This article has been accepted for publication in the IEEE International Conference on Computer Communications (INFOCOM Workshops), 2018.

eBPF-based Content and Computation-aware Communication for Real-time Edge Computing

Sabur Baidya[1], Yan Chen[2] and Marco Levorato[1]

[1]Donald Bren School of Information and Computer Science, UC Irvine
e-mail: {*sbaidya, levorato*}*@uci.edu*
[2]America Software Laboratory, Huawei, e-mail: *y.chen@huawei.com*

***Abstract*—By placing computation resources within a one-hop wireless topology, the recent edge computing paradigm is a key enabler of real-time Internet of Things (IoT) applications. In the context of IoT scenarios where the same information from a sensor is used by multiple applications at different locations, the data stream needs to be replicated. However, the transportation of parallel streams might not be feasible due to limitations in the capacity of the network transporting the data. To address this issue, a content and computation-aware communication control framework is proposed based on the Software Defined Network (SDN) paradigm. The framework supports multi-streaming using the extended Berkeley Packet Filter (eBPF), where the traffic flow and packet replication for each specific computation process is controlled by a program running inside an in-kernel Virtual Machine (VM). The proposed framework is instantiated to address a case-study scenario where video streams from multiple cameras are transmitted to the edge processor for real-time analysis. Numerical results demonstrate the advantage of the proposed framework in terms of programmability, network bandwidth and system resource savings.**

I. Introduction

In the Internet of Things (IoT), interconnected sensors collect and transmit data for analysis to remote servers [1]. However, the delivery of content-rich data may incur delay or loss due to the limited network resources. Indeed, network congestion may impair the ability of the system to support real-time services, such as video surveillance, traffic monitoring, smart transportation or the virtual reality (VR) and augmented reality (AR) [2]. For this family of applications, the cloudlets [3] and edge computing [4] paradigms mitigate the issue of latency by placing computation-capable devices within a one-hop wireless topology. However, in many network scenarios of interest, the coexistence of these demanding data streams with other services over constrained wireless networks necessitates new technical solutions.

Recent frameworks based on Software Defined Networks (SDN) [5] have demonstrated the ability to improve network resource management using dynamic flow control and Network Function Virtualization (NFV) [6]. At the communication level, Software Defined Radios (SDR) [7] have been used to dynamically adapt the parameters of wireless transmissions. However, the main challenge of effectively utilizing the available bandwidth to support real-time applications producing large volume traffic stands. To this aim, the notion of Quality of Computing (QoC) [8] has been recently proposed to relax interference constraints on IoT data streams and facilitate their coexistence.

In this paper, we propose a computation-aware communication control framework for real-time IoT applications generating high-volume data traffic processed at the network edge. Driven by QoC requirements, the framework provides real-time user-controlled packet replication and forwarding inside the in-kernel Virtual Machines (VM) using an extended Berkeley Packet Filter (eBPF) [9]. The implementation uses the concepts of SDN and NFV to achieve highly programmable and dynamic packet replication. Resource allocation is semantic and content-aware, and, in the considered case, informed by the structure of data encoding. Numerical results are provided from real-world experiments demonstrating the enhanced adaptability and efficiency of the proposed solution.

The rest of the paper is organized as follows. In section II, we describe the real-time edge computing scenario and formulate the problem. Section III discusses related work. In Section IV, we present the architecture of the proposed framework and in section V, we describe the computation-aware communication protocol and provide the implementation details. Section VI provides numerical results validating the proposed approach. Section VII concludes the paper.

II. Edge Computing for Real-time Applications

One of the core concepts used in this paper is edge computing, which can reduce the computation latency of real-time applications. It is generally assumed that the edge servers are more powerful compared to the sensors, but offer inferior performance compared to cloud servers in terms of computation capabilities. We consider an application scenario where video data from different video sensors (cameras) are sent to the edge server, which runs in parallel several real-time processes analyzing the streams as shown in Fig. 1. Each computation process may use partial data from multiple sensors to accomplish its task. For instance, a computation process aimed at fine-grained object detection may require high quality video whereas coarse-grained object detection (e.g. object counting) can operate on relatively lower quality and lossy video. We remark that this setup corresponds, for instance, to Urban IoT scenarios and, in particular, city monitoring.

