Towards Bandwidth Optimization in Fog Computing using FACE Framework

Rosangela de Fátima Pereira Marquesone[1], Érico Augusto da Silva[1], Nelson Mimura Gonzalez[1], Karen Langona[1], Walter Akio Goya[1], Fernando Frota Redígolo[1], Tereza Cristina Melo de Brito Carvalho[1], Jan-Erik Mångs[2] and Azimeh Sefidcon[2]

[1]*Laboratory of Computer Networks and Architecture, Escola Politécnica, University of São Paulo, Brazil*

[2] *Ericsson Research, Sweden*

Keywords: Fog Computing, Bandwidth Optimization, Video Surveillance.

Abstract: The continuous growth of data created by Internet-connected devices has been posing a challenge for mobile operators. The increase in the network traffic has exceeded the network capacity to efficiently provide services, specially for applications that require low latency. Edge computing is a concept that allows lowering the network traffic by using cloud-computing resources closer to the devices that either consume or generate data. Based on this concept, we designed an architecture that offers a mechanism to reduce bandwidth consumption. The proposed solution is capable of intercepting the data, redirecting it to a processing node that is allocated between the end device and the server, in order to apply features that reduce the amount of data on the network. The architecture has been validated through a prototype using video surveillance. This area of application was selected due to the high bandwidth requirement to transfer video data. Results show that in the best scenario is possible to obtain about 97% of bandwidth gain, which can improve the quality of services by offering better response times.

1 INTRODUCTION

The widespread adoption of smartphones, many mobile devices and sensors from the Internet of Things (IoT) has been creating a world with ubiquitous connectivity. According to a report published by Ericsson, only in the year of 2014, 800 million smartphone subscriptions were created worldwide. Consequently, it is expected a total of 5.4 billion subscriptions of mobile broadband by 2020 (eri, 2015).

Mobile subscribers take full advantage of existing wireless network infrastructure, which allows them to enjoy with a wide range of applications, including video streaming, gaming and social networking. These applications require faster response times to provide efficient and satisfying experience to the end user. However, the rising use of these applications results in a continuous growth on the mobile data traffic. As a consequence, the data traffic has been exceeding the capacity of current network infrastructure, impacting application performance and network efficiency.

As the data volume across the network increases, application service providers are being challenged to scale their data center with properly storage and processing capacity. In this context, cloud computing has been used to support the large volume of data consumption by end users. Cloud computing allows service providers to obtain computing and storage resources on demand, providing them with flexibility and dynamic resource allocation. However, even in a cloud infrastructure, the data are generated or terminated in a central location. For this reason, cloud computing cannot solve the problem related to network performance, since the data must be transferred through the network until it finds its destination. Therefore, the more mobile devices and applications are made available, larger and faster network infrastructure becomes necessary.

An approach capable of reducing data traffic across the network is to process the data before it reaches its destination. Edge computing can be a promising way to address this requirement. It consists of storage and computing resources allocated as closer as possible to the edges of the network. This approach makes possible to process the data as soon as it reaches the network, thus reducing the amount of data that must be sent to the final destination. Therefore, edge computing allows the execution of many applications on intermediary nodes, instead of only in their central location. Fog and Edge computing are interchangeable terms according to some authors (Gonzalez et al., 2016; Dastjerdi and Buyya, 2016; Hu et al., 2016).

Given the computing capability offered by fog computing, it is necessary to investigate how to enhance network performance from it. Considering this context, in this paper we propose the FACE. It offers a framework with functionalities capable of either reducing or optimizing the network consumption. The functions provided are (F)iltering, (A)ggregation, (C)ompression and (E)xtraction. In this paper we explained the definition and objectives of each function and also present potential scenarios where FACE can be applied. In order to test the capabilities of our solution, we performed an experiment using FACE framework in a video surveillance application. This context was selected due to the large amount of data generated by video cameras and due to the need of such applications for low latency. Results show that FACE framework provides significant reduction of data traffic, and it can be applied in the network infrastructure in a transparent way.

