Journal of Network and Systems Management manuscript No.
(will be inserted by the editor)

SDN-based Traffic Management Middleware for Spontaneous WMNs

Paolo Bellavista · Alessandro Dolci · Carlo Giannelli · Dmitrij David Padalino Montenero

Received: date / Accepted: date

Abstract Software Defined Networking (SDN), with its clear distinction of control and data planes, as well with its simple paradigm of logically centralized controller with global visibility of the whole targeted network status, is gaining momentum in different scenarios. However, its effective exploitation in Wireless Mesh Networks (WMNs) is still an open and emerging research topic, mainly due to the high dynamicity of some related deployment environments (e.g., spontaneous WMNs) and to the need of efficient solutions capable of locality-enhanced optimizations. Here we originally present motivations, challenges, design guidelines, and a prototype middleware implementation for SDN-based management of selected (most appropriate) traffic flows. Our solution advances the state of the art in the field by i) allowing high flexibility via deployment and de/activation of flow management policies at provisioning time, ii) supporting dynamicity via proper efficient handling node of join/leave events, and iii) increasing scalability via a partially decentralized approach based on dynamically determined WMN "islands", usually managed by separated SDN controllers that can seldom interact to federate their management decisions. In addition to design/implementation insights and to the availability of the prototype code, this paper provides the community with a significant novel contribution in terms of experimental performance results, which quantitatively demonstrate the feasibility and the effectiveness of the proposed approach.

Keywords Spontaneous WMNs · Management Middleware · Software Defined Networking · Overlay Networking · Inter-Flow Management · Quality Management

P. Bellavista
Department of Computer Science and Engineering, University of Bologna, Italy
E-mail: paolo.bellavista@unibo.it

A. Dolci
Department of Computer Science and Engineering, University of Bologna, Italy
E-mail: alessandro.dolci@studio.unibo.it

C. Giannelli
Department of Mathematics and Computer Science, University of Ferrara, Italy
E-mail: carlo.giannelli@unife.it

D. D. Padalino Montenero
Department of Computer Science and Engineering, University of Bologna, Italy
E-mail: dmitrij.padalino@unibo.it

1 Introduction

The availability of the upcoming 5G cellular standard with the already spread WiFi direct technology is paving the way for device-to-device (D2D) communication [1,2]. This emerging approach fosters the adoption of novel service provisioning scenarios that can benefit from the opportunity to consider and use interconnected (and possibly mobile) devices in the local vicinity as resource/service providers. This kind of approach originates and takes first inspiring ideas from traditional Wireless Mesh Network (WMN) solutions, which offered connectivity to final users in a large area by interconnecting multiple (usually fixed) infrastructured wireless routers, managed in a coordinated way to support client authentication/authorization and multi-hop routing. However, the involvement of heterogeneous and mobile devices, carried by users, within the network topology itself (and thus not only as clients getting connectivity) adds significant potential and new features, by enabling impromptu connectivity via spontaneous ad hoc networks. In spontaneous networking, there are no substantial differences in role between WMN routers and user nodes: both are in charge of dispatching packets along multi-hop paths. Such solution can be viewed as a further extension and evolution of traditional hybrid WMNs [3] because heterogeneous and mobile devices in spontaneous networks not only create their own multi-hop network, but also become part of the infrastructure network itself by acting as WMN routers, e.g., to inter-connect infrastructure WMN routers. Below, for the sake of simplicity and briefness, we will indicate these envisioned deployment scenarios with the term of *spontaneous WMN* [4,5], to stress the notable novel aspect of including end user devices in multi-hop network management, to the purpose of providing access to local networking, storage, and computation resources in a dynamic manner. Note that the efficient support and implementation of spontaneous WMNs are also considered key enabling factors for leveraging ad hoc and (partially) mobile cloud computing services and applications (enhanced by fog computing nodes, possibly available in a locality), which are expected to be spontaneously deployable, easy to access, cost effective, and not relying on central servers or connections to infrastructure networks [6].

Even if in their infancy, spontaneous WMNs have already demonstrated to be effective in supporting various types of applications, such as sharing Internet connectivity or sensor data, crowd computing, multimedia search/processing, language processing, social networking, and disaster recovery [7]. In current state-of-the-art spontaneous WMNs, cooperating nodes take management decisions based on their limited scope visibility and without a global knowledge of network topology/conditions, by typically reacting to modifications in resource availability in their own locality. In fact, also because of the non-organized collaborative nature of spontaneous WMNs, the related research has focused so far on most basic support features of packet dispatching at multi-hop distance, by neglecting the maximization of Quality of Service (QoS) provisioning with a per-application view.

In a wider perspective, we can observe that the fields of efficient application and optimization of QoS management are still largely unexplored in the spontaneous WMN research area. This is partially motivated by the fact that traditional traffic engineering solutions, based on strictly enforced resource allocation, are sometimes hard to adopt in spontaneous WMNs. In particular, the general-purpose nature and the limited resources available on end user mobile nodes make challenging to dynamically modify the behavior of nodes with the goal of improving the QoS of the whole network (e.g., because it can benefit from the adoption of a network-wide point of view). For instance, by considering research fields close to spontaneous WMN, [8] proposed a middleware for latency-controlled Mobile Adhoc Network (MANET) communications based on an adaptive approach that switches between the available connectivity opportunities. [9] focused on hybrid networks (spontaneous WMN nodes plus a wireless infrastructure), by modeling packet routing issues as resource scheduling problems.

In other words, dynamic and heterogeneous spontaneous WMNs tend to be managed in a (partially) distributed and disjoint manner: traditional WMN routers are configured by infrastructure network administrators, while mobile nodes offer their resources in a completely best effort way. Such a disjoint approach, without a clear coordination activity, prevents from adopting a unified network traffic management policy,

e.g., to pursue the common objective of maximizing the overall QoS. In particular, in the envisioned use-case, infrastructure WMN routers and mobile nodes share data about the network and computing resources that they are willing to share with others and also provide up-to-date information about their current state, e.g., in terms of traffic load and available memory. Based on these data it is possible to achieve a unified vision of the whole network, thus making easier the provisioning of the required QoS (see Section 3.1 for further details). To this purpose, in the envisioned use-case, also the applications collaborate to maximize the overall network QoS, e.g., by providing information about their forthcoming traffic flows and how such flows should be managed, e.g., in a time-critical manner or not.

This paper has the ambition to show that the Software Defined Networking (SDN) approach, which is emerging in several deployment and application scenarios, has the potential to properly fit also the dynamic and heterogeneous nature of spontaneous WMNs, with the goal of jointly adopting traffic rerouting and engineering features. Notably, adopting the SDN approach in spontaneous WMNs is more complex if compared with traditional WMNs for several motivations: devices are more heterogeneous in terms of hardware and software capabilities; security concerns deny full access to underlying operating system features (e.g., it is hard to have direct access to the possibility of modifying routing tables); and the topology dynamicity is much higher, e.g., nodes can freely join/leave the network or simply stop sharing networking/computing resources. Please also note that the volunteering nature of connectivity sharing in spontaneous WMNs brings to more complex topologies (if compared with traditional WMNs) consisting of multiple and independent IP subnets eventually managed by end user devices and adopting different IP addressing spaces.

On the one hand, given that spontaneous WMN nodes interact to offer and access services in a collaborative manner, there is no a priori knowledge of service availability in them. Thus, it is very effective to have a centralized point of view with full visibility, able to take proper control decisions. To this purpose, the logically centralized point of decision of the SDN Controller can play a relevant role in the field, in particular for wide-scale spontaneous WMNs (see the concept of WMN island in Section 3.2 and Section 4.2). On the other hand, spontaneous WMN nodes are willing to further cooperate to improve QoS, e.g., via synergic exploitation of all available networking opportunities, in many deployment scenarios of emerging relevance such as emergency scenarios or tactical networks. In fact, based on their limited visibility of the network, competing applications/nodes in a spontaneous WMN may use the same (apparently best) multi-hop paths, while erroneously neglecting alternative paths that are to be preferred because of minor load expected in the targeted service provisioning time window. In other words, we claim that the adoption of the SDN approach in spontaneous WMNs can gain deeper knowledge of the available topology and of its state, as well as it can consider application requirements to adapt packet dispatching accordingly, with significantly enhanced per-application QoS management decisions. The efficiency of such solutions, however, should carefully and properly consider the dynamicity of spontaneous WMNs, where nodes can move/join/exit/stop resource sharing in a hardly predictable way, by adequately maintaining the most suitable tradeoff between freshness of traffic status and monitoring intrusiveness.

Here, we significantly extend our previous conference papers [10,11] by proposing and detailing our original architecture and prototype that realize the above idea of WMN-oriented SDN. In addition, we provide a thorough analysis of an extensive set of performance results that quantitatively show the behavior of traffic flows while applying our proposed traffic management features, by also presenting the efficiency of our proposal in terms of computing load and latency. To this purpose, the paper originally presents new technical insights on how to efficiently apply the SDN approach to spontaneous WMNs, as well as the detailed description of the algorithms and architectural components that we have implemented and experimentally assessed on a working prototype. In particular, we propose a middleware where spontaneous WMN QoS management can benefit from the flexibility given by the SDN separation of control/data planes and from the global awareness associated with the logically centralized SDN Controller. In our middleware, Control Agents (CAs), deployed on participating nodes of the targeted spontaneous WMN, interact with their SDN Controller to create the needed visibility of a WMN island status. CAs serve as both agents

towards the remote SDN Controller (by sending node status to their controller and by receiving control commands from it) and agents towards the local data plane (by tuning forwarding rules for traversing packets based on flow IDs, by considering the currently enforced flow rules commanded by the controller through the control plane).

In addition, the paper originally concentrates on the specific challenges of adopting the SDN approach in spontaneous WMNs, by considering WMN peculiarities in terms of dynamic and relatively limited resource availability of participating nodes (at least if compared with intra-datacenter SDN exploitation, which is the most considered deployment scenario in the current SDN literature). To this purpose, our original solution a) allows high flexibility via deployment of new flow management policies at provisioning time, b) properly handles node join/leave events to reduce the associated overhead, in particular in wide-scale deployment environments, and c) identifies WMN islands, usually managed by separated SDN Controllers, that can seldom interact in our framework in order to federate their management decisions.

We strongly believe that the paper can provide the readers with a significant and original contribution about this emerging topic if compared with the existing literature, under several perspectives and for several motivations. First, we address the technical challenges of SDN application to spontaneous WMNs by proposing a general-purpose framework that can be used in heterogeneous contexts and application domains. In fact, our middleware solution can be deployed and activated in a wide range of environments, ranging from Linux/Windows desktop operating systems to mobile Android platforms. In addition, developers can implement their own applications by taking advantage of packet dispatching and QoS management features provided by our middleware. Second, we originally propose a joint solution for the dynamic management of both routing and traffic engineering policies: the proposed policies are novel for spontaneous WMNs and can be also re-used in other frameworks. Third, the flexibility of the proposed architecture allows interested researchers in the community to easily extend it, e.g., by supporting the definition of new routing and traffic engineering policies and their deployment/activation at runtime. Finally, in addition to design/implementation insights and to the availability of the prototype code of the framework (made accessible to the community for possible refinement and extension), this paper originally reports an extensive set of experimental measurements of primary performance indicators over a real deployment testbed; the reported results quantitatively assess and validate the feasibility, effectiveness, and efficiency of the proposed approach for SDN-based QoS management in spontaneous WMNs.

2 Related Work

At first, SDN has emerged primarily to manage switches of closed and geographically centralized environments such as datacenters and department networks via the OpenFlow protocol [12], the de-facto standard supported by networking industrial-grade devices. In particular, the centralized nature of the SDN approach makes it the natural choice for managing networks of small-to-medium size related to a single organization. Moreover, by relaxing requirements in terms of closeness and geographical centralization, SDN is exploited in wide area networks, e.g., to efficiently interconnect different datacenters [13, 14] eventually based on a multi-controller SDN architecture [15, 16]. Readers interested in an in-depth analysis of SDN-related state of the art literature are invited to refer to [17].

