Perfil de Consumo de Uma Aplicação de Reconhecimento Facial no Contexto de Computação de Borda

Karlla Bianca C. Rodrigues[1], **Kleber Vieira Cardoso**[1], **Sand Luz Corrêa**[1]

[1]Instituto de Informática – Universidade Federal de Goiás (UFG)
Caixa Postal 15.064 – 91.501-970 – Goiânia – GO – Brazil

{karllarodrigues,kleber,sand}@inf.ufg.br

Abstract. *Multi-Access Edge Computing (MEC) is an emerging paradigm aimed at providing computing and storage capabilities on the edge of cellular networks, reducing latency for end users of mobile networks. Designing a MEC network involves decisions on the capacity of the computational resources that will be employed at the network edge. This process, in turn, requires knowledge on the resource usage profile of the applications or services that will perform in the MEC infrastructure. In this paper, we present the experimental results of an investigation on resource consumption profile of a facial recognition application running on a MEC infrastructure. For practical insights, the research was centered on the widely used open source software called OpenFace.*

Resumo. *Multi-Access Edge Computing (MEC) é um paradigma emergente que visa o provimento de recursos computacionais na borda das redes de telefonia celular, reduzindo a latência para usuários finais. O projeto de uma rede MEC envolve decisões sobre o dimensionamento dos recursos computacionais que serão empregados na borda da rede. Esse dimensionamento requer o conhecimento do perfil das aplicações ou serviços que executarão na infraestrutura MEC. Neste trabalho, apresentamos os resultados de uma investigação sobre o perfil de consumo de recursos computacionais de uma aplicação de reconhecimento facial que executa em infraestruturas MEC. Para obtermos intuições práticas, utilizamos um software de código aberto denominado OpenFace.*

1. Introdução

O foco principal da próxima geração de redes móveis, denominada redes 5G, é fornecer comunicação ubíqua para máquinas e dispositivos da Internet das Coisas (*Internet of Things* - IoT) e para a comunicação pessoal. Ao mesmo tempo, tecnologias emergentes (e.g., dispositivos vestíveis e realidade virtual/aumentada) e nova aplicações (e.g., Internet tátil, telemedicina e carros autônomos) possibilitam formas diferentes de interação e uso das redes móveis, exigindo requisitos mais exigentes para a satisfação do usuário. Essa realidade desafia as redes 5G em vários aspectos, uma vez que para atender tais requisitos, elas terão que prover altas taxas de dados, menor latência e maior confiabilidade [Zhang et al. 2014].

Particularmente, para satisfazer o requisito de baixa latência, mudanças significativas devem ser implementadas na arquitetura das redes móveis celulares atuais, uma vez que a latência dominante encontra-se entre a rede de acesso (*Radio Access Network*

– RAN) e o núcleo da rede (*core*) [Parvez et al. 2018]. Um paradigma proposto recentemente para reduzir a latência nas redes móveis é denominado *Multi-access Edge Computing* (MEC) [ETSI 2016]. Neste paradigma, serviços, que anteriormente executavam em data centers remotos (*core*), são movidos para a borda da rede (estações bases ou switches de núcleo) com o objetivo de ficarem o mais próximo possível do usuário. Em MEC, assim como em cenários de computação em nuvem, existe uma cooperação entre provedores de conteúdo e operadores de rede de telefonia. Os servidores implantados pelos operadores de rede executam serviços desenvolvidos pelos provedores de conteúdo. Esses serviços são geralmente implementados como um conjunto de funções virtualizadas de rede (VNFs) que executam em máquinas virtuais ou contêineres.

