Fog Computing over Challenged Networks: a Real Case Evaluation

Fog Computing over Challenged Networks: a Real Case Evaluation

Gabriele Castellano
Computer and Control Engineering
Politecnico di Torino, Italy
gabriele.castellano@polito.it

Fulvio Risso
Computer and Control Engineering
Politecnico di Torino, Italy
fulvio.risso@polito.it

Riccardo Loti
Tierra Telematics Design, Italy
rloti@tierratelematics.com

***Abstract*—Fog computing enables a multitude of resource-constrained end devices (e.g., sensors and actuators) to benefit from the presence of fog nodes in their close vicinity, which can provide the required computing and storage facilities instead of relying on a distant Cloud infrastructure. However, guaranteeing stable communication between end devices and fog nodes is often not trivial. Indeed, in some application scenarios such as mining operations, building sites, precision agriculture, and more, communication occurs over Challenged Networks e.g., because of the absence of a fixed and reliable network infrastructure. This paper analyzes the applicability of Fog Computing in a real *Industrial Internet of Things* (IIoT) environment, providing an architecture that enables disruption-tolerant communication over Challenged Networks and evaluating the achieved performance on an open-source prototype implementation.**

I. Introduction

Fog Computing is gaining momentum by extending the Cloud paradigm to the edge of the network, thus enabling end devices (e.g., IoT devices) to benefit from computing and storage facilities in their close vicinity, through the deployment of fog nodes at the edge of the network.

However, guaranteeing stable communication between end devices and fog nodes is not trivial in *Industrial IoT* (IIoT) scenarios, such as mining operations, building sites, precision agriculture, and more, where end devices may be fleets of heavy-duty vehicles, construction equipment, tractors or even smart agriculture sensors, which may not always be connected to a fixed and reliable network infrastructure. Usually, these devices *(i)* generate data (e.g., sensor measurements) during their operating cycle, that have to be sent to applications running on fog nodes and *(ii)* need for periodic updates (e.g., firmware, work plans) from the Fog.

As shown in Figure 1, operations are usually performed on challenging environments, where communication occurs over small-range wireless channels (Wireless Mesh Network) and the network infrastructure presents the characteristics of *Challenged Networks*; therefore, end devices may spend long periods (even their entire operating cycle) in remote locations, without the possibility to directly communicate with a fog node, not even to the Cloud.

Operating in these environments, often end devices are able to delivery generated data and receive updates only when they return close to a fog node, e.g., at the end of their operating cycle. Although in this scenario a real-time transfer of information is not strictly needed, being able to constantly propagate data and receive updates with a certain delay bound would bring considerable advantages, such as *(i)* fewer storage resources needed on end devices to store data; *(ii)* fog applications may constantly perform analytic operations on new data; *(iii)* vehicles may operate for a significant longer time, even days, without going back to their base (i.e., near a fog node).

Fig. 1: Example of a dynamically evolving Wireless Mesh Network connecting multiple construction devices.

In this paper we investigate the applicability of Fog Computing in a real IIoT environment, proposing an architecture that enables communication between end devices and fog nodes without requiring a fixed network infrastructure. Furthermore, fog applications running on end devices (e.g., data producers) are kept unchanged, independently from the actual network connectivity. This potentially enables an entire class of existing applications to operate also on unconventional scenarios such as challenged networks. To this end, we focus a smart agriculture scenario, where a fleet of machineries sends data and receives updates to/from the fog nodes over the MQTT publish/subscribe standard protocol [1], commonly adopted by widespread IIoT applications. In particular, our contribution are as follows.

Architectural Contribution. We propose a communication architecture that enables the delivery of data originally transmitted with the MQTT protocol over possible challenged networks operating between end devices and fog nodes; the propagation exploits occasional contacts occurring between machineries during their movements to establish *opportunistic connections* and exchange data; in this way, data is conveyed toward the fog node by means of the *store-carry-and-forward*

paradigm implemented by a Delay/Disruption Tolerant Networks (DTN). Our architecture also ensures transparency with respect to existing IIoT applications that make use of MQTT APIs, which do not need to be modified.

System Contribution. We present the open source proof-of-concept implementation [2] of a system that exploits our communication architecture.

Experimental Contribution. Performance of our approach has been evaluated through measurements over both a physical prototype and a virtualized system, finding encouraging results and demonstrating advantages and applicability of this paradigm in IIoT environments.

The reminder of this paper is organized as follows. Next section highlights our contribution compared to the existing work. Section III presents the proposed communication architecture, while our prototype implementation is detailed in Section IV. Section V presents our evaluation results and Section VI concludes the paper.

