DOI: 10.1109/mnet.001.2100363

Link layer Connectivity as a Service for Ad-hoc Microservice Platforms

Luis F. Gonzalez[1], Ivan Vidal[1], Francisco Valera[1], and Diego R. Lopez[2]
[1] *Universidad Carlos III de Madrid, Av. Universidad, 30, 28911, Leganés (Madrid), Spain*
[2] *Telefónica I+D, C/ Zurbarán 12, 28010 Madrid, Spain*

***Abstract*— Microservice platforms have brought many advantages to support the deployment of lightweight applications at both near the edge and datacenters. Still, their suitability to support telecommunication and vertical services beyond the network edge is far from being a reality. On the one hand, their flat networking approach does not support the establishment of link-layer connectivity among the different components of telecommunication and vertical services (e.g., access points, routers, specific-purpose servers, etc.) due to their reliance on high-level APIs. On the other hand, their networking approach has not been designed to operate over ad-hoc networks built by the resource-constrained devices that may be available beyond the network edge. This can lead to suboptimal behaviours for the delivery of data traffic between microservices. This paper presents the results of a research collaboration between Universidad Carlos III of Madrid and Telefónica: L2S-M. Our solution provides a programmable data plane that enables the establishment of on demand link layer connectivity between microservices on ad-hoc networks. This solution has the flexibility to execute different algorithms to build traffic paths between microservices, as well as to react against temporary link breakdowns, which could be present in these types of networks. The paper presents a proof of concept for a functional validation of L2S-M, using an aerial ad-hoc network deployed at 5TONIC laboratory in collaboration with Telefonica. The validation results showcase the proper operation of L2S-M as a networking service for microservice platforms in ad-hoc networks, including its ability to reconfigure the programmable data plane when link disruptions occur.**

***Index Terms*— Microservices, Containers, Ad-hoc networks, Software Defined Networking (SDN), Kubernetes**

I. Introduction

Since the worldwide adoption of the Internet to provide digital services, developers and telecommunication operators have put significant effort into the deployment of better applications in both near edge networks and datacenters. However, this proved to be a difficult task using traditional monolithic applications due to their high costs in terms of development and maintenance: A single failure of one component would compromise the whole functionality of an application, introducing longer service cut-offs and recovery times. New architectural models had to be proposed to find solutions to mitigate these issues, leading to the design of microservice architectures.

Microservices architectures allow splitting complex functionalities into smaller interconnected modules, which in turn saves development time and reduces the scope of the failures that may occur in production environments.

For the last couple of years, many microservice platforms have been focusing on cloud environments, where high amounts of computational resources are available and connectivity through Gb/s links can be provided most of the time. Many networking solutions have been developed to communicate these microservices with this environment in mind, mostly consisting of implementing a flat full mesh where all microservices can communicate with each other. Using this model, networking is completely dissociated from the microservices themselves, allowing developers to focus on service functionalities without worrying about networking issues, since they can assume that communication will always be available.

However, cloud environments are not the only areas that have benefited from increased attention over the last couple of years. Both industry and academia have seen distributed technologies, like aerial networks (i.e., networks composed of several Unmanned Aerial Vehicles – UAVs), as perfect candidates for service provisioning in the future.

These networks have demonstrated their ability to deploy networking applications to deliver novel services that other technologies struggle to implement. However, the components that build these applications must establish communication through the IP or link layers, and current networking solutions for microservice platforms are not able to accurately implement this behaviour due to their reliance on high-level APIs. Instead, they leave all routing aspects to the underlying protocol running at the network, which might perform decisions that are not entirely beneficial for certain traffic types [1]. This approach limits the potential of microservice platforms as suitable candidates to deploy complex networking applications in distributed heterogeneous ad-hoc networks.

This paper presents the technical results of a research collaboration established between Universidad Carlos III de Madrid and Telefónica. This paper will introduce a novel solution to enable microservice connectivity over ad-hoc networks: L2S-M. Our solution is a networking service for microservice platforms able to build a programmable data plane that supports the establishment of link layer connectivity between microservices. This solution is specialised in ad-hoc networks with resource-constrained devices, although it can be used in other types of environments. The design of L2S-M leverages Software Defined Networking (SDN) technologies, this way supporting different routing algorithms to build data paths among microservices on wireless ad-hoc networks. Our solution has been validated in 5TONIC, i.e., the 5G Telefonica Open Network Innovation Centre (5TONIC), founded by

Telefónica, making use of a production cloud-environment and a wireless ad-hoc network infrastructure.

II. Related Work

Telecommunication operators and developers have experienced an increment in user demand for faster and more efficient services. However, monolithic applications (i.e., all service components packed as a single unit) have proven to be insufficient to fulfil these demands due to their lesser degree of flexibility in terms of failure recovery and distribution [2]. In contrast, microservices have risen as one of the main alternatives to counter all its issues. By splitting applications into different modules under the principles of modularity, scalability, and resiliency [3], service providers can distribute complex applications over multiple machines and/or domains. Several well-known applications use this microservice architecture to supply millions of users each day, including Spotify, Twitter or Netflix [4].

