PREPRINTS

17th International Workshop on Real-Time Networks – RTN 2019

Stuttgart, Germany, July 9, 2019
In conjunction with the 31st ECRTS

https://www.ecrts.org/rtn/

Chairs

Guillermo Rodriguez-Navas
Nokia-Bell Labs

Ramon Serna Oliver
TTTech

Workshop Program

9:00	Welcome and Workshop Opening
9:15 – 10:30	Keynote Speech
	5G and Real-Time-Networks: A practical example Matthias Jablonowski, Nokia
10:30 – 11:00	Coffee Break
11:00 – 12:30	Session #1: Dependability and Real-Time
	DynaMO - Dynamically tuning DSME Networks Harrison Kurunathan, Ricardo Severino, Anis Koubaa and Eduardo Tovar
	A Preliminary Roadmap for Dependability Research in Fog Computing Zeinab Bakhshi and Guillermo Rodriguez-Navas
	Towards a Dynamic Replication of Messages in a Network with Flexible Real-Time Guarantees Alberto Ballesteros, Manuel Barranco, Sergi Arguimbau, Marc Costa and Julián Proenza
12:30 – 14:00	Lunch
14:00 – 15:30	Session #2: Collaborative Session and Posters
	Real-time Network Management for Commercial Aircrafts Thibault Delmas, Luigi Iannone, Bruno Monsuez and Jean-Pierre Garcia
	First Analysis of the AVB's Stream Reservation Protocol in the Context of TSN Daniel Bujosa, Drago Čavka, Ines Alvarez and Julian Proenza
15:30 – 16:00	Coffee Break
16:00 – 17:30	Session #3: Scheduling and Routing
	Routing Heuristics for Load-balanced Transmission in TSN-Based Networks Mubarak Ojewale and Patrick Meumeu Yomsi
	Symphony - Routing Aware Scheduling for DSME Networks Harrison Kurunathan, Ricardo Severino, Anis Koubaa and Eduardo Tovar
	SlotSwapper: A Schedule Randomization protocol for Real-Time WirelessHART Networks Ankita Samaddar, Arvind Easwaran and Rui Tan
17:30	Wrap-up and Workshop Closure

Workshop Chairs

Guillermo Rodriguez-Navas
Nokia-Bell Labs, Israel

Ramon Serna Oliver
TTTech, Austria

Program Committee

Ahlem Mifdaoui, DISC, University of Toulouse, France

Borislav Nikolic, IDA/TU Braunschweig, Germany

Iñaki Val, IK4-IKERLAN, Spain

Jean-Luc Scharbarg, INP/ENSEEIHT, University of Toulouse, France

Julian Proenza, Universitat de les Illes Balears, Spain

Luis Almeida, University of Porto, Portugal

Paul Pop, TU Denmark, Denmark

Rodrigo Coelho, TU Kaiserslautern, Germany

Svetlana Girs, Mälardalen University, Sweden

Ye-Qiong Song, LORIA, France

Zdenek Hanzalek, CTU in Prague, Czech Republic

Keynote

5G and Real-Time Networks: A practical example

Real-Time Networks hold great value for industrial users. This session will discuss a practical example of the maritime industry. Operators of container terminals in seaports are looking into automated operations of their container handling equipment. While first semi-automated systems are deployed in ports worldwide, the industry is now looking into 5G and Real-Time-Networks to enable full automation. This session will explain the use case, will map according industrial requirements to the features of 5G and will speak about related research activities.

MATTHIAS JABLONOWSKI

Global Practice Lead – Ports & Roads, Nokia

Matthias Jablonowski is global practice lead of the Ports and Roadways programs at Nokia. Being intrigued by the opportunities of connected technologies and digital transformation, he works with port authorities and terminal operators on Port 4.0 and terminal automation projects as they embark on their smart ports journey. With road operators he discusses next generation traffic technologies and cooperative intelligent transportation systems enabling safer and less congested roadways. Matthias has been instrumental in the expansion of Nokia into the Transportation industry.

DynaMO - Dynamically tuning DSME Networks

Harrison Kurunathan, Ricardo Severino, Anis Koubaa, Eduardo Tovar
CISTER/ISEP and Prince Sultan University
Porto, portugal and Saudi Arabia
{hhkur,rarss,emt}@isep.ipp.pt,akoubaa@psu.edu.sa

ABSTRACT

Deterministic Synchronous Multichannel Extension (DSME) is a prominent MAC behavior first introduced in IEEE 802.15.4e supporting deterministic guarantees using its multisuperframe structure. DSME also facilitates techniques like multi-channel and Contention Access Period (CAP) reduction to increase the number of available guaranteed timeslots in a network. However, any tuning of these functionalities in dynamic scenarios is not explored in the standard. In this paper, we present a multisuperframe tuning technique called DynaMO which tunes the CAP reduction and Multisuperframe Order in an effective manner to improve flexibility and scalability, while guaranteeing bounded delay. We also provide simulations to prove that DynaMO with its dynamic tuning feature can offer up to 15-30% reduction in terms of latency in a large DSME network.

KEYWORDS

IEEE 802.15.4e, DSME, Multisuperframe tuning

1 INTRODUCTION

IEEE 802.15.4 [2] is one of the legacy protocols that supports low-rate communication with Guaranteed Time Slot (GTS) allocation mechanism that provides guaranteed bandwidth for time-critical data. However, it suffered from limited scalability as the number of GTS provided was restricted to 7. The enhancement of this protocol, the IEEE 802.15.4e [1], [10] rectifies this problem by the provision of multichannel and CAP reduction techniques. DSME is supported by a multisuperframe structure (Fig. 1) which is a stack of several superframes containing a Contention Access Period (CAP) and Contention Free Period (CFP) for communication. This multisuperframe structure is defined by a Multisuperframe Order (MO).

DSME also introduces a new technique called CAP reduction with which the number of GTS resources to accommodate transmissions can be further increased. This is achieved by removing the CAP in a multisuperframe except for the first, hence radically increasing the number of available GTSs. To invoke CAP reduction in the network, the coordinator has to send an Enhanced Beacon (EB) with a CAP reduction primitive.

Traditionally, DSME networks require a careful planning of its several MAC parameters, such as MO and CAP Reduction usage, by an experienced network engineer, to achieve adequate QoS levels. As of now, these values are determined statically at the beginning of the network. In scenarios where traffic or the number of nodes can change, which is increasingly becoming a common place in large-scale IoT networks, static settings inevitably lead to some kind of compromise in terms of delay or throughput that can only be addressed by devising mechanisms that can adapt on-the-fly to new conditions.

The main contribution of this paper are as follows:

- We introduce a dynamic Multisuperframe and CAP reduction tuning technique (DynaMO) that yields better QoS performance in terms of delay.
- We provide a numerical analysis to calculate the overall delay of the network.
- We evaluate DynaMO using the simulation platform "OpenDSME" to validate our analytical model.

In the following section we provide a brief literature survey. In section III we discuss the problem, then in Section IV, we provide the DynaMO algorithm and discuss is functionality. Later in Section V, we provide a numerical analysis for delay. We complement this analysis using simulation in Section VI. We wrap up our work with conclusions and discussions in Section VII.

2 RELATED WORKS

In our previous research [12], we observed reduced delay in DSME network when CAP reduction was utilized. But this analysis was only made for a static network. There have also been several research works like [3] and [5] in which the performance of DSME was analysed. However in these simulative studies, features like the CAP reduction and superframe structure were kept static. We believe this static configuration can be an impediment to the overall Quality of Service of the network.

In classic IEEE 802.15.4, researchers in [6] and [14] have used algorithms to adjust Superframe Order (SO) at the coordinator by considering parameters of end devices such as queue size, queuing delay, energy consumption per bit and data rate. This helped in improving the overall network life time. In one of our earlier works [8], in contrast to the traditional explicit allocation of GTS in IEEE 802.15.4, we used implicit allocation as the number of GTSs is limited. We were able to produce betterment in QoS in terms of bandwidth utilization.

The literature in varying the structure of MAC to improve QoS is not limited to DSME. Mashood Anwar [4] studied the variations in superframe of LLDN an other key MAC behavior of IEEE 802.15.4e and was able to provide an insight on the tuning of superframe to yield better network performance. Several parameters like sensors refresh rate, number of devices accommodated in network, data payload exchanged between the devices and even different levels of security were analyzed in this work.

We believe that dynamic tuning of the multisuperframe parameters such as MO and CAP reduction primitives has a possibility to yield better network performance. Hence we investigated several scenarios of DSME networks and propose a dynamically tunable multisuperframe scheme that yields better performance in terms

Conference'17, July 2017, Washington, DC, USA
2019. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

of delay. In what follows, we present the scenarios of the problem that DynaMO helps to overcome.

3 BACKGROUND TO THE PROBLEM

The DSME network provides deterministic communication using its beacon enabled mode in which the entire time frame is separated into multisuperframes accommodating several superframes as shown in Figure 1. The superframe is defined by *BO*, the *Beacon Order* which is the transmission interval of a beacon in a superframe, *MO* the *Multi superframe Order* that represents the enhanced beacon interval of a multi-superframe and *SO* the *Superframe Order* that represents the beacon interval of a superframe within a Multisuperframe duration. The number of superframes in a multisuperframe can be given by $2^{(MO-SO)}$ and the number of superframes that a multisuperframe should accommodate is set by the PAN coordinator and is conveyed to the nodes via an Enhanced Beacon (EB) at the beginning of each Multisuperframe.

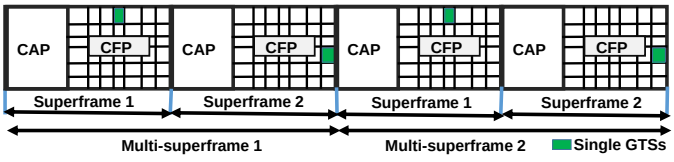


Figure 1: Superframe structure with *BO*=3, *MO*=3, *SO*=2

Under CAP reduction, all the superframes in a multisuperframe can be converted into complete CFPs except for the first. In accordance to the standard, both CAP reduction and MO are determined statically at the start of a multisuperframe by the Personal Area Network Coordinator (PAN-C). The network that is statically defined at the beginning will have limited capabilities to cope with constantly evolving network with joining and leaving of the nodes. Some of the adverse results can be *"an improper bandwidth allocation either due to not enough GTS slots"* or *"wasted bandwidth increasing the contribution to the delay."*

Having a routing layer such as RPL (Routing Protocol for Lossy Networks) over DSME is a fundamental mechanism to solve this problem. In our approach, an updated routing tree of the varying network topology is provided to the PAN-C by the RPL. As the number of nodes changes (via association/disassociation), RPL updates this information and the PAN-C generates a schedule spread into the available GTSs resources. A detailed report on implementing RPL over DSME can be found in [11].

In this contribution, we design an algorithm that is able to set the most adequate value of MO and toggle CAP reduction considering the needed resources. In doing so, we are able to minimize latency. The necessary changes to the values of the MO or the CAP reduction primitive are sent in the beacon payload of an EB at the beginning of every multisuperframe. Hence, with a dynamic evolution of a wireless sensor network with addition/removal of nodes new values for MO and CAP reduction primitives can be dynamically set, eventually improving the overall QoS of the network.

Algorithm 1 DynaMO

Input BO, SO, MO, CAP reduction Primitive
Pairwise transmissions from RPL: $((a_1,a_2),(a_1,a_3).....(a_2,a_1).....(a_i,a_N))$
$N_{Channels}$ and N_{TS} ϵ $(1,7+(NCAP))$

Initialization
repeat
Schedule R= Required number of resources to accommodate the network
Resource test: check $N_{CFP} = R$ in a multisuperframe

Case 1: less resources
  while $N_{CFP} < R$ **do**
CAP Reduction = ON;
    if resource test = true **then**
Print: DynaMO is successful.
    else MO = MO + 1;
    end if
  end while

Case 2: abundant resources
  while $N_{CFP} > R$ **do**
CAP Reduction = OFF;
    if Resource test = true **then**
Print: DynaMO is successful.
    else MO = MO - 1;
    end if
  end while
Loop Repeat: Every multisuperframe duration

4 DYNAMO ALGORITHM

In this section, we introduce an efficient multisuperframe tuning algorithm called DynaMO. The general idea of this algorithm is *adaptively increasing and decreasing the multisuperframe structure based on the evolution of GTS allocation requirements over time.*

Algorithm 1 presents the DynaMO adaptive network algorithm and Table 1 presents the notation used for the description of the algorithm.

Notations	Description
N	total number of nodes
a_i	node a_i where $i\epsilon(1,N)$
$N_{Channels}$	number of channels = 16
T_i	index of the timeslot in the multisuperframe
N_{CFP}	total number of GTSs in the CFP of a multisuperframe
$NCAP$	number of GTS added when CAP reduction is activated

Table 1: Notations for DynaMO

As the network grows/diminishes dynamically, the routing layer will update the topology and forward the respective schedules that

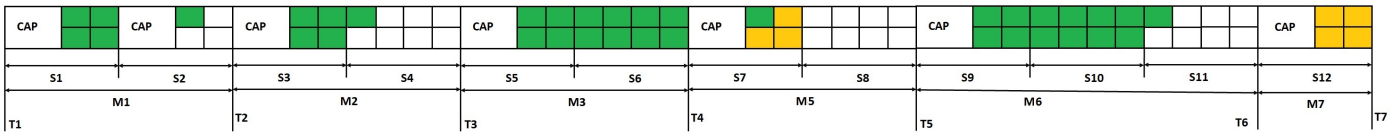


Figure 2: multisuperframes in DSME network

contain the list of pair-wise GTSs transmissions. This is provided as an **input** (Algorithm line 1). Let us consider pairs of neighbor nodes (a_i, a_N) to transmit between each other. This transmission list will be provided as a bitmap to the link layers using the RPL backbone for every beacon interval.

The PAN Coordinator has access to all information needed to establish a multi-channel GTS allocation, including, the number of channels ($N_{Channels}$), the number of the GTSs time slots (N_{TS}) and the total available GTS resources ($N_{CFP} = N_{Channels} * N_{TS}$). The number of time slots can sometimes vary if the CAP reduction primitive is activated. In such a case, the number of time slots will be $7 + N_{CAP}$, where N_{CAP} is the number of time slots added via CAP reduction. The PAN-C initially randomly determines the values of BO, MO, and SO and the CAP reduction primitive. Any change in the network is reported to the PAN-C or the routing parent nodes for every multisuperframe interval. The delay taken to accommodate a new network will depend on the size of the multisuperframe.

In our algorithm, we first determine the number of resources that need to be allocated in the network. This is achieved through a near optimal scheduling algorithm such as simulated annealing [15] or Symphony [11]. In fact, an optimal schedule must use the minimum number of time slots and channels so that minimal latency can be achieved. The nodes must also be placed in such a way that there is no overlapping transmissions amongst them.

5 DELAY ANALYSIS UNDER CAP REDUCTION

For our numerical analysis first we derive the value of $N_{CFP(n)}$, which is dependent on the values of the MO, BO and SO. This value is calculated to know the overall GTSs resources available under CAP reduction, then we calculate its respective delay. D_{Max} represents the maximum delay a transmission has to undergo for a successful GTS allocation in a multisuperframe.

In accordance to the standard, there will be an Inter Frame Spacing (IFS) period between every successful transmission. Depending on their size if less than *aMaxSIFSFrameSize*, it is called Short Inter Frame Spacing (SIFS), else it is called Long Inter Frame Spacing (LIFS). Under LIFS, the size extends for a minimum period of *minLIFSPeriod* symbols. This IFS contributes to the delay along with other parameters such as L_{frame}, the frame length, R_s, the symbol rate and R_b the bit rate. In accordance to research work [13] done towards calculating delay in a superframe intervals, the maximum delay can be given as:

$$D_{max} = \begin{cases} D_{SIFS} = \dfrac{(L_{frame} \times R_s)}{R_b} + minSIFSPeriod, \\ D_{LIFS} = \dfrac{(L_{frame} \times R_s)}{R_b} + minLIFSPeriod \end{cases} \quad (1)$$

The duration of the multisuperframe slot will depend on the multisuperframe order (MO) issued by the PAN coordinator. This varies with respect to topology obtained through RPL. Let T_{MS} be the duration of the multisuperframe slot, N_{MD} be the total number of symbols forming the multisuperframe, N_{MD_i} be the total number of symbols forming the multisuperframe since the value of $SO = 0$,

$$T_{MS} = \frac{N_{MD}}{T_{CAP} + T_{CFP}} = N_{MD_i} \times 2^{MO-4} \quad (2)$$

Equation 2 stands true for a scenario with CAP reduction for a single multisuperframe period encompassing all the GTSs in the CFP time period. It also considers a CAP region of duration T_{CAP}.

A single GTS can span across several superframe slots, and so we should provide a constraint on it. GTS must be greater than the total forward delay D_{max}. Let us consider N_{min} to be the minimum number of superframe slots a single GTS can extend over. The total forward delay D_{max} can be given by:

$$D_{max} = T_{MS} \times N_{min} \quad (3)$$

As we consider a critical data oriented network, we neglect the delay that occurs in the CAP region of the traditional IEEE 802.15.4. Under CAP reduction the absolute number of GTSs is not certain, however it can be expressed as $m \times N_{CFP}$, where m is the number of channels and N_{CFP} is the timeslots in CFP. From these, the maximum number of GTSs that can be allocated to devices can be given by:

$$N_{CFP(n)} = min\left(\left\lfloor \frac{(T_{CAP} + T_{CFP})(1 - \frac{T_{CAP}}{T_{MS}})}{N_{min}} \right\rfloor, m \times N_{CFP}\right) \quad (4)$$

As given in Figure 2, for the need of simplicity, we consider a CFP with just 2 timeslots and 2 channels (4 available GTSs resources), this can be generalized for a larger number of channels. In this Figure we present several scenarios across the different time intervals. A delay analysis was performed for all these scenarios.

The scenarios (Figure 2) taken for the numerical analysis are listed as follows:

(i) From T1 to T2: This is a multisuperframe in which normal DSME without CAP reduction is employed. The multisuperframe in this scenario is expected to support 5 GTS transmissions. It should be

noted that without CAP reduction, the superframe has to wait for a "duration of CAP" before it is able to transmit.

