# Leveraging Distributed Protocols for full End-to-End Softwarization in IoT Networks

Esteban Municio*, Niels Balemans†, Steven Latré*, Johann Marquez-Barja†
*IDLab—Department of Mathematics and Computer Science, University of Antwerp—IMEC, Antwerp, Belgium
{esteban.municio,steven.latre}@uantwerpen.be
†IDLab—Faculty of Applied Engineering, University of Antwerp—IMEC, Antwerp, Belgium
{niels.balemans,johann.marquez-barja}@uantwerpen.be

***Abstract*—Current Software Defined Networking (SDN) techniques allow improving network control and flexibility. However its use in IoT is not trivial because IoT networks are unreliable and highly resource-constrained. Among some of the existing solutions proposed in the literature, Whisper enables SDN-like control over the packet forwarding and cell allocation of IoT devices by injecting in the network artificial, but still standard compliant messages that alter the default protocol behavior. Since Whisper uses carefully computed routing and scheduling messages that are compatible with the distributed protocols run in the network, it reduces the overhead in the network and operates without modifying the IoT devices' firmware. However, as other SDN-on-IoT technologies, Whisper is currently limited to the IoT network scope and remains as yet another independent network management silo. In this paper we propose a new higher-level architecture that allows to fully integrate the IoT SDN network management into a network operating system, such as ONOS, by using Whisper in order to provide an integral end-to-end softwarization. We also describe the interaction between the Whisper platform and the orchestrator and test our solution with real 6TiSCH-compatible hardware in the ONOS platform. Finally, we discuss the requirements and technical challenges to fully leverage Whisper to provide an efficient and programmable end-to-end control over an heterogeneous network domain.**

***Index Terms*—IoT, SDN, 6TiSCH, RPL, ONOS, Whisper**

## I. Introduction

Software Defined Networking (SDN) is considered today the tipping point that changed how networks are built and operated. SDN allows a network programmability level and a fine-grained resource management that is almost impossible to obtain with traditional distributed network protocols. While operators are widely using SDN in wired and optical production networks, the development and deployment of SDN in wireless networks is still on-going, specially in Internet of Things (IoT) networks.

IoT networks are highly resource-constrained in terms of reliability, energy and throughput. Since these limitations impede a direct mapping of wired SDN techniques to IoT, a significant research effort has been done in order to accommodate the IoT constraints to the SDN environment [1]. Although some SDN-on-IoT solutions can cope with most of the constraints, the in-band signaling overhead, the increase in energy consumption and the uncoupling between the routing and scheduling layers still leave partially unanswered the question of: "*Is SDN actually interesting for IoT networks?*".

In order to shed more light to that question, we previously proposed Whisper [2] to "*softwarize*" the already existing IoT distributed protocols to provide the network with centralized control. Specifically, it leverages the Routing Protocol for Low-power and Lossy Networks (RPL) [3] and the 6Top Protocol (6P) [4] in the 6TiSCH stack to exert control in both routing and scheduling layers without adding a new SDN-specific protocol in the network nodes. This allows offering SDN-like capabilities with reduced overhead and energy consumption, while being compatible with current IoT standards.

However Whisper is currently limited to isolated IPv6 Low-power Wireless Personal Area Networks (6LoWPANs). This means that it is not possible to have a complete end-to-end network management from the very same IoT devices' to the core network. The contribution of this work is first, to propose a new fully end-to-end SDN architecture that includes and considers the IoT domain and second, to enhance Whisper for its integration within a network operating system, including the design of a new south-bound protocol. Finally we discuss the implementation details, present results in real hardware in order to validate the full platform, and give further insights on the potential benefits of using Whisper to provide a full network softwarization that includes the IoT network segment.

## II. Background

### A. What is Whisper and why to use it?

The use of IoT in industrial deployments to monitor and manage mission control critical infrastructures demands high reliability, reduced latency and low energy consumption. Although current Industrial IoT protocols such as the ones included in the 6TiSCH protocol stack already fulfill most of these requirements [5], they implement statically defined decisions (i.e., how routing and scheduling planes accomplish a predefined objective function). However real industrial deployments require additional flexibility and dynamic network management in order to react to malfunctioning nodes, blocked wireless channels or sudden battery depletion in nodes.