Uma decisão importante no projeto de redes MEC é onde, dentro da rede de borda, colocar os servidores e como dimensioná-los. Para responder essas questões é necessário conhecer o perfil da demanda de rede e de processamento das aplicações que serão instanciadas na borda da rede. No contexto de MEC, o perfil da demanda de rede tem sido geralmente estudado usando registros de dados oferecidos por operadoras de rede [Blondel et al. 2012, Di Francesco et al. 2015] ou obtidos de forma colaborativa (*crowd-sourced*) [Malandrino et al. 2016]. A demanda de processamento, no entanto, tem sido pouco investigada, tendo sido tratada recentemente em [Malandrino et al. 2018]. No trabalho em questão, os autores apresentam um modelo para estimar a demanda de processamento necessária para servir uma certa demanda de rede. Os autores consideram três tipos principais de tráfego, a saber, vídeo, mapas e jogos.

De fato, de acordo com estimativas da CISCO, em 2022, 79% do tráfego global em redes móveis será de vídeo [CISCO 2019]. Em [Malandrino et al. 2018], os autores consideram o tráfego de vídeo gerado apenas por serviços de *streaming*. No entanto, dado o número crescente de câmeras de monitoramento em diversas cidades, um cenário comum em cidades inteligentes será serviços de reconhecimento facial executando em servidores hospedados na borda da rede (servidores MEC) e conectados a câmeras e dispositivos IoT. Sistemas de reconhecimento facial identificam automaticamente faces a partir de um conjunto de imagens. Para atingir esse objetivo, esses sistemas geralmente executam em duas fases. Na primeira, denominada detecção, o sistema busca localizar a parte da imagem que contém a face. Na segunda, denominada reconhecimento, o sistema, com o auxilio de um algoritmo de aprendizado de máquina, compara a face detectada com um conjunto prévio de outras faces já conhecidas. Portanto, um sistema de reconhecimento facial possui um comportamento muito diferente de serviços de vídeo por *streaming*, não só por fazerem uso de conteúdo ao vivo, mas também por utilizarem algoritmos de aprendizado de máquina que podem requerer uma quantidade significativa de processamento. Apesar dessas diferenças, até onde sabemos, apenas um trabalho na literatura [Garcia-Saavedra et al. 2018] abordou a caracterização do perfil de consumo de sistemas de reconhecimento facial.

Neste trabalho, preenchemos essa lacuna apresentando os resultados experimentais de uma investigação sobre o perfil de consumo de recursos computacionais de uma aplicação de reconhecimento facial que executa em infraestruturas MEC. Para obtermos intuições práticas, a investigação foi centralizada em um software de código aberto amplamente utilizado, denominado OpenFace [Baltrusaitis et al. 2018].

Este trabalho está organizado da seguinte forma. A Seção 2 introduz conceitos

básicos referentes ao paradigma MEC bem como trabalhos relacionados à caracterização do perfil de consumo de aplicações que executam em infraestruturas MEC. A Seção 3 apresenta uma descrição do serviço OpenFace. A metodologia utilizada nesta investigação, bem como os resultados experimentais obtidos são apresentados na Seção 4. Finalmente, a Seção 5 apresenta as conclusões e direções futuras para este trabalho.

2. Fundamentos e Trabalhos Relacionados

Uma das primeiras motivações para o surgimento da computação de borda foi a necessidade de reduzir a latência fim-a-fim de aplicações móveis que necessitavam descarregar dados nas nuvens. Em 2009, o trabalho de [Satyanarayanan et al. 2009] mostrou que a implantação de uma infraestrutura semelhante às usadas em ambientes de computação em nuvem, porém em uma escala menor, a apenas um salto dos dispositivos móveis sem fio, poderia reduzir sensivelmente a latência fim-a-fim das aplicações que executam nesses dispositivos. Em 2015, a *European Telecommunications Standards* (ETSI) denominou de *Multi-Acces Edge Computing*, uma instância de tal infraestrutura de borda. A ideia básica consiste em prover recursos computacionais e de armazenamento nas redes de acessos (RAN), de forma que serviços e aplicações que demandem baixa latência funcionem mais próximos dos usuários [Patel et al. 2014]. A Figura 1 exemplifica uma instância de uma infraestrutura MEC conectada à uma infraestrutura de computação em nuvem.