The remainder of this paper is organized as follows. In Section 2 we present a contextualization about fog computing, describing its definition and distinctions of cloud computing. The description in detail of the FACE framework is presented in Section 3. In Section 4 is described our experiment with a video-surveillance use case. The tests, results and analysis are presented in Section 5, and finally, Section 6 brings the conclusions and future work.

2 FOG COMPUTING

The current wireless network infrastructure provides the connection between the end users and the data center from application providers. A user of a mobile phone when performing a search using a browser on its device counts on the wireless network infrastructure to upload her/his request and deliver it to the destination. The response to the user's request will be brought to her/his device by the wireless network infrastructure. Therefore, the request from the user has to reach the central server in order to provide the response to the user. The data generated by an end user on the edges of the wireless networks will have to travel through the network until it finds its destination. The problems with this centralized processing (conventional scenario) are:

- Delay insertion to the response time. This occurs due to the time that it takes for the request to be transferred from the user device to the central server and then back, to bring the response to the end user. This time can compromise the performance of latency-sensitive applications such as video streaming and gaming. Therefore, the faster that a response can be issued to the end user, the better the network meets the application response time requirements.
- Traffic addition to the network. Since all the end-users requests have to be transferred through the network until it finds its central server, more traffic is added to the network. With the growing number of smartphone subscriptions, the telecom operators will have to provide networks that support more traffic and more users.

From these two observations, we can conclude that telecom operators need to find innovative approaches to support the continuous growth of data traffic. Therefore, fog computing emerges as an alternative to meet this requirement (Margulius, 2002; Pang and Tan, 2004). Considered an extension of cloud computing paradigm, this concept was defined for the first time in 2012 by the work of Flavio Bonomi et al. (Bonomi et al., 2012) According to the authors, fog computing is a virtualized platform that brings cloud computing services (compute, storage, and network) closer to the end user, in the edges of the networks. By offering these services, the platform can better meet the requirements of time-sensitive applications. It implies that the cloud computing resources will be geographically distributed to locations closer to the end users and/or IoT endpoints. This distributed infrastructure allows to get bandwidth savings by applying data processing across intermediary processing nodes, rather than only in the cloud.

3 FACE FRAMEWORK

The FACE framework consists of a set of specific functions that aims at improving response time and avoiding sending unnecessary traffic through the network. It was designed to perform data processing in the edge or intermediary nodes of the network. As depicted in Fig. 1, the functions can be hosted in a box composed by compute and storage resources. This box may be placed either at the base station level (setup 1) or right after it (setup 2). In both cases the box is added to the network topology and is responsible for intersecting the data, applying the functions to data processing, and delivering the data results to the next component of the network.

To reduce the data traffic, decrease overhead and allow better management of the network bandwidth, this framework is composed by four different functions: Filtering, Aggregation, Compression, and Extraction. The name FACE is an acronym formed by the first letter of each function name. They can be either individually or jointly applied to the input traffic. This choice is determined by each individual use case.

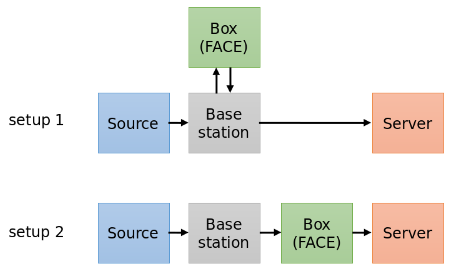

Figure 1: The box and the two network topologies.

In this section we describe the definition of each function and also present potential scenarios that may benefit from the FACE functions.

Filtering

The filtering function allows reducing data traffic by either analyzing or processing the data at the box. All data collected are computed by this function and only the data that satisfy a group of predefined filtering criteria will be sent on the network. The criteria consist of measures and/or attributes defined by the application service providers. Therefore, the filtering function offer bandwidth savings by excluding data before it reaches the central server. In addition, the same approach can be used for data cleaning mechanisms, by defining criteria to detect and discard inconsistent and corrupted data.