Current literature on flexibility and programmability of IoT networks focus on importing the SDN paradigm [6], [7]. However the use of SDN on Low-power and Lossy Networks (LLNs) networks is challenging since in an LLN, nodes are highly resource-constrained devices, links are unreliable

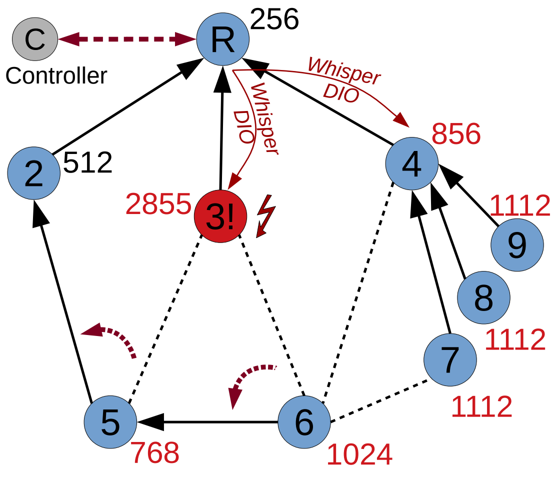
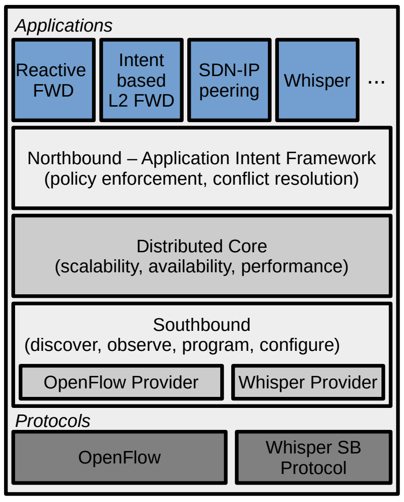


Fig. 1: Example of how Whisper can alter packet forwarding through the injection of "*fake*" RPL messages. Nodes 5 and 6 can avoid to forward packets towards node 3 upon battery depletion in node 3, and re-route their traffic through node 2 (to not overload node 4).



Applications
Reactive FWD
Intent based L2 FWD
SDN-IP peering
Whisper
...
Northbound – Application Intent Framework (policy enforcement, conflict resolution)
Distributed Core (scalability, availability, performance)
Southbound (discover, observe, program, configure)
OpenFlow Provider
Whisper Provider
Protocols
OpenFlow
Whisper SB Protocol

Fig. 2: ONOS platform architecture.

and limited in bandwidth, and the multi-hop wireless mesh topology implies the need for in-band signaling. Nonetheless, a number of works have engineered solutions to circumvent these constraints [8]–[10]. However these solutions still rely on a reliable in-band signaling channel and face important challenges due to the significant signaling overhead when scaling up the network. Other works such the one we presented, Whisper [2], partially solve these problems by combining SDN techniques with distributed IoT protocols.

In order to exert network control, a Whisper controller delivers carefully computed standard messages to the nodes to artificially change their routing and scheduling behavior. By doing this, IoT networks can be managed without modifying any bit in the firmware of the already deployed nodes (e.g., no additional SDN-specific software is needed). Due to the presence of the standard distributed protocols, signaling is minimum and the IoT network can fully perform even without the continuous presence of an SDN controller. However, this comes with the cost of a slight reduction in the network programmability. For example, routes are required to form a Direction Oriented Directed Acyclic Graph (DODAG) since RPL is a gradient-based routing protocol.

Whisper is currently designed for 6TiSCH networks since its implementation is based on RPL for the routing management and on 6P for the scheduling management. Figure 1 shows an example of how Whisper works. It shows a DODAG where each node receives a rank, which is the metric used in RPL to calculate the best path towards the root (R). Ranks are distributed through DODAG Information Objects (DIOs) messages and allow each node to select the neighbor with lowest perceived rank as its preferred parent. Whisper injects altered DIO messages in the network with an artificial rank to alter the parent choice. This implies that Whisper can centrally alter the routing table of the decentralized RPL protocol.

