Monitoring KPIs in Synchronized FTDMA Multi-hop Wireless Networks

Guillaume Gaillard, Dominique Barthel, Fabrice Theoleyre, Fabrice Valois

Monitoring KPIs in Synchronized FTDMA Multi-hop Wireless Networks

Guillaume Gaillard[3], Dominique Barthel[1], Fabrice Theoleyre[2], and Fabrice Valois[3]

[1]Orange Labs R&D, F-38240, Meylan, France. dominique.barthel@orange.com
[2]ICube, Université de Strasbourg / CNRS, F-67412 Illkirch, France. theoleyre@unistra.fr
[3]Univ Lyon, INSA Lyon, Inria, CITI, F-69621 Villeurbanne, France.
{fabrice.valois,guillaume.gaillard1}@insa-lyon.fr

February 26, 2016

Abstract

Smart Cities rely on smart devices connected to the Internet of Things (IoT). The wireless multi-hop networks are a key building block to enable the IoT. We focus on a mutualized deployment, where a network operator offers IoT connectivity to multiple urban clients, with their specific Quality of Service (QoS) requirements. Hence, such a network must guarantee some minimum level of end-to-end delivery ratio and delay, and isolate each traffic from the others. In this paper, we aim at providing the tools to individually monitor and verify the requirements of each client. We propose to use the IETF 6TiSCH stack (IPv6 over the Time Slotted Channel Hopping mode of IEEE 802.15.4e), because it is a promising basis for meeting the latency and packet delivery ratio constraints, and because it intrinsically provides flow isolation. We propose a mechanism that collects the monitoring blocks for each client / application, by piggybacking Information Elements (IEs) onto the data packets. We show that, by using piggybacking, we save up to 45% on the monitoring overhead compared to a traditional approach.

Keywords: Wireless Sensor Network, Network Monitoring, Key Performance Indicators, Piggybacking, FTDMA

1 Introduction

Smart cities should offer an extensive connectivity for a large collection of smart devices. However, the wireless environnement becomes unstable in dense topologies where several companies use connected devices [11]. New Internet of Things (IoT) network operators (ITNOs) emerge and offer IoT connectivity to several independent clients (e.g. a gas company and a smart parking company). The ITNO contracts, with each client, a specific agreement that guarantees minimum requirements (delay, reliability) for their applications [5]. Then, the ITNO configures its network to respect these commitments, allocating the resource in its routers, while maintaining flow isolation for privacy and management reasons.

For instance, a street lighting company contracts resource with the ITNO (Fig. 1). We designate by ingress router the first equipment - owned by the operator - that relays the packets from the leaf nodes (i.e. the devices of the clients, e.g. light sensors). The data are forwarded through the operated multi-hop wireless network, and transmitted to the client Information System via a gateway. The client may require that 95% of its packets are delivered, within 15 minutes.

The ITNO may deploy a mutualized infrastructure,

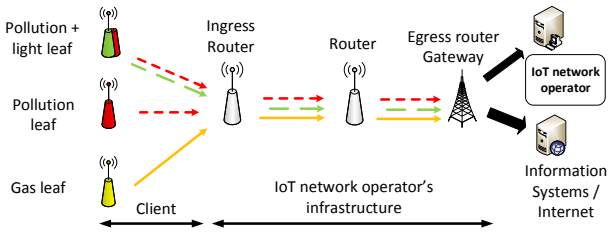


Figure 1: The nodes of the operated urban IoT network.

using the standards proposed by the IETF 6TiSCH Working Group (IPv6 over the Time Slotted Channel Hopping (TSCH) mode of IEEE 802.15.4e). Indeed, 6TiSCH proposes a protocol stack dedicated to industrial wireless sensor networks. It provides flow isolation and reliability on the radio channel by taking advantage of a Frequency-Time Division Multiple Access (FTDMA) technology [11].

The clients generally require guarantees on the following Key Performance Indicators (KPIs) [5]:

1. the end-to-end delivery ratio: the ratio between packets entering the IoT at the ingress routers and packets received at the gateway, for each leaf and application.

