Power Aware Media Delivery Platform Based on Containers

Jimmy Kjällman, Miika Komu, Tero Kauppinen
NomadicLab, Ericsson Research, Finland
Email: firstname.lastname@ericsson.com

***Abstract*—Use of mobile devices for media consumption has become popular and is expected to grow significantly in the coming years. Optimizing energy consumption in this context can bring several benefits, especially to users of battery-powered devices where the consumption is directly noticeable. In this paper we describe a platform where the cloud computing model and container-based virtualization is utilized for deploying and executing functions that optimize media streams for improved energy efficiency.**

I. Introduction

Energy efficiency is getting considerable attention in the fields of IT and ICT due to goals for improved sustainability and cost savings, and consequently power saving opportunities and mechanisms for, e.g., data centers, networks, and devices are being investigated. In addition, maximizing battery lifetime in, e.g., mobile devices is another important topic is this area.

Another notable trend is that media consumption over the Internet has become very commonplace, and thus a large portion of Internet traffic is comprised of media delivery. For example, the share of video is currently over 60% of consumer Internet traffic according to some estimates, and it is still expected to grow [1]. A significant portion of this media content is delivered via Content Distribution Networks (CDNs). Moreover, video consumption on mobile devices is also increasing rapidly.

Given these premises, attempting to achieve power savings within the context of online content delivery appears to be relevant in general. In this paper, we focus on utilizing power saving methods related to media stream processing, e.g., as a sub-component of CDNs or as a complementary mechanism to CDNs. More specifically, we look mainly at deploying cloud *computing* instances into nodes in the network, i.e., in a *distributed cloud*, in order to optimize media delivery to mobile devices[1]. Existing solutions in this space include, for instance, deployment of proxies or similar functions that shape traffic or transcode media in order to allow devices (and potentially also networks to some extent) to save energy [3][2]. In our work, we plan to reuse such existing functions.

In particular, our goal is to make it possible to deploy such media stream processing functions – which provide the energy savings – automatically and quickly on demand into suitable locations in the network, e.g., when a new media stream is started. The choice of location for a function can based on information about, for instance, available resources and other constraints, location and type of media consumers and groups, media stream characteristics, and power consumption. We also envision that the cloud nodes running these functions can range from constrained Wi-Fi access points (APs) in local networks to more powerful hardware used in data centers (DCs). In other words, the computing infrastructure to be used in this context can be very heterogeneous, but despite that, it would still be desirable to manage and run instances using only one sufficiently generic mechanism.

In order to address these issues we propose that the media processing functions are deployed as *containers*, which have gained attraction in recent years as a lightweight and (at least to some degree) portable application packaging, process isolation, and execution mechanism in clouds. The most notable embodiment of the concept of containers is currently *Docker*[3].

In the following section, we describe the "building blocks" of our proposed platform in a bit more detail and provide arguments for using them in our context. We also give some examples of media processing functions that could be managed and deployed with this system, and what kind of energy savings could thus be achieved.

II. Design

A. Distributed Cloud

As mentioned, we envision that we can utilize a ubiquitous and heterogeneous cloud infrastructure for executing media processing functions. A high-level overview of this kind of a distributed cloud is shown in Figure 1. This cloud environment spans across central data centers (DCs) and regional DCs, edge DCs (e.g., even in nodes at base stations), and local computing facilities, including constrained network nodes such as Wi-Fi APs.

In the last category of nodes mentioned above, for example the *Capillary Networks* architecture for Internet-of-Things (IoT) enables computing in local Linux-based wireless gateways (e.g., Raspberry Pis with ARM CPUs), which are part of the cloud and managed by service providers. [4].

In this context we can note that work towards enabling cloud computing capabilities at the edge of the mobile network

[1]I.e., cloud *storage* (used, e.g., by Cloud CDNs [2]) and distribution of content into storage nodes acting as media sources is out of scope in this paper.

[2]The actual energy savings that can be achieved varies quite much across different techniques.

[3]https://www.docker.com/

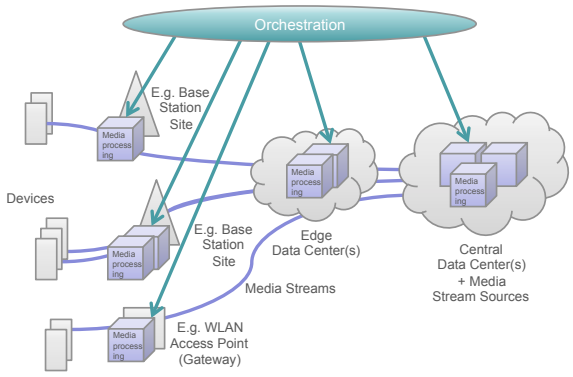

Fig. 1. Media processing functions in a distributed cloud

has started in standardization bodies. For example, ETSI Mobile Edge Computing [5], [6] and IEEE Open Mobile Edge Cloud[4] are such initiatives. Computing near devices, even in constrained nodes, is also part of the *fog computing* concept, which is in focus for the Open Fog Consortium[5]. Thus we can conclude that including nodes beyond DCs into cloud systems may be becoming more common, which would increase the feasibility of realizing our proposed platform.