Scheduling is controlled in an equivalent way. Whisper relies on the 6P protocol and, by default, on the Minimal Scheduling Function (MSF) [11]. In 6TiSCH, nodes allocate cells (formed by a timeslot and a channel offset) in a local manner according to the traffic demands using a specific scheduling function (e.g., MSF). However Whisper can build a complementary scheduling function on top of MSF (e.g., that optimizes latency) by delivering 6P commands that add or delete the required cells in the nodes. By doing this, Whisper allows network programmability with minimal signaling overhead and without the need of a new SDN-specific protocol.

## B. The need for Network Operating Systems

However, Whisper has been presented as a solution for a single 6LoWPAN only, without supporting the integration with other SDN environments. Yet a full end-to-end network softwarization that includes both wired and wireless segments is crucial for a network operator. Assume for example that a wired link in the core network fails. An SDN controller could re-route its traffic through other available paths. If a sink node in an IoT network fails as well, traffic from the sensor nodes would also need to be re-routed to other sinks if possible. An efficient traffic re-routing that takes in account the state of the core network would only be possible by having an integrated end-to-end SDN controller that controls all network domains.

*1) Related work:* In the optical/wired segment the use of Network Operating Systems (NOS) [12], [13] is already common to program network layers in a platform agnostic-manner. One of the most extended NOS solutions in both research and production environments is the open-source Open Network Operating System (ONOS) project [14]. ONOS is a distributed, modular SDN control platform that allows high levels of scalability, availability and performance in large operator networks. While ONOS is mainly focused on the optical/wired segments, some works have studied how to extend the control to wireless sensor networks as well [15]–[17] (e.g., by using SDN-WISE [8]).

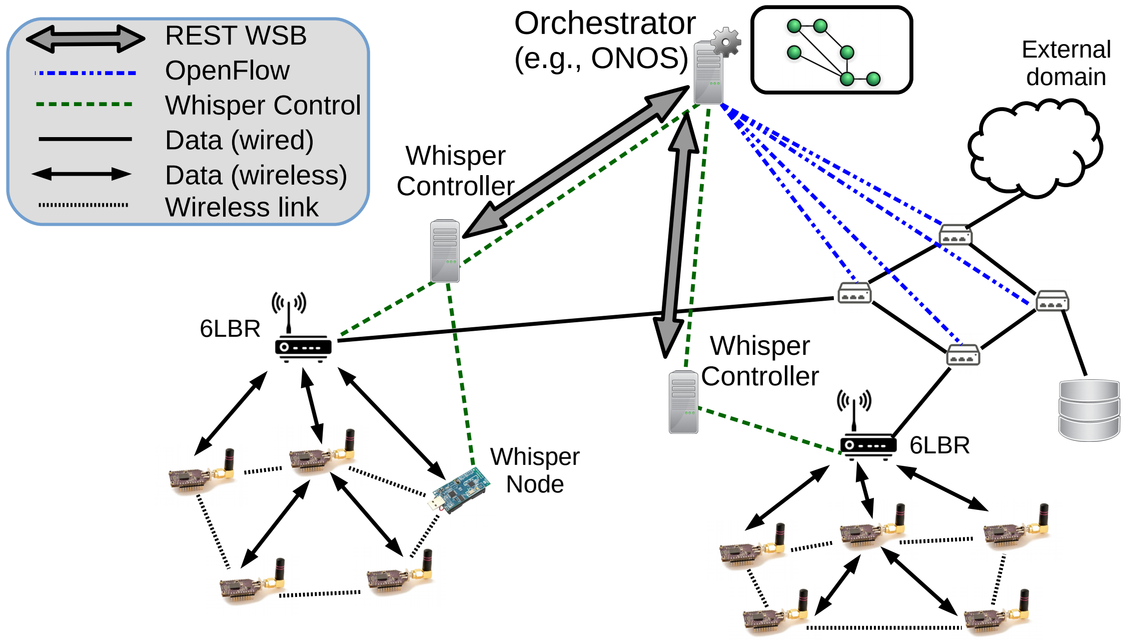


Fig. 3: Example of the orchestration architecture using Whisper.