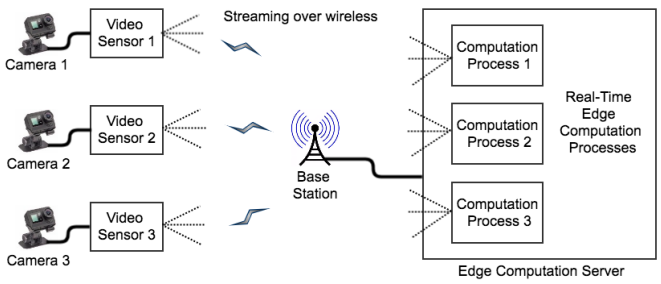

Fig. 1: Real-time edge-based scenario: each computation process uses partial data from various sensors.

Thus, in the considered multi-sensor multi-computation scenario, each sensor data might be reused at different system scales for different computational tasks. Note that if the sensors are mobile, the subset of sensors contributing to a specific computation process may also change over time. The "quality" of data a sensor should provide to a specific application is determined by the QoC requirements of the associated computation algorithm. Here, we define these requirements in terms of a set of metrics measuring characteristics of data delivery, such as loss and delay.

Let say, then, that at time T, N video sensors $\{S_1, S_2, ..., S_N\}$ are streaming video to M computation processes $\{C_1, C_2, ..., C_M\}$ running at the edge server. The i_{th} sensor data stream S_i is used by the j_{th} computation process C_j with quality Ω_{ij}, where $i = 1, 2, ..., N$ and $j = 1, 2, ..., M$. In a naive approach, each sensor S_i would transmit M streams of data over the network with full transmission quality Q_i^{full}. Denote the bandwidth consumption stream S_i with quality Q as $S_i(Q)$ with B_i. The total bandwidth consumption for transmission with full quality is:

$$B_{total} = \sum_{i=0}^{N} \sum_{j=0}^{M} S_i(Q_i^{full}) = M \sum_{i=0}^{N} S_i(Q_i^{full}) \quad (1)$$

As an alternative approach, instead of sending M instances of $S_i(Q_i^{full})$ to M processes, the data streams are replicated in-network – that is, at the network edge – and sent to the M processes in order to reduce the network load to $B_{total} = \sum_{i=0}^{N} S_i(Q_i^{full})$. Thus, the number of streams is reduced to the minimum, but they are still transmitted at maximum quality Q^{full} to support any requirement the computation processes may impose. In the framework we propose, we further reduce the network load by dynamically tuning the quality of the data streams produced by the sensors to that required by the computation processes at any given time.

In the framework proposed herein, packets are replicated inside an in-kernel VM at the data link layer (layer 2) before forwarding the packet to upper layers. This results in a decreased use of memory at the upper layers of the network protocol stack. Our framework further reduces the system memory and CPU usage by incorporating content-based selective packet cloning, which is driven by the requirements of the individual computation processes.

In the light of these functionalities and objectives, in this paper we address the following technical challenges:

i) Determine the quality of the data streams transmitted by each sensor at run-time.
ii) Select which of the data streams received at the network edge need to be forwarded, and to which computation process(es) at any point of time.
iii) Efficiently manage network and computation resources used by each computation process.

In order to solve the aforementioned challenges, we propose an approach based on SDN whose goal is to seamlessly control the resources of the network from the user space at the application layer.

III. Related Work

Several recent contributions focus on real-time IoT data streaming in edge computing architectures. A real-time video surveillance application is proposed in [10] where the transmission policy of the sensor is interference aware to facilitate coexistence with other communications. In [11], the authors adopt a stochastic optimization approach to maximize the efficiency of bandwidth usage by efficiently offloading computation to edge servers. However, the architectures presented in these works are not programmable. In [12], the authors present an edge-assisted SDN-based framework for the dynamic selection of the network used to transport data from real-time applications, and [13] proposes an SDN-based resource management framework to perform content caching and server selection. However, the aforementioned works do not address content reusability and flow orchestration framework at the network edge.

