unstructured P2P networks, the procedure from which to generate a network and choose peers has a crucial random component [3][4], and another alternative bases such choice on the measurement of the Round-Trip Time (RTT) of a connection establishment [4] - a seemingly better alternative than a random choice but far from optimal, partly because message delay is only one of many important metrics to consider, and perhaps a misleading strategy for applications that prefer high transfer rates, as the path with less delay is not necessarily the one with the best data throughput. It thus may not come as a surprise that P2P traffic often crosses network boundaries multiple times [5][6], even considering that the content is, or could easily be, in the client's network proximity [6][7]. As such, the fundamental problem is that P2P applications (or others that regard host choice at application level, for that matter) lack of sufficient reliable knowledge on the underlying network infrastructure and, as such, the host selection process cannot be made optimally. It is with this issue in mind that the "Application-Layer Traffic Optimization" (ALTO) IETF working group [8] arose, and as a solution proposes a server-client architecture and associated protocols that allow clients to query a secure set of servers, who would be provisioned by reputable sources and subsequently available for querying on information that regards the network (including but not limited to topological organization, link properties, or costs defined by one-way delay, maximum bandwidth, hop count, etc.). As to be expected, enabling a communication channel for reputable and updated network information would be ideal for applications to properly reason on situations where hosts must be chosen to connect to, and the consequences of a more optimal mode of operation regarding peer choice would be beneficial for both service clients, who can achieve higher application performance, but also for the entities who provide such information to the server, as they can now more easily transmit to client applications the network status information that would allow a more optimal and compliant consumption of infrastructural resources and, as such, could result in less overall network traffic and reduced congestion probability.

This paper starts by presenting an overview of the ALTO solution (mainly the architecture and its use cases) and the developed ALTO system prototype. Complementary to these, it is also presented the illustrative testing scenarios and related results, which aim to compare host-choosing algorithms in regard to efficiency on network resource usage. Up to now, as far as the authors' knowledge, this is one of the first prototype implementations of the proposed ALTO system along with the presentation of the corresponding illustrative test results.

This paper is organized as follows: Section II presents the ALTO system architecture and characterizes its data resources. Section III presents the developed prototype system modules for the proof of concept of the ALTO system. Section IV overviews the tools and technologies used for the implementation phase. Section V depicts the testing environment and explains how the benchmarking will be executed. Section VI presents the benchmarking results and corresponding discussion. Finally, in Section VII, conclusions are made and future work topics are suggested.

II. System Architecture

This section presents the main ALTO system architecture. Figure 1 presents the main system entities and how these must interact with each other. At its core, the ALTO system is a well-defined structure on how network information must flow as to guarantee two key requirements: i) there is a common interface to allow for network data to be uploaded from a provider to a well-known server; ii) there is a common interface to allow for network data to be downloaded by a client from a well-known server.

As it can be also seen from Figure 1, and in accordance to the ALTO architectural specifications [9], the system architecture delineates three distinct layers.

- *Provisioning layer*: any entity that has valuable network information that it wishes to provide. Examples include authoritative information from ISPs or network managers (such as static policies), collected statistics from third party applications (such as QoS measurements), and information retrieved from routing protocols.
- *Server layer*: section that encapsulates what the system needs to correctly store and provide network information. Note that it is a single logical layer, but multiple physical server layers can exist to ensure service requirements.
- *Client layer*: any entity that could benefit from network information that is provided by the ALTO server layer. Any of the examples given consist of an ALTO client, but they can take many forms. For example, an ALTO client can be embedded on a P2P application to process through the tracker-retrieved information, or be embedded on the Tracker itself as a way for it to act as a proxy on behalf of the P2P client that queried it. Many other use cases exist, as the information retrieved from the server layer can be helpful at the application layer on many levels, whether it be choosing a mirror server that contains a file, deciding the best moment to connect to a host, among many other examples.