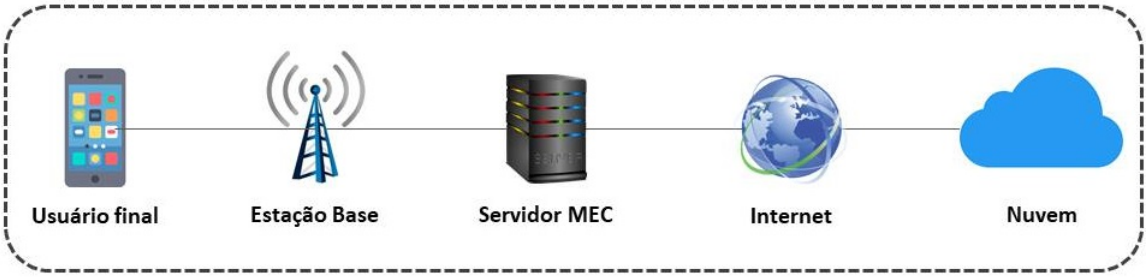


Figura 1. Fluxo básico de um serviço MEC.

Além de cunhar o termo MEC, a ETSI também definiu uma arquitetura de referência para infraestruturas MEC. Os elementos dessa arquitetura estão agrupados em três níveis principais, como mostrado na Figura 2 e descrito a seguir.

- Módulo de gerenciamento do sistema MEC: engloba as entidades relacionadas ao gerenciamento do ciclo de vida das aplicações e o orquestrador MEC. O orquestrador possui o conhecimento geral e informações atualizadas sobre a topologia de rede e os recursos disponíveis nos servidores de borda (servidores MEC). Ele também mantém uma lista das aplicações correntes, garante as políticas definidas pelos operadores e prepara os gerenciadores de infraestrutura virtual (*Virtual Infrastructure Manager* – VIM) para lidar com a instanciação de aplicações. O orquestrador é responsável pela instanciação, término e realocação de aplicações com base nos requisitos dos serviços e disponibilidade geral dos recursos.
- Módulo de gerenciamento dos *hosts* MEC: engloba as entidades responsáveis pelo gerenciamento dos servidores MEC onde serão alocadas as aplicações. Os servidores MEC contém a infraestrutura de virtualização responsável por fornecer recursos como CPU, memória, rede e armazenamento para as aplicações que

executam na borda. Possui também as funcionalidades essenciais para executar as aplicações em uma infraestrutura virtual específica (máquinas virtuais ou contêineres).

- Infraestrutura de rede: encarregada de fornecer recursos para a conexão das aplicações com os diferentes tipos de redes.

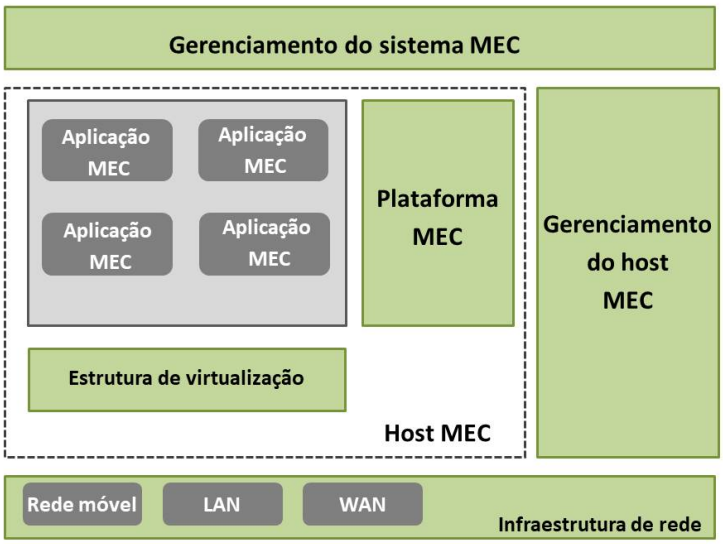


Figura 2. Arquitetura MEC simplificada.

