POLITECNICO DI TORINO
Repository ISTITUZIONALE

From Megabits to CPU Ticks: Enriching a Demand Trace in the Age of MEC

Original
From Megabits to CPU Ticks: Enriching a Demand Trace in the Age of MEC / Malandrino, Francesco; Chiasserini, Carla Fabiana; Avino, Giuseppe; Malinverno, Marco; Kirkpatrick, Scott. - In: IEEE TRANSACTIONS ON BIG DATA. - ISSN 2332-7790. - STAMPA. - 64:1(2020), pp. 43-50. [10.1109/TBDATA.2018.2867025]

Availability:
This version is available at: 11583/2711884 since: 2020-02-29T17:52:48Z

Publisher:
IEEE

Published
DOI:10.1109/TBDATA.2018.2867025

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright
IEEE postprint/Author's Accepted Manuscript

©2020 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collecting works, for resale or lists, or reuse of any copyrighted component of this work in other works.

(Article begins on next page)

17 April 2024

From Megabits to CPU Ticks: Enriching a Demand Trace in the Age of MEC

Francesco Malandrino, *Member, IEEE,* Carla-Fabiana Chiasserini, *Fellow, IEEE,* Giuseppe Avino, Marco Malinverno, Scott Kirkpatrick, *Life Fellow, IEEE*

Abstract—All the content consumed by mobile users, be it a web page or a live stream, undergoes some processing along the way; as an example, web pages and videos are transcoded to fit each device's screen. The recent multi-access edge computing (MEC) paradigm envisions performing such processing *within* the cellular network, as opposed to resorting to a cloud server on the Internet. Designing a MEC network, i.e., placing and dimensioning the computational facilities therein, requires information on how much computational power is required to produce the contents needed by the users. However, real-world demand traces only contain information on how much data is downloaded. In this paper, we demonstrate how to *enrich* demand traces with information about the computational power needed to process the different types of content, and we show the substantial benefit that can be obtained from using such enriched traces for the design of MEC-based networks.

1 Introduction

Dynamic web pages, targeted advertisement, user-generated content have all increased the *computation* required to assemble the pages displayed by web browsers. Mobile services, and the mobile devices consuming them, have made this trend even more evident, and virtually all content mobile users see is the result of multiple steps of real-time, on-the-fly processing. Prominent examples include videos, that are transcoded to match the screen size and resolution of the device playing them, and social networks, which decide what to show to their users based on their identity and location.

Where such processing ought to be performed is a fundamental question. The traditional approach was to use ad hoc servers placed within each content provider's network; cloud computing, with shared virtual servers placed in the Internet, provides a more efficient but substantially equivalent alternative [1]. The recent multi-access edge computing (MEC) paradigm takes a different approach: it envisions moving services, i.e., the servers providing them, as close to mobile users as possible. Servers will not be localized in remote datacenters, but at different entities within the mobile core network itself, from core switches to base stations [2], [3]. Similarly to cloud scenarios, there is a measure of cooperation between content providers and network operators. The servers deployed by the network operators will run services developed by the content providers, usually coming as a set of *virtual network functions* (VNFs) running on virtual machines or containers.

Designing a MEC network requires making decisions on where, within the network edge, servers shall be placed, and how to dimension them. Both questions require a deep knowledge of the data demand the network will have to serve: more exactly, we need to know (i) how much data the users will require; (ii) the type of such data, e.g., the mobile apps requesting it, and (iii) how much computational power will be needed to produce these data. The first two items have been widely studied: using operator-provided [4], [5] and crowd-sourced [2], [3] traces, it is possible for researchers to obtain a fairly good picture of the data demand of mobile networks, including its time and space evolution, as well as the services contributing to it.

Estimating the *processing power* needed to serve a given network demand, on the other hand, is much more challenging. The processing required to generate one gigabit per second of video data is not the same as the processing needed for the same quantity of gaming updates or maps. Additionally, it makes a significant difference whether the data is consumed by a small number of users enjoying a high bandwidth, or through a larger number of lower-rate connections. *None* of the currently available real-world traces contain all the information needed to distinguish these cases.

We fill this gap by developing and demonstrating a methodology to *enrich* existing demand traces with computational power information, translating (so to say) megabits of downloaded data into CPU ticks consumed at the servers. Specifically, in this paper we (i) consider a large-scale, crowd-sourced cellular demand trace, already used in big data applications [6] ; (ii) perform an extensive set of experiments, linking the quantity of downloaded data in different conditions (type of service, number of users,...) to the amount of CPU power required at the servers; (iii) assess the impact of enriching our traces on the resulting planning of the MEC network. Importantly, we make *publicly available* realistic user demand traces, as well as our experimental dataset linking user demand to computational burden [7].

We stress that, although MEC was our original motivation and we use it as a test case, the problem we address is

- *F. Malandrino, C.-F. Chiasserini, G. Avino and M. Malinverno are with Politecnico di Torino, Italy. C.-F. Chiasserini is also with CNIT and a research associate with CNR-IEIIT, Torino, Italy. S. Kirkpatrick is with the Hebrew University of Jerusalem, Israel.*

more fundamental and consistent with the high-level goal of big data research, that is, to turn data into *information*. In our case, data come from real-world measurements, and the information we seek has to do with next-generation networks based on MEC. In spite of all their (perceived) abundance and the (apparent) easiness with which they are processed, real-world measurements can only deal with present-day systems and, thus, they are not naturally well-suited to study future ones. In this paper, we demonstrate how this limitation can be overcome with the help of additional data, real-world experiments, and domain knowledge.

The remainder of the paper is organized as follows. We begin by reviewing related work in Sec. 2. Then we present the real-world demand traces we use as a starting point in Sec. 3, and describe the experiments we perform to enrich them in Sec. 4. Sec. 5 introduces and discusses our MEC network design strategy. After presenting our numerical results in Sec. 6, we conclude the paper in Sec. 7.

2 Related work

Our study is connected to three main categories of prior work: papers presenting real-world mobile traces and datasets; studies addressing MEC in general and MEC-based 5G networks specifically; works doing the latter using the first.