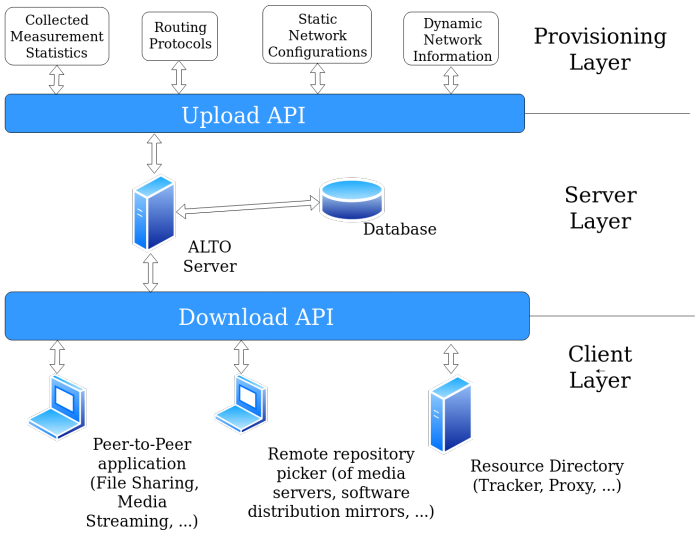

Figure 1. ALTO System Architecture

The information that flows between the entities is characterized in the form of an ALTO resource. An ALTO resource is characterized by having the following metadata [9]:

- A unique and immutable URI from which to be accessed.
- An IANA-registered media content type.
- A list of IANA-registered accept types, in situations where query parameters are required.
- A list of capabilities, e.g., whether or not it allows filtering or data calendarization.
- A list of resources it refers to, if applicable.

Alongside this information, the network content itself is presented. There are, at the time of writing, four different types of resources considered in the ALTO system [9]. Figure 2 displays them, along with illustrative abstract examples of each. More specifically, the resource types are the following:

- *Network map*: aggregation of a set of endpoints into a single identifier called PID. The reasoning of how such aggregation is handled is left to the provisioning parties. For example, a PID could group endpoints by the Autonomous System they reside in, geographic proximity, or one or many subnets. The reasoning of such a resource is to allow for better scaling of ALTO systems, as it is more efficient to group endpoint costs between endpoints with similar properties.
- *Endpoint property map*: mappings of the properties that an endpoint possesses. For example, the endpoint's geographical position or connection type.
- *Cost map*: mappings of pairs of routing entities to a cost associated with its path. These entities can either be PIDs or endpoint addresses, and the costs can

range from generic routing cost, hop count, packet loss percentage, etc. The full range of metrics currently considered in the ALTO project is available on the ALTO cost metrics document [10].

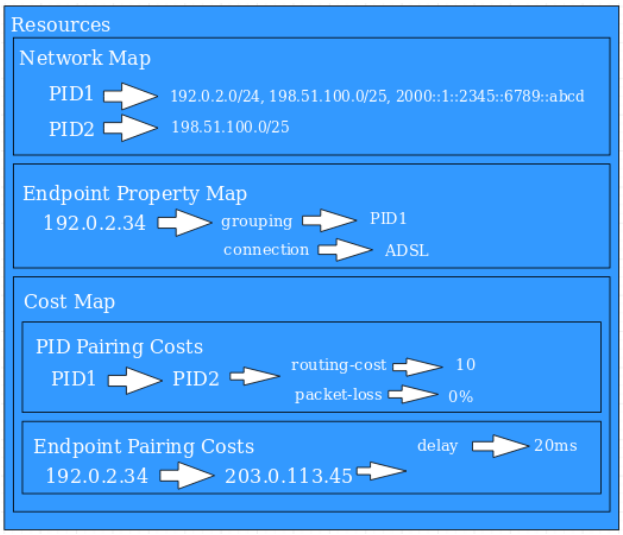

Figure 2. ALTO System Resources

III. PROTOTYPE SYSTEM MODULES

This section aims to present how the ALTO system prototype for proof of concept was implemented as to test its usefulness in a file-sharing P2P network. A useful ALTO system prototype should implement the core functionalities of the three architectural layers displayed on Fig. 1. For the simulation to correctly perform, there should exist: i) a provider entity able to retrieve network information and upload valuable metrics to a central server; ii) a client entity able to retrieve metrics from a central server, and iii) a server able to interface with the aforementioned entities in order for information to flow between a provider and a client. The three layers were implemented as follows:

