In-band Network Monitoring Technique to support SDN-based Wireless Networks

Jetmir Haxhibeqiri*, Pedro Heleno Isolani†, Johann M. Marquez-Barja†, Ingrid Moerman* and Jeroen Hoebeke*
* IDLab, Ghent University – imec, Ghent, Belgium
† IDLab, University of Antwerp – imec, Antwerp, Belgium

***Abstract*—Most industrial applications demand determinism in terms of latency, reliability, and throughput. This goes hand in hand with the increased complexity of real-time network programability possibilities. To ensure network performance low-overhead, high-granularity, and timely network verification techniques need to be deployed. The first cornerstone of network verification ability is to enable end-to-end network monitoring, including end devices too. To achieve this, this paper shows a novel and low overhead in-band network telemetry and monitoring technique for wireless networks focusing on IEEE 802.11 networks. A design of in-band network telemetry enabled node architecture is proposed and its proof of concept implementation is realized. The PoC realization is used to monitor a real-life SDN-based wireless network, enabling on-the-fly (re)configuration capabilities based on monitoring data. In addition, the proposed monitoring technique is validated in terms of monitoring accuracy, monitoring overhead, and network (re)configuration accuracy. It is shown that the proposed in-band monitoring technique has 6 times lower overhead than other active monitoring techniques on a single-hop link. Besides this, it is demonstrated that (re)configuration decisions taken based on monitored data fulfill targeted application requirements, validating the suitability of the proposed monitoring technique.**

***Index Terms*—In-band Network Telemetry (INT), WiFi, SDN, Industry 4.0, 5gEmPower.**

I. Introduction

Network technology boundaries are being pushed further, with Industry 4.0 being one of the driving forces. System-level operation and the quest towards more automation is driving machines to perform tasks that require tighter network delivery guarantees. Hand in hand with this, networks are becoming increasingly more complex and diverse, significantly complicating their management for human administrators. This is also witnessed by several trends in wireless networking, like Software Defined Networking (SDN), Software Defined Radios (SDR) [1], 5G [2], IEEE 802.11ax [3], etc., which all introduce more advanced (re)configuration capabilities to meet application demands. Similarly to increased network complexity, applications are demanding more capacity and lower communication latencies. Considering both, network complexity and application demands, networks should enable techniques for on-the-fly performance monitoring, verification, and (re)configuration.

The increased network complexity will ultimately lead to the need of real-time network management that is based on temporal, historical and frequency of network data. To this end, full cognitive-based network management mechanism can decide to adapt the network resources leveraging such monitoring data as well as the Service Level Agreements (SLAs), number of users and demanded network performance [4]. For such an automatic control and cognitive network management, at least a three phase process is necessary. In the first phase, network performance is monitored and the monitored information is collected. The second phase relates to analyzing collected information based on application requirements and deciding on network reconfiguration. In the last step, the network is actually reconfigured. Throughout time, network performance is verified according to application requirements.

The first phase depends on the network monitoring technique used. Current network monitoring techniques can be active or passive [5]. Active monitoring techniques inject additional traffic (such as probe packets) between endpoints in the network for measurement purposes. Contrary, in case of passive monitoring a central entity periodically polls network devices for performance statistics. Polling traffic can use off-the-band links, not impacting data flows, or in absence of control links, it will share the bandwidth with data traffic. Consequently, such traffic injection into the network will be bandwidth consuming, impacting traffic data flows. Passive based monitoring is limited only to network devices statistics, with no information from the end devices. Moreover, both techniques lack the flow specific monitoring information and in case of scheduled network, they do not show the real performance experienced by the data traffic. A number of monitoring protocols based on such techniques already exists, such as Net-Flow [6], IP Flow Information Export (IPFIX) [7], Simple Network Management Protocol (SNMP) [8], etc.

Until now such network monitoring techniques are used mainly for network troubleshooting purposes. For network performance verification finer-grained, higher flexibility, and timely monitoring techniques are required. Due to the continual nature of network performance verification, active monitoring techniques are not the most feasible techniques for this. Moreover, in wireless networks monitoring overhead should be minimal due to bandwidth limitations. Besides, the monitoring techniques should be able to adapt to wireless network dynamics and application needs, support wireless channel related parameters and detect reliability issues on per-hop basis. Last but not least, the monitoring techniques should give a holistic view on all network devices, down to end devices.

The concept of in-band network telemetry (INT) is a new

technique that offers low overhead monitoring possibilities. It overcomes the drawbacks of other techniques: enables end-to-end performance view of the network (including end devices), does not inject any new packets in the network and information is collected per-hop, per-packet, and on flow basis. Recently a draft proposal for INT standardization was issued by the Internet Engineering Task Force (IETF). The draft includes the telemetry data format [9] as well as INT encapsulation for different protocols [10]. However, currently, all standardization focuses on supporting INT for wired networks, mainly how switches and routers can collect and process telemetry data. Introducing in-band telemetry to wireless network needs new packet design, new inter-layer communication as well as new monitoring options for wireless-related parameters. Moreover, the reliability need to be collected on hop basis, that is not possible with current INT draft.

This paper investigates how the INT concept can be introduced to the wireless SDN and usage of INT data monitoring for network reconfiguration. This work is motivated by the need of (i) interaction between the application layer and INT layer to give freedom to the applications to control end-to-end monitored information down to end devices, (ii) low-overhead monitoring of wireless devices by limiting the impact of the monitored traffic on channel contention, (iii) real-time network (re)configuration and verification based on INT monitored data and (iv) per-hop reliability detection in addition to other wireless parameter monitoring. This work will contribute to each of the four mentioned challenges.

As an extension of the previous paper [11], the INT-node architecture is extended and unified no matter what the role of the INT node is. The API towards the application layer is extended with real-time and on-the-fly support for configuring the INT parameters, while the API towards the physical layer supports additional wireless parameters. Second, a new technique to determine reliability on hop-basis is introduced as part of the INT options. Last and the most important, the INT-based node architecture is integrated in the wireless SDN architecture where monitored data is used for network reconfiguration and verification. The whole INT node architecture is implemented on user-level space on top of Wi-Fi commodity hardware, and its integration with wireless SDN is verified in a real test-bed.

The remainder of this paper is structured as follows. In section II, related works in network monitoring and network management are presented. In section III we provide a short overview of INT concept for wired networks and we motivate the use of INT for wireless networks. In section IV we discuss wireless INT options, the design of INT-enabled node architecture, its proof-of-concept implementation, network controller implementation, and its interface with INT-enabled node. Based on the motivation and contribution given in this section, Section V validates the INT concept and its implementation. In section V, INT overhead and its impact on channel contention is analyzed. Further, the PoC INT-enabled node implementation and its integration with wireless SDN is validated in a real industrial test-bed where network reconfiguration and verification is done based on monitored data and the application layer requirements. Lastly, the technique to detect reliability issues on per-hop basis is validated and the convergence model of the detected link reliability to real link reliability is given. Finally, section VI concludes the paper and gives some possible future works.

II. Related Work

Efficient network monitoring is a cornerstone for reliable network management and network performance verification. Accurate and timely monitoring of the network will impact the accuracy of management applications in SDN-based networks. Currently, network monitoring techniques consist in either network device statistics polling [12], [13] or active probing [14]. Though they decrease the communication overhead using adaptive polling rate [12], [15] or probing rate [16], still introducing special traffic for network monitoring purposes impacts the network behavior. In addition, they are not sufficient in localizing the network performance problems [17]. Therefore, in-band monitoring techniques are more beneficial in reducing network overhead and providing end-to-end performance monitoring with no impact on network behavior.

Recently, network telemetry has gained interest in the research community. This is mostly related to the advances in SDN technology and the need for timely network monitoring and configuration. A number of studies on INT realization for different host environments and networks have been performed [18]–[21]. A number of demos have been shown too, illustrating how INT can be used to diagnose network problems as well as to improve network performance [22], [23].

Authors in [24] present a hybrid in-band and active probing monitoring framework for network function virtualization (NFV) monitoring. The IntOpt solution comprises of active probing between SDN controller and INT enabled network devices that will add the telemetry information to active probes. The P4-enabled programmable switches [25] are used to implement the INT-enable nodes in the network. Still, the presence of active probing will introduce limited overhead during network operation. In case of wireless networks, such approach will be less feasible as not all the nodes can have direct control links with the controller. Moreover, no application-INT layer interaction exists at the end nodes.

Similarly, in [18] an INT layer was implemented in Open vSwitch (OVS) by combining the P4 INT implementation with extended Berkeley Packet Filters (eBPF) [26]. The P4 INT implementation was used to generate the INT logic, while the eBPF was used to load the compiled code into specific points inside the Linux kernel. The implementation was evaluated in terms of CPU consumption when the OVS kernel was modified to include the INT layer. It was shown that the CPU overhead is 0.3% and 1% for data rates of 1 MBps and 100 MBps, respectively. Though a kernel level INT implementation will benefit from higher data rates processing possibilities, it lacks the flexibility of on-the-fly changing of the processing logic. The P4 INT implementation for network monitoring was used also in [23] and tested in simulator environment.

In addition to INT logic in network devices, timely INT information processing at sink nodes is crucial too. In [19] an implementation of an INT collector entity is presented. It is based on the extraction of network events based on the raw INT data. Therefore, the amount of logged data is reduced as well as CPU usage to process the data. Other INT collector architectures are presented in [21] and [27]. Authors in [20] present a network architecture for knowledge-defined self-driven networking. They implement an INT for an ONOS [28] controller using the P4 language [29]. The implementation is evaluated in simulator (mininet with bmv2) for its average processing delay and CPU usage.

So far, P4 language is the most widely used for INT node logic implementation by the research community. However, due to their kernel level implementation, designs ([18], [19]) lack the flexibility on adapting the processing logic on-the-fly based on the application needs. Moreover, current P4-based INT implementation does not support flexibility in choosing the flows which to be monitored. The INT data are added to each packet, increasing the overhead in the network [30]. On the other hand, other designs are evaluated only in simulator environment [20], [21], [23], [30] while all the cited designs focus only on wired networks. In this paper, we present an INT design for wireless networks. We implement its PoC in the Click router [31] framework, that is a user-space implementation that gives the flexibility of changing processing logic on real-time by offering application and INT layer interaction. Thus, the application layer can determine which data flows to be monitored and when. Such an implementation does not require any low-level changes either on the network stack or on the driver itself. In addition we validate the implementation on a real industrial test-bed. Compared to previous studies that are focused only on wired networks, or monitoring-only for wireless sensor networks [32], we focus on the validation of INT for WiFi-based SDN, believing that its adoption can greatly improve wireless network monitoring and network (re)configuration speed.