II. Related Work

Although the applicability of Fog Computing in IoT scenarios has deserved a considerable amount of attention [3], there are still many open research topics, particularly with respect to industrial use cases [4], [5]. The state-of-the-art IoT in industries is summarized in [6], where authors describe key applications and identify the following as main technical challenges: service discovery methods and object naming services [4], [5], scalability on the number of connected things [5]; heterogeneity of underlying communication protocols [7]; lack of architectures for sensor networks communication, resilience to physical network disruption, and node peering [8].

The relevance of Fog Computing in IIoT is addressed in some recent works [9], [10]. In [9] authors identifies the main IIoT challenges, demonstrating Fog Computing as a key enabling technology to address them, also detailing the main infrastructure components. [10] proposes a Fog Computing-based architecture to make information become timely accessible anywhere in industrial automation systems. However, just few works on Fog Computing/IoT address the problem of connectivity on challenged networks. To this end, [11], [12] propose to extend IoT connectivity over disrupted environments with the help of the DTN approach. Particularly, [11] provides an implementation of Constrained Application Protocol (CoAP) over DTN to enable IoT devices based on CoAP to operate on a disrupted environment, while [12] analyzes the behavior of MQTT for Sensor Networks (MQTT-SN) over a DTN implementation.

This paper extends the existing work with a real experimental setup, focusing *(i)* on transparency toward existing IIoT applications regarding communication protocols APIs, and *(ii)* definition of the actual physical network setup.

III. Communication System Architecture

This section presents a communication system architecture that enables the MQTT data propagation between end devices and fog nodes over challenged networks.

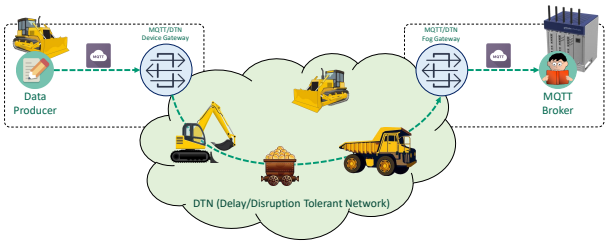


Fig. 2: Overall communication architecture: data is wrapped in DTN bundles and propagated through the *store-carry-and-forward* paradigm.

A. Overall Approach

Our approach models both fog nodes and end devices (i.e., operating machinery) as nodes of a DTN. Thus, the *store-carry-and-forward* paradigm, exploits the continuous movement of the operating machinery to enable data flowing toward the destination. Data is delivered by the DTN protocol, which extends the existing network stack (e.g. TCP/IP, Bluetooth, etc.) with a *bundle protocol* layer that encapsulates application messages; these are delivered hop-by-hop to another DTN node based on tunable forwarding behaviors, such as *epidemic* (data is replicated on all encountered nodes), *prophet* (data is forwarded only in the direction that looks the best), and more [13]. Nodes of the DTN are identified by an Endpoint Identifier (EID) (e.g., `dtn://device1`), while each application is identified by extending the local EID with an *application token* (e.g., `dtn://device1/app1`).

An high level view of our communication architecture is depicted in Figure 2; in particular, it shows the communication between an originating vehicle (on the left) and the target Fog Computing node (on the right), which hosts the MQTT Broker. In order to deliver produced data to the MQTT Broker, each vehicle sends a data bundle to other peers through a small-range wireless connection. This process is repeated by the second vehicle when it detects other opportunistic connections, until the data carrier enters in range of the fog node, hence the data bundle is delivered to the destination (together with other bundles possibly collected in the meanwhile).

As shown in the figure, in order to preserve transparency on end applications, namely to allow them to interact with the MQTT primitives without being aware of the underlying (not connected) network, each node features a *gateway* that conveys MQTT messages over the DTN.

B. General Architecture

We describe our system architecture distinguishing between *upstream communication* (Device-to-Fog) and *downstream communication* (Fog-To-Device).

Upstream communication. Sensors operating on end devices produce information that needs to be collected and routed toward fog nodes, located at the edge of the network.

Figure 3 details both the architecture of an end device that produces and publishes data and the architecture of the fog node where a broker receives published data. Both devices feature a *DTN Daemon* that extends the network stack with the support for the DTN bundle protocol and a *Convergence*

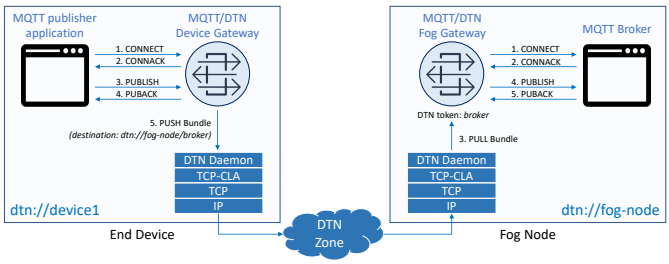


Fig. 3: Upstream communication system overview.

