A Reference Model and Prototype Implementation for SDN-based Multi Layer Routing in Fog Environments

Paolo Bellavista
DISI, University of Bologna
Bologna, Italy
paolo.bellavista@unibo.it

Carlo Giannelli
DMI, University of Ferrara
Ferrara, Italy
carlo.giannelli@unife.it

Dmitrij David Padalino Montenero
DISI, University of Bologna
Bologna, Italy
dmitrij.padalino@unibo.it

***Abstract*—If compared with Cloud computing, Fog computing is proving to support challenging scenarios imposing strict delay requirements, e.g., tactile Internet and Industrial Internet of Things (IIoT), and increased flexibility, e.g., dynamic Smart City and users' follow-me provisioning case. In fact, by exploiting computing, storage, and connectivity resources in the proximity of sensors and actuators (for IIoT) and of mobile nodes carried by citizens (for Smart Cities), significant portions of services and functionalities can be migrated outside datacenters. However, such scenarios are characterized by increased heterogeneity of nodes in terms of hardware/software, of time-varying applications possibly offered by multiple service providers at the same time, and frequent joining/leaving of nodes as a typical behavior.**

To overcome these issues, the paper originally proposes Multi-Layer Advanced Networking Environment (Multi-LANE), a Multi Layer Routing (MLR) solution based on Software Defined Networking (SDN) that specifically targets the emerging and promising Fog-based deployment environments. Multi-LANE dynamically selects and exploits (even at the same time) different routing strategies and mechanisms suitable for applications with heterogeneous features and requirements. Based on its centralized point of view, our Multi-LANE SDN controller determines the most suitable path and configures the proper MLR forwarding mechanism, ranging from traditional IP and sequence-based overlays to more articulated ones based on the inspection of payload content types and values. In addition to design/implementation insights and to the availability of the Multi-LANE prototype, this paper also provides the community with a significant contribution in terms of novel models for forwarding mechanisms specialized for Fog computing scenarios.

***Index Terms*—Software Defined Networking, Overlay Networking, Management Middleware, Multi-layering.**

I. Introduction

The adoption of the Internet of Things (IoT) is currently spreading in industrial environments and Smart Cities, with each scenario characterized by specific requirements. In particular, the Industrial Internet of Things (IIoT) specializes the IoT by imposing stricter requirements in terms of not only reliability and security, to ensure that connected physical objects are correctly configured/accessed only by authorized users, but also performance, to always keep monitored parameters up-to-date and to promptly deliver and actuate commands. Smart City environments represent another interesting evolution of IoT. Through IoT, municipalities can support and provide novel and smarter services, ranging from transportation, parking, lighting, traffic, and waste management to safety and law enforcement. Also note that citizens can not only access to Smart City services, but also collaborate to share resources and applications by allowing the dynamic integration of their mobile devices within the Smart City environments. However, there is the need of providing innovative solutions to support the flexible management of network resources, also considering that in these scenarios the network may include several autonomous subnets with differentiated management authorities, hardware capabilities, and service availability.

Fog Computing for IIoT and Smart City. In IIoT and Smart City scenarios, Fog computing is proving to be a viable solution [1], [2]. By deploying multiple devices with computing, storage, and connectivity capabilities closer to sensors and actuators (for IIoT) and to mobile nodes carried by citizens (for Smart Cities), it is possible to migrate some services and functionalities typical of the Cloud outside datacenters. For instance, by managing data on (and eventually issue commands from) Multi-Access Edge Computing (MEC) nodes [3], the latency between an alarm generation and a proper countermeasure is dramatically reduced while the quality of experience perceived by citizens improves, since there is no need of sending packets back and forth the Internet.

Fog scenarios encompass not only MEC nodes (typically deployed and managed by telco operators), but also other Edge fixed/mobile devices deployed closer to end-users (e.g., within subnets where sensors, actuators, and citizens' smartphones/tablets reside) and other nodes closer to the Cloud providing connectivity, storage, and computing services. Such nodes provide developers with a suitable environment to deploy novel applications, ensuring better latency and reliability performance if compared with the traditional global Cloud. In the following, we use the Fog node term to generically indicate any node within the Cloud-to-device continuum.

If compared with Cloud datacenters, Fog environments for IIoT and Smart City scenarios are characterized by:

1) *increased heterogeneity* of nodes in terms of hardware/-software characteristics. Every Fog node can differ in

terms of CPU performance, available RAM and storage, wired/wireless interfaces, operating system and (more generally) software libraries/capabilities;

2) *time-varying applications of multiple service providers* running at the same time with differentiated requirements. Fog viability (also from an economical point of view) benefits from the capability of hosting different applications, by allowing their exploitation for different use cases to fulfill (eventually conflicting) requirements of different users;
3) *frequent joining/leaving of nodes* as a typical behavior, not only due to failures or upgrade of nodes such as in datacenters. Fog nodes are typically managed by different administrators and by users willing to temporarily share their computational and networking resources.

SDN-based MLR to Enhance Fog Environments. The centralized approach of Software Defined Networking (SDN) represents a valuable support to better manage heterogeneous Fog environments. It allows to more easily gather requirements of different applications running in Fog environments and to take management decisions considering the state of the whole network, e.g., to enforce traffic engineering policies and perform rerouting and multicasting [4], [5].

To properly enforce such solutions it is required to introduce the notion of traffic flow, thus making a relationship among different packets related to the same service instance. OpenFlow [6] supports such mechanism by extending traditional IP forwarding with matching rules that take advantage of information available on headers at different OSI layers, such as MAC/IP/TCP or UDP. However, this solution cannot be easily adopted in Fog environments based on overlay networks, since packet forwarding may differ from the traditional IP one (e.g., we adopted DSR in our overlay network middleware [7]). Moreover, traditional IP forwarding rules do not allow, e.g., to easily duplicate packets on intermediary nodes or specify unique receiver identifiers apart from IP addresses. For these reasons, we state that in Fog environments it is more suitable to adopt a per-flow point of view by exploiting a flow id to specifically tag packets on senders.

Exploiting only the flow id to differentiate traffic flows greatly limits packet management granularity. Applications tagging packets with flow ids could require finer granularity, i.e., managing in a different manner different packets of the same traffic flow. For instance, in case of a traffic flow generated by a temperature sensor, packets carrying values that are almost equal to previous ones should have a regular priority while in the case of abrupt temperature change the associated packet should have higher priority.

On the opposite, legacy network services not tagging packets with flow ids cannot achieve the benefits of traffic management and rerouting/multicasting mechanisms (or cannot dispatch packets towards multi-hop paths at all). In fact, packets sent via traditional socket-based applications are forwarded in relation to the IP rules present at routing nodes. Thus, typically there is the need of setting proper IP forwarding rules to correctly dispatch packets at multi-hop distance to targeted Fog nodes. However, the packet forwarding procedure based on flow ids is typically performed at the overlay network layer, without any modification of IP routing tables in the underlying operating system.

The MLR proposal. To overcome these issues we propose the Multi-Layer Advanced Networking Environment (Multi-LANE) middleware, originally adopting a Multi Layer Routing (MLR) approach in conjunction with SDN to fully enhance the capabilities of heterogeneous Fog environments. Our Multi-LANE SDN data plane is the first to adopt MLR by selectively exploiting multiple forwarding mechanisms even at the same time, each one suitable for different Fog applications with heterogeneous requirements. The Multi-LANE SDN control plane exploits dispatching features supported by the overlay network to make possible the interaction among the centralized SDN controller and Fog nodes, allowing to send monitoring information and control messages back and forth. Then, based on application requirements provided by clients and its centralized knowledge about the network status, the Multi-LANE SDN controller is able not only to identify the most suitable path but, most relevant, to select the proper MLR forwarding mechanism, e.g., either IP- or overlay-based, by spreading on sender and intermediary Fog nodes appropriate forwarding rules. Then, applications can start sending traffic flows with the forwarding mechanism suitable for their needs.

The remainder of the paper is organized as follows. Section II outlines our MLR model while Section III presents the architecture and selected implementation insights of our Multi-LANE middleware. Section IV reports quantitative in-the-field performance results, while related work and conclusive remarks end the paper.

II. SDN-based Multi Layer Routing (MLR) model

Different applications running on top of the same Fog environment typically have different capabilities and requirements, pushing for the adoption of differentiated forwarding and management rules. As a general consideration, lower layer forwarding rules based on native IP impose minimum overhead, but require to modify routing tables at the operating system and do not easily allow to satisfy requests of different applications at the same time. Higher layer forwarding rules based on overlay networking, instead, more easily provide sophisticated management capabilities, at the cost of increased computational overhead on Fog nodes.

The MLR approach (combined with SDN) allows applications of the same Fog environment to use native IP, overlay networking information, and payload content to drive packet dispatching, even at the same time and based on application capabilities and requirements. For instance, consider a video streaming and a weather monitoring application running in the same Fog environment and dispatching packets via multi-hop paths. A starting configuration dispatches video and weather data via the overlay network not to impose IP table reconfiguration, at the cost of increased overhead. In case of an alarm, e.g., due to the need of locating a kidnapped person, the MLR approach switches the dispatching of video packets by

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2020.2995903, IEEE Transactions on Network and Service Management

selected cameras to native IP, thus with increased performance but requiring the collaboration of intermediary Fog nodes to manage the involved routing tables. While looking for the kidnapped person, weather packets are dispatched via the overlay network at reduced priority, e.g., by randomly dropping 50% of them, thus providing additional networking resources to the video stream.