(ii) From T2 to T3: This is a multisuperframe with CAP reduction employed in it. Unlike the previous discussed case, the final transmission need not wait for a CAP.

(iii) From T3 to T4: This is a multisuperframe with CAP reduction employed and the number of transmissions it has to accommodate is 13. But the MO in this scenario is static, the final transmission of this use case also has to wait for an entire CAP period before its transmission.

(iv) From T4 to T5: This is a multisuperframe with CAP reduction employed with a static MO, but it should be noted that it just needs to accommodate 3 GTSs. As a result of this 8 GTSs remain unoccupied contributing to the wasted bandwidth eventually affecting the overall throughput of the network.

(v) From T5 to T6: This holds the same condition as scenario *iii*, but with DynaMO, PAN-C counts the number of transmissions to be accommodated by the CFP. As value is above the number of timeslots available, it increases the MO by 1 adding a superframe to the multisuperframe. In this use case, the MO is 2, thus joining 3 superframes within a multisuperframe, eventually reducing the overall delay.

(vi) From T6 to T7: In this case the number of GTSs to be accommodated is 4. PAN-C deploys CAP reduction in this scenario eventually providing a single superframe to accommodate the 4 transmissions. This method will reduce the wastage of bandwidth thus increasing the throughput.

We calculated the delay of the network for all the use cases as mentioned above using Equation 2. We considered a network that dynamically grows and thus demanding more GTSs resources. For CAP reduction scenarios, we take the value of MO to be 1. For this numerical analysis we consider idle time to be 0 and a constant bit rate of 1kbps.

From Figure 3, it can be noted that under traditional DSME, the transmission delay of the GTS frames starts to increase at a point where the multisuperframe cannot allocate more GTSs. As the MO is constant, delay inevitably starts to increase when enough resources are not available, imposing a transmission deference to the next superframe. However, if CAP reduction is triggered, delay is much smaller when compared to the normal DSME, as more GTSs resources are available. With DynaMO, the MO is increased when more resources are needed, hence, it provides better results than networks with solely CAP reduction enabled (by 15%) and DSME networks with constant, non-dynamic settings (by 35%).

6 SIMULATION ANALYSIS

For evaluating DynaMO, we use the OpenDSME simulation platform [7]. OpenDSME is a OMNET++/C++ simulation based environment that is dedicated for the simulation of the IEEE 802.15.4e DSME protocol. OpenDSME also provides the possibility of implementing a viable network layer on top of it. The DSME sublayer of OpenDSME employs a typical slot based reservation system for a schedule that is provided by the top layer.

In our model, we provide BO, MO, SO and the CAP reduction primitives as a direct input. Other network simulation parameters such as traffic rate, the burst size, the interference and the mobility

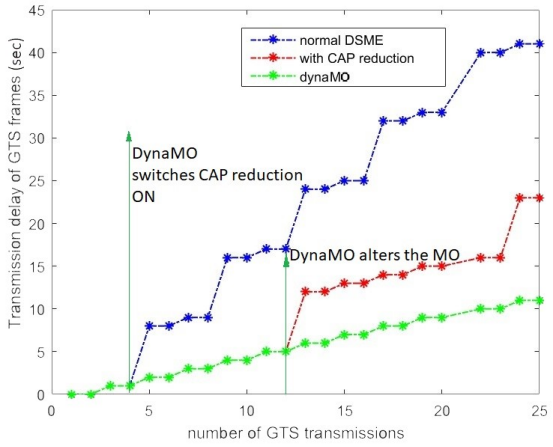


Figure 3: Comparison in terms of delay

Application type	BO	SO	MO	CAP reduction
Delay sensitive	6	0	1	Enabled
Reliability sensitive	8	3		Disabled
Energy Critical	14	1	14	Enabled
High throughput	10	5	6	Disabled
Large scale	10	1	8	Enabled

Table 2: Application scenarios for BO,MO,SO variation

models are also be given directly. Furthermore, there is also a possibility to input the schedule in accordance with a static schedule. We have also incorporated delay and throughput parameters [9] in the network definition files to obtain the appropriate output for the network simulated. The simulations were carried out on a mesh network and the overall network delay was observed.

IEEE 802.15.4e standard provides certain suggestive values for BO, SO and MO for application specific scenarios (Table II). These values when kept static provide us a multisuperframe format with a specific number of superframes. For the delay sensitive settings BO, SO and MO is 6,0,1, hence number of superframes within a multisuperframe will be 2. In such a case, a transmission need not wait for a long time for the eventual transmission. However, when kept static, it may result in increased latency.

In Table 3, we provide the parameters that we have used for all the scenarios we put under extensive simulations.

6.1 Comparison against static CAP reduction

For this comparison we calculate the values of the overall delay of the network with respect to the number of GTSs transmissions. For this simulation we analyze the delay of 50 nodes under different traffic rates ranging from 5-75 Kbps for CAP reduction and without CAP reduction scenarios in Fig 4. This result complements our theoretical analysis shown in Figure 2, clearly showing DynaMO in action.

With a limited number of GTSs transmissions, the delay performance does not have a significant decrease with the scenarios without CAP reduction (5,10,15 transmissions). Delay performance

Parameters	6.1 against CAP reduction	against different traffic rates	6.2 against high throughput settings
Packet Length	75B	75, 100B	75, 100B
Packet Traffic Interval	50, 30, 15ms	50, 30, 15ms	50, 30, 15ms
Destination	sink	sink	sink
MAC Queue Length	30	30	30
MAC Frame Retries	7	7	7
BO	6	10	6, 10
SO	3	5	3, 5
MO	DynaMO	6, DynaMO	4, 6, DynaMO
Number of Nodes	5 to 50	5 to 50	5 to 50
Traffic Rate	15, 25, 75k Kps	15, 25, 50, 75 Kbps	25, 50, 75, 100 Kbps
CAP Reduction	DynaMO	OFF	ON/OFF/DynaMO

Table 3: Simulation Parameters

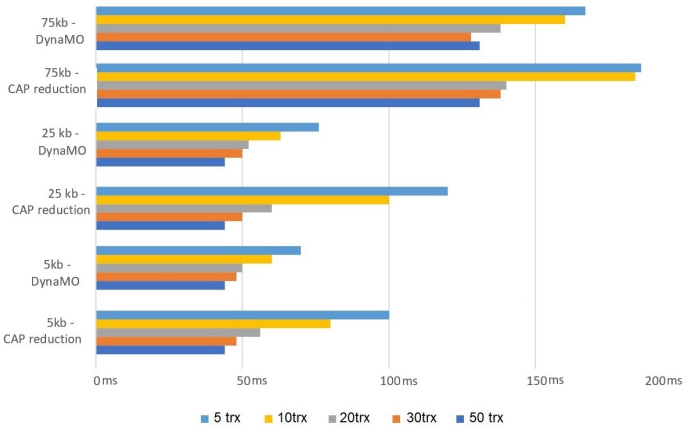


Figure 4: Delay analysis against static CAP reduction settings

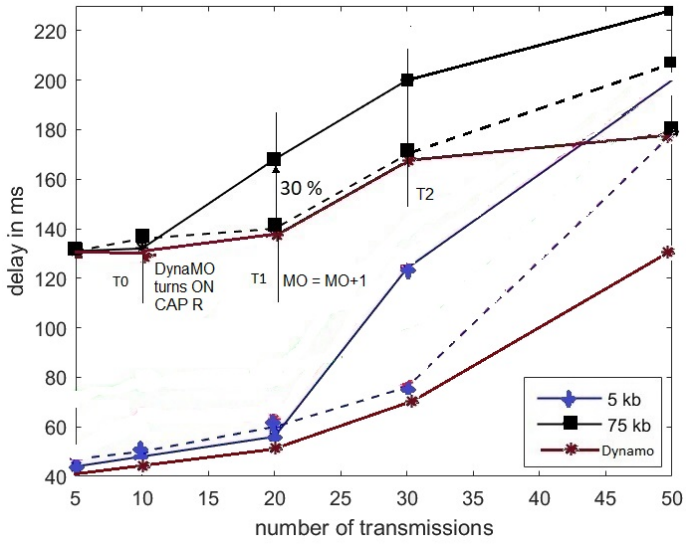


Figure 5: Delay analysis for 75 and 5 Kbps traffic rate

is in-fact sometimes better without CAP reduction when the number of nodes is less than 10, due to less wasted bandwidth. However, as the number of transmissions increases, with CAP reduction, delay is minimized. This is due to the fact that nodes need not wait till another superframe duration to accommodate the transmissions that did not occur during the initial superframe interval.

DynaMO switches the CAP reduction parameters according to the resource requirements and hence doesn't compromise on the delay for those scenarios in which CAP reduction is still not needed, offering a clear advantage over static settings.

For clear understanding, the example of DynaMO is demonstrated along with the 75Kbps and the 5Kbps case in Figure 5. The dotted lines represent the scenario with static CAP reduction. Initially, the CAP reduction is OFF providing minimal delay (similar to the scenario without CAP reduction), whereas at T0, due to the scarcity of the resources, the CAP reduction is turned ON dynamically and we can witness a reduction in delay by almost 30%. Above 20 scheduled transmissions, an increase in MO under DynaMO further maintains a lower delay in comparison to static settings including the CAP reduction enabled setting.

6.2 Comparison against Delay sensitive and high throughput settings

In this experiment, we compare the static high throughput settings and the static delay sensitive settings (dotted lines) with DynaMO. In Figure 6, we demonstrate this comparison over 100Kbps. The other traffic rates also have a similar behavior. OpenDSME does not allow the value of SO to be set to '0' by default. So we took another delay sensitive setting of BO, SO and MO to be 6,3,4 such that the number of superframes within a multisuperframe will be 2 and every beacon interval will have 4 multisuperframes.

The delay is always higher in the high throughput setting, and this gap increases with traffic rate. The higher MO in the high throughput settings causes a wastage of bandwidth which results in additional delay, contrary to the time-sensitive settings in which the superframes are closely packed. We observe almost 20-25% reduction of delay under delay sensitive settings when the number of transmissions is maximized. However, relying on static settings which provide shorter MO is often not an adequate solution, as it

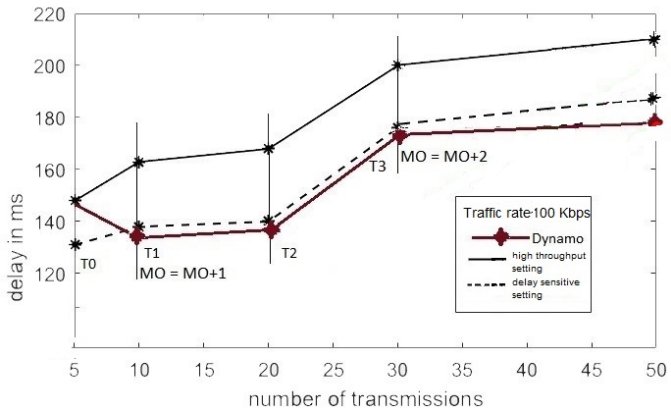


Figure 6: Delay Analysis against delay-sensitive settings

can compromise the QoS if the network needs to accommodate an increase in traffic.

In Figure 6, at T0, we start DynaMO with a high throughput setting, consisting of one superframe in a multisuperframe. However, as the timeframe moves on to T1 and the number of transmissions increases, DynaMO automatically adapts its MO based on the number of resources. In this case, by increasing MO, DynaMO packs more superframes within the beacon interval, providing more GTS bandwidth and eventually obtaining lesser delay.

We can observe a significant reduction in delay, even when compred against the static delay-sensitive settings. Notice, that the delay-sensitive setting does not outperform DynaMO in terms of delay when the the number of transmissions is lesser. Although this could somewhat appear counter-intuitive, as the number of transmissions increases, the short MO is not able to accommodate the transmissions causing a deference of transmissions to the subsequent superframes. This increases delay and its effect is particularly visible above 35 scheduled transmissions. With DynaMO employed, we are able to witness 15-30% reduction in delay when compared to the standard presets.

7 FUTURE WORK

In this paper we introduced an efficient multisuperframe tuning technique that can switch CAP reduction and tune the MO on demand, on a dynamic DSME network. From our simulations and numerical analysis, we learn that static settings are an impediment when it comes to large scale DSME network. With our tuning technique, we were able to obtain 15-35% of reduction in the overall delay of the network.

The network analysis in this paper was focused on delay over a mesh network. DynaMO also impacts other QoS parameters such as throughput and bandwidth utilization, and these will be objective of further work, while applying our technique into other different topologies and scenarios. We hope this algorithm will be part of a package aiming at dynamically improving the QoS of DSME, as we believe this is necessary for this protocol to achieve its full potential.

Though DSME has all the factors to become a de-facto protocol for critical IoT, not much research work has been done on implementing it in real platforms, nor over real time operating systems. We intend to implement DynaMO and DSME over a Commercial off The Shelf Technologies (CoTS) to better assess its capabilities over real hardware.

ACKNOWLEDGEMENTS

This work was partially supported by National Funds through FCT/MEC (Portuguese Foundation for Science and Technology) and co-financed by ERDF (European Regional Development Fund) under the PT2020 Partnership, within the CISTER Research Unit (CEC/04234); also by FCT/MEC and the EU ECSEL JU under the H2020 Framework Programme, within project ECSEL/0002/2015, JU grant nr. 692529-2 (SAFECOP). This work is also support by the Research and Translation Center (RTC) at Prince Sultan University.

REFERENCES

[1] 2011. IEEE Standard for Local and metropolitan area networks–Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs). *IEEE Std 802.15.4-2011 (Revision of IEEE Std 802.15.4-2006)* (Sept 2011), 1–314. https://doi.org/10.1109/IEEESTD.2011.6012487

[2] 2016. IEEE Standard for Low-Rate Wireless Networks. *IEEE Std 802.15.4-2015 (Revision of IEEE Std 802.15.4-2011)* (April 2016), 1–709. https://doi.org/10.1109/IEEESTD.2016.7460875

[3] Giuliana Alderisi, Gaetano Patti, Orazio Mirabella, and Lucia Lo Bello. 2015. Simulative assessments of the ieee 802.15. 4e dsme and tsch in realistic process automation scenarios. In *Industrial Informatics (INDIN), 2015 IEEE 13th International Conference on*. IEEE, 948–955.

[4] M. Anwar and Y. Xia. 2014. IEEE 802.15.4e LLDN: Superframe configuration for networked control systems. In *Proceedings of the 33rd Chinese Control Conference*. 5568–5573. https://doi.org/10.1109/ChiCC.2014.6895891

[5] S. Capone, R. Brama, F. Ricciato, G. Boggia, and A. Malvasi. 2014. Modeling and simulation of energy efficient enhancements for IEEE 802.15.4e DSME. In *2014 Wireless Telecommunications Symposium*. 1–6. https://doi.org/10.1109/WTS.2014.6835017

[6] Joseph Jeon, Jong Wook Lee, Jae Yeol Ha, and Wook Hyun Kwon. 2007. DCA: Duty-cycle adaptation algorithm for IEEE 802.15. 4 beacon-enabled networks. In *Vehicular Technology Conference, 2007. VTC2007-Spring. IEEE 65th*. IEEE, 110–113.

[7] Florian Kauer, Maximilian Köstler, Tobias Lübkert, and Volker Turau. 2017. OpenDSME-A portable framework for reliable wireless sensor and actuator networks. In *Networked Systems (NetSys), 2017 International Conference on*. IEEE, 1–2.

[8] Anis Koubaa, Mário Alves, and Eduardo Tovar. [n. d.]. i-GAME: an implicit GTS allocation mechanism in IEEE 802.15. 4 for time-sensitive wireless sensor networks. In *Real-Time Systems, 2006. 18th Euromicro Conference on*. IEEE, 10–pp.

[9] Harrison Kurunathan. 2019. OpenDSME support file. https://github.com/harrisonkurunathan/throughputnedfile. (2019).

[10] Harrison Kurunathan, Ricardo Severino, Anis Koubaa, and Eduardo Tovar. 2018. IEEE 802.15. 4e in a Nutshell: Survey and Performance Evaluation. *IEEE Communications Surveys & Tutorials* (2018).

[11] Harrison Kurunathan, Ricardo Severino, Anis Koubâa, and Eduardo Tovar. 2018. *RPL over DSME: A Technical Report*. Technical Report. CISTER-Research Centre in Realtime and Embedded Computing Systems.

[12] Harrison Kurunathan, Ricardo Severino, Anis Koubâa, Eduardo Tovar, et al. 2017. Worst-Case Bound Analysis for the Time-Critical MAC behaviors of IEEE 802.15. 4e. In *13th IEEE International Workshop on Factory Communication Systems Communication in Automation (WFCS 2017). 31, May to 2, Jun, 2017.*

[13] Pangun Park, Carlo Fischione, and Karl Henrik Johansson. 2009. Performance analysis of GTS allocation in beacon enabled IEEE 802.15. 4. In *2009 6th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks*. IEEE, 1–9.

[14] Ricardo Severino, Nuno Pereira, and Eduardo Tovar. 2014. Dynamic cluster scheduling for cluster-tree WSNs. *SpringerPlus* 3, 1 (2014), 493.

[15] Peter JM Van Laarhoven and Emile HL Aarts. 1987. Simulated annealing. In *Simulated annealing: Theory and applications*. Springer, 7–15.

A Preliminary Roadmap for Dependability Research in Fog Computing

Zeinab Bakhshi

Mälardalen University

Sweden

zeinab.bakhshi@mdh.se

Guillermo Rodriguez-Navas

Nokia Bell Labs

Israel

guillermo.rodriguez-navas@nokia-bell-labs.com

ABSTRACT

Fog computing aims to support novel real-time applications by extending cloud resources to the network edge. This technology is highly heterogeneous and comprises a wide variety of devices interconnected through the so-called fog layer. Compared to traditional cloud infrastructure, fog presents more varied reliability challenges, due to its constrained resources and mobility of nodes. This paper summarizes current research efforts on fault tolerance and dependability in fog computing and identifies less investigated open problems, which constitute interesting research directions to make fogs more dependable.