Many real-world traces come from volunteers, such as the MIT Reality Project [8] and the Nokia Mobile Challenge. These traces include a great deal of valuable information; their main shortcoming is the limited number of participants (in the case of the Nokia Mobile Challenge, around two hundred). This scale is adequate to study, for example, user mobility or encounter patterns, but studying a whole cellular network requires information about many more users. Mobile operators are typically reluctant to release demand and deployment information to the scientific community. An exception is represented by the Data For Development dataset by Orange [4], including mobility information for 50,000 users in Ivory Coast, as well as CDR (call-detail record) information for phone calls and SMS messages. However, the Orange trace only includes voice calls and SMS, and is severely restricted by heavy anonymization – each ID encountered gets a new coded identity for each "ego site" to which they are a neighbor. In other cases [5], mobile operators have released demand or deployment information to individual research teams under non-disclosure agreements; however, these traces typically include only one operator and/or only one city.

MEC has been recently introduced [9] as a way to move "the cloud", i.e., the servers processing mobile traffic, closer to the end users, thus reducing latency and traffic load across the network infrastructure. Network Function Virtualization (NFV) is widely regarded to as an enabling technology for MEC (see, e.g., [9]). Recent works have studied the radio techniques needed to enable MEC [10], its relationship to the Internet-of-things [11] and context-aware, next-generation networks [12]. Closer to our scenario, the authors of [13] study how caches and servers should be placed in the network as its load changes over time. Regarding MEC and caching, a prominent application is mobile video streaming. As an example, [14], [15] account for layered video coding techniques, and address the problem of placing the right layers at the right cache – with [14] also accounting for cooperation between operators. Other works [16], [17] aim at *foreseeing* the content demand, in order to proactively populate caches or serve users.

TABLE 1
The WeFi datasets

	Atlanta	Los Angeles	San Francisco
Time of collection	Oct. 2015	Oct. 2015	Mar. 2015
Covered area [km^2]	55×66	46×73	14×11
Total traffic [TB]	9.34	35.61	9.18
Number of records	13 million	81 million	60 million
Unique users	9,203	64,386	14,018
Unique cells	12,615	36,09	14,728

5G will significantly exploit the MEC paradigm, and a substantial body of research is devoted to the problem of placing VNFs across the network providers' servers. As an example,[18], [19], [20], [21], [22] tackle the problems of VNF placement and routing from a network-centric viewpoint, i.e., they aim at minimizing the load of network resources. In particular, [18] seeks to balance the load on links and servers, while [19] studies how to optimize routing to minimize network utilization. The above approaches formulate mixed-integer linear programming (MILP) problems and propose heuristic strategies to solve them. [20], [21] and [22] formulate ILP problems, respectively aiming at minimizing the cost of used links and network nodes, minimizing resource utilization subject to QoS requirements, and minimizing bitrate variations through the VNF graph.

Not many works however exist that combine real-world traces and multi-access edge computing. Among the most recent ones, [2] studies the price (in terms of additional infrastructure) of deploying caches within the cellular core network. Compared to our work, [2] only focuses on caching and vehicular traffic, and it only considers the dataset for the city of Los Angeles.

Our earlier work [3] sets in the same scenario as this paper: the traces introduced in Sec. 3 were used to study the trade-offs between network latency and server utilization in MEC network design. The results presented in [3] use the quantity of data downloaded by users as a proxy metric for computational capabilities required at the servers, and their limited applicability represents one of the main motivations behind this paper.

3 Input data

WeFi, now acquired by TruConnect Technologies, is an Android application providing location-specific information on the speed, security, and reliability of nearby Wi-Fi networks. At the same time, it collected data on the activity of its users, including location, mobile phone activity, and available connectivity options. For our study we use three datasets, coming from the U.S. cities of Atlanta, Los Angeles, and San Francisco, characterized by the features summarized in Tab. 1.

Datasets are organized in *records*, each containing: time and GPS location; anonymized user identifier; current operator and cell identifier; active application on the smartphone

and amount of data it downloads. New records are generated every time any of the above changes (e.g., the user moves or switches apps), or a one-hour period elapses.

The WeFi datasets represent a real-world, *live* snapshot of both mobile networks and their users, and have three features that make them especially relevant to our study. First, they contain information on several cities, different for traffic demand profile and network deployment. Furthermore, they include multiple mobile operators, with different deployment strategies, e.g., usage of micro- and macro-cells. Finally, they allow us to know the individual application generating each traffic flow.

The WeFi data has been already used in the field of *big data*, e.g., in the preliminary analysis [6] and follow-up works. The purpose therein was to identify patterns of living, commuting, fast food consumption, work activities and recreation for tens of thousands of people. That case study confirms that the content and quality of the WeFi datasets are sufficient not only for networking studies, but also for social observations and government actions.

Dataset available. While we cannot share the dataset we use, we produced a synthetic trace with the same time and space characteristic of its original counterpart, available for download from [7]. For more details on the generation of such synthetic traces, the interested reader is referred to [23].

4 Enriching the Mobile Data Traces

As mentioned in Sec. 1, our high-level goal is to use the *demand* information available in the datasets to understand the *deployment* we need, i.e., how much computational capacity shall be placed within the network, and where. To this end, we perform three main steps:

1) we design and perform a set of experiments measuring the computational load associated with different types of traffic, as detailed in Sec. 4.1;
2) we use the experimental data to train a *model* connecting the two quantities, as described in Sec. 4.2;
3) we exploit the model to estimate the computational capacity needed to serve the traffic we observe in our dataset, as discussed in Sec. 4.3.

4.1 Experimental setup

We focus on three different, highly representative types of traffic, namely, video streaming, gaming, and maps. For reproducibility purposes, we restrict ourselves to open-source programs, namely:

- for video, the FFserver server and the VLC client;
- for gaming, the Minecraft server and the corresponding Minecraft Pocket mobile client;
- for maps, the OpenMapTiles server [24], based on data from OpenStreetMap [25].

For our experiments, we use a testbed, composed of two computers – a client and a server – connected via a Wi-Fi link. Both the client and the server are commodity, off-the-shelf computers equipped with Intel Core i7-7700T processors and running Ubuntu Linux 16.10. On the client machine, we run the Genymotion Android emulator, which in turn emulates a varying number of mobile clients. The server machine hosts one instance of the server application, containerized within Docker.

In streaming tests, we use three videos with different duration, resolution, and file size; for gaming, we employ three different moving patterns throughout the Minecraft world; for maps, we generate requests for randomly-chosen $20 \times 20\,\mathrm{km}^2$-areas. In both cases, we vary the number of clients between 1 and 8, and use the FRep program to drive the emulator, thus guaranteeing uniform, reproducible patterns of UI actions, e.g., taps and swipes.