Fig. 1 compares overlay network (up) and physical network (down) data planes. The physical network of the Fog environment consists of multiple subnets, each one virtually managed in an autonomous and uncoordinated manner. For instance, in Smart City environments the network backbone provided by municipalities can be enhanced and extended by fixed and mobile Fog nodes provided by citizens and organizations. Moreover, in IIoT scenarios the shop floor network can be integrated with subnets comprising a subset of involved appliances and machines. In such scenarios, native IP forwarding is possible only if the routing table of each node has rules towards every other node. For instance, to send a packet from the left-most to the right-most node of Fig. 1-down, there is the need of deploying an IP forwarding rule towards IPf6 in every intermediary node. Overlay network routing exploits information available within packets themselves to identify a path, such as the sequence of nodes the packet must traverse to reach the destination. For instance, a packet from node a to node f could contain the [c, d, e, f] sequence (note that in this case nodes are identified by their unique ids, not by their IP addresses).

Since there is no single solution perfectly fitting any application-specific feature and performance requirements, we claim that a Fog environment can highly benefit from MLR. In fact, through MLR, we can provide IIoT and Smart City applications with the capability of selecting the forwarding mechanism they deem most appropriate, even by activating different mechanisms at the same time in the same network. Moreover, the adoption of the SDN approach can relevantly support the development of such an MLR solution. To this purpose, we propose to exploit the overlay network to dispatch control plane packets (Fig. 2, up). In this manner, information and commands sent between clients and the SDN controller can take advantage of the routing flexibility of overlay networks, e.g., to identify destinations based on a network-independent unique node id rather than an IP address that could change or could be duplicated in different subnets of the same multi-hop Fog environment. In other words, the SDN controller can dispatch packets related to the control plane notwithstanding how (and if) routing tables on intermediary Fog nodes have been configured. On the contrary, the data plane can exploit MLR to support multiple sub-layers, e.g., by providing forwarding rules at both the physical and the overlay layer (Fig. 2, down), each one characterized by different capabilities.

Below, we detail the forwarding model we have identified by splitting among three primary categories (Table I), i.e., based on native IP forwarding rules, exploiting information available in the header of overlay network packets, and considering the application payload within overlay network packets. Let us note again that, by following our proposed MLR approach, different data plane mechanisms can be adopted at the same time, eventually even by the same application for different packets/flows with a fine-grained granularity.

A. *Native IP*

From a forwarding abstraction point of view, the native IP forwarding mechanism resides at the bottom layer, since it directly exploits features provided by the Fog node operating system.

At this layer, we identify two forwarding mechanisms. The basic one adopts the *IP longest prefix matching* mechanism to forward IP packets based on the destination IP address applied to the local routing table. In addition, it is possible to adopt the more sophisticated *policy based routing* mechanism, by considering additional information such as sender IP address, IP protocol, transport protocol ports, and packet payload size in addition to the destination IP address. In this manner, different packets sent between the same source/destination

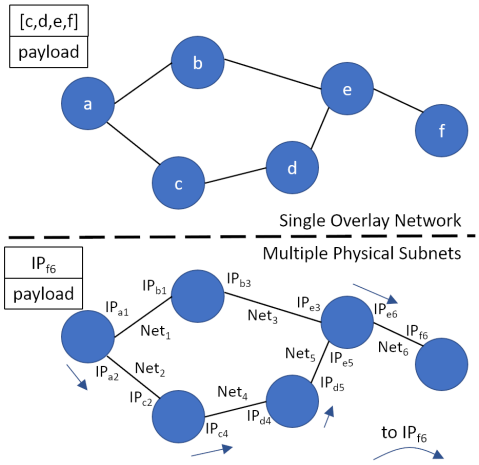


Fig. 1: Fog environment composed of multiple physical subnets (down) interacting to create a single overlay network (up).

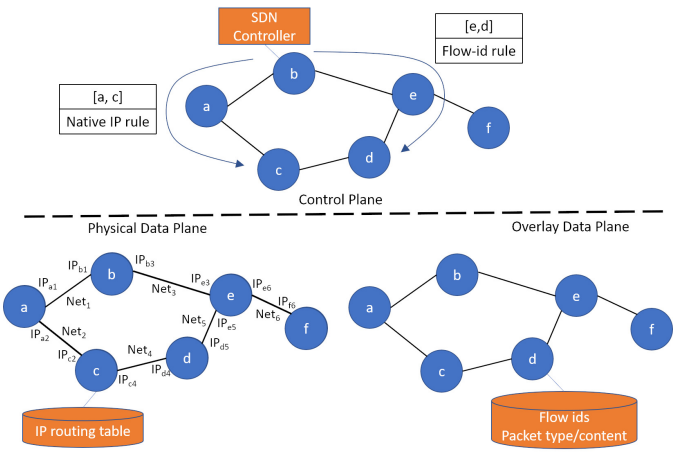


Fig. 2: Control plane (up) is used to send forwarding rules, both for IP routing (down, left) and overlay (down, right) data planes.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2020.2995903, IEEE Transactions on Network and Service Management

TABLE I: The SDN-based MLR Model.

Forwarding mechanism		Main activity of SDN		Pros	Cons
		Control Plane	Data Plane		
Native IP	Destination IP address	IP table modification	IP header read	Legacy-compatible, no sender topology knowledge	Privileged access required, conflicting IP addresses, conflicting rules
	Policy based routing	IP table modification	IP header read	Also multi-path	Privileged access required, conflicting IP addresses, conflicting rules
Overlay Network Packet Header	Path node sequence	Network topology provisioning	Overlay packet header read	table independent, no conflicting rules, no conflicting IP addresses	Topology knowledge, single-path, no rerouting
	Destination identifier	On demand id to path mapping	Overlay packet header read	Also dynamic rerouting, multi-path	Per-packet management only
	Flow id and other header fields	Forwarding rules modification	Overlay packet header history	Also traffic engineering, rerouting/multicasting	Per-flow management only
Application payload	Payload type	Forwarding rules dynamic deployment	Payload check	Also per-type granularity	Required knowledge of payload type
	Payload content	Forwarding rules dynamic deployment	Payload deserialization	Also per-content granularity	Required knowledge of payload content semantic
	Payload content correlation	Forwarding rules dynamic deployment	Deserialization and history	Also inter-packet content granularity	Required knowledge of previous content
	Inter-flow content correlation	Forwarding rules dynamic deployment	Deserialization and history	Also inter-flow content granularity	Required knowledge of other traffic flows

couple can be managed in a differentiated manner, e.g., by forwarding tiny and huge packets towards different paths or by selecting the path in relation to the UDP/TCP destination port.

To support such solutions, the SDN controller must dispatch proper routing rules to Fog nodes between the sender and the receiver (see routing rule towards IPf6 in Fig. 1). Then, once routing rules are set on Fog nodes, the sender can start sending packets in a completely transparent manner, without any network topology knowledge and, most relevant, without modifying network applications on clients. On the sender side, the only requirement is that the sender application (or another companion software module) interacts with the Multi-LANE SDN controller via the overlay network based control plane to properly require the configuration of IP forwarding rules along the path. On the contrary, on the receiver application there is no need of any modification/upgrade.

However, let us note that since such a solution need to modify routing tables, the Multi-LANE control agent (i.e., the software module on Fog nodes interacting with the SDN controller) must have privileged access to the operating system, representing a potential security threat. Moreover, there could be conflicting forwarding rules, since the destination node is identified by the IP address and in such Fog environment there is no guarantee of IP address/subnet uniqueness, since each subnet can be managed and configured in an autonomous manner. However, policy based routing partially solves this issue by allowing to discriminate among different destinations also based on transport protocol and sender/receiver ports. Finally, traditional IP forwarding rules do not natively provide a mechanism to easily manage packets, e.g., to enforce traffic engineering and packet rerouting/multicasting.

B. Overlay network packet header

The overlay network provides forwarding mechanisms by abstracting from and extending features provided by the underlying operating system. Of course, it exploits IP to dispatch packets at one-hop distance, i.e., among Fog nodes belonging to the same IP subnet. However, the actual forwarding among different subnets between the sender and the receiver is managed by the overlay network itself. As a primary and notable consequence, in this case there is no need of modifying operating system routing tables to allow multi-hop packet dispatching, since physical connections (i.e., TCP and UDP sockets) are exploited only to send packets among adjacent nodes. Moreover, as long as nodes of the overlay network have unique node ids, e.g., by exploiting a 128-bit long Universally Unique IDentifier (UUID), there is no risk of conflicting forwarding rules. In fact, such a solution finely works even if different subnets of the same Fog environment exploit the same IP addressing space (of course, such subnets cannot be adjacent).

At this layer, forwarding mechanisms exploit information available in the header of the overlay network packet. In particular, we identify three forwarding mechanisms primarily differing in relation to the header information they use. In case the overlay network supports DSR-like routing, senders can directly specify the *path node sequence*, i.e., the ordered set of Fog node identifiers the packet should traverse in a step-wise manner. For instance, to send a packet from node a to node f in Fig. 1, the sender application can specify either [b, e, f] or [c, d, e, f] (then translated to [IP_{b1}, IP_{e3}, IP_{f6}] or [IP_{c2}, IP_{d4}, IP_{e5}, IP_{f6}] IP address sequences by the overlay network) to send the packet towards either the upper or lower path respectively. In this case the sender must specify the whole path based on the knowledge of the Fog environment topology it can retrieve from the SDN controller. In any case, it can specify only one path for each packet (even if the SDN controller identifies multiple ones) and, in case of path disruption, there is no possibility of rerouting it towards another path.

