Managing Programmable Low-End Wireless Networks through Distributed SDN Controllers

André Scheibe, Luciano Paschoal Gaspary, Weverton Cordeiro
Institute of Informatics (INF), Federal University of Rio Grande do Sul (UFRGS)
Porto Alegre, Brazil
{andre.scheibe,paschoal,weverton.cordeiro}@inf.ufrgs.br

***Abstract*—Low Power Wide Area Network (LPWAN) have recently attracted attention as a potential alternative to power affordable internet access to remote communities and help bridge the digital divide. Nevertheless, realizing the full potential of LPWAN devices as an alternative to bring remote communities to the internet requires addressing two technical challenges. The first one is making LPWAN devices suitable for exchange of data flows running under standard protocols like TCP/IP. The second one is devising a fully distributed management approach tailored for a network composed of autonomous LPWAN devices that are reachable only through unstable and unreliable wireless links. We addressed the first challenge in a prior work, with a conceptual solution for programmable low-end devices, i.e., LPWAN devices whose behavior can be redefined using programming languages for the data plane like P4. In this paper, we address the second challenge by presenting an algorithmic approach for SDN-based distributed management for programmable low-end networks. In summary, we consider that each programmable low-end device in a community network is autonomous and has its independent SDN controller. For coordinated management, they use control messages transmitted over unreliable and constrained links (up to 300 kbps), hence the need for a lightweight and fault-tolerant management approach. We experiment our SDN-based distributed approach with LTP (Lightweight Tunnel Protocol), a proof-of-concept protocol written in P4 for powering programmable low-end networks. We provide evidence, through a series of experiments, that our algorithmic approach for fully distributed SDN-based management is capable of maintaining stable network connectivity despite running over unstable and unreliable links.**

***Index Terms*—Low Power Wide Area Networks (LPWAN), Programmable Forwarding Planes, P4, Distributed SDN Controllers**

I. Introduction

Aiming at network management solutions that enable the seamless incorporation of adaptive and automated management approaches, thus offering new ways to operate and manage computer networks, we have observed in the past decades the vertiginous growth in the adoption of Software Defined Networks (SDN) [1], [2] and Programmable Data Planes (PDP) [3], [4]. These novel technologies, together, enabled network researchers and practitioners to leave behind closed-source and proprietary solutions and migrate to network structures with a behavior that can be freely redefined using Domain-Specific Languages (DSLs) like Lyra [5] and P4 [4]. With these features, it is possible to design, experiment, and implement novel ideas on the network without going through the industrial process of network control device suppliers [6].

As a result, the community has focused its efforts on developing distributed control solutions and addressing its shortcomings. Solutions such as Elasticon [7] which presents elastic controller structures (dynamically increasing and reducing the quantity) or Hyperflow [8], ONOS [9] and ONIX [10] which have a partitioning approach to network in various areas reducing latency and improving resiliency [11] has been showing good results. Nevertheless, these solutions were designed with high-speed, large-scale networks in mind or for use in data centers.

In contrast to these large structures contemplated by SDN approaches and solutions, we observe the growth of the Internet of Things (IoT) using low-end networks [12]. Numerous practical applications, such as environmental monitoring, smart cities and smart homes, and asset tracking [13] make use of low power and low data rate devices. Combining the flexibility of SDN networks with Low Power Wide Area Network (LPWAN) devices [12]–[14] would allow the adoption of these devices in novel solutions such as providing affordable internet access to remote communities and helping bridge the digital divide [15].

LPWAN devices, however, are resource-constrained, resulting in very low throughput on constrained wireless links[1]. Such restrictions make it impossible to use traditional structures for implementing SDN networks to manage networks with these components. Therefore, it is necessary to develop a novel network structure, with distributed control algorithms, to set up this point-to-point communication. This structure must be developed considering a distributed scenario, as autonomous and independent nodes, responsible for controlling the network and processing the situation of the networks with which it communicates, using an unreliable and unstable channel (data plane) to manage, through low-speed links.

In our previous work [15], we introduced a conceptual model and a proof-of-concept implementation of a solution for programmable low-end networks, i.e., low-end devices whose packet processing behavior can be redefined using Domain

[1]We considered as reference – in our research and our design of a proof-of-concept prototype – a LoStik USB LoRa device by Ronoth [16] and a LoRa/LoRaWAN Raspberry Pi SX127X HAT module by Dragino [17]. For these reasons, we assumed a 300 kbps narrow-band link, achievable with the SX127X HAT module.

ISBN 978-3-903176-42-3 © 2021 IFIP

Specific Languages (DSL) for programmable data planes like P4 [4]. Our solution enabled using LPWAN devices for exchanging TCP/IP flows over constrained and ureliable wireless links. In this paper, we take a step further and discuss an approach for distributed management of programmable low-end networks. In summary, we make the following research question: can we manage low-cost programmable networks through distributed SDN controllers? The positive answer unlocks the use of a multitude of low-cost devices combined with the possibility of changing their behavior dynamically using open programming standards.