Some recent contributions proposed flow orchestration schemes using NFV at the network edge. Examples include NetFATE [14], a framework based on Open vSwitch (OVS) [15] running at the mobile edge. In [16], the authors proposed the platform NFVnice, which is built on OpenNetVM for the user space-controlled scheduling of NFV chains. However, while providing some level of flow control, these frameworks do not provide sufficiently low level control to extract in real-time for the content transported by the data streams and inform content- and computation-based policies. Different from these contributions, the framework proposed herein adopts a content and computation-aware approach for resource management and dynamic control, and realizes an adaptable and programmable user-controlled platform.

IV. Proposed Architecture

To develop the proposed framework, we use a built-in kernel feature called extended Berkeley Packet Filter (eBPF). In the following, we briefly introduce eBPF and, then, describe in detail the framework.

Packet filtering was first proposed by Mogul et al. [17] to provide user-controlled filtering of a subset of packets in the network stack of kernel which is known as CMU/Stanford packet filter (CSPF). The Berkeley Packet Filter (BPF) [18] has been developed to overcome the stack-based instructions and tree-based filtering model of CSPF. BPF employs a register based instruction set with directed acyclic control

flow graph (CFG) for filtering mechanism which provides a significant performance gain over CSPF as CFG can retain the packet parsing states to avoid redundant comparisons. The BPF filter was implemented in Linux kernel 2.1.75 which supports 32 bit registers for instructions. However, a recent development of BPF called (eBPF) [19] in kernel 3.18 and above, further improves its performance by introducing ten 64 bit registers and employing the JIT (just-in-time) compiler. Also, eBPF programs can be invoked from different layers of the network stack, e.g. socket, qdisc, and drivers. This latter feature enables BPF to process the captured packets before forwarding them to the subsequent layer, and load user defined program inside the BPF in-kernel virtual machine to process the traffic dynamically. Also eBPF maps can be shared between the user and kernel space to enable seamless control. The recent development of BPF within the context of the open-source project IO Visor [20] resulted in the BPF compiler collection (bcc) with LLVM back-end and clang front-end. The main advantage is that the front-end code can be written in high level languages such as python or C, which is translated in the bytecode by the BPF compilers to be loaded inside in-kernel virtual machine.

Based on these recent advancements, the schematics of the proposed framework are shown in Fig. 2, whose components are described in the following.

Quality of Computing Requirements: The QoS requirements of each computation process are continuously evaluated at the communication and computation layers performance over a temporal window. In the considered case, the QoC requirements are used to determine the minimum quality and portions of the data stream needed by the computation process. This parameter is reported by the computation processes to the application program, which updates the corresponding eBPF maps. The edge, in turn, reports it back to the corresponding sensor, which only transmits the required portion of the data stream and suppresses remaining portions to reduce the network bandwidth used by the stream. Based on the determined requirements, the appropriate packets to be transmitted are identified by the sensor using a Deep Packet Inspection (DPI) module [8] at the application layer.

Layer 2 forwarding: Traffic forwarding to specific set of interfaces can be achieved by ingress/incoming and egress/outgoing port mapping through eBPF. In the proposed implementation, the user space application dynamically sets the ingress and egress port pairs in the eBPF map.

DPI eBPF Function: A DPI module is also implemented at the edge filters to select the packets to be forwarded to the computation process(es). This feature enables the implementation of content and computation-specific filtering policies. The DPI eBPF module runs inside the kernel, but can be accessed from the user space in real-time.

Selective Packet Cloning and Transmission: The function of this module is to reduce the usage of computation resources at the edge servers. To this aim, the module employs content-aware selective cloning of packets to be forwarded toward a specific computation process, thus minimizing the socket buffer allocation of reusable packets. The eBPF program running inside the in-kernel VM reads the QoC requirement for each specific computation process through the eBPF maps, and makes decisions in real-time regarding the cloning policy and packet forwarding.

