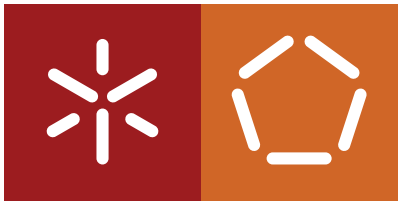
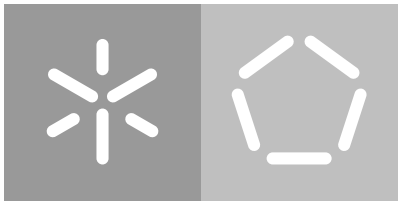


Universidade do Minho
Escola de Engenharia
Departamento de Informática

Paulo Edgar Mendes Caldas

Development of a system compliant with the Application-layer Traffic Optimization protocol

January 2021



Universidade do Minho
Escola de Engenharia
Departamento de Informática

Paulo Edgar Mendes Caldas

Development of a system compliant with the Application-layer Traffic Optimization protocol

Masters dissertation
Integrated Master's in Informatics Engineering

Dissertation supervised by
Pedro Nuno Miranda de Sousa

January 2021

AUTHOR COPYRIGHTS AND TERMS OF USAGE BY THIRD PARTIES

This is an academic work which can be utilized by third parties given the compliance of the rules and good practices regarding author and related copyrights, which are internationally accepted.

Therefore, the present work can be utilized according to the terms provided in the license shown below.

If the user needs permission to use the work in conditions not foreseen by the licensing indicated, the user should contact the author, through the RepositóriUM of University of Minho.

License provided to the users of this work

Attribution-NonCommercial-ShareAlike
CC BY-NC-SA
https://creativecommons.org/licenses/by-nc-sa/4.0/

[This license allows others to remix, transform and build upon this work, to non-commercial purposes, as long as appropriate credit is given, and the new contributions are licensed under the same license as the original]

ACKNOWLEDGEMENTS

I would like to firstly thank my advisor, professor Pedro Nuno Sousa, who was always present in any moment I struggled and required input to improve on my work.

I would also like to thank my family for financially and emotionally supporting me through my academic journey, the friends I've made along the way that made me see the best in people, and last but not least my dog Oscar who showed me unconditional love like only a dog could.

I finally also thank you, the reader - as a work unused is no work at all, may you find some value in it.

STATEMENT OF INTEGRITY

I hereby declare having conducted this academic work with integrity. I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration.

I further declare that I have fully acknowledged the Code of Ethical Conduct of the University of Minho.

Paulo Edgar Mendes Caldas

_____

ABSTRACT

With the ever-increasing Internet usage that is following the start of the new decade, the need to optimize this world-scale network of computers becomes a big priority in the technological sphere that has the number of users increasing, as are the *Quality of Service (QoS)* demands by applications in domains such as media streaming or virtual reality.

In the face of rising traffic and stricter application demands, a better understanding of how *Internet Service Providers (ISPs)* should manage their assets is needed. As an effort to optimize the Internet, one important concern is how applications utilize the underlying network infrastructure over which they reside. An evident issue is that most of these applications act with little regard for ISP preferences, as can be evidenced by their lack of care in achieving network proximity among neighboring peers, a feature that would be preferable by network administrators and that could also improve application performance. However, even a best-effort attempt by applications to cooperate will hardly succeed if ISP policies aren't clearly communicated to them. A system to bridge layer interests has thus much potential in helping achieve a mutually beneficial scenario.

The main focus of this thesis is the *Application-Layer Traffic Optimization (ALTO)* working group, which was formed by the *Internet Engineering Task Force (IETF)* to explore standardizations for network state retrieval. The working group devised a request-response protocol where authoritative and trustworthy entities provide guidance to applications in the form of network status information and administrative preferences, with the intent of achieving layer cooperation during normal application operations as a means to reach better Internet efficiency through the optimization of infrastructural resourcefulness and consequential minimization of its operational costs. This work aims to implement and extend upon the ideas of the ALTO working group, as well as verify the developed system's efficiency in a simulated environment.

Keywords: Application-Layer Traffic Optimization, Content Distribution Networks, Network Optimization, Peer-to-Peer, Traffic Engineering

RESUMO

Com o uso cada vez mais acrescido da Internet que acompanha o início da nova década, a necessidade de otimizar esta rede global de computadores passa a ser uma grande prioridade na esfera tecnológica, que vê o seu número de utilizadores a aumentar, assim como a exigência, por parte das aplicações, de novos padrões de Qualidade de Serviço (QoS), como se vê em domínios de stream multimédia em tempo real ou realidade virtual.

Face ao aumento de tráfego e a padrões de exigência aplicacionais mais restritos, uma melhor compreensão é necessária de como os fornecedores de serviços Internet (ISPs) devem gerir os seus recursos. Numa tentativa por otimizar a Internet, um ponto fulcral é o de perceber como as aplicações utilizam os recursos da rede sobre a qual residem. Um problema aparente é a falta de consideração que estas e outras aplicações têm pelas preferências dos ISPs durante a sua operação, como as aplicações P2P pela sua falta de esforço em obter proximidade topológica com os vizinhos na rede overlay, que caso existisse seria preferível por administradores de rede e teria potencial para melhorar o desempenho aplicacional. Todavia, uma tentativa de melhor esforço por parte das aplicações por cooperar não será bem-sucedida se tais preferências não são claramente comunicadas. Um sistema que sirva de ponte de comunicação entre as duas camadas tem portanto bastante potencial na tarefa de atingir um cenário mutuamente benéfico.

O foco principal desta tese é o grupo de trabalho ALTO, que foi formado pelo IETF para explorar estandardizações para recolha de informação do estado da rede. Este grupo de trabalho especificou um protocolo de pedido e fornecimento de recursos onde entidades autoritárias auxiliam aplicações com informação sobre estado de rede e preferências administrativas, como forma de obter cooperação entre camadas durante operação aplicacional, para melhor otimizar a Internet através de uma mais eficiente utilização de recursos infraestruturais e a consequente minimização de custos operacionais. Este trabalho pretende implementar e alargar as ideias do grupo ALTO, bem como verificar a eficiência do sistema desenvolvido num ambiente simulado.

Palavras-Chave: Application-Layer Traffic Optimization, Content Distribution networks, Engenharia de Tráfego, Otimização de rede, Peer-to-peer

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

ACL Access-Control List.
ADSL Asymmetric digital subscriber line.
ALTO Application-Layer Traffic Optimization.
ANE Abstract Network Element.
API Application Programming Interface.
AS Autonomous System.

BGP Border Gateway Protocol.

CAN Content Addressable Network.
CaTE Content-Aware Traffic Engineering.
CDN Content Distribution Network.
CDNI Content Distribution Network Interconnection.
CORE Common Open Research Emulator.
CPU Central Processing Unit.

DHT Distributed Hash Table.
DiffServ Differentiated services.
DNS Domain Name System.
DoH DNS over HTTPS.
DoS Denial of Service.
DPI Deep Packet Inspection.
DTO Data Transfer Object.

EGP Exterior Gateway Protocol.
EMEA Europe, the Middle East and Africa.

FCC Federal Communications Commission.
FTP File Transfer Protocol.

GNP Global Network Positioning.
GSLB Global Server Load Balancing.

HTTP Hypertext Transfer Protocol.

HTTPS Hypertext Transfer Protocol Secure.

ID Identifier.
IDMaps Internet Distance Map Service.
IETF Internet Engineering Task Force.
IGP Interior Gateway Protocol.
IP Internet Protocol.
ipv4 Internet Protocol version 4.
ipv6 Internet Protocol version 6.
IRD Information Resource Directory.
ISP Internet Service Provider.