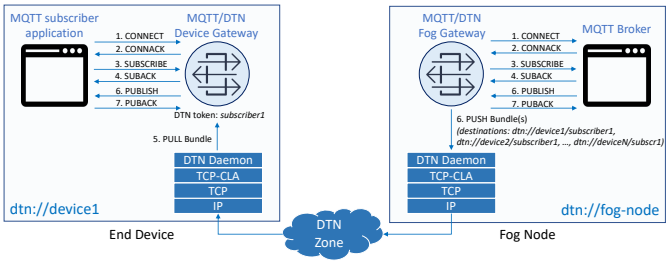


Fig. 4: Downstream communication system overview.

Layer Adapter that enables DTN communication over the TCP stack (TCP-CLA). Each end device hosts an *MQTT Publisher*, i.e., the application that generates data through equipped sensors, and an *MQTT/DTN Device Gateway*. The role of the latter is twofold: it *(i)* emulates a local MQTT Broker thus allowing the application to transparently connect and publish data, apparently relying only to the MQTT protocol, and *(ii)* encapsulates received messages into bundles with destination `dtn://fog-node/broker`, which are delivered to the DTN Daemon. Bundles are then pushed on the output queue, thus making them ready to be forwarded to other mobile devices. On the other hand, the fog node hosts the data receiver, i.e., a *MQTT/DTN Fog Gateway* and an *MQTT Broker*. Similarly to the counterpart on the end device, the gateway acts as bridge between the MQTT protocol and the DTN stack: it *(i)* registers the application token, namely *"broker"*, on the DTN Daemon in order to "extract" all bundles with destination `dtn://fog-node/broker` from the DTN and *(ii)* sends data received in this way to the the broker, by performing an MQTT publish that preserves the original MQTT payload.

This approach introduces an abstraction layer that enables transparency on applications: the producer continues to generate data using the usual MQTT protocol, without being aware of how messages are propagated toward the broker. The only required change on end devices is to configure the broker address as `localhost`, which could even be avoided by running a TCP transparent proxy on the end device itself.

Downstream communication. In this case, data flows from the MQTT Broker to *MQTT Subscribers* on the end devices. Messages may be either data generated by other end devices (and collected through upstream communication) or sent from Fog/Cloud control applications (e.g., firmware updates). Note that downstream messages may have multiple destinations, as there may be more than one subscriber on the same topic (e.g., firmware update on a specific model of end device).

Despite in this case the main data stream goes from the fog node to the end devices, some information are needed to travel toward the fog node (namely, MQTT connection and subscribe messages sent by end devices applications). For this, a mechanism analogous to the upstream communication method described so far is used, enabling the MQTT/DTN Fog Gateway to know which are the topics subscribed for each device, so that it can act as a client towards the broker, hence performing a subscribe for all topics of interest.

As shown in Figure 4, in this case the role of the two gateways is inverted. On the fog node, the MQTT/DTN Fog Gateway receives publish messages from the broker; for each message received in this way, it potentially pushes multiple bundles on the DTN (Figure 4, step 6), one for each end device destination (i.e., all the subscribers). On each end device, an MQTT/DTN Device Gateway emulates the broker, thus allowing the final application to subscribe on desired topics even when a connection with the fog node is not available; whenever the gateway pulls a bundle out from the DTN, the enclosed MQTT message is delivered to the subscriber application.

Again, both the application on board of the end device and the broker operate on an abstracted MQTT layer that enables transparency and allows them to send MQTT protocol messages regardless of the presence of a network connection.

C. Proof-of-concept: Telemetry System

The above upstream communication architecture has been used to design a telemetry system addressing a real use case scenario: a fleet of vehicles are equipped with sensors that, periodically, measure temperature, humidity and acidity of the ground (thus we call them *Sensing End Devices*); the stream of data generated this way is collected by a nearby fog node, where some applications perform preliminary analytic before to send aggregated data to the Cloud.

The architecture of our PoC telemetry system is depicted in Figure 5. A fog node is located in a particular point of interest (e.g., base station) and is equipped with a WiFi access point; each end device is able to communicate directly with the fog node when his location is within the range of coverage of the WiFi network. In this case, the end device delivers to the final destination (i.e., the MQTT Broker on the fog node) all the bundles collected through the DTN layer. Two fog applications are attached to the MQTT Broker and consume incoming data: *(i)* a monitoring application, which enables local visualization of raw data, and *(ii)* a filtering and aggregation application, which performs preliminary analytics on raw data and publishes cleaned data on a different MQTT queue. Similarly, but with a larger granularity, the Cloud collects cleaned data from all the fog nodes, in order to aggregate and analyze measurement from multiple locations and enable monitoring, work planning and production optimization.