Figure 4 illustrates an use case that applies the filtering function on pictures taken by surveillance cameras installed along a freeway. The monitoring devices help control the average speed allowed on the freeway. For this example it was assumed that the surveillance is done by eight installed cameras. As the car passes by the first set of cameras a picture is taken at P1. When the car passes by the second set of cameras another picture is taken at P2. Then the function calculates the average speed and if it is over the limit for that freeway the pictures will be sent to the central server to be processed and a ticket will be issued to that car. Assuming the size of a picture is 30KB, each set of eight cameras would send 240KB to the network. However, if we assume that the infraction rate on the freeway is 15%, the filtering function will only allow the 15% of the data to be carried over the network. The data that is collected on the edges of the network is then filtered and only 15% of it is sent to a central location.

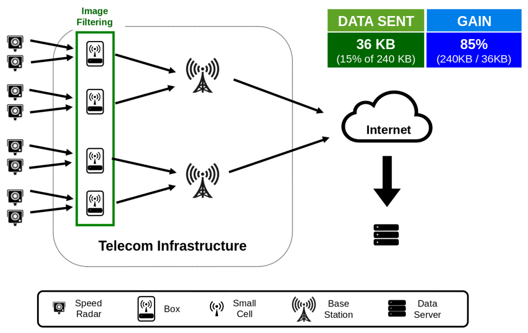

Figure 2: The gains of applying the filtering function.

As stated before, the filtering function can be applied to scenarios where parts of the data can be discarded, in case a set of criteria is not satisfied. An example could be the data collected from sensor networks used to control a device or an environment, e.g. temperature sensor to prevent fires. Using the filtering function, only the data that present abnormal event will be sent to the final destination, allowing to substantially reduce data traffic.

Aggregation

The aggregation function reduces the overhead on the network since it combines and accumulates data to be sent through the network in batches. Therefore, a single connection can be used to send data from multiple users. This solution is suitable for scenarios that collect large amount of small data in frequent times. If the application is not latency sensitive, the aggregation function can store the data and send it only in predefined time intervals. Considering the previous scenario, the surveillance cameras in a freeway, the pictures taken at P1 and P2 do not need to be processed in real-time. It is possible to archive the pictures using the aggregation function and send batches of them in predefined intervals.

While the filtering function requires processing capacity from the box, the aggregation function requires storage capacity to cache the data. Triggering mechanisms can be used to define rules regarding the data delivery. This trigger can also be turned off when the amount of data is lower than usual. For example, applications that concentrates higher data traffic only during the day, do not need to aggregate the data at night. Therefore, it is necessary to investigate which compression mechanism better reduces the data without generating low responses.

Compression

Although the filtering function allows reducing the amount of data, it is not applicable when applications require to store all the data in a central server, or in cases when it is not possible to define filtering criteria for the data. A different strategy to reduce the data traffic is to use the compression function. This function enables the execution of compression mechanisms according to the data structure. For example, pictures collected from video surveillance cameras can be compressed by converting the raw photos to JPEG format.

In order to perform further processing over the data in the central server, service providers must be advised about which algorithm was used for data compression. By knowing the algorithm they can decompress the data. A service provider of video streaming could add its own compression algorithm to the box, so data from video uploads could be reduced.

Due to the large alternatives to compress the data, the execution of compression function can represent a trade-off. While it allows reducing the data size, and consequently reducing data traffic, some algorithms may aggregate an additional overhead during the compression, impacting the performance of latency-sensitive applications.

Extraction

The extraction function, similar to the filtering function, focuses on mechanisms to discard unnecessary data. However, this function enables the data reduction by extracting only an specific portion of the data. For example, considering the aforementioned scenario that takes pictures of a car to calculate the average speed. The extraction function can be used to extract the license plate information from the picture and send to the server only the text with this content, instead of the whole picture. Since the text file with only this specific portion contains just a few kilobytes, the data traffic can be reduced.