*2) ONOS details:* A simplified layered architecture of ONOS is shown in Figure 2. In ONOS, network devices are abstracted independently of the underlying network architecture to allow interoperability between heterogeneous networks. Network devices (e.g., OpenFlow [18] switches) are managed using their specific control protocols. For each device, ONOS includes a driver that implements its communication protocol. The translation between the protocol-specific operations and the abstractions used in the upper layers is done by the Providers, located in the *Southbound* sublayer. In this sublayer, discovery and configuration functions are also implemented. The *Distributed Core*, stores all the information maintained by the system (e.g., topology, states, etc.) and provides the upper layers with path computation functions (e.g., to create and compile path *Intents*). Finally, the *Northbound* sublayer manages the network abstractions through flow rules and policies. This sublayer allows applications (e.g., a DHCP service, a learning switch controller, etc.) to consume and manipulate aggregated information from the *Core* sublayer. Application functionality ranges from displaying network topologies to complex traffic engineering for different traffic classes.

## III. End-to-End Orchestration for IoT

In order to allow Whisper to inter-operate with other SDN systems, we present a new orchestration architecture that integrates the existing Whisper controller in a wider SDN context to enable the convergence of IoT networks with wired/optical networks through SDN-based global control (detailed in Section III-A). This is done by implementing a Whisper module compatible with a NOS (e.g., ONOS) to abstract the IoT network. This abstraction allows monitoring and controlling the IoT network in both routing and scheduling planes. In order to translate high-level control abstractions to actual Whisper primitives, we have also implemented a new Southbound Whisper protocol (detailed in Section III-B) which is available at the controllers through a REST API and is eventually exerted in the 6LoWPAN networks through a local Whisper controller located in the IoT sink (root).

### A. Proposed Architecture

In order to provide full end-to-end softwarization, the SDN capabilities should be present in all the systems present in the network, from sensor to host, including the 6LoWPAN segments. In Figure 3 we present a holistic architecture that is orchestrated by ONOS and where the LLN is softwarized by Whisper. The ONOS controller (it can be distributed) manages directly the wired switches through OpenFlow. However, in order to have control over 6LoWPAN networks, the controller interacts with the local Whisper controllers to translate and relay the controller messages for the IoT nodes. The Whisper controllers are run in the or 6LoWPAN Border Routers (6LBR) and can control the sensor network either directly through the 6LBR itself, or through the Whisper nodes, specific wireless nodes that can be strategically placed in the network to augment the monitor and control capabilities. With such architecture, an orchestrator is able to abstract both wired and wireless networks providing integral end-to-end control

Whisper controllers periodically report to the ONOS controller with network statistics (e.g., topology, link costs, schedules, etc.) through the Whisper REST API (see Section IV for details). With this information, the ONOS controller updates its internal topology stored in its core, and performs actions according to the policies and applications' requirements whenever needed. These actions are delivered through OpenFlow to the SDN-capable switches and through the Whisper Southbound protocol to the IoT nodes (see Section III-B). Traffic

| WSB Segment | Dir | Name | Arguments | Output |
|---|---|---|---|---|
| ONOS Controller Whisper Controller | DL | ParentSwitch | NodeA,NodeB | ResponseCode |
| | DL | AddCell | NodeA,NodeB,Cells* | ResponseCode,CellList |
| | DL | DeleteCell | NodeA,NodeB,Cells*,Clear* | ResponseCode |
| | UL | NetworkUpdate | NodeID,Topology*,LinkCost*,Schedules* | Json file |
| Whisper Controller 6TiSCH | DL | SwitchRemote | TargetNode,FirstHop,Rank,ReliableSwitch* | ResponseCode |
| | DL | SwitchImpersonate | WhisperNode,TargetNode,ImpID,Rank,ReliableSwitch* | ResponseCode |
| | DL | PropagateRank | Rank,NodeID | ResponseCode |
| | DL | 6PRequest | NodeA,NodeB,Cells* | ResponseCode,CellList |
| | DL | UpdateSolicitation | NodeID,Ranks*,Topology*,LinkPDR*,Schedules*,State* | ResponseCode |
| | UL | UpdateReport | NodeID,Ranks*,Topology*,LinkPDR*,Schedules*,State* | Stats |

TABLE I: Most representative messages of the Whisper Southbound (WSB). Optional fields are denoted with *.