In addition to monitoring, network needs to be (re)configured based on monitored data and its application requirements. In this context, few frameworks capable of controlling and managing wireless networks [33]–[35] exists. End-to-end latency, end-to-end jitter, and throughput requirements are, besides challenging to be controlled, unknown due to coarse granularity of the monitoring information. In [36] authors propose an SDN-based approach that takes advantage of the SD-RAN architecture and the centralized view from the network resources to orchestrate slices and thus meet the QoS requirements. In this work, we will make use of the controller proposed in [36] for network (re)configuration based on INT monitored data.

III. In-band Network Telemetry (INT)

In this section, we will give a detailed introduction to in-band network telemetry, how INT headers are formed and processed. In the second sub-section, we will motivate the usage of INT in wireless networks and a number of use cases that will benefit from it.

A. Background

In-band network telemetry is a new way of monitoring for wired networks where monitoring information is appended directly to the data packets. The source node that generates the data packet attaches the initial INT encapsulation header as well as INT header. The INT encapsulation header format is composed of the INT header type, INT header length and the next protocol field [10].

Header type (1B) can take four different values based on the following INT header: pre-allocated hop-by-hop INT header type, incremental hop-by-hop INT header type, proof of transit INT header type and end-to-end header type. All three INT header types might be present at the same packet simultaneously. The ***header length (1B)*** contains the length of the INT header in 4 octets, excluding the length of encapsulation header itself. The ***next protocol (2B)*** field contains the protocol that follows the INT protocol header. If INT header is followed by another INT header then the next protocol field takes values for the next INT header type. If no other INT header is appended the next field specifies the next protocol that follows INT data options. The next protocol depends on the type of INT encapsulation. In case when INT is Ethernet encapsulated [10] next protocol field specifies IPv4 or IPv6, otherwise if INT is encapsulated as IPv6 hop-by-hop option then next protocol fields determines used transport protocol (e.g. UDP or TCP). To this end, the encapsulation in an available IPv6 option will not break the routing functions in non-INT enabled routers.

The INT encapsulation header is followed by the INT type header. In Figure 1, the INT encapsulation header is followed by the incremental hop-by-hop INT type header. The incremental INT hop-by-hop header contains the following fields: *namesapce ID*, *node length*, *flags* and *remaining length*. ***Namespace ID (2B)*** identifies the INT namespace that should be known by each node in the network. The ***node length (5b)*** field specifies the data length that each node will add as a multiple of 4-octets, that depends on the number of '1's in the trace type vector. The **flag bits (4b)** are used to specify if the packet has overflown (there is no space left to further add telemetry data), if the packet needs to be forwarded back to source (loop-back bit), or if the telemetry data needs to be processed immediately at every INT enabled node in the network. The ***remaining length (7b)*** field determines the remaining length of data that can be added by the intermediate nodes before the INT data options are considered to be overflown. The ***trace type (4B)*** field specifies the data types to be collected for the packet by each INT enabled node along the path. Practically, there can be up to 32 different data types that can be collected. According to the current draft [9] only 13 bits are used to collect information such as: ingress and egress port, received time, queue length etc. This paper will focus on hop-by-hop telemetry data. For the header formats of the other INT types we refer the reader to [9], [37].

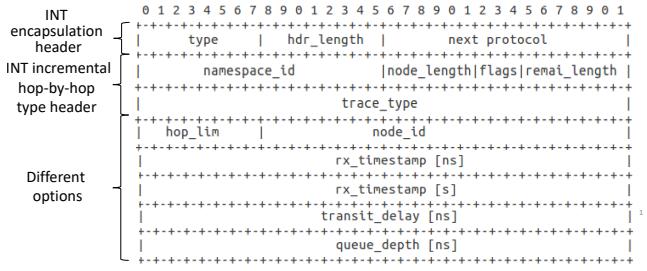


Fig. 1. INT information packet.

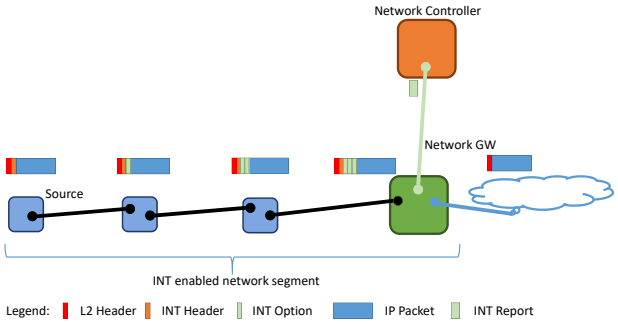


Fig. 2. Simple INT enable network example.

A simple INT enabled wired network example is shown in Figure 2. Depending on its type, INT data is processed by different nodes in the network that have different INT processing logic. The node that initiates the data packet we call as the INT source node and it is responsible for creating INT header(s). By adding the necessary information, INT source node determines the information to be monitored, flows to be monitored and INT frequency. For this an INT source node architecture should support interaction between application and INT layer. Intermediate INT nodes will add INT data based on the initiated INT header. At the end, the destination node or the INT termination node (network gateway) will extract the INT data. Such information can be used by other network entities, such as network controller, network optimization entity, or traffic visualizer entity.

INT specification was initiated by the P4 language consortium in 2016 [25]. Now, INT standardization is continuing under the name of In-situ Operations, Administration, and Maintenance (IOAM) by the network working group of IETF. It resulted in a number of drafts regarding different topics: the data fields for IOAM [9], Ethernet encapsulation of IOAM [10], and proof of transit [37]. Throughout this paper, INT and IOAM will be used interchangeably.

B. Towards INT for Wireless and its Benefits

In conventional networks, network monitoring was triggered by the need for network problem troubleshooting. Mainly active monitoring techniques were used at times when some miss-behavior of the network was observed. Different tools such as *ping* or *zing* [38] were used to determine round-trip time latency, end-to-end throughput, or packet losses. Such techniques are bandwidth consuming, and in case of wireless networks, will increase the channel contention between devices. Other approaches consisted of polling the network devices to retrieve various statistics. Such a technique is adopted by SNMP [8], that is widely used for network management and monitoring. SNMP uses a client/server approach, where SNMP manager device (client) polls the SNMP agents (servers) for device statistics. SNMP manager collects aggregated monitored statistics from network devices that are updated in certain time intervals. In addition, SNMP operates only on network devices, down to APs. Both of these elements can be a bottleneck for wireless devices. First, the aggregation statistics interval might be high to detect wireless channel issues. Second, the absence of monitored information from the clients, limits the network knowledge on the real client experience. Though some statistics are already collected on the AP side on client basis (e.g. number of tx/rx packets), other information, like e.g. latency or RSSI experienced at client side, can not be derived from those statistics.

Next to network management, a tighter network-application interaction will imply better fulfillment of application requirements. To this end, applications should be able to interact with the monitoring plane in order to set the monitoring parameters and targets. Thus an in-band monitoring approach will be beneficial in case of wireless networks in both directions. First it will overcome the problems with current monitoring techniques: collects information down to end-devices and offers a flexibility on collected information granularity. On the other hand, as the monitoring is extended down to the end device, INT offers the possibility for better application-network interaction where applications can trigger and manage monitoring themselves. We will describe a non-exhaustive list of use-cases where in-band monitoring is beneficial.

For instance, in many Industry 4.0 applications, in addition to problem troubleshooting, network monitoring is used to verify whether performance metrics are met or not, in order to avoid any down times. Active measurements by sending special probe packets, is not desirable, as they affect operations. Moreover, as the traffic is scheduled in time too, active measurements will not show the experienced performance of real data traffic. In addition, in different process control loop or emergency applications, strict end-to-end latency requirements should be maintained over time. As said, INT can monitor parameters on a per-hop basis. Thus, it permits to precisely determine the latency bottleneck in the network.

Robotic applications are characterized by network dynamics, where nodes connectivity and communication path changes over time. Path changes can not be detected using active measurements as they show only end-to-end statistics. Thus, INT monitoring techniques will enable per-hop statistics, showing the network segments with performance issues as well as timing of such issues.

Recently, time sensitive networking (TSN) is becoming a standard for time-critical industrial applications. Time-sensitive networking concept is making through to the wireless world too, mainly driven by private 5G networks. In order

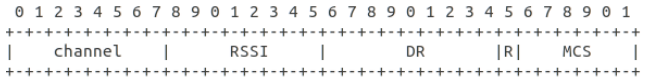


Fig. 3. Wireless link telemetry info option format.

to maintain the performance of critical processes, network must be monitored over time. Similarly, in this case, in-band and low overhead monitoring techniques will be the best option. First of all, monitored packets will experience the same performance as data packets, will follow the same schedule (in time and frequency), and will offer end-to-end insight into the performance including end devices.

IV. Wireless INT Design and Implementation

The design of INT-enabled network and its implementation, as outlined in this section, targets the usage of INT in wireless networks, currently focusing on WiFi. In the following subsections we will describe the wireless telemetry options, INT-enabled node architecture and its PoC implementation, as well as network controller used for the network verification.