To lessen the burden on the sender application, the overlay network can also support slightly higher-level communication abstraction features by allowing senders to specify only the *destination identifier*. In this case, it is the overlay network in charge of dynamically discovering available paths towards the destination Fog node and to select the best one, by adopting a proper path selection algorithm ranging from very simple, e.g.,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2020.2995903, IEEE Transactions on Network and Service Management

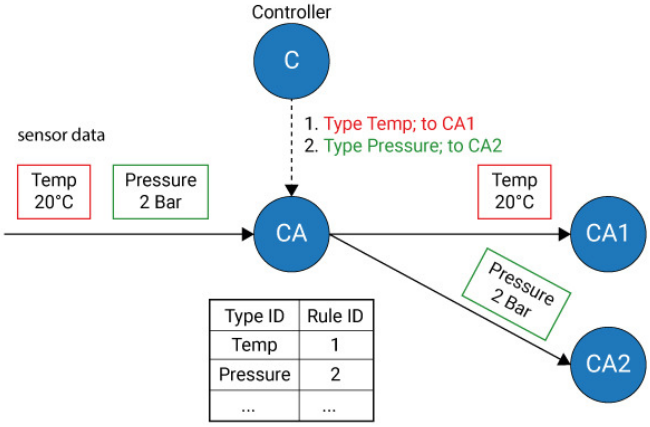
the shortest path, to more articulated, e.g., the most reliable path in terms of node mobility [8], ones. To this purpose, the control agent in the sender node intercepts traversing packets and, in a completely transparent manner, interacts with the SDN controller to map the destination id to a path in terms of set of traversed nodes. Note that such a solution is adopted in a per-packet manner, i.e., different packets sent to the same destination can go towards different paths. Moreover, if the currently adopted path is not available anymore, not only the sender but also Fog nodes can dynamically discover an alternative path and reroute the packet towards the destination.

Senders can also tag packets related to the same traffic flow with a unique *flow id*. In this manner, Fog nodes can easily correlate different packets and manage them in a proper manner. In particular, by tagging packets with the flow id it is possible to adopt traffic engineering mechanisms by providing higher priority to a traffic flow at the expense of others or to support multicasting of the same packet towards different destinations in a step-wise fashion. In this case, the SDN controller deploys proper forwarding rules on Fog nodes and then each node autonomously manages traversing traffic flows by applying such rules. Finally, note that together with the flow id, such mechanisms i) can need to maintain some history of recent packets (e.g., inter-packet delay average and standard deviation) and ii) can also use other packet header fields provided by the overlay network and available on Fog nodes, e.g., the hop count, the set of previously traversed nodes, or the destination port.

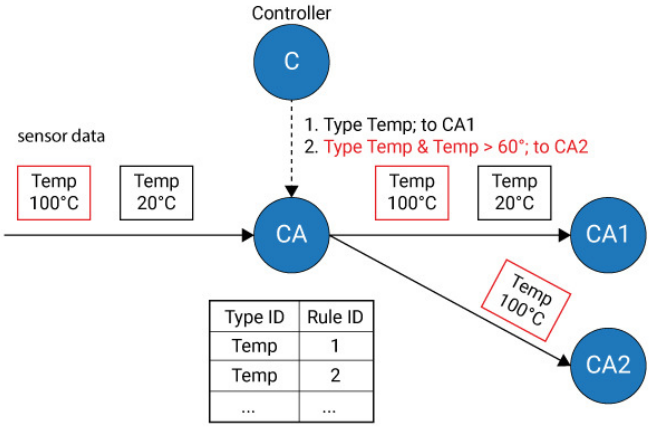
C. Application payload

Further abstracting, it is possible to specify forwarding rules based on the content provided by applications, in an Information Centric Networking (ICN) fashion [9]. Also note that such information can be (and usually are) exploited together with overlay packet header ones. For instance, it is possible to identify the set of paths based on the destination id and then to select a path based on the payload content. To this purpose, Fog nodes must have some knowledge about the semantic of the content sent among nodes, with the purpose of applying proper forwarding rules. It is worth noting that, in general, it is not possible to statically identify the set of content applications can generate. For this reason, the SDN controller should be flexible enough to create forwarding rules based on content syntax and semantic not yet available at node deployment time and, most relevant, control agents should be able to dynamically receive, deploy, and enforce such rules on Fog nodes.

At this layer, forwarding mechanisms mainly differ in relation to the required knowledge and inspection of packet payload. In the most straightforward case, forwarding rules only consider the *payload type*. In this case, while the SDN controller and intermediary nodes must know the kind of content is sent by applications, the additional complexity is only due to check the type of payload. By considering also the *payload content*, there is the notable additional overhead of payload deserialization. In fact, it imposes the relevant burden



(a) Payload type.



(b) Payload content.

Fig. 3: Application payload routing examples.

of recomposing the message payload at each Fog node. Such a task can be very CPU intensive and memory consuming, also considering that Fog nodes can have limited hardware capabilities, with the notable shortcoming of potentially delaying packet dispatching. However, this mechanism allows to specify much finer-grained management rules, e.g., by rerouting important content towards better paths or by selectively assigning traffic engineering priorities not only based on the flow id but also on the dynamically computed relevance of content. Fig. 3 presents examples of payload type and content type overlay network routing based on rules provided by the SDN controller on node C. In the former case (Fig. 3a), the CA node dispatches packets to CA1 and CA2 in case the carried content is of type Temp and Pressure, respectively. In the latter case (Fig. 3b), the CA node dispatches Temp packets towards CA1 if the temperature value is lower than or equal to 60°C, towards CA2 otherwise.

Further increasing the level of information awareness and exploitation, it is possible to correlate current and previous content either of the same or even of different traffic flows. In both cases the complexity of control agents relevantly increases. In particular, in case of *payload content correlation* there is the need of storing (part of) information carried by previous packets, also adopting proper mechanisms to decide

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2020.2995903, IEEE Transactions on Network and Service Management

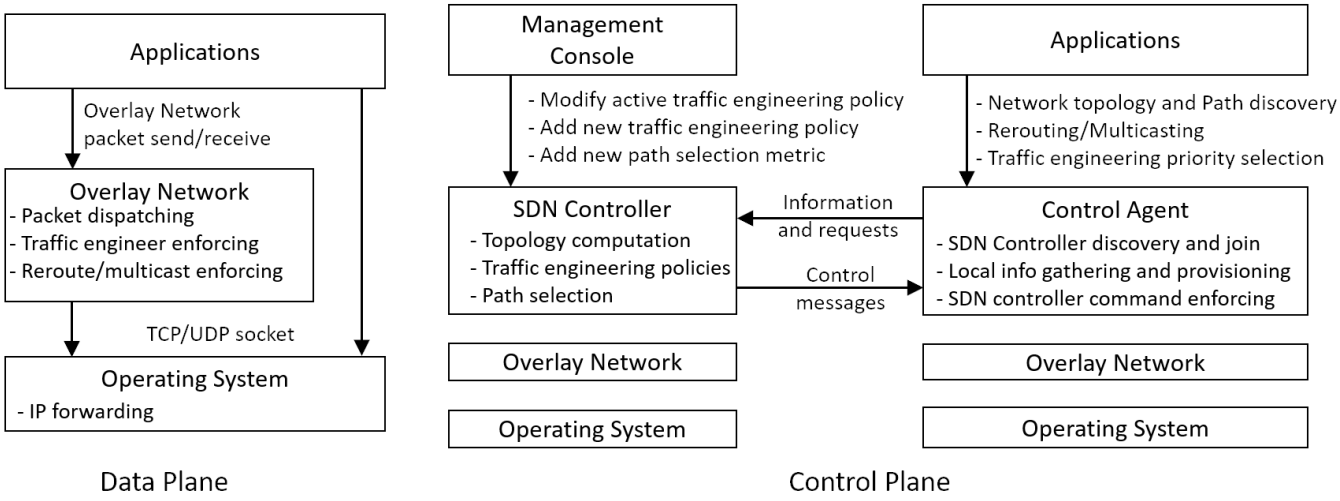


Fig. 4: High-level architecture of Multi-LANE data plane (left) and control plane (right).

when previous content should be removed/pruned from memory. However, it is possible to specify meaningful forwarding rules such as by assigning higher priority to packets of a traffic flow identified by a flow id only if the carried value differs from the average of the previous ten values more than a percentage. Finally, in case of *inter-flow content correlation* it is also possible to specify rules that have an impact on other traffic flows, e.g., by temporarily lowering the priority of every flow but the current one in case of very important and urgent information with a strict temporal deadline.

III. Implementing the MLR Approach in Multi-LANE

A. Multi-LANE architecture and selected implementation insights

Fig. 4 presents the high-level architecture of our Multi-LANE solution. The data plane (Fig. 4, left) is based on two primary components. At the bottom layer resides the operating system performing native IP forwarding and providing traditional UDP/TCP sockets to applications. Then, the overlay network (based on our previous solution [7]) exploits lower level features to dispatch packets and enforce traffic engineering and reroute/multicast forwarding rules. Moreover, it provides to applications communication API to send/receive unicast/multicast/broadcast packets at multi-hop distance within the overlay network. Applications can adopt the MLR approach by exploiting operating system, overlay network, or both mechanisms, with a fine-grained per-packet granularity. The control plane (Fig. 4, right) exploits the overlay network to dispatch information and control packets among Fog nodes and the SDN controller. Each Fog node runs a control agent:

- discovering available SDN controllers and joining to the most suitable one, e.g., based on hop distance or RTT. During the join phase, the control agent also provides local forwarding capabilities, e.g., if local security policies allow to modify operating system IP rules;
- periodically gathering and sending to the SDN controller local information about, e.g., the number of interfaces, local IP addresses, and the available bandwidth for each interface (additional details in [5]);
- waiting for and applying SDN controller commands, e.g., asking to modify IP forwarding rules or overlay network traffic engineering policies;
- providing API to local applications, e.g., to interact with the SDN controller to get the best path towards a destination and to enable multicast in a step-wise fashion, or, eventually, also to retrieve the network topology and locally identify a path towards the destination.