Similar to the compression function, the service providers may determine the application logic in the extraction function. There are many approaches to it. Text mining algorithms can be applied on text data, and only the output data from this computation can be sent through the network. Pattern recognition techniques and image segmentation can also be applied in the box, allowing to determine the application logic over the data in a faster response time. The data sent to the server are the one that must be recorded.

The FACE functions can also be jointly applied to the input traffic. When two or more functions are applied to the input traffic, the data reduction can be even higher. These possibilities result in many benefits, such as reduce costs related with network infrastructure for Telecom operators, optimize the performance of application service providers and improve end users experience. The experiment and evaluation of FACE framework is presented in the next section.

4 PROPOSED ARCHITECTURE

In order to perform an experiment to analyze the performance of the FACE framework, we designed an use case applied to video surveillance. We demonstrate our strategy to apply the compression function over image data in a box closer to the base station. The tests were performed to identify the bandwidth savings by applying this function and to evaluate the overhead to intersect, apply the processing and deliver the data.

Fog computing applied to video surveillance can improve the performance of the system by reducing the amount of data transmitted across the network. This reduction allows improving the application response times, since the transmission time is decreased. The overall response time of the system can also be improved by the bandwidth optimization. A strategy to decrease the amount of data in a video surveillance scenario that does not require high quality images is to convert the images to grayscale mode and resize them to a lower resolution. Therefore, before the video data reaches a cloud infrastructure or a data center to be processed, these optimizations can be applied in a modular compute and storage device, which is located closer to where the images have been generated (e.g. cameras). Considering video surveillance systems composed by several cameras, this approach can result in significant bandwidth reduction and high quality services.

In a traditional video surveillance scenario, the source continuously sends images (`IMG.png`) directly to a server. The system must provide an online view of the scenes, specially for the ones that presents some activity (Nilsson and Axis, 2017). As illustrated in Figure 3, we designed an architecture that intercepts the images from the source, redirecting them to the Processing Node (PN), where, if applicable, the compression functions are applied, and only then the images are sent to the final destination. This model applies compression to a given image if there is a difference between this image and the previous one.

We proposed two ways to connect the PN to the rest of the system: setup 1 connects the PN to the