2. the end-to-end delay of the data packets: the transit time of each packet from the ingress routers to the gateway.

In this work, we propose mechanisms to monitor the end-to-end performance of the applications of the clients. These mechanisms enable the ITNO to prove at any time that the requirements of the clients are fulfilled. The challenge consists in efficiently measuring the performance of a large number of applications. Indeed, each client has its own Quality of Service (QoS) criteria, and requires that each leaf be monitored independently. For instance, a lighting company requires that 95% of the packets are delivered for *each* leaf. Our solution suits a multi-service multi-client operated architecture.

The contributions of this paper are threefold:

1. we present a standardized way to store and retrieve monitoring information with Constrained Application Protocol (CoAP) [11];

2. we propose a piggybacking mechanism adapted to 6TiSCH networks to measure the end-to-end delivery ratio in an operated network; we explain how we may exploit Absolute Slot Number (ASN) to measure the end-to-end delay;

3. we quantify the cost of our proposal in terms of communication overhead.

The paper is organized as follows: we detail the related work (§2) and describe our model (§3). Then, we detail our contributions in sections 4 and 5; we finally evaluate the gain of our solutions (§6) and conclude (§7).

2 Scientific and technical background

2.1 Monitoring wireless multi-hop networks

In wireless networks, bandwidth is more expensive than on the wire: packets cannot be dedicated to the monitoring plane without significant impact on the user traffic. Therefore the active request-response model of Simple Network Management Protocol (SNMP) does not suit.

In Wireless Sensor Networks (WSNs), the radio channel is scarce, unreliable [2] and shared by numerous sensors. The devices have hard constraints in terms of energy, memory, and CPU resources. Thus, monitoring strategies must save energy and capacity. Unfortunately, the channel access is costly in wireless networks, and energy costs grow non-linearly with the packet size [3]. Thus, we have to reduce both the number of packets and their size.

Network monitoring in IoT may have several objectives. The operator needs to detect the node failures, and the depletions of energy [14]. To make a diagnosis, the ITNO may also need to know which path was followed by each packet [6]. In the same way, unreliable links should be detected to be discarded from the routing topologies [10].

Data aggregation mechanisms [14] enable to reduce the size of the monitoring payload, but at the cost

of accuracy: information is averaged over a set of sensors, or over a long period. A tradeoff consists in choosing the aggregation that minimizes monitoring costs while still detecting faults [8]. In this work, we cannot average monitoring metrics: all of them are required to monitor the Key Performance Indicators (KPIs) and *prove* that the latter are fulfilled.

Two main modes exist to collect monitoring information [12]: an active mode (where some in-band resources are dedicated to the monitoring) and a passive mode (where monitoring information is only inferred from pre-existing data and control packets [9]). In this work, we quantitatively compare the two approaches.

2.2 Measuring the end-to-end delivery ratio

The end-to-end delivery ratio is a major KPI for an application. Zhao *et al.* propose to infer it at the gateway by comparing the sequence numbers of the received packets with the generated ones [14]. However, we cannot adopt this approach to assess the KPI: the packet delivery ratio can only be computed when a packet is finally received. Between two receptions, no information can be inferred, which is problematic, particularly when the traffic is not perfectly periodical. This technique is inaccurate if the ITNO does not know the *exact* generation pattern of each flow.

Lahmadi *et al.* [8], implement a poller/pollee scheme to reduce the number of monitoring packets: a *poller* directly asks a set of *pollees*. The poller aggregates the information to send it back to the gateway. The authors use the IETF protocols (IPv6 over Low power Wireless Personal Area Networks (6LoWPAN), IPv6 Routing Protocol for Low-Power and Lossy Networks (RPL) [11]) to relay the monitoring information. Because the delivery information is aggregated in the pollers, the ITNO cannot compute the end-to-end packet delivery ratio for each flow. Thus, no aggregation is possible.