The SDN controller:

- receives information from Fog control agents and exploit such information to generate a network graph representing the network topology;
- maintains a database of best path selection algorithms and traffic engineering policies;
- receives requests from sender applications (via their local Fog control agents) to get, e.g., the set of available paths towards a destination by applying a given path metric;
- provides an API to, e.g., switch among traffic engineering policies or deploy new path selection and traffic engineering policies. The SDN controller API can be locally exploited to create a graphical management console or to remotely interact via remote procedure calls.

We have developed a Java prototype of our Multi-LANE middleware not only to demonstrate the feasibility and the efficiency of the presented model/features, but also to provide the community with a working solution to foster the research in this field. The source code can be found at https://github.com/DSG-UniFE/ramp (native IP supported on Linux, the rest on Linux/Windows/macOS).

Fig. 5 presents a detailed overview of the Fog node architecture. In the control plane, the Control Agent component provides communication API to local applications and interacts with the SDN controller by exploiting the overlay network (supported by our previous RAMP solution) to send locally gathered information and receive forwarding rules. Based on received commands, the Control Agent component

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2020.2995903, IEEE Transactions on Network and Service Management

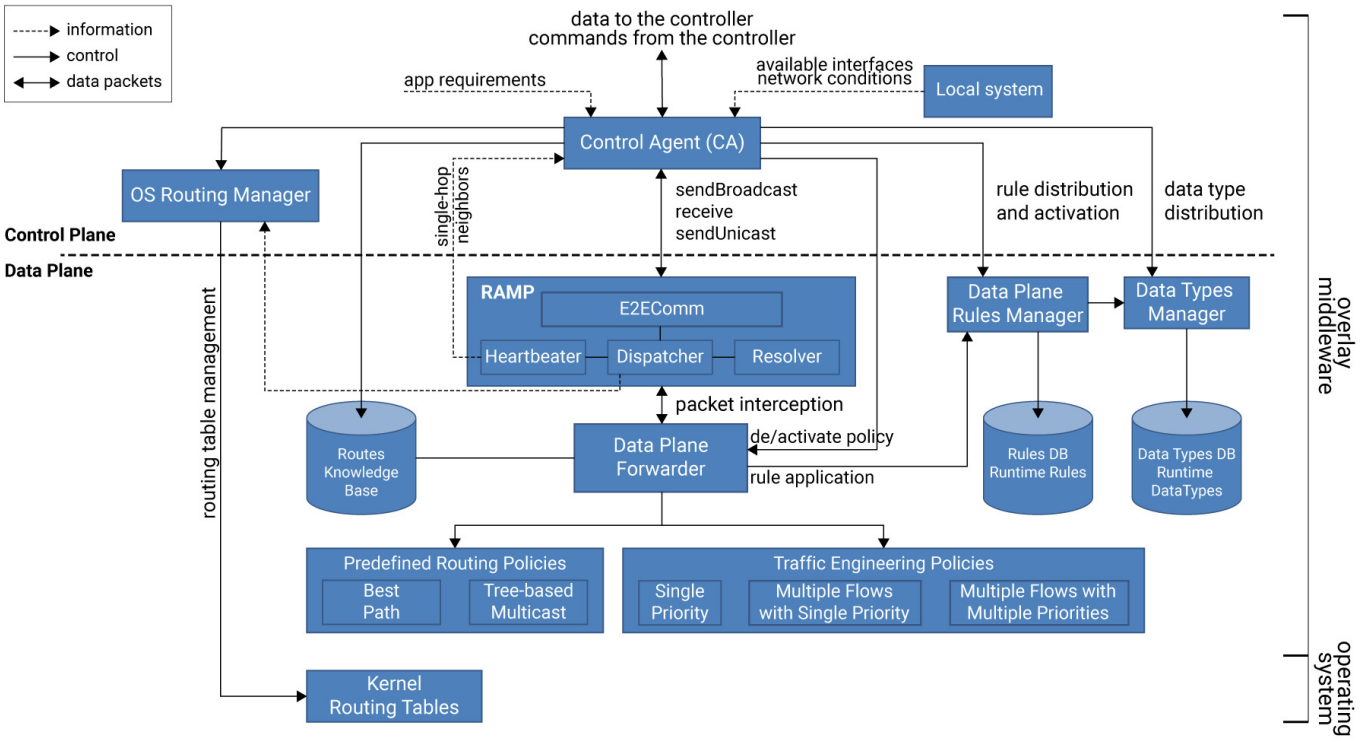


Fig. 5: Multi-LANE node implementation insights.

can exploit the OS Routing Manager to modify the OS-level routing tables or can interact with Data Plane Rules Manager and Data Types Manager to deploy new forwarding rules and new content types, respectively. In addition, the Control Agent can de/activate predefined routing rules and traffic engineering policies (presented in [5]). In the data plane, the Data Plane Forwarding component intercepts overlay network packets and forwards them accordingly to the forwarding rules set by the Control Agent component. Also note that Kernel Routing Tables logically reside in the data plane, even if outside the Multi-LANE solution implementation.

B. Control-plane inter-node interactions

By properly exploiting Multi-LANE components, it is possible to support MLR communication features presented in Section II. Let us note that supported communication layers also greatly differ in the control plane phase. In particular, we identify three primary characterizing aspects of the control plane:

- which nodes are involved in the control plane phase, i.e., with or without the involvement of intermediary Fog nodes;
- where primary control plane actions happen, i.e., on the sender, on the SDN controller, or on Fog nodes;
- the complexity of the sender, either dummily accessing network features in a transparent manner or smartly interacting with the local control agent in a direct way.

Fig. 6a shows the sequence diagram of the native IP control plane. First of all, the sender application provides to the SDN controller (by exploiting the API of the local control agent) the destination node (identified by the node id) it is going to send packets to. Optionally, the sender can also specify the best path metric that should be adopted and other information in case of policy based routing. Then, the SDN controller applies the best path metric to identify a path between the sender and the receiver, generates proper IP forwarding rules, and sends them to Fog nodes along the path. Control agents of Fog nodes along the path receive forwarding rules and apply them locally. In case every Fog node correctly applies the rules, the SDN controller notifies to the sender the destination IP address it can use to send packets via the native IP data plane (otherwise, an abort message).

We currently support three different path computation metrics (but additional metrics can be easily added):

- breadth first, by looking for a path from a source to a destination with the breadth first search;
- minimum network load, by considering the current network load of each link;
- minimum intersections, considering the amount of flows already allocated to each link of the topology.

It is worth noting that the minimum intersections metric tends to provide different paths from the source to the destination and vice versa, with the notable benefit of fully exploiting available link diversity also in the case of multi-path TCP connections managed at the operating system level. For instance, Fig. 7 presents how our Multi-LANE solution allows IP packets of the same multi-hop TCP connection to flow towards different paths on opposite directions, thus taking full advantage of the network topology. In particular, packets related to the multi-hop TCP connection between node CA1 and node CA5 flow towards CA2 in a direction and towards

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2020.2995903, IEEE Transactions on Network and Service Management

CA3 in the other direction.

In the native IP solution there is the need of interacting with every Fog node in the path, since routing tables at the operating system must be properly configured. However, only the SDN controller must have knowledge about the network topology and how to compute the best path. Finally, the sender node can be very simple, since it only interacts with the local control agent to specify the destination IP and then it waits for an acknowledgment together with the destination IP.

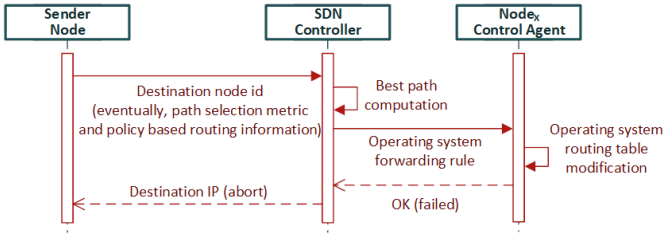
Fig. 6b shows the sequence diagram of the overlay network with path node sequence control plane. First of all, the sender application gathers the up-to-date network graph from the SDN controller (via the local control agent). Based on this information, the control agent of the sender node can autonomously identify the best path towards the destination and then it can start sending packets taking advantage of the overlay network data plane.

Differing from the previous solution, intermediate Fog nodes are not involved in the control plane phase since there is no need of modifying their forwarding rules. Moreover, the SDN controller only maintains the network topology based on its centralized point of view while the sender node actually exploits the network graph to identify the best path. Let us stress that while this solution imposes on sender nodes network overhead (to gather the network graph) and computational overhead (to identify the path), it ensures high flexibility, since the sender can adopt the path comparison metric it deems most appropriate.

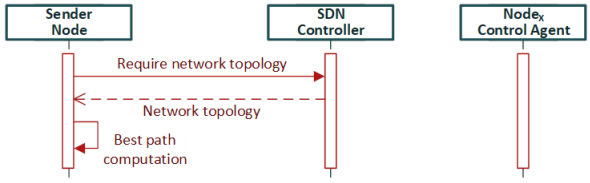
Fig. 6c shows the sequence diagram of the overlay network based on destination node identifier control plane. First of all, the sender application provides to the SDN controller the unique node id of the receiver, eventually also specifying the best path metric the SDN controller should adopt. Then, the SDN controller replies with a path towards the destination, exploited by the sender to dispatch packets via the overlay network.

Even in this solution there is no need of involving intermediary Fog nodes. However, in this case it is the SDN controller in charge of identifying the best path, with the notable benefit of making easier the development of the client and of limiting the communication overhead (no need of transferring the network graph).