IV. Prototype Implementation

This section presents a prototype of the above Telemetry System, which is based on a fog node provided by Nebbiolo

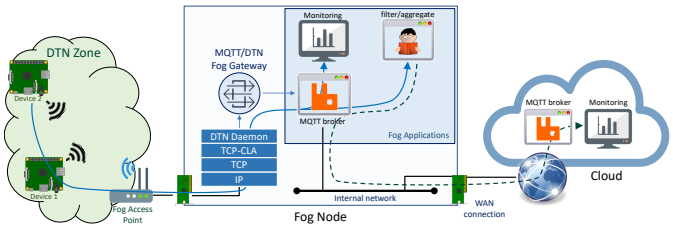


Fig. 5: Overview of the telemetry system realized as a Proof-Of-Concept of the proposed communication architecture.

Technologies [14] and a number of Raspberry Pi as sensing end devices, plus a remote virtual machine in the Cloud.

A. Sensing End Device

Each sensing end device is a Raspberry Pi (Model B+, with 512 MB of RAM), equipped with a Raspbian GNU/Linux 9.1, kernel 4.9.41 and an additional WiFi dongle based on the Realtek RTL8188EU; the MQTT protocol implementation is based on the *Paho MQTT* project [15].

Network Configuration. Opportunistic wireless communications between two close end devices has been performed through the WiFi technology in IBSS (a.k.a. Ad-Hoc) mode.

To enable the Ad-Hoc connection on the network adapter, the Realtek driver available for the Raspberry Pi have been slightly modified, due to the presence of some bugs on the implementation of the IBSS protocol (bug fixes have been release on a forked repository, available at [16]). In particular, the bugs related to the *IBSS merge* algorithm, which addresses the case when two devices configured in IBSS mode on the same network are not able to communicate if they become reciprocally visible after a long period of remoteness. This happens because the IBSS mode operates through BSSID negotiation among nodes of the same Ad-Hoc network; when some devices join the network after a previous negotiation, all peers need to realign and chose a common BSSID. The WiFi Direct technology was also considered in place of the WiFi Ad-Hoc mode, but it was dropped because of the necessity to define *master* (formally, *group owner*) and *slave* roles, which is not straightforward when two arbitrary end devices connect together; in addition, it required also a DHCP server on the master, which features a very similar problem.

In our setup, the wireless interface of each end device is configured in *concurrent mode*, i.e., it exports to the OS two virtual interfaces. One is used for the Ad-Hoc communication among end devices and binds only to the IPv6 protocol stack, so that device-to-device communication exploits IPv6 Link Local addresses without the necessity of a DHCP server, which would be needed instead in case of the IPv4 protocol. The second interface is used for the connection with the fog node access point and it is configured with an IPv4 address obtained from the DHCP server running on the fog node itself. For the sake of precision, the second interface was needed because of the unavailability of the drivers to enable the WiFi ad-hoc mode on the fog node as well.

DTN Configuration. Our DTN network runs on the IBR-DTN [17] implementation of the Bundle Protocol. The DTN Daemon exports a socket-based interface to applications, with multiple Convergence Layer Adapters (CLAs) supporting various protocols (TCP, TLS, etc.). Moreover, it uses the Internet Protocol Neighbor Discovery (IPND) to identify nearby peers and perform the binding process. The original IBR-DTN implementation included a bug affecting IPND while communicating over IPv6; our fixes to the problem have been accepted by the maintainer and included in the official repository.

On each device, the DTN Daemon has been configured with the EID equal to the host name. Moreover, we selected an *epidemic* routing strategy; hence, an end device always attempts to transfer carried bundle to any connected peer, no matter the direction where the latter is moving. Flooding is stopped when the device connects with the final destination. Devices store carried bundles in a local SQLite database.

Inter-Devices Connection. After the physical connection between two devices has been established (through WiFi Ad-Hoc), the DTN Daemon on each device detects the presence of a new neighbor, through IPND beacons sent every second. At this point, the two DTN Daemons establish a TCP connection and start to exchange bundles through the TCP-CLA over IPv6. The connection with the fog node follows a similar workflow, except that the DTN Daemon operates over the IPv4 stack, thus it waits for an IP address from the DHCP server before exchanging beacons with the IPND protocol.

MQTT/DTN Device Gateway. In our prototype, the MQTT/DTN Device Gateway is a Python module that *(i)* emulates an MQTT Broker providing a transparent interface to publish messages for sensing applications, and *(ii)* interacts with the API exported by the IBR-DTN Daemon to create bundles and push them on the output DTN queue. The EID of the fog node, statically configured on each end device, is used as destination.