CCS CONCEPTS

• **Computer systems organization** → **Reliability**; **Availability**; **Redundancy**;

KEYWORDS

Fog Computing, Edge Computing, Internet of Things, Real-time, Fault tolerance, Dependability

1 INTRODUCTION

Fog computing is a recent computational paradigm, first introduced by Cisco, to extend cloud computing computational resources, closer to the edge of the network [7, 19]. Fog is a middle layer between the cloud and the devices to have more efficient data processing, effective analysis and storage scalability. It also reduces the amount of data transmitted to the cloud [14]. There is a general understanding that this technology is suitable for Cyber-Physical Systems, IoT and Industrial IoT (IIoT) in different application areas. For instance, smart cities, agriculture domains, vehicular systems, industrial automation, health-care and robotics. It is also claimed that fog represents a solution to improve latency for distributed control systems in general.

According to Bonomi et al. [7] fog computing has the following characteristics, a) Low latency and location awareness; b) Supports geographic distribution; c) End device mobility; d) Capacity of processing with a high number of nodes; e) Wireless access; f) Real-time applications and g) Heterogeneity. These characteristics make fog computing a suitable solution for overcoming problems manifested by the use of traditional cloud computing in Internet of Things (IoT), like high mobility and low latency, but they also give rise to new dependability challenges. Note that each of the factors mentioned above represents a difficulty for achieving dependability, so the combination of all of them makes the whole undertaking even more challenging.

Dependability is the ability of a system to supply trusted and available services. A *dependable* system is a system which is able to avoid service failures that are more frequent and more severe than is acceptable. There are many dimensions that should be considered to analyze whether a fog-based solution is dependable, such as availability, reliability, performability, maintainability; which are well-known dependability attributes (or requirements) [4]. At the same time, there are different ways to implement a dependable system, for instance using fault tolerance algorithms and redundancy techniques. Given the interest in fog computing and the difficulties it introduces in terms of dependability, it is important to understand how dependability and fault tolerance are addressed in the literature on fog computing.

This paper summarizes fog computing dependability requirements and discusses the gap, in terms of dependability, of the existing solutions with respect to the desired dependability requirements. After presenting a basic hierarchical structure of fog architecture, in this paper we will 1) identify and classify current research approaches for dependability in fog computing, 2) compare different proposed solutions considering traditional dependability notions for critical systems, and 3) discuss research gaps related to fog computing dependability. We realized that there is a range of terms alternatively used for fog computing by authors in the literature. For instance edge clouds, cloudlets, mobile edge computing, etc. We considered these terms as related technologies to fog computing in our study. The remainder of this paper is organized as follows. In Section 2 we present the fog computing architecture. In Section 3, we review current approaches for dependability solutions in fog computing. In Section 4 we discuss the gaps between current research approaches and fog computing dependability requirements and finally we conclude our work in Section 5.

2 FOG COMPUTING ARCHITECTURE

Fog computing is a highly virtualized platform that provides storage, communication, computation, controlling, machine learning services in a decentralized network closer to devices [1, 15]. To the best of our knowledge, there is no reference architecture for fog computing, however, there are basic architectures for fog proposed in the literature, like [15, 21, 25]. The proposed architectures are mostly constituted by a three-layer structure, as depicted in Figure 1, which includes a layer between cloud and devices, known as fog layer. This fog layer carries out the task of computation from clouds closer to the network edge.

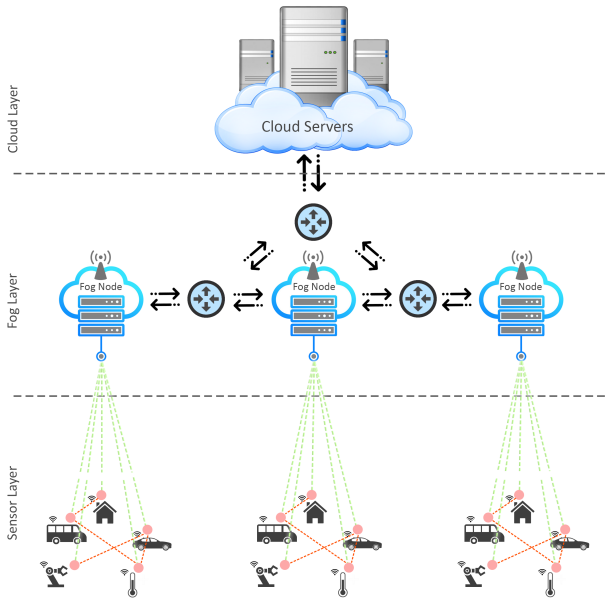


Figure 1: The basic hierarchical architecture of fog computing.

There is a somehow diffuse border between fog computing and the paradigm called edge computing, but there is an important difference that we will apply in this work: edge computing does not preclude the existence of a cloud to which the intermediate nodes are connected. However, whenever edge is used in combination with cloud, one can arguably say that both paradigms are equivalent. For this reason, we also investigated systems introduced as edge computing but have considered them as instances of fog.

In the following we will describe each layer of the hierarchical fog computing architecture:

2.1 Cloud Layer

This layer consists of multiple, powerful computational resources, storage and servers, which are capable of processing, analyzing and storing large amounts of data. Cloud computing provides services for different application domains, for instance, vehicular systems, smart cities, smart factories, health-care, etc. [25]. The clouds are efficiently managed and scheduled by some control strategies to improve utilization of the cloud resources. Although cloud computing is empowered by huge computational resources and storage capacities, for certain tasks, e.g. those requiring low latency, it might be better to release their execution to other parts of the system, closer to the edge [27].

2.2 Fog Layer

According to the OpenFog consortium, the fog computing model moves computation from the cloud closer to the network edge, by placing geo-distributed computational resources between the cloud and sensor layer [21]. The Fog computing layer is composed of fog platforms (the fog nodes) which rely on highly virtualized resources running under hypervisors. Fog platforms are constituted by a large number of fog nodes consisting of routers, switches, Wireless Access Points (WAP), Road Side Units (RSUs), gateways, wireless set-top boxes, network bridges and cellular base stations [6, 11].

These fog nodes, which can be fixed or mobile, are distributed in different geographical locations to provide services in proximity of edge devices. Given that the edge devices (Sensor layer) can be mobile, the Fog layer should enable reallocation of tasks and resources at runtime. In fact, the high mobility characteristic of fog computing typically gives the impression that fog nodes enter in and out the network, which may give rise to novel availability issues. In terms of security, the existence of this intermediate level also increases the attack surface of the system considerably.

2.3 Sensor Layer

This layer is bottom layer in the hierarchical architecture which consists of devices, sensors, actuators in a physical environment; for instance, vehicles, smart cards, IoT devices, etc. Devices in this layer are geographically distributed, can be fixed or mobile, and require minimal computational resources, being typically very energy-constrained. Usually utilized as smart sensing devices, they sense data and gather information, and then send it to the upper layer for processing, storage and distribution [2].

3 CURRENT DEPENDABILITY APPROACHES FOR FOG COMPUTING

Dependability approaches for fog computing are mainly proposed to address dependability objectives, redundancy models and fault management solutions. Figure 2 present a summary of the approaches in our literature review.

3.1 Dependability Objectives

Dependability requirements for fog computing are not clearly defined, as fog computing is a very recent technology. Our review of existing literature shows that authors differ significantly from each other in terms of the types of faults and errors they address, the method applied and even the dependability requirements themselves. Our study shows that the most common objectives are improving availability, reliability and Quality of Service (QoS). The ways to improve these attributes are typically based on redundancy models which are explained in the following subsection. Our study also shows that scalability, i.e. the ability to provide service for a large number of devices in the Sensor layer, is a crucial aspect of Fog. This can be related to the dependability attribute of performability.

3.2 Redundancy Models

Proposed redundancy models has been applied at different levels of the systems architecture: the communication links, the computing nodes and the application software. For instance, regarding network connectivity, Cau et al. used 5G communication to satisfy network reliability [10]. Wiss & Forsstrom. consider higher network connectivity as availability by using SCTP protocol instead of TCP [29].

Other works also consider the possibility of node failure. Itani et al. proposed dynamic failure recovery to improve node availability [16]. Zhou et al. used message broadcasting to check node availability for offloading tasks in case of fog node or link failures [33]. Okafor et al. proposed using of Spin-Leaf topology in fog network to ensure availability [20].

There is an interesting family of solutions that rely on software reallocation in order to increase service reliability/availability. Saqib & Hamid. proposed a task off-loading solution to ensure reliable computation in fog computing and IoT network [26]. Aral & Brandic. focused on QoS of VMs in an edge network infrastructure [3] and Osanaiye et al. proposed a live VM migration framework to increase QoS by improving availability of VM fog nodes [22]. Rimal et al. focused on improving system performance to promote QoS [24].

But, although authors allegedly address all these requirements, quantitative goals which would help us to define system thresholds are seldom or partially reported. In the scheme proposed in [9], authors considered strategies to minimize bandwidth and storage usage in which they reported percentage values of the gap between optimal scheme and practical measurements, lower than 6.2% and 30% for bandwidth and storage usage respectively.

Availability of replicated nodes or links are checked using different monitoring tools [17] or calculated via mathematical methods [9] or the use of machine learning algorithms [3].

With respect to the applied redundancy schemes, we found out that all types of redundancy have been used by different authors. It was observed that sometimes natural redundancy has been used for path redundancy, for instance as provided by wireless broadcast in

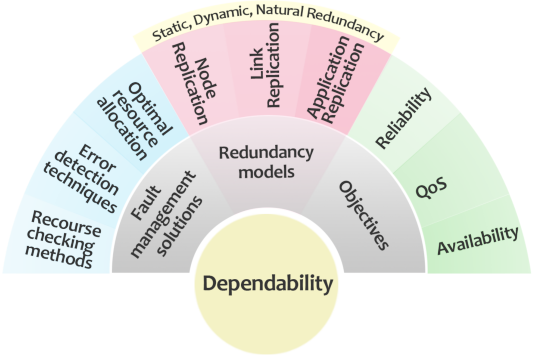


Figure 2: Summary of current dependability approaches for fog computing.

[10]. The most common approach is Primary/Backup redundancy, with reconfiguration upon failure. Schemes relying both on Active replication (also known as Active/Active or Hot stand-by) and Passive replication (also known as Active/Passive or Cold stand-by) were found. For instance, Banson et al. proposed a dynamic path selection method based on Software defined networks (SDN), leveraging SDN monitoring tools to check the links availability status [5]. In another work [12] Maximum Distance Separable code (MDS) is used for dynamic clustering to find redundant nearby nodes. Although there are some approaches using dynamic redundancy, most often static allocation of redundancy is used. Cau et al. proposed static signal forwarding to available nodes in case of node failure in the network [10]. Other works also proposed passive replication in which virtual backup resources are pooled and shared across multiple virtual infrastructure [28, 31].

3.3 Fault Management Solutions

Solutions for fault management in fog computing proposed in the available publications on dependability and fog computing shows that authors have more importantly focused on these specific problems: a) Optimal allocation of redundancy, to reduce utilization. An instance is the work of Mennes et al. [18] which proposed an algorithm for optimal application placement; b) Techniques for error detection and reconfiguration upon failure, like e.g. Chervaykov et al. proposed a reconfigurable data storage system based on Redundant Residue Number System (RRNS) [13] and Xiao et al. [30] proposed a re-transmission method to re-send data in case of links or nodes failures or delay issues in a fog network; c) Methods for checking availability of redundancy, like monitoring tools, especially tailored for resilient networks [32].

4 DISCUSSION

Our study has provided us with very useful information about the current state of the art regarding dependability and fog computing. We have identified a number of research topics that seem to have

received much attention from the research community. Namely: the trade-off between resource utilization and fault tolerance, the use of redundancy methods to increase availability and, last, the trade-off between reliability and timeliness, particularly for node replication schemes.

However, there is also an extensive list of challenges that have received very little attention. In the following, we summarize the open research problems that, in our opinion, deserve further investigation. The list does not intend to be exhaustive, but it defines a preliminary roadmap of the issues that need to be addressed next.

Introducing more complex failure modes. We noted that only simple (benign) failure modes have been considered in the literature. Authors typically consider crash and omission failures for communication links and available vs. non-available node failures (i.e. Stop failure semantics). However, more complicated failure modes like Byzantine or arbitrary failures, late performance and failures due to malicious faults remain unaddressed in this heterogeneous fog environment with complicated functionality. Another aspect that deserves more research is identifying system specification failures. For instance, late performance, bad design or wrong demand expectation/dimensioning might cause general failures as they have been disregarded while designing dynamic mechanisms such as dynamic reallocation of software. To give just an example, both intentional and unintentional Denial of Service (DoS) failures are possible in systems that do not properly handle oversized loads, even in cases where system allows dynamic changes. We believe that as the technology extends to more domains, the nature and severity of the faults that need to be addressed will have to be clarified.

Integration of multiple levels of redundancy. Since fog is a complex, multi-layered architecture, we need to consider failure probability in each layer of fog computing. So far, redundancy schemes have been proposed individually, and the potential interference between them has not been investigated. This also includes clarifying the interaction between application and data replication throughout the architecture, including data source and data transmission, which can be upward (from clouds to fog), downward (from sensors to fog) or internally cashed in fog node. All of this makes the fault-tolerant replication model more complicated, comparing to cloud and traditional critical systems.

Security issues aggravated by faults. We found out that there are a number of papers dealing with security [8, 13, 20, 22], but none of them addressed security for fog computing in the presence of unintentional faults. On the other hand, methods to achieving replication securely under differing threat models has not been specifically surveyed to provide secure redundancy techniques.

Error propagation through the fog structure. Uncontrolled error propagation is an important problem in any dependable system. The usual way to handle this problem is by defining and substantiating appropriate error-containment regions. This work has not been done for existing fog computing architectures. This aspect is related to the security problems discussed above, since correct error-containment is a good support for security, but it also concerns non-malicious faults, which can spread as subsystem errors and cause unexpected failures in other parts of the system. In a highly-dynamic system like the fog, poor handling of error propagation might even lead to instability system-level problems. This also opens an opportunity to investigate novel methods for error forecasting and dynamic error containment.

Fault recovery and node reintegration. Current approaches studied in this work have investigated different methods for fault detection, fault-tolerance, fault prevention and fault diagnostics. However, in a long-lifed system like the fog, it is also needed to develop methods that allow faulty components to recover and be reintegrated in the system operation. This can prevent system failure or shut down caused by fast redundancy attrition.

Scalability concerns. Fog nodes should be able to provide services for a large number of heterogeneous devices in different application areas. These application domains can require large-scale deployment of nodes, also for safety-critical domains. For instance, firefighting, transportation systems and industrial robotics. When a fog node fails in such large-scale critical systems, it is usually difficult to coordinate the huge number of sensors and devices in the presence of faults or to recover from failures. Similar unknown risks, caused by the large system size and the massive number of components, might be found in future applications.

A comprehensive fault management framework is missing. A fault management framework is a part of large network management structure. This framework can address faults in a higher level as well as designing a high level management infrastructure for addressing faults in a system [23]. Although there are some frameworks proposed for Fog computing and fault tolerance in Fog networks, they do not address faults in all aspects. For instance, considering connection failure, node failure, application placement, task management, etc. combined in the same framework package. As indicated above, certain notions like fault diagnosis and fault treatment have received little attention, as well as the threats posed by malicious faults. All these aspects should also be integrated in this fault management framework. Similarly to other large-scale networked systems, the fog allows application of novel methods based on statistical learning, such as machine learning, in order to identify anomalies and forecast faults, but this type of work is still in its infancy. A suitable framework should include methods for data collection and collection, and a repository of considered faults and mitigation techniques.

5 CONCLUSION

This paper has reviewed the current state of the art regarding dependability and fog computing. We have summarized the current research efforts and discussed a list of open research problems.

ACKNOWLEDGMENT

This research has received funding from the European Union's Horizon 2020 Research and Innovation programme under the Marie Skłodowska-Curie grant agreement No 764785, and also from the VINNOVA project 2018-02437.

REFERENCES

[1] M. Aazam and E. Huh. 2016. Fog Computing: The Cloud-IoT, IoE Middleware Paradigm. *IEEE Potentials* 35, 3 (May 2016), 40–44.

[2] Nasir Abbas, Yan Zhang, Amir Taherkordi, and Tor Skeie. 2018. Mobile edge computing: A survey. *IEEE Internet of Things Journal* 5, 1 (2018), 450–465.

[3] A. Aral and I. Brandic. 2017. Quality of Service Channelling for Latency Sensitive Edge Applications. In *2017 IEEE International Conference on Edge Computing (EDGE)*. 166–173.

[4] A. Avizienis, J. C. Laprie, B. Randell, and C. Landwehr. 2004. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing* 1, 1 (Jan 2004), 11–33.

[5] K. E. Benson, G. Wang, N. Venkatasubramanian, and Y. Kim. 2018. Ride: A Resilient IoT Data Exchange Middleware Leveraging SDN and Edge Cloud Resources. In *2018 IEEE/ACM Third International Conference on Internet-of-Things Design and Implementation (IoTDI)*. 72–83.

[6] Kashif Bilal, Osman Khalid, Aiman Erbad, and Samee U. Khan. 2018. Potentials, trends, and prospects in edge technologies: Fog, cloudlet, mobile edge, and micro data centers. *Computer Networks* 130 (2018), 94 – 120.

[7] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. 2012. Fog Computing and Its Role in the Internet of Things. In *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing (MCC '12)*. ACM, New York, NY, USA, 13–16.

[8] H. P. Breivold and K. Sandström. 2015. Internet of Things for Industrial Automation – Challenges and Technical Solutions. In *2015 IEEE International Conference on Data Science and Data Intensive Systems*. 532–539.

[9] J. A. Cabrera, D. E. Lucani., and F. H. P. Fitzek. 2016. On network coded distributed storage: How to repair in a fog of unreliable peers. In *2016 International Symposium on Wireless Communication Systems (ISWCS)*. 188–193.

[10] E. Cau, M. Corici, P. Bellavista, L. Foschini, G. Carella, A. Edmonds, and T. M. Bohnert. 2016. Efficient Exploitation of Mobile Edge Computing for Virtualized 5G in EPC Architectures. In *2016 4th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud)*. 100–109.

[11] X. Chen and L. Wang. 2017. Exploring Fog Computing-Based Adaptive Vehicular Data Scheduling Policies Through a Compositional Formal Method-PEPA. *IEEE Communications Letters* 21, 4 (April 2017), 745–748.