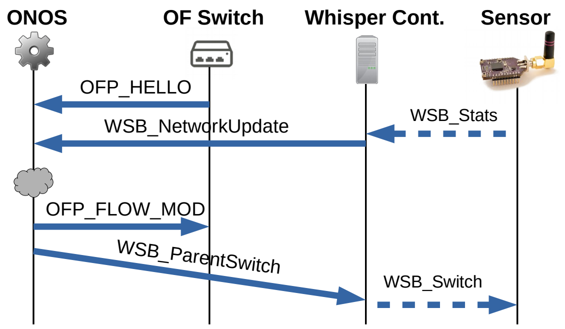


Fig. 4: Example of the messages exchanged, using OpenFlow for the wired segments and the Whisper Southbound protocol (WSB) to manage the 6LoWPAN network.

flows are routed through end-to-end *Intents* from the sensor node to the destination. This way, changes in the traffic paths do not compromise the performance of the flows, since *Intents* ensure an end-to-end path based on agreed constraints. Upon network changes, the *Intent* will re-route automatically the flow to accomplish its constraints.

### B. Whisper Southbound protocol

The Whisper Southbound protocol is an enhanced, generalized version of the Whisper primitives described in [2] to make them compatible with a generic SDN controller. Table I describes the most relevant messages. The *ONOS Controller - Whisper Controller* segment is an abstraction of the full Whisper protocol that hides the complexity of the 6TiSCH network to the SDN controller (e.g., ranks, RSSI, etc.). It consists only by 4 messages: *ParentSwitch* to perform the re-routing of next hop of a sensor node, *AddCell/DeleteCell* to manage nodes' schedules and *NetworkUpdate*, which contains incoming aggregated information from the 6TiSCH network.

In the *ONOS Controller - Whisper Controller* segment, the protocol is augmented with the characteristics of each specific 6TiSCH network. This means that the Whisper controller translates the abstracted messages from the SDN controller to the actual Whisper primitives needed to perform SDN controller's orders (e.g., Figure 4). For example, the *ParentSwitch* message has to be translated to one or more messages in the 6TiSCH network (e.g., it could require a *SwitchRemote* message and a *PropagateRank* message each of them with specific Rank values). Likewise, a number of *UpdateReports* from different Whisper nodes are aggregated at the Whisper controllers in one single *NetworkUpdate* message destined to the ONOS core. Each one of the control messages sent in the *ONOS Controller - Whisper Controller* segment can be directly mapped to one of the Whisper primitives [2].

## IV. Implementation details

### A. Whisper module for ONOS

The Whisper module for ONOS mainly consist of two parts: the *Whisper Provider* submodule and the *Whisper Protocol* submodule (see Figure 2). The *Whisper Provider* is in charge of communicating the network abstraction to the ONOS core, adding and updating links, devices, hosts and intents. The *Whisper Protocol* feeds the *Whisper Provider* with information coming from the Whisper Southbound protocol. In order to send and receive *NetworkUpdate* messages, a REST API is deployed in the ONOS controller to be accessible for all the local Whisper controllers. Control messages are also delivered through the REST API deployed at the Whisper controllers.

From the point of view of ONOS, sensor nodes are treated as "*special*" switches. However, since sensor nodes are IPv6-enabled, the *Whisper Provider* adds virtual hosts to each sensor node to assign them IPv6 addresses. This way path *Intents* can be created end-to-end directly from sensor nodes to hosts. Finally, additional elements at the application level have been included to modify the graphic user interface and to add a Whisper command line interface.

### B. Local Whisper controller

The local Whisper controllers run inside OpenVisualizer, a tool to interconnect a 6TiSCH network into the Internet. OpenVisualizer is included in the OpenWSN project [19], which is currently the most up-to-date implementation of 6TiSCH. On one side, the Whisper controller communicates with the ONOS controller using REST operations. Parallelly, the controller also deploys the same REST API to receive commands from the orchestrator (see again Figure 3).