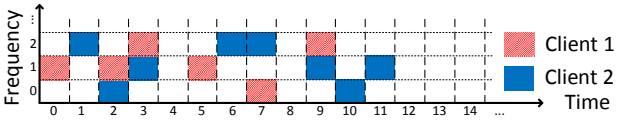


Figure 2: The time-frequency cells of the FTDMA Schedule.

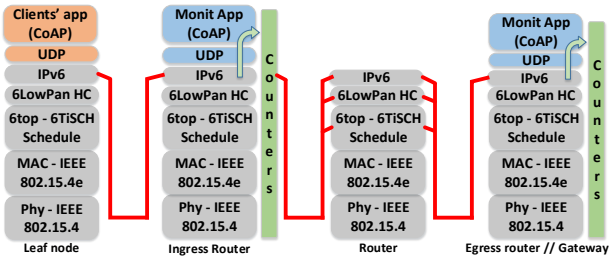


Figure 3: The use of the IETF 6TiSCH protocol stack on the operated network nodes.

2.3 The 6TiSCH stack

The 6TiSCH Working Group [11] aims at defining a family of protocols to bridge the gap between IEEE802.15.4e TSCH and the IP world. The FTDMA approach enables to control the resource allocation finely, guaranteeing traffic isolation among different applications / clients.

The Time Slotted Channel Hopping (TSCH) mode of IEEE 802.15.4e schedules the communications of a sensor network over time-frequency cells (Fig. 2). 6TiSCH allows nodes to follow a periodic communication schedule in a multi-hop way. At each timeslot, the channel hopping changes the relationship between the logical channel numbers and the physical frequencies. This gives more robustness by spreading the risk of interference.

6TiSCH enables the ITNO to process each end-to-end flow by assigning it a dedicated set of cells. Thus, 6TiSCH enables flow isolation and differentiated QoS.

2.4 Monitoring with 6TiSCH

6TiSCH enables the ITNO to run CoAP [11] on its routers (Fig. 3), as a monitoring application. This reduces the communication overhead because CoAP, among the web transfer protocols, is more compact

than HTTP. It also gains efficiency in terms of code re-use: CoAP is commonly used to carry applications and existing device management formats such as COMI [13]. The CoAP Observe [7] strategy enables routers to be CoAP-servers-only and directly send spontaneous updates when changes occur.

Today, as far as we know, no solution that addresses the needs of the ITNOs has been proposed that provides end-to-end network monitoring mechanisms for wireless sensor networks, despite the recent effort of the 4G/5G community to support M2M traffic. We propose in this work to address this challenge.

3 Use-case and model

In this work, we propose to measure in a 6TiSCH network two major KPIs: the end-to-end delivery ratio and the end-to-end delay. In IEEE 802.15.4e TSCH, the medium is divided into time-frequency cells (Fig. 2) that permit the transmission of one frame. The frames are classically limited to 127 bytes. Each ingress router runs a monitoring application, using CoAP (Fig. 3). We define specific Information Elements (IEs) to piggyback monitoring information onto transiting packets (data, RPL control unicast packets, etc.).

IEs represent a normalized way to implement an opportunistic piggybacking, by exploiting the remaining space in the frames without modifying the data structure. On the contrary, using static IP header extensions generates overhead and can not profit from traffic adaptation.

We consider that each node executes one or several applications (e.g. light control, pollution monitoring). Each node has to attach to the network, selecting one or several ingress router(s). We only consider the in-transit QoS: the performance is not guaranteed on the first hop since the leaf node is not owned by the operator, but by the client itself.

We consider that an application generates messages of uniform size, to simplify the performance evaluation and the interpretation. Practically, a client may negotiate a maximum size for its messages. Our monitoring mechanism would adapt to the size of each forwarded data packet.

Each message may be fragmented into one or several IP packets that transit through the IoT network (Fig. 3). An application has to specify a UDP port, imposed by the ITNO. Since we use the standard IP stack, the operator is then able to identify an application by inspecting the tuples $< leaf\ IPv6_{@},\ UDP_{port} >$ in the packets.