To answer this question, in this paper we present an algorithmic approach for a fully distributed SDN-based management of programmable low-end network. We consider a community network structure with distributed management, an independent controller for each programmable low-end device, using the data plane to transmit messages between controllers and ensure the status update network between all devices. Based on this concept, we implement seven network scenarios using the Lightweight Tunnel Protocol (LTP) [15] to reduce communication overhead by replacing L2/L3 headers with labels that identify each combination of source and destination. This operation maximizes goodput achieving gains of 23%. In summary, our contribution is the presentation of an algorithmic approach for delivering fully distributed SDN-based management to programmable low-end networks for autonomous and independent controllers.

The remainder of the paper is organized as follows. We cover background and related work in Section II. In Section III, we introduce our novel SDN network structure concept, which will be exercise in Section IV, where we present the design and implementation of a novel Lightweight Tunnel Protocol (LTP) and its controls. In Sec. V, we present seven scenarios for using the LTP protocol, whereas in Section VI we provide an extensive evaluation of the scenarios. Finally, we close the paper in Section VII.

II. Background and Related Work

There are numerous alternatives for implementing distributed management in the literature. Nevertheless, assuming that every controller has an up to date view of the network status remains an open challenge. This is particularly special in scenarios where one does not have a communication channel between the controllers and as such does not have a guarantee that control messages sent are delivered.

Onix [10], Hyperflow [8], and ONOS [9] solutions are examples of distributed control platforms that, through the various connected controllers, store a total view of the network. Each solution, however, uses different means of sharing information and ensuring control plane scalability.

Hyperflow, for example, has in each controller the global view of the network and uses an event propagation system to update controllers with changes to the network state (only events that change the state of the network). In this way, all controllers make local decisions and propagate those that affect the state of the network, reducing the flow of events. However, the disadvantage of this implementation is precisely this mechanism is designed to transmit only infrequent records and that it has no guarantee of delivery order, which can lead to inconsistencies.

To control the traffic of events shared between controllers, ONOS places a timestamp on each event. When a network change occurs and an event is registered, it is initially stored in the device where it occurred (source) and propagated to the other controllers with the identification information of the source programmable low-end device, the date and time stamp, and the event number. This way, when receiving the event, the destination programmable low-end device is able to determine if the information is useful or obsolete and, in the second case, discard the event record [18].

As in ONOS, in ONIX each controller is responsible for only a part of the network and, in this way, stores a partial view of the network structure. In the ONIX implementation, however, the structure of the network is stored in a NIB (Network Information Base) where each controller is responsible for a part of the NIB. The sync cost of this database that can lead to controller overload is the main disadvantage of this implementation [18].

Recent work comparing solutions for implementing distributed control [11], [18], [19] has highlighted OpenDayLight [20]. The main advantage is that, unlike the other methods presented, which are intended for use in large networks and data centers or even in service providers, this implementation for presenting open source aimed at the community, which guarantees compatibility with numerous new applications, including works geared towards the IoT.

All solutions presented, however, make clear the dependence on a secure and stable communication channel to exchange synchronism messages between controllers. The use of low-end devices for network communication, however, only allows communication with low capacity links and a lot of packet loss. Thus, a new network structure is needed to allow the use of distributed controllers for programmable low-end networks scenarios.

III. A Distributed SDN Controller for Programmable low-end Networks

In this paper, we present an algorithmic approach for fully distributed SDN-based network management for programmable low-end networks with distributed controllers. We contemplate in our approach low-end devices with limited resources (e.g., memory and processing capabilities) and that operate under unfavorable network conditions and, consequently, provide low transfer rates. To achieve this goal, we use data plane programmability (PDP), developing protocols that can operate in this scenario and motivating other network professionals to use the same strategy to develop new applications for low-end devices.

First, we need to understand what disciplines must be addressed for SDN to be an effective solution in a given network scenario. According to [11], distributed control has challenges

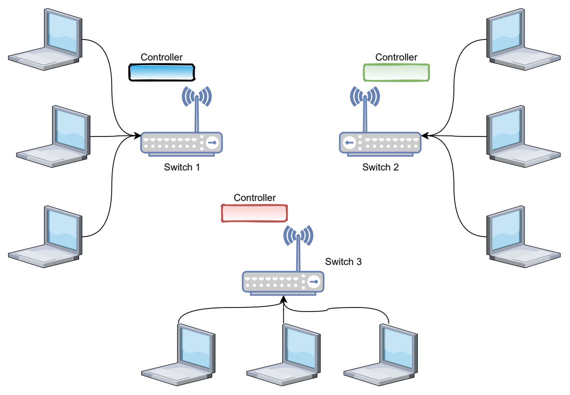


Fig. 1: Programmable low-end networks with each device using its independent controller.

in scalability (decentralizing and dividing the load of controllers), reliability (ensuring consistency between controllers), consistency (finding the best relationship between consistency and performance), interoperability (lack of standards open for communication between controllers), monitoring (monitor with little impact), and security (decentralize as much as possible to reduce impact in the event of a failure or attack).