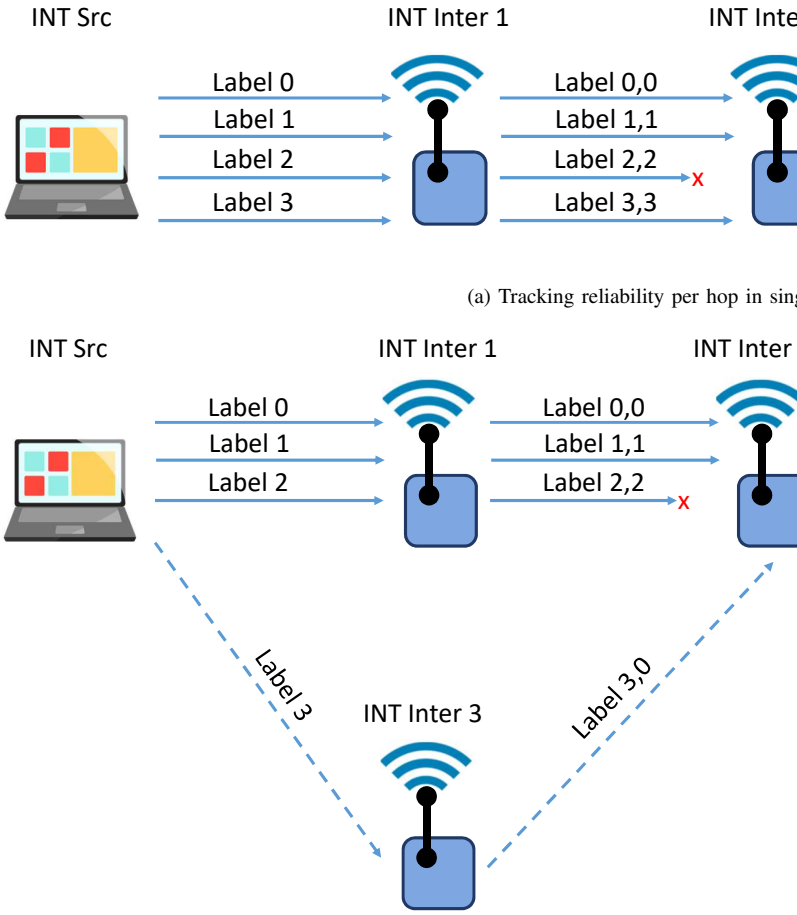
A. Telemetry Options

The proposed design enables the collection of telemetry data related to wireless links. Another aspect it considers is the ability to monitor per-hop reliability, in addition to end-to-end reliability. To achieve this, two new hop-by-hop options are defined, the presence of which is indicated by bits 12 and 13 in the hop-by-hop trace type bit vector.

1) Wireless Telemetry Option: Depending on the openness of the wireless card's driver, various information about the wireless link can be collected, including: RSSI, SNR, MCS, contention window size, channel info, data rate, and reception time. For now, we only consider RSSI, data rate, channel info, reception time, MCS index and retransmission flag.

The wireless link telemetry option format is shown in Figure 3. The reception time is included by setting bit 2 and 3 of trace-type, for setting the second and sub-second part respectively. The ***channel (1B)*** field specifies the wireless channel at which the link is operating, the ***RSSI (1B)*** field specifies the received signal strength of the packet, the ***DR (9b)*** field specifies the data rate used, the ***R (1b)*** flag specifies if the packet was retransmitted or not, while the ***MCS (6b)*** specifies the modulation and coding scheme index used for the packet. All wireless options together are 4 bytes aligned for easy processing.

Currently, we use layer 2 encapsulation of INT information, with the INT header and option data field residing between layer 2 and layer 3 headers. This way, also APs can add INT data to data packets between wireless nodes connected to the same AP, that is layer 2 communication. As a downside, packet processing problems will occur in case the INT enabled packet traverses a router that is not INT enabled. To this end, we will limit INT usage in private networks, where single subnet is used and packets do not traverse any router.



(a) Tracking reliability per hop in sing

(b) Tracking reliability per hop in diffe

Fig. 4. Tracking INT reliability per hop in different scenarios.

2) Per-hop Reliability State Option: The current INT standardization draft only supports end-to-end reliability monitoring [9]. This is achieved by including a counter option in the end-to-end INT type header. INT sink node will track the counter and the number of INT packet loses can be calculated at any time. However, this method cannot determine at which hop the packet was lost. Therefore, we propose to add a per-hop reliability state option in the hop-by-hop INT type. This option contains a single 4-bytes ***label*** field that is used by each intermediate node to count packets per flow. So, up to $2^{32}-1$ continuous lost packets can be detected. To differentiate between different flows, the namespace ID is used. Thus, each device can handle up to 2^{16} different data flows at any time.

The collection of per-hop reliability is explained in Figure 4. Every node, including the source node, will use the ***label*** option to add an incremental label to every packet. In the example shown in Figure 4a, the first two packets arrive successfully at the sink node. The third packet gets lost between the first and second intermediate node. Upon arrival of the fourth packet at the sink, the recorded labels are used to detect the loss and its location. As the incremental label has

'discontinuities' the actual loss happened at the *'discontinuity'* link, in this case at the second link.

Per-hop reliability monitoring can be achieved in the presence of path changes too, as shown in Figure 4b, where the flow changes the path right after an INT packet was lost on the previous path. As, next to the labels, also the hop-by-hop node IDs are monitored, the sink can detect the path change. The last packet that is received in the sink node has the labels from three different nodes (3,0,2) (Figure 4b). Based on collected node IDs, the sink node detects that the path has changed, and thus the label is reset. However, since the second intermediate node was part of the previous path too, the received labels in the current packet should have been (3,0,3). As the label is different, the sink node detects that the packet loss occurred between the first and second intermediate node.

B. INT-enabled Node Architecture

The INT enabled node architecture is shown in Figure 5. The INT *layer* resides between layer 2 and layer 3 and is responsible to add/extract/process the telemetry info. The INT layer is composed of three engines: INT source engine, INT inter engine and INT sink engine that implements the logic for INT packet processing based on the node role. In addition to INT engines, the INT layer communicates with the application layer using a ZeroMQ[1] messaging entity to share the information regarding the flows to be monitored.

As shown in Figure 5, via the ZeroMQ messaging entity the INT layer becomes aware of the flows that need to be monitored by getting flow specific information, like: flow ID, source and destination port and IP address, INT frequency and trace type bit vector. This information is necessary for the INT source engine to initialize the INT header. Such a design gives the opportunity to application layer to interact with INT layer. To this end, application layer can adjust the INT data granularity by adjusting on the fly the INT period based on traffic flow requirements. In addition, the trace-type bit vector can be configured by application layer too by adding/removing certain bits based on the needed information. However, no matter the trace-type imposed by the higher layers, INT source engine will always set at least the node ID bit. Last but not least, application layer can add/delete flows to be monitored using the ZeroMQ messaging entity.

The pub/sub messaging model was chosen to keep the node architecture agile and dynamic. To this end, the application layer and the INT layer is kept decoupled and message polling is avoided. Contrary, adoption of other messaging models, like e.g. push/pull, would decrease the efficiency of packet processing as for each data packet INT layer would need to pull the application requirements from the socket. The adoption of a dynamic API between application and INT layer based on pub/sub model, enables the application layer to change in real-time the INT parameters to be monitored and the frequency of monitoring. In addition, ZeroMQ is chosen as it is a lightweight implementation of a messaging library that supports different models on how messages can be shared. Differently from the most used and well-known messaging protocol of MQTT[2], ZeroMQ can support brokerless, broker-based and distributed broker pub/sub model. For our communication between application layer and INT layer we use ZeroMQ broker-less pub/sub messaging model that will avoid a single point of failure in the stack.

If the node is an intermediate node, it only adds telemetry data. For each packet that arrives from lower layers, INT layer determines if the packet is for itself or needs to be routed further. If the packet needs to be routed further then the packet will be processed by the INT intermediate engine. Differently from INT source engine, INT intermediate engine will access the wireless link information from the wireless card itself and create the wireless option header. For this the INT layer should have an API for the telemetry info it wants to access, e.g. API for queues state, API to access other low level wireless info and reception time, etc. This API is shown as *Phy API* in Figure 5.

Finally, if the packet has reached the destination and is an INT-enabled packet it will be processed by INT sink engine. The INT sink engine extracts the INT data and sends the data to different entities in the network, such as a network controller or visualizer. The parameters that can be configured at the INT sink node are the IP address of the controller(s)/visualizer(s). Additionally, various APIs can be used to communicate between the INT sink and controller(s), e.g. push/pull, broker-based communication, and request/reply. Such an API is named *NC API* in Figure 5. The INT sink node can concatenate a number of telemetry measurements in order to decrease the traffic towards the network controller(s). It can even process the INT data locally and share only the data that satisfies certain conditions, e.g. latency or losses higher than a threshold.

C. Proof of Concept (PoC) Implementation

The Click Router framework [31] is used to implement a proof of concept (PoC) of our proposed INT enabled node architecture. As we opt for the dynamic interaction between application and INT layer, and the ability to change on-the-fly the monitored information based on application needs, Click router offers this flexibility. Click Router is a modular software router toolkit that enables packet processing in user-level space [31]. In addition to our custom Click router extensions, we implemented three new Click elements that represent three different INT engines: INT source, INT intermediate and INT sink engines. Each of them can be used in the same node and process the packets based on traffic flow information, as shown in Figure 5.

Click router exposes a TUN interface towards the application layer. A TUN interface is a kernel network layer virtual network device. It operates at the network layer, accepting only IP packets. Thus the application layer sends directly all its traffic through this TUN interface. Once the packets have

[1]https://zeromq.org/

[2]https://mqtt.org/

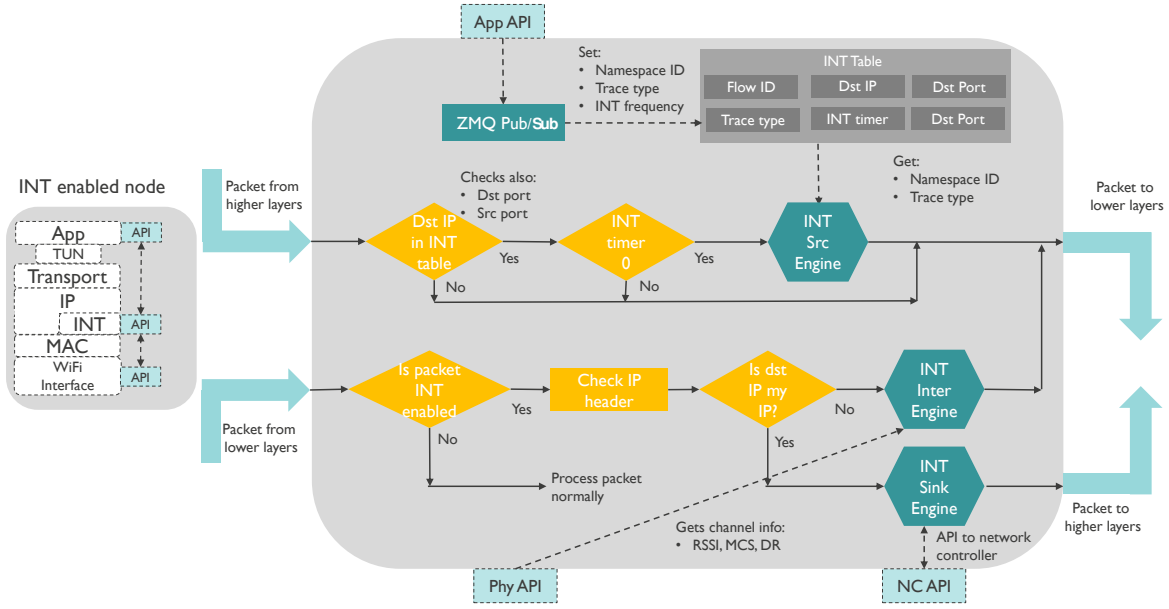


Fig. 5. INT enabled node architecture.