- Provider layer: A developed program expects the user to input configuration files that provide network maps and costs of links. Given those, the program prompts the user for what information he wishes to upload. Furthermore, a developed module allows for the creation of a shortest path map of a given metric and within a given source, through the Dijkstra algorithm. As an extra feature, it uploads a called "multi-cost map" [11] of said shortest path, with the associated costs of such path (for example, consider that the algorithm calculates the shortest path between network nodes A and B considering a generic routing cost and, for instance, also associates to that path the required number of traversed hops, and the bandwidth of the lowest provisioned link along it). Finally, the module uses a developed class entitled *ALTOResourceUploader* as way to interface with a server. This class is thus responsible for encoding the information and following the uploading protocol designed for this work (as one is currently not specified by the ALTO working group), which consists of a POST request of a JSON object to a private upload URL that the ALTO server is hosting.
- Server layer: According to the IETF ALTO working group specifications, the ALTO server must implement a RESTful interface [9]. As such, our implementation goes as follows: a web server hosts two types of URLs, one responsible for upload actions, and another responsible for download actions. Both servlets use the developed class *ALTOServer*, which encapsulates all the behavior associated with an ALTO Server - the upload and download of ALTO resources, most specifically. As to be expected, this class handles all business logic, which includes but is not limited to data storage, data parsing, information directory updating, and anomaly detection (e.g. incorrect parameters, non-existing requested resource, etc.). For data storage and retrieval needs, the server must contain an active database instance reference to connect to.
- Client layer: The client layer is instantiated in the form of a file-sharing P2P based client. This client is capable of splitting a file into many shards, and both request and provide shards on the network. As input, the client takes the files it wishes to share with its peers, and the mode of operation. More specifically, these modes of operations are the following:

  - IDLE: Only provide the given files to the network.
  - RANDOM: Retrieve a single file whilst, in case of multiple peers having the same shard, using a random-based peer selection algorithm.
  - PING: Retrieve a single file whilst, in case of multiple peers having the same shard, prioritizing the one that, at the moment of selection, has the lowest RTT measurement of a probe packet sent to him.
  - ALTO: Retrieve a single file whilst, in case of multiple peers having the same shard, prioritizing the one that contains the lowest "routing-cost" metric from the client's origin. This implies that the default network map and cost map of an ALTO server must be retrieved from a previously configured ALTO server.

IV. TOOLS AND TECHNOLOGIES

This section briefly references the tools and overall pieces of technology that were used in the development of the developed ALTO system prototype and in the experimental part of the work. Integral to most of the implementation phase

was the Java platform [12], which allowed the creation of Java applications. The reasoning for such choice was mainly due to the existing experience with the Java language and all the development and deployment tools associated with it, which helped with the development of the prototype, however, a more important reason stems from the project's possible magnitude in case of future work, and as such an object oriented paradigm with a well proven development environment seemed fitting. Both the network data provider and a P2P client were packaged as Java applications that could be run from the terminal, as it would allow for easier test automation via bash [13] scripting. As for the server layer, Apache Tomcat [14] was used to deploy a WAR file which was packaged by the Maven [15] build tool. This WAR file contained the implementation of Java Servlets which were responsible for handing client HTTP requests. Following an MVC architecture, the servlets, acting as the controller layer, resorted to Java classes that contained most of the business logic. The database of choice was MongoDB [16], due to the fact that data can be stored and retrieved in a JSON format, the same that should constitute any ALTO resource when in an HTTP response [9] – this possibility to minimize translation between data storage and data to be transmitted made development less complex and easier to validate.

The main tool for the testing environment was the CORE network emulator [17] to create the network topology and to run real applications (i.e. the developed modules of the prototype) in virtualized Linux based stations. Within this environment, a bash script would be run to execute all the applications needed to initiate all the three ALTO layers, provider, server and client, in the appropriate network hosts. Additionally, the Vtysh [18] terminal client allowed for the modification of OSPF network routing costs in the second testing scenario. Finally, the Vnstat [19] tool was used to monitor network traffic, as to calculate the overall used bandwidth.