The increase of popularity of this new architectural approach is a direct consequence of the adoption of virtualization techniques, in particular container technology, by both industry and academia. With containers, a host can allocate part of its own resources into a new "emulated" machine, which runs isolated from the original host. Containers differ from the traditional virtualisation paradigms because, instead of executing a full host Operative System (OS) into virtualized resources, some of its functionalities are relied into the host kernel instead, saving computational resources in the process [5].

Thanks to its lightweight nature, containers are regarded as ideal platforms to host microservices. Nevertheless, since microservices will be deployed across multiple computational entities alongside a distributed infrastructure, it is essential to be able to manage their execution and establish their relationships to build a complex service.

Due to their increasing popularity, there are multiple choices for container management in the commercial and open-source space, highlighting Docker Swarm [6], specialised in Docker containers; or OpenStack [7], one of the most recognized cloud solutions that support container development. Nevertheless, the most popular solution to manage containerised workloads is Kubernetes [8], thanks to its high degree of flexibility, simplicity, and constant support from a large community of developers. Research in container management shows that Kubernetes is regarded as the best performing solution in production environments [9]. Kubernetes has also been used as a template to build other advanced microservice platforms with enhanced functionalities, such as OpenShift [10] from RedHat, or other fully orchestrated microservice platform solutions like Amazon Elastic Container Service [11], which also defines the logic relation between all microservices used to build a complex application.

As it has been previously mentioned, microservices must be interconnected among each other to provide a full working application. This fact has led to the creation of a set of specifications to manage networking resources for container-based microservices: the Container Networking Interface (CNI). CNI consists of a specification and libraries to write plugins to configure network interfaces in Linux containers, allowing their interaction with physical interfaces [12]. In principle, this specification oversees the definition of how networking services must manage, enable, and attach networking interfaces into the container network space. This component is an agnostic element, compatible with multiple container runtimes and platforms.

Since CNIs are only "templates" used to define networking configurations, developers must define their own approaches to apply these standards into their own solutions. CNI plugins are designed to perform all actions defined by the CNI specifications. Afterwards, each CNI plugin implementation must define how the rest of communications must be performed, i.e., how all the microservices present over an infrastructure will be able to communicate with each other. As an example, some CNI plugins establish virtual extensible LANs (VXLANs) among nodes of a microservice platform in order to guarantee IP level communications among microservices.

Table 1 provides a comparison of the main CNI plugins used in the microservice management platforms.

III. A Networking Service for ad-hoc networks

A. *Networking services in microservice platforms*

Traditional networking service solutions in microservice platforms have relied on a flat-networking approach where all microservices are able to reach each other and send information through high-level APIs (for example, RESTful APIs). This approach is optimal for applications whose information can be delivered at the application level, helping microservices to "forget" about the networking tasks and focus on providing their functionality to the system to build a more complex service.

However, this approach is not suitable when communications cannot be established through the application layer, which is the case of networking functions used in telecommunication services. For example, a routing function must be connected with other routing functions at the link layer (e.g., through point-to-point links) to properly route the incoming traffic to the corresponding destination.

Table 1 depicts the most prominent networking services available for different microservice platforms. As it can be seen in the table, these solutions lack the tools to deal with wireless ad-hoc networks.

B. *Solution description*

Wireless ad-hoc networks have radically different characteristics compared to datacenter networks. Nodes use other peers as relays for establishing connectivity amongst all of them. In our vision, these networks will be deployed in scenarios where a wide variety of heterogeneous devices with different characteristics in terms of computational power and battery lifetime will be interconnected, building topologies for the provision of network functionalities such as Access Points, firewalls, etc. However, to allow these services to operate correctly, communication at the link-layer level is necessary. Traditional networking solutions in microservice platforms lack the necessary tools to enable this behaviour, leaving all the traffic delivery decisions to the underlying protocol running over the network.

	Flannel	Calico	Kubenet	Weavenet	OpenShift SDN	Linen	L2S-M
Developer/Owner	CoreOS (Open-Source)	Tigera (Open-Source)	Kubernetes (Open-Source)	Weave Works	RedHat	Open-Source	Open-Source (UC3M + Telefónica)
Does it require dependencies in each node?	No, fully containerised solution	No, fully containerised solution	Linux libraries required	No, fully containerised solution	No, fully containerised solution	Yes, OVS-switches must be installed and active in every node.	No, fully containerised solution
Main CNI plugin. networking characteristics.	Flat networking model using VXLANs between nodes. Does not implement further functionalities	Provides full networking stack through overlay networking (IP tunneling). Flexible routing mechanisms and plugin selection. Implements network policy enforcement	Simple flat networking approach. Connection between nodes left to cloud providers	Builds overlay Layer 2 network between containers and provides automatic discovery mechanisms through DNS and load balancing.	Builds and overlay network using OVS switches in conjunction with SDN technology to provide project-level isolation.	Level 3 overlay networking (VXLAN) based on switching through OVS Switches. Allows the use of SDN Controllers to modify switch flows.	Builds a programmable data plane to establish link layer connectivity between containers. Uses SDN technology to modify the data plane behaviour.
Integration over microservice platforms	Only compatible with Kubernetes	Compatible with multiple platforms, including Kubernetes, OpenShift, Docker EE and OpenStack	Only compatible with Kubernetes in Linux platforms	Compatible with multiple Kubernetes distributions (Kubernetes, Amazon ESC…) and microservice platforms such as Mesos or Marathon.	Designed for the OpenShift Container Platform (Kubernetes-based).	Only compatible with Kubernetes	Compatible with the principal platforms (Kubernetes, Apache Mesos, Marathon, etc,)
Supports point-to-point links between containers?	No	No	No	Yes, but with limitations. Agents on every node are in charge of routing over a single Overlay Layer 2 network.	No, it can separate services but not establish point-to-point links between containers.	No	Yes
Community support and expansion	Widely extended due to its simplicity and effectiveness	Widely extended solution in big production environments due to its high functionality and flexibility	Generally used alongside a cloud provider or single node clusters, mostly for training purposes	Widely extended in production environments for its functionalities and easy setup.	Used in OpenShift installations, one of the most popular Kubernetes distributions.	Very limited community adoption. Discontinued in late 2017	Under development