Nos últimos anos, vários trabalhos foram publicados com foco em aspectos técnicos de computação de borda [Taleb et al. 2017], arquiteturas MEC [Abbas et al. 2018], revisões de atributos e casos de uso que se beneficiam da computação de borda [Liu et al. 2018], bem como modelos matemáticos para alocação de recursos em ambientes MEC [Garcia-Saavedra et al. 2018]. Particularmente, em [Garcia-Saavedra et al. 2018], os autores apresentam uma breve caracterização do perfil de consumo de uma aplicação de reconhecimento facial. No entanto, o foco do trabalho é um modelos de alocação de recursos que comporte aplicações MEC e *virtual RAN* (vRAN) e o perfil de consumo da aplicação é realizado apenas para parametrizar o modelo de alocação proposto. Como consequência, os autores não discutem, de forma abrangente, a metodologia utilizada na caracterização da aplicação.

Em [Malandrino et al. 2018], os autores apresentam uma metodologia para avaliar o consumo de CPU a partir de demandas geradas pela rede, ou seja a partir do volume de dados baixados de aplicativos de dispositivos móveis. Os autores consideram três tipo de aplicativos: vídeo, jogo e mapa. Apesar de focarem na demanda de CPU de aplicações MEC, os serviços investigados em [Malandrino et al. 2018] apresentam comportamentos significativamente diferentes daqueles encontrados em aplicações de reconhecimento facial.

De fato, dado o número crescente de câmeras conectadas à Internet e instaladas em espaços públicos, cenários envolvendo esse tipo de câmera enviando dados em tempo real para uma infraestrutura de computação de borda têm sido recorrentemente referenciados como casos de uso MEC. Em [Dautov et al. 2018], os autores utilizam esse cenário

para validar uma arquitetura de computação baseada em *cluster* de servidores na borda da rede de acesso. Outro trabalho [Muslim and Islam 2017], utiliza um cenário similar para investigar os benefícios de se processar uma aplicação de reconhecimento facial na borda da rede em comparação com o processamento local no dispositivo móvel (celular). Nenhum dos dois trabalhos, no entanto, caracterizam o perfil de consumo de recursos computacionais de aplicações de reconhecimento facial. Na seção seguinte, descrevemos a aplicação de reconhecimento facial utilizada neste trabalho.

3. OpenFace

Aplicações de vídeo produzem uma quantidade expressiva de dados, que frequentemente precisam ser processados em tempo real. Isso implica em um alto consumo de recursos computacionais, levando em consideração o fato de ser necessária a manutenção de um fluxo de dados entre dois canais e também o armazenamento em memórias secundária [Wuytack et al. 1996].Além da necessidade de se possuir uma boa infraestrutura que forneça esse recursos de forma segura, o processamento de vídeo exige uma latência muito baixa, já que na maioria das vezes a saída do processamento interage com os humanos (vídeos de realidade aumentada, por exemplo) e também uma grande largura de banda, por se tratar muitas vezes de vídeos com alta resolução [Ananthanarayanan et al. 2017]. Essas duas características demonstram que aplicações de vídeo se beneficiam de infraestruturas MEC.

Dentro do espectro do processamento de vídeo podemos destacar um tipo de serviço bastante relevante nos dias atuais: o reconhecimento facial. Como mencionado anteriormente, sistemas de reconhecimento facial são capazes de detectar e identificar faces em vídeos gravados ou em tempo real. O funcionamento desses sistemas consiste em aplicar diferenciados algoritmos que detectam expressões faciais e, a partir disso, verificar e comparar cada quadro de um determinado vídeo. O OpenFace é uma ferramenta disponível em código aberto que realiza esse tipo de processamento. Na primeira parte do processo, a ferramenta detecta as faces e realiza um pré-processamento criando uma entrada normalizada para treinar os algoritmos. Depois do pré-processamento, é obtida uma imagem em alta resolução que não pode ser fornecida ao algoritmo. Portanto, a segunda parte do processo é reduzir a dimensão da entrada para que a classificação das faces possa ser realizada[Amos et al. 2016].