V. TESTING ENVIRONMENT

This section aims to detail the environment in which the tests were performed, and the methodologies used to gather data restuls. The main goal is to create a reasonably believable network environment where an overlay network of a P2P application would reside, evaluate network and application performance on the task of retrieving a file spliced among several peers. To achieve this, a 300 MB file was divided into ten 30MB chunks and randomly distributed throughout the peers in the network, and a client P2P application (from now on called testing host) retrieves information from the P2P tracker of where those chunks reside, sequentially retrieves each chunk and, in cases of a chunk residing in many peers, performs one of the three non-idle modes of operation specified in the previous section. As can be visualized on Fig. 3, the network topology consists of a backbone area that connects many edge networks. The edge networks are where the P2P client applications reside (labeled *"P2PCli#"*), save for a single edge network where the auxiliary servers are hosted: i) the ALTO server, ii) the network information provider, and iii) the P2P tracker. All backbone routers are

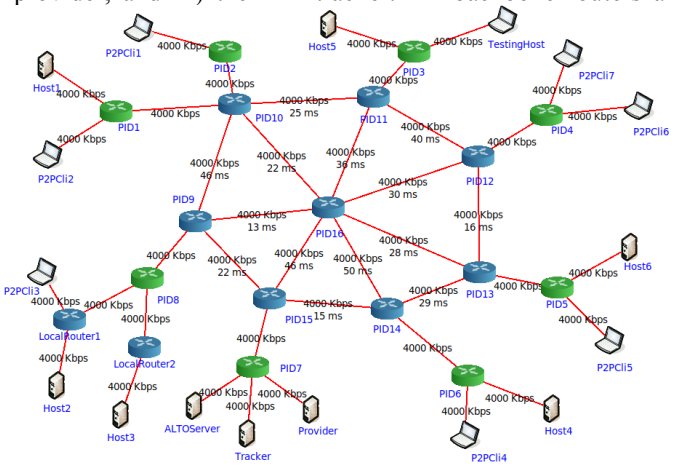

Figure 3. First scenario test topology

labeled with a PID number, and routers that serve as a gateway will have their PID number aggregate all the subnets that it gateways from. The network provider will read from a JSON file that contains the bandwidth values and corresponding OSPF link routing costs of each link in the backbone area and, using the Dijkstra algorithm, will derive the shortest path map from the PID representative of the testing host, to all the PIDs of the existing gateway routers. All links in the backbone area have been setup to contain a propagation delay of 20-50ms, randomly distributed between the links, as to create a more heterogeneous and unpredictable network behavior. Taking this into account, the tests will be performed in two types of scenarios:

- All network links have a bandwidth of 4 Mbps, as per Figure 3. The network was configured with OSPF link costs of 1 for every backbone connection and the provider calculates the shortest distance between the testing host and any gateway router (which in this case represents the minimum number of hops).

- To create a even more heterogeneous scenario, all network links have a bandwidth of 4 Mbps, save for three under-provisioned ones that are seen in light blue in Figure 4, which have capacities of 1 Mbps. Such links were assigned with higher OSPF link costs. This will cause that some paths change their routing costs or, in some cases, the network paths between some routers integrate distinct intermediary routers.

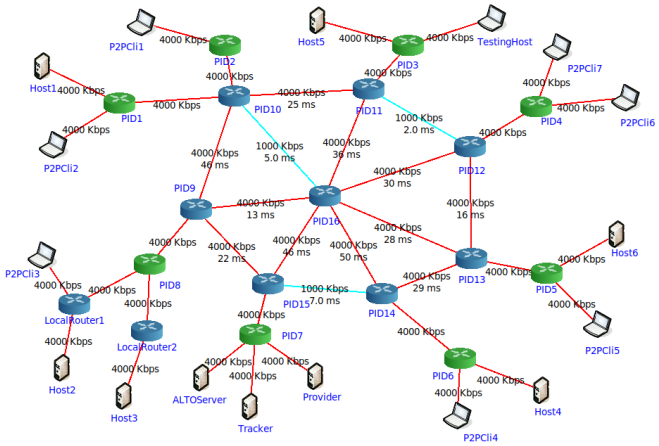

Figure 4. Second scenario test topology

Per each scenario, a total of fifteen measurements were performed, five per P2P operation mode (RANDOM, PING, and ALTO). The measured metrics will be the time to download the file and the total amount of network traffic flowing along the all network backbone area.

VI. SIMULATION RESULTS AND DISCUSSION

This section now focuses on presenting and discussing the obtained results in the considered scenarios. For the first scenario, Table I displays the obtained download times (average, maximum and minimum) and the total backbone bandwidth used for all three methods. As also depicted in Fig. 5, the total backbone bandwidth used per method shows that an ALTO-assisted download was clearly much more resource friendly, as it nearly halved the total bandwidth usage when compared to the alternative methods. The results clearly show that the use of the ALTO system allow a much more efficient use of the network resources, which a much more lower amount of P2P traffic traversing the core network, caused by the fact that the shortest path graph calculated in this scenario, where all links have equal bandwidth and costs, resulted in a way of reaching the needed chunks in the fewest hops possible. Regarding the download time, the ALTO method was marginally, although consistently, faster. By contrast, the random and ping-oriented methods were less efficient in the number of hops needed to retrieve the chunks, thus the observed higher download times and total load in the network.