[12] X. Chen, X. Wen, L. Wang, and W. Jing. 2018. A Fault-Tolerant Data Acquisition Scheme with MDS and Dynamic Clustering in Energy Internet. In *2018 IEEE International Conference on Energy Internet (ICEI)*. 175–180.

[13] Nikolay Chervyakov, Mikhail Babenko, Andrei Tchernykh, Nikolay Kucherov, Vanessa Miranda-López, and Jorge M. Cortes-Mendoza. 2019. AR-RRNS: Configurable reliable distributed data storage systems for Internet of Things to ensure security. *Future Generation Computer Systems* 92 (2019), 1080 – 1092.

[14] A. V. Dastjerdi and R. Buyya. 2016. Fog Computing: Helping the Internet of Things Realize Its Potential. *Computer* 49, 8 (Aug 2016), 112–116.

[15] Pengfei Hu, Sahraoui Dhelim, Huansheng Ning, and Tie Qiu. 2017. Survey on fog computing: architecture, key technologies, applications and open issues. *Journal of Network and Computer Applications* 98 (2017), 27 – 42.

[16] May Itani, Sanaa Sharafeddine, and Islam ElKabani. 2018. Dynamic multiple node failure recovery in distributed storage systems. *Ad Hoc Networks* 72 (2018), 1 – 13.

[17] A. Jonathan, M. Uluyol, A. Chandra, and J. Weissman. 2017. Ensuring reliability in geo-distributed edge cloud. In *2017 Resilience Week (RWS)*. 127–132.

[18] R. Mennes, B. Spinnewyn, S. Latré, and J. F. Botero. 2016. GRECO: A Distributed Genetic Algorithm for Reliable Application Placement in Hybrid Clouds. In *2016 5th IEEE International Conference on Cloud Networking (Cloudnet)*. 14–20.

[19] M. Mukherjee, R. Matam, L. Shu, L. Maglaras, M. A. Ferrag, N. Choudhury, and V. Kumar. 2017. Security and Privacy in Fog Computing: Challenges. *IEEE Access* 5 (2017), 19293–19304.

[20] Kennedy Chinedu Okafor, Ifeyinwa E Achumba, Gloria A Chukwudebe, and Gordon C Ononiwu. 2017. Leveraging fog computing for scalable IoT datacenter using spine-leaf network topology. *Journal of Electrical and Computer Engineering* 2017 (2017).

[21] OpenFog Consortium Architecture Working Group. 2017. OpenFog Reference Architecture for Fog Computing. *OpenFog* February (2017), 1–162.

[22] O. Osanaiye, S. Chen, Z. Yan, R. Lu, K. R. Choo, and M. Dlodlo. 2017. From Cloud to Fog Computing: A Review and a Conceptual Live VM Migration Framework. *IEEE Access* 5 (2017), 8284–8300.

[23] Lilia Paradis and Qi Han. 2007. A Survey of Fault Management in Wireless Sensor Networks. *Journal of Network and Systems Management* 15, 2 (01 Jun 2007), 171–190.

[24] B. P. Rimal, D. Pham Van, and M. Maier. 2017. Mobile-Edge Computing Versus Centralized Cloud Computing Over a Converged FiWi Access Network. *IEEE Transactions on Network and Service Management* 14, 3 (Sep. 2017), 498–513.

[25] José Santos, Tim Wauters, Bruno Volckaert, and Filip De Turck. 2017. Fog computing: Enabling the management and orchestration of smart city applications in 5G networks. *Entropy* 20, 1 (2017), 4.

[26] M. T. Saqib and M. A. Hamid. 2016. FogR: A highly reliable and intelligent computation offloading on the Internet of Things. In *2016 IEEE Region 10 Conference (TENCON)*. 1039–1042.

[27] Subhadeep Sarkar and Sudip Misra. 2016. Theoretical modelling of fog computing: A green computing paradigm to support IoT applications. *Iet Networks* 5, 2 (2016), 23–29.

[28] W. Wang, H. Chen, and X. Chen. 2012. An Availability-Aware Virtual Machine Placement Approach for Dynamic Scaling of Cloud Applications. In *2012 9th International Conference on Ubiquitous Intelligence and Computing and 9th International Conference on Autonomic and Trusted Computing*. 509–516.

[29] T. Wiss and S. Forsström. 2017. Feasibility and performance evaluation of SCTP for the industrial internet of things. In *IECON 2017 - 43rd Annual Conference of the IEEE Industrial Electronics Society*. 6101–6106.

[30] Y. Xiao, Z. Ren, H. Zhang, C. Chen, and C. Shi. 2017. A novel task allocation for maximizing reliability considering fault-tolerant in VANET real time systems. In *2017 IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*. 1–7.

[31] Wai-Leong Yeow, Cédric Westphal, and Ulaş Kozat. 2010. Designing and embedding reliable virtual infrastructures. In *Proceedings of the second ACM SIGCOMM workshop on Virtualized infrastructure systems and architectures*. ACM, 33–40.

[32] Xiao Yuan, Chimay J. Anumba, and M. Kevin Parfitt. 2016. Cyber-physical systems for temporary structure monitoring. *Automation in Construction* 66 (2016), 1 – 14.

[33] B. Zhou, A. V. Dastjerdi, R. N. Calheiros, S. N. Srirama, and R. Buyya. 2017. mCloud: A Context-Aware Offloading Framework for Heterogeneous Mobile Cloud. *IEEE Transactions on Services Computing* 10, 5 (Sep. 2017), 797–810.

Towards a Dynamic Replication of Messages in a Network with Flexible Real-Time Guarantees

Alberto Ballesteros
DMI, Universitat Illes Balears
Palma de Mallorca, Spain
a.ballesteros@uib.es

Manuel Barranco
DMI, Universitat Illes Balears
Palma de Mallorca, Spain
manuel.barranco@uib.es

Sergi Arguimbau
DMI, Universitat Illes Balears
Palma de Mallorca, Spain
sergi.arguimbau@uib.es

Marc Costa
Universitat Illes Balears
Palma de Mallorca, Spain

Julián Proenza
DMI, Universitat Illes Balears
Palma de Mallorca, Spain
julian.proenza@uib.es

ABSTRACT

Distributed Embedded Systems (DES) typically have real-time and dependability requirements. Moreover, if they have to operate in dynamic operational contexts, they need to be adaptive. That is, they must be able to automatically and autonomously rearrange in response to changes. In order for a DES to be adaptive, its underlying subsystems must be flexible. The implementation of the flexibility, just like the implementation of the real-time and dependability, cannot be done in an orthogonal manner since it entails the collaboration of various subcomponents at different levels of the architecture. The DFT4FTT project proposes a self-reconfigurable infrastructure for implementing DES with real-time, reliability and adaptivity requirements. One of the most relevant fault tolerance mechanisms is the dynamic replication of messages, that makes it possible to tolerate transient faults affecting the network. In this paper we describe more in-detail the design and implementation of this mechanism.

1. INTRODUCTION

Modern Distributed Embedded Systems (DES) operate under dynamic *operational contexts*. On the one hand, the *operational requirements* - which include what the system has to do (its functionality), the real-time guarantees it has to provide, and the reliability it has to exhibit - can change in an unpredictable manner. On the other hand, the *operational conditions*, that is the circumstances under which the system has to operate can also change in an unpredictable manner. Note that the operational conditions include changes in the environment and changes in the system itself, due to faults. This kind of DESs must be able to rearrange to adequate to the new operational context, while maintaining its real-time and dependability behaviour. That is, this kind of DESs must be able to *adapt*. Adaptivity can be achieved following the next steps: monitor the environment and the system itself; determine when a relevant event has happened; decide on an counteraction to address the effects of this event; and, finally, carry out said action. Furthermore, these abilities must be carried out automatically and autonomously. Some examples of adaptive DES are: autonomous vehicles, machinery in a smart factory and self-repairable devices.

Adaptivity is an interesting property and, in the scope of this work, we explore it to achieve two different goals. In general, adaptivity can be used to construct a more efficient DES. This is because a DES that is able to dynamically change the assignment of computational and communication resources to the different functionalities does not need to be dimensioned for the worst case scenario. Moreover, adaptivity is also appealing from a dependability perspective. The reason is that it allows the implementation of dynamic fault tolerance mechanisms, which are more effective than the static ones. An example is the ability recover a faulty node replica, thus, maintaining the level of reliability of the DES during a longer period of time. Another example is the ability change the fault tolerance strategy at runtime depending on the operational requirements of the DES.

To properly construct a DES that is adaptive, it is necessary that the underlying subsystems are *flexible* enough to support the desired adaptive functionality. However, the implementation of the flexibility cannot be done in an orthogonal manner since it entails the collaboration of various subcomponents at different levels of the architecture. For instance, if we want the DES to be able to change its operation at runtime, it must implement mechanisms allowing it to load/unload tasks into/from the nodes, as well as register and unregister their associated communications. Consequently, an holistic approach must be followed, that is, the system must be considered as a whole and implement specific services at different levels of the architecture to provide such flexibility.

Note that this lack of orthogonality does not only affect the implementation of the flexibility, but also the implementation of the real-time and the fault tolerance mechanisms. Consequently, DESs with real-time, reliability and adaptivity requirements must be constructed in the form of what we call a *complete infrastructure*, that is, as a set of interrelated hardware and software components (the architecture) together with a set of in-built mechanisms that make it possible to fulfil all these requirements, both at the node and at the network level.

To support real-time highly-reliable adaptive DESs, the *Dynamic Fault Tolerance for the Flexible Time-Triggered Ethernet* (DFT4FTT) project [3] proposes a complete infrastructure with advanced fault-tolerance capabilities while taking into account the above-mentioned aspects.

At the node level, DFT4FTT provides high reliability by means of active replication with majority voting. That is,

each critical task is executed in parallel in several nodes of the DES, which we call *Computational Nodes* (CNs). To provide flexibility at this level, we proposed a centralized architecture in which a so-called *Node Manager* (NM) can reconfigure at runtime the allocation and replication of tasks into the CNs [3].

At the network level, we have designed DFT4FTT to rely on the Flexible Time-Triggered Replicated Start (FTTRS) [5], a switched-Ethernet implementation of the Flexible Time-Triggered (FTT) communication paradigm. FTT makes it possible for the nodes of a DES to exchange traffic with real-time guarantees. Moreover, FTT provides *full flexibility* in the communications, that is, on the one hand, it supports the exchange of periodic as well as aperiodic traffic with different real-time requirements and, on the other hand, it allow to change the real-time requirements of the traffic at runtime. High reliability is achieved by means of fault tolerance. Specifically, permanent network faults are tolerated by replicating the network, while transient faults are tolerated by proactively retransmitting the critical messages.

Note however that, so far, the temporal replication of messages in DF4FTT was static. Such static replication can be inefficient, or even ineffective, when facing the changing operational conditions in which adaptable systems operate. To overcome this issue, in the present paper we propose how to make the proactive retransmission mechanism of DFT4FTT dynamic. Specifically, we propose to change the number of message replicas to be sent at runtime (referred to as k hereafter) depending on the current operational context.

It is important to highlight that, although all this work has been developed in the scope of the DFT4FTT project, the ideas here presented are quite generic and, thus, they can be applied in different communication subsystems.

The rest of the document is organized as follows. First we discuss the related work and introduce the main features of DFT4FTT. Second, we explain how the system can detect the need for changing k from a system-wide perspective, the guidelines for dynamically determining the proper value of k, and how to consistently and reliably propagate any change to all the nodes. Third we describe a partial implementation and a set of experiments that demonstrate the feasibility of the ideas presented here. Finally, we summarize the paper's contributions and point out future work.

2. RELATED WORK

During the last decades several architectures have been proposed for providing real-time and fault-tolerant services for the execution and/or the communication of tasks in distributed systems, e.g. MAFT,/FTP-AP, Delta-4, GUARDS, EMC, DREAMS [1]. Some of them like Delta-4 and DREAMS do even provide services to reallocate and reschedule tasks at runtime. Nevertheless, some of these architectures require complex communication protocols at the application or transport layer to provide node fault tolerance, others require costly adhoc network topologies, while others are generic architectures that do not provide any specific strategy for replicating the nodes or the network.

On the other hand, several Ethernet protocols do provide some real-time and/or fault-tolerance properties. Some of the newer ones such as PRP, AFDX, TTEthernet and specific TSN standards can even provide zero recovery times by means of spatial redundancy, e.g. TSN's IEEE 802.1CB. However, none of these protocols provide temporal replication of messages to efficiently tolerate transient link faults, i.e. at most, they use proactive message replication to send critical messages through the available redundant paths so as to tolerate permanent link faults. Moreover, they either do not support online rescheduling or do imply a rescheduling latency that is not adequate to timely react to critical situations requiring a fast reconfiguration [5].

3. OVERVIEW OF DFT4FTT

As already introduced, DFT4FTT relies on FTTRS [5] to implement the communication subsystem. In FTTRS the regular nodes of the DES are called *slaves* and they are interconnected by means a duplicated star. As seen in Fig. 1, in DFT4FTT the slaves correspond to the CNs. The communications among slaves is managed by the Node Manager (NM), which embeds an FTTRS *master*.

The master organizes the communication in fixed-duration slots called *Elementary Cycles* (ECs). In turn, each EC is divided in several windows: Trigger Message Window (TMW), Synchronous Window (SW) and Asynchronous Window (AW). The EC starts with the master transmitting the so-called *Trigger Message* (TM). The purpose of this message is twofold: it notifies the slaves about the start of a new EC and it polls the set of periodic messages that they have to transmit during that EC. Then, during the SW, the slaves transmit the corresponding periodic messages, as instructed by the TM. Finally, during the AW, the slaves transmit their pending aperiodic messages.

FTTRS follows a publisher-subscriber communication scheme. Every slave willing to transmit must publish messages through a dedicated logical virtual communication channel called *stream*. Each stream has a set of attributes that characterize the properties of the traffic it conveys. Some examples of real-time attributes are the size of the messages, the deadline and, in case of a periodic traffic, the periodicity. In turn, slaves willing to receive must subscribe to the corresponding streams.

Regarding the tolerance to faults affecting the network, note that transient network faults are typically tolerated using Automatic Repeat Request (ARQ). That is, when a message is lost, the receiver notifies the transmitter that the message was not received and, then, the transmitter retransmits said message. Note that this approach is efficient in terms of bandwidth but it introduces a non-significant delay that penalizes the real-time response of the network [5]. That is why proactive retransmission is more suitable in real-time systems, in general, and in DFT4FTT, in particular. Specifically, publishing nodes transmit each critical message k times in advance to maximize the probability of the subscribers to receive at least one message replica. Note that k is a new attribute of the streams and that, so far, it has been static in DFT4FTT.

At the node level, each *functionality* in the system is im-

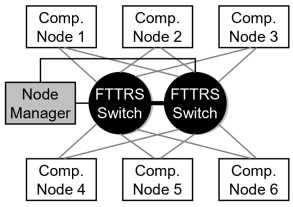


Figure 1: System architecture.

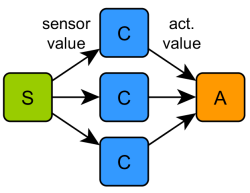


Figure 2: App example.

plemented by means of an *application*. An application is composed of several interconnected tasks that can be executed in parallel or sequential manner. In turn, a task is the minimum unit of computation and can be deployed in any CN. As an example, Fig. 2 shows a control application composed of three tasks connected in sequence: a sensing task (S); a triplicated control task (C), where each one of its replicas executes in parallel; and an actuation task (A).

Note that an application is executed as a set of sequential phases involving execution of tasks and transmission of messages. Thus, a given *system configuration* basically includes the allocation and replication of tasks, the number of proactive retransmission of critical messages (k), and the schedule of tasks and messages.

At runtime, the NM is the responsible for detecting when a new configuration is required, find a new configuration that fulfills all the operational requirements (functionality, real-time guarantees and reliability), and propagate the new configuration to al the CNs.

4. DYNAMIC MESSAGE REPLICATION

Making dynamic the proactive retransmission of messages basically consists in (1) detecting when the operational context changes; (2) determining, accordingly, on the specific number of message replicas (k) for each type of message (and stream); and, finally, (3) propagating the new replication parameters to the nodes of the DES. While all these three steps are covered next in subsections 4.2, 4.3 and 4.4, let us first introduce the two general policies that can be followed to address them.

4.1 Dynamic replication policy

Although the fault tolerance mechanisms of a DESs are not orthogonal to each other and to the other architecture mechanisms, the NM can still follow two main policies to temporally replicate messages in a dynamic manner, namely the *network-level policy* and the *system-level policy*.

According to the *network-level policy*, the NM would make its decisions on how to temporally replicate messages taking into account the operational context exclusively from a network-level perspective. In other words, given a set of streams and the level of criticality of each one of them, the NM would take into account how many replicas of each message of said streams are being proactively retransmitted, what is the number of available links (i.e. the available link spatial redundancy), and how the environment is affecting the quality of the links (e.g. their BER). In particular, the NM would set k for each stream to ensure, with a high probability, that at least one message replica of each critical stream reaches all the destination nodes. Note that there are several strategies to decide on k, as proposed in [2]. The NM can consider that the quality of the links used to transmit a given stream is the worst (maximum) probability with which messages are being lost in any of them and, thus, transmit the stream using (the same) conservative value of k in each one of those links. Alternatively, the NM can discriminate the quality of each link and, thus, use a different value of k in each one of them [2].

In contrast, following the *system-level policy* the NM would make its decisions considering the operational context from the perspective of the whole system, i.e. considering the available (spatial) redundancy at multiple levels of the DFT4FTT system architecture. This would result in a more efficient message replication, since the DFT4FTT's fault-tolerance mechanisms of these additional redundancy levels can indirectly tolerate transient faults affecting the messages, without having to ensure that at least one message replica reaches the destination in a link basis. The main fault-tolerance mechanisms DFT4FTT includes at its other levels are: (1) the replication of critical tasks (in different nodes), so that each critical data message is transmitted several times (once per task replica) even if each task replica is configured to send only one copy of that message; (2) the *Distributed Consistent Majority Voting* (DCMV) [4], which allows replicated tasks to achieve a consensus thereby tolerating faults affecting not only task replicas themselves, but also their ability to transmit/receive critical data messages; (3) the *Cc-vector Exchange Protocol* (CVEP) [4], which allows tolerating bursts by proactively retransmitting several replicas of each critical data message in several consecutive ECs; (4) the reintegrating mechanisms proposed in [4], which allow recovering transiently-faulty nodes; and (5) the duplication of the network, which provides up to 4 partially-redundant physical paths between each pair of nodes [5].