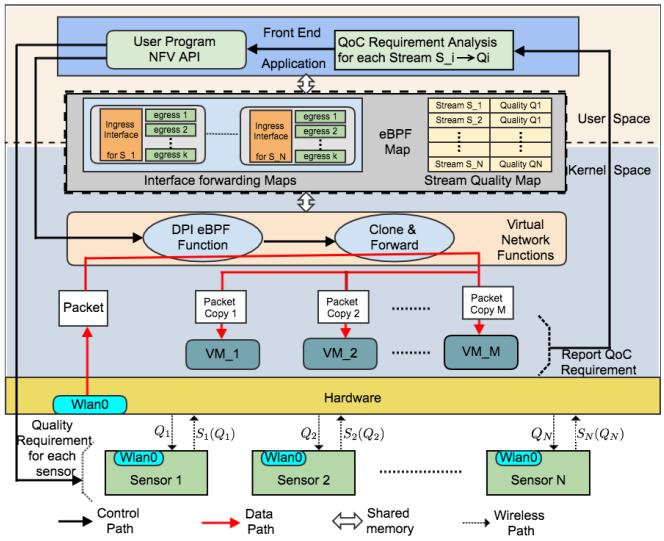

Fig. 2: Schematics of the content- and computation-aware stream control framework proposed in this paper.

V. Content- and Computation-Aware Communication Control Protocol

In this section, we make specific the optimization problem and the control strategy integrated in the framework to the application scenario we consider. We remind that the objective of the multi-sensor and edge system is to minimize total bandwidth usage, while meeting the QoC requirements, that is, maintaining the QoC metric γ_j above a threshold δ_j for each computation process C_j. If each sensor uses a single stream S_i with transmission quality Q_i as described earlier, the optimization problem can be expressed as:

$$\min_{Q_i} \sum_i S_i(Q_i) \quad \text{s.t.} \quad \gamma_j \geq \delta_j, \ \forall i = 1,..,N; \ \forall j = 1,..,M. \tag{2}$$

To solve this problem, we first make the communication between the sensor and edge computation-aware. The sensor determines a transmission quality Q_i that satisfies Ω_{ij} for all $j = 1, 2, ..., M$. Thus, we define effective quality of stream S_i as:

$$Q_i^{eff} = \bigcup_{j=1}^{M} \Omega_{ij} \tag{3}$$

Intuitively, Q_i^{eff} is smaller than or equal to Q_i^{full}. The reduction in terms of network usage can be expressed as the difference between B_{total} and effective access network bandwidth B_{eff}:

$$B_{saved} = \sum_{i=0}^{N} S_i(Q_i^{full}) \ - \ \sum_{i=0}^{N} S_i(Q_i^{eff}) \tag{4}$$

We further reduce Q_i^{eff} by making the communication content-aware. In the context of the specific application scenario considered in this paper, the video data stream is differentially encoded. As a result, the data stream is composed of packets transporting information relative to reference frames (full pictures) or differential frames (encoding differences with respect to reference frames). In [8], we provided a preliminary study on the effect of packet loss on the performance of object detection algorithms. Our work illustrates the much larger impact of packet loss localized in portions of the stream transporting reference frames compared to the loss of packets transporting differential frames. The solution we proposed concentrates interference on differential frames, rather than equally spreading it over the data stream.

In this paper, we use this observation to build a content-aware packet filtering strategy. Specifically, when a reduction in the overall used bandwidth is necessary, the filter first drops packets that are transporting differential frames instead of uniformly dropping packets. Herein, we use a pre-built map, shown in Table 1, between the QoC requirement and packet loss in the two classes of content – that is, reference and differential frames. The advantage in terms of used bandwidth to achieve a given object detection accuracy is apparent. Object detection is performed using the Speeded Up Robust Feature (SURF) detection [21]. We summarize the steps of the filtering strategy and describe how content and computation-aware communications are implemented at the edge server and sensor side in Algorithm 1.

To implement the DPI eBPF function, and the cloning and forwarding function, we use Virtual Network Functions (VNF) which can run inside in-kernel VM. The VNFs can be invoked by the application program using APIs as shown in Listing 1. This allows the dynamic modification of the policies in the shared BPF maps. We further optimize the DPI VNF implementation by setting the MTU size as multiple of transport stream (TS) packet size (188 bytes); so that the module does not scan every byte of the UDP payload, rather, jumps from one TS header of 4 byte size to the next TS header in the packet. This minimizes the number of instructions needed in the BPF. The implementation can integrate any future VNF accessible though a BPF API at the user program.