Table 1: Networking solutions for microservices comparison

This paper presents a networking service for microservice platforms with the objective of providing network connectivity as a service in ad-hoc networks: L2S-M. In principle, L2S-M will deliver a programmable data plane where any container in an ad-hoc network can be connected to any other one managed by the platform, regardless of its placement within the network. This programmable data plane will allow microservices to be connected at the link layer, through point-to-point or multi-access links, which can be created on demand. L2S-M will be able to apply traffic engineering mechanisms based on different metrics (e.g., energy consumption, traffic delay...) to select the best possible path. This in turn allows the networking service to build an appropriate data path between different services. More concretely, L2S-M will enable the creation of virtual networks on demand that microservices will be able to attach to. These virtual networks will provide link layer connectivity to microservices attached to them. This effectively supports the establishment of point-to-point or multi-access links among microservices. The isolation of particular services can be preserved by simply attaching their applications to specific virtual networks, enabling the secure operation of microservices spread over the ad-hoc network infrastructure.

To fulfil this objective, this service will take advantage of IP tunnelling mechanisms (e.g., VXLAN or GRE tunnels) to establish point-to-point links between neighboring nodes of the ad-hoc network. In this regard, the main component of the solution is the presence of programmable switches at each of the compute nodes of the ad-hoc network. Each one of these