Uma das funcionalidades dessa aplicação nos permite utilizar o serviço de reconhecimento facial em tempo real através da criação de um servidor que disponibiliza o serviço de forma local ou na Web. Ao acessarmos esse serviço através de um navegador, com uma reprodução de vídeo já iniciada, o software detectada as faces mostradas e as marca com um pequeno quadrado verde que recebe uma etiqueta de “Desconhecido”, ou seja, aquele rosto ainda não recebeu nenhuma identificação. Caso o usuário decida treinar a aplicação, ele pode identificar uma pessoa, mas a aplicação é capaz de reconhecer somente uma face, etiquetando todos os outros diferentes rostos com a mesma identificação. A atribuição dada a uma face só é válida para a sessão do navegador que se usa naquele instante de tempo, não sendo salva nenhuma identificação após o encerramento do serviço.

Existem duas maneiras de utilização da versão Web do OpenFace. A primeira é uma configuração de ambiente manual, que exige a instalação de todos os pacotes e de-

pendências, as quais os desenvolvedores disponibilizam os scripts e o código necessário para a criação do ambiente. A segunda maneira é utilizar um contêiner Docker. O Docker é uma ferramenta de virtualização que oferece o conceito de conteinerização para isolar aplicações dentro de um mesmo ambiente. Para que um determinado contêiner possa funcionar, é criada uma imagem executável que possui todo o necessário para o funcionamento da aplicação que se deseja executar [Docker 2019]. Para a realização dos nossos experimentos utilizamos a estratégia de conteinerização do servidor OpenFace, como explicaremos na seção a seguir.

4. Avaliação Experimental

Para realizar a caracterização do perfil de consumo do OpenFace, preparamos o ambiente de teste (*testbed*) conforme ilustrado na Figura 3. Nessa *testbed*, dois computadores, um cliente e um servidor, foram conectados através de uma rede local com velocidade de 100 Mbps. Como cliente, usamos um computador desktop com um processador Intel Core i7 3770, memória RAM de 8GB e sistema operacional Linux com a distribuição Ubuntu 16.04. Para o servidor, usamos um computador com dois processadores Intel Xeon Silver 4114, memória RAM de 128 GB e o sistema operacional Linux com a distribuição Ubuntu 18. Os sistemas de virtualização utilizados no servidor consistiu em hipervisor Xen e a plataforma Docker.

O cenário emulado corresponde a um usuário com um dispositivo móvel (celular) capturando um vídeo através de uma sessão de navegação Web. Os quadros capturados são enviados a um servidor MEC, onde o serviço OpenFace executa. O serviço OpenFace recebe os quadros enviados e processa a detecção e o reconhecimento facial. Após o reconhecimento, o OpenFace envia, como resposta ao navegador, uma área quadrada delimitando a face detectada, bem como um rótulo de reconhecimento.

Em nosso experimento, o dispositivo móvel do usuário foi emulado pela máquina cliente do nosso *testbed*. Para emular a câmera de vídeo do dispositivo, usamos o v4l2loopback [GitHub 2016], um módulo do *kernel* Linux que permite a criação de dispositivos virtuais de câmera. Esses dispositivos virtuais podem ser alimentados com *streamings* de vídeos e são enxergados pelas aplicações como um dispositivo real instalado no computador. Para emular a captura de um vídeo no dispositivo móvel, utilizamos a ferramenta FFmpeg [Coudurier et al. 2019], amplamente utilizada para a gravação, conversão e transmissão de vídeo e áudio. Por fim, o navegador Firefox foi utilizado para emular uma sessão Web onde o vídeo é capturado e requisições são enviadas ao servidor OpenFace para análise dos quadros capturados.

O servidor MEC foi emulado pela máquina servidora do nosso *testbed*. Nesse servidor, criamos uma máquina virtual com quatro CPUs virtuais, memória RAM de 16 GB, e instalamos o sistema operacional Linux com a distribuição Debian 10. O serviço OpenFace foi implantado dentro dessa máquina virtual usando uma imagem Docker disponível em [dockerhub 2019].