The KPIs are measured over a fixed period, that depends on the application. For instance, a delivery ratio for a gas meter may be measured over 1 day, while a lighting system requires a minimum reliability over a shorter period: 1 hour.

Here we only focus on the uplink direction: the sensors report their measures to a gateway, connected to the Internet. The KPIs for the downlink direction, uncommon in the current IoT, are not considered in this work.

4 Measuring KPIs for a client in a wireless multi-hop network

We detail here how we measure the end-to-end delay and delivery ratio, with reduced bandwidth and energy costs.

4.1 Measuring the end-to-end delay

In 6TiSCH, FTDMA *de facto* brings time synchronization to the nodes with a typical sub millisecond accuracy [11]. We timestamp the events using the ASN: each slot corresponds to an increment of the 5 bytes ASN value. Consequently, the resolution is in the order of the timeslot duration (typically 10ms). The ASN rolls over after 350 years. We consider this largely sufficient for the client's requirements. Moreover, IEEE802.15.4e enables to piggyback information onto data frames, as Information Elements (IEs). IEs are containers structured as Type, Length, Value fields.

We propose the following simple solution:

1. when the ingress router receives a packet, it inserts an Information Element containing the current timestamp. This IE uses 10 bytes (5 for the

IE type and length, and 5 for the timestamp itself);

2. when the gateway receives the packet, it extracts the timestamp, and computes the end-to-end delay. This metric is then stored in the Information System.

4.2 Measuring the end-to-end delivery ratio

This section details our proposal for measuring the delivery ratio, i.e. the ratio between the number of packets generated by a leaf and the number of packets received by the gateway. The ITNO computes the ratio at the ingress and at the egress points, for a given time period.

We can not use the classical sequence numbering approach, because the data traffic pattern is unknown a priori. We would not be able to detect losses until a subsequent packet reaches the gateway (in an unbounded time).

We count the number of packets entering the ingress routers. More precisely, an ingress router increments a counter associated with a given tuple $< IPv6_{@},\ UDP_{port} >$.

For each application running on one or several leaf nodes, the ingress router has to periodically report to the gateway the values of all the counters associated with this application. Since each application may have a different reporting period, all of them are handled independently. Thus, the ingress router creates a vector of all the counters for a given application identified by a UDP port (Fig. 4). The ingress router finally constructs a piece of monitoring information (denominated *monitoring block*) which contains:

- the UDP port UDP_A;
- the ASN corresponding to the time when the monitoring block was created;
- the vector of tuples $< leaf\ IPv6_{@},\ counter >$.

For each application, the ingress router stores the corresponding monitoring block in a buffer that is dedicated to the monitoring.

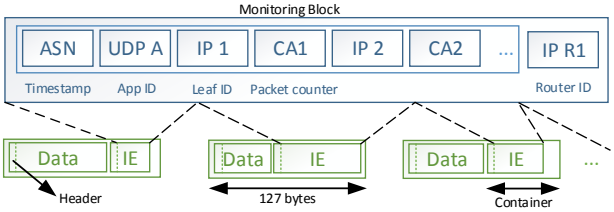


Figure 4: The periodic vector of counters' values for a given application A, on a given ingress router.

The leaf nodes may communicate with different ingress routers during the same monitoring period. Thus, each IP address has to be specified in the blocks in order to let the gateway re-assemble the information.

The size of the vector depends on the number of leaf nodes attached to an ingress router:

$$Sizeof(Vector(App)) = Sizeof(ASN) + Sizeof(UDP_{port}) + nb_{dev}(App) * (Sizeof(IP_{@}) + Sizeof(counter)) \quad (1)$$

with $Sizeof(V)$ denoting the size of the variable V, and nb_{dev} the number of devices attached to the ingress router for this application. The remaining variables are given in table 1.

5 Collecting the monitoring information

5.1 An On-demand mechanism

We detail here how the monitoring information, distributed across all the ingress routers, can be accessed on demand: the operator explicitly requests the information from each ingress router. It sends a CoAP GET message to obtain the monitoring block, for a given UDP application, in a given time-period.