We have decided to follow the system-level policy to design and implement the dynamic replication of messages. We believe that this policy is better, since it can exploit the different DFT4FTT's redundancy levels to provide the necessary reliability in a more cost-effective manner. This does not mean, however, that other redundancy levels could substitute the message replication. Although other levels can indirectly improve the tolerance to faults affecting the transmission of messages, it is important to deal with each fault as close as possible to its point of origin both in the space and the time domains. In general, if the errors generated by a fault propagate to a higher level, they can provoke faults at that level that, then, will manifest in manners that are more severe and difficult to handle.

4.2 Detection of the need for changes

In this section we outline which changes of the operational context make it necessary to update the number of message replicas to be proactively retransmitted, i.e. k. Then, we explain which information can be monitored to detect these changes.

Since the system must be (re)configured in an holistic manner, the explanation of what changes trigger the update of k must be done from a system-wide perspective. At system's start-up the NM sets k to a conservative value, considering the different levels of redundancy of DFT4FTT, but assuming that the network has to deal with the harshest environment in which the system is going to operate. Then, the NM decides to dynamically reconfigure the system, including k, when it encounters the following situations. First, if the environment actually becomes more benign, then the NM can reduce k to save energy, or to ease future configurations in which it could need to fit new streams (e.g. if new tasks are put into execution). In any case, k should always be conservative enough to prevent the system from failing while, in the future, it carries out reconfiguration actions to increase k again so as to deal with an increasingly harshly environment. In this later case note that, if the network has no available-enough bandwidth to increase k (e.g. if it had to accommodate new tasks and their streams), then the NM would need to reconfigure the redundancy at the other levels. This reconfiguration can consist in evicting non-critical tasks

(or even reduce the number of task replicas), and thus their streams, to free bandwidth so as to accommodate higher values of k. In any case, the NM has to find a configuration in which the redundancy degree at the different levels of its architecture guarantee, as a whole, the desired system reliability. Second, if the system loses (spatial) redundancy at any of its other levels due to faults - e.g. if a node, task, or link fails -, then the NM should increase k taking into account the just-mentioned considerations about the available bandwidth. Analogously, if the system regains redundancy thanks to its reintegration mechanisms, then the NM can conservatively reduce k as explained before. Third, if the operational requirements (functionality, RT guarantees, or reliability) of the system change, then the NM may need to change the set of tasks to be executed (and with them their interdependencies and streams). If so, it will need to find a new configuration with an adequate redundancy degree in each level of its architecture, including again a conservative value for k.

It is important to note that it may be impossible to find a new configuration that fulfills all the requirements when the operational context changes. If so, the NM needs to find a new configuration in which the system provides its services with an adequate/acceptable level of reliability (even though it is in a degraded manner from the functional point of view).

Next we outline the mechanisms DFT4FTT can use to monitor the environment and the system itself, so as to detect the just-described situations that require a system reconfiguration.

For detecting changes in the environment that affect the network, the NM can use different mechanisms. First, several radiation sensors could be placed in the system from which the NM could measure how harshly the environment is and, thus, estimate the expected rate of transient link faults. Second, the NM can also estimate the rate of transient faults affecting each link through which it receives messages, by using the counter of dropped incoming messages (due to errors in the channel) provided by the Ethernet card it uses to communicate through that link. Third, in DFT4FTT each CN periodically transmits to the NM an *I Am Alive* (IAA) message which piggybacks information contained in the TM. The NM can use the percentage of IAA omissions from a given CN to estimate the probability with which the transmitted TM does not reach that CN and, thus, the rate of transient faults affecting the link through which it transmits messages to that CN. Finally, the CVEP retransmission mechanism mentioned in Sec. 4.1 requires task replicas to send ACK messages [4], which then are used by the switch to determine which critical data messages each task replica was able to transmit/receive. The NM can use this information to estimate the rate of transient link faults as well.

To detect changes in the available spatial redundancy, the NM can use the just-mentioned mechanisms (except the first one). For instance, if a CN omits its expected IAA during a relatively long period of time, the NM will diagnose that CN as permanently faulty. Conversely, if that CN reintegrates, and from then on successfully transmits its IAA, then the NM will detect that the CN is available again.

Finally, for detecting changes in the operational requirements, the NM includes application-dependant knowledge (codified at design time) about both the environment and how the system should operate accordingly [3]. For instance, in an autonomous vehicle, the NM can use this knowledge to detect when the terrain changes and, then, determine what are the new operational requirements to adequately drive.

4.3 Determination of the configuration

To determine a new system configuration we propose that the NM carries out a search among all the possible configurations to find a valid one, i.e. one that fulfills all the operational requirements. Some of the search techniques we are considering for this purpose are: heuristic-based techniques like branch and bound with a greedy algorithm, metaheuristic-based techniques like Tabu search [6] and solvers like SMT solvers [8].

In particular, for a given configuration to be considered as valid, it must fulfill the real-time requirements of the tasks and messages, as well as the reliability requirements. Thus, the search technique must contain both an holistic scheduler analyzer and a reliability analyzer. In any case, the search process can require more or less computation time and storage capacity. The number of possible configurations can be huge depending on the number of aspects considered in the search, e.g. the allocation and replication of tasks, k, etc. Moreover, the scheduling and reliability analyses of each configuration can also take a non-negligible amount of time. Thus, we are assessing the performance of the above-mentioned search techniques so as to decide whether the search should be carried out at runtime or completely/partially pre-calculated offline.

Independently of the tool(s) the NM will finally use, next we outline the strategy we propose to decide how many replicas should be proactively retransmitted for each type of message. In this sense, we differentiate among *data messages*, the *Trigger Message* (TM) and *control messages*.

Data messages are used by CNs to transmit the application-level data among tasks. Of those messages, we propose to temporally replicate only the ones that are considered as critical, e.g. the ones that replicated tasks use to exchange the information they need to reliably vote on. To reduce the complexity of finding an adequate system configuration, the NM must calculate a conservative value of k that is common to all critical data messages. Note however, that this value should not be calculated assuming the transient failure rate of the link that is encountering more errors. This is so because the cause that leads a link to be specially error prone is not necessary a harshly environment (that would affect the other links in a similar manner), but it can also be a local mechanical/electrical defect of that link. Therefore, the calculation of the common value of k should be based on the radiation sensors' measurements or on a trimmed mean of the transient failure rates estimated for each link. Then, if a given link shows to be more error prone than expected, the NM should find a new configuration with a higher value of k for that specific link. In any case, the fault diagnosis mechanisms of DFT4FTT should be good enough to diagnose a specially error-prone link as permanently faulty and, then, to discard it from the available spatial redundancy of the network.

As explained above, the TM plays a key role in the operation of the DF4FTT communication subsystem. Consequently, it is vital to transmit this message in a reliable manner. With that being said, note that we already constructed a model of FTTRS and made a sensitivity analysis to determine the impact of the replication of the TM in the system reliability [4]. The results showed that the system

reliability improves as the k of the TM increases; but that this improvement starts to become negligible when k is increased from 3 to 4. Moreover, the difference in bandwidth usage when using 2 or 4 TM replicas is also negligible. Thus, it makes no sense to add complexity to the system trying to dynamically replicate the TM if the gain in bandwidth does not justify it. Consequently, we propose to set the number of the TM replicas to a fixed value of $k = 4$.

Control messages are the messages used by both the NM and the CNs to help in managing the operation of the system. Some examples are the *Master Command Message* (MCM) sent by the NM as later described in Sec. 4.4, and the IAA message CNs (see Sec. 4.2). Which value of k is the appropriate one depends on both the criticality of the specific control message and how frequently it is transmitted. We propose to replicate only the control messages that are critical. In principle critical control messages should be dynamically replicated as data messages are. However, if the difference in the bandwidth used by a critical control message is negligible when comparing $k = 2$ with $k = 4$, then we propose to consider a fixed value of $k = 4$ for it, so as to reduce the need for reconfigurations. For instance, consider the case of the MCM. As it will be explained, the NM uses the MCM to consistently propagate the changes on the system configuration to the CNs and, thus, it is as critical as the TM. However, the MCM is rarely transmitted since reconfigurations are not expected to be frequent. Thus, we propose to use a fixed value of $k = 4$ for this control message.

4.4 Propagation of the configuration

Once the NM has decided on a new configuration (including the new values of k), it is necessary to propagate the information about that configuration to the CNs. In particular, the new values of k are then used by the CNs, in the transmission, to issue the correct number of message replicas and, in the reception, to check that the correct number of replicas has been received. Moreover, this propagation has to be done consistently in both the NM and all the CNs. This means that the NM and the CNs have to update their internal databases with the same configuration information at an equivalent time.

As said in Sec. 3, k is an attribute of the stream. Thus, for the particular case of the propagation of the new values of k, DFT4FTT could rely on the mechanism FTTRS already provides to consistently update any stream attribute [5]. However, this mechanism has two important shortcomings. First, it cannot propagate configuration information regarding aspects other than the stream-related ones. Second, even propagating only the different values of k of a given system configuration would require the transmission of many individual and replicated control messages, thereby limiting the real-time response of the propagation itself. Therefore, here we propose a new mechanism for propagating a whole configuration in a more efficient manner. In any case, since the focus of this paper is on the communication, this mechanism will be described in terms of the databases used to store the stream attributes. However, everything said here can be applied to the other databases.

First of all, the stream database of the NM and the CNs is composed of two databases: (1) the *read database*, which is used in the normal operation to consult the attributes of the streams; and (2) the *write database*, which keeps all the system changes pending to be applied.

Once the NM has introduced all the changes of a new configuration into its write database, it sends the full list of changes to all the CNs by means of the Master Command Message (MCM), which is broadcast as an aperiodic control message. Moreover, as already mentioned, the MCM is proactively retransmitted several times to ensure that it is received by all the CNs and, thus, that there is no data inconsistency.

Upon the reception of the MCM, each CN updates its write database with the content of said message. After that, the NM decides in which EC the read databases should be updated with respect to the write databases. Then, the NM sends a commit order inside the TM of said EC. This order indicates that the NM is going to update its read database at the end of the EC and, thus, instructs each CN to do so with its own read database. Specifically, the NM and the CNs update their read database in a new dedicated window placed at the end of the EC, we call the *Commit Window*. Neither the NM nor the CNs are operating with their databases during this window and, thus, they can update the read database without any risk.

5. EXPERIMENTATION

Since DFT4FTT is an ongoing project, its implementation is subjected to the fully definition of its internal mechanisms. In this sense, all the work herein discussed has been implemented; except the search of configurations within the NM, which as already explained is responsible for determining the value of k for each stream. Specifically, in the context of the work herein presented (the dynamic replication of k), we have implemented the mechanisms that make it possible, for each kind of message (and stream), to: generate k message replicas, modify k at runtime, and propagate the new k to the CNs. Moreover, we have tested the correct operation of all these mechanisms in conjunction by switching between configurations where k varies for all kind of messages (and streams). The results of these experiments demonstrated the feasibility of the work herein proposed.

Apart from that, these experiments allowed us to obtain evidences to support some of the decisions taken during the design process. In particular note that, from an intuitive point of view, the quality of the synchronization between the NM and the CNs should decrease as the value of k for the TM increases. Thus, we carried out an experiment to check that the decision of not dynamically replicating the TM and consider a constant value of $k = 4$ for it does not compromise this quality.

Fig. 3 depicts the testbed for this synchronization-related experiment. The NM periodically transmits the TMs to two CNs. The CNs, when receiving the TM replicas, calculate when to trigger the start of the Synchronous Window (SW) (Sec.3). In-between the NM and the CNs we placed a NetAnalyzer [7], which is a dedicated piece of hardware that timestamps messages with high time resolution. The data collected by the NetAnalyzer helps to characterize the transmission of TMs and, thus, rule out that a potential lack of synchronization is due to jitter in this transmission. The synchronization quality is measured by a microcontroller which, by means of GPIOs connected to the NM and the CNs, timestamps the start of each EC and its corresponding SW as seen by each CN. The synchronization quality for each EC is calculated as the difference of time between the beginning of the SW in each one of the two CNs.

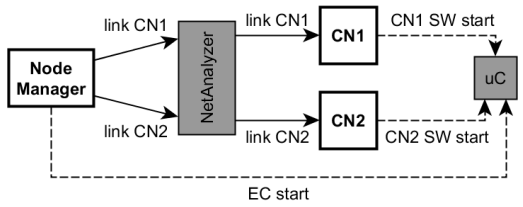


Figure 3: Testbed for the experimentation.

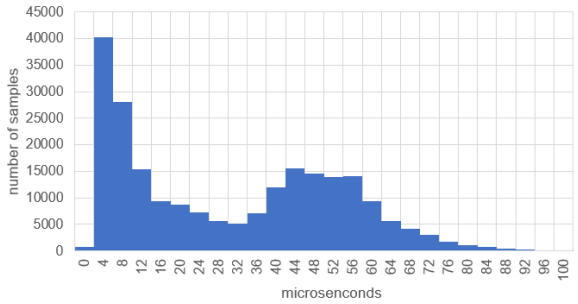


Figure 4: Histogram of the SW delay in both CNs for $k = 4$.

To properly carry out this experiment we conducted multiple tests. Specifically, we conducted tests for 2, 3 and 4 TM replicas. In each one of these three cases we injected all possible error scenarios affecting the reception of the TM replicas, except the error scenario in which all the TM replicas are corrupted. Moreover, each error scenario is repeated 1000 times to have multiple samples of each one of them.

Regarding the transmission of the TM replicas, we measured the delay and jitter of the time between the transmission of: (1) the first TM replica sent in each pair of consecutive ECs; (2) the same TM replica sent to CN1 and CN2; (3) each pair of consecutive TM replicas sent in the same link. The results of the tests showed that the behaviour of the NM, in terms of predictability, is quite good. The periodicity with which the NM transmits the TMs, the delay in the transmission of the same TM replica to the two CNs, and the separation between TM replicas in the same link are quite constant; independently of the value of k and the injected faults. However, it should be noted that we noticed a small number of outliers. We believe this is because the tests were carried out using a software implementation of the FTTRS switch/master and, thus, this issue should disappear in a final hardware implementation.

As concerns the results in the reception, i.e. the quality of the synchronization of the CNs, we did not notice any relevant loss of synchronization; neither due to the TM's k nor due to the injected faults. The results for $k = 4$ are summarized in Fig. 4. Specifically, this figure shows the delay with which CNs have determined the beginning of the SW in each EC, grouped from 0 to 100 us.

A value of zero would indicate a perfect synchronization. Although the synchronization is not perfect, most of the samples are between 4 and 60 us; which is reasonably good, tanking into account that we are using the just-mentioned software implementation. Again some outliers, which are not shown in the figure to better fit the page layout, are present. However in a future hardware implementation the quality of synchronization is expected to drastically improve.

6. CONCLUSIONS

In this paper we present the design and partial implementation of the dynamic message replication mechanism of the DFT4FTT architecture. This mechanism has the potential to increase DESs tolerance to transient faults affecting the network in a cost-effective manner; by taking advantage not only from the temporal redundancy of messages, but also from the redundancy of the different levels of the architecture.

We explained how the system can detect the necessity for reconfiguring itself so as to adequately tolerate transient faults; the guidelines for how to decide on a proper number of replicas for each kind of message in any new configuration; and how to consistently and reliably propagate such a new configuration to all the nodes of the DES. Then, we outlined the implementation we have done of the ideas presented here in our ongoing implementation of DF4FTT; as well as the tests we carried out to demonstrate their feasibility. Moreover, we described an exhaustive experiment through which we corroborated that the replication we propose for the TM both, tolerates faults and does not negatively impact the synchronization of the nodes.

In the short term we will propose how to use an adequate search technique to find proper system reconfigurations by taking into account, among other aspects, the degree of temporal redundancy of messages (k).

7. ACKNOWLEDGMENTS

This work was supported by project TEC2015-70313-R (Spanish *Ministerio de economía y competividad*) and by FEDER funding.

8. REFERENCES

[1] I. Álvarez, A. Ballesteros, M. Barranco, D. Gessner, S. Derasevic, and J. Proenza. Fault Tolerance in Highly-Reliable Ethernet-based Industrial Systems. In *Proceedings of the IEEE (Early Access)*, 2019.

[2] I. Álvarez, M. Barranco, and J. Proenza. Mixing Time and Spatial Redundancy over Time Sensitive Networking. In *Proc. 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, Luxemburg, 2018.

[3] A. Ballesteros, J. Proenza, and P. Palmer. Towards a Dynamic Task Allocation Scheme for Highly-Reliable Adaptive Distributed Embedded Systems. In *Proc. 22th IEEE Int. Conf. on Emerging Tech. and Factory Autom. (ETFA)*, Limassol, 2017.

[4] S. Derasevic. *Node Fault Tolerance for Distributed Embedded Systems based on FTT-Ethernet*. PhD thesis, University of the Balearic Islands, 2018.

[5] D. Gessner, J. Proenza, M. Barranco, and A. Ballesteros. A fault-tolerant ethernet for hard real-time adaptive systems. *IEEE Transactions on Industrial Informatics*, 15(5):2980–2991, May 2019.

[6] F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, 13(5):533 – 549, 1986.

[7] Hilscher. NetAnalyzer.

[8] W. Steiner. An Evaluation of SMT-Based Schedule Synthesis for Time-Triggered Multi-hop Networks. In *31st IEEE Real-Time Systems Symposium*, pages 375–384, Nov 2010.