B. Fog Node

The fog node is a Nebbiolo NFN-300, with a single compute node fogLet NFL-1000-C [18]. It features an Intel Core-i5 4402E and runs the Nebbiolo Fog Operating System [19], which provides a KVM-based hypervisor that hosts various VMs, each one dedicated to different operations (e.g., administration, Cloud interaction, applications), and Docker container support. In our setup we deployed a Linux virtual machine (OT1VM) that hosts the needed architecture modules as Docker containers.

MQTT Broker. RabbitMQ has been used as MQTT server (a.k.a. broker) on the fog node, executed in a Docker container on the OT1VM and configured so that all messages received from sensing end devices (i.e., with topic `sensors-data`) are duplicated and pushed on two message queues (namely, `data-monitoring` and `data-aggregation`). These queues are consumed respectively by a monitoring software (we used the ELK stack [20] for this purpose) and an application that performs filtering and aggregation in order to prepare data for the Cloud.

DTN Configuration. The IBR-DTN Daemon has been deployed on the Nebbiolo fog node inside a Docker container and configured in the same way as described for the end devices

(Section IV-A). In this case we configured a single TCP-CLA, i.e., the one for the IPv4 stack associated to the adapter directly connected to the WiFi access point (bottom left interface of the fog node in Figure 5). In this case, the EID has been set to `fog-node`.

MQTT/DTN Fog Gateway. The MQTT/DTN Device Gateway is deployed as a Docker container as well; similarly to the counterpart described for the sensing end devices, it has been implemented as a Python application. This module registers the EID `dtn://fog-node/broker` on the IBR-DTN Daemon, so that it is notified whenever a bundle with that destination is received. Notifications are extracted from a queue and managed performing a *pull* operation through the IBR-DTN APIs, thus fetching the new received bundle. The Gateway also includes an MQTT client (based on Paho MQTT), so that messages received through DTN bundles are published on the corresponding queue of the RabbitMQ broker (in our case, `sensors-data`).

C. Cloud

The Cloud component of our telemetry system prototype has been implemented on a remote machine that, similarly to the Fog counterpart, runs all software modules on separated Docker containers: *(i)* an instance of the RabbitMQ Broker that features three queues and collects overall data about temperature, humidity and acidity of the ground, sent by the "filtering and aggregation" applications running on each fog node; *(ii)* an higher level ELK stack-based application that consumes messages from the three queues and enables overall data monitoring and daily aggregation.

V. Experimental Results

The proposal communication architecture has been validated through measurements over our prototype implementation, with tests both on physical devices and on a system simulated through virtualization technologies.

A. Physical devices connection

We performed some tests using physical sensing end devices in order to measure the time needed by the DTN layer to complete the neighbor discovery and bind over WiFi during an opportunistic connection between two devices.

The test-base consists of two Raspberry Pi B+ v1.2 (configured how described in Section IV). The installed version of IBR-DTN is the 1.0.1. Packets exchanged between the devices during the discovery and binding process have been captured on one of the devices (`device 1`), while the opportunistic connection between the devices has been emulated by isolating the other one (`device 2`) in a Faraday cage: the negotiation among them begins when the cage is removed.

The analysis of packet captures over 16 samples test shows that, on average, the time needed by the two devices to be ready to exchange bundles is of $0.8s$, with an uncertainty of $0.4s$. This time lapse covers both the establishment of a communication link and the binding at DTN layer.

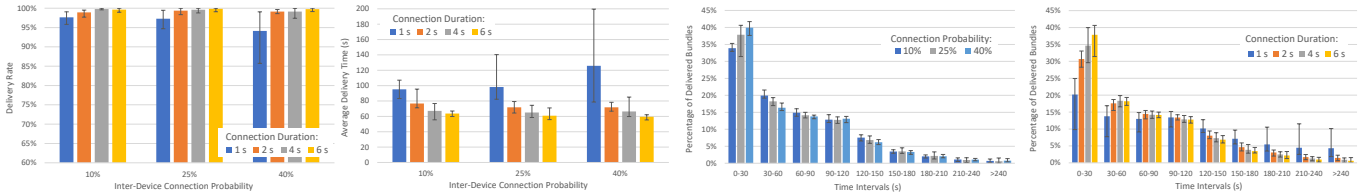
We now analyze the captures of a single reference test. Both devices continuously send beacon frames to advertise their IBSS network. After the first beacon frame from `device 2` (namely, the one previously isolated) is received by `device 1`, they perform the IBSS merge process to establish a physical link, that allows them to receive the IPND beacons used to advertise themselves as DTN nodes. Note that, in our setup, IPND beacons are sent every one second, thus, in the worst case, devices have to wait one second to recognize the presence of a DTN neighbor after the IBSS connection has been established. In our measurement, the first IPND beacon coming from `device 2` has been received $0.27s$ after the first IBSS beacon frame. Thus the IBSS merge algorithm took even less time. in that particular case, we captured the IPND beacon sent from `Device 1` after additional $0.61s$, for a total time of $0.88s$.