In the same way, the ingress routers also implement an autonomous behavior, using CoAP Observe. With a first GET the ITNO configures each ingress router with a given period, so that the ingress router autonomously creates a periodic CoAP message containing a collection of monitoring blocks.

In case some monitoring information is missing, the gateway sends a GET to the corresponding ingress router.

While the on-demand approach permits to accurately diagnose a problem when it is detected, it is not energy efficient because it requires to dedicate some timeslots to the monitoring information.

5.2 An energy-efficient periodic mechanism

We also propose a periodical mechanism which piggybacks monitoring blocks on the data packets. Many data packets contain less than 127 bytes: we fill them to also include monitoring information. Since a timeslot has a fixed duration, we improve the bandwidth occupancy.

At the end of the monitoring period of application UDP_A, each ingress router generates one or several Information Elements which contain the corresponding monitoring blocks (Fig. 4). Then, the IEs are *piggybacked* on the data packets forwarded by the ingress router toward the gateway.

When an applicative message is fragmented, the ingress router piggybacks a monitoring IE on the last fragment if enough space is available (> 5 *bytes*). We denote as *container* the available piggybacking space on each last fragment.

If the monitoring block is small enough (i.e. only a few leaves are attached for an application on this router), only one IE is necessary. Else, the monitoring block is itself fragmented in several IEs (Fig. 4): the gateway is then in charge of re-assembling these monitoring IEs to reconstruct the block.

The quantity of data packets may be insufficient to transmit all the monitoring information. In this case, we propose that the ingress router creates a *monitoring-only* packet if it still has monitoring information to transmit when a timer elapsed. The gateway is able to reconstruct the whole monitoring information with the already received IEs.

5.3 Monitoring isolation

The router has to select the *containers* to piggyback its monitoring blocks. If an ingress router selects a small container, it may waste resource: another larger container may become available later, which would not require to fragment the monitoring information. Inversely, if an ingress router waits too long without finding enough space, a new *monitoring-only* packet is generated, which wastes bandwidth and energy.

So we evaluate two opposite piggybacking strategies:

1. **mixed application** piggybacking: no selection is done. The routers select the first container available in *any* packet forwarded by the ingress router;

2. **application isolation** piggybacking: the monitoring information concerning the application UDP_A is only piggybacked on the data packets generated by the application UDP_A. We preserve flow isolation, i.e. monitoring of the application UDP_A does not impact the performance of the application UDP_B.

The last case is clearly more constraining. In particular, applications which generate packets with a size close to the MTU do not have enough space to piggyback all the monitoring information.

6 Evaluating the monitoring overhead

In this section, we quantify the impact of piggybacking the monitoring information on the data packets compared to using *monitoring-only* packets. The results were obtained using a Monte-Carlo simulation of the behavior of an ingress router.

We first compute the total amount of data and monitoring traffic, in bytes, and in number of packets and blocks. We deduce the number of containers of each application, given the fragmentation of the data packets. Then, we set the data message arrivals according to each scenario and strategy. For instance, the messages are ordered by decreasing size for the lower bound case in mixed application.

6.1 Parameters and assumptions

In order to study heterogeneous multi-application scenarios in the context of smart cities, we adopt the

Table 1: The monitoring variables on an ingress router.

Parameter	Value	$\Gamma(App)$
Water Metering App.	100 B, 2/day	4 leaves
Smart Parking App.	90 B, 24/day	8 leaves
Pollution Monit. App.	10 B, 24/day	2 leaves
Fragmentation header	5 B	
IE header	5 B	
6TiSCH stack headers	62 B	
Compressed UDP port	1 B	
Compressed IP address	2 B	
ASN	5 B	
Counter value	1 B	

average configuration described in the ETSI technical report [1]. More precisely, our study copes with 3 typical applications: water telemetering, smart parking, and pollution monitoring. We arbitrarily choose their traffic patterns and parameters, as summarized in Table 1. Moreover, we suppose a uniform deployment over an homogeneous and large city. For each application, only a part of the leaf sensors are able to support the traffic (see $\Gamma(App)$ in Table 1).