Routing Heuristics for Load-balanced Transmission in TSN-Based Networks

Mubarak Adetunji Ojewale
mkaoe@isep.ipp.pt

Patrick Meumeu Yomsi
pmy@isep.ipp.pt

CISTER Research Centre, ISEP
Polytechnic Institute of Porto, Portugal

ABSTRACT

A carefully designed routing synthesis can help system designers achieve a better load balancing in TSN-based networks and avoid congestion. To this end purpose, this work proposes two heuristics referred to as (1) LB-DRR, which aims at achieving a better load balancing and compute as much disjoint routing paths as possible for each replicated flow; and (2) CR-DRR, which recomputes paths for time-sensitive flows in congestion situations. Extensive simulations demonstrate that the proposed approach outperforms the classical Shortest Path (SPA) and the weighted Equal Cost Multi-path (wt-ECMP) algorithms in terms of the maximum load transmitted on a link by more than 70% and 20%, respectively.

Keywords

Time Sensitive Networking; Routing Algorithms; Congestion; Load-Balancing

1. INTRODUCTION

Ethernet in its original specification was not designed with real-time communication in mind. The IEEE Time Sensitive Networking (TSN) Task Group [7] acknowledged this fact and has been investing considerable workforce to come up with a set of new standards to address this limitation. In this context, the group has designed sophisticated mechanisms to achieve temporally predictable and reliable transmission of packets over switched Ethernet networks. Specifically, key features like flow-synchronization; -management; -control; and -integrity, have been instanced. For a given network, deriving an efficient and cost-effective flow control scheme is paramount. It would make it possible for users and operators to centrally and dynamically discover; configure; monitor; and report on the capabilities of switches and end-stations (a.k.a. nodes) [9]. In a nutshell, the TSN flow control mechanism can be considered from two perspectives: (1) the scheduling (i.e., when each flow shall be transmitted); and (2) the routing (i.e., on which path each flow shall be transmitted). Dürr et al. [2] demonstrated that the scheduling problem of real-time (a.k.a. time-sensitive) flows can be reduced to the No-Wait Job Shop Problem (NW-JSP), which is NP-Hard. On another front, Wang and Crowcroft [14] proved that any routing problem that is subject to two or more independent additive or multiplicative tree constraints is NP-Hard. This is the case for time-sensitive flows, unfortunately. They are subject to timing, bandwidth, cost and reliability constraints. Consequently, seeking for an exact solution is very challenging and computationally expensive. Designing efficient heuristics is the only viable alternative.

In recent years, the scheduling problem has received significantly more attention by the research community than the routing. However, Nayak et al. [10], Singh [12], and Gavriluţ et al. [4] among others raised voices and stressed on the importance of routing in achieving low latency, predictability, and reduced architecture cost. In this work, we follow the same path and focus on the routing problem of TSN flows as an improper routing strategy may increase the number of transmission operations, thereby incurring additional delay. Also, it may increase the blocking time of flows in the network if too many flows try to simultaneously traverse the same path. We believe that a strategy that minimizes the number of transmission operations and the blocking times suffered by each flow would help get around and/or mitigate these situations.

▷ **Limitations of the state-of-the-art.** The TSN standard on path control and reservation [8] recommends the Constrained Shortest Path First (CSPF) routing scheme for the transmission of time-sensitive flows (see page 71). It dictates that this scheme

> "*essentially performs shortest path routing on the topology that only contains the links meeting the constraint(s).*"

From this quote, it follows that CSPF is similar to the Shortest Path Algorithm (SPA) in its operation. Consequently, it is also exposed to congestion and increased blocking time for flows. To illustrate this claim, let us consider the network topology in Figure 1, where six nodes (Node 1 to Node 6) and six switches (S_1 to S_6) are connected by full duplex links. Nodes communicate through flow transmissions over the links and switches. In this example, we consider three flows – flow $f1$ (green) is transmitted from Node 1 to Node 6; $f2$ (yellow) from Node 2 to Node 5; and finally, flow $f3$ (brown) is transmitted from Node 4 to Node 4. We assume that the CSPF routing policy is adopted and all valid paths from each source to each destination node allows each flow to satisfy its end-to-end timing requirement. Then, all these flows are transmitted via the "direct link" (in red) between S_1 and S_6, thus increasing the eventual blocking time over this link for each flow. This state of facts

Copyright retained by the authors.
RTN'2019, Stuttgart, Germany

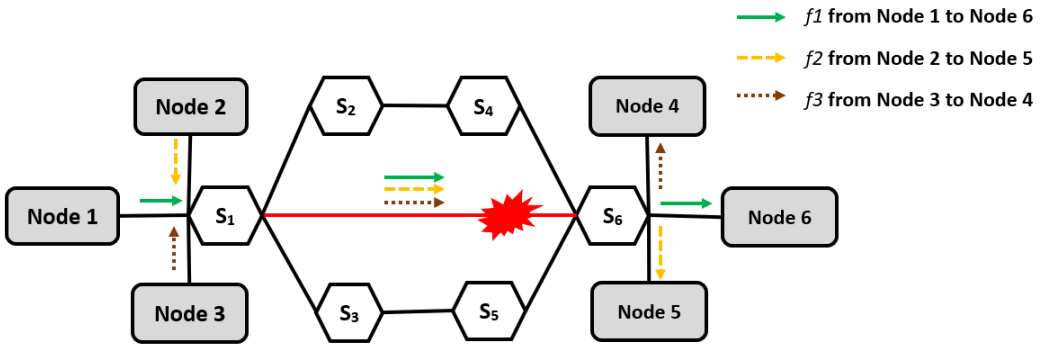


Figure 1: Congestion under CSPF routing policy.

makes this link the potential single point of failure of the network and may cause congestion despite the high level of connectivity. The same limitation applies to the Equal Cost Multi-Path (ECMP) and the weighted ECMP (wt-ECMP) routing schemes [12], unfortunately. The basic idea of these two routing schemes is as follows. Under ECMP, instead of computing a single shortest route like this is the case with SPA, multiple shortest routes are computed and from these, one or several routes are selected arbitrarily. The wt-ECMP scheme distinguishes itself from ECMP only in the selection mechanism. Here, for all the computed shortest routes, a "weight" is assigned to each route to make sure that selection is not performed in an arbitrary manner.

▷ **Our contribution.** To get around the aforementioned hurdles and to fully take advantage of the network connectivity, we suggest the adoption of a routing strategy that ensures load balancing, i.e., a strategy that distributes the transmission operations among the links as even as possible. In addition, this approach ensures that no link becomes the only potential point of failure of the network. In this scope, this paper proposes two heuristics, referred to as LB-DRR and CR-DRR, with the following objectives:

- ▷ LB-DRR: which aims at finding a feasible route for each flow so that the traffic on each link is minimized[1].
- ▷ CR-DRR: which aims at computing alternative routes for each flow in a situation of congestion at run-time.

Although the proposed routing schemes are motivated by the limitations observed in the specifications of TSN, they can be ported (with minor efforts) to a large portion of real-time Ethernet networks.

▷ **Paper organization.** The rest of this paper is structured as follows. Section 2 presents the model of computation and introduces the notations adopted in this work. Our proposed heuristics (LB-DRR and CR-DRR) are detailed in Section 3. Section 4 reports on the experiments carried out and discussions about these. Section 5 discusses the related works on the topic in the literature. Finally, Section 6 concludes the paper and provides future research directions.

2. MODEL OF COMPUTATION

In this section, we define the network topology and the flows specification assumed throughout this paper. Also, we introduce the notations and parameters necessary for a good and crystal clear understanding of our proposed heuristics.

▷ **Network topology specification.** We modelled the network as an undirected graph $G = (V, E)$, where the set $V = N \cup S$ of vertices in G is composed of a finite set N of nodes and S of switches (see Figure 1 for an example). The vertices are connected by a set E of full duplex links or edges. This means that each edge $e \in E$ is defined by a couple $(v_1, v_2) \in V \times V$ of two connected nodes.

▷ **Flow specification.** By default, every TSN-based network addresses *recurrent* (periodic and/or sporadic) flows grouped in classes (e.g., CDT, Audio/Video, etc.). These flows are transmitted within so-called cycles[2] and within a cycle, each flow is treated individually (irrespective of its period) [10]. When all flows are released simultaneously (as assumed in this work), we can safely restrict our attention to a single cycle (the first one). As such, we consider a set of n *aperiodic* time-sensitive flows $F \stackrel{\text{def}}{=} \{f_1, f_2, \ldots, f_n\}$. Each flow $f_i \stackrel{\text{def}}{=} (src_i, dst_i, rep_i, C_i, T_i, D_i) \in F$ is characterized by a 5-tuple, where: (1) src_i is the source node; (2) dst_i is the destination node; (3) rep_i is the replication level (i.e., the number of replicates of f_i allowed to be transmitted from src_i to dst_i); (4) C_i is the size; and finally (5) D_i is the deadline of the flow, i.e., the latest time instant by which at least one copy (original or replicates) of f_i must reach dst_i. We assume all flows are uni-cast, i.e., each flow has a unique destination. We define the set of replicates of f_i as $rep_{f_i} \stackrel{\text{def}}{=} \{f_{i,1}, f_{i,2}, \ldots, f_{i,rep_i}\}$ and assume that each flow and all its replicates are transmitted *simultaneously* over the network.

[1]This heuristic also makes sure that replicated flows are transmitted on routes as disjoint as possible.

[2]The length of each cycle is computed as the Least Common Multiple (L.C.M.) of the periods of all flows.

3. PROPOSED SOLUTION

In this work, we assume that all edges are homogeneous (i.e., they all have the same characteristics and are interchangeable). Before we detail our proposed routing strategy, let us first define a number of concepts for a better understanding of our approach from the reader standpoint.

DEFINITION 1 (ROUTE). *A* route r_i *of flow* f_i *is defined as an ordered list* $\langle (src_i, v_{i,1}), (v_{i,1}, v_{i,2}), \ldots, (v_{i,p}, dst_i) \rangle$ *of edges that can be traversed by* f_i *from its source to its destination.*

DEFINITION 2 (VALID ROUTE). *A* valid route *for* f_i *is defined as any route* r_i *that meets its timing requirement* D_i.

DEFINITION 3 (LENGTH OF A ROUTE). *The length of a route* r_i *denoted by* $len(r_i)$ *is defined as the number of edges along the route.*

DEFINITION 4 (LOAD OF A EDGE). *For every edge* $e = (v_1, v_2) \in V \times V$*, we define the* load of e*, denoted by* $load(e)$*, the sum of the sizes of all flows traversing* e*. Formally, the load of edge* e *is defined by Equation 1.*

$$load(edge) \stackrel{\text{def}}{=} \sum_{f_i \text{ traversing } edge} C_i \quad (1)$$

DEFINITION 5 (MAXLOAD OF A ROUTE). *The* MaxLoad *of a route* r_i*, denoted by* $Maxload(r_i)$*, is defined as the* maximum *load of all edges in* r_i*. Formally, the MaxLoad of route* r_i *is defined by Equation 2.*

$$Maxload(r_i) \stackrel{\text{def}}{=} \max_{edge \in r_i} \{load(edge)\} \quad (2)$$

At this stage, we have all the tools we need to describe our proposed routing solution. The basic idea is as follows. In contrast to the traditional routing schemes (e.g., SPA, ECMP and wt-ECMP), where the underlying strategy is to focus on finding the shortest route for each flow, here we explore all the valid routes. If we denote by R_i the set of all valid routes for flow f_i, then our routing strategy consists in selecting the route that results in the best load distribution in R_i, i.e., the route that minimizes the cost function defined in Equation 3.

$$Cost(r_i, K) \stackrel{\text{def}}{=} Maxload(r_i) + K \cdot len(r_i) \quad (3)$$

In this Equation 3, parameter $K > 0$ is a penalty constant value defined by the user. This parameter is meant to penalize the routes with longer lengths. To make a long story short, it must be looked at as trade-off. It must be set in such a way that the weight of $K \cdot len(r_i)$ in the cost function is significant and $Maxload(r_i)$ does not dominate it and vice-versa. In the latter case, if $K \cdot len(r_i)$ dominates $Maxload(r_i)$, then the cost function would behave like wt-ECMP. On the other front, *Maxload*(r_i) is computed to penalize solutions where some edges in the route are transmitting a high number of flows[3]. Last but not least, if several routes return the same lowest-cost value, then we select one of these routes in an arbitrary manner. Formally, for each flow f_i, its best route $Best(f_i)$ is defined by Equation 4.

$$Best(f_i) \stackrel{\text{def}}{=} \min_{r_i \in \text{valid\_routes}} \{Cost(r_i, K)\} \quad (4)$$

[3]Hence making these edges become potential bottlenecks.

In this equation, variable "valid_routes" denotes the set of all valid routes for flow f_i. Consequently, wt-ECMP is a special case of the proposed approach, where parameter K is sufficiently large and $K \cdot len(r_i)$ dominates $Maxload(r_i)$. Now, we can proceed with the details of our proposed routing schemes.

⋈ **On load-balancing (Algorithm 1)**. The load balancing routing scheme (LB-DRR) takes three components as inputs, namely: (1) the network topology G; (2) the set F of flows to be routed; and finally (3) the user-defined penalty variable K. In the description of the algorithm, the notation $|A|$ refers to the cardinal of set A.

For each flow f_i, after the initialization phase (lines 1 to 3), LB-DRR computes the best route by using Equation 4 (line 6). Then, the load of all edges on this route is updated (line 8) and the selected route is appended to the list of best routes R_i of flow f_i. If the number of replicas of f_i is strictly greater than zero, then all the edges that have already been traversed by the original flow f_i are recorded in variable *used_edge* (line 12). Next, all the valid routes are computed (line 13) and the route $r_{i,j}$ that has the minimum overlap with *used_edges* is selected for replica $f_{i,j}$ (with $j \in [1, rep_i]$) (line 15). If several routes return the same minimum overlap with *used_edges*, then one of these routes is selected arbitrarily and the load of all edges on $r_{i,j}$ is updated (line 17). Note that the edges belonging to *used_edges* are also updated so as to take into account those traversed by replica $f_{i,j}$ (line 19). Thereafter, route $r_{i,j}$ is appended to R_i (line 20) and R_i, which is the list of selected routes for f_i and its replicas, is appended to the list R of the selected routes for all flows (line 23). When this process is completed for all f_i to be transmitted, the algorithm returns the list R (line 25).

⋈ **On congestion recovery (Algorithm 2).** This algorithm, referred to as CR-DRR, is based on the *Tabu metaheuristic* [5] and is reactive in that it aims at re-routing the flows caught in a congestion situation. In a nutshell, the main intuition behind any tabu-based meta-heuristic is to temporarily mark some moves as forbidden so as to force the algorithm to seek for alternative solutions, potentially better in comparison to the current one with respect to a given metric. With this concept in mind, the CR-DRR scheme operates as follows. It takes five components as input: (1) the network topology G; (2) the original routing configuration for all flows R; (3) the congestion threshold *cgst_threshold*[4]; (4) the list of the loads on each edge (*load*); and finally (5) the user-defined penalty variable K. All congested edges according to parameter *cgst_threshold* are stored as "tabu-edges" (*csgt_edges*) and are temporarily removed from the network topology (line 2). For every congestion situation, we initialized the set of congested routes *csgt_routes* (i.e., all routes containing at least one congested edge); the set of flows (*csgt_flows*) traversing the congested routes; and the new set of routes R_{new} to include all routes in R except the congested routes (lines 3 to 5). Then, we seek for alternative routes on the new topology for each congested flow $f_i \in csgt\_flows$ (line 7). From these alternative route(s), we select the best route[5] r_i by using Equation 3 (line 9).

[4]This parameter defines the upper-limit of the load admissible on an edge, otherwise it is deemed as congested.

[5]Again, if several routes return the same minimum cost, then we select one of these in an arbitrary manner.

Algorithm 1: LB-DRR routing scheme.

Data: Network topology G; Set of flows F; Constant K
Result: List of best routes for each flow in F

```
1  R ← empty list[];
2  edges ← Set of all edges in G;
3  load ← zeros[|edges|];
4  foreach f_i ∈ F do
5      R_i ← [];
6      Compute r_i = Best(f_i) (see Equation 4);
7      foreach edge ∈ r_i do
8          load[edge] = load[edge] + C_i;
9      end
10     R_i.append(r_i);
11     if rep_i > 0 then
12         used_edges ← {edge ∈ r_i};
13         routes = valid_routes(G, src_i, dst_i);
14         for j = 1 to rep_i do
15             r_{i,j} = arg min_{r∈routes} (|used_edges ∩ {edge ∈ r}|);
16             foreach egde ∈ r_{i,j} do
17                 load[edge] = load[edge] + C_i;
18             end
19             used_edges = used_edges ∪ {edge ∈ r_{i,j}};
20             R_i.append(r_{i,j});
21         end
22     end
23     R.append(R_i);
24 end
25 return R
```

We check if re-routing flow f_i will not cause congestion on any edge in r_i (line 10). If it does, we leave f_i on its original route old_r_i (line 24). At the end of this process, we update the list *load* in two phases: (i) on the old route old_r_i: we deduct C_i from all edges (line 12) and (ii) on the new route r_i: we augment C_i to all the edges (line 15). We update $cgst\_edges$ (lines 17 to 22). In case there is no alternative route for f_i in $New\_Topology$, it is kept on its original route old_r_i (line 27). Finally, the computed route is appended to R_{new} (line 29) and when all congested flows have been re-routed, the list R_{new} is returned for all flows (line 31).

Algorithm 2: CR-DRR routing scheme.