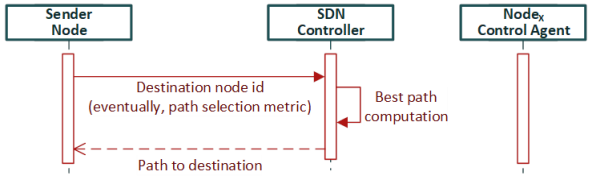
Fig. 6d shows the sequence diagram of the overlay network based on flow id control plane. First of all, the sender application provides to the SDN controller the id of the destination node (multiple ids in case of multicasting), eventually also specifying the required priority. Then, the SDN controller identifies the best path(s), generates a flow id, and interacts with Fog nodes to deploy/configure rerouting, multicasting, and/or traffic engineering rules. If every Fog node correctly installs/configures required rules, the SDN controller provides the flow id to the sender application. Then, the sender application can start sending packets via the overlay network simply by tagging packets with the given flow id, without any specific node sequence. In fact, in this case Fog nodes forward packets actively contributing in a step-wise manner based on previously deployed local forwarding rules.



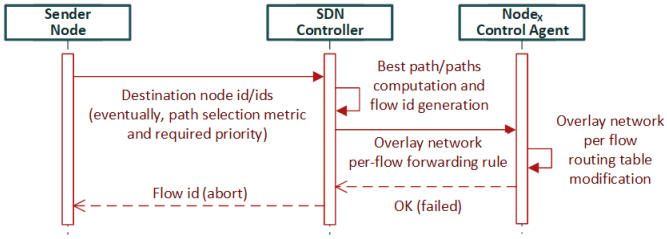
(a) Native IP.



(b) Overlay network, path node sequence.



(c) Overlay network, destination identifier.



(d) Overlay network, flow id.

Fig. 6: Sequence diagrams of primary control plane actions.

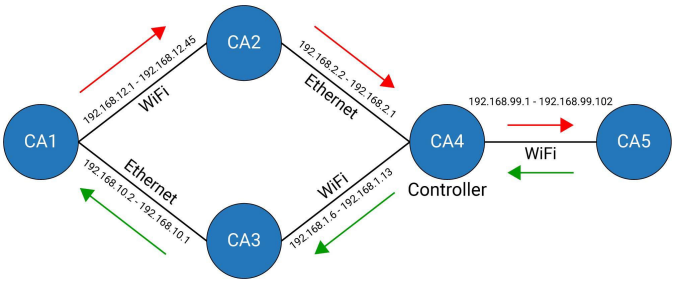


Fig. 7: Multi-path bidirectional TCP connection.

In this solution the sender application is very simple, since it only gathers the flow id and tags packets accordingly. On the contrary, there is a deep involvement of Fog nodes, since each node in the path must read the packet to verify if there is the need of applying rerouting, multicasting, or traffic engineering rules.

Finally, in case of overlay networking based on payload type and payload content there are additional costs. In the control plane, the SDN controller must also provide more articulated rules to make control agents aware of the structure

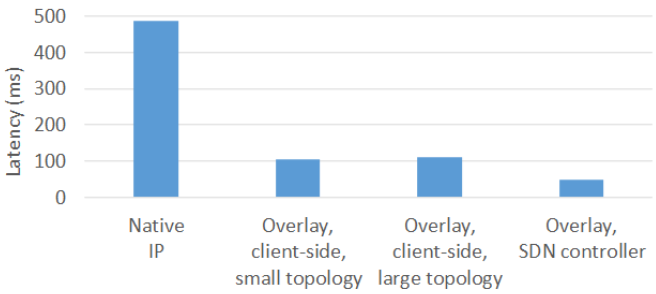
This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2020.2995903, IEEE Transactions on Network and Service Management

of payload content, e.g., its serializable Java class or Protobuf structure. In the data plane, Fog nodes must inspect each packet, eventually also saving its content to enable payload and inter-flow correlation solutions. Apart from these notable differences, control plane sequence diagrams do not differ from the flow id one and thus we do not provide them.

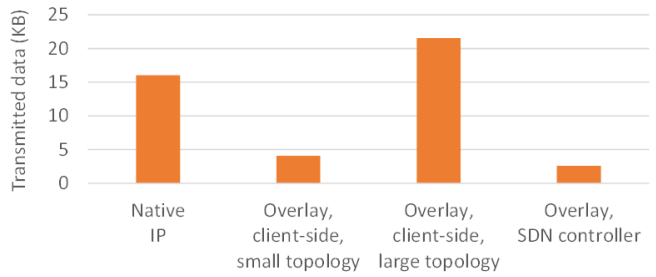
IV. Multi-LANE In-the-Field Experimental Evaluation

Based on the implemented Multi-LANE prototype, we have tested the primary features it supports at the control and data planes. We have organized a real testbed consisting of Raspberry Pi3 Model B+ connected one another via either Ethernet or IEEE 802.11 in different configurations, e.g., the one depicted in Fig. 7. For each single-hop link there is a different IP subnet, in order to stress the management complexity and efficiency of the target environment. Moreover, in the tested environments, devices and services are discovered via a basic controlled-flood approach, thus dispatching discovery messages along every available path (but avoiding loops by dropping duplicated messages). In case of more articulated topologies, scalability can be increased by adopting a tree-based control packet dispatching schema, e.g., the one presented in [10].

In particular, control plane performance analysis reports about the time required (see Fig. 8a) and bytes transmitted (see Fig. 8b) to set up a multi-hop path from node CA1 to node CA5 in Fig. 7. In case of native IP, the time required to set up the two-way path is 488 ms. It is worth noting that about 286 ms are required by OS Routing Manager to modify routing tables at the operating system level via the iproute2 command, while the rest is due to packet dispatching and path computation. The total amount of transferred data is 16.0 KB, accounting for the request packet sent by node CA1, packets with routing rules sent by the SDN controller (one packet for nodes CA1 and CA5, two packets for Node CA2, CA3, and CA4) with string commands control agents should apply (and related acks), and the final response to the requesting node. In case of overlay network with path computation on the client node, performance results also depend on the size of the topology, since the topology graph must be transferred from the SDN controller to the client. For instance, by considering the topology in Fig. 7, the required time is 106 ms and the transmitted data 4.1 KB, 898 bytes for the request and 3289 bytes for the response with the topology graph (see Fig. 8a, "Overlay, client-side, small topology"). The latency is much lower than the native IP case since there is no need of reconfiguring operating system routing tables. However, note that the performance lowers in case of more articulated topologies. Just to provide an example, in case of a larger topology with 12 nodes and 36 links, as expected, the associated performance indicators get worse (see Fig. 8a, "Overlay, client-side, large topology"): the message with the topology increases from 3289 bytes to 21184 bytes (total size of 21.6 KB) while the time required slightly increases from 106 ms to 112 ms (the increased payload does not negatively



(a) Path setup latency.



(b) Path setup transmitted bytes.

Fig. 8: Control plane path setup performance.

impact on achieved performance since there is no congestion). In case of overlay network with path computation on the SDN controller and the path provided to the sender, the latency further lowers to 48 ms and the amount of transferred data to 2.6 KB, taking advantage of path computation on the SDN controller with no need of transferring the topology graph.

Data plane performance results compare the different routing mechanisms when sending packets from CA1 to CA5 (see Fig. 7) for 15 s at varying frequency (from 1 msg/s to 200 msg/s) with a fixed payload size of 5 KB and using only one path. Let us note that by adopting only one path it is possible to stress the load on intermediary nodes, thus allowing to better differentiate and compare the overhead imposed by the different forwarding mechanisms. Moreover, the sender CA1 in Fig. 7 dispatches packets in a sequential manner: in the native IP case the sender periodically sends 5 KB messages exploiting the same socket while in the overlay network case each message is sent by exploiting the *sendUnicast(...)* API.

Fig. 9a presents the average end-to-end packet latency, that is the average time required for the receiver on node CA5 to receive packets sent from the sender on node CA1. It is worth noting that forwarding mechanisms based on the overlay network achieve almost the same performance of the native IP one for traffic load till 50 msg/s, demonstrating the efficiency of the Multi-LANE solution. At 100 msg/s the latency increases in the most challenging cases of content and type payload inspection, while flow id and DSR-like node sequence achieve only slightly higher values if compared with the IP native baseline solution. Then, at 150 and 200 msg/s emerges a considerable difference. In fact, the great amount of transmitted packets imposes a notable computational load on the intermediary node, causing the slowdown of more

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2020.2995903, IEEE Transactions on Network and Service Management

computational intensive forwarding solutions, i.e., payload type and payload content. In addition, even the flow id solution presents a notable rise in the average packet dispatching time, since it requires that each node performs a lookup procedure to identify the next hop in relation to the flow id. Such observation is also supported by the analysis of the CPU load on node CA4 running the Multi-LANE solution. In fact, Fig. 9b clearly outlines the additional computational cost of payload type and payload content networking. Instead, the node sequence solution imposes very limited overhead, thus allowing to achieve performance results very close to the native IP one. In relation to Fig. 9b, let us note that the average CPU reaches its maximum at about 80% since it represents the average value of the whole test, also considering starting and ending phases with limited CPU consumption (and thus lowering the overall average CPU consumption).