In all experiments, we measure two quantities: the amount of data generated by the server program, and the computational capacity it consumes. The former simply corresponds to the traffic outgoing from the `docker0` virtual interface created by Docker; the latter is obtained by polling the `/proc/<PID>/stat` file relating to the server process. The FFserver and Minecraft servers are single-threaded, so there is only one process to account for. The OpenMapTiles server, on the other hand, includes several processes, including a database and a web server; in that case, we present the aggregate CPU consumption figures.

4.2 Results and model

Fig. 1(a), Fig. 1(b) and Fig. 1(c) summarize our experimental results. Each blue dot therein corresponds to an experiment, e.g., a different combination of video file and number of clients; its position in the plot is determined by how much data was served in that experiment, and the corresponding load on the server. Computational load values are expressed in *CPU ticks*, the minimum unit of CPU scheduling in the operating system. In Linux, each tick represents 10 ms; as an example, saying that a process takes 200 ticks to complete means that the CPU was assigned to that process for 2 s [1].

The first thing to notice is the scale of the two plots. As one might expect, serving video implies transferring massively more data than hosting a game server, and therefore video is rightfully regarded as one of the applications consuming most of the bandwidth offered by networks. Looking at the y-axis, on the other hand, a different aspect emerges: the amount of CPU resources associated with gaming and, to a lesser extent, maps, dwarfs the one required by video.

This disproportion makes intuitive sense: while streaming a video requires little more than reading a file from disk and feeding it into a network stream, game and map servers have to perform complex operations such as keeping track of the game status or rendering the maps. However, its scale is somehow unexpected, and serves as a clear reminder that designing and deploying the *computation* part of a network, i.e., where and how to place its computational power, using only the traffic demand as a guideline is likely to result in poor performance and low efficiency.

The vastly different relationships between served traffic and CPU load are also evident from the linear fit we obtain

1. The real execution time might be longer, e.g., if the process is preempted by a higher-priority one during its execution.

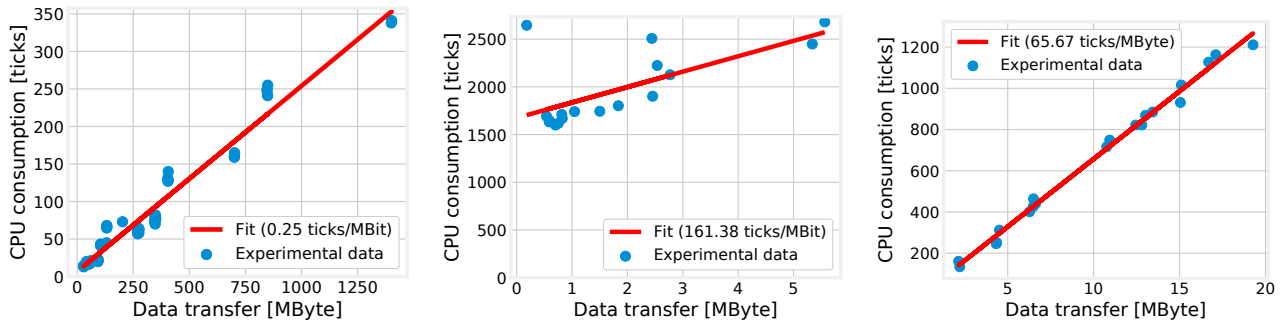

Fig. 1. Data (blue dots) and linear fit (red line) for the video (a), gaming (b) and maps (c) applications. Fitting error (RMSE) are 4%, 6% and 2% respectively.

from the two sets of experiments, represented by red lines in Fig. 1. The fitted relationships are as follows:

$$\text{CPU}_{[\text{ticks}]} = \begin{cases} 0.25 \times \text{traffic}_{[\text{Mbit}]} + 6.76 & \text{for video,} \\ 161.38 \times \text{traffic}_{[\text{Mbit}]} + 1675.03 & \text{for gaming,} \\ 67.44 \times \text{traffic}_{[\text{Mbit}]} - 7.53 & \text{for maps.} \end{cases} \quad (1)$$

Looking at Fig. 1, it also important to observe how the fit is much better for maps traffic than for gaming: the root mean square error (RMSE) for maps is 2% while it is equal to 6% for gaming. The reason is that the quantity of CPU required by gaming applications significantly depends on such factors as the actions performed by the players – a fact that leads to a larger variability.

The fact that all fitted relationships are linear is not especially relevant *per se*, nor particularly surprising; indeed, it makes intuitive sense that the amount of work needed to produce a certain type of data grows, more or less linearly, with the quantity of data to produce. What matters the most is the *slope* of the fitted lines, which changes by several orders of magnitude from one traffic category to another.

In the following, we will indicate with τ_k the number of CPU ticks needed to process one megabyte of traffic of category k; for instance, $\tau_{\text{video}} = 0.25$ CPU ticks/Mbyte (i.e., the slope of the red line in Fig. 1(a)), $\tau_{\text{gaming}} = 161.38$ ticks/Mbyte (i.e., the slope of the red line in Fig. 1(b)), and $\tau_{\text{maps}} = 67.44$ ticks/Mbyte (i.e., the slope of the red line in Fig. 1(c)).

4.3 Enriching the dataset

As a preliminary step, we need to decide, for each app we observe in the dataset, whether its traffic can be considered video-like, gaming-like, or map-like (or none of them). We perform such an assignment as follows:

- traffic coming from YouTube, Netflix, TimeWarner, ShowBox, Twitch, DirectTV, FoxSports, FoxNews, is tagged as video-like;
- traffic coming from Minecraft, World of Warcraft, Riptide, Grand Theft Auto, Rollercoaster Tycoon, This War of Mine, Titan Quest, Unkilled, is tagged as game-like;
- traffic coming from Google Maps and Waze is tagged as map-like.

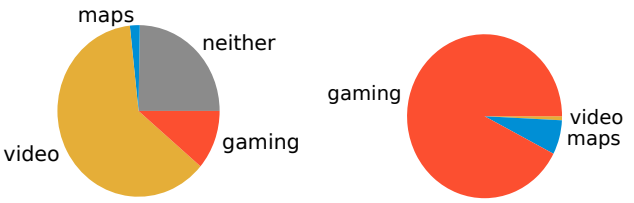

Fig. 2. Categories traffic demand belongs to (left); amount of computational power required to serve video-, and gaming- and maps-like traffic (right).