B. System simulation

To evaluate performance on a real use case, i.e., when a fleet of end devices exchange generated data through opportunistic connections, we created a simulation environment (released at [2]) by virtualizing the Sensing Devices with Docker containers. Communication occurs through a virtual switch whose connection are dynamically reconfigured with new OpenFlow rules injected by our simulator, thus mimicking vehicle movements over time and their opportunistic connections. The communication occurs over Open vSwitch v2.6.0, while containers are deployed on Docker v17 running on a VM with 16 CPU cores, 12GB of RAM, Linux kernel 4.4.0-96 (the host machine features two octa-core Intel Xeon E5-2660 @ 2.2 GHz CPUs).

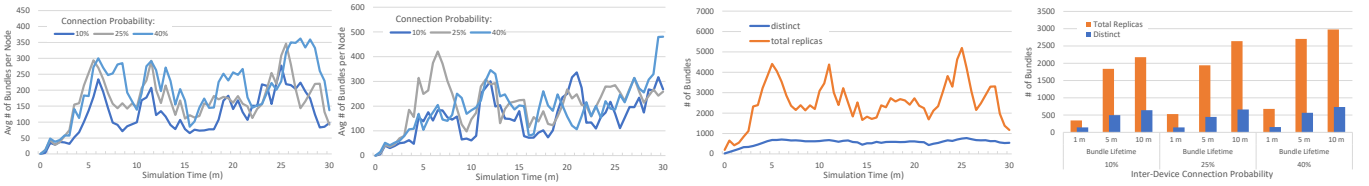
Simulation Parameters. Each simulation lasts 30 minutes, with 15 devices randomly connected to each other. The duration of each opportunistic connection have been varied among 1, 2, 4 and 6 seconds, while the probability of inter-device connections among 10%, 25% and 40%. The purpose of varying these parameters was to identify the minimum requirements in terms of inter-device connections to make our system work properly. We also varied the lifetime of bundles in the DTN among 1, 5 and 10 minutes. In each simulation, each device generates new measurement data every 5 seconds, while its probability to be connected with the fog node is 10%. For each configuration we ran 10 simulation, collecting mean, max and min values.

Delivery Rate and Time. Figure 6a shows the percentage of bundles that, at the end of each simulation, have successfully been delivered to the destination. The bundle lifetime has been set to 5 minutes. Results show that, when the inter-device connections last only 1 second, an high probability of meeting other peers leads to, counter intuitively, worse performance. This is because, since devices do not have the time to exchange large data, more connections just increase the number of duplicated bundles in the network. In all the other cases, almost all bundles (more than 98%) have been successfully delivered. Increasing the duration or the probability of inter-devices connections does not visibly improve performance.



(a) (b) (c) (d)

Fig. 6: (ab) Bundle delivery rate (a) and average delivery time (b) varying the probability of inter-device connections; values are shown comparing different connection time intervals (1, 2, 4 and 6 seconds). (cd) Distribution over time of delivered bundles, for different probability of inter-device connections (c) and different connection time intervals (d). All values refer only to bundles already expired at the end of simulations.



(a) (b) (c) (d)

Fig. 7: (ab) Average number of bundle stored on each device during the simulation for different probability of inter-device connections, with a bundle lifetime of 5 minutes (a) and 10 minutes (b). (c) Storage overhead over simulation time compared to distinct bundles. (d) Average storage overhead varying the probability of inter-device connections; values are shown comparing different bundle lifetimes (1, 5 and 10 minutes).

Figure 6b shows the average delivery time of all bundles. In all configurations, an higher connection duration gives a slightly better average delivery time. Values are noticeably worse with connections of 1 second, and, also in this case, degrade increasing the connection probability.

Delivery Distribution. Figures 6c and 6d show the distribution of bundles successfully delivered to the destination over time, respectively for different connection probabilities (with a connection duration set to 6s) and for different connection duration (with a connection probability set to 25%). In particular, Figure 6c shows that more than one third of bundles are delivered in less than 30s, while ≈95% reached the destination in less than 3 minutes. In this case an higher connection probability noticeably increases the percentage of bundles delivered in the first 30s, while leaving the long term result unchanged. Figure 6d shows that shorter connections give a slightly smaller percentage of bundles delivered in the first 30s, but a similar overall behavior (≈95% of bundles are delivered in less than 3 minutes), except for the case with connections of just 1 second, where the delivery time is visibly more distributed.