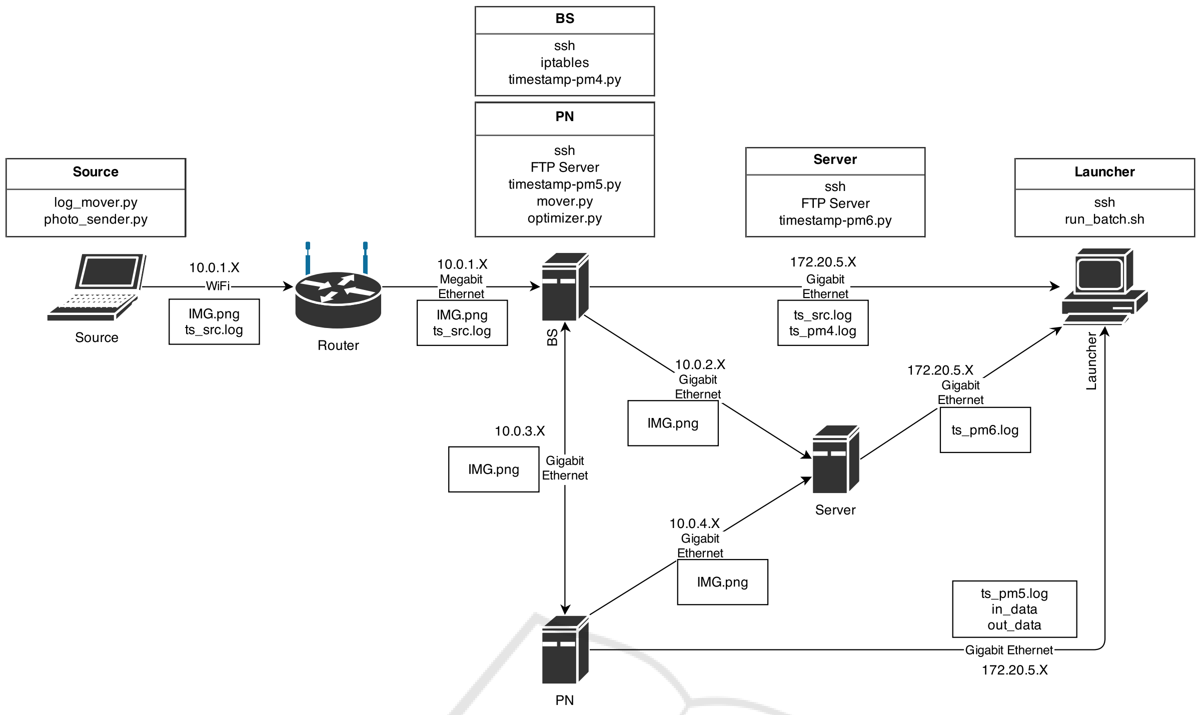

Figure 3: System architecture to apply compression function closer to the network.

Base Station (BS) only, the PN receives images from the BS and sends them back, so BS can send them to the server; setup 2 is a connection in series, where PN is connected between the BS and the server, so the PN receives images from BS and send them directly to the server.

Data redirection is done through `iptables` configuration. The `iptables` are an administrative tool for IPv4 packets filtering and NAT (Network Address Translation) and they are used to control packets in transit within the machine through rules definitions. The compression takes place in the PN and it is done by the `optimizer`. This component is responsible for comparing two images and deciding whether it should apply compression. If so, the `optimizer` lowers the image resolution and converts it to greyscale, which reduces its size.

Within the PN there are the `local_mover` and `remote_mover`. These mechanisms are responsible for moving images within the machine file system and starting the transfer between PN and server. These mechanisms implement time-based differentiation algorithm, which allows new files detection and the subsequent processing of the previous ones. The idea behind it is that if there is a new file listed in the file system, the old ones are safe to be processed, without any synchronization risks or race conditions. The system generates two types of log files: timestamp logs and bandwidth log. The first one extracts latency metrics and the second one evaluates the bandwidth gain by keeping track of the size of incoming and outcoming images.

5 TESTS AND RESULTS

In this section we present the testbed setup, the test description and the metrics used in the prototype. We also present results obtained from a set of experiments performed to evaluate the effectiveness of the FACE framework.

5.1 Testbed Setup

According to the architecture presented in Fig. 3, the testbed is composed of one notebook with camera (source), one wireless router, three physical machines (BS, PN and server) and one PC (launcher), connected to the laboratory network infrastructure. Table 1 shows the hardware specs used in the testbed. The wireless router is a Netgear WPN824 v3 and we use Secure Shell (SSH) to establish access to the hosts.

We used the laptop webcam to run the test because we needed a custom application capable of collecting some of the latency metrics. However, in a real video surveillance scenario any type of camera with image transfer capabilities could be used.

Table 1: Testbed hardware specs.

Host	Processor	RAM
Source	Intel Core i7-3635	8 GB
BS	Intel Xeon E3-1230	16 GB
PN	Intel Xeon E3-1230	16 GB
Server	Intel Xeon E3-1230	16 GB
Launcher	Intel Core i5-2500	8 GB

5.2 Tests Description

In order to identify the latency introduced by our solution and the amount of bandwidth gain it can generate, we have used the metric files collected from the testbed hosts. These files allowed us to identify the amount of time consumed by each operation performed during the tests.