The Whisper intelligence is located at the Whisper controller. First, it needs to aggregate and compile partial information from the 6TiSCH network to be delivered to the ONOS contoller in a *NetworkUpdate* message. Some network information is directly available in the Whisper controller. For example, the DODAG topology is directly obtained from the

Destination Advertisement Object (DAOs) messages, which arrive to the Whisper controller through OpenVisualizer. However, in order to obtain nodes' ranks the Whisper controller require to send *UpdateSolicitation* messages either to the DODAG root or to the Whisper nodes. They subsequently will internally send unicast DODAG Information Solicitation (DIS) messages to request the rank and report it back to the controller. Scheduling information and 6P sequence numbers are obtained by tracking 6P messages in the root and Whisper nodes. Finally, links between neighbors and their PDR, are tested and managed through source routed pings, DIS messages and 6P commands.

On the other hand, the Whisper controller also needs to receive the control messages from the ONOS controller and perform the required actions. In this sense, the Whisper controller needs to calculate which primitives are needed for a specific action. While scheduling related primitives are mapped directly to their corresponding Whisper Southbound message (the Whisper ONOS module performs the scheduling management), routing related primitives requires to calculate which primitives have to be sent and with which artificial rank value. This is done using the algorithm Switch Parent algorithm described in [2].

## V. Results

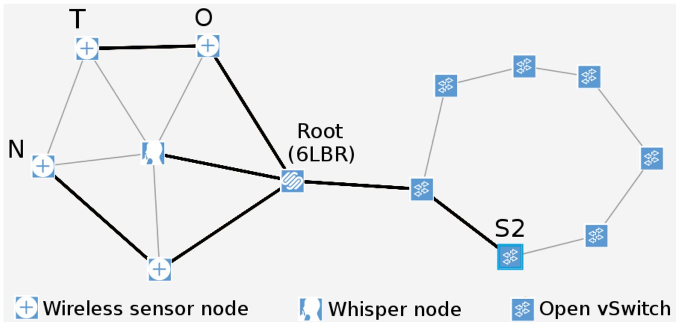
In order to experimentally validate the integration between ONOS and Whisper, we use a small 6TiSCH network and a wired SDN-enabled network where we test the end-to-end routing and scheduling management use case. We have deployed 6 OpenMotes-CC2538 nodes [20] running OpenWSN REL-1.24.0 to build the 6TiSCH network. For the wired network, we have emulated 7 OpenFlow-enabled switches through Mininet [21]. In order to orchestrate both networks we use ONOS 2.1.0 which includes the Whisper module[1].

Figure 5 shows the tested network. Each sensor node sends periodic data (1 packet every 3 seconds) to a host connected to the switch *S2* through an *Intent* (bold lines in the figure). In the wireless segment, the actual physical path of the data packets is directly mapped to the logic topology of the 6TiSCH network. The test simply consists of performing a routing change by the decision of the ONOS controller, first in the wireless segment, which also includes scheduling control, and later, a path change in the wired network.

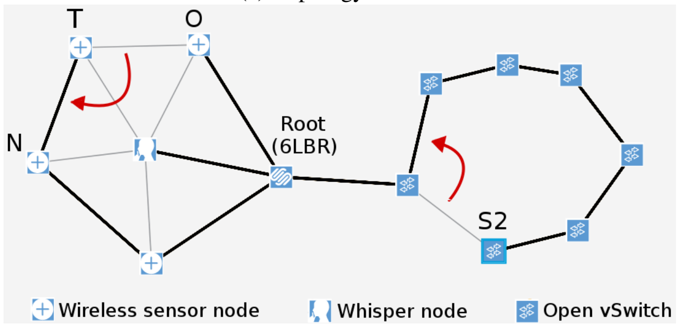
In order to test this, we log the traffic from the wireless sensor node *T* (target) to a host connected to the switch *S2*. Figure 6 shows the evolution of the latency of that traffic during the experiment, showing the latency in the wireless segment, the latency in the wired segment and the total end-to-end latency (TSCH+Wired). In this experiment we show how the same ONOS controller instance can alter the paths in both the wired and the wireless segment.

From the bootstrap of the network, the Whisper node probes the network nodes, obtaining the full physical topology by

[1]All Whisper related implementation developed for ONOS and OpenWSN is available in https://github.com/imec-idlab/whisper-repository



(a) Topology before.



(b) Topology after.