been processed by the Click framework and their next hop has been determined, they are processed by the INT layer. The INT layer will add the INT header in between layer 2 and layer 3. The interaction between INT layer and application layer is implemented using ZeroMQ pub/sub messaging API, as designed in previous section. INT source Click element consists of two main functions. The first function, named *IntSrcEngine::SetNS()*, keeps track of the registered flows from the application layer that need to be INT enabled. This function labels all the packets coming from TUN interface that need to be initialized with INT header. The second function, named *IntSrcEngine::InitHdr()*, checks for the labeled packets and initializes the INT header based on the information taken from the INT table.

The wireless interface is set in monitoring mode in order for the INT intermediate engine to extract physical layer data. Physical layer parameters, like RSSI, data rate, MCS index, channel, retransmission flag and the reception time are extracted from the radio tap header and saved in a meta-data annotation that is appended to the packet. Thus, the *Phy API* in Figure 5 is implemented using monitor mode of the wireless interface. If the packet needs to be forwarded to another node in the network, the INT hop-by-hop type header is processed first. Based on the trace type the required information is added in the INT data option space. At this point, the related wireless info is taken from the appended meta-data of the packet and the meta-data are cleared. Finally, the packet is forwarded to the selected outgoing interface.

The INT data collection can be terminated either at the destination node or at a gateway node at the border of an INT enabled network segment. INT sink engine will process all the INT information and create a JSON data structure that includes the information in a structured way. The INT sink Click element consists of three main functions. The first function, named *IntSinkEngine::ProcessInt()*, processes the INT end-to-end header type, while the *IntSinkEngine::ProcessIntHbH()* function processes the INT hop-by-hop header type and all the added information. The last function, *IntSinkEngine::CreateJSON()*, creates a JSON structure out of the INT information and, using the NC API, shares it with the other interested entities in the network. In the current implementation, such data structure is published to a central broker using a pub/sub API. Other network entities (network controller, visualizer) subscribe to this central broker and will get the collected telemetry data.

D. Network Controller and Visualizer

In the validation section, we will use a network controller that will take decisions based on collected monitored information using INT. As such, we will validate the wireless INT concept and its implementation. Therefore, for the sake of completeness, in this section we will describe the used network controller. The network controller prototype is designed as a modular application with independent and integrated functionalities for SDN monitoring, visualization, and configuration. An existing implementation of an SDN platform for heterogeneous RAN, called 5G-EmPOWER [34], is extended. The 5G-EmPOWER platform is composed by a few components including an SDN controller and an agent, called 5G-EmPOWER Operating System (OS) and 5G-EmPOWER agent.

The SDN controller, besides comprising the network intelligence and abstraction of the network elements in the infrastructure layer, it also implements the interface to communicate with its agents. Such an interface enables communication between the 5G-EmPOWER OS and the 5G-EmPOWER agent

and it is implemented using the OpenEmpower protocol [39]. The agent allows to manage radio access nodes and expose the current statistics towards the controller. However, such statistics do not include end node statistics and do not contain flow related information. Therefore, to be able to recognize and, hence, add INT measurements into data packets, we have extended the agent implementation. In this manner, packets with INT measurements reaching the sink node comprise the monitoring information of the entire flow.

The framework adopts the network slicing concept for the wireless access segment where a specific portion of airtime is allocated to each slice at each transmission round of the scheduler. First, frames are classified into the different queues as slices, based on the definition of traffic rules (e.g., OpenFlow rules [40]). Then, frames belonging to such slices/queues are dequeued following the Airtime Deficit Weighted Round Robin (ADWRR) scheduling algorithm [41]. The ADWRR scheduling algorithm is responsible to assign a fraction of the airtime (*quantum value*) to each slice/traffic rule according to its relative priority. The idea is that, at each round, the scheduler will scan all the slices for any packet to be sent. Once it finds a slice with a non-empty queue, it will increase the deficit counter of that slice by its quantum value. Then the packet air time is calculated (based on the data rate used for the link conditions). If the packet air time is lower than the accumulated deficit counter, the packet is being transmitted otherwise it will wait for the next round. Everytime a packet is transmitted the deficit counter is decreased by the packet airtime. Once the slice queue is empty or the calculated packet air time is longer than the accumulated deficit counter the scheduler will pass to the next queue. Therefore, when one slice has low quantum it gives the possibility to other slices with higher quantum to transmit more often.

Conceptually, network slicing is the abstraction that can provide precise resource allocation and traffic isolation among users and services. Authors in [42] present two variants of network slicing. The first abstracts the different services and ensures Quality of Service (QoS) within them, called Quality of Service Slicing (QoSS). The second defines slices as the traditional idea of network virtualization where a precise subset of network resources is allocated to each tenant and full control is provided, called Infrastructure Sharing Slicing (ISS). In our work, we focus on QoS within a slice as being a service, hence, we use the QoSS definition.

For visualization, we use an open-source time-series data visualization platform TICKStack [43]. TICKStack supports different adapters to interface with data series. We used PUB/SUB API to publish JSON data structures to the TICKStack broker. Then data were visualized using different graphs by only feeding in certain fields from the JSON structure. TICKStack has another benefit that different alarm thresholds can be set in order to detect any misbehavior in the network.

V. Validation

To validate the proposed design and implementation, we first evaluate the INT overhead to data packets. We also benchmark the INT overhead to active probing overhead in terms of measurement reports produced for the same number of channel contention. Secondly, we validate the implementation of the INT node architecture in a real industrial environment testbed [3]. The validation encompasses the detection of wireless network behaviour based on application-driven INT monitoring data collection. Such measurements are then used by the network controller on decision making for re-configuring the network. We further validate detection of the reliability issues on per-hop basis compared to ground truth in an emulated environment. Lastly, we determine how fast the INT reliability measurements converge to the real link reliability when using different INT frequencies than the data frequencies.

A. INT vs Active Probing Overhead

INT usage does not come without packet overhead. The INT overhead depends on the number of hops packet traverse, number of collected information, the option format and the way how INT is added. Nevertheless, it is minimal compared to active probing as no additional packets have to be generated. INT overhead as a function of communication hops can be calculated according to the following formula:

$$\begin{aligned} INT_{overhead} &= 4 + 8 * INT_{HBH} \\ &+ INT_{HBH} * (h * 4 * TRACE_{LNGT}) \\ &+ INT_{ETE} * (4 + 4 * E2E_{OPTIONS}); \end{aligned} \quad (1)$$

where INT_{HBH} is 1 when the hop-by-hop header is present, h is the number of hops the packet has passed, $TRACE_{LNGT}$ is the number of set bits in the trace type byte vector in the INT hop-by-hop type header, INT_{ETE} is 1 when the INT end-to-end header is present and $E2E_{OPTIONS}$ is the number of options in the end-to-end INT header. The unit of the formula is in bytes.

In order to monitor the performance of wireless links, we have to enable at least three bits in the trace type: the node ID option, the wireless telemetry option and the hop-by-hop reliability option. Next to this, the end-to-end header should be present with the counter option. So, the INT overhead on data packets for monitoring of wireless links will be:

$$\begin{aligned} INT_{wireless} &= 4 + 8 * INT_{HBH} \\ &+ INT_{HBH} * (h * 4 * 3) \\ &+ INT_{ETE} * (4 + 4 * 1) \\ &= 20 + 12 * h \end{aligned} \quad (2)$$

In case when active probing is used there will be an additional probe request/reply packet at each wireless link. For IEEE 802.11, this is further increased by two acknowledgements (ACKs) at layer two, two Request-to-send (RTS) and two Clear-to-send (CTS) packets (assuming the RTS/CTS mechanism is used). Ignoring the length of the probe request/reply packets, the layer 2 overhead to transmit the probes over a single link will already be: 2*20 bytes L2 ACKs, 2*20

[3]https://doc.ilabt.imec.be/ilabt/wilab/

Hops	1	2	3	4	5	6
Data space left for MTU of 2304 B	2272	2228	2172	2104	2024	1932

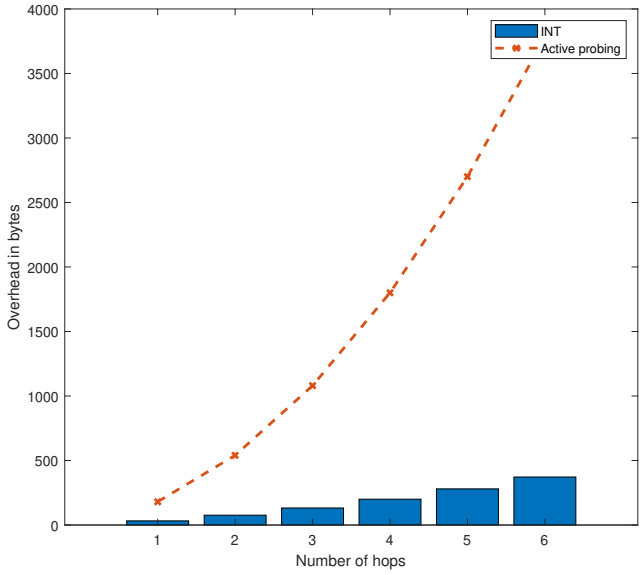


Fig. 6. Overhead comparison between INT and active monitoring for different number of hops. The table shows the data space left for the IEEE 802.11 MTU of 2304 bytes.

bytes for RTS and 2*20 bytes for CTS, 60 bytes in total. If INT is used for a single hop, only 32 bytes overhead to data packet will be introduced. So, the usage of INT will decrease the monitoring overhead by $\sim$ 50% (not counting here the probe request/response length of packets). For a ping request/reply packet, the ICMP payload can be as low as 20 bytes, resulting in a total of 60 bytes, counting for IPv4 and layer 2 headers too. So, over a single link, an additional 180 bytes are sent if active probing is used, compared to only 32 bytes of INT. This is around 6 times less overhead. In case of multi-hop communication, this percentage is increased further as shown in Figure 6, e.g. for 6 hops, INT has $\sim$ 8 times lower overhead compared to active probing. The table in Figure 6 shows the remaining space for data in the packet for a maximum transmission unit (MTU) of 2304 bytes (IEEE 802.11 MTU).