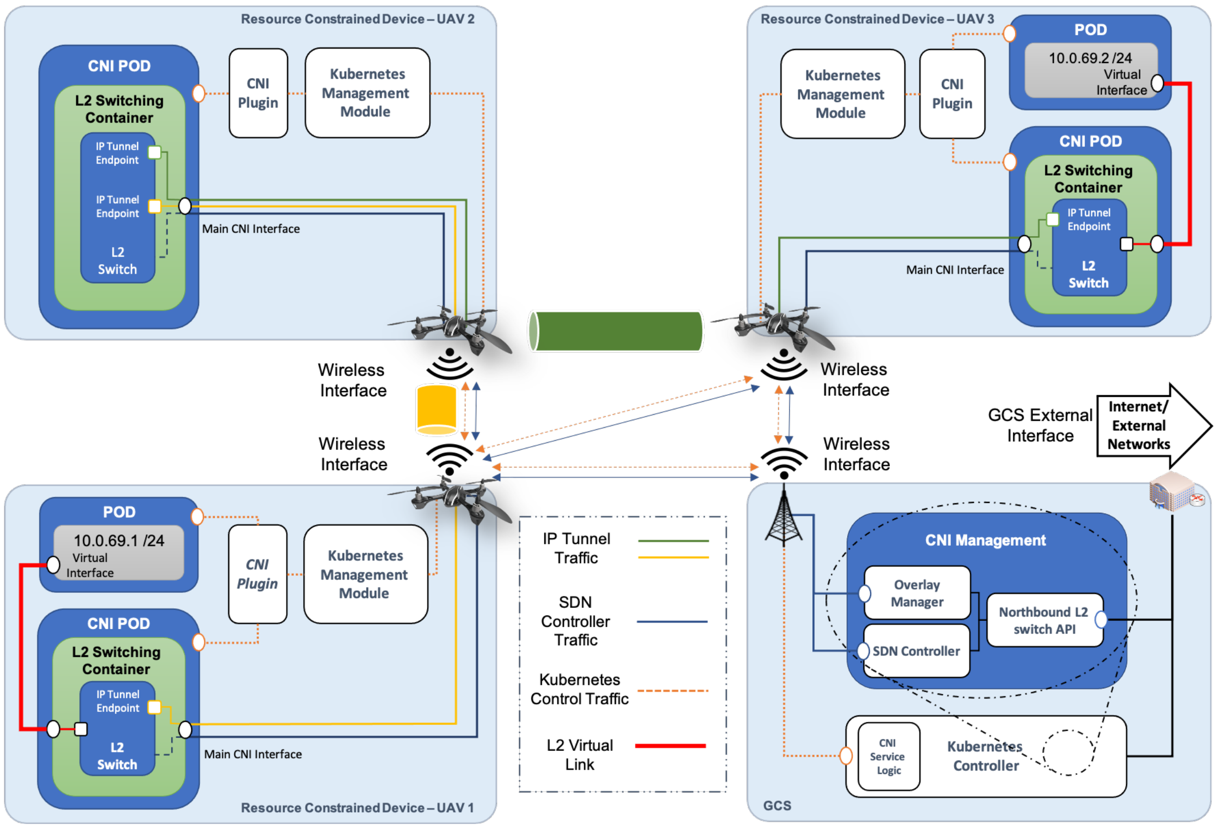

Figure 1: Kubernetes-based L2S-M design over an aerial ad-hoc network

switches will be executed in a container to take advantage of the benefits of virtualisation technologies. The interconnection of neighboring switches is performed through IP tunnelling mechanisms. Using this method, a programmable switch infrastructure is built with the peers interconnected over the ad-hoc network.

In this solution, we introduce the Overlay Manager as an entity that will be always monitoring the ad-hoc network and modify the topology of programmable switches, either dynamically (i.e., choosing different neighbours after a link breakdown) or manually. The combination of both technologies allows the networking service to build the programmable switch infrastructure between containers, regardless of their location in the ad-hoc network.

To support the creation of virtual networks over the programmable data plane offered by the link layer switches, L2S-M relies on a Software-Defined Networking (SDN) controller. This SDN controller will interact with the switches to modify their traffic rules, specifying which ports must be used to forward traffic to other containers located across the network. Moreover, this element may react against unforeseen events related with intermittent connectivity, interacting with the data plane switches in such a way that the service downtime can be considerably reduced. This alternative improves “pure-SDN" approaches in two key ways. First, since L2S-M builds an overlay of point-to-point links between virtual switches, it allows the utilization of off-the-shelf SDN software implementations, which do not need to include specific developments to support wireless ad-hoc networks. Second, it provides the flexibility to use different routing algorithms to build data paths among microservices, with the configuration of proper forwarding rules at the switches.

Figure 1 depicts the implementation of L2S-M as a Kubernetes CNI plugin over an aerial network composed of three UAVs and one Ground Control Station (GCS). It is relevant to mention that this service can be exported to other microservice platforms, since its functional entities and their relationships will be preserved in other solutions.

To clarify the service functionality, the most relevant steps that will be executed by the networking service when attaching a new node are the following:

1) Attachment of a node to the ad-hoc network infrastructure: Once a compute node is attached to an ad-hoc network infrastructure, the Kubernetes controller will send all the control information used by the Kubernetes Manager Module for its configuration. This module oversees the configuration of all the Kubernetes services, and it is also responsible for periodically communicating with the controller to update the resource status, lifetime checks, etc. Among its tasks, it will configure and instantiate the main container provided by the CNI plugin: the programmable Level 2 switch (CNI Pods in the figure UAVs). This switch container will only have a single interface used for external communications (Main CNI Interface).

Meanwhile, the Overlay Manager module (embedded in the CNI Management at the controller) will instruct the nodes to build the IP tunnels with their neighbors, basing its decisions on the desired topology. This information will be sent as part of the Kubernetes management information. Once it arrives at the corresponding nodes, the CNI plugin will build the IP Tunnels using the chosen technology and attach the IP tunnel endpoints to the switches, finishing the data plane that containers will use to send its data to other peers, depicted in the figure by the tunnels.

2) Deployment of a microservice: Using the same CNI plugin logic on each node, every new microservice container will be connected to the switch through a virtual interface generated by the CNI plugin module, creating the virtual link shown in red color in Figure 1.

With the switch infrastructure built between the platform nodes, the SDN controller, which is also part of the CNI Management module, can start providing the programmability aspect to the data plane. The SDN Controller, assisted by an algorithm (that may take into account one, or several, network metrics such as hop distance or energy consumption), will send flow information to all the switches in the network, defining their forwarding rules and the ports they must use for every traffic flow. The switches communicate with this controller at special events, allowing the controller to react against connectivity related events to manage them accordingly. Like the Overlay Manager traffic, all this traffic will be included in the controller traffic sent by Kubernetes.

3) Exchange of data traffic: The combination of all these components will allow containers to use the programmable data plane to communicate through link layer connectivity. Its sequence will be the following: data will be outputted from the virtual interface into the UAV 1 link layer switch, which will select the corresponding port to output its data based on the information provided by the SDN Controller. Afterwards, it will encapsulate this data and send it using the corresponding IP tunnel, forwarding its information through its main interface to reach the intermediate UAV (UAV2). As it can be seen in the figure, this node selects an intermediate UAV even though it has direct connectivity with the destination, an action that cannot be performed by other networking services since they use the underlying routing protocols (e.g., MANET protocols that select the shortest path). Once it reaches the other IP tunnel endpoint at UAV 2, the sequence is repeated until it reaches the destination link layer switch in UAV 3, which will forward the decapsulated data through its attached virtualized port, reaching the destination container.