By specifically considering wireless environments, [18] proposes to extend the SDN approach towards a hybrid architecture, with a centralized SDN Controller gathering and pre-processing information and several distributed nodes (typically base stations providing connectivity) performing decision-making and configuring the data plane of mobile nodes. The SDN approach is also adopted in access/transport networks, e.g., to dynamically reroute traffic in case of bad weather potentially degrading the performance of wireless network infrastructures [19]. Other solutions propose to adopt the SDN paradigm in wireless environments to address specific features or specific application requirements. For instance, [20] exploits the SDN paradigm to improve mobility management of nodes in infrastructure environments by adopting

a centralized component in charge of managing the identity-location relationship of nodes and spreading such information to distributed Radio Access Networks (RANs). [21] exploits a centralized SDN Controller to improve the overall QoS of a campus wireless infrastructure by optimally splitting mobile nodes in WiFi Direct clusters. [22] dynamically reroutes traffic flows among close mobile nodes connected to the same middle-level gateway to avoid that packets have to go back and forth through the core of the network. Finally, [23] deploys a CA on mobile nodes associated with infrastructure IEEE 802.11 Access Points (APs) to allow their remote management, e.g., to more easily support access control and to improve the efficient and fair exploitation of the shared wireless medium. A more detailed comparison of solutions adopting the SDN approach in wireless environments can be found in [24].

By focusing on more traditional WMNs based on fixed routers, [25] adopts the SDN approach in multi-hop wireless networks to more efficiently identify shortest paths and disjoint multiple paths between senders and receivers. To this purpose, the SDN controller exploits its global point of view to provide single-path and multi-path routes, by identifying the best path based on energy and hop count. [26] adopts SDN in WMNs to perform traffic engineering, as proposed in our paper. However, the primary goal there is to easily identify congested paths and then to reroute affected traffic flows towards alternative non-congested paths, without any consideration about traffic prioritization. Similarly, [27] focuses on traffic engineering for WMNs by taking into consideration some primary characteristics of wireless networks, such as the availability of multi-channel and multi-radio carriers. The proposed heuristic algorithm is able to find a suboptimal solution for AP and route selection in polynomial time. The solution ensures that every active flow gets at least a minimum bandwidth while maximizing the overall aggregated throughput and while minimizing the number of gateways to deploy.

[28] presents the performance achieved with OpenFlow in a multi-interface WMN testbed. The main goal is to stress how heterogeneous mesh routers can affect OpenFlow performance in single- and multi-channel scenarios. To this purpose, the paper compares OpenFlow with BATMAN, AODV, and HWMP routing protocols. As the most notable achievement, the paper demonstrates that OpenFlow achieves better performance in multi-channel environments with homogeneous routers, even if at the cost of additional overhead in case of multiple flows (the number of requests to the SDN controller depends on the number of traffic flows). [29] specifically focuses on the issues related to SDN controller availability in case the network is temporarily partitioned in multiple subnets and then merged again due to intermittent connectivity. The main goal is that each router promptly selects one SDN controller if the previous one is not available anymore. The proposed solution is performed by each router separately by adopting a "master selection" mechanism rather than a "master election" one, with routers that adopt a simple control logic oriented to the selection of an SDN controller within a hierarchy of SDN controllers. [30] further investigates issues related to partitioning of WMNs by adopting an opportunistic and self-configurable approach, the former to better consider that the network may have intermittent connectivity, the latter to allow the dynamic redirection of requests towards the most suitable SDN controller. To this purpose, routers can autonomously discover and select available SDN controllers, while in case of SDN controller unavailability routers can coordinate themselves to set up a new controller. [31] presents an integrated solution to emulate IEEE 802.11s SDN-enabled WMN scenarios by jointly exploiting the NS-3 simulator, Linux network namespaces, and the Open vSwitch multilayer virtual switch. The paper demonstrates that the adoption of the SDN approach can relevantly improve the achieved throughput within WMN networks, in particular when supporting multimedia streams.

Finally, let us note that spontaneous WMNs share some significant aspects and technical challenges with MANETs: for example, in both environments nodes can freely join and leave the network (while routers of traditional WMNs are added/removed/relocated seldomly); or nodes that dispatch traffic flows along multi-hop paths can be heterogeneous in terms of hardware capabilities (while routers of traditional WMNs tend to have similar capabilities). Traditionally, the dynamicity and heterogeneity of MANETs have limited the adoption of the SDN in such environments and only recently a few contributions have started to emerge. For instance, [32] 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. [33] adopts the adaptive principles of the SDN approach to seamlessly recover from disruptions of computation-intensive MANETs. Similarly, [34] 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. [35] outlines an initial research work effort to adopt SDN in adhoc networks on-top-of smartphones, by briefly sketching its proposed components and supported features.

In short, to the best of our knowledge, our work is the first one proposing and experimenting design/implementation guidelines for scalable and federated SDN Controllers in a wide-scale and geographically distributed scenario of spontaneous WMNs, in particular of spontaneous WMNs composed of loosely-coupled and interworking WMN islands. 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 QoS management in spontaneous WMNs.

3 Motivations and Challenges for SDN-based Quality Management in WMNs

3.1 Problem statement and motivations for SDN in WMNs

We state that existing state-of-the-art solutions for spontaneous WMN management present three primary shortcomings: purely distributed approach based on only local visibility, management policies only defined at system design time, and limited capabilities in considering (possibly conflicting) application QoS requirements. By delving into finer details, we model the addressed issued along three primary directions, based on three different points of views:

1. **from the network monitoring point of view, limited capabilities of unified network visibility**: traditionally, spontaneous WMN management is based on a purely distributed approach exploiting local interaction of neighbor nodes [36], with the main purpose of supporting inter-node single-hop connectivity and multi-hop packet dispatching. This approach has made possible to reduce the imposed overhead, but at the cost of potentially adopting sub-optimal management decisions due to the limited knowledge of the overall environment and its global state;
2. **from the network control point of view, lack of supporting time-varying network management objectives.** Traffic management policies are usually identified at system design time by taking into consideration a specific objective, e.g., to support the timely delivery of huge messages or the efficient dispatching of multimedia streams to multiple receivers. Sometimes traffic management policies are difficult/expensive to be modified after node deployment and can hardly evolve depending on changing requirements at service provisioning time (e.g., specialized WMNs with application-specific and static QoS optimization);
3. **from the application point of view, lack of fulfillment/management of eventually conflicting application requirements**: also due to the distributed nature of WMNs, related QoS solutions do not consider possibly conflicting QoS requirements of different applications running on-top-of different nodes. For instance, multiple nodes could simultaneously exploit the same (apparently) large-bandwidth path to transmit packets, incurring in inter-flow bandwidth competition that can relevantly reduce the actual throughput.

Based on these considerations, the paper originally proposes to adopt the SDN approach in WMNs, and in particular in its challenging sub-class of spontaneous WMNs, to support QoS differentiation and the delivery of enhanced services, by taking into account also application-level and dynamically changing requirements. We claim that spontaneous WMN traffic management can relevantly benefit from the adoption of the SDN approach thanks to the following primary characteristics:

- centralized point of view allowing to take better resource management decisions by exploiting:
  - full knowledge about network topology and its state to take proper information-based decisions;

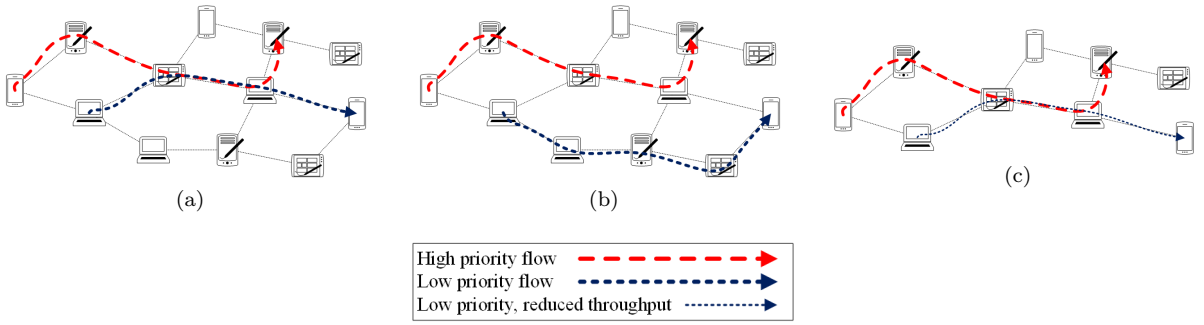


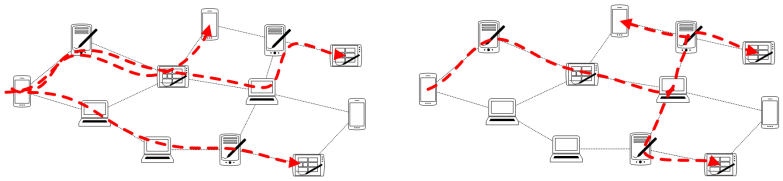
Fig. 1 Interfering flows: (a) without SDN, (b) with alternative path, and (c) with reduced throughput lower-priority flow.

  - visibility of application requirements, which can be provided by nodes prior to actual traffic generation and exploited to reduce conflicting traffic;
- enforcement of WMN-wide traffic management and QoS policies by supporting:
  - routing policies, to dynamically modify the path(s) that packets traverse from senders to receivers, e.g., to avoid congested links;
  - traffic engineering policies, to tune the amount of resources that traffic flows can exploit, e.g., by ensuring larger bandwidth to high-priority services.

Furthermore, the adoption of the SDN approach allows to flexibly switch among different traffic management policies in a dynamic and seamless manner, even supporting the deployment of new policies at service provisioning time. In this manner it is possible to provide general-purpose middleware solutions able to adapt and enhance network traffic management capabilities of WMNs by integrating and enabling if and when strictly required new ad-hoc software components designed to achieve specific goals, e.g., providing larger bandwidth to high priority traffic flows or minimizing traffic overhead of multimedia streams.

Let us note that adopting SDN in WMNs represents a notable step forward to the state-of-the-art literature, since it allows to efficiently maximize the QoS considering not only the state of the network and the specific characteristics (and requirements) of each traffic flow, but also to jointly consider characteristics (and requirements) of different flows dispatched towards the WMN at the same time. Furthermore, it also potentially extends the WMN (re)usability and lifespan, by allowing to dynamically evolve it based on changing objectives/requirements and thus increasing its usefulness in different heterogeneous environments.

To better explain how SDN adoption can support QoS management in spontaneous WMNs, let us consider two possible examples representing notable use cases: two huge traffic flows with different priority levels competing for the same path (see Figure 1) and a live multimedia stream sent to multiple nodes (see Figure 2). In the former case, two flows are transmitted towards the same sub-path. Without the enforcement of a suitable traffic management policy, even if one flow has higher priority, the two flows tend to interfere (and thus slow down) one another (Figure 1a). Instead, with SDN, it is possible to exploit the centralized and global point of view of the SDN Controller to either look for an alternative (eventually with limited quality) path for the lower priority flow (Figure 1b) or (if an alternative path is not available) inform intermediate nodes that they should temporarily reduce the throughput of the low priority flow to offer additional networking resources to the other one (Figure 1c). In the latter case, without the enforcement of a proper traffic management policy, the sender generates a distinguished traffic flow for each receiver, even if the flows share part of the same path (Figure 2a). Instead, it is possible to exploit the capabilities of the SDN Controller to identify a spanning tree and to require nodes in that spanning tree to duplicate the multimedia stream only if and where required, i.e., at roots of the interested sub-trees of the application-level multicast distribution tree (Figure 2b).



(a) (b)

Fig. 2 Multimedia flows sent to multiple receivers: (a) without SDN-enabled multicast, (b) with SDN-enabled multicast.