Compared to the active probing techniques, INT outperforms in terms of overhead. However, INT overhead related to the data packet itself depends on how often INT information is added, what is the data-rate of the traffic flow, the number of hops packet traverse, and actual data packet length.

Figure 7 shows the INT overhead for different parameters configuration. When data packet length is small then the INT overhead in the whole packet is larger compared to larger data packets. The INT overhead is increased with the number of hops too. In Figure 7a it can be seen that for small packets of 100 bytes the INT overhead can be up to $\sim 175\%$ of the data packet length for 8 hops. But this overhead is much smaller for packet length of 1000 bytes (only $\sim 15\%$). INT period unit can be in time units or in number of packets. In the later case, the INT overhead will depend on the data rate of the traffic flow. In case of the period expressed in number of packets, INT overhead depends only on packet length, but not on data rate. This two cases are shown in Figures 7b and 7c, respectively. When INT information is added to each packet (though this is not realistic), the number of supported flows in the same path will decrease due to the INT overhead. Such information is shown in Figure 7d. For flows with shorter data packet length, the INT overhead will be higher, decreasing the number of supported flows more. Such information is independent of the data rate that is used by data flows.

In wireless communication, it is important to keep the channel contention low. The channel contention relates to the number of channel access required by the nodes in a certain amount of time. In addition to pure decrease of link overhead (bytes sent), INT decreases the number of channel access by avoiding the generation of additional packets, like probes, ACKs, RTS, and CTS packets. This decreases the contention in the channel, improving link performance in terms of throughput as well as latency. In case of active probing monitoring techniques, the number of channel access per link is increased when new packets are introduced in the network. On the other hand, for INT, those L2 packets are generated only for the data packet itself, keeping the channel access overhead low. Thus, we want to quantify the number of INT reports that we can generate for certain application data rate compared to active probing, by keeping the number of channel access the same as when MTU packets are sent.

For this, we perform the following analysis. Let DR_i be the *i-th* application datarate, h_j number of hops used in network and L the maximum packet length in the network (IEEE 802.11 MTU is 2304 B). The number of INT-enabled MTU packets, X, that can be sent over a unit of time by the application i will be:

$$X_i = \frac{DR_i}{L} \tag{3}$$

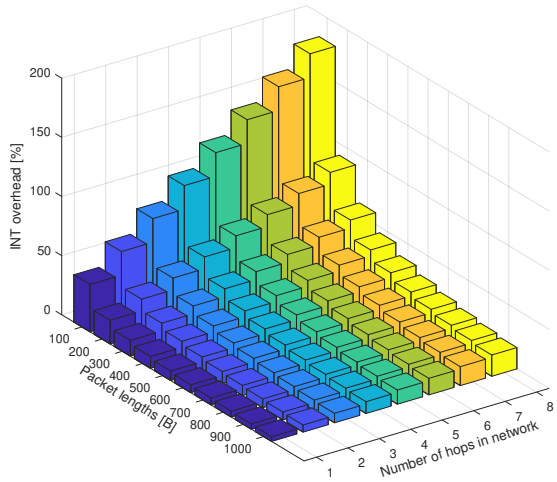
When no INT is added to the packet then the number of packets that can be sent will be higher:

$$Y_i j = \frac{DR_i}{L - INT_{overhead_j}} \tag{4}$$

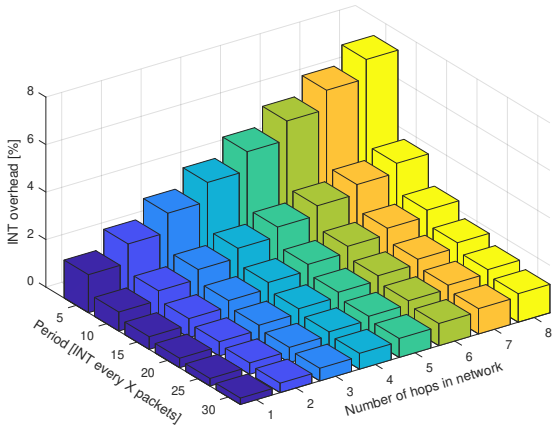
where j is the number of hops in the network. The channel access requirement per time is the same as the number of packets generated per time. To keep the channel access for data packets the same, in case of active probing, the number of reports will be:

$$Z_{ij} = \frac{Y_i j - X_i}{2} \tag{5}$$

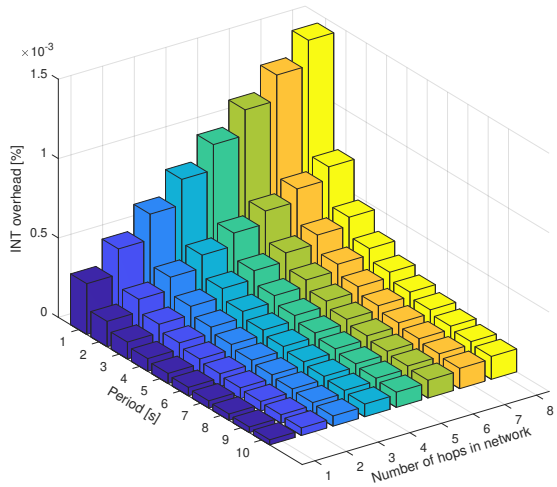
A factor of $1/2$ is taken as for each probing report at each link we need single channel access for sending the probe and another one for receiving the reply. Then the difference of reports that can be generated using INT compared to active probing over a unit of time for the same number of channel access for data packets will be $Rep_{ij} = X_i - Z_{ij}$, i



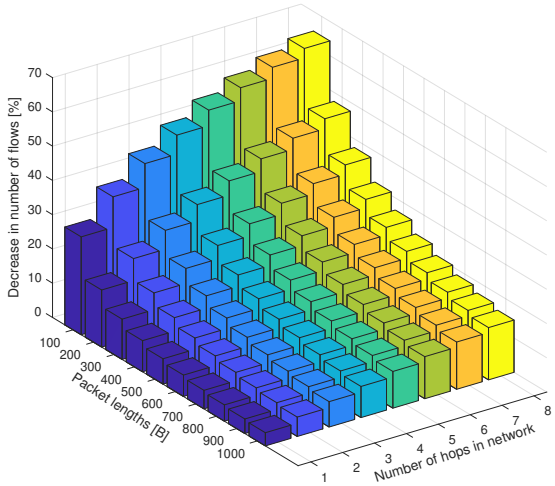
(a) Overhead vs different packet size and number of hops.



(b) Overhead vs different packet period and number of hops. Fixed data packet size of 500B.



(c) Overhead vs different INT period and number of hops. Fixed data rate of 100Mbps.



(d) Decrease in number of flows vs different data packet length and number of hops.

Fig. 7. INT overhead for different parameters configuration.

being the application and j the number of hops. In Figure 8 the Rep_{ij} matrix is plotted. For highly intensive application requirements (like motion control) we can send between ~ 25 to $\sim$ 30 (number of hops 1 to 8) more telemetry enabled reports compared to probing reports in a unit of time. When the application data rate requirements are lower, then the difference decreases too (like in the control loop use case of 0.5 Kbp). As higher the data-rate requirement by the application, as higher the difference will be. However, the difference between the number of hops is not significant, as seen in Figure 8.

B. Real Time Network Monitoring

To validate INT layer implementation, we monitored our industrial WiFi network in the IIoT test-bed [4] for a period of 1 hour. Wireless network under test shared the same channel with 3 other networks in the test-bed. The goal was

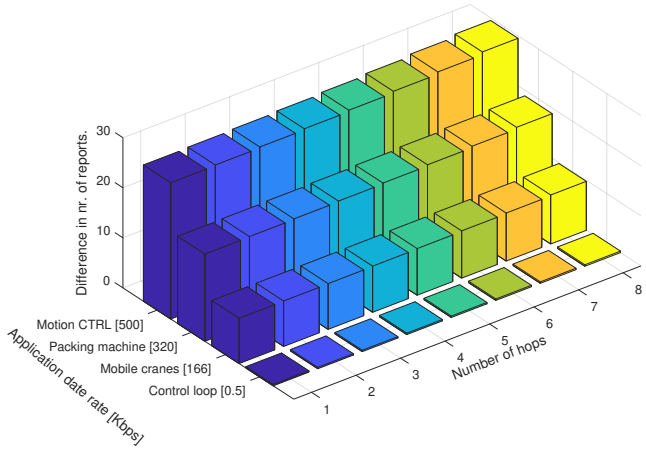


Fig. 8. Difference between number of INT reports and active probing reports for different application requirements [2] keeping the number of channel access requirements the same for data packets.

[4]https://doc.ilabt.imec.be/ilabt/wilab/

to verify and validate whether we are able to identify link quality changes using INT measurements for INT-enabled end devices, when new data flows are introduced in the network in addition to critical flows. Therefore, we generated two different traffic flows between two nodes in our network and we introduced INT to both flows. The first flow (flow A) represents a time critical flow requiring latency lower than 10 ms, while the second flow (flow B) latency requirement was less critical, lower than 30 ms. Iperf [5] was used for traffic flow generation. Both flows were generated for 10 minutes while for the next 3 minutes there was no traffic. Such a period of enabling/disabling traffic was run for 1 hour. Both end devices and the AP to which they were connected were INT-enabled devices. INT information period was set to 5 seconds. In this case, we will focus on the end-to-end latency performance, rather than in throughput itself. Application requirements are compared to monitored data to take the decision on network reconfiguration.