It is important to point out that control traffic (i.e., traffic exchanged between the Kubernetes controller and the UAVs) will not use the programmable data plane. Given the reduced load of this traffic in comparison with data traffic [13], L2S-M relies on the underlying routing protocol of the wireless ad-hoc network to exchange this type of traffic. Therefore, the programmability aspects are only provided for the data plane, whose traffic patterns can vary depending on multiple parameters like traffic class, application types, etc. This is especially useful in wireless ad-hoc networks, where strategies like the shortest-path might not be the most suitable for data plane communications (as we shown in our previous works in [1,13]) since, due to their intermittent connectivity, other paths might go through healthier peers that will maintain a more stable connection, which in turn can reduce service cutoffs). With the adoption of SDN technologies, L2S-M allows using different routing algorithms to create data paths over the ad hoc networks, so other strategies based on other metrics (e.g., a combination of shortest-path and energy consumption) can be used.

C. *The advantages of L2S-M*

This novel networking service approach for microservice platforms tackles several issues that most available plugins have not fully resolved. In these cases, they usually take the flat networking approach, which lacks the flexibility required in ad-hoc environments where link disruptions between nodes are a common occurrence. In consequence, this leads to suboptimal traffic paths for microservices data traffic in ad-hoc networks since the networking service is not able to influence the traffic paths being used. Furthermore, some of these solutions are not entirely containerised, so they do not benefit from the advantages of container technology.

In contrast, L2S-M allows the provision of network connectivity as a service by deploying a completely programmable data plane where containers can be directly reached using link layer connectivity. Thanks to the introduction of SDN technology, it will be able to influence networking paths through a variety of algorithms, as well as allowing the service to react in advance to changes performed in the network to select the most appropriate paths to distribute data plane information without tempering with their network layer configuration, including its upper layers. Some examples of algorithms that could be used by L2S-M are metric-based algorithms (such as the one used in BABEL protocol specialized in mobility), reactive algorithms (used in AODV), or geographic-based algorithms [14]. This solution allows the implementation of different traffic engineering features by creating link layer networks and attaching microservices to the appropriate ones, isolating them from the rest. This can be useful to adapt the platform to the needs of a particular service (for example, a temporary increase of user demand) or the prioritisation of services with more strict requirements (e.g., emergency services).

Table 1 compares the most popular CNI Plugin solutions available now for the Kubernetes platform against the proposed solution. As it can be seen in the table, L2S-M's strengths highlight its potential as a viable networking service for ad-hoc networks against other networking service solutions in microservice platforms.

IV. Proof of Concept and Validation

A. *Building the proof of concept*

This proof of concept was performed in the 5G Telefonica Open Network Innovation Centre (5TONIC), created by Telefónica in Madrid, as an experimental telecommunications facility focused on 5G technologies.

This proof of concept uses part of the scenario depicted in Fig. 1: we use two of the aircrafts, UAV 1 and UAV 3, as compute nodes of the network, as well as the GCS. We consider that the UAVs remain hoovering (static position) in direct line of sight

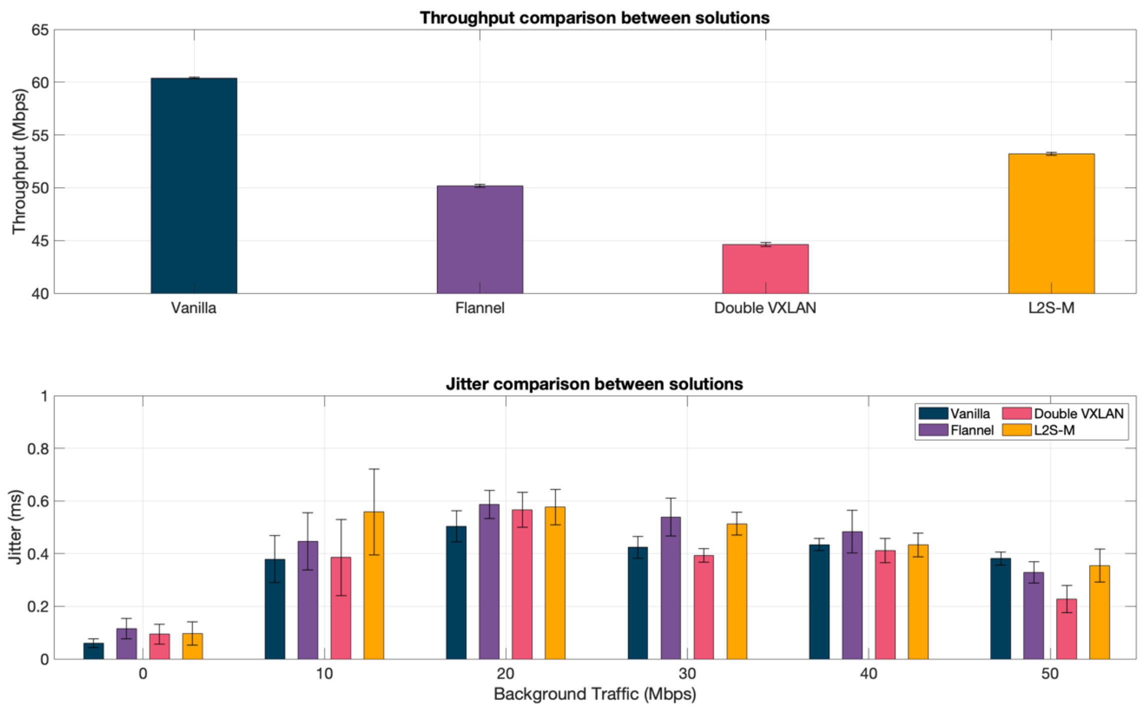

Figure 2: Average Throughput and Jitter comparison for traffic between UAV 1 and UAV 3 for each networking solution against L2S-M

with each other, building the fixed network topology depicted in the figure.