Listing 1: APIs for accessing eBPF VNFs

```
// Access eBPF functions through API
 bpf = BPF(src_file = "ebpf_vnfs.c", debug=0)
function_DPI = bpf.load_func("vnf_DPI", BPF.
    SCHED_CLS)
 ...
function_clone_forward = bpf.load_func("
    vnf_clone_forward", BPF.SCHED_CLS)
 ...

// Access and update eBPF shared Maps
pol_map = bpf.get_table("policymap")
pol_map[pol_map.Key(idx)] = pol_map.Leaf(arr, 0)
 ...
```

VI. Performance Evaluation

In this section, we first describe the experimental setup, and then provide numerical results assessing the performance of the proposed framework.

Algorithm 1: Content & Computation-aware Communication

Input: $\mathbf{S} = \{S_i : i = 1, 2, ..., N\}$: Sensors
$\mathbf{C} = \{C_j : j = 1, 2, ..., M\}$: Computation processes
$\mathbf{\Omega}[i, j]$: Quality Requirement of S_i for C_j
Output: Sensor to Computation Map : $\mathcal{G} : S \mapsto C$
$\mathbf{Q}[n]$: Sensor Data Transmission Quality
$\mathbf{\Delta}[i, j]$: Edge Suppression Factor

Function EdgeControl $(\mathbf{S}, \mathbf{C}, \mathbf{\Omega})$:
  $\{\mathbb{I}_i\}$: Interface list for S_i
  for $i = 1$ *to* N **do**
    $\{\mathbb{I}_i\} \leftarrow$ **null**
    for $j = 1$ *to* M **do**
      if C_j *includes stream* S_i **then**
        $\{\mathbb{I}_i\} \leftarrow \{\mathbb{I}_i\} + I_j$
      else
        continue
    $Q_i^{eff} = \bigcup_{j=1}^{M} \Omega_{ij}$; $\mathbf{Q}[i] = Q_i^{eff}$
    for $k = 1$ *to* $|\mathbb{I}_i|$ **do**
      A_k : Application on k_{th} VM
      δ_i : Packet loss tolerance of A_k for sensor data S_i
      $\mathbf{\Delta}[i, k] = \delta_i$
      $\mathcal{F}$: Frame type obtained by DPI of S_i for A_k
      if $\mathcal{F} \in$ *Reference Frame* **then**
        Clone and forward packets to interface I_k
      else
        Apply $\mathbf{\Delta}[i, k]$ suppression at interface I_k
        Clone and forward packets to interface I_k

Function SensorControl $(\mathbf{Q}[n])$:
  for $i = 1$ *to* N **do**
    α_i : Loss tolerance of S_i for transmission quality $\mathbf{Q}[i]$
    P_i : List of packets to be suppressed using DPI
    $S_i \leftarrow S_i - P_i$; Transmit S_i

A. Experimental Setup

We implemented the eBPF program on a network edge server with 8 core CPUs. All the VMs run on QEMU hypervisor and we used the docker version 1.11.2 to define network containers. The implementation is based on Linux kernel 4.7. As sensor data stream, we used a real-world video of 640x360 pixels and 372 frames encoded in H.264/AVC format at 30 frames per second. The video is converted to transport stream (TS) with the ffmpeg tool and transmitted from the sensor to the edge over UDP.

B. Results

We evaluate the performance of the framework in terms of programability, layer 2 forwarding performance, bandwidth utilization as a function of the computation requirement,

Object Detection (%)	0.5	1.0	2.0	5.0
Uniform packet loss	0.95	0.84	0.46	0.1
Differential packet loss	0.99	0.96	0.74	0.4
	Packet	loss	(%)	

Table 1: Object detection as a function of packet loss when the uniform and selective packet dropping policies are used.

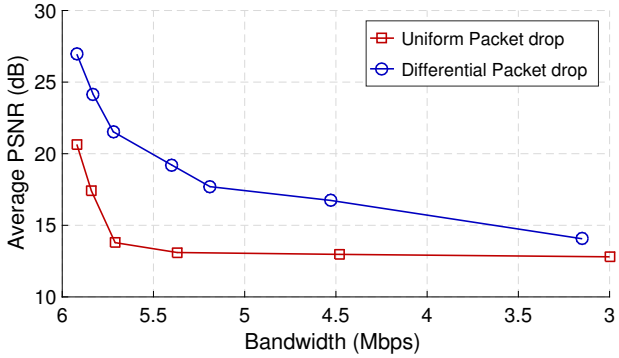

Fig. 3: PSNR as a function of channel capacity.