3.2 Open Technical Challenges

Notwithstanding the advantages pointed out in the previous section, the adoption of the SDN approach in highly dynamic WMNs such as spontaneous ones has peculiar aspects and technical challenges to carefully consider for proper specialization, in order to achieve a usable, industry-ready, and efficient solution. First of all, the centralized nature of the SDN approach does not fit with the typical (and well accepted) distributed management of spontaneous WMNs, since the latter provides higher degree of scalability and self-management. However, we claim that it is possible to combine the higher degree of control that a centralized approach supports with the higher scalability and easier self-management of a distributed one, via the proper adoption of a **partially decentralized solution**. In particular, we propose to partition large spontaneous WMNs into multiple (so-called) WMN islands, each of them controlled by a single and almost autonomous SDN Controller, capable of local autonomous operations but also of lazy interaction with other federated SDN Controllers. This direction of solution opens up novel technical challenges, not yet solved in a satisfying way in the related literature, such as how to define the size of a WMN island, how to split/merge WMN islands in case they become too large/small, and how to support the dynamic interaction among adjacent SDN Controllers, to mention the primary ones.

Furthermore, SDN solutions for spontaneous WMNs must carefully consider the need to provide **high flexibility**. In fact, it can be required to frequently tune/modify traffic and QoS management policies, as well as to deploy novel policies at service provisioning time. Primary considerations relate to a flexible approach to easily de/activate and introduce new management policies at runtime, with the relevant overall objective of providing a general-purpose middleware solution whose behavior can be easily adapted in relation to the specific (and eventually time-varying) goals of WMNs.

Finally, SDN solutions for spontaneous WMNs have also to support a **high degree of dynamicity**, since spontaneous WMN nodes can join and leave their localized network frequently and the SDN Controller should be promptly but lightweightly informed about the currently available resources and network links. To this purpose, it is required to identify a meaningful set of monitoring data (and their refreshing constraints) allowing to adequately assess the current state of the WMN and the degree of satisfaction of the targeted application requirements. Moreover, it is required to deal also with the WMN dynamicity caused by the possibly frequent join/leave of nodes due to their physical mobility, via lightweight protocols that are adequately scalable.

4 Our Middleware Architecture for SDN-based Management of Spontaneous WMNs

Figure 3 presents the high level architecture of the proposed middleware, adopting the traditional SDN division between the Control Plane configuring and tuning traffic management mechanisms and the Data Plane actually managing traffic flows based on the information provided by the Control Plane.

By considering its primary macro-components, the Control Plane Manager has multifold objectives:

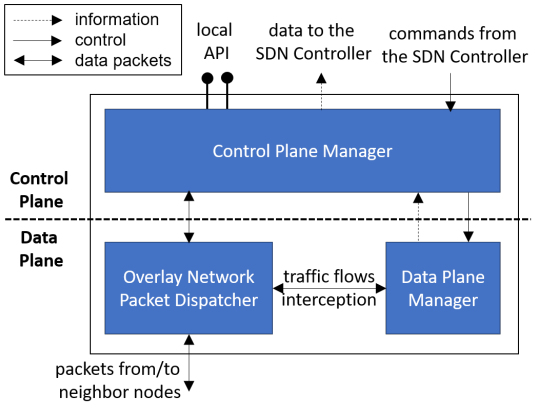


Fig. 3 High-level representation of SDN node main components.

1. it interacts with the remote SDN Controller to dispatch local state information and receive commands;
2. it manages the underlying data plane to appropriately enforce the policies provided by the SDN controller;
3. it provides Application Program Interface (API) for local applications interested in taking advantage of SDN capabilities, e.g., to send a traffic flow at high priority or to send the same traffic flow to multiple receivers. Let us note that the Control Plane Manager supports these API by interacting with the remote SDN Controller to notify local application requirements, the latter also appropriately informing other nodes, e.g., a traffic flow has higher priority or has to be sent to multiple nodes.

The Data Plane Manager actually enforces traffic policies by intercepting traversing traffic and appropriately managing packets, e.g., by delaying the transmission of low priority packets to benefit high priority ones. Note that since we target spontaneous WMNs, packet dispatching should be performed by taking into consideration the specific nature of this class of WMNs, e.g., nodes often belong to different IP subnets with possibly clashing IP addresses with only local validity. To this purpose there is the need of an Overlay Network Packet Dispatcher component supporting overlay networking, i.e., dispatching packets independently from how underlying (possibly heterogeneous) links/IP sub-networks have been autonomously and independently created. The Data Plane Manager receives information from the Control Plane Manager, appropriately manages traversing traffic flows, and exploits the Overlay Network Packet Dispatcher to communicate with the SDN controller; note that the SDN Controller runs on a regular WMN node, thus requiring to exploit the overlay network to send/receive packets to/from other WMN nodes.

In addition, in our architecture, for each WMN island, we assume that there is one WMN node that takes the role of SDN Controller, e.g., by adopting a distributed protocol for node selection based on computing capabilities and (for mobile nodes) on energy availability, as well as incentive mechanisms, taken from the existing related literature (not described here because they are out of the primary scope of the paper). Note that such protocols and incentives have been already proposed and are considered relatively traditional in contiguous research fields, such as MANETs, in order to support cluster head selection and to foster collaboration [37]. Due to the dynamicity of spontaneous WMNs, WMN nodes have to dynamically discover the location of available SDN Controllers and also to periodically repeat the same procedure to verify that the current SDN Controller is still available and/or to identify alternative better SDN Controllers, e.g., because closer. In addition, nearby nodes can cooperate to select a new SDN Controller, e.g., since the current one has left a limited battery lifetime, by gracefully migrating the set of managed nodes and other state information to a node, e.g., with the charger plugged in, acting as the new SDN Controller. Finally, the SDN Controller has the duty of discovering nearby SDN Controllers that manage its adjacent WMN

islands and of interacting with them in a loosely-coupled way to lazily coordinate some aspects of QoS management for inter-island traffic flows.

Let us note that the dynamic and possibly intermittent nature of spontaneous WMN connectivity may limit the reliability of control message dispatching. However, the adoption of reactive rerouting mechanisms in case of path disruption [5] together with opportunistic networking solutions [38] for delay-tolerant messages, e.g., to deploy a new traffic engineering policy, can relevantly increase the resiliency of the control plane. In addition, by providing maximum priority to control plane messages it is possible to grant their prompt delivery even in case of traffic congestion, by briefly delaying data plane traffic flows. In this manner we can ensure reduced latency even in the challenging case of control and data plane messages sharing and competing for the same networking resources (of course, better performance can be achieved in the simpler scenarios with control and data plane exploiting different networks and thus not interfering one another). In addition, scalability can be further increased by adopting a tree-based control packet dispatching schema, e.g., the one presented in [39].

4.1 Monitoring Features of our Middleware

As better detailed in Section 4.2, in our middleware a wide-scale and geographically distributed WMN can be managed by a few interconnected SDN Controllers, while each SDN Controller is in charge of managing its part of the network in an almost autonomous way. To support the SDN Controller in the decision process, it is required to collect status information about its specific part of the network. Let us stress again that spontaneous WMNs are much more dynamic and sparse if compared with networks in datacenters. In fact, spontaneous WMNs consist of devices carried by users, eventually moving close to other devices (thus allowing to join an available spontaneous WMN island or create a new one) or moving away in an abrupt manner. In addition, devices in spontaneous WMNs are much more heterogeneous in relation to CPU/RAM capabilities, battery lifespan, available sensors, installed operating system, and related libraries/software.

To this purpose, each node (via its CA) periodically provides two kinds of information it can directly observe, either *static information*, ranging from outgoing links identified by locally valid IPs to characteristics/size of node CPU and RAM, or *dynamic information*, ranging from current exploitation of networking resources such as actual throughput of each outgoing link to current node conditions, e.g., its average CPU consumption and available RAM. We will see in our proposed architecture that static and dynamic information allows the SDN Controller to achieve a weighted graph representation of the WMN topology, which also provides information about its current state.

By delving into finer details, our middleware collects

- network statistics, for each interface of each node, including:
  - nominal bandwidth;
  - sent/received bytes from node startup;
  - sent/received data packets from node startup;
  - transmission errors from node startup;
- hardware statistics, for each node, comprising:
  - number and characteristics of physical/logical CPUs;
  - average system computational load in the last minute;
  - CPU activity percentage in the last minute;
  - total and available memory.

In addition, applications running on-top-of spontaneous WMN nodes should provide forthcoming requirements, by roughly estimating how and how much they intend to exploit the network in the future. In particular, we envision that application requirements can be provided in a simple but sufficiently meaningful way based on five primary elements that we have selected to describe the expected future traffic flows:

- *who*, the receiver nodes identified by their unique IDs, one ID for unicast traffic flows, multiple IDs for multicast traffic flows;
- *why*, the type of content such as multimedia stream or file transfer;
- *what*, an estimation of the amount of traffic, e.g., in kbit/s for a multimedia stream or kbit for a file transfer;
- *when*, in how many seconds the sender intends to start the traffic flow;
- *how long*, the time a multimedia stream will last or a (soft) deadline by when the file should reach the destination.

Let us note that status monitoring should be designed to have limited impact and low intrusiveness. In fact, in our solution, each node provides its current local conditions once every 20s (anyway configurable in order to comply with different deployment scenarios), by sending a message payload of about 3000 bytes, thus incurring in an average overhead of about 150 byte/s. In addition, if a node does not update its state for more than 60s (again default but configurable value), the SDN Controller assumes that the node has left the network and updates the maintained topology.

More sophisticated and detailed information can be identified, gathered, and provided to the SDN Controller by simply extending/modifying the already implemented message exchanges of our current middleware prototype. However, we believe that the above elements already represent a good tradeoff between functionality coverage and costs. In fact, the dynamic and inherently best-effort nature of spontaneous WMNs prevents from the adoption of a much wider set of data (as it is appropriately done in datacenters), by enabling anyway the most relevant QoS management support functions that are needed in this context and that are described below.

4.2 WMN Island Management Features of our Middleware

As already anticipated, adopting the SDN approach in spontaneous WMNs has to carefully consider the importance of providing a solution that can be dynamic and scalable.

To improve dynamicity, since nodes can freely and abruptly join/leave the WMN very frequently, there is the need of an efficient solution to easily register (and unregister) nodes to the SDN Controller. To this purpose, each CA dynamically discovers available SDN Controllers and joins the best one, adopting a proper metric. For instance, a CA can select the SDN Controller with minimum Round Trip Time (RTT) to optimize the communication latency or the closest SDN Controller in terms of hop number to reduce the overhead imposed on intermediary nodes to dispatch monitoring/control packets. Despite the adopted metric, in the current version of our protoyped solution, islands stem from the interaction of nodes and available SDN Controllers, without any coordination among nearby controllers. In the future we plan to further investigate this aspect to support coordination among SDN Controllers, e.g., to ensure load balancing by splitting the managed network into islands with the same number of served nodes. After the registration procedure, each CA periodically refreshes its registration to the selected SDN Controller, which removes a node if it does not receive a refresh message for a given time period. In addition, each CA periodically looks for other SDN Controllers that could have joined the network and it possibly joins a new SDN Controller by leaving the previous one.

To improve scalability and availability, multiple distributed SDN Controllers may be employed, thus adopting a partially decentralized approach [40]. In fact, the spontaneous and intermittent nature of WMN connectivity (in particular if compared with traditional datacenters) prevents from the adoption of a unique centralized SDN Controller in charge of monitoring and managing hundreds or thousands of nodes. Each distributed SDN Controller has the primary goal of enforcing QoS in its *WMN island*, i.e., a subset of nodes residing at relatively small multi-hop distance and sharing common interests. In other words, in the same large WMN, multiple distributed and independent SDN Controllers could be in charge of managing different WMN islands (thus, with a relevant difference if compared with ONOS [41], where a logically

centralized SDN Controller is distributed mainly to increase scalability and fault tolerance). Let us note that we deem as unlikely that spontaneous WMN nodes exploit very long paths, e.g., consisting of tens of hops. In fact, interworking nodes define the borders of their WMN islands themselves, in a dynamic and application-dependent way, based on their needed service interactions. For instance, this is the case of a WMN island composed of a tourists group moving from place to place, one individual close to the other (group mobility): in general, the set of nodes joining a WMN island is often and naturally identified by the interaction of people sharing the same social activity. However, there are cases in which the borders of a WMN island cannot be only defined by users' behavior or in which such borders would be too wide, e.g., thousands of people in a stadium, thus requiring to dynamically split one WMN island into multiple ones. Note that algorithms for dynamic splitting/merging of WMN islands are out of the scope of this paper; we plan to address them in our future work, as better detailed in Section 6.