Besides the benefits of the SDN central point of view for network monitoring purposes, we took advantage of it to perform network slicing. In this manner, we make use of the INT measurements to configure the network slices and verify the impact of such re-configurations in real-time. As discussed in section IV-D, the network controller has full control over AP where slices can be instantiated, modified, and deleted whenever is needed. It has to be stressed that the time slicing is done in the upper MAC layer and it applies only to downlink in AP. Still, both flows will share the same hardware queue of the network interface card (NIC). We defined two configurations that the network controller can perform. First, dedicated slices with equal airtime configurations are created for both flows in the AP. Equal portions of airtime, *quantum* values, are allocated to each slice in each transmission round [41]. For all slices, such quantum values are initialized with the same value of 12000 µs, meaning that slices equally share the resources on the AP. This initial quantum configuration for all slices is set to the time needed to transmit a standard Ethernet frame (MTU of 1500 bytes) at lowest data rate.

At each slice reconfiguration interval, the network controller re-configures the slices within the AP based on feed-in data from INT. We adopted a policy as proof-of-concept where, in this case, the airtime configuration of slice without QoS is decreased, consequently, leaving more consecutive transmission rounds to the slice with QoS. In this manner, the airtime configuration for the slice of flow A is maintained while the configuration for the slice of flow B is decreased to a minimum defined value of 10 µs. Thus, giving higher priority to the flow A and meeting its requirements. When QoS requirements are met for flow A, slice quantum configuration for the flow B is increased by 10% of its value at each reconfiguration interval by making sure that such increase does not impact the QoS flow. Of course, in this case the throughput for the second flow will decrease, as theTable I presents the workload and all configuration changes performed during the evaluation period.

[5]https://iperf.fr/

The end-to-end latency for both flows during a period of 10 minutes is shown in Figure 9. In the first 3 minutes, the network controller was using the INT data to determine whether the application requirements are met. As end-to-end latency requirement is not met, after 3 minutes the network controller takes the decision to adjust the configuration of the slice competing for resources with the QoS slice. As we can see, after such slice reconfiguration, the end-to-end latency decreases under the required levels for the first flow (flow A). It is interesting to notice that, the end-to-end latency from both flows (flow A and B) decrease when such reconfiguration is performed. Such a behavior happens because the airtime portion of the slice dedicated to flow B is decreased and, consequently, the scheduling algorithm running on the AP needs more iterations to gather sufficient airtime to transmit its frames. As a result, due to upper MAC queue overflow, some of the incoming frames are not enqueued and, thus, flow B throughput will decrease. In other words, fewer frames are sent to the NIC from flow B, making NIC available for longer periods for flow A. Consequently, there is less contention for both flows, improving the latency on both flows on the same AP (flow A and B).

Figure 10 shows collected information for each hop and each flow. Here we show the retransmission flag, MCS index used, and the RSSI value of the packet. It is seen that when the number of retransmissions is high (see first hop information, Figures 10a and 10c, respectively) the MCS index is lower. While in the second hop (see Figures 10b and 10d, respectively) the number of retransmissions during the whole experiment duration is 0. This makes that in second hop the MCS index to be higher. Both of these information can be related also to the RSSI values shown in the graphs. The second node has higher RSSI value compared to the first node, therefore better MCS index and no packet retransmissions. Thus, wireless INT can detect the link quality changes in each hop. It has to be highlighted that the second hop has better channel performance due to the node positions in the test-bed, shorter distance. This can be also detected based on the monitored RSSI values.

TABLE I
WORKLOAD AND PARAMETERS USED DURING THE EXPERIMENTATION

Parameter	Value	Description
Slice reconfiguration interval	3min	Interval in which controller averages over INT data to take decision for reconfiguration of slices.
Minimum quantum configuration	10µs	The minimum quantum value a slice receives at each transmission round of the scheduler.
Quantum increase rate	10%	The quantum increase rate applied whenever QoS requirements are met.
INT measurement interval	5sec	Interval in which INT information is added into data packets.
Traffic duration	10min	Duration of the UDP traffic in which INT information is added.

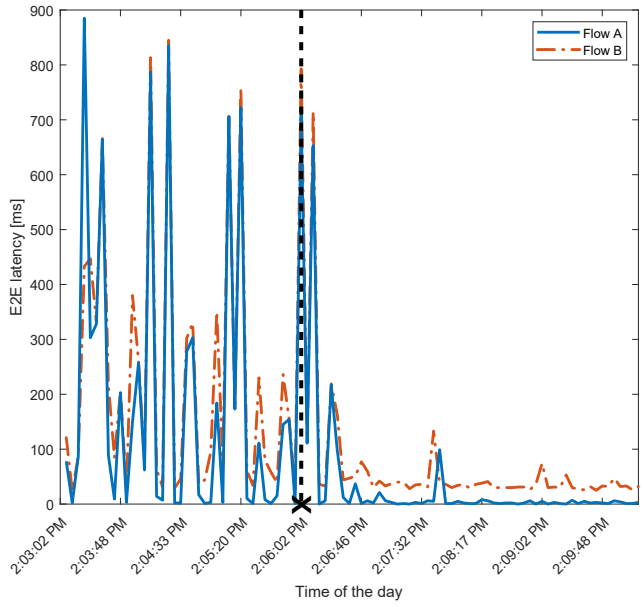


Fig. 9. End-to-end latency for two flows before and after slice configuration. The vertical line marks the reconfiguration moment.

C. Hop-by-hop Reliability

In order to validate per-hop reliability monitoring capabilities of our INT layer, the setup shown in Figure 4a was used. Each link between all involved network entities was set to have a packet loss ratio of 10%. In total, 2000 data packets were sent with a 0.5 second interval between consecutive packet transmissions. INT information was added to every generated data packet between the source and the sink. All data packets are treated to be part of the same data flow, sharing the same namespace ID. All measurements were done using an emulated link connection in Click router framework. By using emulated links, we have the possibility to also track the packet losses at each link and compare them to the collected INT reliability data. Table II shows the number of lost packets reported by INT and the one reported by the emulated link Click element. It is seen that all losses are correctly reported by INT on a per-hop basis.

TABLE II
FLOW REQUIREMENTS

Hop number	Lost packets reported by INT	Lost packets ground truth
Hop I	210	210
Hop II	160	160
Hop III	154	154

D. Convergence of the INT PLR to the Link PLR

In order to limit monitoring overhead, INT information does not have to be appended to each data packet that traverses the network. Applications can require to add INT information to data packets at specific periods, maintaining a certain monitoring information granularity. In terms of network reliability, this might result in incorrect packet loss ratios. This depends on the frequency of data transmissions and the actual frequency of adding INT data.

Let p be the probability that a packet is lost, T the time period over which the packet loss ratio is calculated and t the data packet period. We want to determine how long we need to average the losses of the INT packets, T_{INT}, in order for the INT packet loss ratio to reach the actual data packet loss ratio. The number of packet losses over a data period T can be calculated by the following equation:

$$PL_{data} = p * \frac{T}{t} \tag{6}$$

We aim to let $PL_{data} \approx PL_{INT}$, after a certain time, thus:

$$p * \frac{T}{t} \approx p_{INT} * \frac{T_{INT}}{t_{INT}}; \tag{7}$$

When INT data are added periodically the probability to lose an INT packet will be similar to the probability of losing a data packet, thus $p \approx p_{INT}$, when the data period is longer than the burst error length. From equation 7 it can be determined for how long INT packet losses need to be averaged in order to converge to the real data losses:

$$T_{INT} \approx t_{INT} * \frac{T}{t} \tag{8}$$

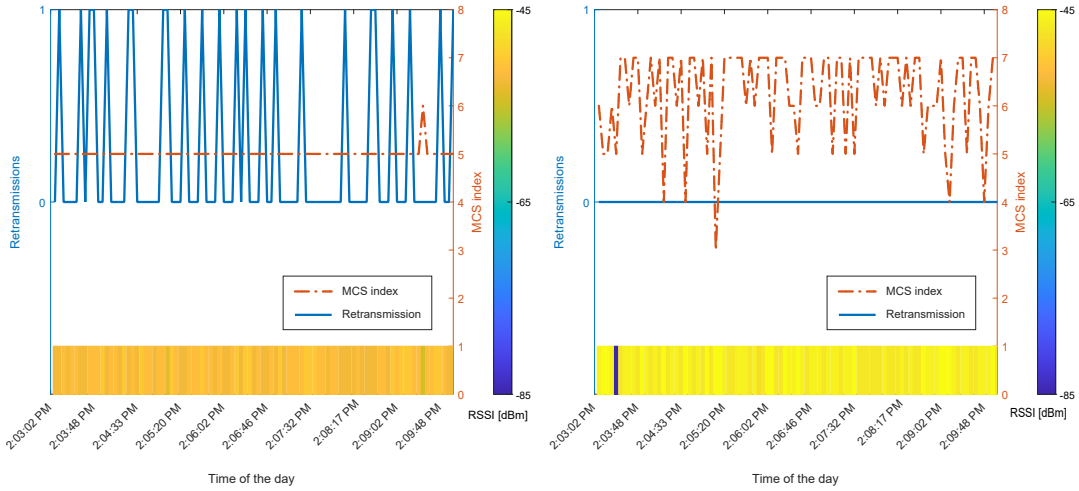
From equation 8 it can be concluded that in order to converge to the actual data PLR, the averaging window for the INT losses needs to be proportionally increased according to the selected INT period.

We evaluate the above equation by means of a set of measurements for a link with a packet loss ratio of 10% and data being generated every 0.5 seconds. The convergence of the INT PLR to the real data PLR is shown in Figure 11. In Figure 11a, when the INT information is added to every data packet, the right PLR is found after averaging the collected monitoring data for around ∼250 seconds. When the INT period is increased to 1 second, the convergence time is doubled, becoming ∼500 seconds. The same happens for an INT period of 2 seconds, where convergence is achieved after ∼1000 seconds. For other INT periods, (4, 6 and 8 seconds, see Figure 11b) the PLR does not converge even at 1000 seconds. According to the formula, it will converge around ∼2000 s, ∼3000 s and ∼4000 s, respectively.