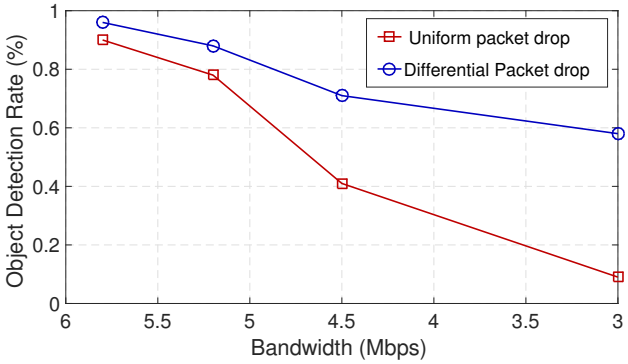

Fig. 4: Object detection rate as a function of channel capacity.

system resource usage and scalability. We tested the real-time redirection of traffic to the corresponding interfaces with respect to dynamic modifications in the mapping between the network source and destination ports, and the programmability demonstrated to be seamless.

Layer 2 forwarding performance: As discussed previously, the selective cloning and forwarding mechanism is implemented at Layer 2 using the DPI module at the edge server. The program is executed inside the in-kernel VM which runs with low level instructions. As a consequence, the flow does not incur significant delay traversing through the upper layers of the network protocol stack. Moreover, the layer 2 forwarding function in eBPF is connected to the *qdisc* of the kernel similarly to the Linux bridge. Hence, the eBPF switch achieves analogous layer 2 forwarding speeds as Linux bridge. We compare them by measuring their average throughput using the "packetgen" utility setting the packet size to 512 bytes. We further compare the two cases using TCP and UDP. Both eBPF and Linux Bridge achieve $\sim$ 1 Gbps throughput over UDP and $\sim$ 2.5 Gbps throughput over TCP (without TCP Segmentation Offload (TSO) and Generic Segmentation offload (GSO)).

Computation Performance: We assess the performance of the proposed framework in terms of quality of the received video with respect to the network bandwidth utilization. First, we measure the average Peak Signal-to-Noise Ratio (PSNR) of the video by suppressing packets at the sensor. Fig. 3 shows that as bandwidth usage is reduced by uniformly dropping an increasing number of packets, the PSNR decreases very sharply compared to the case where packet drop is focused on differential frames. The DPI-based differential packet drop achieves significantly higher PSNR for any given bandwidth compared to the uniform, and non-selective, packet drop. For instance, when the available capacity of the communication link is equal to 5.5 Mbps, the selective filtering strategy achieves a PSNR equal to 18 dB, whereas the PSNR obtained using a uniform drop strategy is equal to 14 dB. A PSNR equal to 15 dB requires a capacity equal to 5.75 Mbps and 3.6 Mbps in the uniform and selective strategy, respectively.

Further, we assess the performance of SURF-based object detection with respect to channel usage (see Fig. 4). The selective strategy provides a significant gain in terms of object detection rate compared to uniform packet drop for any given channel usage. For instance, when the available channel capacity is equal to 4.5 Mbps, the object detection rate is equal to 0.4 and 0.7 in the uniform and selective drop strategy, respectively.

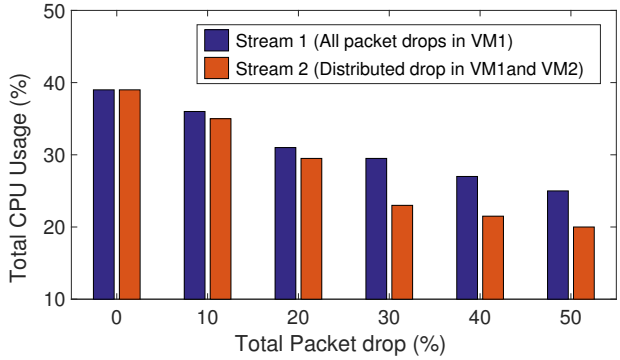

Fig. 5: CPU utilization as a function of packet drop rate.