In general, results depicted in all graphs of Figure 6 suggest that our approach is suitable in scenarios where inter-devices opportunistic connections during at least 2 seconds can be established with a probability of 10% (or higher) over time, and applications tolerate *(i)* a latency of, at most, 3 minutes to receive the 95% of total data and *(ii)* a data loss inferior to 2%, which is perfectly reasonable for a telemetry system.

Storage Requirements. Since we used an *epidemic* routing algorithm (namely, a device sends a copy of every carried bundle each time an opportunistic connection is established), we performed some measurements to evaluate the storage capacity required on each end device.

Figure 7a shows the average number of bundles stored on each node during the simulation, for different connection probabilities and the bundle lifetime set to 5 minutes. If end devices establish connections with a probability of 10%, the number of bundles per device stabilizes between ≈100 and ≈250 after the initial transient. As expected, higher connection probabilities lead to a larger average amount of carried bundles. However, in all cases the amount of bundles does not increase indefinitely over time (when fully operational, it starts to ranges between two values); this behavior means that, on average, after a certain time the network is able to deliver bundles at the same rate they are generated, thus keeping the storage occupation bounded.

Figure 7b evaluates, instead, the scenario when the bundle lifetime is 10 minutes. Even if, as we know from Figure 6c, more than the 95% of bundles are delivered in less than 3 minutes, increasing the bundle lifetime from 5 to 10 minutes significantly increases the average number of bundles stored on each device (e.g., this time the case with a connection probability of 10%, ranges between ≈100 and ≈300); indeed, even if a copy of a distinct bundle is delivered to the destination, other devices will continue to exchange, and thus duplicate, that bundle until its expiration time. Moreover, in this case the system needs more time to stabilize within a given range.

In general, since the size of each bundle exchanged in our use case is ≈260 bytes, results point out that the storage requirement on end devices is very low (less than 150 KB).

Storage Overhead. We compared the total number of bundles in the network over time with the number of distinct ones (Figure 7c); results refer to a configuration with lifetime of 5 minutes and connection probability of 25%. The graph shows that, while the number of distinct bundles converges to an (almost) constant value (≈700), the actual number of replicas carried on the DTN ranges between ≈2000 and ≈4500 (the number of bundles on the network ranges between 3 and 7

times higher than the number of distinct ones).

Figure 7d compares the average storage overhead for different inter-device connection probabilities and for different bundle lifetimes (1, 5 and 10 minutes). The graph shows that the ratio between total replicas and distinct bundles increases with the connection probability, for any bundle lifetime. On the other hand, we notice a significant difference on the number of bundles (both distinct and replicas) when lifetime is increased from 1 to 5 minutes, while the same does not happen bringing the lifetime to 10 minutes. Indeed, since on average only $\approx 55\%$ of bundles are delivered in less than 1 minute after their generation (as seen in Figure 6c), such a short lifetime makes devices to discard a significant amount of still valid (i.e., not delivered) bundles. This is not the case with a lifetime of 5 and 10 minutes, because almost all packets ($\approx 95\%$) are delivered in less than 3 minutes, as shown in Figure 6c. Therefore, a lifetime of 5 minutes is reasonable in real setups.

Latency Overhead. Finally, we measured the latency overhead introduced by our DTN-based communication infrastructure, through some comparisons with the plain TCP/IP routing in case of a static and fully connected network topology, namely, it is possible to establish a stable (possibly multi-hop) connection between source and destination. Measured latency values are shown in Table I.

TABLE I: End-to-end average latency introduced by our infrastructure, compared to a plain TCP/IP connection.

	TCP/IP	DTN stack
single-hop	0.551 ms	7.810 ms
multi-hop	3.556 ms	52.675 ms

The table shows latency both *(i)* in a single hop communication, namely there are no intermediary devices between source and destination, and *(ii)* in a communication with seven intermediate devices. In general, the DTN bundle protocol introduces a significant latency overhead (additional 7 ms on the first hop). Even if this may constitute a limitation on hard real-time applications, it certainly does not affect delay-permissive applications such as a telemetry system, or firmware updates propagation, which instead benefit of the disruption-tolerant connectivity provided by our approach and are enabled to operate even over challenged environments.