JSON JavaScript Object Notation.

LSPD Label Switched Path Database.

MAC Media Access Control.
MPLS Multiprotocol Label Switching.
MTR Multi-Topology Routing.
MVC Model-View-Controller.

NETCONF Network Configuration Protocol.
NetPaaS Network Platform as a Service.

OSPF Open Shortest Path First.
OSPFv2 Open Shortest Path First Version 2.

P2P Peer-to-Peer.
PaDIS Provider-Aided Distance Information System.
PC Personal Computer.
PID Provider-Defined Identifier.
PoP Points of Presence.

QoE Quality of Experience.
QoS Quality of Service.

RAM Random-Access Memory.
RBAC Role-Based Access Control.
REST Representational state transfer.
RFC Request for Comments.

RTT Round-Trip Time.

SDN Software Defined Networking.
SNMP Simple Network Management Protocol.
SQL Structured Query Language.

TCP Transmission Control Protocol.
TED Traffic Engineering Database.
TLS Transport Layer Security.

URL Uniform Resource Locator.

XMPP Extensible Messaging and Presence Protocol.

1 EXPERIMENTS

The purpose of this chapter is to overview the experimentation phase of the project, which contains the work done to deploy and measure how applications behave when using either classic or *Application-Layer Traffic Optimization (ALTO)*-guided solutions when faced with some of the constructed scenarios, that force some decision which impacts the underlying network. Whereas the developed unit tests in the implementation stage aim to verify the correct functionality of separate units of code pertaining to the system, the execution of the entire system as a whole to serve a set of hypothetical use cases can help achieve a better grasp on how correctly the system behaves .

Adjacent to the goal of testing the system's behavior in an emulated environment, the experimentation phase also aims to embed in such environment a list of case study scenarios where applications could leverage the ALTO system to their advantage, and subsequently observe and measure if and how the ALTO server can help the client with its network insight that guide the client in taking application decisions that aim for a win-win scenario between the overlay and underlay. As comparison, other known application-network interaction strategies will also be observed and their results measured as a means to compare their impact in comparison to one that utilizes the implemented system.

Findings on existing application-layer traffic optimization interactions and the proposal of the ALTO protocol made on Chapter **??**, together with the specified system extensions on Chapter **??** leads one to believe that a theoretical mutually beneficial scenario exists in an ALTO approach that could not exist in one where only one of the layers gets all the input when applying traffic optimization decisions. This chapter, however, puts those theoretical scenarios into a practical environment that could be replicated by those reading this work, and exposing the created scenarios and collected data can corroborate the theoretical conclusions, as well as leaving an opportunity for future discussion on how the system behaved, including its performance, its success in aiding clients, other existing client options that could be a better route, system shortcomings, etc. In general, this discussion benefits the ALTO project and can give more maturity to the system as it was put through multiple test scenarios against other common strategies.

Section 1.1 displays the chosen technologies for tasks pertaining to the experiments. Section 1.2 focuses on the required steps taken to setup the testing environment. This includes the design and deployment of a network topology to be emulated, the creation of mock applications to serve as clients for the system, and the design and deployment of application and network status measurement tools. Section 1.3 follows with the individual overview of the devised scenarios to be executed, and with it experiment specifics such as the initial problem, what strategies will be tested to solve it, how many runs will be made per strategy, and what metrics will be measured. Finished the experiments, Section **??** will display the obtained results that were collected emulated environment, and Section **??** after that will discuss these results and how they fare with the theoretical findings.

1.1 TECHNOLOGIES USED

The *Common Open Research Emulator (CORE)* [1] was used as a network emulator and represents the backbone of the experiments as a whole as it will serve as the background for the running scenarios. This tool allows for the creation and emulation of network environments, and with it are included the abilities to construct network topologies and manipulate properties of the member nodes, which can include network routers, switches, and host machines, that will all be used for the designed experiment scenarios. Additionally, link connection properties can themselves be customized, as parameters like maximum bandwidth, packet loss percentage, or packet delay can be meticulously customized, and in fact will be in the upcoming scenarios as a means to simulate a given circumstance that may occur in a realistic environment, such as link inefficiency that results from peak traffic hours. Another property that was of great importance for the selection of CORE on this work is that, on top of the virtual network environment, arbitrary code can be run on behalf of a given entity and can be addressed to another, acting as if it were an actual network. This will be leveraged to run software pre-packaged in the emulator, such as routing protocols that are essential for the correct expected behavior of a simulated network, but also to schedule software execution that was developed for this work, which includes the ALTO server, network state providers - e.g., probing daemons and application feedback collectors - and system clients for the *Peer-to-Peer (P2P)* and *Hypertext Transfer Protocol (HTTP)* mock applications that will be devised to play out a particular experiment scenario, and which will have embedded into it an ALTO client to interface with the server for

council. As the simulation tool runs on Linux and builds a simulated network that behaves very much like a real one, well known real-application tools can be used on top of it in other needed areas, including the deployment and measuring phases, which gives plenty of flexibility on tool selection.

The execution of arbitrary code on the network nodes is accomplishes with the vcmd [2] tool, that runs the specified commands in control channels that are created at runtime by CORE - for example, the following command executing a ping to address "10.0.0.1" with origins on node "P2P-Client-1", and on simulation session "12345":

Listing 1.1: Execution of an example command through the control channel of a given node

```
$ vcmd -c /tmp/pycore.12345/P2P-Client-1 -- ping 10.0.0.1
```

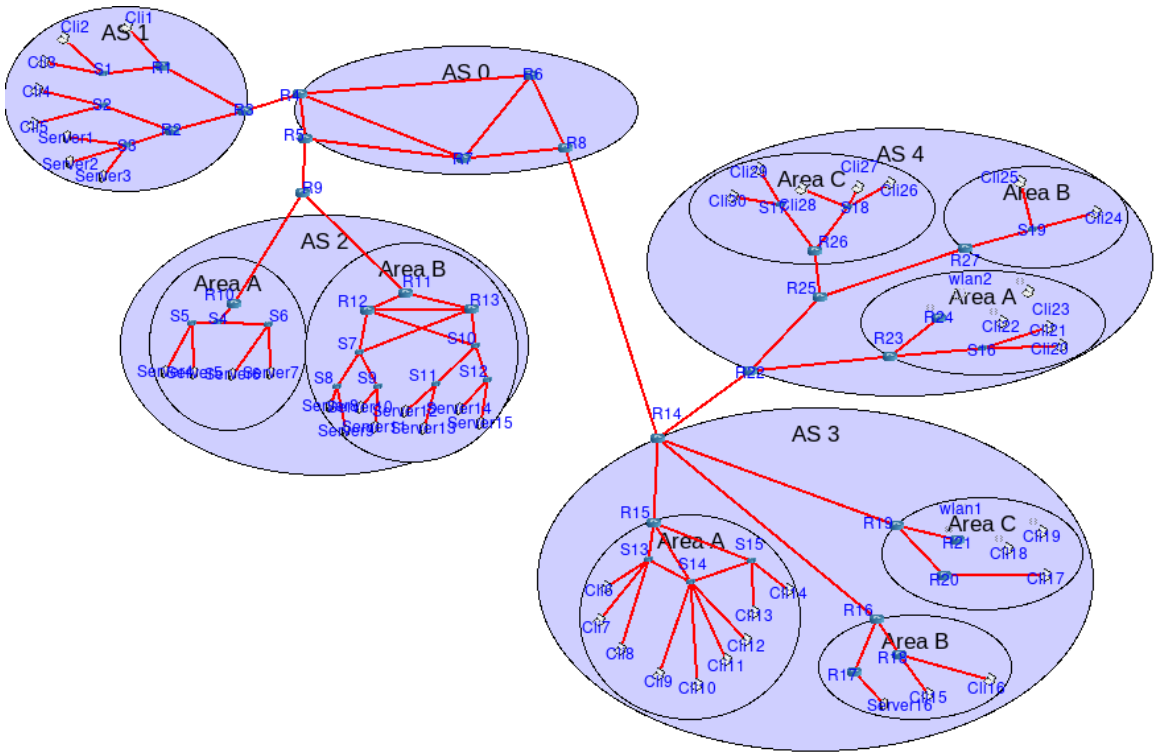
Python [3] will be utilized to implement all simple software prototypes whose purpose is uniquely to test the application in a real scenario. This includes the P2P file-transfer applications, the HTTP servers and clients, and the throughput-intensive activities done by the data servers. Appended to this programming language will also be the task of application monitoring, which includes the retrieval of performance statistics - doing so in the application's code itself, instead of using external tools, because more fine grained access exists and individual tasks can be monitored for how long and how well they perform. The choice of this language over others is simply that these software prototypes are not intended to be highly optimized, nor are they to be complex. Instead, their mode of operation is supposed to be simple in nature, to remove complex variables that might make the experiment results harder to infer upon, and to make reasoning and replication of experiment results easier. Python seems then like a good fit due to its easy syntax, its interpreted nature that skips work that would otherwise be needed for compilation that might increase performance - but is not required - and, finally, its massive collection of helpful libraries.