Thus, to achieve the aforementioned standards, we developed an implementation model which, as can be seen in Figure 1, is based on providing a controller for each programmable low-end device, thus making it independent from the others, since there is no controller centralized nor a master/slave hierarchy. This type of implementation in an environment with limited connection and even processing resources is only possible considering some requirements that must be addressed by the programmable low-end device and the controller to achieve traffic reduction between these two devices or between controllers by limiting the flow to events that are needed to write network status changes to each programmable low-end device's match+action tables. Recall that the limited bandwidth available in a point-to-point communication between low-end devices means that a secure and stable connection between controllers is unfeasible. Thus, we need to develop controls and guarantees so that the synchronization is performed between devices through the data plane causing the least possible impact. Therefore, our approach concatenates the events transmitted between controllers to data packets, which makes some implementation changes necessary both in the programmable low-end device and in the controller.

To make it easier to understand, we divide the analysis considering the responsibilities that will be transferred to each of the elements.

In addition to the usual activities, among the requirements that must be met by the controller, we highlight:

- Identify the device and event. The absence of a centralized controller forces the devices to identify themselves because, following the example of ONOS [9], this method distinguishes the flow and allows another controller to verify whether it has processed this change/instruction.
- Add to the data packet the instruction with network status change that will be sent through the data plane to the other controller.
- Separate from the received packet the instruction (event) and the data that must be returned to the swith to be forwarded in the flow. Processing the instruction is already a controller assignment in traditional SDN networks.
- Store status of all received events. It will allow one to analyze if the rule has already been processed (and can be discarded) and/or if it is necessary to record this change on this device. This control allows delivery guarantee avoiding the problems identified in Hyperflow [8].

With the flow changes listed above, some changes are also required in the low-end device programming, including:

- Identify and differentiate packets in the flow that have only data (and must be simply forwarded in the flow) from packets that have the instruction+data set.
- Store and change the status of received events (in the same way as in the controller) to check if it is necessary to discard, forward to the controller or retransmit an instruction (if the processing confirmation is not received by another controller).

In summary, our implementation meets the requirements listed by [11]: scalability (with distributed controllers), reliability (ensures consistency between controllers by controlling the status of each event), consistency (transmitting only the necessary events and ensuring that were processed), Interoperability (presents a new concept that can be adopted by any professional), monitoring (the status of each event is stored in the devices and does not consume flow) and security (distributed, independent controllers responsible for only one programmable low-end device).

To consolidate understanding of this approach, we briefly review in the following section LTP (Lightweight Tunnel Protocol) [15], a protocol devised as an exercise of using LPWAN devices for programmable low-end networks. We also discuss how each of the changes suggested above was implemented to ensure network status updates across all devices.