Para que o serviço OpenFace pudesse realizar a detecção e reconhecimento de faces durante o experimento, escolhemos um vídeo extraído do Youtube e que transmitia, durante toda sua extensão, rostos de pessoas conversando de forma aleatória. Para esse experimento, não consideramos a parte de áudio do vídeo. O vídeo foi então convertido para o codec H264/AVC, com resolução de 1920x1080, ou seja, um conteúdo com

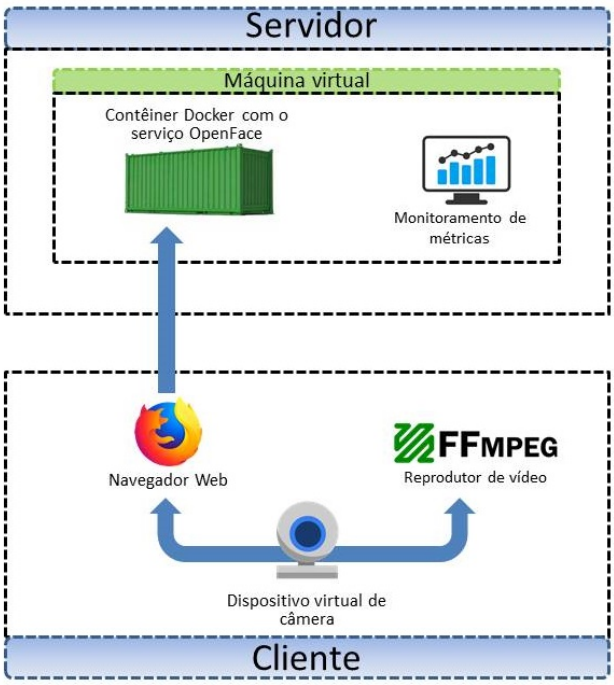


Figura 3. Configuração do ambiente de testes

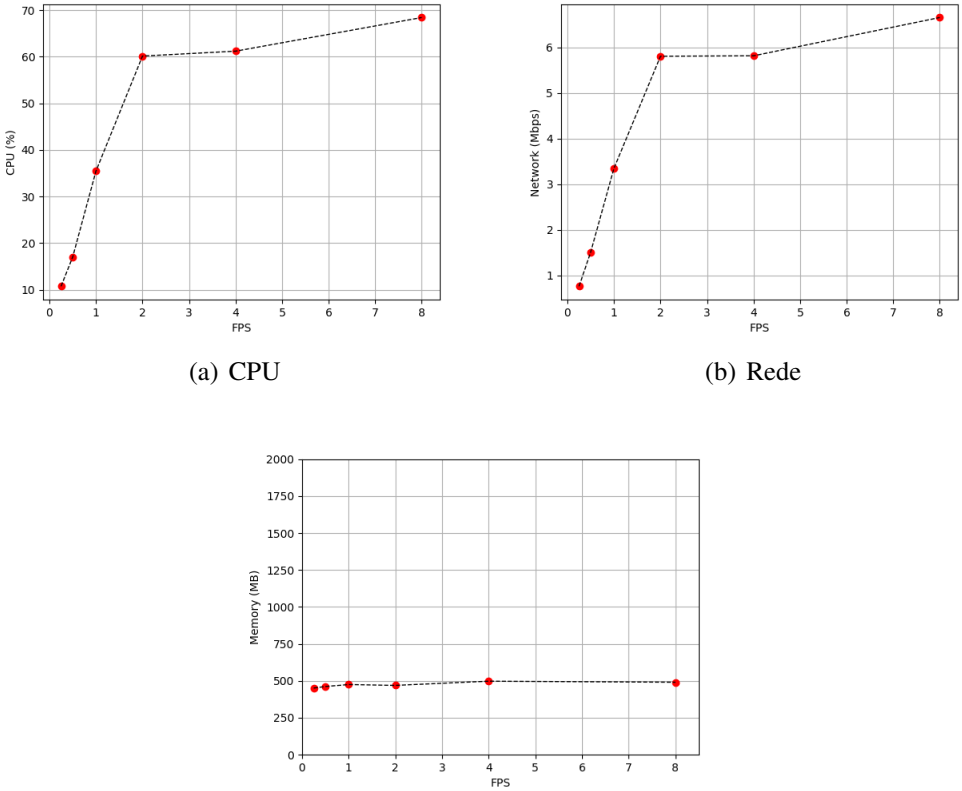
alta resolução. Através de um comando FFmpeg, iniciamos a reprodução do vídeo, alimentando o dispositivo virtual de câmera no lado do cliente. Em seguida, usando o IP e porta onde o serviço OpenFace executava, abrimos uma sessão no navegador Web com o intuito de enviar as requisições ao servidor. O serviço OpenFace recebia os frames enviados, realizava a detecção e reconhecimento facial e enviava a resposta (área retangular e rótulo) ao navegador para cada frame recebido. Nesse experimento, no entanto, não utilizamos nenhum treinamento prévio, pois o intuito foi avaliar o consumo de recursos computacionais do serviço e não a acurácia do serviço.