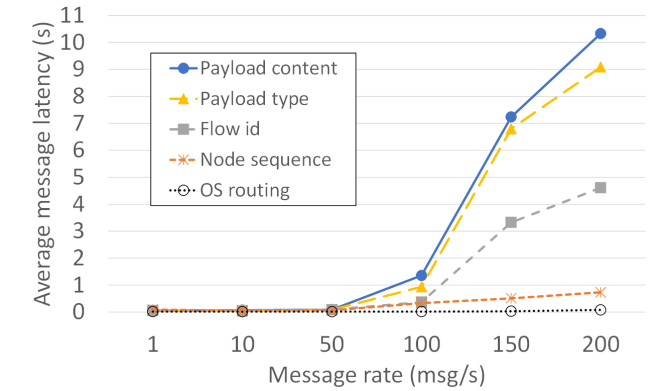
To further show how the computational load differently impacts on performance of proposed forwarding solutions, we introduce a packet ordering metric, computed on the receiver node CA5 as

$$\sum |(arrivalSeq\#) - (departureSeq\#)|/\#sentPkts \quad (1)$$

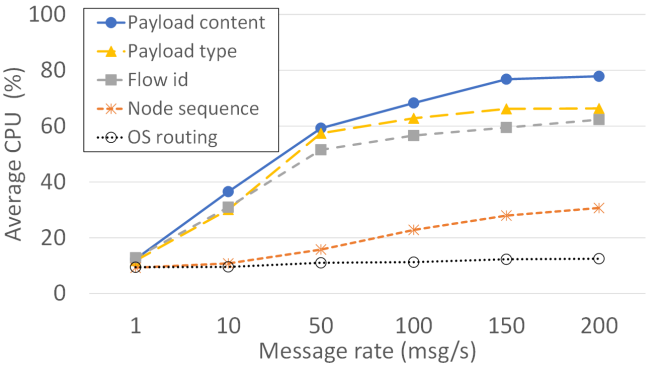
i.e., considering the sum of mismatch distances among expected and actual packet arrival sequence numbers, normalized with the amount of sent packets. Fig.9c shows how packet ordering is almost perfect till 50 msg/s. Then, the payload content solution starts to significantly worsen at 100 msg/s, while flow id and node sequence worsen at 150 msg/s. Such performance results are also due to the non-blocking asynchronous nature of our overlay network middleware solution based on a pool of thread managing packet dispatching, while the native IP solution delivers packets in order since the only one sender thread sequentially delivers packets to the receiver.

To better analyse and compare the efficiency of our proposed middleware, Fig. 10 presents the Cumulative Distribution Function (CDF) of the latency, by focusing on 100 msg/s (see Fig. 10a) and 200 msg/s (see Fig. 10b) while removing 5% of outlier values. In case of 100 msg/s, only the payload content solution presents high latency, with values even greater than 5 s. Instead, in case of 200 msg/s both payload type and payload content present latency values greater than 20 s and Flow id up to 12 s, while Node sequence is much more efficient (most of its values are close to the OS routing solution). In addition, Table II concisely depicts two different metrics, one for efficiency in terms of CPU load and one for efficacy in terms of packet timely arrival, defined as the percentage of packets arrived within 2 s for a medium packet rate of 100 msg/s (TCP as OS transport protocol). Of course, payload content and type present worse performance for both CPU load and packet delivery; however, let us stress that this is the drawback of adopting a solution that is much more powerful in terms of routing rule expressiveness.

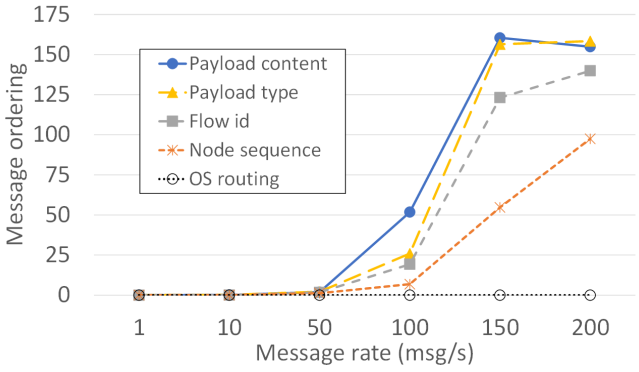
To better analyse how payload size impacts on most computing intensive solutions, Fig. 11 compares the time required to apply type and content payload forwarding rules on the intermediary node CA2. In this case, we fixed the message rate at either 50 msg/s (thus before CPU load starts to considerably increase) or 200 msg/s (thus in the challenging case with saturated computing resources) and we varied the payload size from 1 to 10 KB. Fig. 11a demonstrates that the payload content solution imposes an additional overhead if compared with the payload type one. However, as long as the computational load is limited, the time required to inspect the content of messages is in the order of few ms. Moreover, the payload size does not considerably influence the rule application time, rising from 1.8 ms for a 1 KB payload to 3.1 ms for a 10 KB payload in the more challenging content payload solution. Fig. 11b shows how at higher message rate the payload content solution imposes additional overhead



(a) Data plane packet latency.



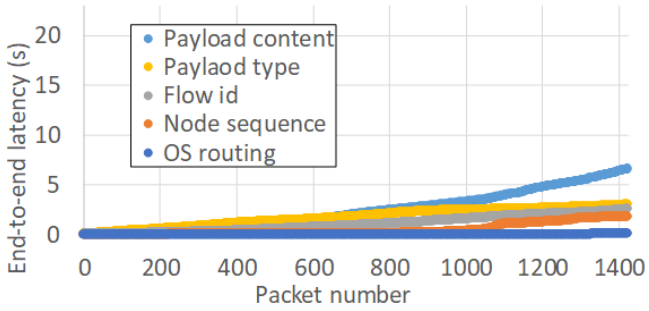
(b) Data plane CPU load.



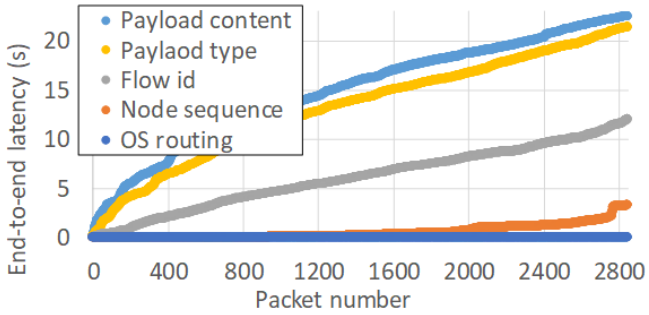
(c) Data plane packet ordering.

Fig. 9: Data Plane performance (fixed payload of 5 KB).

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2020.2995903, IEEE Transactions on Network and Service Management

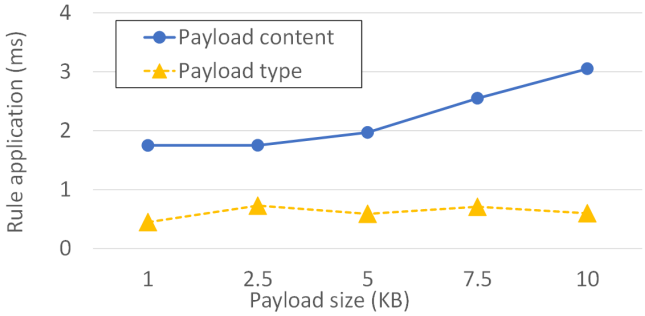


(a) 100 msg/s.

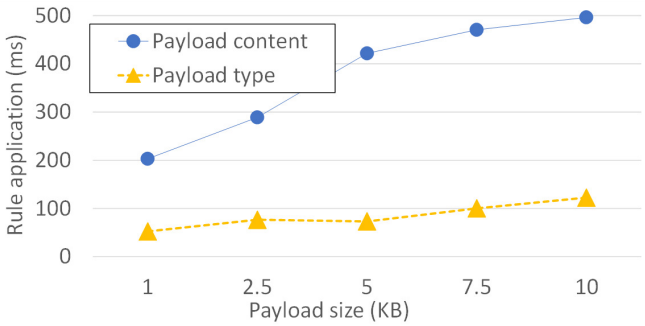


(b) 200 msg/s.

Fig. 10: Latency CDF.



(a) Message rate fixed at 50 msg/s.



(b) Message rate fixed at 200 msg/s.

Fig. 11: Type and content payload rule application.

on the intermediary node, much greater than the payload type one. Moreover, in this case the payload size clearly affects achieved performance. In fact, the greater the content size, the more time is required to apply the payload content solution, rising from about 200 ms with a 1 KB payload to about 500 ms for a 10 KB payload. Let us note that it is possible to improve the achieved performance by adopting more efficient de/serialization mechanisms. In particular, in the tested environment the payload consists of several arrays each one with tens of floats serialized by adopting the default Java serialization mechanism, designed to maximize flexibility at the cost of increased computational and size overhead. For instance, by adopting protobuf it is possible to greatly reduce the CPU load of de/serialization procedures (additional details in our previous work [11]). However, regardless of the adopted de/serialization mechanism OS routing, Node sequence, and Flow id will be always more efficient, since they do not need to deserialize the packet payload on intermediary nodes.

Finally, we have tested the capability of our solution to switch among forwarding strategies in a dynamic manner, also considering time-varying network conditions and application requirements. To this purpose, we developed a vibration monitoring application: a sensor on node CA5 (see Fig. 7) gathers vibration data and send them to CA1. Each

TABLE II: Comparison of performance at 100 msg/s.

	OS routing	Node sequence	Flow id	Payload type	Payload content
CPU load (%)	11.28	22.82	56.65	62.81	68.29
Packets within 2 s (%)	100.00	98.27	79.07	52.37	47.47

Listing 1: Vibration data packet.

```
public class VibrationData extends
    AbstractDataType implements Serializable {
  private int maxVibrationIndex;
  private byte[] rawData; // 200 bytes
  private long sentTimestamp;
  // follow getters and setters
}
```

packet contains the maximum vibration index till the previous packet together with the set of sampled data (see Listing 1, `maxVibrationIndex` and `rawData` respectively). To reach the destination, the packet can flow towards either CA3, the default path used by regular packets, or CA2, a low latency path (30 ms lesser latency) that should be exploited only for urgent packets. The goal is to wisely dispatch packets, by tuning the adopted forwarding mechanism in relation to the importance of carried information and the currently achieved performance. In particular, if `maxVibrationIndex` is in the [0,10[range everything is fine, if it is in the [10,20[range there is the need to better investigate the situation, otherwise it means that there is an issue. Moreover, it is mandatory that packets reach the destination in at most 250 ms.