IV. A Review of Lightweight Tunnel Protocol (LTP) and Management Controls

In our prior work [15], [21], we introduced an exercise of an LTP protocol for programmable low-end networks. In this section, we review the core concepts behind LTP and then introduce our proposal of an algorithm for distributed control of programmable low-end devices, exploring the mechanisms and means used to meet the additional programmable low-end device and controller requirements discussed in the previous section. Thus, let us explore what needs to be considered in a lightweight protocol to manage controllers in a distributed and independent way, separating the new assignments that need to be in the control plan (and it is the controller's responsibility) from what needs to be ensured by the data plan (programmable low-end device responsibility).

```

# omitted for brevity

LTP_CPU = "\375"
LTP_DEF = "\376"
LTP = 0
DEV_ID = "\001"

EGRESS_PORT = 1


def main(p4info_file_path, bmv2_file_path):
  global LTP_CPU, LTP_DEF, LTP, DEV_ID, EGRESS_PORT

  arp_table = {}
  ip_cache = {}
  tag_cache = {}

# omitted for brevity

while True:
  print("Packet-in - BEFORE")
  packetin = s1.PacketIn()
  print("Packet-in - AFTER")
  if packetin.WhichOneof('update')=='packet':
    print("Packet-in message update received")

    metadata = packetin.packet.metadata
    for meta in metadata:
      metadata_id = meta.metadata_id
      value = meta.value

    output_metadata = "\000\000"

    packet_orig=packetin.packet.payload
    packet_string = hexlify(packet_orig)

    # omitted for brevity

    if packet_orig[0] == LTP_DEF:
       # omitted for brevity

    elif packet_orig[0] == LTP_CPU:

      packet = packet_orig[4:]
      pkt = Ether(_pkt=packet)

      eth_src = pkt.getlayer(Ether).src
      eth_dst = pkt.getlayer(Ether).dst
      ether_type = pkt.getlayer(Ether).type

      # omitted for brevity

      if ether_type == 2048:
        ip_src = pkt[IP].src
        ip_dst = pkt[IP].dst

        # omitted for brevity

        DEVSRC_ID=packet_orig[1]
        LTPSRC_ID=packet_orig[2]

        # Check if ip_src and ip_dst combination is in arp_table
        if (ip_src, ip_dst) not in ip_cache and (ip_dst, ip_src) not in ip_cache:
          # omitted for brevity

          # Write ip_src and ip_dst in arp_table
          ip_cache.setdefault((ip_src, ip_dst), (DEVSRC_ID, LTPSRC_ID))

          # Write ip_dst and ip_src in arp_table (return way)
          ip_cache.setdefault((ip_dst, ip_src), (DEVSRC_ID, LTPSRC_ID))

          # Write rule in switch table.
          writeIpv4BuildRules(p4info_helper, sw=s1, dev_id=DEVSRC_ID, tag_id=
                ↪ LTPSRC_ID, ip_src=ip_src, ip_dst=ip_dst)

        else:
          # omitted for brevity

    else:
      # omitted for brevity

      pkt = Ether(_pkt=packet_orig)

      eth_src = pkt.getlayer(Ether).src
      eth_dst = pkt.getlayer(Ether).dst
      ether_type = pkt.getlayer(Ether).type

      # omitted for brevity
      if ether_type == 2048:
        ip_src = pkt[IP].src
        ip_dst = pkt[IP].dst
        proto = pkt[IP].proto

        # omitted for brevity
        # Check if ip_src is in arp_table
        if ip_src not in arp_table:
          # omitted for brevity
          arp_table.setdefault(ip_src, (eth_src, value, eth_dst))

          # omitted for brevity
          writeARPReply(p4info_helper, sw=s1, dst_ip=ip_src, dst_mac_addr=eth_src,
                ↪ sw_port=value, sw_port_mac=eth_dst)

        if (ip_src, ip_dst) not in ip_cache and (ip_dst, ip_src) not in ip_cache:
          # omitted for brevity
          # Is a new combination, increase LTP value
          DEVSRC_ID=DEV_ID
          LTP=LTP + 1
          LTPSRC_ID=LTP

          # Write ip_src and ip_dst combination in arp_table
          ip_cache.setdefault((ip_src, ip_dst), (DEVSRC_ID, LTPSRC_ID))
          # Write ip_dst and ip_src combination is in arp_table (return)
          ip_cache.setdefault((ip_dst, ip_src), (DEVSRC_ID, LTPSRC_ID))

          # omitted for brevity

          # Write rule on switch table
          writeTagBuildRules(p4info_helper, sw=s1, dev_id=DEVSRC_ID, tag_id=LTPSRC_ID,
                ↪  eth_src=eth_src, ip_src=ip_src, ip_dst=ip_dst, port=EGRESS_PORT)

          # omitted for brevity

          writeIpv4BuildRules(p4info_helper, sw=s1, dev_id=DEVSRC_ID, tag_id=LTPSRC_ID
                ↪ , ip_src=ip_dst, ip_dst=ip_src)

          # Input LTP_CPU in the package
          output_metadata = "\000" + LTP_CPU

        else:
          # omitted for brevity
          # Write rule on switch table
          writeTagBuildRules(p4info_helper, sw=s1, dev_id=DEVSRC_ID, tag_id=LTPSRC_ID,
                ↪  eth_src=eth_src, ip_src=ip_src, ip_dst=ip_dst, port=EGRESS_PORT)

          output_metadata = "\000" + LTP_DEF

    # omitted for brevity

    # Preparing packet to send to switch
    packetout = p4info_helper.buildPacketOut(
      payload = packet_orig,
      metadata = {
        1: output_metadata
      }
    )
    # omitted for brevity

    # send packet to switch
    s1.PacketOut(packetout)
    # omitted for brevity
```

Fig. 2: Excerpt of the Python Controller code for the Lightweight Tunnel Protocol (LTP).

A. Protocol Overview

To serve low-cost devices with unreliable wireless links, we propose in [15] a lightweight data transfer protocol we call Lightweight Tunnel Protocol (LTP). This proposal reduces the overhead by replacing L2/L3 headers with a label (TAG) that identifies the communication host pair (source/destination). These TAGs are single-hop, point-to-point identifiers, meaning they are only valid for wireless transmission between a pair of low-end devices. Labels are created on one programmable low-end device, replace the packet's link and network headers, are transmitted wirelessly to the other device where the replacement is undone, and the packet is forwarded normally.

Note that LTP protocol does not use a communication channel between controllers, as it injects instructions into the data packet itself to transmit them to the other devices to update the network status. To do so, it uses some controls embedded in the header and processed during its handshake and following a state machine. Next we take a closer look at how we implement the controller features.

B. Distributed SDN Control for low-end Networks

We present in Fig. 2 an excerpt of the algorithm for distributed control of programmable low-end networks[2]. In this code, we assume a low-end network in which each node acts autonomously and there is no reliable channel for exchanging management information. It means that the controller must establish communication channels between nodes and assume that this coordination is subject to failures due to packet loss. The key management tasks that must be performed in a low-end network are listed next:

1) First management task: Identify the device and event. We can see lines 6 and 7 in Fig. 2 where the controller starts the LTP variable (identifier of the source/destination tuple) and sets the `DEV ID` (Device Identification) variable with the number of this device (in the code, number 1). As noted, this identifier must be unique in the universe of controllers that are communicating and must be previously configured. The combination of the device number that originated the traffic and a sequential event number (TAG - representing each source/destination combination) guarantee that this identifier will not be repeated (and was not used before). Once the flow is identified with a unique tag, it will be possible to control all devices if the processing of this network change has already been carried out.