Finally, for the task of network monitoring, to collect data representative of the impact that a given application strategy had in the infrastructure, the Linux file "/proc/net/dev" will be read in the virtual environment of each node, which contains the total amount of bytes that entered and left their interfaces during the scenario's run. Parsing all the data from each node and grouping them by area and *Autonomous System (AS)* is performed on a shell script, following the execution of the given scenario, and utilizing common text manipulation tools such as awk [4].

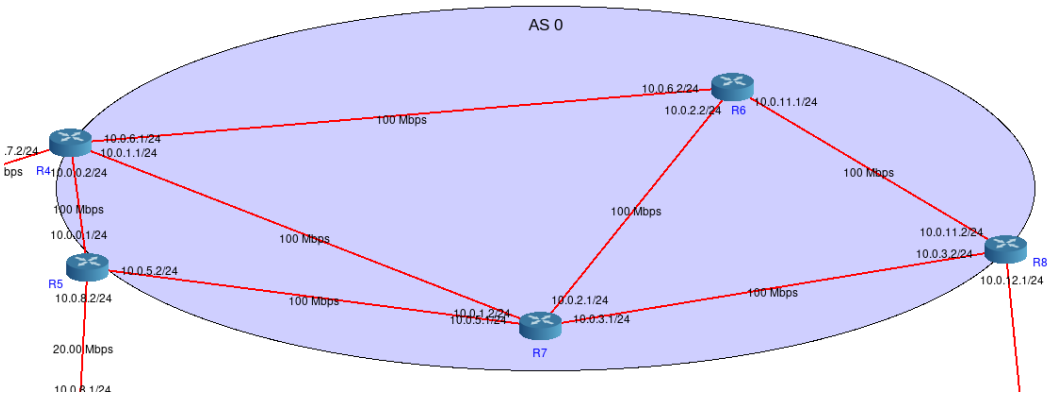
1.2 SETUP

Figure 1.1 displays the topology that will act as the main environment for all the devised experiments, with partitioned views to be subsequently introduced. Figure 1.1a presents a simplified global view the network as a collection of ASs. It was designed with the intent of reflecting, at a smaller scale, the structure of the Internet, in particular with it being an aggregation of multiple, heterogeneous, domains, each with their own topological properties and internal policies, with them being administrated by different organizations. A single backbone network, shown on Figure 1.1b, provides connectivity between many ASs and, to do its job correctly, a high degree of path redundancy exists between its routers, and the links have better capabilities than those associated with stub networks. Figure 1.1c shows AS 1, a simple topological structure consisting of five *Personal Computers (PCs)* and three dedicated servers, connected with the help of switches and routers, that eventually connect to a single edge router that links with the backbone. Figure 1.1d shows AS 2, which is representative of a data center with two *Open Shortest Path First (OSPF)* areas, both constructed with a hierarchical organization common for data center networks. Links in these regions are also highly capable and high traffic peak times are expected to occur. AS 3, shown in Figure 1.1e, is a slightly more complex stub network compared to AS 1, but has the same structure, with the addition of having three OSPF areas instead of one, and a variety of nodes and links with different properties - for example, the links in area A are generally better, whilst area C has wireless connections in it that are expected to have worse performance and be less reliable. Finally, AS 4, depicted on Figure 1.1f, connects directly with AS 3, meaning that it also acts as a transit AS to the rest of the network. Similarly to AS 3, it consists of a stub network accessed by many end users and some servers, and both node and link properties vary accordingly.

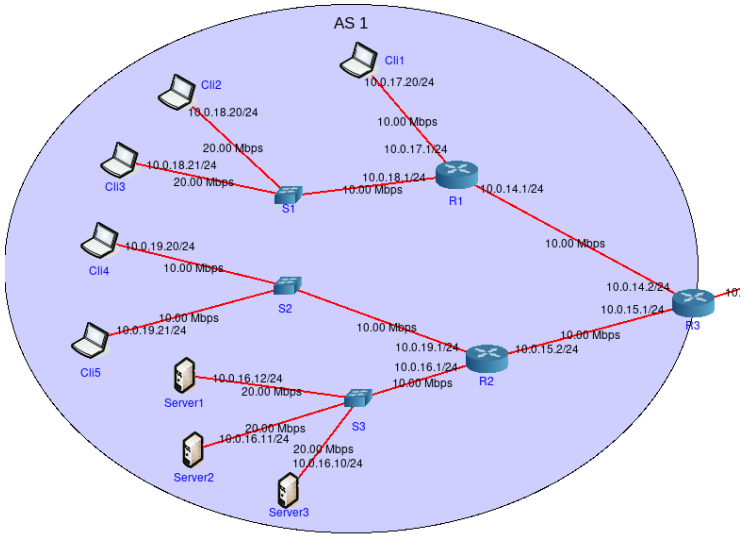
Each node on the network has a given purpose that is represented as a node label, and link labels are used to specify connection properties. Unless stated otherwise with these labels, all other properties are equal throughout the network. Some prepackaged CORE services need to be enabled to assure network connectivity - mainly *Open Shortest Path First Version 2 (OSPFv2)* and *Border Gateway Protocol (BGP)* - and scripting is used to, at the beginning of the simulation, bootstrap programs in specific nodes.