We have used two Parrot Beebop UAVs onboarding a single board computer (RPi Model 4 with 4 Gb of RAM for UAV 1 and RPi Model 3B with 2 Gb of RAM for UAV 3). In a realistic scenario, these aircrafts would be directly under the control of a terrestrial GCS. In this case, we assume that the GCS also provides a terrestrial compute node through a Mini ITx with 8GB of RAM and 8 CPUs (Ubuntu 20.04 amd64). All aircrafts use a single wireless interface to establish an ad-hoc network through Wi-Fi technology, particularly in the 2,4 GHz band over channel 7, set up in close proximity to each other. The GCS compute node uses an ASUS USB-N10 wireless adapter for wireless communications.

Over the deployed ad-hoc network, a Kubernetes platform was deployed using Kubeadm (version 1.21) with Docker 20.10.6 as its container runtime. Moreover, the Flannel CNI Plugin hasbeen chosen as the networking solution for the cluster to provide external connectivity and distribute control plane data to the containers (in Kubernetes terms, pods) deployed in the worker nodes (Kubernetes term for compute nodes).

To build the programmable data plane presented in Section III, we deployed one instance of Open vSwitch (OVS) on each worker of the Kubernetes cluster in a Kubernetes pod, taking advantage of the container technology to simplify its deployment, set-up and failure recovery.

For the connection between containers and switches, the Multus CNI Plugin was used. This plugin allows the additional creation of virtual interfaces inside pods and their attachment to other network resources. In this case, it will create a virtual ethernet (Veth) interface, which acts as a traditional ethernet cable inside the host. This interface allows the pods to connect to the switch.

The interconnection of switches for the exchange of data plane traffic through IP Tunnels is already performed beforehand using Linux native VXLAN tunnels, since this proof of concept does not feature an overlay manager yet.

Finally, a RYU SDN controller (version 4.3) was installed in the Kubernetes controller node to modify the behaviour of the switches. By default, the switches do not have any kind of forwarding rules enabled (i.e., they act as traditional switches). Therefore, the SDN controller will run one of its default application that uses a simple Spanning Tree Protocol (STP) to module the links established through the VXLAN tunnels, avoiding potential loops in the network as well as providing mechanisms to counteract a link breakdown between the aircrafts, applying the shortest-distance strategy to deliver traffic in the network (although any protocol/strategy could be used, we selected this method to ensure the proper traffic delivery to the nodes) and ensuring that the network can find another path once a link is down.

B. *Functional validation*

To establish the viability of the solution, we performed several measurements to detect noticeable variations in terms of bandwidth consumption between one standard Kubernetes networking solution (Flannel) and our proposed design. In a previous work [15] we addressed the potential limitations that Flannel could have to provide direct connectivity between microservices. In this previous work, we created VXLAN interfaces in the pods establishing direct link layer connectivity on top of Flannel's IP connectivity. This forces the use of nested VXLANs, as Flannel uses VXLAN IP Tunneling for delivering the information between nodes. Due to its link layer approach, we included this model (double VXLAN) in our comparison.

For the validation, two batteries of tests were performed for the following networking approaches: pure-Flannel, Flannel with additional VXLANs and L2S-M. To test the impact of each solution with respect to the direct connectivity between nodes, we also included a vanilla approach in the comparisons. These