2) Second management task: Add in the data packet the instruction with network status change that will be sent through the data plane to the other controller. Every time it is necessary to make a change to the network, the LTP identifier will be incremented (line 111 of Fig. 2) and the rule recorded in the device tables (line 122 of Fig. 2). In our protocol, the `LTP CPU` variable (set in line 4 of Fig. 2) identifies that a network status change record has been created and should be propagated. To do so, the variable is added to the packet (line 129 and lines 141-146 of Fig. 2), allowing this instruction to be identified in the destination device when processed.

3) Third management task: Separate from the received packet the instruction (event) and the data that must be returned to the device to be forwarded in the flow. Since instruction and data are received at the device concatenate, it is initially necessary for the device to identify that it is receiving instruction. Subsequently, it must forward this instruction to the controller, which needs to be able to separate the instruction from the packet, analyze and verify that it has not processed this change yet, and forward the rest of the packet so that the device can handle it by sending the packet to the next-hop or the destiny. To do so, as noted in the previous item, we add the variable `LTP CPU` in the packet that was transmitted to the destination. When a device identifies that it has received a new instruction, that is, it has received `LTP CPU` (line 43 in Fig. 2), it separates the rest of the packet (line 45 in Fig. 2) and performs the remaining deals on the packet.

4) Fourth management task: Store status of all received events. As we mentioned before, this control will allow us to analyze if the rule has already been processed (and can be discarded) and/or if it is necessary to record this change in this device. The controller starts the `arp table, ip cache, and tag cache` lists at lines 15,16, and 17 in Fig. 2, respectively. All instructions received and processed by the controller are stored in these lists (line 68,71,102,115,117 in Fig. 2), thus it is possible to identify if a rule/event had already passed through this device.

[2]Kindly note that the code shown in the paper partially differs from the one in our GitHub repository [21], for the sake of legibility and space constraints.

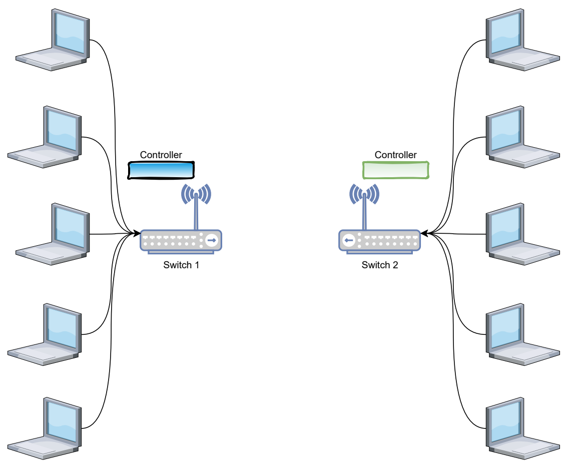


Fig. 3: Scenario 6 with 2 programmable low-end devices and 5 hosts per switch.

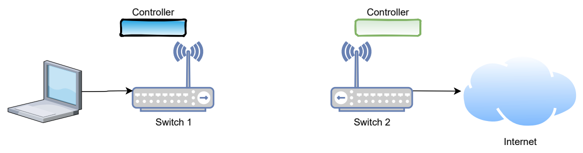


Fig. 4: Scenario 7 with 2 programmable low-end devices, 1 host and internet access.

C. Management Features in the Programmable Device

To conclude the presentation of our approach, let us quickly review the features handled by the device in the scenario as these were explored in more depth in [15].

- Identify and differentiate packets in the flow that have only data (and must be simply forwarded in the flow) from packets that have the instruction+data set. Using look ahead in the parser (P4 `lookahead()`) the device is able to differentiate a packet with an instruction (it has `LTP CPU`) or it is a data packet that must be treated.
- Store and change the status of received events. Through a state machine presented in [15], the device is able to verify if the instruction should be forwarded to the controller, discarded, or just follow the flow.

V. Community Network Evaluation Scenarios

The analysis of the feasibility of our algorithmic approach for fully distributed SDN-based management of programmable low end networks was carried out from the perspective of some scenarios. Except for scenario 6 and 7, all other scenarios use the initial topology of 3 programmable low-end devices and 3 stations on each device (see Fig. 1). We also provide evidence of the goodput gains considering a community network using LTP in comparison to a network without the use of this protocol, called standard network (STD).

1) Scenario 1: Compares structure using the LTP protocol and standard structure with the variation of the links in 64 kbps, 128 kbps, 256 kbps and 512 kbps and varying protocol between UDP and TCP, considering a payload of 128 bytes.

2) Scenario 2: Compare structure using the LTP protocol and standard structure with the variation of the links in 64 kbps, 128 kbps, 256 kbps and 512 kbps and varying protocol between UDP and TCP, considering a payload of 512 bytes.

3) Scenario 3: Compare structure using the LTP protocol and standard structure with the variation of the links in 64 kbps, 128 kbps, 256 kbps and 512 kbps and varying protocol between UDP and TCP, considering a payload of 1024 bytes.

4) Scenario 4: Compare structure using the LTP protocol and standard structure with the variation of the links in 64 kbps, 128 kbps, 256 kbps and 512 kbps and varying protocol between UDP and TCP, wih payload limited to MTU.