TABLE I. System Test Results [Scenario 1]

Mode	Total Load (Megabytes)	Value Type	Download Time (s)
Random	4245,63	Avg	663,44
		Min	662,81
		Max	664,51
Ping	4238,73	Avg	666,67
		Min	666,57
		Max	666,84
ALTO	2422,07	Avg	661,51
		Min	661,33
		Max	661,73

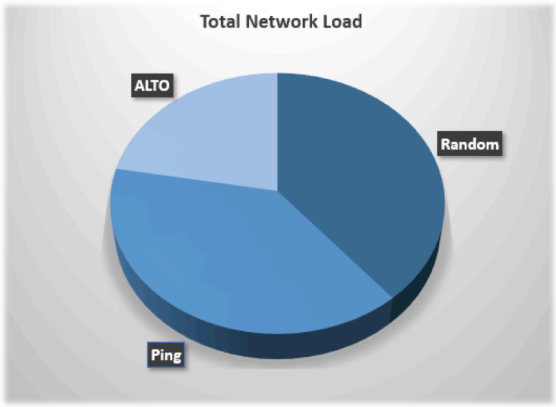

Figure 5. Network Load Comparison

For the second scenario, Table II displays the results in a similar structure to the previous mentioned, and the trend remains equal to the first scenario. In fact, as visible in Table II, the ALTO assisted method presents again much lower values for the total amount of P2P traffic traversing the network, which represents economical gains due to a more efficient use of the available network resources. Despite that, the total load difference between the ALTO method and the other ones is not so expressive as in the previous experiment. This is explained by the fact that in the second scenario some OSPF costs were changed and some path routing costs no longer express the minimum number of hops between the routers. Nevertheless, the total load difference between the methods is still expressive, clearly benefiting the ALTO method.

As regards the download times (also expressed in Fig. 6), similar reasoning can be made as for the previous scenario. The important conclusion to draw is the fact that the ISP gains obtained in the save of network resources, were not obtained with the cost of the degradation of the P2P application running in the network. On the contrary, the results observed in Table II and Fig. 6 show also a marginal gain in the downlad times obtained when the P2P application uses the ALTO system. Thus, the resutls observed in both scenarios clearly illustrate and corroborate the advantages of an ALTO assisted system.

TABLE II. System Tests Results [Scenario 2]

Mode	Total Load (Megabytes)	Value Type	Download Time (s)
Random	3465,91	Avg	663,94
		Min	662,95
		Max	664,98
Ping	3545,99	Avg	667,65
		Min	667,54
		Max	667,74
ALTO	2596,21	Avg	662,33
		Min	662,18
		Max	662,70

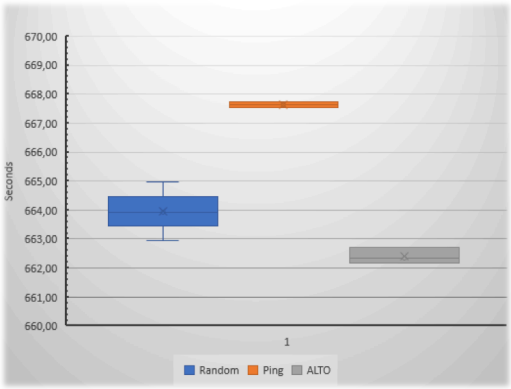

Figure 6. Download Times Comparison

The results allow us to conclude that the method that was guided by the ALTO system was favorable on both scenarios – this method was faster even on a very lenient topology, and such speed disparity would increase with less well provisioned topologies where bottlenecks and less path redundancy can occur with higher probabilities. The overall network usage was massively decreased, and its important to note that such was accomplished with no performance hit. As such, a P2P client application that could obtain privileged information on the network topology could be more efficient in the peer-selection process, and thus greatly reduce the network resources needed to retrieve all chunks from a file.