We assume that the maximum payload of a data packet is 65 bytes (added to 62 bytes of headers, this reaches the maximum frame size of 127 bytes). Beyond 65 bytes, the data messages are fragmented. We assume that the network implements the 6LoWPAN fragmentation and header compression [11]. This yields 5-byte long fragmentation headers and compressed UDP and IP headers (Tab. 1). Each fragment therefore has 60 available bytes.

The monitoring frequency is the main variable of the evaluation. A relevant monitoring solution should not generate more information than the application itself. Hence, we set the monitoring frequency to a percentage of the data message frequency. We vary the monitoring frequency between 0% and 120% to observe its impact on the overhead.

For instance, 8 leaf nodes contribute to the parking application and generate one message each hour. A monitoring frequency of 50% (of the data frequency) means a monitoring block is generated every 15 minutes:

$$50\% * 8 * 1 \; message/hour = 4 \; message/hour \quad (2)$$

We compare the following two opposite procedures:

1. the **shared container** procedure: when a packet has to be forwarded, the ingress router fills all the remaining space with all the monitoring information it has buffered so far. In particular, a packet may carry multiple Information Elements. Thus, all the packets have a payload close to the maximum of 127 bytes, including the headers;

2. the **exclusive container** procedure: a packet contains at most **one** monitoring block. If the IE does not fill all the container, some bandwidth is wasted. However, this solution is very simple to implement.

The choice of the procedure is orthogonal to the choice of the piggybacking strategy (mixed application or application isolation, cf. section 5.3). Moreover, the ITNO also chooses between the two procedures for the *monitoring-only* packets.

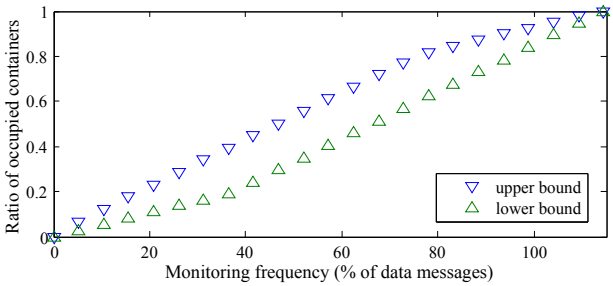
We calculate the number of occupied containers and the corresponding monitoring overhead (including the headers and the potential fragmentation of the blocks) according to each procedure (e.g. in the shared container procedure, the blocks are fragmented to fill the containers).

6.2 Occupancy ratio of the containers

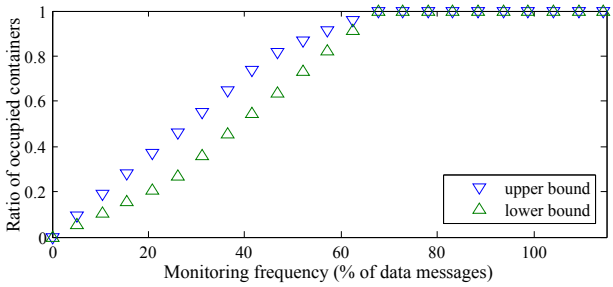
Figure 5 illustrates the ratio of containers which are actually filled by piggybacked monitoring information. This metric measures the saturation of the monitoring mechanism.

However, the monitoring efficiency also depends on the distribution of the size of the different containers, and on the order in which they appear at the ingress router. Thus, we construct the following upper and lower bounds:

1. best case (lower bound): we assume that the ingress router always forwards the largest packets first (decreasing order of the packet size). This constitutes an optimistic scenario: the fragmentation is minimized since the largest containers are used first;



(a) Shared Containers (several monitoring blocks per container)



(b) Exclusive Containers (one monitoring block per container)

Figure 5: Ratio of Containers which piggyback monitoring information (Saturation metric).