5) Scenario 5: Analyzes the structure with the LTP protocol using 256 kbps links, UDP packets between 2 hosts of each device: h1-h4, h7-h2, h5-h8.

6) Scenario 6: Analyzes the structure with the LTP protocol using 256 kbps links, UDP packets in a network with 2 devices and 5 stations each (see Fig. 3).

7) Scenario 7: Analysis of the download of a file from the internet by a host connected to a programmable low-end device (s2) that is connected to a second device (s1) with an internet connection. The connection between the two devices uses the LTP protocol and a 256 kbps link (see Fig. 4).

VI. Evaluation

In our previous work [15], we provided evidence of the technical feasibility of using LPWAN as programmable networking devices whose behavior can be redefined using languages like P4 [4]. In this section, we present evidence of the effectiveness of the distributed management approach for programmable low-end networks, by depicting the results of an evaluation considering experimental and production flows exchanged between nodes. We wrote LTP on P4_16 and implemented it considering a bmv2 switch on mininet. The controllers were written in Python2.7 with the help of the packet manipulation tool scapy [22]. To generate network flows between hosts and measure throughput, we used iperf3.

We present the results of the first analysis on Table I and Table II, which together present 32 variations of experiments performed in our scenarios varying link speed, payload size in addition to the use or not of the proposed protocol. The tables present the result of 30 rounds of experiments in each with 30-second intervals between them. Observe in the the percentage gain column compared to the payload size column that the biggest gains are related to low payload sizes, exactly because the ratio between the reduction offered by replacing L2/L3 headers by TAG are greater than to this ratio when the payload tends to the maximum MTU size. Thus, as we emphasize, our solution is interesting for environments where the flow of packets with low payload is intense, such as in networks with low-end devices, where we achieved a gain of approximately 23% using UDP/IP.

Another important analysis, considering the Table II is that, even if the gain and the average are visibly higher, due to the large standard deviation of the averages in the measurements resulting from the behavior and controls imposed by the TCP/IP protocol, it is not possible to say in every experiment, with a 99% confidence level, that our solution gives a better result.

If we analyze Fig. 5, which demonstrates throughput measurements with experiments using iperf3 in scenarios varying payload size between 128 bytes, 512 bytes, 1024 bytes, and maximum MTU size comparing scenarios with and without LTP and 256kbps link speed with TCP/IP traffic, it is easier to verify what we claim. Results measured with LTP, even if they show a slight advantage over measurements without LTP, often overlap, due to TCP/IP's flow control and congestion control mechanisms that lead the protocol to constantly adapt its throughput.

Analyzing, however, in isolation the figure 5(a) with the measurements of this same comparison, but considering only the 128-byte payloads, we observe that the gain is about 26%, even higher when compared to the gain seen in UDP/IP. The low payload makes the link limit not reached, thus the flow and congestion controls are not triggered. Note that the gain in the UDP observations [15], which demonstrates throughput measurements with experiments using iperf3 in scenarios varying payload size between 128 bytes, 512 bytes, 1024 bytes, and maximum MTU size comparing scenarios with and without LTP and 256kbps link speed with UDP/IP traffic, is easier to observe exactly because it does not have these throughput limiting controls.

Even if this solution is implemented in an environment that has the characteristic of sending packets predominantly with the maximum payload (reaching MTU), the result achieved in both TCP/IP and UDP/IP is 3% as can be seen in the two tables. Even if first impressions are not considerable results, in a resource-constrained environment, any gain is important. Anyway, in the Sec. VII we list some projects to maintain this gain in a network environment with unfavorable conditions such as networks with low-end devices, as well as list works that aim to improve these values.

VII. Final Considerations

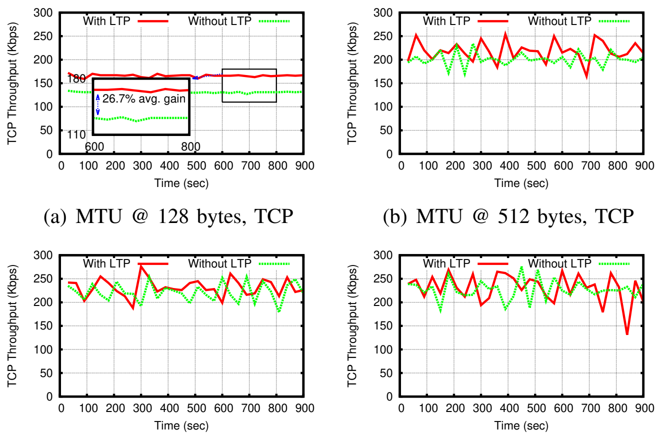
The use of distributed controllers as a method to achieve scalability in SDN has been a target of the research community for a long time. The solutions currently offered still need to solve new challenges of this approach, such as ensuring the synchronism of the state of the network in all equipment and balancing the load between controllers, in addition to being aimed at large networks, data centers or service providers.

TABLE I: UDP/IP Experiments with various link speeds, payload lengths, and confidence level 0.99.