B. Containers

Assuming that we have a distributed cloud computing infrastructure in place, we need to address how to deploy software instances in it. For a couple of reasons, we propose using containers, such as Docker, and related orchestration mechanisms, as enablers for our proposed architecture.

Firstly, containers provide a service *packaging* mechanism. That is, a standard way of bundling an application and its dependencies (such as libraries) and other files into a container *image*. Such images can be automatically retrieved, e.g., from a container image repository (such as a Docker registry) when needed.

Secondly, containers can be used as a lightweight *virtualization* mechanism for executing applications in an isolated environment in a host operating system. Until quite recently, hypervisor-based virtual machines (VMs) have very commonly been used for this purpose. Containers, however, have a lower overhead in general than VMs: e.g., a lower memory footprint (in particular since they do not run a guest OS image that VMs need[6]) and a lower performance overhead (e.g., wrt. networking) [7], [8]. Moreover, they do not require hardware-level virtualization support in the hosts[7]. All of these characteristics are beneficial when deploying functions into different types of nodes, including constrained nodes, in the cloud.

In addition, containers have fast (even sub-second) startup times, so in general they are nowadays often chosen for services that need to be launched rapidly on demand.

Container instances themselves also have a slightly lower power consumption compared to VMs. For example, a difference in the scale of $\sim$5% has been observed with some networking-intensive workloads [9]. The consumption of containers is close to the consumption of processes running natively on hosts. This can provide a lower energy consumption in the distributed cloud nodes and thus in our system overall compared to using VMs.

C. Container orchestration

While containers provide a basic abstraction for packaging, distributing, and executing services, we of course also need a component that orchestrates the containers and handles issues such as scheduling of container instances into suitable (and available) hosts in the cloud.

Several generic container orchestration mechanisms exist. Examples of open source software in this category are Docker Swarm[8], Google Kubernetes[9], and Apache Mesos[10].

From the point of view of energy efficiency, the orchestration function would ideally launch media processing containers into nodes that would provide optimal end-to-end energy consumption – possibly while still giving more weight to power savings in end devices when needed. In practice, however, the placement algorithm in the orchestrator might not have all the required information to handle the placement completely optimally.

D. Media processing

In order to illustrate and concretize how to use our platform and how to achieve energy savings, we present (on a relatively high level) a few example functions below. But these are, of course, not the only media processing functions that could be deployed with this rather generic platform. Potentially different power consumption optimization functions could also be combined, as in *service chaining*, so that the same stream passes through multiple containers in this system.

1) Traffic shaping proxy: Mobile devices commonly save energy by turning their network interfaces inactive when there is no traffic to send or receive. Thus sending traffic in bursts instead of in a continuous stream can save energy in a mobile terminal. This can be achieved, e.g., by buffering small parts of the stream, e.g., in a proxy. Shaping of traffic into bursts is most beneficial in the close proximity of devices, e.g., in Wi-Fi APs [10]. The reason to push this functionality to the edge is that the traffic might otherwise be subject to other shaping, etc., on its way to the destination. By using this method, power savings up to 65% have been measured in mobile terminals a Wi-Fi network [10].

[4]http://ieee-sdn.blogspot.fi/2015/11/ieee-sdn-initiative-launcing-mobile.html

[5]http://www.openfogconsortium.org/

[6]Notably, another alternative in this space is VMs based on *unikernels*, which have a minimal guest OS that includes only the essential functionality required by each application.

[7]On the other hand, the applications in the containers must be able to run on the same OS (incl. kernel) and OS version that the host has. Typically this OS is a Linux distribution. Furthermore, the isolation provided by Linux containers may currently be less secure than with hypervisor-based VMs.

[8]https://www.docker.com/products/docker-swarm

[9]http://kubernetes.io/

[10]http://mesos.apache.org/

With the proposed platform, a container containing a traffic-shaping proxy function could be deployed automatically into a local Wi-Fi AP that is part of the distributed cloud. In that case the orchestrator needs to know which AP the user is connected to. If device connectivity is managed centrally, as in the case of the Capillary Networks architecture [4], this information is readily available from a *connectivity manager* function. Alternatively, if a device sends the request for the stream over HTTP, the AP itself (e.g., a static web proxy in it) can add its own identity to the outgoing request, and this information can be passed to the orchestrator. The client can be explicitly redirected to obtain the content via the traffic shaping proxy when it has been instantiated in the AP.