As summarized in Fig. 2(left), about 66% of the traffic we observe in our dataset can be classified as video-like, about 15% of it as game-like, while the amount of map-like traffic is much smaller. Much of the traffic that cannot be tagged comes from social networking applications, whose behavior cannot be easily studied due to the lack of open-source social network servers.

Once the assignment is made, we can use the relationship in (1), learned in Sec. 4.2, to add a new column to our dataset. The column, called `cpu_ticks`, expresses how much computational power is required to generate the traffic reported in each line of the dataset. Notice that relationships other than linear can be accounted for at no additional complexity.

Fig. 2 further highlights the gap between the quantity of traffic generated by different applications and the corresponding CPU load. Gaming and maps represent a small fraction of the total traffic (Fig. 2(left)); however, by applying (1), we obtain the results shown in Fig. 2(right): gaming applications consume the vast majority, over 90%, of all computational resources. This further highlights the importance of accounting for the computational load of different types of traffic in network design.

4.4 Discussion

Our trace enrichment strategy has some limitations: it does not account for all the traffic types, and might not perfectly model the behavior of all servers.

The first issue is evident from Fig. 2(left): around 25% of all traffic cannot be classified as either video-like, gaming-like, or maps-like. On the other hand, our analysis is able to account for over 70% of all present-day traffic, and video,

gaming and maps are the applications that are expected to grow the most in the near future.

The second issue has to do with the fact that we model the behavior of YouTube and Netflix using FFserver, of World of Warcraft through Minecraft, or of Google Maps using OpenStreetMap. On the one hand, this might sound a bit bold. On the other hand, it is true that the problems faced by different applications of the same type – and the solutions thereto – tend to overlap. This explains, as an example, the existence of gaming *engines* such as Unity or Microsoft XNA, providing the developers of a heterogeneous set of games with homogeneous solutions to a small set of common problems.

In summary, our analysis provides valuable guidelines highlighting the existence of a mismatch between the quantity of traffic and the corresponding CPU load, as well as its magnitude. Additionally, the methodology we employ is general, and works unmodified in cases where additional traffic types and/or applications are taken into account, or if larger-scale experiments can be performed.

Dataset available. The results of our experiments are available for download from [7].

5 MEC DESIGN

We now show how the enhanced dataset we created can be used to devise a MEC design strategy. Our input data are represented by:

- a set of base stations;
- the expected/predicted traffic at each of them and its characteristics;
- the network connectivity.

Given the above, we have to dimension the network computational capabilities, i.e., to decide (i) where to place the MEC servers, and (ii) the capacity they should have.

We solve this problem accounting also for the specific applications we deal with, thus developing an *application-aware deployment*, accommodating the requirements of each application. Indeed, such requirements can be substantially different for different *categories* of applications. As an example, we might be willing to serve such (comparatively) delay-tolerant content as video-like and map-like traffic through a server located in the core network, while real-time services like mobile gaming will require their servers to be much closer to the end users, possibly at individual base stations. It is important to stress that, in pure NFV/MEC fashion, we allow the same servers to serve multiple applications concurrently, provided that the server capacity is not exceeded.

At last, note that our input data is represented by the traffic that is *actually* served by the base stations. Consistently with MEC design best practices [26], [27], we do not need to explicitly account for access-network issues such as congestion and interference. Indeed, the coverage quality experienced by users influences the amount of data they are able to upload/download, and this aspect is captured by our system model and MEC design.

5.1 Network topology

The identity and positions of base stations $b \in \mathcal{B}$, as well as their demand $\delta(b,k,t)$ are readily available from the trace we describe in Sec. 3. However, we have no information about the topology of the backhaul cellular network. Indeed, mobile operators are extremely reluctant to disclose this information, and virtually all works in the literature resort to synthetic topologies based on current best practices. Following [28], we assume a *fat-tree* topology, where:

- base stations are grouped into *rings* of ten;
- every ten rings, there is an aggregation-level *pod*;
- every ten pods, there is a core-level switch.

The topology has a fan-out of 2, i.e., we connect every ring to the two closest pods, and every pod to the two closest core switches, while switches themselves are connected in a full mesh. The network topology we generate can be represented as a graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$.

We also generate a DAG $\mathcal{D} = (\mathcal{N}, \mathcal{F})$, to keep track of which nodes can serve each base station. In particular, we generate a (directed) edge $(n_1, n_2) \in \mathcal{F}$ between nodes such that:

- n_1 and n_2 are connected, i.e., $(n_1, n_2) \in \mathcal{E}$, and
- n_1 belongs to a network level immediately higher to the one of n_2, e.g., n_1 is a ring and n_2 is a base station.

We will say that node n_1 is a *parent* of n_2 if $(n_1, n_2) \in \mathcal{F}$. Base stations have no children, i.e., they are leaves in the DAG, while core switches have no parents, i.e., they are roots.

Notation. We denote by $n \in \mathcal{N}$ all the nodes of the cellular network, from base stations to core-level switches, and by $b \in \mathcal{B} \subset \mathcal{N}$ the base stations among them. The physical distance between any two nodes n_1 and n_2 is denoted by $d(n_1, n_2)$. Contents are specific to *category* $k \in \mathcal{K}$, e.g., video, gaming, or maps, and time is discretized into steps $t \in \mathcal{T}$. We denote by $\delta(b,k,t)$ the demand from users covered by base station b, for contents of category k and during step t.

5.2 MEC design

Designing our MEC network means making two decisions, each corresponding to a binary variable:

- whether we should place a server at node $n \in \mathcal{N}$, expressed through variable $y(n) \in \{0,1\}$;
- whether the traffic coming from base station $b \in \mathcal{B}$ shall be served by the server placed at node $n \in \mathcal{N}$, expressed through variable $x(b,n) \in \{0,1\}$.