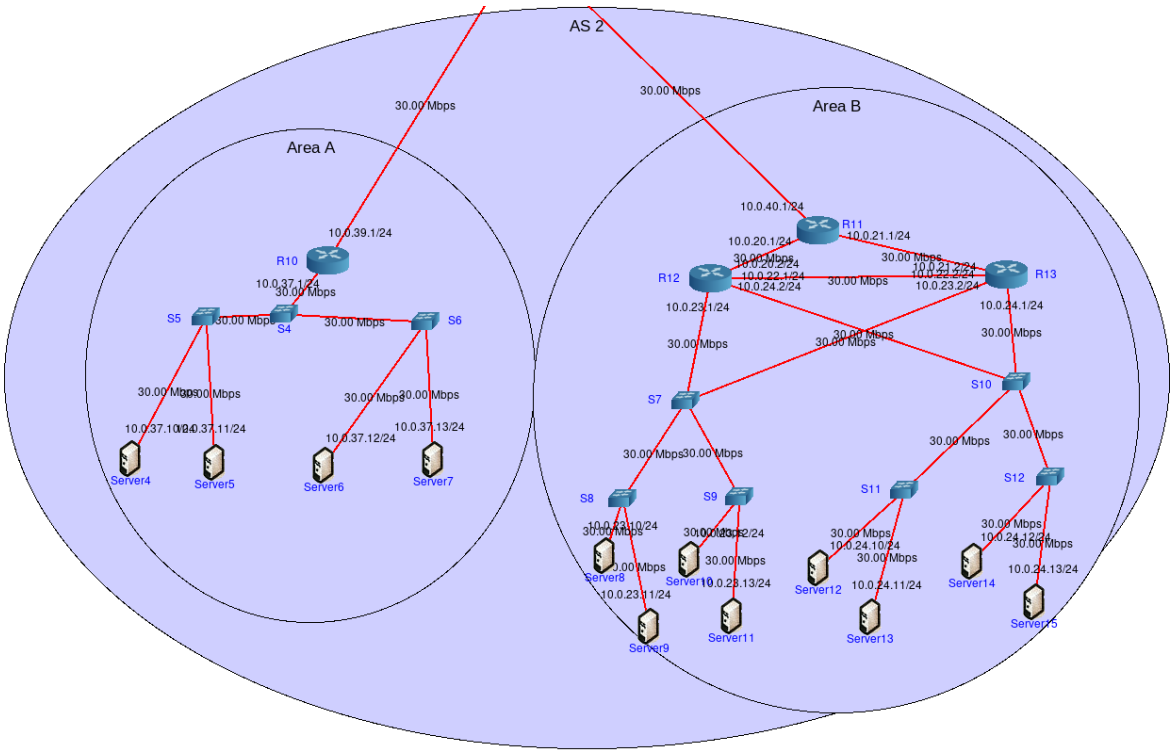
(a) Global view of the network



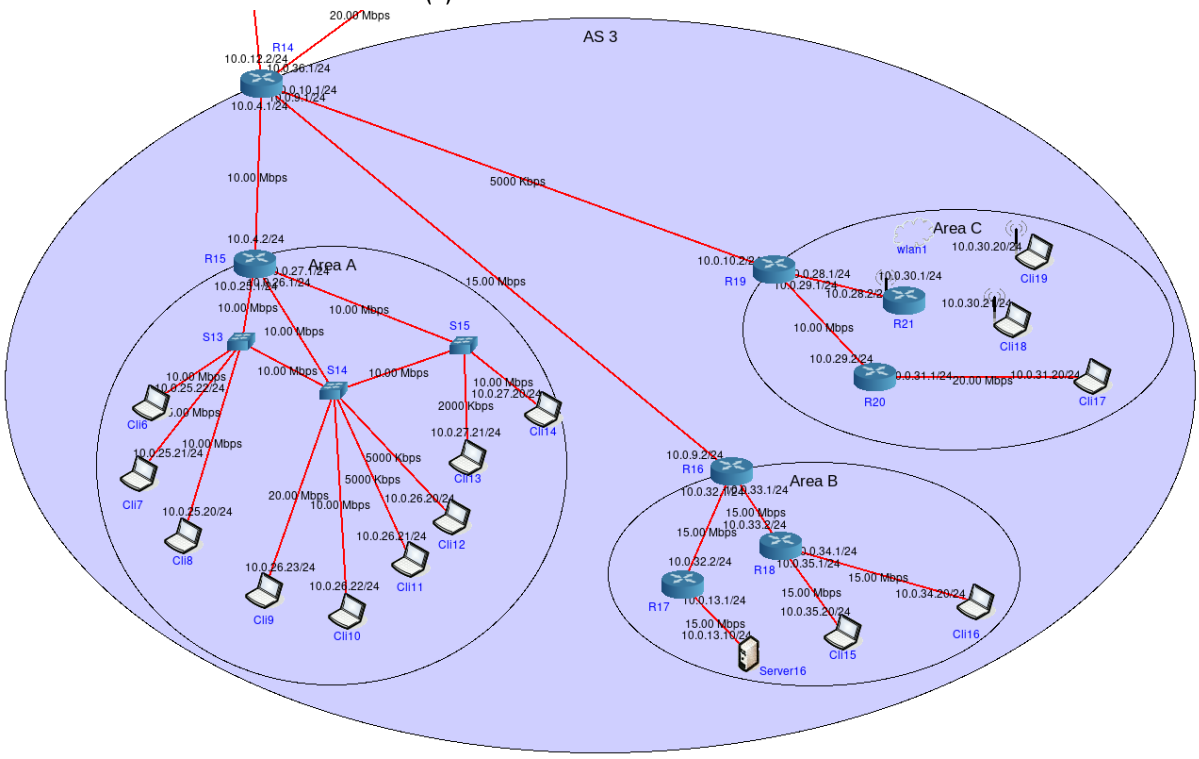
(b) Detailed view of AS0



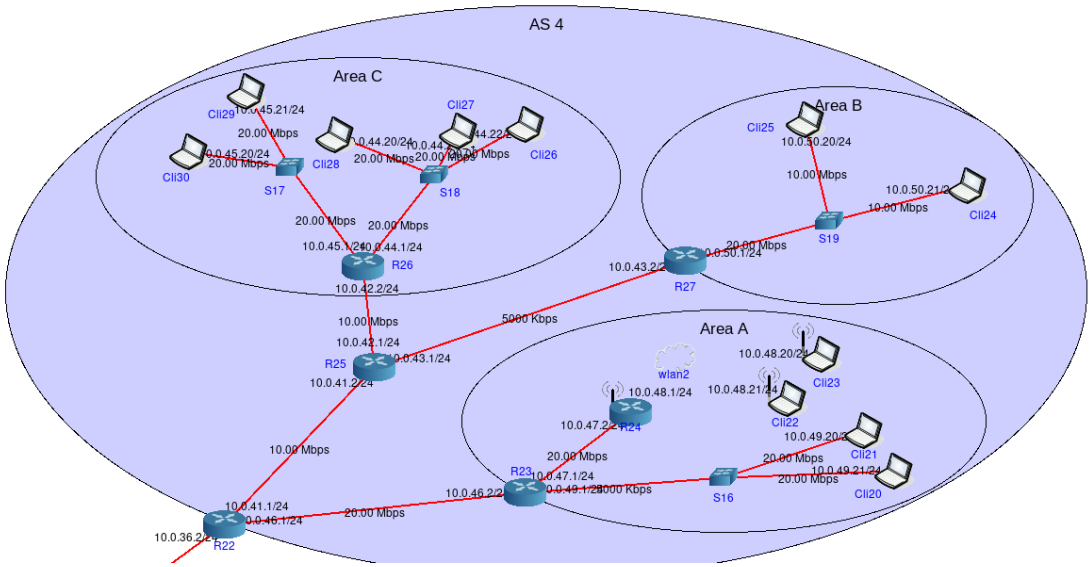
(c) Detailed view of AS1



(d) Detailed view of AS2



(e) Detailed view of AS3



(f) Detailed view of AS4

Figure 1.1: Network topology and integrated ASs

1.3 SCENARIOS

This Section describes the three devised scenarios to test the impact to the overlay and underlay that results of applications performing traffic optimization decisions with and without the implemented ALTO solution. Section 1.3.1 focuses on a P2P application who needs to decide, whenever a file fragment is served by multiple peers, with whom to establish connection. Section 1.3.2 tackles the issue of an HTTP server subjected to periodic load spikes, and how clients can selectively choose the timing to initialize the resource request. Section 1.3.3 deals with the challenge of selecting a given HTTP server to retrieve a content from whenever a mirror cluster is available.