2) Stream content adaptation: Another method used for optimizing media streams is to alter to stream content, such as its encoding, in the network to be more power-friendly to receivers. As a very simple example, a high-quality stream from a content server could be transcoded in a proxy for a receiver (or even a group of receivers) that does not desire to receive a high-quality stream [3].

In this case, a transcoding container can be deployed close to a stream source, if possible. This can potentially save energy in the network, in addition to devices.

3) Stream pausing: An even more simple approach to potentially save energy in devices is to just pause a video stream when the content is not changing. This can be the case when monitoring a stream from a surveillance camera, for instance.

In this case the container processing the video stream can also be deployed close to a stream source. In the most straightforward case, the container can, for example, inspect the video stream and determine whether the video is changing enough to justify sending it onward.

III. Implementation

We have created an initial implementation of the platform based on Docker containers and the Docker Swarm orchestrator, which allows using the standard Docker API for deploying containers into clusters of hosts that run the Docker daemon.

Our implementation is currently based on a two-tier model, where a separate orchestrator determines the nodes or locations (e.g., a local gateway or a central DC) where containers should be deployed, and then passes this information to Swarm, as Swarm's API provides the possibility to specify *constraints* and *affinities* for controlling placement of containers in a cluster. In other words, our orchestrator is the component that will implement placement algorithms that are specific for our media processing cases, while Swarm simply launches container instances in the specified hosts.

IV. Conclusion and Future Work

In this paper we present an initial design of a platform that can be used for flexibly deploying media processing functions as containers in a distributed cloud. These functions can then modify media streams in order to enable power savings in terminals.

Future work includes implementation and evaluation of actual use cases and development and/or application of suitable placement algorithms and mechanisms for the media processing functions. In other words, there are still many details to work out in this area – the purpose of this paper is just introduce the concept and some initial thinking around it.

Acknowledgment

This work is supported by the European Celtic-Plus project CONVINcE (CP2013/2-1) and was partially funded by Finland (Tekes), France, Sweden and Turkey.

References

[1] *Cisco Visual Networking Index: Forecast and Methodology, 2014-2019 (White Paper)*, Cisco, 2015. [Online]. Available: http://www.cisco.com/c/en/us/solutions/collateral/service-provider/ip-ngn-ip-next-generation-network/white_paper_c11-481360.pdf

[2] M. Wang, P. P. Jayaraman, R. Ranjan, K. Mitra, M. Zhang, E. Li, S. Khan, M. Pathan, and D. Georgeakopoulos, "An overview of cloud based content delivery networks: Research dimensions and state-of-the-art," *Transactions on Large-Scale Data- and Knowledge-Centered Systems XX*, vol. 9070 of Lecture Notes in Computer Science, 2015.

[3] M. A. Hoque, M. Siekkinen, and J. K. Nurminen, "Energy efficient multimedia streaming to mobile devices - a survey," *IEEE Communications Surveys & Tutorials*, vol. 16, 2012.

[4] O. Novo, N. Beijar, M. Ocak, J. Kjällman, M. Komu, and T. Kauppinen, "Capillary networks – bridging the cellular and iot worlds," in *2nd IEEE World Forum on the Internet of Things*, 2015.

[5] *Mobile-Edge Computing - Introductory Technical White Paper*, ETSI, 2014. [Online]. Available: https://portal.etsi.org/Portals/0/TBpages/MEC/Docs/Mobile-edge_Computing_-_Introductory_Technical_White_Paper_V1%2018-09-14.pdf

[6] *ETSI GS MEC-IEG 004 V1.1.1 (2015-11) Mobile Edge Computing (MEC); Service Scenarios*, ETSI, 2015. [Online]. Available: http://www.etsi.org/deliver/etsi_gs/MEC-IEG/001_099/004/01.01.01_60/gs_MEC-IEG004v010101p.pdf

[7] R. Morabito, J. Kjällman, and M. Komu, "Hypervisors vs. lightweight virtualization: A performance comparison," in *"First Workshop on Containers (WoC)"*, 2015.

[8] W. Felter, R. Ferreira, R. Rajamony, and J. Rubio, "An updated performance comparison of virtual machines and linux containers," IBM, 2014. [Online]. Available: http://domino.research.ibm.com/library/cyberdig.nsf/%20papers/0929052195DD819C85257D2300681E7B/$File/rc25482.pdf

[9] R. Morabito, "Power consumption of virtualization technologies: an empirical investigation," in *"First International Workshop on Sustainable Data Centres and Cloud Computing (SD3C)"*, 2015.

[10] M. A. Hoque, M. Siekkinen, and J. K. Nurminen, "On the energy efficiency of proxy-based traffic shaping for mobile audio streaming," in *IEEE Consumer Communications and Networking Conference (CCNC)*, 2011.