Note that none of the decision variables depends on the time step $t \in \mathcal{T}$; this reflects the fact that deployment decisions are made periodically, with a time period much longer than a single time step, accounting for the evolution of data demand over the previous period. Needless to say, we cannot serve anything on servers that do not exist, i.e., it must be:

$$x(b,n) \leq y(n), \quad \forall b \in \mathcal{B}, n \in \mathcal{N}.$$

Algorithm 1 Greedy MEC design

Require: $k, L_{\max}, \delta(b,k,t), \mathcal{B}, \mathcal{N}$
1: **for all** $b \in \mathcal{B}$ **do**
2: **if** $\max_{t\in\mathcal{T}} \delta(b,k,t) > 0$ **then**
3: $y(b) \leftarrow 1$
4: $x(b,b) \leftarrow 1$
5: **end if**
6: **end for**
7: **while** `latency` $\leq L_{\max}$ **do**
8: $\mathcal{P} \leftarrow \{(n_1,n_2) \in \mathcal{N}\colon y(n_1) > 0 \wedge y(n_2) > 0 \wedge (n_1,n_2) \in \mathcal{F}\}$
9: $\mathcal{P} \leftarrow \mathcal{P} \cup \{(n_1,n_2) \in \mathcal{N}\colon y(n_1) > 0 \wedge y(n_2) > 0 \wedge \exists n_3 \in \mathcal{N}\colon (n_3,n_1) \in \mathcal{F} \wedge (n_3,n_2) \in \mathcal{F}\}$
10: $n_1^\star, n_2^\star \leftarrow \arg\max_{\mathcal{P}}$ `score`(n_1,n_2)
11: **consolidate**$(n_1^\star, n_2^\star)$
12: **end while**
13: **return** $x(b,n), y(n)$

Algorithm 2 The consolidation procedure

Require: n_1, n_2, k
1: **if** $(n_1,n_2) \in \mathcal{F}$ **then** ▷ parent-children, like Fig. 3(top)
2: **for all** $b \in \mathcal{B}\colon x(b,n_2) > 0$ **do**
3: $x(b,n_2) \leftarrow 0$
4: $x(b,n_1) \leftarrow 1$
5: **end for**
6: $y(n_2) \leftarrow 0$
7: **else** ▷ siblings, like Fig. 3(bottom)
8: $n_3 \leftarrow n_3 \in \mathcal{N}\colon (n_1,n_3) \in \mathcal{F} \wedge (n_2,n_3) \in \mathcal{F}$
9: **for all** $b \in \mathcal{B}\colon x(b,n_1) > 0$ **do**
10: $x(b,n_1) \leftarrow 0$
11: $x(b,n_3) \leftarrow 1$
12: **end for**
13: **for all** $b \in \mathcal{B}\colon x(b,n_2) > 0$ **do**
14: $x(b,n_2) \leftarrow 0$
15: $x(b,n_3) \leftarrow 1$
16: **end for**
17: $y(n_1) \leftarrow 0$
18: $y(n_2) \leftarrow 0$
19: $y(n_3) \leftarrow 1$
20: **end if**

The objective of the problem can be stated as deploying the smallest possible number of servers subject to delay constraints, i.e.,

$$\min_{x,y} \sum_{n\in\mathcal{N}} y(n).$$

Optimally setting the binary x- and y-variables subject to constraints on the need to serve all traffic requires solving an ILP problem[2], which is notoriously [29] impractical even for modestly-sized problem instances. We therefore devise a greedy design procedure, able to efficiently make good-quality deployment decisions.

5.2.1 Greedy design procedure

Our greedy design procedure is inspired to hierarchical clustering and summarized in Alg. 1. It takes as input (Line 0) the content category k to consider, its maximum processing latency $L_{\max}$, its demand over time $\delta(b,k,t)$, and the sets $\mathcal{B}$ and $\mathcal{N}$ of, respectively, base stations and network nodes. This implies that Alg. 1 makes *per-category* decisions, and is thus able to account for the fact that different categories of content, with different latency limits $L_{\max}$, might require different deployment strategies. Notice that with *latency* we indicate the time spent within the core network, which is itself a component of the total, end-to-end service time.

The algorithm starts (Line 1) from a solution where each base station that ever serves at least one user requiring content category k (i.e., $\max_{t\in\mathcal{T}} \delta(b,k,t) > 0$) has its own server, i.e., $y(b) = 1$. Then, as long as the latency resulting from our deployment does not exceed the threshold $L_{\max}$ (Line 7), we select a pair of nodes (n_1,n_2) to *consolidate* together, reducing the number of servers at the price of potentially increasing the service latency.

The consolidation procedure is depicted in Fig. 3. It involves two nodes n_1 and n_2 such that either n_2 is n_1's parent, or n_1 and n_2 are siblings, i.e., have a common parent n_3. In both cases, the server(s) at the child(ren) are removed, and all base stations they used to serve are served by the parent. Every time we perform the consolidation procedure, the number of servers deployed in the topology decreases by one unit, at the cost of potentially increasing the latency.

In Line 8–Line 9 of Alg. 1, we construct a set $\mathcal{P}$ of pairs of nodes eligible for consolidation, i.e., such that (i) they both have a server, and either (ii) n_1 is n_2's parent (Line 8), or (iii) both n_1 and n_2 have a common parent n_3. In Line 10, we select the nodes $n_1^\star$ and $n_2^\star$ with the highest score, i.e., the most suitable to consolidate, and call the **consolidate** procedure with those nodes as an argument. As discussed in Sec. 5.2.2, different definitions of score can be considered, leading to different deployment strategies.

Alg. 2 details the consolidation procedure, and takes as input the nodes n_1 and n_2 to consolidate. If n_2 is n_1's parent, i.e., we are in the situation of Fig. 3(top), the server at n_2 is removed and any base stations that were served by n_2 are served by n_1 (Line 1–Line 6 of Alg. 2). If n_1 and n_2 are siblings, i.e., we are in the situation of Fig. 3(bottom), then we first identify the node n_3 that is a parent to both n_1 and n_2 (Line 8). Afterwards, servers at both n_1 and n_2 are removed, a new server at n_3 is created, and all base stations that were served by n_1 or n_2 are served by n_3 (Line 7–Line 19 of Alg. 2).