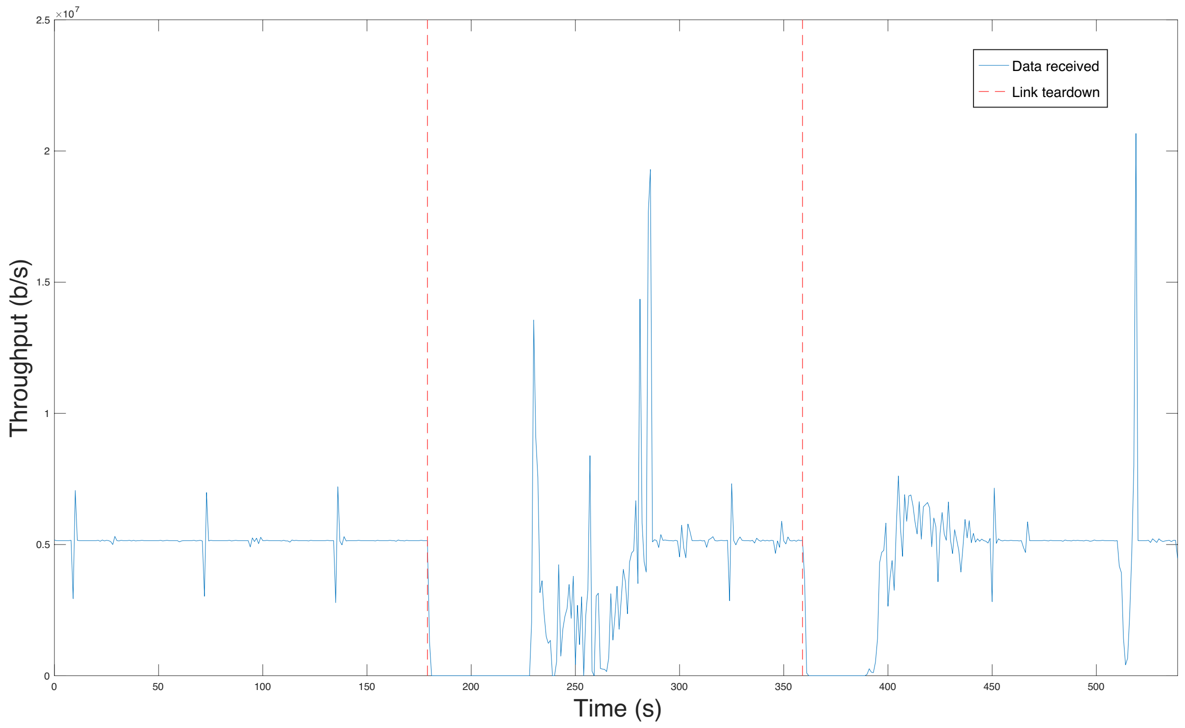

Figure 3: Throughput evolution during a link disruption over the network

tests were performed from UAV 1 to UAV 3 using the iperf3 traffic generation tool. In the case of vanilla, we directly used the hosts themselves to run the iperf application, while in the virtualisation solutions we used K8s pods instead. Two types of tests were used to obtain different measurements:

- Transmit a TCP flow at the maximum possible available rate to measure the average available bandwidth in Mb/s.
- Transmit a UDP flow of 5Mb/s to determine the average jitter in ms. In these tests, background UDP traffic was introduced to emulate incoming traffic (0 Mb/s, 5 Mb/s, 30 Mb/s and 60 Mb/s), using the same traffic solution for each one of the networking alternatives we compared. In other words, we generated background traffic at the client pod (host in the vanilla case) and sent it using the same networking solution used to send the 5 Mb/s flow (vanilla for the vanilla case, Flannel for the Flannel case, etc.).

All tests were performed for 60 seconds, repeated 20 times in a row. All measurements were performed in the laboratory premises offered by 5TONIC. The results for each test can be seen in Figure 2.

As it can be appreciated, our solution provides higher throughput measurements in comparison to the rest of the networking solutions, achieving the closest performance to the vanilla approach (i.e., without any virtualization solution). Regarding the observed jitter, all the solutions achieve similar values to the vanilla approach (with minor differences of maximum 0.3 milliseconds) According to these tests, L2S-M does not significantly impact the jitter metrics of the traffic flows, increasing performance in terms of throughput compared to the Flannel and the double VXLAN approaches.

Finally, in order to verify that our solution has the capacity to react against link failures, we emulated a link breakdown between UAV 1 and UAV 3 while a 5Mb/s data flow was being transmitted between the pods located at the aircrafts. In this case, we tear down the link due to a transient error over the link, lasting 180 seconds until the link is brought up again. Therefore, we force its traffic to go through the GCS once the link is down, checking that the SDN controller can properly react to this breakdown to deliver the traffic between workers. As it has been mentioned before, the RYU controller uses a simple STP algorithm to avoid loops in the network, so it will force the switches to build a new path through STP after the direct link has been turned offline.

In Figure 3, the provision of a video-streaming service with a 5Mb/s flow, cutting the direct link between the aircrafts after 180 seconds can be appreciated. As it can be spotted in the graph, after 30 s where the protocol is finding the new route, communication is re-established. 180 seconds after the failure, the link between the aircrafts is up again, in turn triggering the SDN controller to force the switches to run the STP again, which after a while the system recovers as it was initially shown. It is important to point out that these 30s could be significantly reduced in realistic scenarios by using more refined algorithms and/or other specific protocols, but this simple algorithm allows us to verify the capacity of our system to recover from sudden disruptions. It is relevant to point out that our system has the flexibility to execute different algorithms to recompute traffic paths dynamically for a variety of purposes, not only to react against unexpected disconnections.

V. Conclusion and Future Work

In this paper, we have presented L2S-M, a solution that offers link layer connectivity as a service to container interfaces managed by microservice platforms. It fulfils this objective by building a programmable data plane where microservices can establish Link layer connectivity with any other container. This networking solution can use a wide variety of algorithms to rebuild the traffic paths on demand in the programmable data plane to obey different necessities (isolation, traffic engineering, etc.) or react against sudden changes in the ad-hoc network.

This approach can potentially improve microservice connectivity for ad-hoc networks in comparison to other available networking services, as it can be seen in the first validation performed in the paper at the 5TONIC laboratories in collaboration with Telefónica.

Our future work will include the full realisation of the view presented in this paper by developing the networking service, as well as testing its implementation and performance over realistic scenarios of aerial ad-hoc networks. These scenarios will consider the utilization of different routing algorithms to create data paths among microservices over wireless ad-hoc networks.