1.3.1 Peer Selection in P2P File Transfer

1.3.1.1 Overview

All nodes in the network labeled as "CliN", with N from 1 to 32, actively serve all ten equally sized fragments of a 1GB file to other peers, and the overlay bootstrapping method includes informing the tracker, labeled as "Tracker", about what file fragments they serve.

"Cli2" wishes to retrieve the file, and to do so he firstly contacts the tracker for tracking information, which is given in the form of a file containing a mapping between each fragment *Identifier (ID)* and the available peers, alongside with the file's checksum for validation.

Table 1.1 shows what endpoints possess what file fragment IDs. For comparative reference, information is given about each fragment's location, i.e., the area and AS of the host serving it. Aditionally, the connection is also described with metrics from the perspective of the querying peer. These metrics include the maximum theoretical bandwidth as the minimum link bandwidth in the optimal path from Cli2 to each peer regarding this metric, and the sample *Round-Trip Time (RTT)* measurement the client obtained after 10 sequential ping commands prior to the scenario's start.

Fragment ID	Endpoint	AS	Area	Max bandwidth (Mbps)	Sample RTT (ms)
x00	10.0.18.21	1	A	20	0.172
x00	10.0.24.20	2	B	10	0.994
x00	10.0.35.20	3	B	10	1.174
x01	10.0.18.20	1	A	resides locally	0.057
x01	10.0.26.21	3	A	5	1.236
x01	10.0.48.21	4	A	10	12.238
x02	10.0.19.21	1	B	10	0.846
x02	10.0.24.21	2	B	10	1.014
x02	10.0.35.20	3	B	10	1.174
x03	10.0.24.21	2	B	10	1.014
x03	10.0.27.21	3	A	2	1.661
x03	10.0.31.20	3	C	5	1.456
x04	10.0.18.21	1	A	20	0.172
x04	10.0.26.23	3	A	10	1.111
x04	10.0.24.20	2	B	10	0.994
x05	10.0.19.20	1	B	10	0.820
x05	10.0.24.20	2	B	10	0.994
x05	10.0.24.21	2	B	10	1.014
x06	10.0.27.21	3	A	2	1.661
x06	10.0.48.20	4	A	10	12.278
x06	10.0.49.21	4	A	5	1.260
x07	10.0.48.21	4	A	10	12.238
x07	10.0.49.21	4	A	5	1.260
x07	10.0.50.21	4	B	5	1.616
x08	10.0.50.20	4	B	5	1.627
x08	10.0.50.21	4	B	5	1.616
x08	10.0.35.20	3	B	10	1.174
x09	10.0.25.22	3	A	10	1.170
x09	10.0.26.21	3	A	5	1.236
x09	10.0.31.20	3	C	5	1.456

Table 1.1: Description of each fragment to be requested

After retrieveing the mapping, the client will sequencially request each fragment from its selected peer, finalizing with the merge all the fragments into a single file and the calculation of the checksum that will be compared with the one previously provided.

An ALTO server resides in the same AS and maintains resources - specifically, a local network map that groups peers within locality as "Area A" of "AS1", "Area B" of "AS2", or externally to "AS1". It also maintains global resources - specifically, an

endpoint cost map indicating expected bandwidth and delay between all the peers participating in the overlay P2P network.

The variable actions to be tested are how the client selects which candidate peer, from the available pool, will be selected to serve a given file fragment. Table 1.2 displays the different algorithms to be used.

Tracker Algorithm	**Description**
Random	Randomly elect among all available peers
RTT	Select the peer with the smallest probe packet RTT measurement average in 10 pings
ALTO - Local	Retrieve the local network map from the requesting peer's ALTO server and discover the peers whose *Provider-Defined Identifier (PID)* matches the requesting peer, choosing randomly if multiple options exist within that group
ALTO - Global	Retrieve the global multicost endpoint cost map from the requesting peer's ALTO server by selecting the *Transmission Control Protocol (TCP)* throughput and one way delay metrics, and selecting the peer that maximizes throughput with a delay no bigger than 5 milliseconds.

Table 1.2: Peer selection algorithms to be tested in scenario 1

The experiment seeks to examine network resource usage and application performance for each algorithm. Table 1.3 presents the measurements that will be collected during the experiment runs.

Measurement	**Units**	**Description**
Execution Time	Seconds	Total amount of time required to select and retrieve all file fragments
Network Traffic	Megabytes	Total amount inbound traffic to each network AS and area

Table 1.3: Measurements to be taken in scenario 1

1.3.1.2 Analysis of Results

Figure 1.2 shows the execution times for each of the variable methods in this scenario. It appears that the method of probing for path delay was the worst, at 1096 seconds of runtime, closely followed by the method of randomly selecting between the available peers. Considerably faster are the ALTO-related options. The one considering a local server with its own information aided the application in achieving a 913.19 seconds execution time, whereas a global approach was able to considerably shave more time, at around 821.92 total seconds.

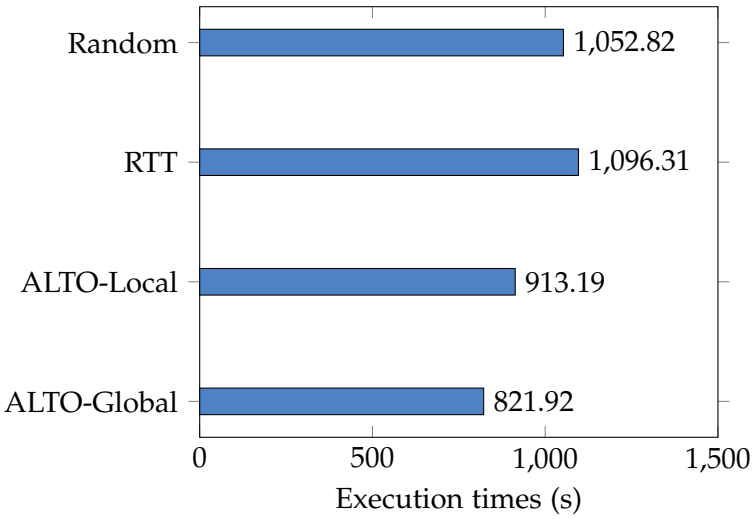


Figure 1.2: Execution times measured in scenario 1

Figure 1.3 show the measured traffic influx into the network's ASs and their areas. It can be immediatelly identified that the delay probing method incurred in considerable amount of inbound traffic being generated, in all ASs. A random approach reduced that value significantly, in particular the amount of traffic that entered AS1 and its areas which were almost reduced in half, but the ALTO approaches were both equally proficient and had the lowest amount of incoming traffic into AS1.