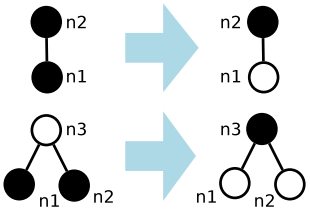

Fig. 3. The consolidation procedure. Circles represent nodes in $n \in \mathcal{N}$; black ones correspond to nodes that host a server, i.e., $y(n) = 1$, white ones to nodes that do not, i.e., $y(n) = 0$. If n_1 is n_2's parent (top), the server at n_2 is removed and all base stations it served, are served by n_2's parent n_1. If n_1 and n_2 are siblings (bottom), the servers at n_1 and n_2 are removed, and a new one is created at their parent n_3.

2. We skip the proof, based on a reduction from the SAT problem, in the interest of space.

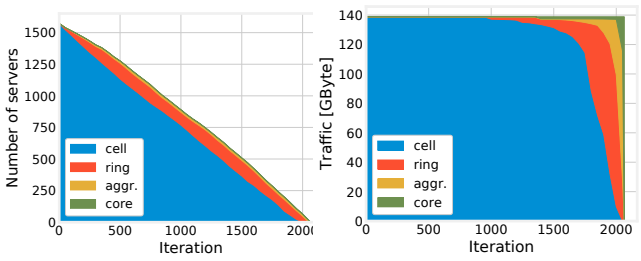

Fig. 4. Location-based scores: number of servers (left) and amount of traffic processed (right) at the different network levels, for each iteration of Alg. 1.

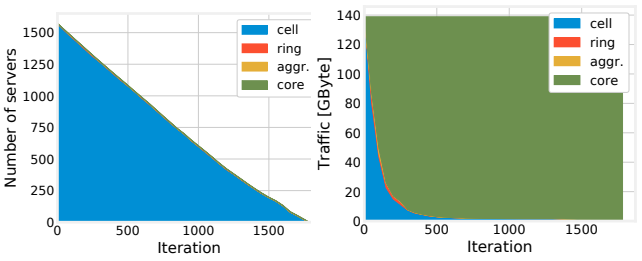

Fig. 5. Load-based scores: number of servers (left) and amount of traffic processed (right) at the different network levels, for each iteration of Alg. 1.

5.2.2 Score definitions

A key feature of our approach is that it can support multiple deployment *strategies* through the one *algorithm* Alg. 1. We are able to do so by considering multiple definitions of the score associated with consolidating two network nodes, i.e., the implementation of function `score` called in Line 10. Recall that we are still focusing on a single content category k.

Location-based It is often desirable to consolidate nodes that are physically close to each other, as this typically translates into a shorter travel time between the users and the servers serving them. This corresponds to giving higher scores to pairs of nodes that are close to each other, i.e.,

$$\texttt{score}(n_1, n_2) = -d(n_1, n_2). \tag{2}$$

Load-based An alternative deployment strategy takes into account the load of each server, and tries to avoid consolidating nodes whose demands have a similar time evolution, i.e., whose peak times tend to overlap. The rationale is that by doing so we can decrease the capacity requirements for the consolidated servers, which depend upon the *peak* of the combined load. More formally, let us define a serve vector $\vec{s}(n)$ for each node n. $\vec{s}(n)$ vectors have $|\mathcal{T}|$ elements, and each element $s(n)_t$ represents the total demand for contents of category k by users at base stations that are served by node n during time step t:

$$s(n)_t = \sum_{b \in \mathcal{B}} x(b, n)\delta(b, k, t).$$

Given the $\vec{s}(n)$ values, we can define the score related to the (n_1, n_2) pair as

$$\begin{aligned}\texttt{score}(n_1, n_2) = \max_{t \in \mathcal{T}} \tau_k s(n_1)_t + \max_{t \in \mathcal{T}} \tau_k s(n_2)_t + \\ - \max_{t \in \mathcal{T}} \tau_k \left(s(n_1)_t + s(n_2)_t\right).\end{aligned} \tag{3}$$

where the first two terms of the second member of (3) represent the peak loads of nodes n_1 and n_2 before consolidation; the third term is the peak load of the combined server, after consolidation. A high-scoring consolidation operation will involve nodes with high peak loads (first two terms, with positive sign) that can be combined into a new, low-load server (third term, with negative sign).

Recall that the factor τ_k in (3) expresses how many units of computational power (e.g., CPU ticks) are needed to generate one unit of traffic (e.g., one megabyte) of category k, as obtained in Sec. 4.3. Using non-enriched traces corresponds to assuming $\tau_k = 1, \forall k \in \mathcal{K}$.

5.2.3 Multiple categories

As mentioned in Sec. 5.2.1, both Alg. 1 and Alg. 2 make decisions on a *per-category* basis, and are therefore able to reproduce the fact that different applications can require different deployments. In the following, we consider the presence of different application categories and describe how our approach can be easily leveraged to realize an application-aware deployment. We proceed as follows:

- we divide application into categories;
- we run our greedy deployment procedure separately for each category;
- we combine the resulting deployments.

Running Alg. 1 separately for different categories also means evaluating Line 7 therein using the maximum latency values $L_{\max}$ of each category. Similarly, the demand-aware distance (3) is computed separately for each category, only accounting for the contents falling in the current category.

6 Numerical Results

In this section, we investigate how deployment strategies and score definitions (Sec. 6.1), using enriched traces (Sec. 6.2) and considering per-category latency limits (Sec. 6.3), impact the resulting MEC deployment and its effectiveness. For sake of brevity, we only present results for one of the three operators included in our trace. High-resolution versions of the plots for all operators, as well as the Matplotlib source code to generate them, are available from [7].

6.1 Effect of the deployment strategy

The first aspect we are interested in is the impact of the score definition we adopt, i.e., whether we use (2) or (3) to implement the `score` function in Alg. 1. To this end, we first assume no latency limit, i.e., $L_{\max} = \infty$, and study (i) how much traffic is processed at each level of the network – base station (BS), ring, aggregation, core – and (ii) how many servers are deployed therein.