To achieve the aforementioned objectives, we have developed an MLR-based application where the sender exploits the node sequence mechanism by specifying the default path [CA4, CA3, CA1]. Moreover, the intermediate node CA4 collaborates by adopting the payload content mechanism to dispatch packets in relation to carried values, eventually triggering the adoption of the OS routing mechanism whenever required. Delving into finer details, Listing 2 sketches our Java

implementation of the payload content rule deployed on CA4, allowing to dynamically manage the delivery of vibration data. To this purpose, the `applyRule` method is invoked by the Data Plane Forwarder for every traversing packet exploiting the RAMP middleware and thus the overlay network. Whenever a `vibrationDataId` is identified, the rule deserialize the payload to appropriately manage the packet. In case the `maxVibrationIndex` is low it discards 90% of the packets and sends it towards the default path, thus still delivering information to the receiver while limiting the network load on the network. In case, the `maxVibrationIndex` is medium it delivers every packet, still towards the regular path. In case the `maxVibrationIndex` is high, it invokes the `findNewFastestPath` method of the local CA to require a faster path to the SDN controller.

Let us note that the `findNewFastestPath` method triggers two control plane mechanisms by interacting with the SDN controller. First of all, it activates the `Get Path Protocol` to retrieve a faster path towards the destination; in this case, the protocol is based on a one-shot request/response interaction to send application requirements and to receive the new path node sequence. In addition, while waiting for the `Get Path Protocol` response it also activates a `Get OS Route Protocol` to proactively configure the OS routing path from CA5 to CA1. to this purpose, the control plane exploits following messages:

- `OS Routing Request`: from CA4 to SDN controller containing CA id, destination id, application requirements and path selection metric;
- `OS Routing Add Route`: from SDN controller to intermediary nodes of the computed path to configure their routing tables;
- `OS Routing ACK/Abort`: from intermediary nodes to the SDN controller reporting the routing table modification result;
- `OS Routing Push`: from the SDN controller to CA5 to notify the sender application the availability of an OS routing path towards CA1.

Once received then new path node sequence, CA4 starts modifying the payload header to reroute packets towards CA2. Furthermore, the receiver on CA1 monitors the end-to-end latency (by exploiting the `sentTimestamp` within vibration packets) and triggers an alert as soon as the latency is greater than 250 ms. In this case, it interacts with the local CA, in charge of notifying the SDN controller to inform the sender that it should switch to the OS routing mechanism.

Fig. 12 presents the per packet latency of vibration data. In the first phase, the message rate is 50 msg/s and the `maxVibrationIndex` is set to 5 and thus most of the packets are discarded. After about 2000 ms the `maxVibrationIndex` value rises to 15, with the consequence that the payload content rule on node CA4 does not drop packets anymore. Then, after about 5200 ms the `maxVibrationIndex` is set to 25, triggering the identification of a faster path. To this purpose, it is worth noting

Listing 2: Vibration payload content rule.

```
public class VibrationDataPlaneRule
 extends AbstractDataPlaneRule
 implements Serializable {

 private long vibrationDataId = 123456;
 private boolean dropPacketsEnabled = true;
 private boolean fastPathFound = false;
 private String[] newFastestPath;

 public void applyRule(UnicastPacket up) {
  UnicastHeader header = up.getHeader();
  long dataTypeId = header.getDataType();
  if (dataTypeId == vibrationDataId) {
    VibrationData message = (VibrationData)
      Utils.deserialize(up.getBytePayload());
    int value = message.getMaxVibrationIndex();
    if (value <= 10 && dropPacketsEnabled) {
      int probability = Utils.randomInt(10);
      if (probability < 9) dropPacket(up);
    } else if (value <= 20) {
      dropPacketsEnabled = false;
    } else if ( ! fastPathFound ) {
      if (dropPacketsEnabled) {
        dropPacketsEnabled = false;
      }
      newFastestPath =
        findNewFastestPath(header);
      fastPathFound = true;
    }
  }
  if(fastPathFound) {
    up.setDestination(newFastestPath);
}}}
```

that the first packet routed towards the faster path suffers a slightly greater latency, since its delivery is delayed by the control plane `Get Path Protocol` between node CA2 and the SDN controller to identify the new path, lasting about 36 ms. Following packets reach the destination in a prompter manner, taking advantage of the lower latency of the path towards CA2. However, after about 8100 ms the sender increases the message rate to 150 msg/s, e.g., to better sample vibration data. The increased network load negatively affects the message delivery, since the increased load on nodes to dispatch packets saturates the CPU. At about 9200 ms the end-to-end latency exceeds 250 ms and thus the receiver alerts the SDN controller requiring the sender to switch to OS routing. In this manner, vibration data can reach the destination again in a prompt manner with a reduced latency. Note that some packets previously sent exploiting the payload content mechanism are still flowing towards the destination exploiting the overlay network and reach the destination later than some packets sent via OS routing.

In conclusion, the performance results measured on real deployment environments for our Multi-LANE working prototype not only demonstrate the feasibility of the proposed solution, but also its capability of adapting packet dispatching in relation to (eventually time-varying) requirements in terms of either expressiveness or limited overhead. In particular,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2020.2995903, IEEE Transactions on Network and Service Management

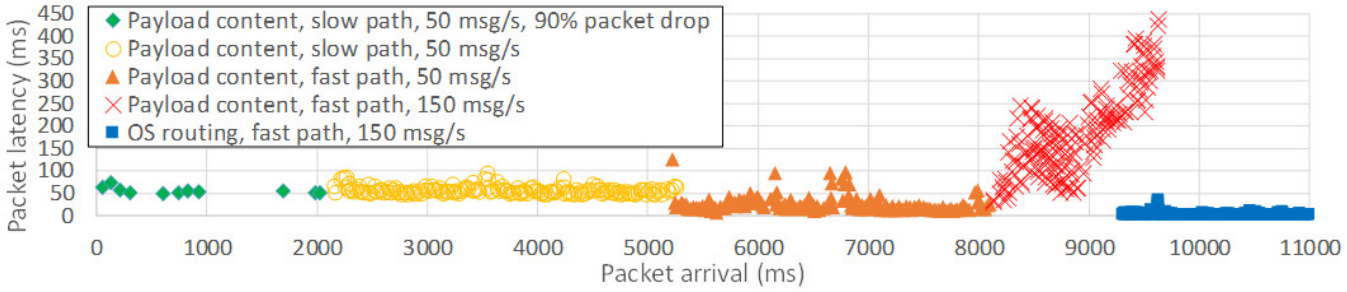


Fig. 12: MLR-based dynamic management of vibration data.

overlay network solutions exploiting type and payload content provide much richer packet dispatching rule expressiveness, e.g., by allowing to selectively drop packets based on the payload content, at the cost of relatively low computing overhead. However, the additional computation is limited, allowing to achieve performance similar to the native IP solution till 100 msg/s on resource-limited nodes such as Raspberry Pi ones, much less performing than currently spread laptops and smartphones. At higher packet rates, only native IP and overlay network node sequence forwarding mechanisms are efficient enough to allow prompt packet dispatching. However, the former has the notable drawback of imposing to manage operating system routing tables, requiring administrative access and incurring in long path setup latency.

V. Related Work

While SDN emerged primarily to manage switches of closed and geographically centralized environments, its adoption has quickly proven its benefits also in more dynamic scenarios with relaxed requirements in terms of closeness and geographical centralization.

Considering Fog and Edge computing, [12] proposes to exploit SDN to deliver and deploy new services in IoT environments in a faster and more cost-effective manner. In addition, SDN can make easier the dynamic integration and collaboration of Edge and Cloud environments [13]. The SDN approach can be also adopted together with Blockchain to deliver a fully-distributed Cloud architecture based on Fog nodes [14]. Finally, SDN is fruitfully adopted to support load balancing in Fog environments [15], [16]. Interested readers can refer to [17], [18] for comprehensive studies on the adoption of SDN in Edge and Fog environments.

Similarly to our proposal, the very recent effort of the Fog05 initiative [19] targets Fog environments characterized by nodes with limited and heterogeneous resources and networks with a high degree of node de/attaching rate. Fog05 pushes on the adoption of a key/value store to provide a distributed storage, monitoring, and configuring solution. Moreover, it supports the runtime deployment of services in terms of VMs, containers, Unikernels, and binary code to increase Fog node flexibility and configurability. However, Fog05 does not consider networking issues arising from Fog environments composed of several nodes. In fact, Fog05 assumes that Fog nodes are located within a single network, which cannot be granted in dynamic environments composed of Fog nodes managed by different users/authorities. For these reasons, we consider our SDN-based MLR solution as complementary w.r.t. the Fog05 initiative.

Similar approaches have been proposed also in research fields that can be considered as predecessors of Edge computing. Considering WSNs, [20] exploits the SDN approach to efficiently manage cooperative communication and task execution among nodes while [21] exploits SDN to allow the adoption of a flow splitting algorithm minimizing the traffic load. Furthermore, some solutions not only adopt the SDN approach but also exploit OpenFlow-like protocols. For instance, [22] proposes to adopt an OpenFlow extension to more easily adapt the behavior of a WSN to dynamically changing goals and applications and more efficiently enforce different policies. SDN-WISE [23] extends OpenFlow to optimize the communication among sensor nodes and the SDN controller and to program nodes as finite state machines.

Focusing on the adoption of the SDN approach in MANET environments, only recently a few contributions have started to emerge. For instance, [24] presents a solution to offload the cellular network by controlling mobile nodes (and their routing tables) based on a centralized SDN controller residing in the fixed infrastructure. [25] adopts the adaptive principles of the SDN approach to seamlessly recover from disruptions of computation-intensive MANETs. Similarly, [26] applies the SDN approach to mobile clouds with the primary goal of adapting/tuning the network in relation to varying wireless environments, e.g., due to mobility and unreliable wireless link conditions.