The tests were performed with five different scenarios:

- Test 0: is the conventional scenario, without the processing node;
- Test 1A: uses the processing node in the Setup 1, without compression functionality;
- Test 1B: uses the processing node in Setup 1, but now with compression functionality;
- Test 2A: uses the processing node in the Setup 2, without compression functionality;
- Test 2B: uses the processing node in Setup 2, but now with compression functionality;

Regarding latency measurements, each machine listens to a set of events, which are described in Table 2. When triggered, these events generate entries in the timestamp log, containing the event type and the time they occurred. These two pieces of information are used to extract the amount of time consumed by each step. We used Network Time Protocol (NTP) to ensure the machines had their clocks synchronized.

PN is responsible for accounting the amount of data received and sent. Once the image transfer between source and PN is completed, it registers the image size in `in_data` file and, before sending the image to server (after processing it), in registers the image size in `out_data` file.

Algorithm 1 describes the steps for setting up the testbed. The launcher runs code that sets the network topology and triggers applications in the other machines, making remote call with `ssh`. These applications remove files generated in the previous test and also collect the metrics for the current test, according to the current topology. Once all applications are running, the launcher waits for the test to be completed, collects all metric files and generates the analysis.

There are two environments considered: wired and wireless. The first one is used as a reference for system architecture validation and the second one is the target environment. We set the sending rate based on preliminary network analysis that showed that the network could only handle one image per second. We repeat each test ten times. For both test environments, we used two test loads: no movement and movement. The first one gives the results for the best case scenario for the application, since it implies intensive use of compression function. The second one gives results on the worst case scenario.

5.3 Results

Table 3 and 4 display the results from the tests performed on the wired and wireless environments, respectively. The tables contain all the average latency for each operation, the total latency and the bandwidth gain.

In both environments, the latency values for the two topologies proposed are similar, even when compared among different test loads. The PN introduced about 2.37 seconds of latency in the wired environment and 5.12 seconds in the wireless, but only 0.03 seconds have been spent with compression. The rest of the time has been spent with disk operations, such as move and copy within the PN.

Most of the time operations was spent with image transmission: before the compression starts, the image transfer must be completed, and before the image can be sent to the final destination, the image compression must be complete. These rules are implemented through the mover time-based differentiation algorithm.

Algorithm 1: Test execution algorithm.

```
 1: for each scenario in scenarios_list do
 2:     SETUP_TOPOLOGY(scenario)
 3:     for i ← 0 to n_repetitions do
 4:         CREATE_LOCAL_METRICS_FOLDER
 5:         for each host in remote_hosts do
 6:             CLEAN(host, images_folder)
 7:             CLEAN(host, metrics_folders)
 8:             RUN_PROGRAMS(scenario, host)
 9:         end for
10:         SLEEP(test_duration)
11:         for each host in remote_hosts do
12:             COLLECT_DATA(host)
13:         end for
14:         GENERATE_LOG_ANALYSIS
15:     end for
16: end for
```

Table 2: Timestamps description.

Timestamp	Host	Event	Program
T1	Source	Image is created.	`photo_sender.py`
T2	BS	Receives packet requesting FTP transfer for server.	`timestamp.sh`, `read_stamps.py`
T3	BS	Redirects packet requesting FTP transfer to PN.	`timestamp.sh`, `read_stamps.py`
T4	PN	Start receiving the photo in its FTP income folder.	`timestamp-pm5.py`
T4A	PN	Mover is called to move the image from the income folder to optimizer input folder.	`mover.py`
T5	PN	Image is transferred for optimizer input folder.	`timestamp-pm5.py`
T5A	PN	Optimizer starts the optimization process.	`camera_v1.py`
T5B	PN	Optimizer finishes the optimization process.	`camera_v1.py`
T6	PN	Image is transferred to optimizer output folder.	`timestamp-pm5.py`
T6A	PN	Mover is called to start FTP transfer to BS (topology 1) or server (topology 2).	`mover.py`
T7	Server	Start receiving the image in its FTP income folder.	`timestamp-pm6.py`

Table 3: Wired test results.