Para mensurar o consumo de CPU (porcentagem de utilização), utilização de memória (megabytes), bem como a demanda de rede (mbps) no lado do servidor, executamos um script de monitoramento do sistema dentro da máquina virtual onde o serviço OpenFace executava. O script realizava a coleta de dados a cada dois segundos. Além disso, para entender como o consumo desses recursos variavam à medida que a demanda de rede aumentava, variamos a taxa de frames por segundo (FPS) enviado ao servidor, empregando, ao todo, seis taxas diferentes, a saber:

$$FPS = [0,25; 0,5; 1; 2; 4; 8] \tag{1}$$

Para cada taxa de FPS, realizamos dez testes, cada um com 100 segundos de duração. Todo fluxo de dado foi gerado por apenas um único cliente, ou seja, avaliamos as métricas existentes de uma única sessão do navegador Web. A Figura 4 ilustra os resultados. Podemos notar que o consumo de CPU tem um comportamento muito similar à demanda de rede à medida que a taxa FPS aumenta. Isso evidencia um relacionamento crescente linear entre consumo de CPU e demanda de rede. Percebemos também que o consumo de memória se mantém praticamente constante à medida que a taxa FPS

aumenta.



(a) CPU

(b) Rede

(c) Memória

Figura 4. Gráficos que ilustram o consumo de CPU, rede e memória, de acordo com a variação de FPS

5. Conclusão e Trabalhos Futuros

Neste trabalho, apresentamos a caracterização do perfil de consumo de uma aplicação de reconhecimento facial em um ambiente típico de MEC. Resultados iniciais indicam uma relação de linearidade crescente entre o consumo de CPU e o tráfego (demanda) de rede. Por outro lado, o aumento da taxa FPS tem pouco impacto no consumo de memória.

Como trabalhos futuros, pretendemos apresentar uma caracterização mais abrangente do perfil de consumo de recursos de aplicações de reconhecimento facial variando também o número de usuários, a qualidade dos vídeos, mensurando o atraso e realizando a caracterização por fases, ou seja, detecção e reconhecimento. Pretendemos também comparar os resultados obtidos com o trabalho em [Malandrino et al. 2018].

Referências

Abbas, N., Zhang, Y., Taherkordi, A., and Skeie, T. (2018). Mobile edge computing: A survey. *IEEE Internet of Things Journal*, 5(1):450–465.

Amos, B., Ludwiczuk, B., and Satyanarayanan, M. (2016). Openface: A general-purpose face recognition library with mobile applications.

Ananthanarayanan, G., Bahl, P., Bodík, P., Chintalapudi, K., Philipose, M., Ravindranath, L., and Sinha, S. (2017). Real-time video analytics: The killer app for edge computing. *IEEE Computing Society*.

Baltrusaitis, T., Zadeh, A., Lim, Y. C., and Morency, L. (2018). Openface 2.0: Facial behavior analysis toolkit. In *2018 13th IEEE International Conference on Automatic Face Gesture Recognition (FG 2018)*, pages 59–66.