Data: Network topology G; Original routing configuration R; List of loads on each edge (*load*); Congestion threshold $cgst\_threshold$; Constant K
Result: A new routing configuration R_{new}

```
1  cgst_edges ← {edge in G | load(edge) > cgst_threshold};
2  New_Topology ← G \ cgst_edges;
3  cgst_routes ← {r ∈ R | r ∩ cgst_edges ≠ ∅};
4  cgst_flows ← {f_i traversing a route in cgst_routes};
5  R_new ← R \ cgst_routes;
6  foreach f_i ∈ cgst_flows do
7      routes ← valid_routes(New_Topology, src_i, dst_i);
8      if (routes ≠ ∅) then
9          r_i = arg min_{r∈routes}(Cost(r, K));
10         if (Maxload(r_i) ≤ cgst_threshold) then
11             foreach edge ∈ old_r_i do
12                 load[edge] = load[edge] − C_i ;
13             end
14             foreach edge ∈ r_i do
15                 load[edge] = load[edge] + C_i ;
16             end
17             foreach edge ∈ cgst_edges do
18                 if (load(edge) ≤ cgst_threshold) then
19                     New_Topology =
                         New_Topology.add(edge);
20                     cgst_edges = cgst_edges \ {edge};
21                 end
22             end
23         else
24             r_i = old_r_i
25         end
26     else
27         r_i = old_r_i
28     end
29     R_new.add(r_i);
30 end
31 return R_new
```

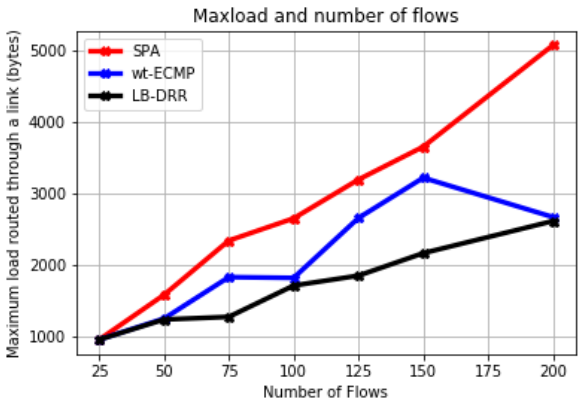
4. EXPERIMENTAL RESULTS

In this section, we report on the experiments conducted on synthetic workloads to evaluate the performance of the proposed heuristics (LB-DRR and CR-DRR) in terms of maximum load transmitted on an edge against SPA and wt-ECMP. Then, we assessed the scalability of the proposed algorithms to demonstrate their applicability.

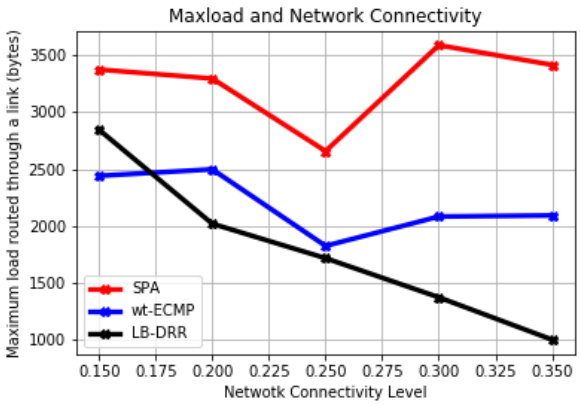
▷ **Setup.** We considered a TSN network, modeled as an *Erdős-Rényi graph* [3] with 50 nodes and a connectivity level falling in the interval $[0.15, 0.35]$. We set $K = 100$ and randomly generated up to 1000 real-time flows in the window $[25, 200]$. For each flow, we assume that its size is between 200 and 1000 bytes and its replication level is randomly chosen between 0 and 2. Also, to constrain the solution space (i.e., to limit the set of valid routes for each flow), we consider the deadline of each flow in the range of 2 to 5 time units and assume a constant traversal time of 1 time unit per edge. In the first batch of experiments, we assumed the LB-DRR routing scheme and the second batch, we assumed the SPA routing scheme. In the latter case, we applied the CR-DRR algorithm to re-route the flows under congestion situations.

▷ **Results and discussion.** From the first batch of experiments, we observed that LB-DRR reduces the maximum load transmitted on an edge (Maxload) by 70.3% and 23.3% in average as compared to SPA and wt-ECMP, respectively. Figure 2a shows the Maxload for each routing scheme when the numbers of flows varies and LB-DRR clearly dominates both SPA and wt-ECMP. By varying the connectivity level of the network (see Figure 2b), we observed that LB-DRR performs better as the network connectivity increases and its Maxload decreases significantly. Note that higher connectivity brings about longer run-time overhead due to the increasing number of routes to be considered.

Figure 3a illustrates the scalability of LB-DRR w.r.t. increasing number of flows. Regarding the increase of the number of flows, we observed that LB-DRR scales linearly, but very slowly (it took only 26 seconds to compute routes

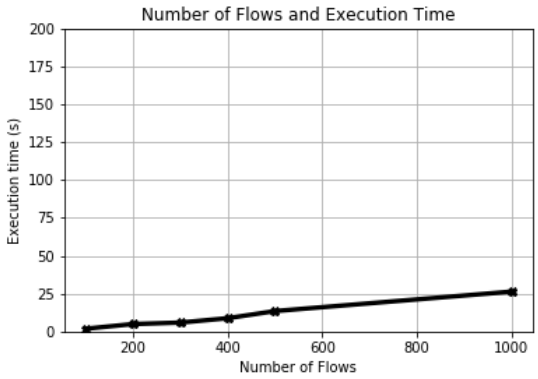


(a) Load balancing: LB-DRR vs. SPA and wt-ECMP.

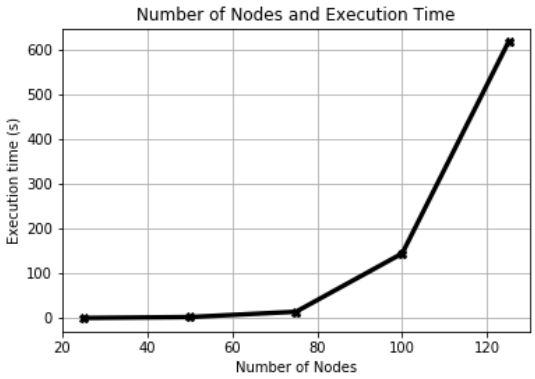


(b) Performance improvement w.r.t. network connectivity.

Figure 2: Load balancing and performance improvement of LB-DRR**.**



(a) Scalability w.r.t. number of flows.



(b) Scalability w.r.t. number of nodes.

Figure 3: Scalability of LB-DRR**.**

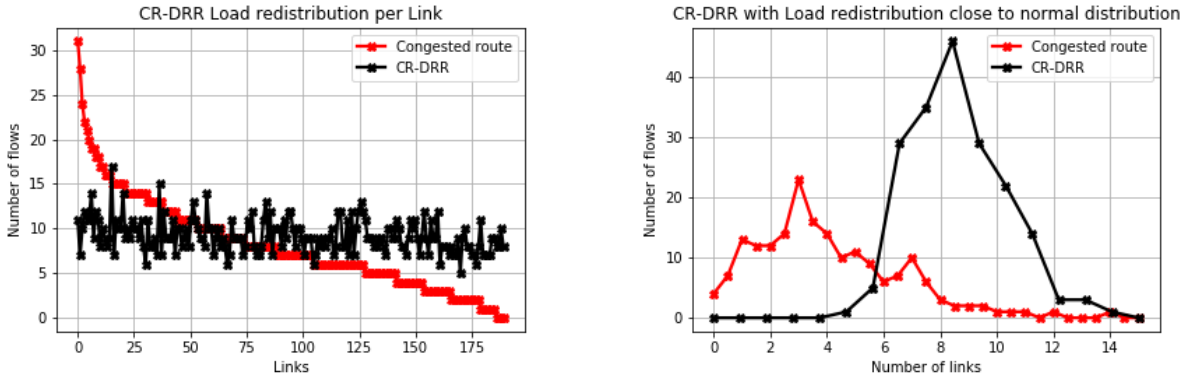
for 1000 flows). Now, regarding the increase of the number of nodes, we set the network connectivity level to 0.2 and consider 100 real-time flows. Figure 3b shows that the execution time of LB-DRR grows exponentially as the number of nodes exceeds 75. However, it could still compute routes for 125 nodes in 11 minutes.

From the second batch of experiments, we routed 750 real-time flows by using SPA, and we observed a huge congestion on the selected routes. Figure 4 shows the congestion recovery and the load redistribution results. In Figure 4a, the network load was initially unbalanced (see the red curve) with several flows routed only on a limited number of edges (see the peak on the far-left), while several edges were left unused (see the long tail to the far-right). By applying the CR-DRR scheme, a tremendous improvement has been observed (see the black curve). Figure 4b shows the load distribution of the congested network before and after CR-DRR is applied. From this figure, the load distribution curve of CR-DRR is close to the normal distribution. Finally, CR-DRR presented the same behavior as LB-DRR in terms of scalability.

5. RELATED WORK

Traffic routing of time sensitive (real-time) flows is non trivial [9]. Routing optimization has been well studied in literature and sophisticated techniques have been proposed [6, 13]. However, contributions on TSN routing schemes have started less than a decade ago. In this context, both the rapid spanning tree protocol and shortest path bridging algorithms have widely been adopted in practice [11]. On another front, the IEEE802.1 Qca standard [8] specifies the Constrained Shortest Path First routing algorithm for TSN transmissions, but this algorithm does not prevent congestion situations and can increase contention in the network, unfortunately. Arif and Atia [1] proposed a methodology to evaluate the routes of a TSN end-to-end connection, but load-balancing was not part of their objectives.

Nayak et al. [10] explored ILP-based algorithms for routing time sensitive flows in TSN networks with Time Aware Shapers (a.k.a. IEEE-802.1Qbv). The proposed approach in their work differs from ours in that it does not address the congestion and load-balancing problems. Targeting a better load balancing for a TSN network, Singh [12] presented an algorithm, based on meta-heuristics, capable of routing new traffic flows at runtime with minimal overhead. But, the proposed approach adopts the shortest path algorithm (SPA) as initial solution and not all feasible routes are considered. This limits the solution search-space, unfortunately. Gavrilut et al. [4] also took the same path and



(a) CR-DRR adopted to recover from congestion

(b) Load distribution under CR-DRR.

Figure 4: CR-DRR **Congestion recovery.**

proposed heuristic methods for topology and routing synthesis. Their method tries to achieve an optimal usage of the switches and links as well as an efficient routing of flows. However, they did not consider load-balancing. This paper fills this gap: it solves the problem of load-balancing, disjoint routing for duplicated flows and dynamic re-routing in congestion situation.

6. CONCLUSION

In this work, we proposed two routing heuristics, referred to as LB-DRR and CR-DRR, in order to address the problems of load-balancing and congestion in TSN-based networks. We evaluated the performance of the proposed schemes against the popular SPA and wt-ECMP routing algorithms and showed an improvement of more than 70% and 20%, respectively. This improvement has been observed w.r.t. the maximum load transmitted on an edge. On another front, the proposed heuristics exhibited high scalability w.r.t. an increase in the number of flows. Given these promising results, we plan to investigate both the routing and scheduling of time-sensitive flows simultaneously. Also, it would be interesting to address multicast flows; develop techniques to reduce the search space of valid routes as the number of nodes increases and finally quantify the impact of these techniques on the end-to-end timing requirements.

Acknowledgment

This work was partially supported by National Funds through FCT/MCTES (Portuguese Foundation for Science and Technology), within the CISTER Research Unit (UID/CEC/04234)

7. REFERENCES

[1] F. A. R. Arif and T. S. Atia. Load balancing routing in time-sensitive networks. In *Problems of Inf. Science and Technology (PIC S&T), Third Int. Scientific Practical Conf.*, pages 207–208. IEEE, 2016.

[2] F. Dürr and N. G. Nayak. No-wait packet scheduling for ieee time-sensitive networks (TSN). In *Proc. of the 24th Int. Conf. on RTNS*, pages 203–212. ACM, 2016.

[3] L. Erdős, A. Knowles, H.-T. Yau, J. Yin, et al. Spectral statistics of erdős–rényi graphs i: local semicircle law. *The Annals of Probability*, 41(3B):2279–2375, 2013.

[4] V. Gavrilut, B. Zarrin, P. Pop, and S. Samii. Fault-tolerant topology and routing synthesis for IEEE time-sensitive networking. In *25th Int. Conf. on RTNS*, pages 267–276. ACM, 2017.

[5] F. Glover. Tabu search: A tutorial. *Interfaces*, 20(4):74–94, 1990.

[6] M. D. Grammatikakis, D. F. Hsu, M. Kraetzl, and J. F. Sibeyn. Packet routing in fixed-connection networks: A survey. *Journal of Parallel and Distributed Computing*, 54(2):77–132, 1998.

[7] IEEE. Time-Sensitive Networking Task Group.

[8] IEEE. *IEEE Standard for Local and metropolitan area networks— Bridges and Bridged Networks - Amendment 24*. IEEE, 2016.

[9] A. Nasrallah, A. Thyagaturu, Z. Alharbi, C. Wang, X. Shao, M. Reisslein, and H. Elbakoury. Ultra-Low Latency (ULL) Networks: The IEEE TSN and IETF DetNet Standards and Related 5G ULL Research. *IEEE Comm. Surveys & Tutorials*, pages 1–59, 2018.

[10] N. G. Nayak, F. Duerr, and K. Rothermel. Routing Algorithms for IEEE802. 1Qbv Networks. *RTN workshop, ECRTS*, 2017.

[11] P. Pop, M. L. Raagaard, S. S. Craciunas, and W. Steiner. Design optimisation of cyber-physical distributed systems using IEEE time-sensitive networks. *IET Cyber-Physical Systems: Theory & Applications*, 1:86–94, 2016.

[12] S. Singh. Routing Algorithms for Time Sensitive Networks. Master's thesis, Univ. of Stuttgart, 2017.

[13] B. Wang and J. C. Hou. Multicast routing and its qos extension: problems, algorithms, and protocols. *IEEE network*, 14(1):22–36, 2000.

[14] Z. Wang and J. A. Crowcroft. Quality-of-service routing for supporting multimedia applications. *IEEE Journal on Sel. Areas in Comm.*, 14:1228–1234, 1996.

Symphony - Routing Aware Scheduling for DSME Networks

Harrison Kurunathan, Ricardo Severino, Anis Koubaa, Eduardo Tovar
CISTER/ISEP and Prince Sultan University
Porto, portugal and Saudi Arabia
{hhkur,rarss,emt}@isep.ipp.pt,akoubaa@psu.edu.sa

ABSTRACT

Deterministic Synchronous Multichannel Extension (DSME) is a prominent MAC behavior first introduced in IEEE 802.15.4e. It can avail deterministic and best effort Service using its multisuperframe structure. RPL is a routing protocol for wireless networks with low power consumption and generally susceptible to packet loss. These two standards were designed independently but with the common objective to satisfy the requirements of IoT devices in terms of limited energy, reliability and determinism. A combination of these two protocols can integrate *real-time QoS demanding* and *large-scale IoT networks*. In this paper, we propose a new multi-channel, multi-timeslot scheduling algorithm called Symphony that provides QoS efficient schedules in DSME networks. In this paper we provide analytical and simulation based delay analysis for our approach against some state of the art algorithms. In this work, we show that integrating routing with DSME can improve reliability by 40 % and by using Symphony, we can reduce the network delay by 10-20% against the state of the art algorithms.

KEYWORDS

IEEE 802.15.4e, DSME, Multisuperframe scheduling

1 INTRODUCTION

Modern embedded systems, coupled with the advancements of digital communication technologies, have been enabling a new generation of systems, tightly interacting with the physical environment via sensing and actuating actions: Cyber Physical Systems (CPS). These systems, characterized by an unprecedented levels of ubiquity, have been increasingly relying upon wireless communication technologies to provide seamless services via flexible cooperation, supporting different *Internet of Things* (IoT) applications. Several of these applications demand increased Quality of Service (QoS), namely regarding determinism, reliability, scalability and no compromise on energy efficiency.

The IEEE 802.15.4e standard provides time critical support for IoT applications by introducing new MAC behaviors like TSCH, DSME and LLDN [10]. Among these MAC behaviors, ***DSME*** - *Deterministic Synchronous Mutichannel Extension* is a very versatile MAC behavior. Like the classic IEEE 802.15.4, it can alternate between CSMA/CA and Guaranteed Timeslots (GTS) to support both best effort and time-critical communications. DSME introduces several features like the multichannel access to increase the scalability and robustness of the network manifold. Despite its many enhanced features, the standard does not specify any network layer for QoS centric routing purposes. Although it can support mesh topology, no intuition is given regarding the right mechanism that can dynamically setup the necessary service.

Integrating a distributed routing protocol like RPL over DSME "helps achieving increased scalability (via routing), while providing robustness to cope with network changes". The challenge lies in the integration of these standards and providing DSME schedules periodically. In this paper we present an approach to integrate DSME with RPL and an algorithm called Symphony to deliver schedules periodically for the DSME associated nodes.

The main contribution in this paper are as follows:

- We overview the DSME and RPL networks and provide a system architecture for efficient integration of these standards.
- We introduce *Symphony*, a time-frequency algorithm that helps DSME nodes to maintain schedules periodically with dynamic changes in the network based on RPL.
- Using simulations we show the advantage of RPL over a traditional DSME network in terms of reliability.
- We use simulations to learn the advantages of Symphony over the state of the art algorithms in terms of delay.

The rest of the paper is structured as follows: in section II, we provide a brief literature survey then in Section III, we give an overview of DSME and RPL, then in Section IV we elaborate the system architecture of RPL over DSME. In Section V, we introduce and discuss our algorithm Symphony. Finally, we provide an in-depth performance analysis of our architecture and compare Symphony with some of the state of the art algorithms for DSME scheduling.

2 RELATED WORKS

Following the standardizing efforts on protocols like 6LoWPAN [12], the *Internet Engineering Task Force* (IETF) has focused on implementing 6TiScH [4], a combination of the TSCH MAC behavior of IEEE 802.15.4e, IPV6 and RPL. Implementing RPL over these standards helped in providing optimal routing for the transmissions and increased the overall reliability. *Orchestra* [5] is one of the open source implementations based on 6TiScH, in which, the nodes automatically compute their own local schedules and maintain several schedules for different traffic scenarios. Orchestra was able to deliver high end-end delivery ratios with a good latency-energy balance. In our work we provide an architecture for the implementation of RPL over DSME networks.

The DSME MAC behavior of IEEE 802.15.4e provides increased determinism and reliability in a multi channel environment. Several researchers like in [13] and [15] have demonstrated the advantages of DSME in terms of lesser delays and aggregate throughputs compared with standard IEEE 802.15.4.

Conference'17, July 2017, Washington, DC, USA
2019. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

There is some literature on developing scheduling algorithms for the enhancements of IEEE 8021.5.4e to provide an optimal service. For example, in case of TSCH, an other prominent MAC behavior of IEEE 802.15.4e, a new enhancement called *Adaptive-TSCH* [3] was developed by Peng Du. In this algorithm, the author provides the nodes, the ability to hop amongst a subset of channels which are deemed reliable based on their respective link qualities. Using this technique an average increase of ETX (Expected Transmission Count) by 5.6 % was observed.