#	Link speed (kpbs)	Payload size (bytes)	LTP Avg	LTP SD	LTP CI	LTP Avg-CI	LTP Avg+CI	UDP/IP Avg	UDP/IP SD	UDP/IP CI	UDP/IP Avg-CI	UDP/IP Avg+CI	Gain %
1	64	128	57.1	0.15	0.08	57.03	57.18	46.3	3.05	1.54	44.73	47.80	23.44
2	64	512	61.1	0.09	0.05	61.01	61.10	56.5	2.75	1.39	55.11	57.88	8.07
3	64	1024	61.7	0.39	0.20	61.52	61.91	59.0	2.69	1.36	57.66	60.37	4.57
4	64	1448	61.9	0.44	0.22	61.72	62.17	59.8	2.58	1.30	58.46	61.06	3.66
5	128	128	114.0	0.41	0.21	113.76	114.17	93.1	2.58	1.30	91.83	94.42	22.38
6	128	512	121.9	0.55	0.28	121.62	122.18	113.0	4.95	2.49	110.51	115.49	7.88
7	128	1024	123.0	0.79	0.40	122.60	123.40	118.0	4.04	2.03	116.00	120.06	4.21
8	128	1448	124.0	0.00	0.00	124.00	124.00	119.3	4.28	2.15	117.11	121.42	3.97
9	256	128	227.8	3.04	1.53	226.27	229.33	184.4	3.82	1.92	182.48	186.32	23.54
10	256	512	243.8	0.91	0.46	243.37	244.29	229.6	4.73	2.38	227.22	231.98	6.20
11	256	1024	246.1	2.32	1.17	244.90	247.23	237.6	5.70	2.87	234.73	240.47	3.56
12	256	1448	246.1	3.55	1.78	244.32	247.88	238.6	5.95	2.99	235.64	241.63	3.13
13	512	128	447.3	7.52	3.79	443.55	451.12	364.7	4.19	2.11	362.59	366.81	22.66
14	512	512	487.8	1.52	0.77	487.00	488.53	453.4	7.89	3.97	449.43	457.37	7.58
15	512	1024	493.6	2.01	1.01	492.62	494.64	472.9	7.78	3.91	468.99	476.81	4.38
16	512	1448	495.2	1.21	0.61	494.56	495.77	480.6	4.51	2.27	478.36	482.91	3.02

TABLE II: TCP/IP Experiments with various link speeds, payload lengths, and confidence level 0.99.

#	Link speed (kpbs)	Payload size (bytes)	LTP Avg	LTP SD	LTP CI	LTP Avg-CI	LTP Avg+CI	TCP/IP Avg	TCP/IP SD	TCP/IP CI	TCP/IP Avg-CI	TCP/IP Avg+CI	Gain %
1	64	128	38.3	4.01	2.02	36.27	40.30	30.0	3.18	1.60	28.41	31.60	27.59
2	64	512	58.4	22.21	11.18	47.21	69.56	49.0	12.28	6.18	42.83	55.19	19.13
3	64	1024	57.2	16.05	8.08	49.14	65.30	55.0	15.55	7.83	47.12	62.78	4.13
4	64	1448	59.1	26.03	13.10	46.02	72.21	57.1	22.89	11.52	45.61	68.66	3.47
5	128	128	80.3	3.15	1.59	78.71	81.88	65.1	1.34	0.68	64.44	65.80	23.30
6	128	512	108.2	21.61	10.87	97.28	119.03	101.4	21.23	10.68	90.76	112.12	6.62
7	128	1024	115.7	28.01	14.09	101.62	129.81	111.0	20.90	10.52	100.50	121.54	4.23
8	128	1448	117.9	25.95	13.06	104.88	130.99	113.6	37.86	19.05	94.56	132.66	3.81
9	256	128	165.9	2.78	1.40	164.50	167.30	130.9	1.30	0.65	130.25	131.55	26.74
10	256	512	218.0	21.13	10.64	207.33	228.60	201.0	14.91	7.50	193.53	208.54	8.42
11	256	1024	231.8	19.09	9.61	222.16	241.37	221.7	19.84	9.99	211.68	231.65	4.56
12	256	1448	235.7	39.16	19.71	215.99	255.41	227.1	32.42	16.31	210.79	243.41	3.79
13	512	128	305.1	29.36	14.78	290.32	319.88	258.0	13.21	6.65	251.39	264.68	18.24
14	512	512	441.2	17.32	8.71	432.49	449.91	406.5	22.68	11.41	395.12	417.95	8.53
15	512	1024	466.9	17.44	8.78	458.16	475.71	446.6	11.76	5.92	440.68	452.52	4.55
16	512	1448	473.4	24.67	12.42	460.98	485.82	460.0	33.80	17.01	442.95	476.98	2.92



(a) MTU @ 128 bytes, TCP

(b) MTU @ 512 bytes, TCP

(c) MTU @ 1024 bytes, TCP

(d) MTU @ 1500 bytes, TCP

Fig. 5: Flow performance with LTP and with standard TCP/IP.

Bringing the benefits of SDN and the possibilities arising from the programmability of the control plan for low-cost devices unlocks this equipment for a multitude of new applications, such as enabling internet access in low-income communities.

But how to implement an SDN solution with distributed controllers on low-end equipment with low throughput and unstable links, without a secure channel for the controllers to communicate? A new approach is needed, with independent programmable low-end devices that use the unreliable link for exchange of management information.