VI. Conclusions and Future Work

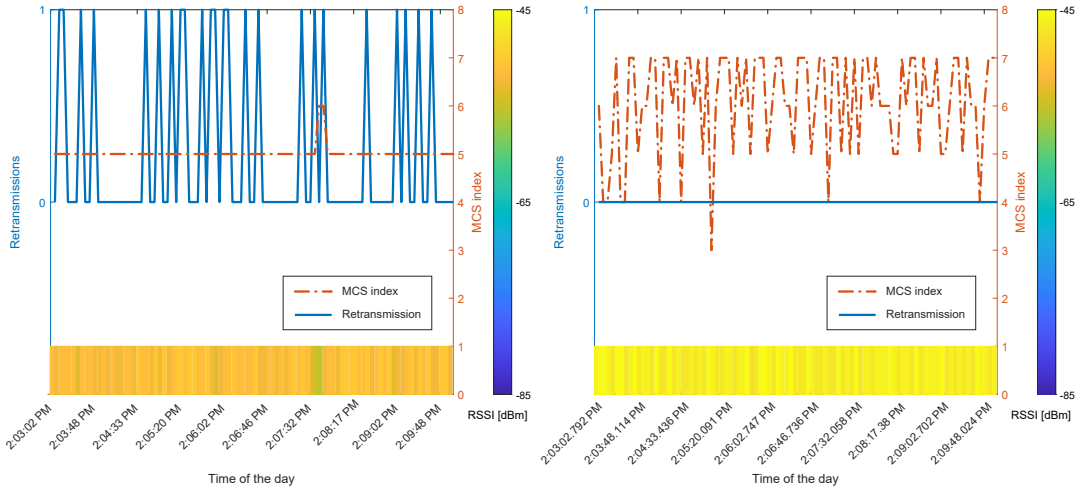
This paper addressed the problem of low-overhead, fine-grained down to end devices monitoring technique for the wireless networks. Inspired by the in-band network telemetry approach, INT is introduced to wireless SDN, overcoming the drawbacks of current monitoring approaches.

A telemetry-enabled wireless node architecture was designed, that is able to collect wireless information. On top of this, per-hop and per-flow reliability can be collected with the proposed labeling method. In addition to telemetry functionalities, the proposed design of telemetry-enabled node



(a) Measured parameters for the first flow in node 1.

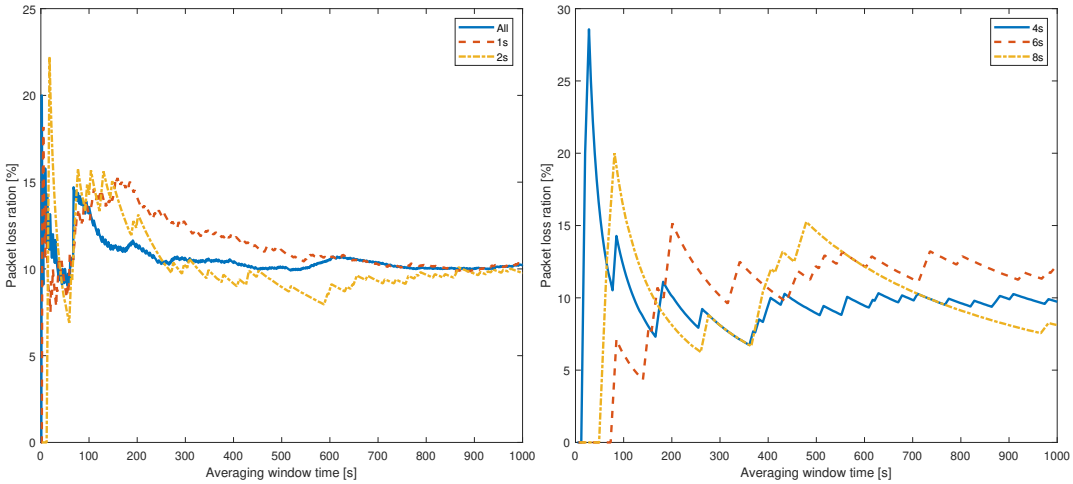
(b) Measured parameters for the first flow in node 2.



(c) Measured parameters for the second flow in node 1.

(d) Measured parameters for the second flow in node 2.

Fig. 10. Measured parameters for different flows in different nodes using INT.



(a) Addition of INT data every 1s, 2s and every packet.

(b) Addition of INT data every 4s, 5s and 8s.

Fig. 11. Convergence of INT reliability to real link reliability based on different INT frequency measurements. The data packets were sent every 0.5 s while INT data were added using different frequency.

architecture was integrated with other network elements, like network controller and network visualizer. Specific APIs between INT layer and higher and lower layers are designed, for on-the-fly information exchange and configuration purposes.

The PoC implementation of such design was carried out using Click router framework. We showed that various information related to the wireless interface can be collected, including RSSI, MCS, channel used, retransmissions as well as packet reception time. The PoC implementation was validated in a real industrial test-bed, where telemetry data were used as input for the network controller to enable automatic (re)configuration. Furthermore, the INT overhead was modeled based on different link and flow parameters, like packet length, data rate, telemetry periodicity, and the number of hops in the network. It was shown that INT has nearly 6 times less data overhead compared to active probing over a single hop. This benefit will increase by increasing the number of hops. In addition, the convergence time between INT reliability and data reliability was modeled. The INT reliability was shown to converge to real data reliability proportionally to the ratio of the data and INT period.

This work shows that in-band telemetry can be used in wireless networks to monitor link reliability, end-to-end delay, and detect network performance issues. Further, this paper gives the foundation for other deployments on top of telemetry-enabled wireless network. Such a technique can be used for network anomaly detection, network performance verification, or cognitive network management in different networks including industrial networks too.

Future investigation on how the INT reports should be shared with the interested entities should be done. When a centralized network configuration approach is followed, then even INT data needs to be collected in a central entity, like the one we proposed in this paper. Other approaches can include in-band monitoring report sharing with the previous nodes. Especially, this can be beneficial for approaches with distributed network configuration and management entities and will remove the risk of single point of failure. In addition,different techniques for INT report sharing and their impact on the network overhead can be studied. This include raw report sharing, event-based sharing, specific parameter sharing etc. Node mobility impact on the INT monitoring reliability is another aspect to be investigated. The absence of data packets in the network can decrease the monitoring frequency too. All this issues are still open research question that can be further investigate in the future.

ACKNOWLEDGMENT

This research was partially funded by the Flemish FWO SBO S003921N VERI-END.com (Verifiable and elastic end-to-end communication infrastructures for private professional environments) project, the Flemish Government under the "Onderzoeksprogramma Artificiële Intelligentie (AI) Vlaanderen" program, and from the FWO-Flanders, under grant agreement G055619N. This research has also received partial funding from the European Union's Horizon 2020 Research and innovation programme under grant agreement No. 826284 (ProTego).

REFERENCES

[1] "Openwifi. available online: https://orca-project.github.io/openwifi-tmp/openwifi-public.pdf," Accessed on 28.11.2019.

[2] "5g for connected industries and automation," *White paper, 5G Alliance for Connected Industries and Automation*, 2018.

[3] "Ieee 802.11ax: The sixth generation of wi-fi," *White paper, Cisco*, 2018.

[4] "Cognitive network management for 5g," White Paper, 5GPP Network Management Quality of Service Working Group, 2017.

[5] A. Mohyuddin and P. Dowland, "The art of network monitoring," *Advances in Networks, Computing and Communications 4*, p. 100, 2007.

[6] B. Claise, G. Sadasivan, V. Valluri, and M. Djernaes, "Rfc 3954: Cisco systems netflow services export version 9," *IETF http://www. ietf. org/rfc/rfc3954. txt*, 2004.

[7] B. Claise, "Rfc 5101: Specification of the ip flow information export (ipfix) protocol for the exchange of ip traffic flow information," *IETF, January*, 2008.

[8] D. Harrington, R. Presuhn, and B. Wijnen, "Rfc 3411: An architecture for describing simple network management protocol (snmp) management frameworks, 2002," *http://tools.ietf.org/html/rfc3411*.

[9] F. Brockners, S. Bhandari, C. Pignataro, H. Gredler, J. Leddy, S. Youell, D. Mozes, T. Mizrahi, P. Lapukhov, R. Chang, D. Bernier, and J. Lemon, "Data fields for in-situ oam, draft-ietf-ippm-iom-data-05," 2019.

[10] B. Weis, F. Brockners, C. Hill, S. Bhandari, V. Govindan, C. Pignataro, H. Gredler, J. Leddy, S. Youell, T. Mizrahi, A. Kfir, B. Gafni, P. Lapukhov, and M. Spiegel, "Ethertype protocol identification of in-situ oam data, draft-weis-ippm-ioam-eth-01," 2019.

[11] J. Haxhibeqiri, I. Moerman, and J. Hoebeke, "Low overhead, fine-grained end-to-end monitoring of wireless networks using in-band telemetry," in *CNSM2019, the 15th International Conference on Network and Service Management*, 2019, pp. 1–5.

[12] N. L. Van Adrichem, C. Doerr, and F. A. Kuipers, "Opennetmon: Network monitoring in openflow software-defined networks," in *2014 IEEE Network Operations and Management Symposium (NOMS)*. IEEE, 2014, pp. 1–8.

[13] S. Shirali-Shahreza and Y. Ganjali, "Traffic statistics collection with flexam," in *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4. ACM, 2014, pp. 117–118.

[14] C. Metter, V. Burger, Z. Hu, K. Pei, and F. Wamser, "Towards an active probing extension for the onos sdn controller," in *2018 28th International Telecommunication Networks and Applications Conference (ITNAC)*. IEEE, 2018, pp. 1–8.

[15] B. Wang and J. Su, "Flexmonitor: A flexible monitoring framework in sdn," *Symmetry*, vol. 10, no. 12, p. 713, 2018.

[16] L. Quan, J. Heidemann, and Y. Pradkin, "Trinocular: Understanding internet reliability through adaptive probing," in *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4. ACM, 2013, pp. 255–266.

[17] P. Lapukhov and R. Chang, "Data-plane probe for in-band telemetry collection," *Internet-Draft draft-lapukhov-dataplane-probe-01, Internet Engineering Task Force*, 2016.