(in seconds)		**BS**	**PN**								**Server**		
Test	**Mov**	T2-T1	T3-T2	T4-T3	T4A-T4	T5-T4A	T5A-T5	T5B-T5A	T6-T5(B)	T6A-T6	T7-T2(6A)	Latency	Gain
0	Yes	4.297									0.002	4.299	0
1A	Yes	4.340	4.66E-6	0.001	1.046	0.003			0.003	1.006	0.185	6.584	0
1B	Yes	4.338	4.53E-6	0.001	1.055	0.003	0.055	0.004	0.003	1.017	0.184	6.659	0.01
2A	Yes	4.332	4.39E-6	0.003	1.057	0.003			0.003	1.011	0.185	6.593	0
2B	Yes	4.338	4.02E-6	0.006	1.214	0.007	0.060	0.004	0.003	1.051	0.186	6.867	0.01
0	No	4.222									0.002	4.224	0
1A	No	4.267	4.61E-6	0.000	1.045	0.003			0.003	1.007	0.185	6.509	0
1B	No	4.269	4.59E-6	0.013	1.134	0.004	0.057	0.004	0.000	1.030	0.184	6.695	0.97
2A	No	4.265	4.51E-6	0.004	1.074	0.003			0.003	1.015	0.183	6.548	0
2B	No	4.262	4.03E-6	0.002	1.060	0.002	0.054	0.004	0.000	1.021	0.184	6.588	0.96

The sending rate was set to one image per second in both environments. The wired network was able to handle this rate and the images were moved at almost the same rate they were created. On the other hand, the wireless network was only able to handle one image every 2.34 seconds, approximately, which resulted in higher latency value. Table 5 shows the comparison between the two environments.

Figure 4 shows the time ratio for each operation inside of the PN (application of the compression algorithm over data). As mentioned before, the main cause for the latency was the characteristic of our solution, which creates a dependency between the latency and the image receiving interval.

Although the system showed a latency insertion in the data transmission, it also presented a major benefit in the bandwidth consumption. Considering the best scenario, where no movement was detected, in comparison to the traditional scenario the compression function resulted in about 97% of bandwidth gain.

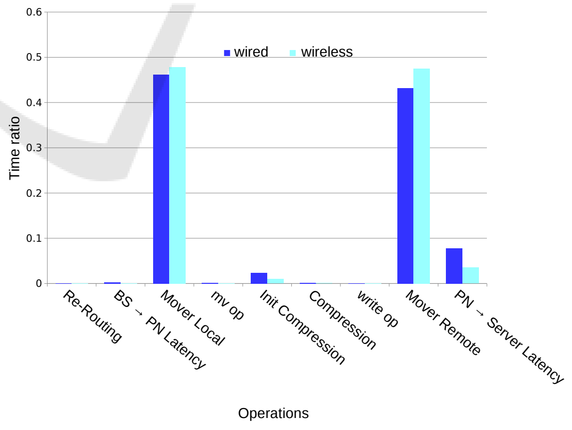

Figure 4: Processing node time composition.

6 CONCLUSION AND FUTURE WORK

In this paper we presented the FACE framework, a fog-computing strategy to reduce bandwidth consumption. It consists of data processing functions

Table 4: Wireless test results.