Finally, let us briefly note that the MLR approach has some similarities with cross-layer routing solutions, since they consider information from different abstraction layers to manage networking resources. For instance, [27] compares many contributions aiming at improving routing protocols in VANETs by jointly exploiting information at the physical, medium access control, and network layers. Instead, [28] pushes for the cooperation between routing, scheduling, and channel allocation functions by allowing such functions to simultaneously update the routing table. However, the typical goal of cross-layer routing is to identify the best path based on heterogeneous information, not to exploit multiple routing mechanisms at the same time like our MLR solution does.

To the best of our knowledge, our work is the first one adopting the MLR approach in conjunction with SDN to manage heterogeneous Fog environments. In particular, our

solution exploits MLR data plane to take advantage of differentiated forwarding mechanisms together with an overlay-network control plane to enable and setup different forwarding mechanisms in a unified and coordinated manner. Moreover, it is strongly original in presenting a working prototype to the community of researchers in the field, to be exploited and extended to foster SDN-based MLR solutions in Fog environments.

VI. Conclusions

The paper presents a novel model and architecture allowing to selectively adopt the proper forwarding mechanism in Fog environments, by considering application requirements as well as the current state of the network. In particular, the adoption of the SDN approach allows to support multiple packet dispatching solutions with differentiated tradeoffs in terms of flexibility/expressiveness of forwarding mechanisms and imposed overhead for both control messages and data dispatching. The encouraging results achieved so far based on the Multi-LANE working prototype are stimulating additional research work. We are mainly working on the development of a large scale Fog environment to finely assess the performance of our solution. We are also developing a companion SDK to make easier the dynamic development and run-time deployment/activation of novel traffic engineering and routing policies.

References

[1] S.M.A. Oteafy, H.S. Hassanein, "IoT in the Fog: A Roadmap for Data-Centric IoT Development", IEEE Comm. Mag, vol. 56, no. 3, 2018.

[2] M. Yannuzzi, F. van Lingen, Anuj Jain, O.L. Parellada, M.M. Flores, D. Carrera, J. Luis Prez, D. Montero, P. Chacin, A. Corsaro, A. Olive, "A New Era for Cities with Fog Computing", IEEE Internet Computing, vol. 21, no. 2, 2017.

[3] T. Taleb, K. Samdanis, B. Mada, H. Flinck, S. Dutta, D. Sabella, "On Multi-Access Edge Computing: A Survey of the Emerging 5G Network Edge Cloud Architecture and Orchestration", IEEE Comm. Surveys & Tutorials, vol. 19, no. 3, 2017.

[4] C. Giannelli, P. Bellavista, D. Scotece, "Software Defined Networking for Quality-aware Management of Multi-hop Spontaneous Networks", Int. Conf. on Computing, Networking and Communications (ICNC 2018), Maui, Hawaii, USA, March 5-8, 2018.

[5] P. Bellavista, A. Dolci, C. Giannelli, "MANET-oriented SDN: Motivations, Challenges, and a Solution Prototype", 19th IEEE Int. Symp on a World of Wireless, Mobile and Multimedia Networks (WoWMoM 2018), Chania, Greece, June 12-15, 2018.

[6] Open Networking Foundation, "OpenFlow", available online at https://www.opennetworking.org/sdn-resources/openflow (last accessed on April 30th, 2019).

[7] P. Bellavista, A. Corradi, C. Giannelli, "Middleware for Differentiated Quality in Spontaneous Networks", IEEE Pervasive Computing, vol. 11, no. 3, March 2012.

[8] P. Bellavista, A. Corradi, C. Giannelli, "Differentiated Management Strategies for Multi-hop Multi-Path Heterogeneous Connectivity in Mobile Environments", IEEE Trans. on Network and Service Management, vol. 8, no. 3, 2011.

[9] C. Fang, H. Yao, Z. Wang, W. Wu, X. Jin, F.R. Yu, "A Survey of Mobile Information-Centric Networking: Research Issues and Challenges", IEEE Comm. Sutveys & Tutorials, vol. 20, no. 3, 2018.

[10] Z. Qin, L. Iannario, C. Giannelli, P. Bellavista, N. Venkatasubramanian, "Smart communications via a tree-based overlay over multiple and heterogeneous (TOMH) spontaneous networks", 2013 Int. Conf. on Smart Communications in Network Technologies (SaCoNeT), 2013.

[11] P. Bellavista, P. Gallo, C. Giannelli, G. Toniolo, A. Zoccola, "Discovering and Accessing Peer-to-peer Services in UPnP-based Federated Domotic Islands", IEEE Transactions on Consumer Electronics, vol. 58, no. 3, August 2012.

[12] J. Pan, J. McElhannon, "Future Edge Cloud and Edge Computing for Internet of Things Applications", IEEE Internet of Things Journal, vol. 5, no. 1, 2018.

[13] K. Kaur, S. Garg, G.S. Aujla, N. Kumar, J.J P.C. Rodrigues, M. Guizani, "Edge Computing in the Industrial Internet of Things Environment: Software-Defined-Networks-Based Edge-Cloud Interplay", IEEE Communications Magazine, vol. 56, no. 2, 2018.

[14] P.K. Sharma, M. Chen, J.H. Park, "A Software Defined Fog Node Based Distributed Blockchain Cloud Architecture for IoT", IEEE Access, vol. 6, 2018.

[15] X. He, Z. Ren, C. Shi, J. Fang, "A novel load balancing strategy of software-defined cloud/fog networking in the Internet of Vehicles", China Communications, vol. 13, 2016.

[16] S. Misra, N. Saha, "Detour: Dynamic Task Offloading in Software-Defined Fog for IoT Applications", IEEE Journal on Selected Areas in Communications, vol. 37, no. 5, 2019.

[17] A.C. Baktir, A. Ozgovde, C. Ersoy, "How Can Edge Computing Benefit From Software-Defined Networking: A Survey, Use Cases, and Future Directions", IEEE Communications Surveys & Tutorials, vol. 19, no. 4, 2017.

[18] F.Y. Okay, S. Ozdemir, "Routing in Fog-Enabled IoT Platforms: A Survey and an SDN-Based Solution", IEEE Internet of Things Journal, vol. 5, no. 6, 2018.

[19] A. Corsaro, G. Baldoni, "fog05: Unifying the computing, networking and storage fabrics end-to-end", 3rd Cloudification of the Internet of Things (CIoT), July 2018.

[20] J. Zhou et al., SDN-Based Application Framework for Wireless Sensor and Actor Networks, IEEE Access, vol. 4, 2016.

[21] G. Li et al., "Traffic Load Minimization in Software Defined Wireless Sensor Networks", IEEE Internet of Things Journal, vol. 5, no. 3, 2018.

[22] T. Luo, H.P. Tan, T.Q.S. Quek, "Sensor OpenFlow: Enabling Software-Defined Wireless Sensor Networks", IEEE Communications Letters, vo. 16, no. 11, Nov. 2012.

[23] L. Galluccio et al., SDN-WISE: Design, prototyping and experimentation of a stateful SDN solution for WIreless SEnsor networks, IEEE Conf. on Computer Communications (INFOCOM), pp. 513-521, 2015.

[24] H.C. Yu, G. Quer, R.R. Rao, "Wireless SDN mobile ad hoc network: From theory to practice", 2017 IEEE International Conference on Communications (ICC), 2017.

[25] V. Balasubramanian, A. Karmouch, "Managing the mobile Ad-hoc cloud ecosystem using software defined networking principles", 2017 International Symposium on Networks, Computers and Communications (ISNCC), 2017.

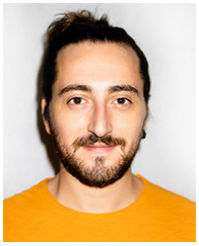
[26] I. Ku, Y. Lu, M. Gerla, "Software-Defined Mobile Cloud: Architecture, services and use cases", 2014 International Wireless Communications and Mobile Computing Conference (IWCMC), 2014.

[27] A. Awang, K. Husain, N. Kamel, S. Aissa, "Routing in Vehicular Ad-hoc Networks: A Survey on Single- and Cross-Layer Design Techniques, and Perspectives", IEEE Access, vol. 5, 2017.

[28] B. Shin, J. Choe, B. Kang, D. Hong, Y. Park, "Cross-layer resource allocation with multipath routing in wireless multihop and multichannel systems", Journal of Communications and Networks, vol. 13, no. 3, 2011.

Paolo Bellavista received the Ph.D. degree in computer engineering from the University of Bologna, Italy, in 2001. He is a Full Professor of distributed and mobile systems at the CSE department of the University of Bologna, Italy. His primary research activities span from mobile middleware to wireless sensor and actuator networks, from pervasive mobile computing infrastructures to industrial Internet of Things, from edge cloud computing to online stream processing in manufacturing industry applications.

Carlo Giannelli received the Ph.D. degree in computer engineering from the University of Bologna, Italy, in 2008. He is currently an Associate Professor in computer science with the University of Ferrara, Italy. His primary research activities focus on Industrial Internet of Things, Software Defined Networking, Blockchain technologies, location/based services, heterogeneous wireless interface integration, and hybrid infrastructure/ad hoc and spontaneous multi-hop networking environments based on social relationships.



Dmitrij David Padalino Montenero is an industrial research fellow at the Interdepartmental Centres for Industrial Research of the University of Bologna, Italy, where he received a Master's Degree in Computer Engineering in 2019. His primary research activities focus on Cloud Computing, Containerization, Industrial Internet of Things, Software Defined Networking, and hybrid infrastructure/ad hoc and spontaneous multi-hop networking environments.