[18] A. Gulenko, M. Wallschläger, and O. Kao, "A practical implementation of in-band network telemetry in open vswitch," in *2018 IEEE 7th International Conference on Cloud Networking (CloudNet)*. IEEE, 2018, pp. 1–4.

[19] N. Van Tu, J. Hyun, G. Y. Kim, J.-H. Yoo, and J. W.-K. Hong, "Intcollector: A high-performance collector for in-band network telemetry," in *2018 14th International Conference on Network and Service Management (CNSM)*. IEEE, 2018, pp. 10–18.

[20] J. Hyun, N. Van Tu, and J. W.-K. Hong, "Towards knowledge-defined networking using in-band network telemetry," in *NOMS 2018-2018 IEEE/IFIP Network Operations and Management Symposium*. IEEE, 2018, pp. 1–7.

[21] N. Van Tu, J. Hyun, and J. W.-K. Hong, "Towards onos-based sdn monitoring using in-band network telemetry," in *2017 19th Asia-Pacific Network Operations and Management Symposium (APNOMS)*. IEEE, 2017, pp. 76–81.

[22] C. Kim, A. Sivaraman, N. Katta, A. Bas, A. Dixit, and L. J. Wobker, "In-band network telemetry via programmable dataplanes," in *ACM SIGCOMM*, 2015.

[23] J. Geng, J. Yan, Y. Ren, and Y. Zhang, "Design and implementation of network monitoring and scheduling architecture based on p4," in *Proceedings of the 2nd International Conference on Computer Science and Application Engineering*. ACM, 2018, p. 182.

[24] D. Bhamare, A. Kassler, J. Vestin, M. A. Khoshkholghi, and J. Taheri, "Intopt: In-band network telemetry optimization for nfv service chain monitoring," in *ICC 2019-2019 IEEE International Conference on Communications (ICC)*. IEEE, 2019, pp. 1–7.

[25] C. Kim, P. Bhide, E. Doe, H. Holbrook, A. Ghanwani, D. Daly, M. Hira, and B. Davie, "Inband network telemetry (int)," *Accessed 15-04-2019.*, 2016.

[26] "ebpf. available online: https://prototype-kernel.readthedocs.io/en/latest/bpf/," Accessed on 28.11.2019.

[27] Serkantul, "Prometheus int exporter. avialable online: https://github.com/serkantul/prometheus_int_exporter," *Accessed 15-04-2019.*, 2016.

[28] ——, "Onos project. avialable online: https://wiki.onosproject.org/display/onos/wiki+home," *Accessed 03-05-2019.*

[29] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese *et al.*, "P4: Programming protocol-independent packet processors," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 87–95, 2014.

[30] D. Suh, S. Jang, S. Han, S. Pack, and X. Wang, "Flexible sampling-based in-band network telemetry in programmable data plane," *ICT Express*, vol. 6, no. 1, pp. 62–65, 2020.

[31] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, "The click modular router," *ACM Transactions on Computer Systems (TOCS)*, vol. 18, no. 3, pp. 263–297, 2000.

[32] A. Karaagac, E. De Poorter, and J. Hoebeke, "In-band network telemetry in industrial wireless sensor networks," *IEEE Transactions on Network and Service Management*, 2019.

[33] E. Zeljković, T. De Schepper, P. Bosch, I. Vermeulen, J. Haxhibeqiri, J. Hoebeke, J. Famaey, and S. Latré, "Orchestra: Virtualized and programmable orchestration of heterogeneous wlans," in *2017 13th International Conference on Network and Service Management (CNSM)*, Nov 2017, pp. 1–9.

[34] E. Coronado, S. N. Khan, and R. Riggio, "5g-empower: A software-defined networking platform for 5g radio access networks," *IEEE Transactions on Network and Service Management*, vol. 16, no. 2, pp. 715–728, June 2019.

[35] T. De Schepper, P. Bosch, E. Zeljković, F. Mahfoudhi, J. Haxhibeqiri, J. Hoebeke, J. Famaey, and S. Latré, "Orchestra: enabling inter-technology network management in heterogeneous wireless networks," *IEEE Transactions on Network and Service Management*, vol. 15, no. 4, pp. 1733–1746, 2018.

[36] P. H. Isolani, N. Cardona, C. Donato, J. Marquez-Barja, L. Z. Granville, and S. Latré, "Sdn-based slice orchestration and mac management for qos delivery in ieee 802.11 networks," in *2019 Sixth International Conference on Software Defined Systems (SDS)*, June 2019, pp. 260–265.

[37] F. Brockners, S. Bhandari, S. Dara, C. Pignataro, J. Leddy, S. Youell, D. Mozes, T. Mizrahi, A. Augado, and D. Lopez, "Proof of transit, draft-ietf-sfc-proof-of-transit-02," 2019.

[38] A. Adams, J. Mahdavi, M. Mathis, and V. Paxson, "Creating a scalable architecture for internet measurement," *IEEE Network*, 1998.

[39] "The OpenEmpower Protocol. available online: https://github.com/5g-empower/5g-empower.github.io/wiki," Accessed on 13.12.2019.

[40] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling Innovation in Campus Networks," *SIGCOMM Computer Communication*, vol. v.38, no. 2, pp. 69–74, mar. 2008.

[41] E. Coronado, R. Riggio, J. Villa1ón, and A. Garrido, "Lasagna: Programming abstractions for end-to-end slicing in software-defined wlans," in *2018 IEEE 19th International Symposium on" A World of Wireless, Mobile and Multimedia Networks"(WoWMoM)*. IEEE, 2018, pp. 14–15.

[42] M. Richart, J. Baliosian, J. Serrati, J. Gorricho, R. Agüero, and N. Agoulmine, "Resource allocation for network slicing in WiFi access points," in *2017 13th International Conference on Network and Service Management (CNSM)*, Nov 2017, pp. 1–4.

[43] "Available online: https://www.influxdata.com/time-series-platform/," *Accessed 15-04-2019.*, 2016.

Jetmir Haxhibeqiri received the Masters degree in Engineering (information technology and computer engineering) from RWTH Aachen University, Germany (2013). In 2019, he obtained a PhD in Engineering Computer Science from Ghent University with his research on flexible and scalable wireless communication solutions for industrial warehouses and logistics applications. Currently he is a post-doc researcher in the Internet Technology and Data Science Lab (IDLab) of Ghent University and imec. His current research interests include wireless communications technologies (IEEE 802.11, IEEE 802.15.4e, LoRa) and their application, IoT, wireless time sensitive networking, in-band network monitoring and wireless network management.

Pedro Heleno Isolano received the B.Sc. degree in information systems from the State University of Santa Catarina, Brazil, in 2012 and the M.Sc.degree from the Federal University of Rio Grandedo Sul, Brazil, in 2015, focused on the management of software-defined networking. He is currently pursuing the Ph.D. degree with imec and the University of Antwerp, Belgium. His current research interests include network functions virtualization, software-defined networking, and programmability of wireless medium access control protocols

Johann M. Marquez-Barja (Senior Member, IEEE) is currently a Professor with the University of Antwerp and a Professor in IMEC, Belgium. He is also leading the Wireless Cluster at IDLab/imec Antwerp. He was and is involved in several European research projects such as CREW, FORGE, WiSHFUL, Fed4FIRE/FAVORITE, Fed4FIRE+, eWINE, CONCORDA, 5G-CARMEN, FLEXNET, FUTEBOL (Technical Coordinator), 5G-Mobix, PROTEGO, InterConnect, and 5G-Blueprint projects (Technical Coordinator). His main research interests include 5G advanced architectures, edge computing, flexible and programmable future end-to-end networks, and the IoT communications and applications. His research interests also include vehicular communications, mobility, and smart cities deployments. He is co-leading the Citylab Smart City testbed, part of the City of Things program, and the SmartHighway testbed, both located in Antwerp, Belgium. He is also interested and active on education development, being actively involved in different research actions to enhance engineering education, in particular remote experimentation in e-learning systems. He is a member of ACM, and a Senior Member of the IEEE Communications Society, IEEE Vehicular Technology Society and IEEE Education Society, where he participates in the board of the Standards Committee. He has given several keynotes and invited talks in different major events, as well as received 30 awards in his career so far, and coauthored more than 120 articles. He is also serving as the Editor and the Guest Editor for different International Journals, as well as participating in several Technical Programme and Organizing Committees for several worldwide conferences/congresses.

Ingrid Moerman received her degree in Electrical Engineering (1987) and the Ph.D. degree (1992) from the Ghent University, where she became a part-time professor in 2000. She is a staff member at IDLab, a core research group of imec with research activities embedded in Ghent University and University of Antwerp. Ingrid Moerman is coordinating the research activities on mobile and wireless networking at Ghent University, where she is leading a research team of more than 30 members. Ingrid Moerman is also Program Manager of the 'Deterministic Networking' track at imec and in this role she coordinates research activities on end-to-end wired/wireless networking solutions driven by time-critical applications that have to meet strict QoS requirements in terms of throughput, latency bounds and dependability in smart application areas like Industry 4.0, vehicular networks and professional entertainment. Her main research interests include cooperative and intelligent radio networks, real-time software defined radio, time-sensitive networks, dynamic spectrum sharing, coexistence across heterogeneous wireless networks, vehicular networks, open-source prototyping platforms, software tools for programmable networks, next generation wireless networks (5G/6G/...), and experimentally supported research. Ingrid Moerman has a longstanding experience in running and coordinating national and EU research funded projects. At the international level, Ingrid Moerman was leading team SCATTER, consisting of researchers from IMEC-IDLab and Rutgers University (US), in the DARPA Spectrum Collaboration Challenge (SC2). The SCATTER team was one of the 10 finalists at the DARPA SC2 championship event organized at Mobile World Congress in LA (October 2019, https://www.spectrumcollaborationchallenge.com).

Jeroen Hoebeke received the Masters degree in Engineering Computer Science from Ghent University in 2002. In 2007, he obtained a PhD in Engineering Computer Science with his research on adaptive ad hoc routing and Virtual Private Ad Hoc Networks. Current, he is an associate professor in the Internet Technology and Data Science Lab of Ghent University and imec. He is conducting and coordinating research on wireless (IoT) connectivity, embedded communication stacks, deterministic wireless communication and wireless network management. He is author or co-author of more than 150 publications in international journals or conference proceedings.