VI. Acknowledgments

This article has been supported by the TRUE5G (PID2019-108713RB681) project funded by the Spanish National Research Agency (MCIN/AEI/10.13039/5011000110) and by the H2020 FISHY Project (Grant agreement ID: 952644).

References

[1] L. F. Gonzalez, I. Vidal, F. Valera, B. Nogales, V. Sanchez-Aguero, and D. R. Lopez, “Transport-Layer Limitations for NFV Orchestration in Resource-Constrained Aerial Networks,” *Sensors*, Nov. 2019., vol. 19, no. 23, p. 5220, doi: 10.3390/s19235220

[2] F. Wan, X. Wu and Q. Zhang, "Chain-Oriented Load Balancing in Microservice System," 2020 World Conference on Computing and Communication Technologies (WCCCT), 2020, pp. 10-14, doi: 10.1109/WCCCT49810.2020.9169996.

[3] R. Petrasch, "Model-based engineering for microservice architectures using Enterprise Integration Patterns for inter-service communication," In 14th International Joint Conference on Computer Science and Software Engineering, 2017, pp. 1-4, doi: 10.1109/JCSSE.2017.8025912.

[4] “Adopting Microservices at Netflix: Lessons for Architectural Design”. Accessed November 25, 2021. [Online] https://www.nginx.com/blog/microservices-at-netflix-architectural-best-practices/

[5] I. M. A. Jawarneh et al., "Container Orchestration Engines: A Thorough Functional and Performance Comparison," ICC 2019 - 2019 IEEE International Conference on Communications (ICC), 2019, pp. 1-6, doi: 10.1109/ICC.2019.8762053

[6] Docker Inc. “Swarm mode overview”. Accessed November 25, 2021 [Online]. https://docs.docker.com/engine/swarm/

[7] Open Infrastructure Foundation. “OpenStack: The most widely deployed open source software in the world”. Accessed November 25, 2021. [Online] https://www.openstack.org

[8] The Linux Foundation. “Kubernetes: Production-Grade container orchestration”. Accessed November 25, 2021. [Online] https://kubernetes.io

[9] David Bernstein. “Containers and cloud: From lxc to docker to kubernetes.” IEEE Cloud Computing, September 2014, vol. 1(3), pp. 81-89, , doi: 10.1109/ICC.2019.8762053.

[10] Red Hat Inc. “Manufacturing at the edge with Red Hat OpenShift”. Accessed November 25, 2021. [Online] https://www.openshift.com

[11] Peter Dalbhanjan. “Overview of Deployment Options on AWS”, Amazon Web Services Inc., 2015.
Accessed November 25, 2021. [Online] https://d0.awsstatic.com/whitepapers/overview-of-deployment-options-on-aws.pdf

[12] The Linux Foundation. “CNI: The Container Network Interface”. Accessed November 25, 2021. [Online] https://www.cni.dev

[13] L. F. Gonzalez, I. Vidal, F. Valera, V. Sanchez-Aguero, B. Nogales and D. R. Lopez, "NFV orchestration on intermittently available SUAV platforms: challenges and hurdles," IEEE INFOCOM 2019 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), 2019, pp. 301-306, doi: 10.1109/INFCOMW.2019.8845040.

[14] L. Gupta, R. Jain and G. Vaszkun, "Survey of Important Issues in UAV Communication Networks," in IEEE Communications Surveys & Tutorials, November 2015, vol. 18, no. 2, pp. 1123-1152, doi: 10.1109/COMST.2015.2495297.

[15] L. F. Gonzalez, I. Vidal, F. Valera, V. Sanchez-Agüero. “A Comparative Study of Virtual Infrastructure Management Solutions for UAV Networks”. In Proceedings of the 7th ACM Workshop on Micro Aerial Vehicle Networks, Systems, and Applications (DRONET), June 24, 2021, pp. 13-18, doi: 10.1145/3469259.3470486

Biographies

Luis F. Gonzalez is a Ph. D. candidate in Telematics Engineering at UC3M. He has been involved in several European and national research projects. His research interests include Network Functions Virtualization (NFV), 5G networking, and Unmanned aerial vehicles (UAVs), publishing in various international conferences and journals.

Ivan Vidal received the Ph.D. in Telematics Engineering In 2008 from the University Carlos III of Madrid, where he is currently working as visiting professor. His research interests include 5G networks, Network Functions Virtualization (NFV), Unmanned aerial vehicles (UAV), network security, and multimedia networking.

Prof. Dr. Francisco Valera received the Telecommunication Engineering degree in 1998 from the Technical University of Madrid (UPM), and the Ph.D. in Telecommunications in 2002 from the Univ. Carlos III de Madrid (UC3M), where he is currently a tenured associate professor and deputy head of the Telematics Engineering Department

Dr Diego Lopez is a Senior Technology Expert in Telefonica I+D, in charge of exploratory activities within the GCTIO Unit. Diego is focused on applied research in network infrastructures, specially on virtualization, data-driven management, new architectures, and security. He chairs ETSI ISG PDL and the NOC of ETSI ISG NFV.