Finally, as a secondary goal if compared with intra-island traffic management, distributed SDN Controllers of neighbor islands may coordinate one another for the QoS management of also inter-island traffic flows. In fact, a generally small subset of peripheral nodes at single-hop distance of one another may reside in different contiguous WMN islands. In this case, it is possible to either relax the enforcement of application requirements by managing each WMN island separately or take proper management decisions by considering also possibly conflicting requirements. In the former case, distributed SDN Controllers may adopt under-optimized control decisions, worsening the QoS as much as nodes residing in different WMN islands generate traffic through the same nodes/paths. In the latter case, distributed SDN Controllers of contiguous WMN islands should interact to share information and coordinate each other to take proper decisions from an inter-island point of view, imposing costs due to additional management complexity, traffic overhead, and delay in the decision-making process.

4.3 Traffic Management Features of our Middleware

Our traffic management policies specify how different traffic flows should be managed in a differentiated way to achieve a given goal, thus ruling and tuning how the data plane of each node should exploit the underlying packet forwarding mechanisms. For example, our middleware is able to modify the route towards the destination(s) with so called *routing policies* (see Section 4.3.1) or to manage competing traffic flows in a differentiated manner with so called *traffic engineering policies* (see Section 4.3.2).

These traffic management policies, when adopted in WMN environments, should fit the WMN characteristics and objectives: since features and goals can vary in relation to the WMN (and the same WMN could vary its objectives during its lifetime), there is the need of not only de/activating available policies at runtime, but also distributing and deploying novel policies after design and deployment of the platform/middleware management solution. Thus, while it is not possible to statically provide an exhaustive set of traffic management policies for any possible deployment/application scenario, we have anyway identified, implemented, and experimentally tested some notable policies that we believe can satisfy a large set of common objectives. In addition, our middleware also supports the runtime selection and even deployment of traffic management policies, by dynamically distributing novel software components at service provisioning on SDN Controller as well as on CAs.

Finally, let us note that only traffic flows tagged with a flow ID are managed by our middleware to enforce a target QoS level, thus applying routing and traffic engineering policies. On the contrary, any other traffic flow is managed via more traditional networking mechanisms and competes for networking capabilities in a best-effort way, e.g., without dynamic re-routing or per-flow traffic prioritization.

4.3.1 Routing Policies

Our routing policies aim at selecting the best paths from a sender to one ore more receivers, eventually modifying the path of a flow at service provisioning time and/or splitting/duplicating the same flow towards multiple paths for performance/reliability purposes. The routing policies currently available in our middleware implementation are Best Path (BP) and Tree-based Multicast (TM).

Let us note that the BP policy involves the control and data plane of sender nodes only, while the TM policy involves the control and data plane of every node of the spanning tree identified by the SDN Controller, i.e., sender nodes and intermediary nodes in the path between senders and receivers. In both cases the routing policy is activated on a per-sender basis, i.e., different senders in the same WMN (or even the same sender) can exploit different routing policies at the same time, with the maximum granularity of one routing policy for each traffic flow. For this reason the adoption of multiple routing policies can be done based on a silos approach, without any need of considering the (eventually) conflicting requirements of different applications running at the same time in the same WMN island.

To exploit the BP routing policy:

- our middleware-supported applications specify to the local CA their requirements (see Section 4.1) together with the path selection mechanism that the SDN Controller should adopt (see below);
- the local CA sends a request message to the SDN Controller;
- the SDN Controller applies the specified path selection mechanism to identify the best path from the sender to the destination based on its global knowledge of the topology and its current status. Then, it replies to the CA by providing the best route together with a unique flow ID;
- the CA configures the local data plane to exploit the given route for packets labeled with the given flow ID;
- the CA provides the flow ID to the application;
- the application starts sending its traffic labeling related packets with the given flow ID;
- the underlying data plane modifies the route of packets in relation to the information received by the SDN Controller.

Thus, the SDN Controller can interact with the CA also to push a novel path for already active traffic flows and flows are rerouted towards the (eventually) new path as soon as the CA receives the reply from the SDN Controller in a completely transparent manner from the application point of view, by simply sending packets tagged by the provided flow ID.

The path selection mechanism specified by applications dictates how the SDN Controller computes the best path. Currently, our solution supports three different mechanisms: a) breadth first, by providing the first path identified by applying the breadth first search algorithm on the topology graph, b) fewest intersections, by identifying the path towards the destination with the minimum amount of active traffic flows, and c) minimum load, by selecting the path with links currently dispatching less packets. In addition, our middleware has been designed modularly to make it easy to extend the SDN Controller by adding more sophisticated path selection mechanisms.

The TM procedure is similar to the BP one, but with a few notable differences. First of all, applications can provide two or more destinations and the SDN Controller computes the best spanning tree from the sender to destination nodes. Secondly, the SDN Controller sends a control message to each node of the identified spanning tree to provide information about the next nodes the traffic should be forwarded to. In other words, each node of the spanning tree receives a control message requiring to send packets labeled with a given flow ID to one single-hop distant node (or more than one in case of flow duplication). Finally, the SDN Controller provides to the sender node the flow ID that the application has to exploit to enable the selected TM policy; this is done only after it has already sent control messages to the spanning tree nodes.

4.3.2 Traffic Engineering Policies

Traffic engineering policies aim at improving the QoS of competing traffic flows considering, e.g., the current network load, application-level requirements, and the relative priority of traffic flows competing for the same network resources. In particular, applications provide their requirements to the SDN Controller, which replies with a flow ID that applications have to label their generated traffic with. In addition, based on application requirements, the status of the network, and the knowledge about currently active flows, the SDN Controller also associates the traffic flow with a suitable priority level and informs nodes about flow ID-priority level mappings. For instance, typically real-time multimedia applications (such as teleconference services) have higher priority than on-demand multimedia applications, which have higher priority than bulk transfers of large files.

Note that traffic engineering policies do not modify the route of traffic flows, but only tune how packets of different traffic flows are managed in case they compete for the same (scarce) resource, e.g., very often the bandwidth of a network interface. By delving into finer details, our solution actually activates traffic engineering policies only if there are multiple interfering flows, i.e., packets labeled with flow IDs associated with different priority levels are exploiting the same outgoing link. For this reason, for each priority level, the data plane takes trace of the time of the last five sent packets (the number of packets can be tuned, but higher values increase the imposed overhead) and computes the related Average Inter-Packet (AIP) time. Then, whenever packets should be transmitted, the channel is considered as busy (and thus the transmission delayed of $D = max[10ms, AIP]$) in the case a time period T related to higher priority traffic flows is not expired, where $T = max[25ms, AIP * 1.25]$ (10/25ms thresholds are configurable values adopted to always ensure a slight delay to lower priority packets if high priority packets have been dispatched recently).

In addition, while routing policies are selected with a per-flow granularity, traffic engineering policies are activated with a WMN island scope. In fact, each WMN island can activate only one traffic engineering policy at a time, decided and imposed by the SDN Controller globally. In case the SDN Controller requires to modify the enforced traffic engineering policy (eventually also involving the dynamic deployment of a new traffic engineering policy), it sends a proper command to the CAs of nodes in the WMN island with the name of the policy that they should activate (together with the policy implementation code in case it is a newly deployed policy). Of course, some of the nodes will receive the command prior to other ones, temporarily incurring in an inconsistent state where (for a short time period) some nodes of the WMN island still enforce the previous policy and some others already work with the new one. In addition, this situation may potentially worsen if a few nodes in the WMN island do not receive the command, e.g., due to packet loss. However, even in these cases, our middleware can quickly achieve consistent state by letting CAs to periodically retrieve, e.g., once every 20s in our current prototype implementation, information about the currently enabled traffic engineering policy from the SDN Controller managing the WMN island. In this way it is possible to ensure a satisfactory level of consistency while imposing limited overhead, at the cost of allowing short transition periods. On the contrary, due to the distributed nature of spontaneous WMNs, we deem less appropriate the adoption of solutions that guarantee strict requirement compliance in terms of synchronization and message delivery. In addition, it is important to consider that the modification of the enforced traffic networking policy should be done only if it is required to modify the goal of the whole WMN island, a case that is much less frequent (several orders of magnitude) than the 20s maximum period of state inconsistency.

In the following we present an overview of already supported and experimentally validated traffic engineering policies that are currently available in our prototype implementation, i.e., Single Flow with Single Priority (SF-SP), Multiple Flows with Single Priority (MF-SP), and Multiple Flows with Multiple Priorities (MF-MP).

Single Flow with Single Priority (SF-SP). The SF-SP policy considers only one level of priority, i.e., a flow is either preemptive or managed in a best-effort manner. Packets of flows at the high-priority level are always dispatched as fast as possible. On the contrary, packets related to flows at the no-priority level

are immediately dispatched only if there are no high-priority packets currently exploiting the same outgoing link, by exploiting the AIP-based mechanism presented above. In case the outgoing link is occupied by high-priority flows, the transmission packet is delayed for a short interval and then another transmission attempt is performed, thus preventing its transmission as long as high-priority packets are transmitted. Note that in case multiple flows at high-priority traverse the same node, their packets are always transmitted as fast as possible, thus competing for network resources in a best-effort fashion.

Listing 1 provides the pseudo-code of the SF-SP policy. From packet headers lines 3 and 4 gather information about flow ID and local interface that the packet should be transmitted to, respectively. In case of high-priority flows (flow ID equal to 1), SF-SP takes trace of the AIP-based channel occupation period (line 6) and the current time (line 7); then, SF-SP ends by allowing the data plane to forward the packet without any additional delay. Instead, in case of no-priority flows, SF-SP exploits the `getSendProgress(sendInterface)` function to determine the quota (percentage) of the previous high-priority packet sent towards the same network interface has been already transmitted. In case the channel is still busy and the number of attempted transmissions is lower than a threshold (line 11), the packet is delayed of an AIP period. In case the channel is considered as not busy, SF-SP ends and the packet is forwarded, but only if the number of attempted retransmissions has not reached the maximum threshold.

Finally, let us not that the current implementation of our solution adopts an on/off approach to decide if no-priority packets should be either forwarded or delayed, i.e., a no-priority packet is transmitted only if the previous high-priority one has been completely transmitted (`sendProgress < 100%` at line 11). However, it is easy to extend the current solution by adopting more articulated fuzzy-based solutions, e.g., by allowing to send no-priority packets in a probabilistic manner if the previous packet has not yet been completely transmitted.

Listing 1 SF-SP policy

```
 1: procedure SFSPRouting(pkt)
 2:   if pkt.flowId is valid and nodeId ≠ pkt.destNodeId then
 3:     flowPriority ← getFlowPriority(pkt.flowId)
 4:     sendInterface ← getSendInterface(pkt.nextHop)
 5:     if flowPriority = 1 then
 6:       sendDuration ← max(25, getAIP(sendInterface) * 1.25)
 7:       sendStartTime ← getCurrentTime()
 8:     else
 9:       sendProgress ← getSendProgress(sendInterface)
10:       attempts ← 0
11:       while sendProgress < 100% and attempts < MAX do
12:         timeToWait ← getAIP(sendInterface)
13:         sleep(timeToWait)
14:         attempts = attempts + 1
15:         sendProgress ← getSendProgress(sendInterface)
16:       end while
17:       if attempts = MAX_ATTEMPTS then
18:         drop packet
19:       end if
20:     end if
21:   end if
22: end procedure
```

Multiple Flows with Single Priority (MF-SP). Similarly to SF-SP, MF-SP considers only one level of priority and thus no-priority flows are transmitted only if there are no high-priority ones. However, in case there are multiple high-priority flows, they are managed in a fair manner, i.e., the available bandwidth is equally shared among high-level flows. To this purpose, the data plane traces per-flow transmitted bytes

in time windows of 2s (default period). Then, considering N high-priority flows, if a flow in the previous period has exploited less than 1/N bandwidth, its packets are transmitted immediately, otherwise they are delayed of Δ ms with probability P. Δ is given by an estimate of the remaining time for the dispatching of the previously sent packet, i.e., $\Delta = (estimatedDuration - elapsedTime) * 1.2$, and P is computed by using the function in Listing 2. In particular, there is no delay in case there is only one flow (condition at line 4 is false and thus the function returns 100%). Otherwise, the probability is obtained from the number of sent packets by which the flow has exceeded its limit (Listing 2, lines 5 to 8). In case of no information about a flow, e.g., it has just started traversing the node, the flow is not delayed.