Fig. 4 and Fig. 5 demonstrate how Alg. 1 and the consolidation procedure work in the case of location-based and load-based scores, respectively. We start at iteration 0 with one server at each BS; then, at each iteration, we reduce the total number of servers through consolidation, replacing BS servers with servers placed at the higher levels of the network topology. We can observe that, while Fig. 4(left)

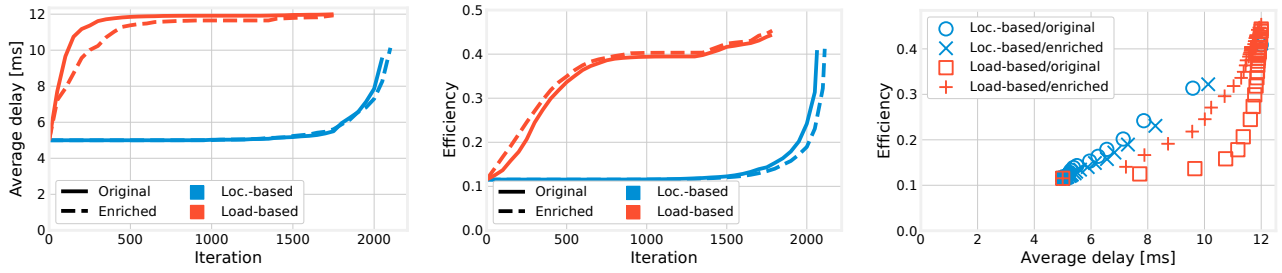

Fig. 6. Latency (a) and efficiency (b) for each iteration of Alg. 1 for different deployment strategies when the original (solid lines) and enriched (dotted lines) trace is used; resulting latency/efficiency trade-offs (c).

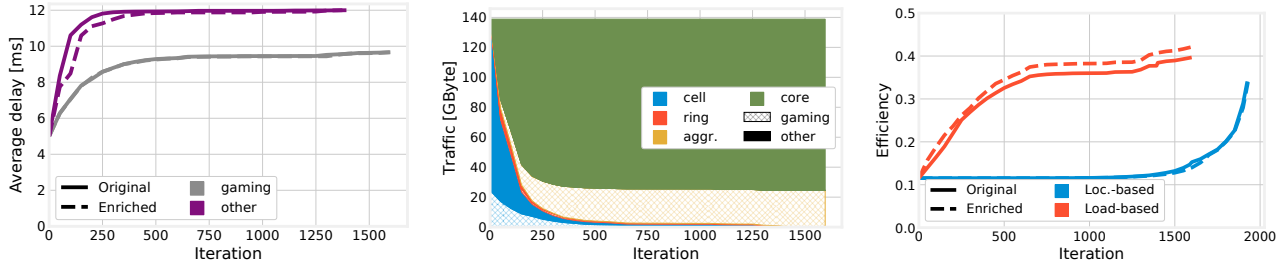

Fig. 7. Finite latency limit for gaming content: latency for each type of content with load-based scores (a); where different types of content are processed in case of load-based scores (b); resulting efficiency (c).

shows a significant number of servers at ring and aggregation nodes, Fig. 5(right) shows a tendency to process traffic either at BSs, or at very few, core-level servers. This is confirmed by Fig. 4(right) and Fig. 5(right): with location-based scores, servers are *lazily* moved from BSs to rings, and then further up, only when required. On the other hand, using load-based scores we aggressively process as much data as possible as high in the topology as possible, so as to smoothen the peaks.

6.2 Efficiency and latency: the importance of ticks

As we have seen, different deployment strategies (i.e., different scores) result in deeply different network planning decisions. In the following, we study (i) how such decisions impact the latency and efficiency of the resulting network, and (ii) if enriching the trace as discussed in Sec. 4 impacts either metric.

We estimate the *latency* using [30] as a reference: the connection between users (UEs in LTE terminology) and base stations (eNBs) requires 5 ms, while additional hops within the backhaul network (e.g., from BSs to rings) require about 2.3 ms each.

As for *efficiency*, we define it as the ratio between the average capacity that is actually required to process the traffic and the total capacity deployed throughout the network:

$$\eta = \frac{\frac{1}{|\mathcal{T}|}\sum_{t\in\mathcal{T}}\sum_{n\in\mathcal{N}}\sum_{k\in\mathcal{K}}\tau_k s(n)_t^k}{\sum_{n\in\mathcal{N}}\max_{t\in\mathcal{T}}\sum_{k\in\mathcal{K}}\tau_k s(n)_t^k}. \quad (4)$$

A network where all servers are always fully utilized would have an efficiency of 1, while a network where servers are almost never used would have almost-zero efficiency. Also notice that in (4) we consider all content categories, and abuse the notation to indicate with $s(n)_t^k$ the amount of data of content of category k that node n must process at time t.

Fig. 6(a) shows that location-based scores are associated with a lower latency than load-based ones. On the other hand, as shown in Fig. 6(b), load-based scores maximize the network efficiency – which, however, hardly exceeds 30%, due to time fluctuations of the demand. It is interesting to compare the solid and dashed lines in Fig. 6(a) and Fig. 6(b), respectively obtained with the original trace and the enriched trace. Using the enriched trace, i.e., ticks instead of megabytes, provides both a lower latency *and* a higher efficiency. This is confirmed by Fig. 6(c), depicting the latency/efficiency trade-offs that can be obtained using different scores and traces.

The difference between location- and load-based performance in Fig. 6(a)–Fig. 6(c) can be interpreted as the benefit obtained by using the enriched trace *in lieu* of the original one. By doing so, we can make better deployment decisions, which consistently result in better efficiency and lower latency.

6.3 The effect of latency limits

We now consider finite latency limits; specifically, we set $L_{\max} = 50$ ms for video and maps and, based on [31], $L_{\max} = 10$ ms for gaming traffic. Fig. 7(a), obtained with load-based scores, shows that different content categories now experience different latencies. In particular, gaming data are always served within its maximum latency limit $L_{\max} = 10$ ms, while other content is served with a higher latency – but still lower than its maximum limit. This is due to the different locations within the network where content is processed, as shown in Fig. 7(b). In the late iterations of the algorithm, we can observe that almost all

gaming content (represented by patterned areas) is served at aggregation nodes, while almost all non-gaming content (solid areas) is served at core nodes. Indeed, based on [30], going from UEs to core nodes entails a latency of 12 ms, which is not compatible with the latency limit for gaming. Furthermore, it is important to note that, even if latency requirements for video and maps would permit the deployment of their servers in the cloud, MEC is still an appealing solution as it avoids transferring large amounts of data over long distances, thus reducing bandwidth consumption.

Being unable to serve gaming content at core servers also has the potential to impair network efficiency; indeed, as it can be seen from Fig. 6(b) and Fig. 6(c), the highest efficiency is reached when most content is processed there. By comparing Fig. 7(c) to Fig. 6(b), we can indeed observe a decrease in the efficiency; however, such a decrease is lower than 10%, which confirms that MEC is able to deliver both low latency and high efficiency.