VII. CONCLUSIONS

This article accomplished the two main goals it set out to tackle. Firstly, a review of the existing documentation of the IETF ALTO project allowed for the contextualization and specification of such project, and the creation of a proof of concept with core functionalities, mainly the ability for clients to retrieve network information from a server that could be provided by trusted users. The specification of the solution and the existing prototype implementation allows for the project continuation and maintenance, as it is well structured and complies to the ALTO projects requirements. Secondly, a testing environment was created to empirically test the ability of a file-sharing P2P network to use network resources within three peer-choosing algorithms – random choice of peers, choice of peers that prioritizes shortest RTT time, and choice of peers influenced by an ALTO infrastructure that provides resources to a client in the form of network and cost maps that could allow to choose the peer with the smallest routing cost.

The paper main contributions were to further present the ALTO project and its purpose within the current needs of ISPs and client applications. Such project came to life in the form of a proof of concept with core functionalities, which were tested in a credible scenario where sufficient conditions were met for it to objectively improve the management of network resources.

Further work should focus on expanding the existing prototype to increase its functionalities, in such a way that project guidelines and requirements are followed. For example, the ability to encode temporal information into costs in the form of cost calendarization or server discovery mechanisms. Research into deployment methods of an ALTO system on a large scale are also of interest, in regards of service availability and response time, for example. Additionally, security analysis and risk assessment are important when considering the confidentiality of some of the network information that can be shared, and the ease-of-use nature of such data. Finally, more research can be done on the effectiveness of the ALTO solution in a real scenario, with focus on understanding what concrete metrics should be shared, with what frequency, and with whom, in order to maximize network and client application performance in a way that complies with ISPs, network administrators, and clients security and privacy policies.

ACKNOWLEDGMENT

This work has been supported by FCT – Fundação para a Ciência e Tecnologia within the R&D Units Project Scope: UIDB/00319/2020

REFERENCES

[1] Wired.com, “P2P Fuels Global Bandwidth Binge”, 2005. [Online]. Available: https://www.wired.com/2005/04/p2p-fuels-global-bandwidth-binge

[2] Cisco.com, "Cisco visual networking index: forecast and trends, 2017-2022 white paper". Updated in 2019. [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-741490.html

[3] V. Vishnumurthy and P. Francis, “On random node selection in P2P and overlay networks”, in Proceedings of INFOCOM’06, 2006.

[4] J. Seedorf and E. Burger, “Application-Layer Traffic Optimization (ALTO) Problem Statement”, RFC 5693, 2009.

[5] V. Aggarwal, S. Bender, A. Feldmann, and A.Wichmann, “Methodology for estimating network distances of Gnutella neighbors”, in Proceeding of INFORMATIK’2004, 2004.

[6] T. Karagiannis, P. Rodriguez, and K. Papagiannaki, “Should ISPs fear peer-assisted content distribution?”, in ACM SIGCOMM, Proceedings of IMC, USENIX Association, 2005.

[7] A. Rasti, D. Stutzbach, and R. Rejaie, “on the long-term evolution of the two-tier Gnutella overlay”, in Proceedings of Global Internet, 2006.

[8] ALTO. https://datatracker.ietf.org/wg/alto/about/

[9] R. Alimi, R. Penno, Y. Yang, S. Kiesel, S. Previdi, W. Roome, S. Shalunov and R. Woundy, “Application-Layer Traffic Optimization (ALTO) protocol”, RFC 7285, September 2014.

[10] Q. Wu, Y. Yang, Y. Lee, D. Dhody and S. Randriamasy, “ALTO performance cost metrics”. Internet-draft draft-ietf-alto-performance-metrics-08, Work in Progress, November 2019.

[11] S. Randriamasy, W. Roome and N. Schwan, “Multi-Cost Application-Layer Traffic Optimization (ALTO)”, RFC 8189, October 2017.

[12] JavaPlatform. https://www.oracle.com/technetwork/java/javase/overview/index.html

[13] GNU Bash. https://www.gnu.org/software/bash/

[14] Apache Tomcat. https://tomcat.apache.org/

[15] Apache Maven. https://maven.apache.org/

[16] MongoDB. https://www.mongodb.com/

[17] CORE. https://www.nrl.navy.mil/itd/ncs/products/core

[18] Vtysh. https://linux.die.net/man/1/vtysh

[19] VnStat. https://humdi.net/vnstat/