In this paper, we demonstrate that it is possible to implement and manage a network with distributed, independent and individual controllers per programmable low-end device. Our protocol guarantees the update of the state of the network for the controllers through information embedded in the packets that travel on the network, that is, it sends messages to update the control plane through the data plane.

The analysis of the results presented proves the feasibility of an interesting throughput gain for the communication profile of this type of network (packets with low payload). These positive results open the doors for new professionals and researchers to explore new applications for low-end equipment, as well as using the approach presented for other scenarios where the behavior and requirements are similar to those faced.

We are working on the implementation of sending periodic telemetry messages with meteorological information that, together with machine learning, can parameterize the antennas to make the best use of the equipment under current atmospheric conditions. Likewise, aiming to increase the gain achieved, especially in scenarios where the payload tends to the maximum MTU size, we are working on the development of scenarios with payload compression, which would further reduce traffic, increasing throughput.

Acknowledgements

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001.

References

[1] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: Enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008.

[2] D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, Jan 2015.

[3] P. Bosshart, G. Gibb, H.-S. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, and M. Horowitz, "Forwarding metamorphosis: Fast programmable match-action processing in hardware for sdn," in *ACM SIGCOMM 2013*. New York, NY, USA: ACM, 2013, p. 99–110.

[4] 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.

[5] J. Gao, E. Zhai, H. H. Liu, R. Miao, Y. Zhou, B. Tian, C. Sun, D. Cai, M. Zhang, and M. Yu, "Lyra: A cross-platform language and compiler for data plane programming on heterogeneous asics." New York, NY, USA: Association for Computing Machinery, 2020.

[6] W. L. da Costa Cordeiro, J. A. Marques, and L. P. Gaspary, "Data plane programmability beyond openflow: Opportunities and challenges for network and service operations and management," *Journal of Network and Systems Management*, vol. 25, no. 4, pp. 784–818, Oct 2017.

[14] M. Bembe, A. Abu-Mahfouz, M. Masonta, and T. Ngqondi, "A survey on low-power wide area networks for iot applications," *Telecommunication Systems*, vol. 71, no. 2, pp. 249–274, 2019.

[7] A. Dixit, F. Hao, S. Mukherjee, T. Lakshman, and R. Kompella, "Towards an elastic distributed sdn controller," *ACM SIGCOMM computer communication review*, vol. 43, no. 4, pp. 7–12, 2013.

[8] A. Tootoonchian and Y. Ganjali, "Hyperflow: A distributed control plane for openflow," in *Proceedings of the 2010 internet network management conference on Research on enterprise networking*, vol. 3, 2010.

[9] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow *et al.*, "Onos: towards an open, distributed sdn os," in *Proceedings of the third workshop on Hot topics in software defined networking*, 2014, pp. 1–6.

[10] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama *et al.*, "Onix: A distributed control platform for large-scale production networks." in *OSDI*, vol. 10, 2010, pp. 1–6.

[11] F. Bannour, S. Souihi, and A. Mellouk, "Distributed sdn control: Survey, taxonomy, and challenges," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 1, pp. 333–354, 2017.

[12] U. Raza, P. Kulkarni, and M. Sooriyabandara, "Low power wide area networks: An overview," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 2, pp. 855–873, 2017.

[13] K. Mekki, E. Bajic, F. Chaxel, and F. Meyer, "A comparative study of lpwan technologies for large-scale iot deployment," *ICT express*, vol. 5, no. 1, pp. 1–7, 2019.

[15] A. Scheibe, W. Reichert, L. Gaspary, and W. Cordeiro, "Programmable low-end networks: Powering internet connectivity for the other three billion," in *2021 IFIP/IEEE International Symposium on Integrated Network Management (IM)*. IEEE, 2021, pp. 187–195.

[16] Ronoth, "LoSitck - USB LoRa Device – Ronoth," 2020, available: https://ronoth.com/products/lostik.

[17] Dragino, "Raspberry Pi HAT featuring GPS and LoRa® technology," 2020, available: https://www.dragino.com/products/lora/item/106-lora-gps-hat.html.

[18] Y. E. Oktian, S. Lee, H. Lee, and J. Lam, "Distributed sdn controller system: A survey on design choice," *computer networks*, vol. 121, pp. 100–111, 2017.

[19] O. Salman, I. H. Elhajj, A. Kayssi, and A. Chehab, "Sdn controllers: A comparative study," in *2016 18th Mediterranean Electrotechnical Conference (MELECON)*. IEEE, 2016, pp. 1–6.

[20] J. Medved, R. Varga, A. Tkacik, and K. Gray, "Opendaylight: Towards a model-driven sdn controller architecture," in *Proceeding of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks 2014*. IEEE, 2014, pp. 1–6.

[21] UFRGS Networks Group, "GitHub - ProgrammableLowEndNetworks repo," 2021, available: https://github.com/ComputerNetworks-UFRGS/ProgrammableLowEndNetworks.

[22] Scapy, "Scapy - Packet crafting for Python2 and Python3," 2020, available: https://scapy.net/.