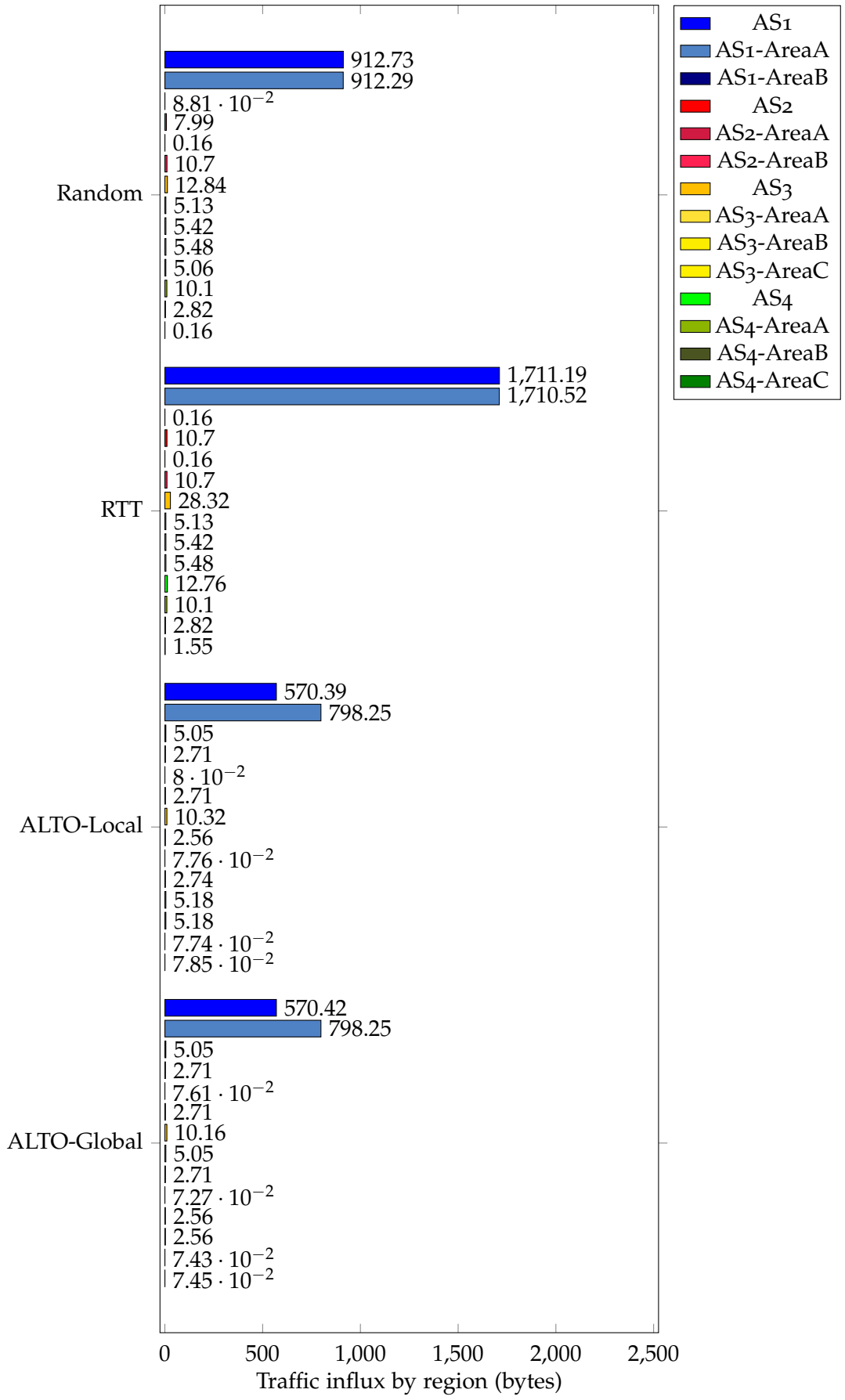


Figure 1.3: Inbound traffic flux by network areas measured in Scenario 1

It seems like a random selection of peers, as was expected, was far from optimal for execution time. Whilst random selection can be thought of as a good strategy to guarantee an even load distribution between peers, acting purely with this goal in mind is clearly not efficient in regards to network resources, as peers with worst network conditions can be chosen, and traffic can more often escape outside of network regions, as indeed was shown on the measurement of traffic that entered AS1, which meant a considerable amount of peers were selected outside of network regions. Perhaps surprisingly, a method of peer selection based on probing measurements targeting packet delay performed worse in regards to execution time and traffic locality. It appears that, for this topology and this overlay network, message delay was a bad indicator of application speed performance, and it favored a lot of peers that resided outside of local regions. The ALTO method revealed to be the most efficient at both runtime and traffic localization. A local approach was simple to implement, required input from only one *Internet Service Provider (ISP)* knowledge domain, and was able to get good time efficiency and very good traffic localization. The client had a prioritization mechanisms that favored area locality, followed by AS locality, and finally random choice if the prior criteria could not be fulfilled. The global approach had concrete information regarding available throughput and one way delay, but no strategy to localize traffic, instead maximizing throughput with delay that did not compromise its *Quality of Service (QoS)* levels. It appears, however, that there was a correlation between higher throughput and locality, and by focusing only on the former, the latter was automatically achieved. This can be seen because the global approach managed to generate the same amount of inbound traffic to AS1 while simultaneously reducing execution time.

1.3.2 HTTP resource request scheduling

1.3.2.1 Overview

"Server1" acts as a server of HTTP content. "Cli1" wishes to retrieve a 500MB file from that server in one single session. The server is subjected to variable, random client loads that affect processing and storage power, and subsequently its ability to serve clients, as well as periodic traffic loads within the AS whenever the server clusters in that system enchange data between themselves for server redundancy and general synchronization. This will be translated in practice as the dynamic variation of the available bandwidth of the link directly connected to the server. This has direct impact

on the theoretical maximum available bandwidth from "Cli1" to the server, whose chronological values are specified in 1.4:

Simulation time (s)	Max bandwidth (Mbps)
0 - 180	2
180 - 360	3
360 - 540	5
540 - ∞	10

Table 1.4: Dynamically applied client-server path delays in scenario 2

The server administrator is aware of the request loads that the server is subjected to and has maintained historical observations that existed in the past, which allowed him to reasonably predict how these will impact the server's performance in the future by detecting peak usage times. The ALTO server local to "Cli1" maintains a calendar cost map displaying the information mentioned in Table 1.4, i.e., it informs the client about the expected available bandwidth in the present and future.

The variable action to be tested is how the HTTP client selects when to initialize its content request with the server. Table 1.5 displays the different client algorithms that will be tested for the task of server request scheduling.

Client Algorithm	Description
Immediate	Immediatelly request the resource from the server
RTT	Continuously probe the network to measure the existing maximum available bandwidth until that value reaches above 10 Mbps
ALTO	Retrieve a calendar cost map from the ALTO server in AS2 of expected path delay to the server, and iniciate immediately when the available bandwidth is expected to be 10 Mbps

Table 1.5: Client algorithms to be tested in scenario 2

Measurements: Table 1.6 presents the measurements that will be collected during the experiment runs.

Measurement	Units	Description
Transfer Time	Seconds	Total amount of time required to transfer the file
Network Traffic	Megabytes	Total amount of traffic that passed through each area in AS1

Table 1.6: Measurements to be taken in scenario 2

1.3.2.2 Analysis of Results

Figure 1.4 shows the obtained transfer times for each of the variable methods in scenario 2. It appears that the ALTO-aided and delay-probing methods were equally circling a 640 second runtime, and an approach of immediately querying the server was faster, at 457.19 seconds.