Finally, it is interesting to observe that the advantage of using enriched traces *in lieu* of the original is clearly visible, in terms of both latency and efficiency.

7 Conclusion and Future Work

One of the challenges we face in the design of next-generation networks is using real-world data about *present-day* technologies to predict their performance. In this work, we took cellular networks as a case study, and demonstrated how a real-world, large-scale dataset about data demand in LTE can be leveraged to design a next-generation, MEC-based network.

To this end, we *enriched* the dataset at our disposal, integrating it with information about the processing power needed to generate traffic of the two main categories of content. We obtained such information through a set of hands-on experiments, based on the Minecraft and FFserver open-source servers. Our results show that using the enriched trace instead of the original one results in better network design, allowing both lower latency *and* higher network efficiency.

References

[1] K. Zheng, T. Taleb, A. Ksentini, C. L. I, T. Magedanz, and M. Ulema, "Research and standards: advanced cloud and virtualization techniques for 5g networks (part ii) [guest editorial]," *IEEE Communications Magazine*, 2015.

[2] F. Malandrino, C. Chiasserini, and S. Kirkpatrick, "The price of fog: A data-driven study on caching architectures in vehicular networks," in *ACM MobiHoc IoV-VoI Workshop*, 2016.

[3] ——, "How close to the edge? Delay/utilization trends in MEC," in *ACM CoNEXT CAN Workshop*, 2016.

[4] V. D. Blondel, M. Esch, C. Chan, F. Clérot, P. Deville, E. Huens, F. Morlot, Z. Smoreda, and C. Ziemlicki, "Data for development: the d4d challenge on mobile phone data," *arXiv preprint*, 2012.

[5] P. D. Francesco, F. Malandrino, T. K. Forde, and L. A. DaSilva, "A sharing- and competition-aware framework for cellular network evolution planning," *IEEE Trans. on Cogn. Comm. and Netw.*, 2015.

[6] S. Kirkpatrick, A. Zmirli, R. Bekkerman, and F. Malandrino, "Mining the Thin Air – for Research in Public Health," *ArXiv preprint 1806.07918*, 2018.

[7] Additional material. https://dl.dropbox.com/s/5uszesbrogp3zch/material.html.

[8] N. Eagle and A. Pentland, "Reality mining: sensing complex social systems," *Personal and ubiquitous computing*, 2006.

[9] ETSI. Mobile edge computing white ppaer. http://www.etsi.org/technologies-clusters/technologies/mobile-edge-computing.

[10] S. Sardellitti, G. Scutari, and S. Barbarossa, "Joint optimization of radio and computational resources for multicell mobile-edge computing," *IEEE Transactions on Signal and Information Processing over Networks*, 2015.

[11] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, 2012.

[12] S. Nunna, A. Kousaridas, M. Ibrahim, M. Dillinger, C. Thuemmler, H. Feussner, and A. Schneider, "Enabling real-time context-aware collaboration through 5g and mobile edge computing," in *ITNG*, 2015.

[13] F. Sardis, G. Mapp, J. Loo, and M. Aiash, "Dynamic edge-caching for mobile users: Minimising inter-as traffic by moving cloud services and vms," in *IEEE WAINA*, 2014.

[14] K. Poularakis, G. Iosifidis, A. Argyriou, I. Koutsopoulos, and L. Tassiulas, "Caching and operator cooperation policies for layered video content delivery," in *IEEE INFOCOM*, 2016.

[15] X. Cai, S. Zhang, and Y. Zhang, "Economic analysis of cache location in mobile network," in *IEEE WCNC*, 2013.

[16] E. Bastug, M. Bennis, and M. Debbah, "Living on the edge: The role of proactive caching in 5G wireless networks," *IEEE Comm. Mag.*, 2014.

[17] M. Ahsan, C. Casetti, C. Chiasserini, P. Giaccone, and J. Haerri, "Mobility-aware edge caching for connected cars," in *IEEE/IFIP WONS*, 2016.

[18] A. Hirwe and K. Kataoka, "LightChain: A lightweight optimization of VNF placement for service chaining in NFV," in *IEEE NetSoft*, 2016.

[19] T. W. Kuo, B. H. Liou, K. C. J. Lin, and M. J. Tsai, "Deploying chains of virtual network functions: On the relation between link and server usage," in *IEEE INFOCOM*, 2016.

[20] A. Baumgartner, V. S. Reddy, and T. Bauschert, "Mobile core network virtualization: A model for combined virtual core network function placement and topology optimization," in *IEEE NetSoft*, 2015.

[21] F. Ben Jemaa, G. Pujolle, and M. Pariente, "Analytical Models for QoS-driven VNF Placement and Provisioning in Wireless Carrier Cloud," in *ACM MSWiM*, 2016.

[22] B. Addis, D. Belabed, M. Bouet, and S. Secci, "Virtual network functions placement and routing optimization," in *IEEE CloudNet*, 2015.

[23] F. Malandrino, C. F. Chiasserini, and S. Kirkpatrick, "Cellular Network Traces Towards 5G: Usage, Analysis and Generation," *IEEE Transactions on Mobile Computing*, 2017.

[24] OpenMapTiles Map Server. https://www.openstreetmap.com.

[25] OpenStreetMap. https://www.openstreetmap.com.

[26] T. Taleb, K. Samdanis, B. Mada, H. Flinck, S. Dutta, and D. Sabella, "On multi-access edge computing: A survey of the emerging 5g network edge cloud architecture and orchestration," *IEEE Communications Surveys & Tutorials*, 2017.

[27] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Transactions on Networking*, 2016.

[28] X. Jin, L. E. Li, L. Vanbever, and J. Rexford, "Softcell: Scalable and flexible cellular core network architecture," in *ACM CoNEXT*, 2013.

[29] P. E. Gill, W. Murray, and M. H. Wright, *Practical optimization*. Academic Press, 1981.

[30] M. Brand and J. Pomy, "One-way Delays in Operating LTE Networks," in *ITU Workshop on Monitoring and Benchmarking of QoS and QoE of Multimedia Services in Mobile Networks*, 2014.

[31] C. Westphal, "Challenges in Networking to Support Augmented Reality and Virtual Reality," in *IETF98-ICNRG Meeting*, 2017.