System Resource Utilization: By selectively suppressing packets and avoiding cloning all the packets at the network edge, the proposed framework also improves system utilization. In the considered case-study, we use two video streams, each of which is processed by two processes running on two separate VMs (VM1 and VM2 respectively). We measure average CPU usage of all VMs with respect to packet drop rate. We distribute packet drop differently in the two streams: all the packets belonging to Stream 1 (first video) are dropped by VM1, whereas VM2 equally drops packets of Stream 1 and Stream 2. Fig. 5 shows that the CPU usage is reduced by $\sim 3\%$ on average for every 10% total packet drop increase when all packet drops are performed on one VM. However, if packet drops are equally distributed between two VMs, then for every 10% total packet drop increase, the CPU usage is reduced by $\sim 5\%$ on average. This indicates that the proposed framework provides an increasing advantage as the differences in QoS requirements of the computation processes increase.

Scalability: In order to test the scalability of the framework, we create a large number of docker containers for parallel computations (see Fig. 6). On the sensor side, we use two sensor devices streaming data. The main goal of this test is to assess whether or not the framework can handle the execution

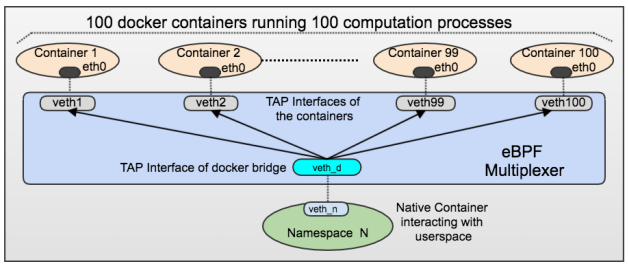

Fig. 6: Multi-streaming test with 100 docker containers.

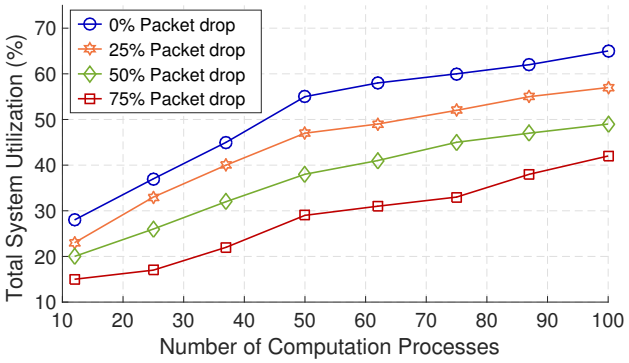

Fig. 7: Variation of total system utilization vs number of containers for different packet drop percentage.

of many parallel instructions in the kernel. Our tests show that due to the light-weight implementation of the DPI and selective clone-forward functions, and the low-level execution of our framework, the framework performs adequately even when the number of computation processes is large. Fig. 7 shows that the total system utilization almost linearly increases with the number of containers. However, the slope decreases as the number of containers grows. This result indicates a tendency of the system resource utilization to become more efficient for large number of containers. Note that the total system utilization increases by $\sim 8\%$ for every 25% increase in packet drop rate, that is, the efficiency increases as the load increases.

VII. Conclusions

The main contribution of this work is the design, implementation and test of an open-source, programmable computation-driven communication control framework for real-time edge computing systems using built-in kernel eBPF. The main features of the proposed system are: *(a)* reduced network utilization to support computation processes; *(b)* highly dynamic network and packet filtering control; and *(c)* efficient system resource utilization at the edge server. Although the framework is demonstrated in a specific application, that is, video processing, the architecture is flexible and can support a broad range of IoT applications. Numerical results obtained by means of real-world testing demonstrate the ability of the proposed system to dynamically adapt data streams supporting remote computation processes.

References

[1] J. Jin, J. Gubbi, T. Luo, and M. Palaniswami, "Network Architecture and QoS Issues in the Internet of Things for a Smart City," in *International Symposium on Communications and Information Technologies (ISCIT), 2012*. IEEE, 2012, pp. 956–961.

[2] A.-S. Ali and O. Simeone, "Energy-Efficient Resource Allocation for Mobile Edge Computing-Based Augmented Reality Applications," *IEEE Wireless Communications Letters*, 2017.