VI. Conclusion

This paper presents a possible approach to Fog Computing in challenged environments, with focus on a real Industry IoT use case. In particular, this work proposes an architecture that enables a fleet of machineries to send data and receive updates to/from fog nodes over the MQTT protocol, largely adopted by widespread IIoT applications. Moreover, the proposed approach aims at keeping applications agnostic toward the type of network connectivity, thus potentially enabling existing applications to operate on an unconventional scenario such as challenged networks. To enable data propagation over such networks, our communication architecture makes use of the *store-carry-and-forward* paradigm, typical of Delay/Disruption Tolerant Networks (DTN). Furthermore, an open-source prototype of the proposed architecture have been created, which has allowed us to present also relevant implementation and configuration details.

Finally, performance of our communication architecture has been evaluated through accurate measurements according to several dimensions; results confirm the advantages and applicability of this paradigm in IIoT environments.

Acknowledgment

The authors would like to thank Nebbiolo Technologies for their support in making this work happening, Luigi Maio and Carlo Pettinato who contributed to the prototype presented in this paper.

References

[1] "Information technology – Message Queuing Telemetry Transport (MQTT) v3.1.1," International Organization for Standardization, Standard, Jun. 2016.
[2] Fog over dtn repository. https://github.com/netgroup-polito/for-over-dtn.
[3] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*. ACM, 2012, pp. 13–16.
[4] L. Atzori, A. Iera, and G. Morabito, "The internet of things: A survey," *Computer networks*, vol. 54, no. 15, pp. 2787–2805, 2010.
[5] D. Miorandi, S. Sicari, F. De Pellegrini, and I. Chlamtac, "Internet of things: Vision, applications and research challenges," *Ad hoc networks*, vol. 10, no. 7, pp. 1497–1516, 2012.
[6] L. Da Xu, W. He, and S. Li, "Internet of things in industries: A survey," *IEEE Transactions on industrial informatics*, vol. 10, no. 4, pp. 2233–2243, 2014.
[7] H. Sundmaeker, P. Guillemin, P. Friess, and S. Woelfflé, "Vision and challenges for realising the internet of things," *Cluster of European Research Projects on the Internet of Things, European Commision*, vol. 3, no. 3, pp. 34–36, 2010.
[8] D. Bandyopadhyay and J. Sen, "Internet of things: Applications and challenges in technology and standardization," *Wireless Personal Communications*, vol. 58, no. 1, pp. 49–69, 2011.
[9] V. Gazis, A. Leonardi, K. Mathioudakis, K. Sasloglou, P. Kikiras, and R. Sudhaakar, "Components of fog computing in an industrial internet of things context," in *Sensing, Communication, and Networking-Workshops (SECON Workshops), 2015 12th Annual IEEE International Conference on*. IEEE, 2015, pp. 1–6.
[10] W. Steiner and S. Poledna, "Fog computing as enabler for the industrial internet of things," *e & i Elektrotechnik und Informationstechnik*, vol. 133, no. 7, pp. 310–314, 2016.
[11] M. Auzias, Y. Mahéo, and F. Raimbault, "Coap over bp for a delay-tolerant internet of things," in *Future Internet of Things and Cloud (FiCloud), 2015 3rd International Conference on*. IEEE, 2015, pp. 118–123.
[12] J. E. Luzuriaga, M. Zennaro, J. C. Cano, C. Calafate, and P. Manzoni, "A disruption tolerant architecture based on mqtt for iot applications," in *Consumer Communications & Networking Conference (CCNC), 2017 14th IEEE Annual*. IEEE, 2017, pp. 71–76.
[13] Z. Zhang, "Routing in intermittently connected mobile ad hoc networks and delay tolerant networks: overview and challenges," *IEEE Communications Surveys & Tutorials*, vol. 8, no. 1, pp. 24–37, 2006.
[14] Nebbiolo Technologies. Bridging the intelligence gap between the cloud and iot and end-points. [Online]. Available: https://www.nebbiolo.tech
[15] Eclipse, "The Paho Project," http://www.eclipse.org/paho/, 2024–2018.
[16] rtl8188eu fork repository. https://github.com/MadLuigi/rtl8188eu.
[17] J. Morgenroth. IBR-DTN. [Online]. Available: https://github.com/ibrdtn
[18] Nebbiolo Technologies. (2017) fogNode overview. [Online]. Available: https://www.nebbiolo.tech/wp-content/uploads/fogNode-OVERVIEW-rev3.pdf
[19] ——. (2017) fogOS overview. [Online]. Available: https://www.nebbiolo.tech/wp-content/uploads/fogOS-OVERVIEW-rev3.pdf
[20] A. Lahmadi and F. Beck, "Powering monitoring analytics with elk stack," in *9th International Conference on Autonomous Infrastructure, Management and Security (AIMS 2015)*, 2015.