Fig. 5: Topology graph displayed in the ONOS web GUI before and after the two path changes.

augmenting the already known DODAG with the existing physical links. This will let ONOS know that the parent switch for node *T* is actually feasible. Around $t = 300$ *s*, the ONOS controller triggers the message *ParentSwitch* to change the next hop of *T* from *O* (old parent) to *N* (new parent). Upon receiving the order, the Whisper controller located in the root of the 6TiSCH network commands the Whisper node to proactively allocate cells between node *T* and node *N* beforehand in order to perform an smooth parent switch without packet loss. Since the new route has one wireless hop more, the TSCH latency increases in about 1.1 *s*. This latency increase will depend on the nodes' scheduling configuration along the path.

The hacksaw pattern shown in Figure 6 for the TSCH latency is a common behavior in 6TiSCH networks when timers for sending packets are uncoupled with the TSCH period. Also, several peaks in latency of about 1 *s* are observed, which are caused by packet re-transmissions (i.e., a PHY drop). If a packet is dropped, it will be re-transmitted in the next TSCH frame, after 101 *slots* x 10 *ms timeslot* = 1.01 *s*.

Afterwards, around $t = 600$ *s*, the ONOS controller orders a subsequent path change in the wired network (e.g., the last link before *S2* is down). In order to do this, the ONOS controller re-compiles the path *Intent* and distributes the required OpenFlow commands to each of the switches. This causes a re-route around the ring topology that increases the latency in 125 *ms* (i.e., 5 extra hops). Wired links are configured each with an artificial added delay of 25 *ms* in order to clearly perceive the path change. Consequently, the accumulative delay also increases in 125 *ms*.

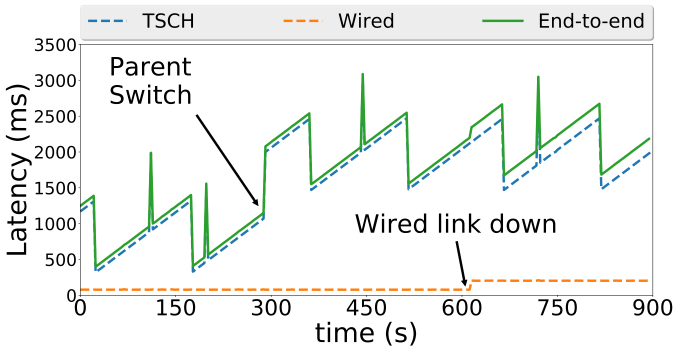


Fig. 6: Evolution of the end-to-end latency from node T to a host connected to switch S2.

## VI. Conclusion and future work

Currently a number of solutions address the implementation of SDN on IoT networks in order to obtain network programmability and flexibility. In between a fully centralized management of the IoT network and a fully distributed one, Whisper stays as a trade-off solution, merging both SDN worlds. It allows to perform centralized network management but still depends on standard distributed IoT routing and scheduling protocol to control the nodes.

In this work we have presented a new higher-level orchestration architecture for Whisper that allows full end-to-end control including the wired segments. We have shown an implementation of a Whisper module for ONOS that allows the orchestrator to interact with both 6TiSCH and wired networks in order to exert an holistic network management, without renouncing to the robustness and efficiency of distributed protocols in the 6TiSCH segment. Additionally, we have created the Whisper Southbound protocol that allows to shift the Whisper scope from the edge to the controller. Finally we have tested the full system composed by an emulated OpenFlow-enabled switch network and a 6TiSCH network using real hardware. Results show that the ONOS controller is able to control the routing in both network segments, and for the case of the 6TiSCH network, also its scheduling plane. Regarding the question, "*Is SDN actually interesting in IoT?*", these results seem to point towards: "*SDN is not only interesting, but also essential for end-to-end flexibility*", and may open a promising research path on efficient IoT management, if not for all IoT deployments, definitively for IoT legacy deployments. The integration with other SDN-on-IoT approaches also remains as an interesting research direction.

## References

[1] H. I. Kobo, A. M. Abu-Mahfouz, and G. P. Hancke, "A Survey on Software-Defined Wireless Sensor Networks: Challenges and Design Requirements." *IEEE Access*, vol. 5, no. 1, pp. 1872–1899, 2017.