2. worst case (upper bound): we assume that it always forwards the smallest packets first. This represents a pessimistic scenario: it maximizes the overhead created by the fragmentation;

Actually, the monitoring solution will perform between these bounds. We clearly see that the bounds are sufficiently tight (Fig. 5) to accurately estimate the performance we would obtain when the packet sizes are randomly ordered.

When we restrict data packets to only piggyback one monitoring IE (exclusive container procedure, Fig. 5b), all the containers are filled when the monitoring frequency is larger than 65% of the data frequency. In this case, *monitoring-only* packets have to be created, which negatively impacts the bandwidth and the energy consumption.

On the contrary, **when we authorize a data packet to carry several monitoring IE in the same container** (shared container procedure), **we significantly reduce the overhead** (Fig. 5a). We do not need to generate new *monitoring-only* packets unless the monitoring frequency exceeds 115%. Thus, although the shared container procedure is more complicated to implement in terms of buffer management, it is very relevant to reduce the overhead.

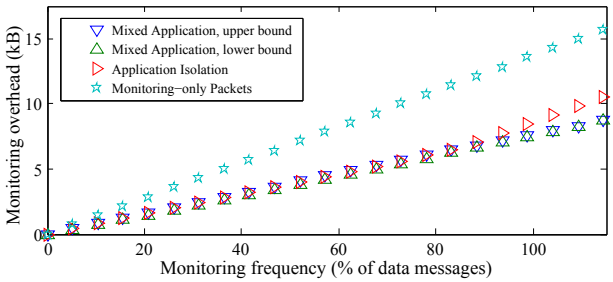
6.3 Overhead of the monitoring mechanisms

We study the overhead generated by the monitoring process – the number of bytes dedicated to the monitoring mechanism (Fig. 6). In particular, we compare the piggybacking solution (the monitoring information uses the space left in data packets) and the dedicated solution (*monitoring-only* packets are created, and use dedicated timeslots). We assume that during one day, an ingress router forwards 50 kB of data traffic on behalf of the three clients (Tab. 1).

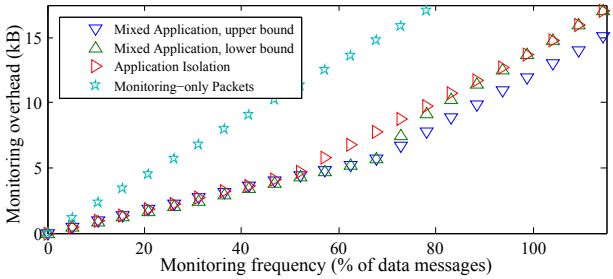
In case we do not use the piggybacking mechanism (stars in Fig. 6), new packets are created when a monitoring block has to be transmitted toward the gateway. This constitutes the worst solution because a new header has to be included in the packet, which mechanically increases the overhead. Besides, we also have to reserve new transmission opportunities (timeslots) for these *monitoring-only* packets. All the other solutions piggyback monitoring information on the data packets, and are much more efficient to reduce both bandwidth and energy consumption. **Piggybacking solutions efficiently reduce the monitoring overhead**.

When we compare exclusive and shared containers (Fig. 6a and 6b), we verify that **authorizing a container to piggyback several different monitoring blocks significantly reduces the overhead**. Indeed, the exclusive container procedure begins to saturate when the monitoring frequency exceeds 60% to 70%: new *monitoring-only* packets have to be generated, increasing the overhead.

We also note that authorizing the monitoring information for one application to use the data traffic of another application (*mixed application* curve) reduces the overhead compared to strictly isolating the traffic from different applications (*application isolation*). However, the gain seems minor. Therefore, we would prefer the application isolation method, be-



(a) Shared Containers (several monitoring blocks per container)



(b) Exclusive Containers (one monitoring block per container)

Figure 6: Overhead of the monitoring solution – number of bytes for the monitoring information.

cause **it enforces strict traffic isolation among different clients / applications without significantly impacting the performance of the system**.