[3] T. Verbelen, P. Simoens, F. De Turck, and B. Dhoedt, "Cloudlets: Bringing the Cloud to the Mobile User," in *Proceedings of the third ACM workshop on Mobile cloud computing and services*. ACM, 2012, pp. 29–36.

[4] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, "Mobile Edge Computing?A Key Technology towards 5G," *ETSI White Paper*, vol. 11, 2015.

[5] B. A. A. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, and T. Turletti, "A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 3, pp. 1617–1634, 2014.

[6] H. Hawilo, A. Shami, M. Mirahmadi, and R. Asal, "Nfv: State of the Art, Challenges, and Implementation in Next Generation Mobile Networks (vEPC)," *IEEE Network*, vol. 28, no. 6, pp. 18–26, 2014.

[7] H. Arslan, *Cognitive Radio, Software Defined Radio, and Adaptive Wireless Systems*. Springer, 2007, vol. 10.

[8] S. Baidya and M. Levorato, "Edge-assisted Computation-Driven Dynamic Network Selection for Real-Time Services in the Urban IoT," in *IEEE International Workshop on Advances in Software Defined and Context Aware Cognitive Radio Networks (IEEE SCAN-2017)*.

[9] A. Begel, S. McCanne, and S. L. Graham, "Bpf+: Exploiting Global Data-Flow Optimization in a Generalized Packet Filter Architecture," in *ACM SIGCOMM Computer Communication Review*, vol. 29, no. 4. ACM, 1999, pp. 123–134.

[10] S. Baidya and M. Levorato, "Content-based cognitive interference control for city monitoring applications in the urban iot," in *Global Communications Conference (GLOBECOM), 2016 IEEE*. IEEE, 2016, pp. 1–6.

[11] Y. Mao, J. Zhang, S. Song, and K. B. Letaief, "Stochastic Joint Radio and Computational Resource Management for Multi-User Mobile-Edge Computing Systems," *arXiv preprint arXiv:1702.00892*, 2017.

[12] S. Baidya and M. Levorato, "Content-based Interference Management for Video Transmission in D2D Communications underlaying LTE," in *International Conference on Computing, Networking and Communications (ICNC), 2017*. IEEE, 2017, pp. 144–149.

[13] Q. Chen, F. R. Yu, T. Huang, R. Xie, J. Liu, and Y. Liu, "Integrated Resource Management in Software Defined Networking, Caching and Computing," *arXiv preprint arXiv:1611.05122*, 2016.

[14] A. Lombardo, A. Manzalini, G. Schembra, G. Faraci, C. Rametta, and V. Riccobene, "An open framework to enable netfate (network functions at the edge)," in *1st IEEE Conference on Network Softwarization (NetSoft), 2015*. IEEE, 2015, pp. 1–6.

[15] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar *et al.*, "The Design and Implementation of Open Vswitch," in *12th USENIX symposium on networked systems design and implementation (NSDI 15)*, 2015, pp. 117–130.

[16] S. G. Kulkarni, W. Zhang, J. Hwang, S. Rajagopalan, K. Ramakrishnan, T. Wood, M. Arumaithurai, and X. Fu, "Nfvnice: Dynamic backpressure and scheduling for nfv service chains," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. ACM, 2017, pp. 71–84.

[17] J. Mogul, R. Rashid, and M. Accetta, *The Packer Filter: An Efficient Mechanism for User-Level Network Code*. ACM, 1987, vol. 21, no. 5.

[18] S. McCanne and V. Jacobson, "The BSD Packet Filter: A New Architecture for User-level Packet Capture." in *USENIX winter*, vol. 46, 1993.

[19] B. Gregg, "ebpf: One small step," *URL http://www.brendangregg.com/blog/2015-05-15/ebpf-one-small-step.html*, May, 2015.

[20] L. Foundation, "Io visor project," *URL http://iovisor.org*, 2015.

[21] "SURF: Speeded up Robust Features, author=Bay, Herbert and Tuytelaars, Tinne and Van Gool, Luc, journal=Computer vision–ECCV 2006, pages=404–417, year=2006, publisher=Springer."