Listing 2 MF-SP policy forwarding probability

```
1: function getMFSPSendProb(flowId)
2:   sendProb ← 100
3:   sentPkt ← getLastPeriodSentPkt(flowId)
4:   if totalFlows > 1 then
5:     sentPktTarget ← lastTotalSentPkt/lastTotalFlows
6:     sentPktOffset ← sentPkt − sentPktTarget
7:     offsetPerc ← (sentPktOffset * 100)/sentPktTarget
8:     sendProb ← 100 − (offsetPerc * lastTotalFlows)
9:   end if
10:  return sendProb
11: end function
```

Multiple Flows with Multiple Priorities (MF-MP). Similarly to MF-SP, MF-MP handles multiple flows at the same priority level in a fair way. In addition, there are multiple priority levels and lower priority flows can always exploit some network resources (based on the priority level and the number of active flows) without waiting for the completion of higher priority ones. In other words, higher priority flows are not completely preemptive and thus lower priority flows are not completely inhibited (they partly compete with higher priority ones). In particular, the probability of transmission is the same of the MF-SP case but divided by the priority level, where the highest priority level is 1, the second priority level is 2, and so on (see Listing 3, line 12). In other words, network resources are evenly shared among flows of the same priority level and with an exponential decrease according to the priority level.

Listing 3 MF-MP policy forwarding probability

```
1: function getMFMPSendProb(flowId, flowPriority)
2:   sendProb ← 100
3:   if totalFlows > 1 then
4:     lastSentPkt ← getLastSentPkt(flowId, flowPriority)
5:     lastTotalSentPkt ← getLastTotalSentPkt(flowPriority)
6:     lastTotalFlows ← getLastTotalFlows(flowPriority)
7:     sentPktTarget ← lastTotalSentPkt/lastTotalFlows
8:     sentPktOffset ← lastSentPkt − sentPktTarget
9:     offsetPerc ← (sentPktOffset * 100)/sentPktTarget
10:    sendProb ← 100 − (offsetPerc * lastTotalFlows)
11:    if higherPriorityTotalFlows > 0 then
12:      sendProb ← sendProb/flowPriority
13:    end if
14:  end if
15:  return sendProb
16: end function
```

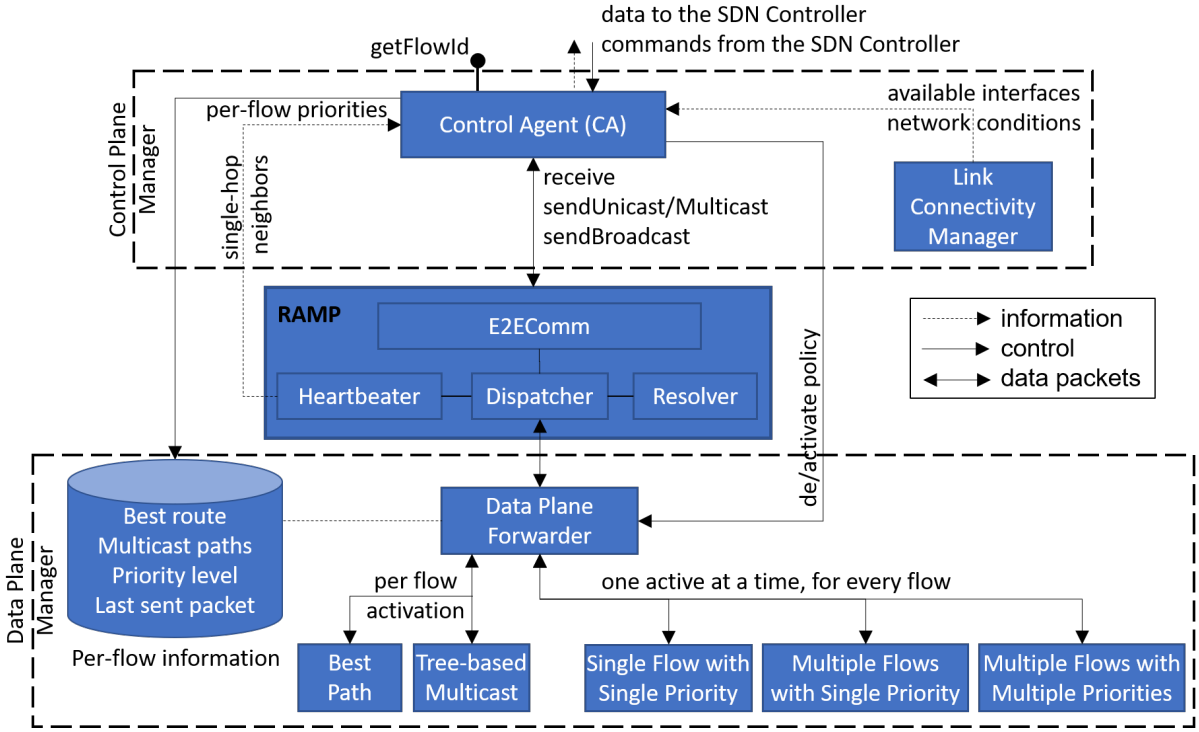


Fig. 4 Overall architecture of SDN nodes.

5 Prototype Implementation and Performance Evaluation

5.1 Primary Implementation Insights

We have designed, developed, and tested the proposed solution not only to demonstrate its feasibility and effectiveness, but also to provide the WMN community with a middleware framework prototype for the flexible, dynamic, and application-driven management of QoS in spontaneous WMNs based on the SDN approach. Figure 4 depicts the overall architecture of our spontaneous WMN node, which extends our past work on the Real Ad-hoc Multi-hop Peer-to-peer (RAMP) middleware. Note that the traditional RAMP middleware only supports the creation of overlay layers to enable spontaneous DSR-like networking; additional details about the RAMP middleware, its architecture, and implementation can be found in [5]). Such overlay layers are exploited by the originally proposed solution to enable SDN capabilities, in particular to dispatch packets related to both control and data planes among adjacent nodes. The source code for Linux/Windows/macOS can be found at `https://github.com/DSG-UniFE/ramp`.

As already anticipated, the components presented in Figure 4 are deployed and activated on each WMN node to allow its participation to its SDN-enabled spontaneous WMN island and to enable the enforcement of routing and traffic engineering policies. In addition, for each WMN island there is one node acting as SDN Controller by registering itself to the local RAMP as an "SDN Controller" service, thus allowing remote CAs to dynamically discover and register to it. Based on the information provided by CAs, the SDN Controller generates a weighted graph (see Figure 5). To this purpose, the current implementation of the SDN Controller adopts the Graph Stream library [42] and allows to apply the Breadth First and the Dijkstra's algorithm based on different cost functions, among the others.

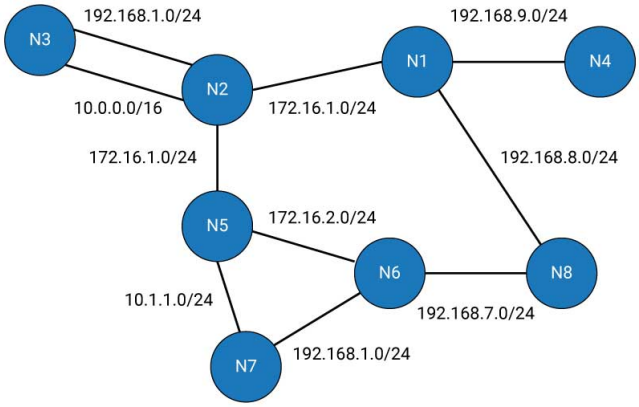


Fig. 5 Example of topology representation.

By delving into finer details, the Control Plane Manager at each node (Figure 4, top) primarily consists of Link Connectivity Manager (LCM) and Control Agent (CA). LCM manages single-hop links and provides network status information.

To provide a portable and easily extensible solution, we have designed and implemented the LCM to manage networking devices and to gather related information over different and heterogeneous underlying platforms. For instance, on Android devices, LCM exploits not only `TrafficStats` to gather network statistics, but also `WifiManager`/`WifiConfiguration` and `BluetoothAdapter`/`BluetoothPan` to manage IEEE 802.11 and Bluetooth interfaces respectively, e.g., also enabling the dynamic connection to hotspots and the dynamic role switching of smartphones from clients to hotspots (and vice versa). Instead, on desktop Windows, Linux, and Mac OS, it exploits the Operating System and Hardware Information (OSHI) [43] library to gather network, CPU, memory, and battery statistics. Monitoring data are gathered on each node for each network interface and are sent to the SDN Controller, together with the set of one-hop neighbors identified by a node ID (unique in a spontaneous WMN) and the IP addresses of their network interfaces. Moreover, to take full advantage of any connectivity opportunity that may arise by the mobility of users (and inspired by [44]), we have designed and developed LCM to reconfigure IP subnets by a) dynamically accessing different hotspots and b) dynamically switching the role of smartphones from clients to hotspots. LCM provides an abstract access to networking devices, by hiding the complexity of accessing heterogeneous wireless protocols such as IEEE 802.11 and Bluetooth. Its API allows to discover available hotspots, to connect to a given peer, to switch a node's role from client to hotspot and vice versa, and to gather network traffic statistics. In addition, it provides a listener interface to allow external components to be notified about any network configuration modification in an event-driven way, e.g., to notify a node state change from client to hotspot, or the handover towards a new hotspot.

CA exposes an API that allows local applications not only to specify destination nodes (one for unicast, multiple ones for multicast), but also to indicate the path selection mechanism that the SDN Controller should apply (with a per packet granularity). To this purpose, applications can invoke the `getFlowId(appReq, destNodeId(s), pathSelectionMetric)` method to enable the SDN-based features of our middleware. At method invocation, the local CA transparently interacts with the remote SDN Controller to require the computation of the best path(s) from the sender to one (for the BP routing policy) or more (for the TM routing policy) destination nodes, based on the specified path selection metric and the associated application requirements. The SDN Controller also computes the applicable priority level (to be exploited by our traffic engineering policies), generates a new flow id to associate with the request, and provides the CA with it. Finally, the `getFlowId()` method returns this flow id to the invoking application;

as already mentioned, it is the application that is in charge of explicitly tagging its forthcoming packets with it when calling the `sendUnicast/Multicast()` methods of the E2EComm component, to enable our middleware-based routing and traffic engineering policies. In addition, CA dynamically discovers the SDN Controller, registers the local node to it, and periodically verifies the availability of the SDN Controller. In case of SDN Controller unavailability, CA looks for another one; if multiple SDN Controllers are available, in our current implementation CA selects the one with lowest RTT (easily modifiable via a modular selection function). Let us point out that the RAMP SDN Controller performs all control plane management operations in a proprietary way at the moment, without employing the OpenFlow protocol in our current prototype implementation. This decision has been taken also by considering that SDN controllers and the OpenFlow protocol are primarily designed to manage traditional IP networks: they are not suitable for considering multi-hop spontaneous WMNs consisting of different IP subnets. The adoption of the RAMP middleware (and its DSR-like routing mechanism) allows us, instead, to dispatch control packets in our targeted deployment environment as previously described; that would not be practically possible by adopting vanilla IP routing mechanisms that require the frequent modification of routing tables at any intermediary node. For the sake of briefness, we do not provide details here about our control plane messages, but the interested readers can refer to the companion RAMP project Web site for the full detail about them (see `https://github.com/DSG-UniFE/ramp/blob/master/docs/RAMP-MULTILane-SDN-ControlPlane.md`). However, let us note that if a mobile node does not support our control plane protocol (or it does not provide the capability of being controlled), it is not possible to actively manage that node in compliance with our deployed policies. In this case, the network administrator can exploit the SDN controller to either limit the networking resources provided to traffic flows to/from that node or even isolate that node at all, e.g., by properly configuring the forwarding rules of its neighbor nodes. Finally, the adoption of a secure channel between nodes and the SDN Controller is of course advisable to ensure the integrity of control packets, among the others. For instance, advanced security mechanisms may involve the mutual authentication of nodes and SDN Controllers via X.509 certificates. Then, SDN Controllers and nodes could sign (or even encrypt) transmitted control packets as well as packets with information (and then send back signed numbered acks) with a twofold objective: on the one hand, ensuring that packets are not modified (or even read) by intermediary nodes; on the other hand, achieving non-repudiability for the transmitted data, thus reducing the risk that nodes produce fake information.