7 Conclusions and perspectives

In this work we adopted the point of view of an IoT network operator (ITNO): we must accurately monitor the network to prove that the requirements of the clients are met. Thus, we presented mechanisms to measure both the end-to-end delay and packet delivery ratio. Because energy is a major constraint in multi-hop wireless networks, we thoroughly evaluated the efficiency of piggybacking techniques. In particular, we compared several strategies, mixing several monitoring information in the same data packet, or on the contrary guaranteeing traffic isolation. We showed that piggybacking mechanisms are efficient to reduce the use of bandwidth and energy, and that a buffering strategy of monitoring information reduces the overall cost.

In future work, we will thoroughly evaluate these mechanisms on the FIT/IoT Lab testbed [4]. In particular, we aim at studying the impact of packet losses and at exploring network coding techniques that may improve the reliability of the monitoring information. We will consider heterogeneous distributions of sensors over a smart city.

References

[1] Spectrum Requirements for Short Range Device, Metropolitan Mesh Machine Networks (M3N) and Smart Metering (SM) applications. *ETSI TC ERM, TR 103 055, v1.1.1*, Sept. 2011.

[2] N. Baccour, A. Koubâa, L. Mottola, M. A. Zúñiga, H. Youssef, C. A. Boano, and M. Alves. Radio Link Quality Estimation in Wireless Sensor Networks: A survey. *ACM Transactions on Sensor Networks*, 8(4):1–33, Sept. 2012.

[3] Z. Cheng, M. Perillo, and W. Heinzelman. General network lifetime and cost models for evaluating sensor network deployment strategies. *IEEE Transactions on Mobile Computing*, 7(4):484–497, April 2008.

[4] E. Fleury, N. Mitton, T. Noel, C. Adjih, V. Loscri, A. M. Vegni, R. Petrolo, V. Loscri, N. Mitton, G. Aloi, et al. Fit iot-lab: The largest iot open experimental testbed. *ERCIM News*, (101):14, 2015.

[5] G. Gaillard, D. Barthel, F. Theoleyre, and F. Valois. Service Level Agreement Architecture for Wireless Sensor Networks: a WSN Operator's Point of View. In *NOMS*. IEEE/IFIP, 2014.

[6] Y. Gao, W. Dong, C. Chen, J. Bu, G. Guan, X. Zhang, and X. Liu. Pathfinder: Robust path reconstruction in large scale sensor networks with lossy links. In *ICNP*, pages 1–10, Oct 2013.

[7] K. Hartke. Observing Resources in CoAP. IETF ID draft-ietf-core-observe-16 (Work In Progress), dec 2014.

[8] A. Lahmadi, A. Boeglin, O. Festor, et al. Efficient distributed monitoring in 6lowpan networks. In *CNSM*, 2013.

[9] Y. Liu, K. Liu, and M. Li. Passive diagnosis for wireless sensor networks. *IEEE/ACM Transactions on Networking*, 18(4):1132–1144, 2010.

[10] H. X. Nguyen and P. Thiran. Using end-to-end data to infer lossy links in sensor networks. In *INFOCOM*, 2006.

[11] M. Palattella, P. Thubert, X. Vilajosana, T. Watteyne, Q. Wang, and T. Engel. 6tisch wireless industrial networks: Determinism meets ipv6. In S. C. Mukhopadhyay, editor, *Internet of Things*, volume 9 of *Smart Sensors, Measurement and Instrumentation*, pages 111–141. Springer International Publishing, 2014.

[12] A. Rodrigues, T. Camilo, J. S. Silva, and F. Boavida. Diagnostic tools for wireless sensor networks: A comparative survey. *Journal of network and systems management*, 21(3):408–452, 2013.

[13] P. van der Stok, B. Greevenbosch, A. Bierman, J. Schoenwaelder, and A. Sehgal. CoAP Management Interface. IETF ID draft-vanderstok-core-comi-06 (Work In Progress), fev 2015.

[14] J. Zhao, R. Govindan, and D. Estrin. Computing aggregates for monitoring wireless sensor networks. In *International Workshop on Sensor Network Protocols and Applications*, pages 139–148. IEEE, 2003.