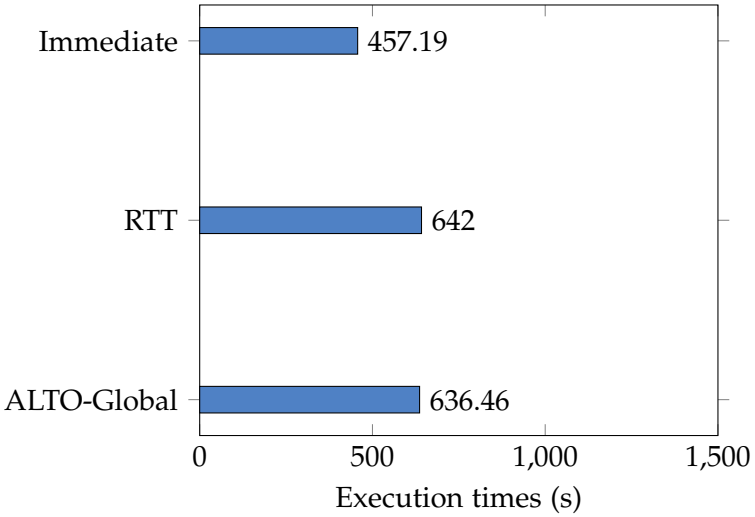


Figure 1.4: Execution times measured in scenario 2

Figure 1.5 shows the measured traffic influx into the areas of AS1. It appears that the method that used bandwidth probing mechanisms took a considerable traffic overhead compared to the other two alternatives.

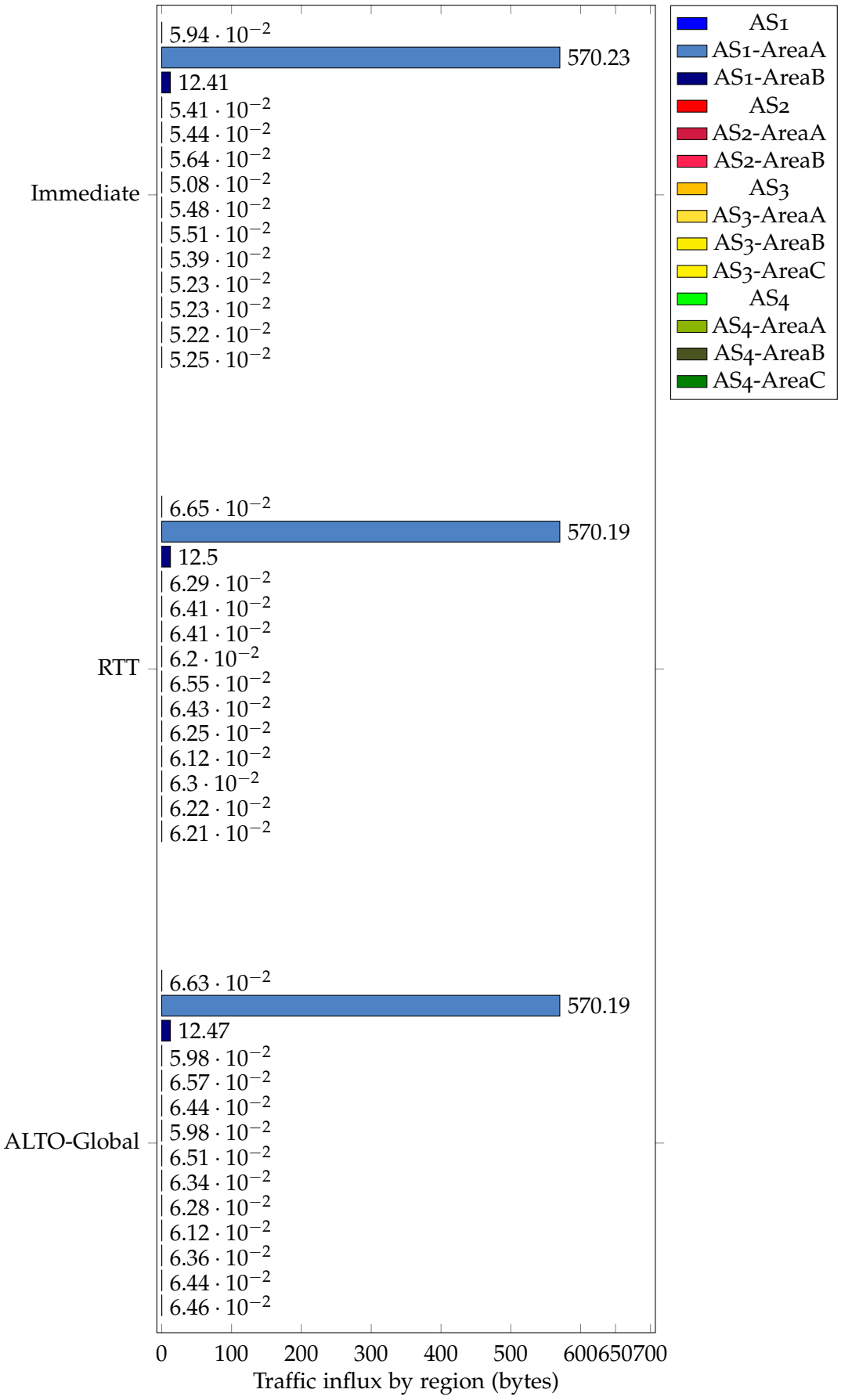


Figure 1.5: Inbound traffic flux by network areas measured in scenario 2

It seems to be that, unsurprisingly, query timing had a big impact on total file transfer time, as these would set the available bandwidth during file transfer. Both the probing and ALTO-aided strategies were able to correctly identify when the server was least overwhelmed by other clients, which justified their improved time results in comparison to immediately querying the server. However, these do not have similar network impacts. Firstly, the maximum bandwidth measuring approach required that the server hosted a measuring tool that the client could query to deduce network behavior, and this is a luxury that does not exist in most real-case scenarios. Secondly, deducing that information without ISP input required that a massive amount of overhead data be generated on a network, that not only used network resources in a sub-optimal manner, but it also added to the increase of the server's already overloaded capacity, which would, in a real-case scenario, further decrease the server's ability to handle client requests. One could note that, in alternative to continuously probing the network, the client could use some exponential backoff strategy as to space out consecutive probing measurements, but that would, at the very least, generate an undesirable amount of overhead traffic, and could never be more quick than retrieving historical server pattern information from a reliable source. This last method was exactly what the ALTO approach did, which was able to nullify any need to generate overhead traffic besides a very small request for the ALTO resource, and simply waited to retrieve the content when the server was most available to, benefiting both sides and improving to a more sustainable server-client architecture and the health of the entire network.

1.3.3 HTTP mirror selection

1.3.3.1 Overview

Description: An HTTP client wishes to retrieve a 500MB file from a server. After querying the public proxy, an index is provided which contains a address lists of the four mirror servers that provide that content. The listing is show in table 1.7

Server address	AS	Area
10.0.16.12	1	B
10.0.37.10	2	A
10.0.23.11	2	B
10.0.13.10	3	B

Table 1.7: Available server mirrors in scenario 3

The mirror servers will be subjected to constant loads at the start of the scenario, that are translated as throughput throttling applied from the client to the servers, specifically the ones shown in 1.8.

Server address	Throughput
10.0.16.12	3.0 ms
10.0.37.10	5.0 ms
10.0.23.11	10.0 ms
10.0.13.10	5.0 ms

Table 1.8: Available path throughput from client to mirror servers in scenario 3j

The ALTO server local to that client provides a global ALTO endpoint property map that contains *Central Processing Unit (CPU)* and *Random-Access Memory (RAM)* load information about all the mirrors, as well as an endpoint cost map containing information about the expected bandwidth and delay from the client to the mirror.

Variables: The variable action to be tested is how the HTTP client selects which mirror server to retrieve the file from. Table 1.9 displays the different client algorithms that will be tested for the task of mirror selection.