The Data Plane Manager (Figure 4, bottom) employs the general-purpose Data Plane Forwarder (DPF) component and its traffic management policy specializations, either for rerouting packets or for applying traffic engineering techniques. Moreover, RAMP serves as the Overlay Network Packet Dispatcher by supporting the creation of the multi-hop spontaneous WMN in a best-effort manner. DPF properly manages packets received by the RAMP middleware on a per-flow basis, by adopting a listener approach that allows to intercept any incoming packet. In particular, DPF exploits information provided by CA (and sent by the SDN Controller) to apply routing policies on a per-flow basis. To this purpose, CA populates a database with <`flow-id, path`> and <`flow-id, [next-hop]`> information for BP and TM routing policies respectively: the former, providing a single full path to which packets labeled with the given flow ID should be forwarded; the latter, one or more next paths to which multicast packets should be sent. By delving into finer details, DPF interacts with the local RAMP Dispatcher to intercept any packet flowing through the node. Then, it gathers the flow id from the packet header and looks up in the Per-flow Information table to verify if and how the current packet should be managed. If the flow id matches a Best route entry, it gathers the new path and provides both the packet and the path to the Best Path component, in charge of modifying the packet header to reflect the retrieved new best path (thus rerouting the packet towards a new path). Then, DPF provides the packet back to the RAMP Dispatcher in charge of forwarding it according to the header information. If the flow id matches a Multicast path entry, it gathers the paths information and provides it to the Tree-based Multicast component, together with the original packet. Then, Tree-based Multicast creates new packets with the same payload and each one with one of the retrieved paths, and

finally provides it to the RAMP Dispatcher. Note that in the case there are no data about the retrieved flow id, PDF does not modify the packet, which therefore follows the path as specified in its original header.

In addition, DPF delegates traffic management to one of the three currently available SF-SP, MF-SP, and MF-MP policies (dynamically de/activated by the SDN Controller via a proper control message sent to CAs). To this purpose, based on information and commands sent by the SDN Controller, CA populates the Data Plane database with <`flow-id, priority-level`> information while other information required to compute the AIP value is locally gathered by DPF itself. In particular, if SF-SP is active, then DPF forwards any traversing packet to the Single Flow with Single Priority component. In case of high-priority packets, Single Flow with Single Priority takes trace of send duration and start time (see lines 6 and 7 of Listing 1); in case of no-priority packets, it exploits the previously stored information to compute the send progress and decide whether packet forwarding should be delayed (see lines 9 and 11 of Listing 1). If MF-SP or MF-MP are activated, DPF forwards the packet to either Multiple Flows with Single Priority or Multiple Flows with Multiple Priorities respectively. Then, these components exploit the functions presented in Listing 2 and Listing 3 to compute their local forwarding probability.

Finally, let us note that DPF first applies routing policies with a per-flow granularity by looking up the Data Plane database. Then, for each SDN-enabled packet (i.e., specifying a flow ID in the header) it applies the currently activated traffic engineering policy to verify if and how much the packet dispatching should be delayed.

5.2 Performance results

To validate the proposed solution and demonstrate the benefit of adopting the SDN approach in spontaneous WMNs (together with the effectiveness of the implemented middleware prototype), here we report some selected performance results that show a) the efficiency of the CA registration procedure and, most relevant, b) the effectiveness of SF-SP, MF-SP, and MF-MP traffic engineering policies when sending multiple traffic flows with different priority levels towards the same path. To this purpose, we have deployed our solution on five heterogeneous nodes: CA1 with Intel Core i5-760 (Quad Core, 2.80 GHz) CPU, 4 GB of RAM, and Microsoft Windows 10 OS, CA2 with Intel Core i3-5005U (Dual Core, 2.00 GHz) CPU, 8 GB of RAM, and Arch Linux OS, and CA3, CA4, and CA5 with Raspberry Pi 3 Model B BCM2837 (Quad Core, Quad Core 1.2GHz) CPU, 1 GB of RAM, and Raspbian Linux OS (see Figure 6). CA1-CA2 and CA3-CA4 links are based on IEEE 802.11g, the other ones on Ethernet. The CA is deployed on any participating node, the SDN Controller only on CA4. It is worth noting that the reported performance results have the primary objective of demonstrating the feasibility of the proposed solution, i.e., that the SDN adoption in spontaneous WMNs can actually allow to improve the provisioned QoS. In addition, they present and outline how the presence of bottleneck links along eventually long paths composed of multiple hops can relevantly affect end-to-end QoS, if flows are not properly managed.

At startup, CA1 starts looking for SDN Controllers by sending discovery packets (size = 190B) via multi-hop broadcast; this triggers a message reply of the SDN Controller (packet size = 206B) and a join command of CA1 to the SDN Controller (packet size = 1729B). The overall time required for the whole procedure is about 98 ms (average value over 100 runs, with a variance less than 5%).

Then, C1 sends two large messages (about 55MB each, in order to stress the deployment environment in a challenging provisioning situation) to CA5 via CA2 and CA3 by exploiting the splitting capabilities of the underlying RAMP middleware (see [5]). Before sending the messages, the sender application on CA1 specifies to the SDN Controller its willingness of transmitting to CA5 at low/high priority (for the first/second message respectively) based on the currently enabled SP policy (previously set by the SDN Controller). Then, the SDN Controller generates a new random flow ID for each flow, e.g., `F-ID-Low` and `F-ID-High`, informs CAs on every node that the flow ID `F-ID-Low`/`F-ID-High` has low/high priority, and replies to the application level on CA1 with the generated flow ID. Finally, the sender application

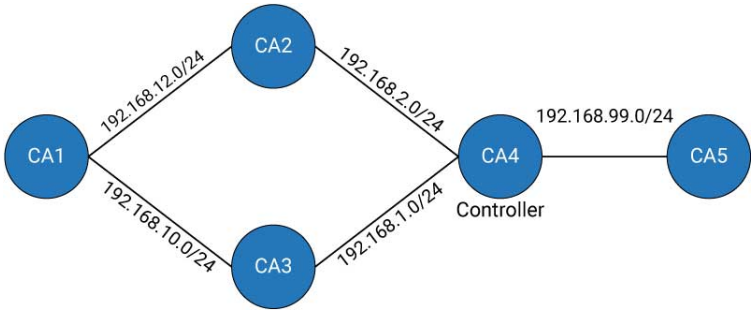


Fig. 6 5 nodes topology.

starts sending the messages labeling the traffic with the `F-ID-Low`/`F-ID-High` flow ID for the first/second message. Note that the application-level labeling of flows can be performed via simple middleware stubs, thus achieving full transparency and easy integration with possible legacy applications.

Figure 7a and Figure 7b report the per-flow throughput and CPU load on the intermediate node CA4 when sending the first low priority message few seconds before the second high priority message, but with our middleware disabled. As expected, when the first flow is the only one transmitted, it is dispatched by saturating the available bandwidth (about 6 MB/s). Then, when also the second flow is transmitted, the two flows start competing for the available bandwidth in a best-effort way, with the consequence that on average the bandwidth is equally split between the two flows. Finally, when the first flow ends, the second one starts acquiring more bandwidth up to its saturation. The time required to transmit the first flow is 14.1 s and for the second flow 14.0 s, thus quantitatively showing that the throughput has been equally shared between the flows. Figure 7c and Figure 7d report the per-flow throughput and CPU load on the intermediate node CA4 when transmitting the two flows above, but with active QoS management through the middleware enforcement of the SF-SP traffic engineering policy. Again, the low priority flow saturates the available bandwidth when it is the only active flow. However, in this case, as long as the high priority flow is transmitted, the dispatching of the low priority flow is correctly inhibited, allowing the former to fully exploit the available bandwidth. In particular, note that our middleware starts inhibiting the low priority flow very promptly as soon as the high priority starts to be received (see Figure 7c, about 7 s). Later on, the high priority flow terminates and then the intermediate node starts transmitting the low priority flow, by saturating the bandwidth till its conclusion. The overall picture clearly confirms that the SF-SP traffic engineering policy can actually inhibit low priority flows in favor of high priority ones. In this case, the first low priority flow takes 19.4 s to completely reach the destination while the second high priority one takes only 9.8 s. In addition, to apply the SF-SP algorithm nodes has an average cost of 2.05 ms. Also note that the CPU load does not relevantly increase when comparing the two solutions: this primarily depends on the amount of transmitted traffic and is a quantitative indication of the efficiency of the proposed solution (same considerations apply for the following deployment environments).

Figure 8a and Figure 8b report the per-flow throughput and CPU load on the intermediate node CA4 when sending three different flows at short distance, with our SDN-based solution disabled. Similarly to the case with two flows and no QoS management, active flows share the available bandwidth in a best-effort way. The three flows take 21.0 s, 24.2 s, and 22.4 s to reach their destinations. Figure 8c and Figure 8d report the per-flow throughput and CPU load on the intermediate node CA4 when transmitting the three flows above but with active QoS management by enabling the MF-SP traffic engineering policy, when assigning low priority to the first flow and high priority to the second and third flows. Initially, the only active priority flow achieves the whole bandwidth. Then, as soon as an high priority flow starts traversing the same node, the low priority flow is competently inhibited, as previously described in the SF-SP case. However, when the same node is traversed by another high priority flow, the MF-SP starts to shape the

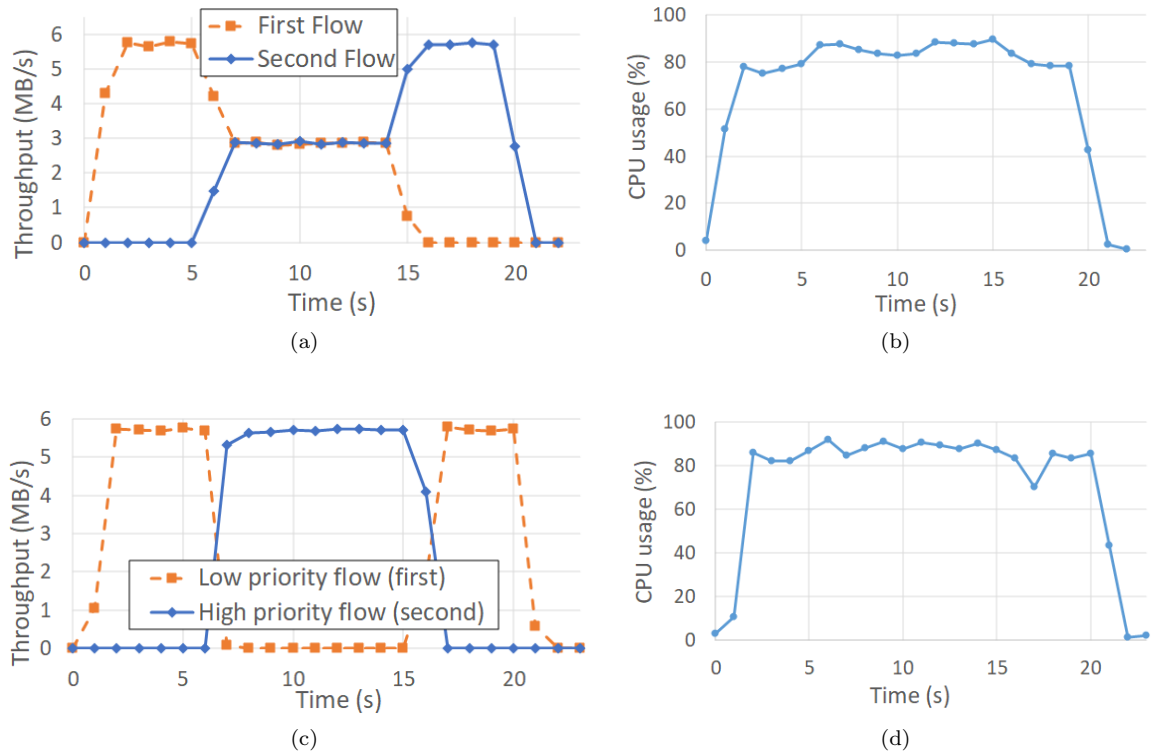


Fig. 7 Two flows: throughput and CPU load without SDN (a) (b), and with SF-SP (c) (d).

traffic to fairly provide bandwidth to the two high priority flows. In this case, the low priority flow takes 29.2 s, while the two high priority ones 16.5 s and 16.3 s. In addition, the computation of the MF-SP approach has an average cost of 2.67 ms. Let us rapidly notice that these results are slightly worse than SF-SP ones because in this case there is the need to consider the slight additional overhead of computing the MF-SP forwarding probability.