[2] E. Municio, J. Marquez-Barja, S. Latré, and S. Vissicchio, "Whisper: Programmable and Flexible Control on Industrial IoT Networks," *Sensors*, vol. 18, no. 11, 2018. [Online]. Available: http://www.mdpi.com/1424-8220/18/11/4048

[3] R. Alexander, A. Brandt, J. Vasseur, J. Hui, K. Pister, P. Thubert, P. Levis, R. Struik, R. Kelsey, and T. Winter, "RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks," RFC 6550, Mar. 2012. [Online]. Available: https://rfc-editor.org/rfc/rfc6550.txt

[4] Q. Wang, X. Vilajosana, and T. Watteyne, "6TiSCH Operation Sublayer (6top) Protocol (6P)," RFC 8480, Nov. 2018. [Online]. Available: https://rfc-editor.org/rfc/rfc8480.txt

[5] X. Vilajosana, T. Watteyne, M. Vučinić, T. Chang, and K. S. Pister, "6TiSCH: Industrial Performance for IPv6 Internet-of-Things Networks," *Proceedings of the IEEE*, 2019.

[6] S. Costanzo, L. Galluccio, G. Morabito, and S. Palazzo, "Software defined wireless networks: Unbridling sdns," in *Software Defined Networking (EWSDN), 2012 European Workshop on*. IEEE, 2012, pp. 1–6.

[7] P. Thubert, M. R. Palattella, and T. Engel, "6TiSCH centralized scheduling: When SDN meet IoT," in *Standards for Communications and Networking (CSCN), 2015 IEEE Conference on*. IEEE, 2015, pp. 42–47.

[8] "SDN-WISE: Design, prototyping and experimentation of a stateful SDN solution for wireless sensor networks."

[9] "TinySDN: Enabling multiple controllers for software-defined wireless sensor networks."

[10] M. Baddeley, R. Nejabati, G. Oikonomou, M. Sooriyabandara, and D. Simeonidou, "Evolving SDN for Low-Power IoT Networks," in *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*. IEEE, 2018, pp. 71–79.

[11] T. Chang, M. Vučinić, X. Vilajosana, S. Duquennoy, and D. Dujovne, "6TiSCH Minimal Scheduling Function (MSF)," Internet Engineering Task Force, Internet-Draft draft-ietf-6tisch-msf-03, Apr. 2019, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-ietf-6tisch-msf-03

[12] A. Greenberg, G. Hjalmtysson, D. A. Maltz, A. Myers, J. Rexford, G. Xie, H. Yan, J. Zhan, and H. Zhang, "A clean slate 4D approach to network control and management," *ACM SIGCOMM Computer Communication Review*, vol. 35, no. 5, pp. 41–54, 2005.

[13] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, "NOX: towards an operating system for networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 3, pp. 105–110, 2008.

[14] 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*. ACM, 2014, pp. 1–6.

[15] "Towards a software-defined Network Operating System for the IoT, author=Anadiotis, Angelos-Christos G and Galluccio, Laura and Milardo, Sebastiano and Morabito, Giacomo and Palazzo, Sergio," in *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*. IEEE, 2015, pp. 579–584.

[16] M. Ojo, D. Adami, and S. Giordano, "A SDN-IoT architecture with NFV implementation," in *2016 IEEE Globecom Workshops (GC Wkshps)*. IEEE, 2016, pp. 1–6.

[17] S. Fichera, M. Gharbaoui, P. Castoldi, B. Martini, and A. Manzalini, "On experimenting 5G: Testbed set-up for SDN orchestration across network cloud and IoT domains," in *2017 IEEE Conference on Network Softwarization (NetSoft)*. IEEE, 2017, pp. 1–6.

[18] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[19] T. Watteyne, X. Vilajosana, B. Kerkez, F. Chraim, K. Weekly, Q. Wang, S. Glaser, and K. Pister, "OpenWSN: a standards-based low-power wireless development environment," *Transactions on Emerging Telecommunications Technologies*, vol. 23, no. 5, pp. 480–493, 2012.

[20] X. Vilajosana, P. Tuset, T. Watteyne, and K. Pister, "OpenMote: open-source prototyping platform for the industrial IoT," in *International Conference on Ad Hoc Networks*. Springer, 2015, pp. 211–222.

[21] K. Kaur, J. Singh, and N. S. Ghumman, "Mininet as software defined networking testing platform," in *International Conference on Communication, Computing & Systems (ICCCS)*, 2014, pp. 139–42.