(in seconds)		**BS**	**PN**								**Server**		
Test	**Mov**	T2-T1	T3-T2	T4-T3	T4A-T4	T5-T4A	T5A-T5	T5B-T5A	T6-T5(B)	T6A-T6	T7-T2(6A)	Latency	Gain
0	Yes	4.519									0.001	4.520	0
1A	Yes	4.725	4.91E-6	0.000	2.175	0.002			0.002	2.164	0.175	9.244	0
1B	Yes	4.755	5.17E-6	0.001	2.494	0.002	0.049	0.003	0.003	2.466	0.172	9.945	0.01
2A	Yes	4.778	4.90E-6	0.001	2.514	0.003			0.003	2.507	0.172	9.978	0
2B	Yes	4.788	5.01E-6	0.002	2.567	0.002	0.048	0.004	0.003	2.545	0.172	10.130	0.01
0	No	4.460									0.001	4.461	0
1A	No	4.656	5.29E-6	0.001	2.076	0.002			0.003	2.064	0.176	8.978	0
1B	No	4.693	5.10E-6	0.000	2.306	0.002	0.049	0.005	0.000	2.288	0.172	9.516	0.97
2A	No	4.710	5.12E-6	0.001	2.236	0.002			0.003	2.227	0.174	9.353	0
2B	No	4.782	5.08E-6	0.001	2.386	0.003	0.047	0.004	0.000	2.377	0.174	9.774	0.96

Table 5: Environment comparison.

Environment	BS to PN (s)	PN (s)	PN to server (s)	Total (s)	Difference (s)	Gain
Wired	0.000	2.192	0.184	2.376	0.113	96.50%
Wireless	0.000	4.735	0.173	4.907	0.425	96.50%

(filtering, aggregation, compression, and extraction) performed in the edges of the network, to reduce the data traffic from the end user until a central location. Each function is applied for a different purpose, and aims at meeting requirements of application service providers. To evaluate the performance of the framework, we designed an architecture that allows to apply compression function over the data between an end device and a cloud server. Results evaluations have identified savings in network infrastructure at the cost of latency increase. The developed prototype showed bandwidth gain in video surveillance when image compression is applied in cases when the scene does not change from one frame to another.

The tests related to the video surveillance scenario showed that most of the latency introduced by the processing node comes from the time spent in the image transmission. The processing node holds the image until it has completed the transmission, so it can be processed and then forwarded. For applications that are not latency-sensitive, this solution can be applied to promote savings in network infrastructure. If applied to an environment with dynamic scene changes, the advantages of using this solution are substantially reduced. Other compression rules can also be applied to fit custom application requirements.

Next steps consist of further tests to investigate how the latency could be improved. A set of tests could be applied to a real video surveillance scenario, using more cameras that continuously send data. It would also be interesting to evaluate the architecture on an infrastructure with telecommunication components, such as base stations and small cells. Since the mover mechanism was the main cause of latency, it is important to study how to optimize it, by reducing the time it takes to start sending images. This achievement would result in a significant latency decrease, and then provide a more effective solution. We also plan to evaluate the other functions provided by FACE framework in scenarios where data traffic is intensive.

ACKNOWLEDGEMENTS

This work was sponsored by the Ericsson Innovation Center, Ericsson Telecomunicações S.A., Brazil.

REFERENCES

(2015). Ericsson Mobility Report. Technical report.

Bonomi, F., Milito, R., Zhu, J., and Addepalli, S. (2012). Fog computing and its role in the internet of things. In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, pages 13–16. ACM.

Dastjerdi, A. V. and Buyya, R. (2016). Fog computing: Helping the internet of things realize its potential. *Computer*, 49(8):112–116.

Gonzalez, N. M., Goya, W., Pereira, R., Langona, K., Silva, E., Carvalho, T., Miers, C., Mångs, J., and Sefidcon, A. (2016). Fog computing: Data analytics and cloud distributed processing on the network edges. In *35th International Conference of the Chilean*.

Hu, W., Gao, Y., Ha, K., Wang, J., Amos, B., Chen, Z., Pillai, P., and Satyanarayanan, M. (2016). Quantifying the impact of edge computing on mobile applications.

Margulius, D. (2002). Apps on the edge. *InfoWorld*, 24(2).

Nilsson, F. and Axis, C. (2017). *Intelligent Network Video: Understanding Modern Video Surveillance Systems*. Taylor & Francis.

Pang, H. and Tan, K.-L. (2004). Authenticating query results in edge computing. In *Data Engineering, 2004. Proceedings. 20th International Conference on*, pages 560–571. IEEE.