Figure 8e and Figure 8f report the per-flow throughput and CPU load on the intermediate node CA4 when transmitting the three flows above but with active QoS management through the enforcement of our MF-MP traffic engineering policy, when assigning low priority to the first two flows and high priority to the third flow. As usually, the first flow initially achieves the full bandwidth as long as it is the only one active over the deployment environment. Then, when also the second low priority flow starts traversing the same node, the first low priority flow is temporarily inhibited because it has recently been assigned with more network resources (similarly to the MF-SP case with two high priority flows). Finally, the same node is traversed by a high priority flow gaining most bandwidth at the expense of the two low priority flows, which start to show reduced throughput. However, note that in this case the two low priority flows are not completely inhibited, but they achieve an overall bandwidth that is half of the high priority one. In this case, the two low priority flows take 27.69 s and 27.18 s, while the high priority one 13.70 s. The MF-MP computation generates an additional cost of 4.09 s. These results are worse than MF-SP ones because the MF-MP forwarding probability is slightly more complex to compute and involves additional data. Finally, also note that the per-flow throughput exhibits some oscillations since the MF-MP algorithm adopts a probabilistic approach based on the amount of recently sent packets and the priority level. However, the

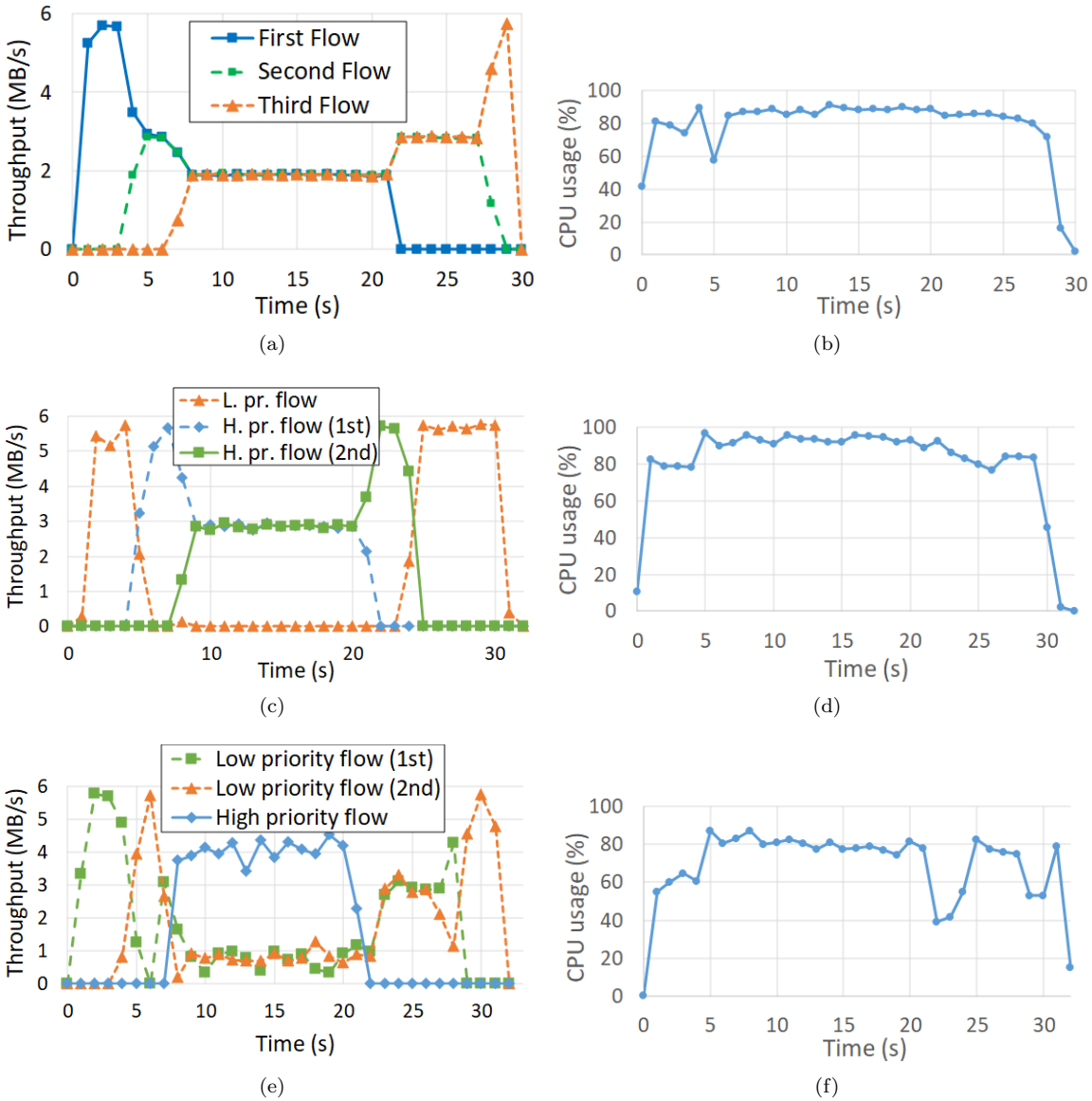


Fig. 8 Three flows: throughput and CPU load without SDN (a) (b), with MF-SP (c) (d), and with MF-MP (e) (f).

overall trends of per-flow throughput in Figure 8e demonstrate that MF-MP is actually able to throttle the throughput of multiple flows that compete over the same transmission link, by fairly considering their priority levels. In addition, the MF-MP algorithm takes some time to identify that the high priority flow has finished and to provide additional available bandwidth to low priority ones. In particular, in Fig. 8f at t = 22-23 s the overall throughput of active flows is lower than the available bandwidth because the high priority flow has already terminated while low priority ones have not yet been able to increase their throughput. The temporarily reduced overall traffic has the side effect of lowering the CPU load. In fact, it is worth noting that the CPU trend substantially follows the traffic load: the greater the traffic, the greater

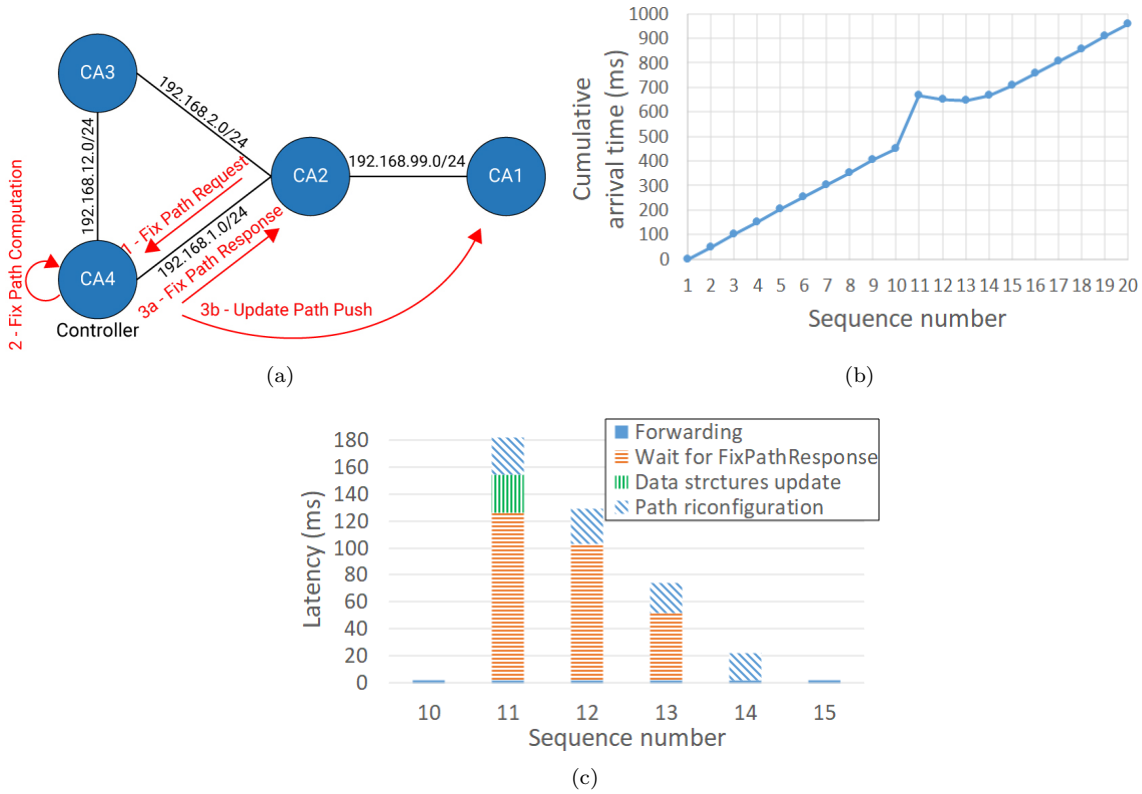


Fig. 9 Topology with the `FixPath` protocol (a), per packet arrival time on CA3 (b), and packet management latency on CA2 (c) of the SDN-based reroute solution.

the CPU usage. In other words, the reported results show that the imposed CPU load is mainly due to packet dispatching, with no relevant differences among the proposed solutions.

Finally, to verify the behaviour of our SDN-based rerouting features, we have tested the proposed solution in a dynamic scenario. The primary goal is to demonstrate that our solution can seamlessly recover from path disruption in a prompt and efficient way, with limited control packet overhead and latency. To this purpose, we adopted the topology in Figure 9a: CA1-CA2 and CA3-CA4 are links based on IEEE 802.11g, the remainder ones on Ethernet, and the SDN Controller runs on CA4. CA1 sends a traffic flow to CA3 via the CA1-CA2-CA3 path, 20 packets per second, each packet with a 1 KB payload. Just after CA2 has delivered the packet with sequence number 10 (packet #10), we abruptly remove the CA2-CA3 link. The unavailability of that link causes a destination unreachable exception when the CA2 node tries to send packet #11. CA2 notices the link unavailability and sends a `FixPathRequest` message to the SDN Controller, specifying the id of the failed link, the involved flowId, the id of the original sender, and the desired destination (packet size 901 B, Fig. 9a, point 1, latency 20 ms). Then, the SDN Controller removes the failed link from the topology graph and computes two paths: one from the intermediary node to the destination and another from the original sender to the destination (Fig. 9a, point 2, latency 17 ms). The former path is sent to CA2 in the `FixPathResponse` message (packet size 1194 B, Fig. 9a, point 3a, latency 32 ms) to inform CA2 on how to reroute packets that the sender has already dispatched by using the broken path. The latter path is sent to CA1 in the `UpdatePathPush` message (packet size 1241 B, Fig. 9a, point

3b, latency 56 ms) to inform the original sender that the following packets of a given flowId should be sent via the new CA1-CA2-CA4-CA3 path. Figure 9b presents the associated per-packet arrival time, by plotting for each packet the cumulative arrival time starting from the first packet arrived at the destination. It is worth noting that only 4 packets present a slightly delayed arrival time, i.e., packets #11, #12, #13, and #14, while following ones exhibit again a regular arrival time pattern. Also note that CA2 sends the `FixPathRequest` message only the first time the packet delivery fails, while following packets are stopped on a semaphore waiting for the reception of the `FixPathResponse` message. Once received, packets waiting at the semaphore are activated, thus allowing to concurrently take advantage of the new path.