Client Algorithm	Description
Random	Randomly select a server
Bandwidth	Probe for available TCP throughput to the mirrors, and select the one with most bandwidth
ALTO	Retrieve an endpoint property map and endpoint cost map from the local ALTO server, and choose the with most available bandwidth and delay below 2ms, whose memory and processing power is below 30%

Table 1.9: Client algorithms to be tested in scenario 3

Measurements: Table 1.10 presents the measurements that will be collected during the experiment runs.

Measurement	Units	Description
Transfer Time	Seconds	Total amount of time required to transfer the file
Network Traffic	Megabytes	Total amount of traffic that passed through each area network and AS

Table 1.10: Measurements to be taken in scenario 3

1.3.3.2 Analysis of Results

Figure 1.6 shows the obtained execution times for each of the variable methods in scenario 3. An approach of randomly choosing mirror servers had the worst performance time wise, with 912.39 seconds, and the TCP throughput measuring method following with 496.57 seconds. Finally, the ALTO-aided approach was the quickest with 456.21 seconds of runtime.

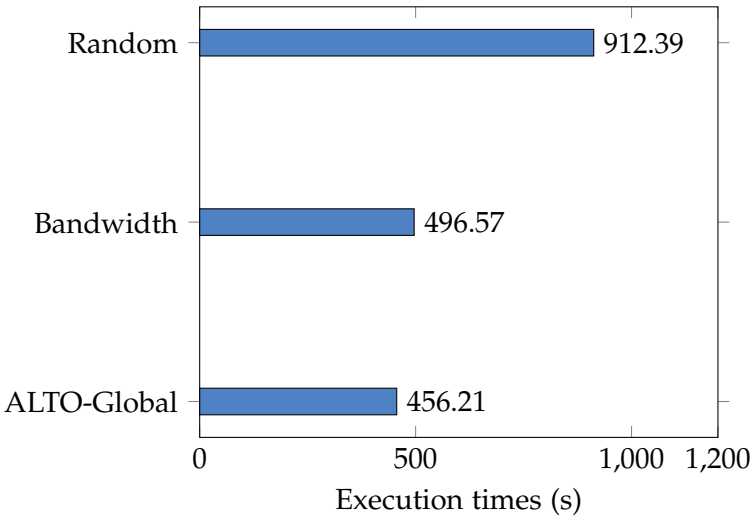


Figure 1.6: Execution times measured in scenario 3

Figure 1.7 shows the measured traffic influx into the network's ASs and their areas. It appears that the approach that probed for path bandwidth had a considerable impacts on the generated traffic influx, and the spikes of traffic in the random and ALTO approach give clues into the selected mirrors - those residing in AS3-B and AS2-B, most specifically.

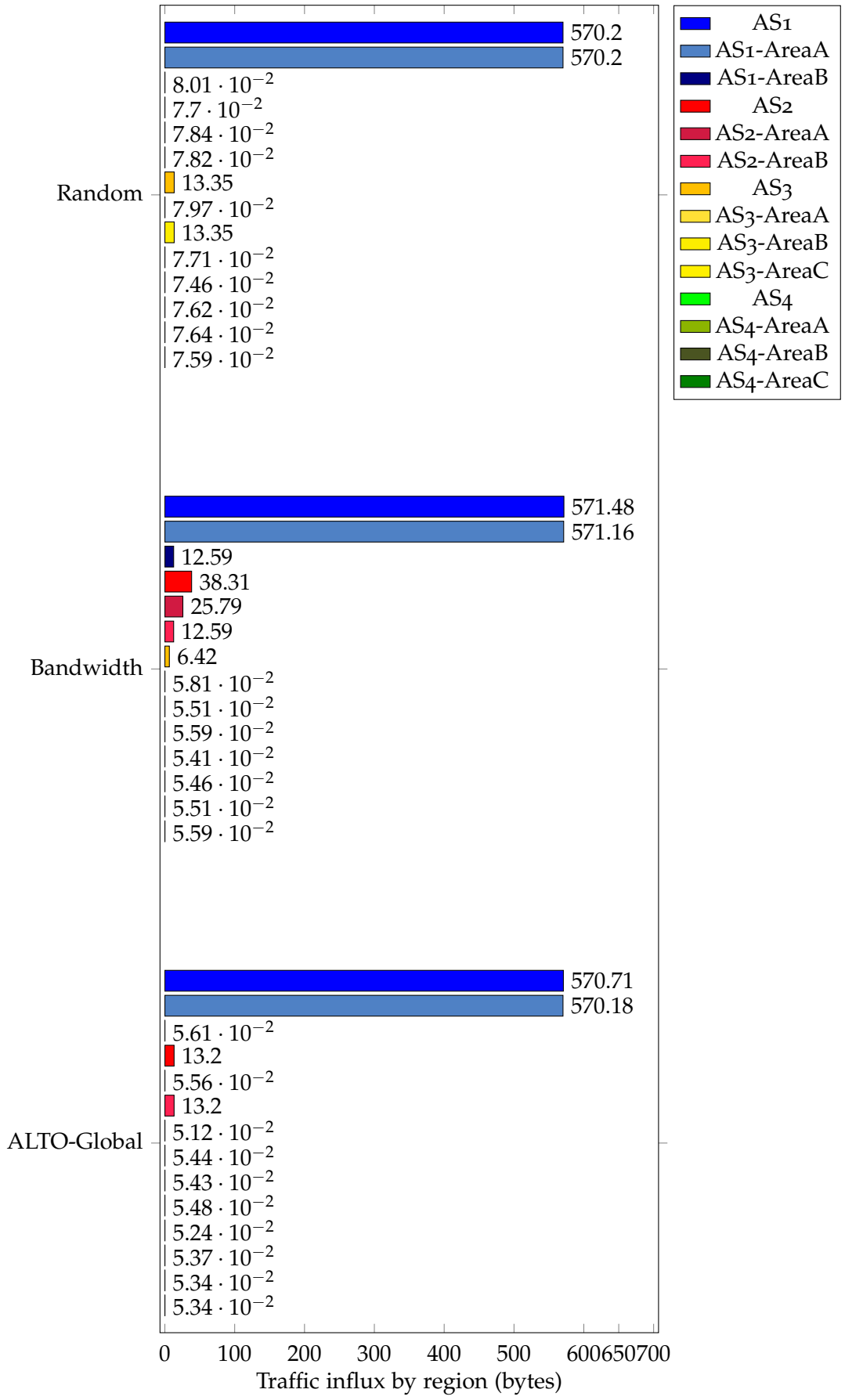


Figure 1.7: Inbound traffic flux by network areas measured in scenario 3

Regarding scenario 3, it seems like the random mirror selection had the worst runtime, probably due to the fact that most of the servers were under load, and randomly selecting the best one, over many runs, was unlikely. A throughput probing approach was able to select the most apt mirror server to provide the server, but at the expense of much traffic overhead required to deduce available throughput. Additionally, this approach required from all mirror servers the hosting of a probing server that made these measurements possible, a luxury not possible in much real-case scenarios. The ALTO approach was able to get immediate ISP input in the form of a combined multi-ALTO domain effort, and quickly query the optimal mirror, resulting in a slightly better runtime than a probing approach.

BIBLIOGRAPHY

[1] Common open research emulator. https://www.nrl.navy.mil/itd/ncs/products/core. Accessed in: 2020-09-20.

[2] vcmd. http://manpages.ubuntu.com/manpages/trusty/man1/vcmd.1.html. Accessed in: 2020-09-20.

[3] Python. https://www.python.org/. Accessed in: 2020-09-20.

[4] awk. https://www.gnu.org/software/gawk/manual/gawk.html. Accessed in: 2020-09-20.