Blondel, V. D., Esch, M., Chan, C., Clerot, F., Deville, P., Huens, E., Morlot, F., Smoreda, Z., and Ziemlicki, C. (2012). Data for Development: the D4D Challenge on Mobile Phone Data. *arXiv e-prints*, page arXiv:1210.0137.

CISCO (2019). Cisco visual networking index: Global mobile data traffic forecast update, 2017–2022 white paper. https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-738429.html#_Toc953326.

Coudurier, B., Hoyos, C. E., Niedermayer, M., and Mahol, P. B. (2019). Ffmpeg documentation. https://www.ffmpeg.org/documentation.html.

Dautov, R., Distefano, S., Bruneo, D., Longo, F., Merlino, G., and Puliafito, A. (2018). Data processing in cyber-physical-social systems through edge computing. *IEEE Access*, 6:29822–29835.

Di Francesco, P., Malandrino, F., Forde, T. K., and DaSilva, L. A. (2015). A sharing- and competition-aware framework for cellular network evolution planning. *IEEE Transactions on Cognitive Communications and Networking*, 1(2):230–243.

Docker (2019). Docker documentation. https://docs.docker.com/get-started/. Online.

dockerhub (2019). Openface. https://hub.docker.com/r/bamos/openface.

ETSI (2016). Multi access edge computing (mec). Technical report, European Telecommunications Standards Intitute.

Garcia-Saavedra, A., Iosifidis, G., Costa-Perez, X., and Leith, D. J. (2018). Joint optimization of edge computing architectures and radio access networks. *IEEE Journal on Selected Areas in Communications*, 36(11):2433–2443.

GitHub (2005–2016). V4l2loopback: Virtual video devices. https://github.com/umlaeute/v4l2loopback.

Liu, H., Eldarrat, F., Alqahtani, H., Reznik, A., Foy, X., and Zhang, Y. (2018). Mobile edge cloud system: Architectures, challenges, and approaches. *IEEE Systems Journal*, 12(3):2495–2508.

Malandrino, F., Chiasserini, C., and Kirkpatrick, S. (2016). The price of fog: A data-driven study on caching architectures in vehicular networks. In *Proceedings of the First International Workshop on Internet of Vehicles and Vehicles of Internet*, pages 37–42.

Malandrino, F., Chiasserini, C.-F., Avino, G., Malinverno, M., and Kirkpatrick, S. (2018). From megabits to cpu ticks:enriching a demand trace in the age of mec. *IEEE Transactions On Big Data*, pages 2332–7790.

Muslim, N. and Islam, S. (2017). Face recognition in the edge cloud. *International Conference on Imaging, Signal Processing and Communication*.

Parvez, I., Rahmati, A., Guvenc, I., Sarwat, A. I., and Dai, H. (2018). A survey on low latency towards 5g: Ran, core network and caching solutions. *IEEE Communications Surveys Tutorials*, 20(4):3098–3130.

Patel, M., Joubert, J., Ramos, J. R., Sprecher, N., Abeta, S., and Neal, A. (2014). Mobile edge computing - introductory technical white paper. Technical report, European Telecommunications Standards.

Satyanarayanan, M., Bahl, P., Caceres, R., and Davies, N. (2009). The case for vm-based cloudlets in mobile computing. *IEEE Pervasive Computing*, 8(4):14–23.

Taleb, T., Samdanis, K., Mada, B., Flinck, H., Dutta, S., and Sabella, D. (2017). On multi-access edge computing: A survey of the emerging 5g network edge cloud architecture and orchestration. *IEEE Communications Surveys Tutorials*, 19(3):1657–1681.

Wuytack, S., Catthoor, F., Nachtergaele, L., and Man, H. D. (1996). Power exploration for data dominated 'video applications. *International Symposium on Low Power Eletronics and Design*.

Zhang, S., Xu, X., Wu, Y., and Lu, L. (2014). 5g: Towards energy-efficient, low-latency and high-reliable communications networks. In *2014 IEEE International Conference on Communication Systems*, pages 197–201.