To better and more explicitly detail how the `FixPath` control protocol affects data plane packets, Fig. 9c reports the latency of packet management on node CA2. In particular, the figure compares the latency of data plane packets affected by the `FixPath` control protocol with immediately previous and following ones. Note that the latency associated with packet forwarding is very low in regular conditions, i.e., less than 1 ms (see packets #10 and #15). Instead, packet #11 presents an additional latency of more than 180 ms, mostly due to waiting for the completion of the `FixPath` control protocol (and related computation), the remainder to data structure updates that are performed when CA2 receives the `FixPathResponse` message from the controller and to path reconfiguration in the header of the data plane packet. The following packets #12, #13, and #14 show decreasing latency because (i) there is no more need for data structure updates (done only once) and (ii) they can take advantage of the `FixPath` control protocol previously triggered by packet #11 (reduction of idle time to wait for the `FixPathResponse` message). Then, starting from packet #15, the latency returns back to minimum regular values less than 1 ms, since in the meantime CA1 has received the `UpdatePathPush` message with the new path; thus, the following packets already use the new path, with no need of path reconfiguration on CA2.

6 Conclusive remarks and future work

The paper presents a novel architecture and middleware implementation based on the SDN approach to enforce differentiated QoS management levels in spontaneous WMNs. In particular, by adopting a centralized SDN Controller for each (sufficiently large) WMN island, we have demonstrated the potential to take better control decisions based on the awareness of not only topology and network status, but also application requirements according to the specifications of participating nodes. Furthermore, the proposed middleware has been designed with flexibility, dynamicity, and scalability in mind, to the purpose of enabling the exploitation of the same WMN for different application purposes and of efficiently supporting the frequent dynamic adding/removal of nodes to/from WMN islands.

In particular, the reported performance results show that our SDN-based middleware can apply not only rerouting, but also traffic shaping policies, by finely tuning how the traffic of different flows is managed by intermediary nodes. In addition, the results point out that the implemented traffic engineering policies, even if simple, allow to fairly assign network resources to different traffic flows. For example, by dynamically selecting the most proper policy in relation to the specific goal of the WMN itself, it is possible either to completely inhibit low priority flows or to provide them with a controllable part of the available bandwidth.

The encouraging results achieved so far, based on a working prototype available to the community of researchers in the field, are stimulating further ongoing research work. On the one hand, we are currently working on the validation of our middleware on multiple and large WMN islands (up to one hundred nodes each). On the other hand, we are working on how to support the lazy interaction and cooperation (federation) of different SDN Controllers and on how to tune the size of WMN islands via the dynamic insertion of new SDN Controllers, if and where required. To this purpose, in case a CA does not discover an SDN Controller fitting its requirements, e.g., because the discovered SDN Controllers exhibit unsatisfactory RTT and/or path length values or there are no SDN Controllers at all, a CA should autonomously promote itself to serve as an SDN Controller, by avoiding that multiple nodes become SDN Controllers at the same

time [45]. In this way, we will enable the dynamic splitting of too wide WMN islands into multiple ones, thus preserving good communication performance between the SDN Controller and the nodes under its direct control. On the contrary, an SDN Controller will downgrade itself to CA-only if the number of its controlled nodes is too low or if there are other nearby nodes able and willing to behave as the new SDN Controller.

Finally, we believe that the availability of the source code of our middleware, originally presented in this paper, can contribute to foster the creation of a vibrant community of academic and industrial researchers, by giving momentum to the adoption and further investigation of the promising SDN approach in spontaneous WMNs.

References

1. K. Liu, W. Shen, B. Yin, X. Cao, L. X. Cai, and Y. Cheng, "Development of mobile ad-hoc networks over wi-fi direct with off-the-shelf android phones", in 2016 IEEE International Conference on Communications (ICC), May 2016, pp. 16.
2. G. Fodor, S. Roger, N. Rajatheva, S. B. Slimane, T. Svensson, P. Popovski, J. M. B. Da Silva, and S. Ali, "An Overview of Device-to-Device Communications Technology Components in METIS", IEEE Access, vol. 4, pp. 3288-3299, 2016.
3. I.F. Akyildiz and Xudong Wang, "A survey on wireless mesh networks", IEEE Communications Magazine, vol. 43, no. 9, pp. S23-S30, Sept. 2005.
4. B. Zaghdoudi, H.K.B. Ayed, I. Gnichi, "A protocol for setting up ad hoc mobile clouds over spontaneous MANETs: A proof of concept", 2016 Cloudification of the Internet of Things (CIoT), Paris, 2016, pp. 1-6.
5. P. Bellavista, A. Corradi, C. Giannelli, "Middleware for Differentiated Quality in Spontaneous Networks", IEEE Pervasive Computing, vol. 11, no. 3, March 2012.
6. J. Lloret, L. Shu, R. Lacuesta, M. Chen, "User-Oriented and Service-Oriented Spontaneous Ad Hoc and Sensor Wireless Networks", Ad Hoc and Sensor Wireless Networks, vol. 14, nos. 1/2, 2012.
7. A. Bader, M.S. Alouini, "Mobile Ad Hoc Networks in Bandwidth-Demanding Mission-Critical Applications: Practical Implementation Insights", IEEE Access, vol. 5, 2017.
8. S.G. Pease, I.W. Phillips, L. Guan, "Adaptive Intelligent Middleware Architecture for Mobile Real-Time Communications", IEEE Transactions on Mobile Computing, vol. 15, no. 3, 2016.
9. Z. Li, H. Shen, "A QoS-Oriented Distributed Routing Protocol for Hybrid Wireless Networks", IEEE Trans. on Mobile Computing, vol. 13, no. 3, 2014.
10. 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.
11. 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.
12. Open Networking Foundation, "OpenFlow", available online at https://www.opennetworking.org/sdn-resources/openflow (last accessed on April 30th, 2019).
13. Y. Yu et al., "Field Demonstration of Multi-Domain Software-Defined Transport Networking With Multi-Controller Collaboration for Data Center Interconnection", IEEE/OSA Journal of Optical Comm. and Networking, vol. 7, no. 2, 2015.
14. Y. Wu et al., "Orchestrating Bulk Data Transfers across Geo-Distributed Datacenters", IEEE Trans. on Cloud Computing, vol. 5, no. 1, 2017.
15. R. Ahmed, R. Boutaba, "Design Considerations for Managing Wide Area Software Defined Networks", IEEE Comm. Magazine, vol. 52, no. 7, 2014.
16. F.X.A.Wibowo, M.A.Gregory, K. Ahmed, K.M.Gomez, "Multi-domain Software Defined Networking: Research status and challenges", Journal of Network and Computer Applications, vol. 87, June 2017.
17. R. Masoudi, A. Ghaffari, "Software defined networks: A survey", Journal of Network and Computer Applications, vol. 67, May 2016.
18. M. Abolhasan, J. Lipman, W. Ni, B. Hagelstein, "Software-Defined Wireless Networking: Centralized, Distributed, or Hybrid?", IEEE Network, vol. 29, no. 4, 2015.
19. F. Yaghoubi, M. Furdek, A. Rostami, P. hln, L. Wosinska, "Consistency-aware Weather Disruption-tolerant Routing in SDN-based Wireless Mesh Networks", IEEE Trans. on Network and Service Management, vol. 15, no. 2, 2018,
20. E. Seo, V. Zalyubovskiy, T.M. Chung, "Design of the Central LISP Management System for the Software-Defined Wireless Network (SDWN)", 15th ACM Int. Symp. on Mobility Management and Wireless Access (MobiWac '17), pp. 93-100, 2017.

21. T.M.T. Nguyen, L. Hamidouche, F. Mathieu, S. Monnet, S. Iskounen, "SDN-based Wi-Fi Direct clustering for cloud access in campus networks", Annals of Telecommunications, vol. 73, no. 3-4, April 2018.
22. R. Saunders, J. Cho, A. Banerjee, F. Rocha, J. Van der Merwe, "P2P Offloading in Mobile Networks using SDN", Symp. on SDN Research (SOSR '16), March 2016.
23. J. Lee et al., "meSDN: mobile extension of SDN", The Fifth Int. Workshop on Mobile Cloud Computing & Services (MCS '14), June 2014.
24. I.T. Haque, N. Abu-Ghazaleh, "Wireless Software Defined Networking: A Survey and Taxonomy", IEEE Communications Surveys & Tutorials, vol. 18, no. 4, 2016.
25. J. Wang, Y. Miao, P. Zhou, M.S. Hossain, S.M.M. Rahman, "A software defined network routing in wireless multihop network", Journal of Network and Computer Applications, vol. 85, May 2017.
26. H. Fu, Y. Liu, K. Liu, Y. Fan, "An SDN-based congestion-aware routing algorithm over wireless mesh networks", Wireless Communication And Sensor Network-Proceedings Of The International Conference (Wcsn 2015), 2016.
27. D. Sajjadi, R. Ruby, M. Tanha, J. Pan, "Fine-grained traffic engineering on SDN-aware Wi-Fi mesh networks", IEEE Trans. on Vehicular Technology, vol. 67, no. 8, 2018.
28. M.A. Filho, M. Ribeiro, C. Celes, A. Santos, C. Oliveira, R. Braga, "Performance Analysis of OpenFlow in a Multi-Interface WMN Testbed", Euro American Conf. on Telematics and Information Systems (EATIS '18). ACM, New York, NY, USA, Article 46, 2018.
29. S. Salsano, G. Siracusano, A. Detti, C. Pisa, P.L. Ventre, N. Blefari-Melazzi, "Controller selection in a Wireless Mesh SDN under network partitioning and merging scenarios", arXiv preprint arXiv:1406.2470, 2014.
30. M. Labraoui, M. Boc, A. Fladenmuller, "Self-configuration mechanisms for SDN deployment in Wireless Mesh Networks", 2017 IEEE 18th Int. Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM), Macau, 2017, pp. 1-4.
31. J.F.G. Orrego, J.P.U. Duque, "Throughput and delay evaluation framework integrating SDN and IEEE 802.11s WMN", 2017 IEEE 9th Latin-American Conference on Communications (LATINCOM), Guatemala City, 2017, pp. 1-6.
32. 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.
33. 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.
34. 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.
35. P. Baskett, Y. Shang, W. Zeng, B. Guttersohn, "SDNAN: Software-defined networking in ad hoc networks of smartphones", IEEE Consumer Communications and Networking Conference (CCNC), Jan. 2013.
36. C. Canali, M. E. Renda, P. Santi, S. Burresi, "Enabling Efficient Peer-to-Peer Resource Sharing in Wireless Mesh Networks", IEEE Trans. on Mobile Computing, vol. 9, no. 3, pp. 333-347, March 2010.
37. B.M.C. Silva, J.J.P.C. Rodrigues, N. Kumar, G. Han, "Cooperative Strategies for Challenged Networks and Applications: A Survey", IEEE Systems Journal, vol. 11, no. 4, Dec. 2017.
38. P. Bellavista, C. Giannelli, S. Lanzone, G. Riberto, C. Stefanelli, M. Tortonesi, "A middleware solution for wireless IoT applications in sparse smart cities", Sensors, vol. 17, no. 11, 2017.
39. 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), Paris, 2013, pp. 1-6.
40. F. Bannour, S. Souihi, A. Mellouk, "Distributed SDN Control: Survey, Taxonomy, and Challenges", IEEE Comm. Surveys & Tutorials, vol. 20, no. 1, pp. 333-354, Firstquarter 2018.
41. P. Berde, et al., "ONOS: Towards an Open, Distributed SDN OS", Workshop on Hot topics in software defined networking, pp. 1-6, Chicago, USA, Aug. 2014.
42. "GraphStream - A Dynamic Graph Library", available online at http://graphstream-project.org (last accessed on April 30th, 2019).
43. "Native Operating System and Hardware Information", available online at https://github.com/oshi/oshi (last accessed on April 30th, 2019).
44. S. Trifunovic et al., "WiFi-Opp: adhoc-less opportunistic networking", 6th ACM Workshop on Challenged Networks (CHANTS '11), pp. 37-42, Las Vegas, Nevada, USA, 2011.
45. C. Cooper, D. Franklin, M. Ros, F. Safaei, M. Abolhasan, "A Comparative Survey of VANET Clustering Techniques", IEEE Comm. Surveys & Tutorials, vol. 19, no. 1, 2017.