There is also some research in implementing multi channel scheduling algorithms for DSME [14] to improve its reliability. In this algorithm several dummy GTSs slots were allocated to occupy the transmissions in case of a transmission failure. However, this approach can impact over the overall delay of the network. In this paper we compare this scheduling algorithm with Symphony.

Several researchers [9], [1] in their work developed analytical and simulation assessments of DSME and TSCH MAC behaviors. They proved that DSME performs better than TSCH in terms of end to end latency when the number of nodes is higher than 30. The enhanced features of DSME like CAP reduction helped in reducing the end to end latency and also achieving better throughput and scalability.

In this paper, we propose merging the functionalities of DSME and RPL and aim at reducing the latency of the overall network. RPL will provide optimal routes based on any objective function such as power efficiency or link reliability, while our proposed algorithm *Symphony* will provides dynamic GTS schedules periodically for the entire network with minimal delay.

3 BACKGROUND TO DSME AND RPL

The DSME network provides deterministic communication using its beacon enabled mode. This beacon enabled mode is supported by multisuperframes that comprises stacks of superframes as shown in Figure 1. Every superframe comprises of a Contention Access Period (CAP) in which the nodes contend to access the channel and a Contention Free Period (CFP) in which the nodes send the data using Guaranteed timeslots (GTSs).

The superframe is defined by *BO*, the *Beacon Order* which is the transmission interval of a beacon in a superframe, *MO* the *Multi superframe Order* that represents the enhanced beacon interval of a multi-superframe and *SO* the *Superframe Order* that represents the beacon interval of a superframe. The number of superframes in a multisuperframe can be given by $2^{(MO-SO)}$. The values of BO, SO and MO are set by the PAN coordinator and is conveyed to the nodes via an Enhanced Beacon (EB) at the beginning of each Multisuperframe. This EB helps in the overall synchronization of the network.

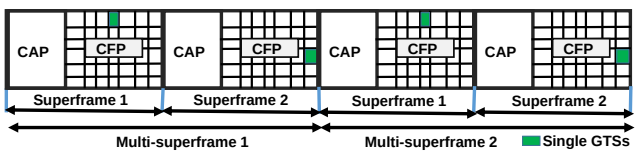


Figure 1: Superframe structure with *BO*=3, *MO*=3, *SO*=2

DSME can use *channel adaptation* or *channel hopping* for multichannel access in the CFP. In channel hopping, the hopping-sequence of the channels for data transmission is pre-determined and the same hopping pattern is repeated till the end of the data transmission. Whereas in channel adaptation, the transmissions are allowed to hop over the channels based on their link quality. The multichannel access mechanisms of DSME allow several transmissions to occur in the same timeslot within different channels. These multichannel access schemes open the possibility of forming complex topologies like mesh for DSME networks.

RPL is a routing protocol that integrates technologies like IEEE 802.15.4 and IPv6 protocols. It supports both mesh as well as hierarchical topologies, and is specifically designed to support networks that are prone to high exposed packet losses and limited resources in terms of computation and energy.

RPL is based on hierarchical Directed Acylic graphs (DAGs) in which a node can associate itself with many parent nodes. The destination node of an RPL is called a sink and the nodes through which a route is provided to internet are called gateways. RPL organizes these nodes as Destination-Oriented DAGs (DODAGs). In an RPL, every router in the system identifies and associates with a parent. This association is done based on an Objective Function (OF). *OF* can be based on quality determining parameters like LQI (Link Quality Indicator) and RSSI (Received Signal Strength Indicator). OF helps in providing an optimal routing path using metrics like latency or power efficiency.

4 SYSTEM MODEL

In this paper, we introduce *Symphony* a dynamic algorithm that provides *"a variety of schedules to fit onto the multichannel DSME - GTSs based on optimal routing decisions made by RPL."*

RPL can use either broadcast or unicast to disseminate the Objective Function metrics using the DODAG Information Object "*DIO*". This information also can be requested using the DODAG Information Solicitation "*DIS*". The routing paths can be disseminated using a Destination Advertisement Object "*DAO*". In an RPL network perspective, when a node wants to join the DODAG it receives a signaling message from a neighbor router, it (*i.*) adds the sender address to its parent list, (*ii.*) computes a rank according to the Objective Function such as reliability determining factors like LQI (Link Quality Indicator) or RSSI, (*iii.*) forwards the updated rank information to the parent.

For the system model we consider a mesh network (Figure 3) with fully functional devices (FFDs) that can receive and transmit messages in the Guaranteed Timeslots (GTSs). The FFDs maintain the schedules locally and have their own superframes to accommodate the nodes associated to them. They also have a routing table to maintain the nodes associated to them. Every superframe carries various kinds of traffic to support symphony, such as the periodic beacons for synchronization, RPL signaling traffic and application data traffic.

In case of a GTS allocation, the allocation-request is sent to the parent node (FFD) through the RPL network. The Symphony algorithm at the coordinators helps to find the most efficient allocation in the time-frequency domain. Symphony aims at "maintaining schedules for all the transmissions in parallel without a overlap". It

chooses specific channels and timeslots for the GTSs transmissions in order to achieve a "interference and a contention free scheduling".

A concrete example of our architecture (Figure 2) is as follows:

- A dedicated beacon broadcast for synchronization between every superframe for every "X" slots, where "X" is the superframe duration of every individual superframe.
- A dedicated beacon broadcast for synchronization every multi superframe for every "Y" slots, where "Y" is the multi superframe duration coordinating every superframe with the duration of "X".
- A Enhanced Beacon common for all coordinators in the network carrying the broadcast + unicast packets for RPL signaling (DIO, DIS, DAO), repeating every "Y" slots. In accordance with the standard, the Enhanced Beacon payload can be a variable and it carries the RPL information.
- Dedicated unicast signal from the slave node to the parent node followed by N unicast signals from the coordinator to the slave nodes.

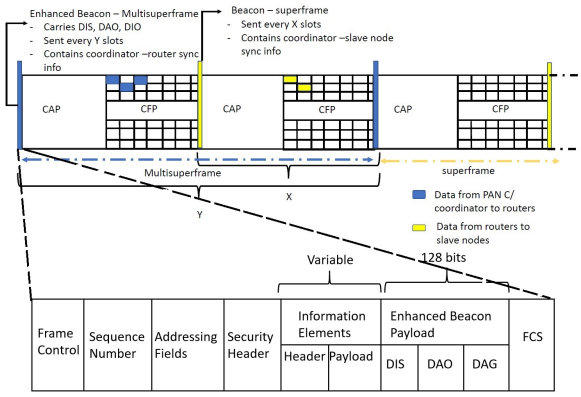


Figure 2: System Architecture

5 SYMPHONY ALGORITHM

Symphony is a routing aware algorithm that was designed based on the methods of solving a *Constraint Satisfaction Problem* (CSP). It performs scheduling based on several decision parameters like that of the classic *eight queens problem* [6]. The optimal assignment of time-slots and frequencies which is done by Symphony is considered to be an NP-Hard problem [7]. Symphony will aim at providing *"dynamic allocation of timeslots based on the routing information provided by RPL."*

This scheduling problem is bounded by two major constraints, which will be a determining factor in establishing an optimal solution.

Constraint 1: *No same nodes either involving in transmission or reception must fall under the same timeslot.*

This constraint helps in avoiding all the interference in the network. The standard offers a possibility for *different* nodes to communicate in a *same* timeslot simultaneously in *different* channels, whereas, the same nodes can communicate in *different* timeslots within the *same* or *different* channels.

Constraint 2 *Maximum number of channels and minimum number of timeslots should be used.*

This constraint is more of a "quality constraint" that helps in establishing the optimality of the algorithm. This constraint helps in achieving the fact that "more bandwidth will not be wasted" and at the same time "minimal timeslots will be used". By satisfying this constraint the overall network throughput and scalability of the network can be significantly increased, concomitantly achieving minimal latency.

For our analysis we take a mesh network with 5 different nodes that are interconnected with each other as shown in Fig 3. This topology is considered to be obtained through RPL. This network model can also be extended to any number of slave nodes with reduced functionality (only receive information). For the schedule placement, we only consider the guaranteed timeslots in the CFP region of the DSME superframe with 3 channels in our model.

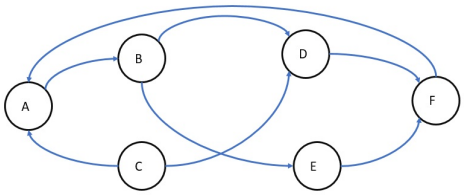


Figure 3: example of a mesh network

A schedule is considered to be optimal when it uses the resources stringently and fully utilizing the multichannel capability of DSME (Constraint 2). The optimality is checked by the following equation:

$$NT = \lceil (n/C) \rceil \tag{1}$$

In the above equation NT represents the number of timeslots occupied, n represents the total number of transmissions and C is the number of channels used. It should also be noted that proving the optimality should satisfy both **Constraint 1** and **Constraint 2**. This optimal schedule can be obtained by an ILP formulation provided in [11].

Our algorithm is a two step process, first we get Transmission Based Ranks (TBR) for the nodes based on the number of routes determined by the RPL. For example, in Figure 3, nodes B and C have a transmission rank of 2, as both the nodes have two links formed from them. We denote this Transmission Based Ranking as *TBR* in our algorithm. As an output of TBR, we group several sets of transmissions based on their respective ranks. In case of identical ranks, we place the elements under a single subset. This step is done in order to avoid any interference conflict in the scheduling (Constraint 1). The subsets are grouped for all the transmission routes provided by the RPL. The algorithm can be used for any number of nodes that are associated with a PAN Coordinator or a router to form respective schedules for the superframe.

For the example provided in Figure 3, we start placing transmissions from C in adjacent timeslots (highest rank). By placing these elements in the adjacent timeslots, we can negate any chances of interference that can occur by the transmissions trying to transmit

Initialize
input: Updated schedule from RPL
step 1
Procedure: make TBR for all the nodes in the network
if TBR succesful **then**
  return value : go to step 2 ;
else if *case of identical ranks* **then**
  Place the elements in a same subset;
else
  The transmissions are invalid
end
step 2
Procedure: place the subset with the highest rank adjacent to each other
Assign adjacent row slots till $subset1 \rightarrow null$
Assign subset 2 in the next row of the first column
if constraint *not satisfied* **then**
  place the transmissions in the first row ;
else if constraint *satisfied* **then**
  continue placing the transmissions till $allthesubsets \rightarrow null$;
else
  The transmissions are invalid
end

Algorithm 1: Symphony

along the same timeslot. Further as the highest rank is placed initially, we devise a better strategy to accommodate the rest of the nodes in a more optimal way, so that less number of timeslots are utilized in scheduling. This step is now followed by the scheduling transmissions from B in the next channel of the same timeslot. This process is then backtracked to assign all the transmissions. Using this algorithm, we receive an optimal solution as shown in Figure 4

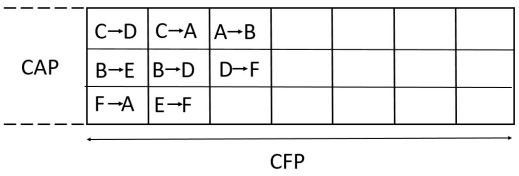


Figure 4: Symphony schedule solution

6 PERFORMANCE ANALYSIS

Our performance analysis of this work is two fold: first we demonstrate the improvement in reliability with routing implemented over a DSME network. Then we use probabilistic analysis to calculate the delay and compare the advantages of symphony over several state of the art algorithms.

Every node in the network derives a ETX (Expected Transmission Count). This is a parameter that is helpful in estimating the frame loss ratio at the link. The ETX is dependent on the forward (P_f) and the backward frame losses (P_b) of the nodes in a network, and this value can be given by:

$$ETX = 1/(1 - P_f)(1 - P_b) \tag{2}$$

ETX can determine the reliability of the links as the parameter represents the inverse of successful packet delivery(P_S):

$$ETX = 1/(P_{Sf} \times P_{Sb}) = 1/Reliability \tag{3}$$

In an RPL enabled network, the nodes will change the routes to the sink when there is a deterioration of the link quality and eventually the overall ETX. The delay also can increase when more additional routes are deployed to reach the sink in case of a failure.

Using OpenDSME [8] an Omnet based simulation platform, we simulated the reliability over a network of 25 nodes with static concentric mobility type. Reliability of the network was calculated based on the number of successful packet delivery as shown in Equation 3. In the radio medium, we introduce a constant interference range to emulate a real-time wireless network. We used a payload of 75 bytes carried in 100 packets over 16 channels of the DSME network in accordance to the standard parameters. Without having routing established for the network layer, it was noted that the reliability of the network depletes steadily with the increase in the number of nodes. We repeated the same experiment with the same network configuration but with generic routing employed in the network layer. We were able to observe that the reliability does not deplete steadily and almost shows 40% betterment results (Figure 5).

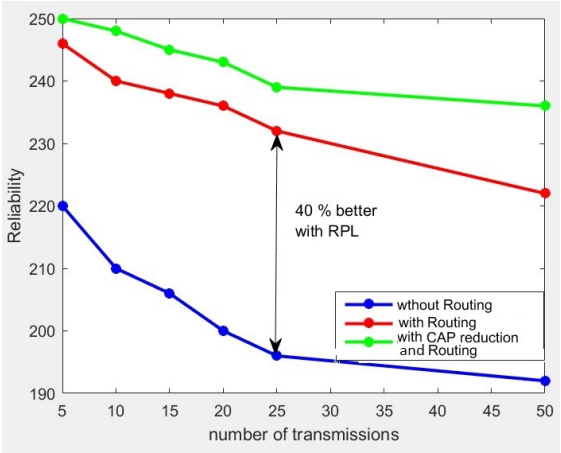


Figure 5: Reliability with generic routing

For the performance analysis of Symphony, we decided to carryout a probability based delay analysis and then complement our findings with simulations carried out in OpenDSME [8]. In both our numerical and simulation analysis we compared the performance of Symphony against state of the art algorithms like MDT [14], best effort DSME scheduling and Random FIFO.

The average transmission delay can be calculated for successfully transmitted GTS frames in the multisuperframe can be given by:

$$\delta = \sum_{i=0}^{\infty} P^{f}_{(i,m)}(i(MI)) \tag{4}$$

Considering the schedule for routing is carried our every multisuperframe, $P^{f}_{(i,m)}$ is the probability that the GTS is successfully transmitted in the i^{th} superframe of the multisuperframe m. MI is the summation of all the individual BIs (Beacon Intervals) within the multisuperframe. To calculate this probability let us take two parameters: X^{s}, the total number of GTS that is successfully transmitted, and $X^{S}_{(i,c)}$, the number of GTS that have to wait i superframes with c channels within a multisuperframe for its successful transmission. Using these parameters the probability $P^{f}_{(i,m)}$ can be formulated as:

$$P^{f}_{(i,m)} = \sum_{0}^{i} X^{S}_{(i,c)}/X^{S} \tag{5}$$

This probability considers the success of all the transmissions within the multisuperframe m. Considering that the first set of GTS frames based on the symphony schedule that gets successfully placed in the initial attempt, they need not wait another superframe interval for their data transmission. Let us consider this as $X^{S}_{(0,c)}$. The value of H varies depends on the success of this transmission.

$$\begin{gathered} X^{S}_{(0,c)} = H(1-P_e), \\ where\ c = (0-16)\ and\ H \in (0,1) \end{gathered} \tag{6}$$

The value of $X_{(i)}$ will be incrementing as with the failures to accommodate a successful transmission. The GTS superframes that wait till the first adjacent superframe to get transmitted successfully can be denoted by $X^{S}_{(1,c)}$, this value can be formulated as:

$$X^{S}_{(1,c)} = H(1-P_e) \tag{7}$$

where, H is the probability of failure to get accommodated within the initial transmission.The value of H can be given as $P_e e^{-BI \cdot c \cdot i\lambda}$, this probability is with an assumption that all the transmissions shall be carried out within the multisuperframe with i superframes and c channels with a GTS arrival rate of λ. Generalizing for all the i superframes, the successful transmissions can be denoted as:

$$X^{S}_{(i,c)} = H^{(i)}(1-P_e) \tag{8}$$

The value of the successfully transmitted GTS in a single superframe can be given as:

$$X^{S} = \sum_{i=0}^{m} H^{(i)}(1-P_e) \tag{9}$$

using the aforementioned equations, the probability to be transmitted in the i^{th} superframe can be calculated as:

$$P^{f}_{(i,m)} = (1-H) \cdot H^{i} \tag{10}$$

and the overall average delay of the network can be given as:

$$\delta = \sum_{i=0}^{m} (1-H) \cdot H^{i}(i(MI)) \tag{11}$$

For the numerical analysis we consider a multisuperframe with 2 superframes over 3 channels. We also consider three arrival rates for the delay analysis. The increase in delay can be due to lesser arrival rates. Lesser arrival rates also can have a negative impact on the throughput of the network. However the multichannel feature in DSME contributes to lesser delay and larger throughput.

We now use the probabilistic approach to calculate the delay of schedule placement within a superframe. Unlike the calculation for the entire multisuperframe, this calculation must be carried out for every timeslot (T_s) of a single superframe. For this case, we take the value of H and replace with H_{tslot} which is the probability of failure to accommodate within the initial timeslot. This aforementioned value can be expressed as:

$$H_{tslot} = P_e e^{-T_s \cdot c \cdot i\lambda} \tag{12}$$

In order to generalize the aforementioned equation, let us consider that all the timeslots have an equal size for all the i superframes in the multisuperframe. Hence we can derive a formulation for the delay for single GTS that fails to occupy the first timeslot and moves to the next. Now we derive the delay for a timeslot to be:

$$\delta_{timeslot} = P_e e^{-T_s \cdot c \cdot i\lambda}(T_l)/(1 - P_e e^{-T_s \cdot c \cdot i\lambda}) \tag{13}$$

For numerical analysis, we compared symphony with MDT [14] and brute-force FIFO algorithms [2]. This method is also used for the GTS scheduling allocation in the OpenDSME framework [8] for DSME implementation. The analysis shown in Figure 6 provides the Transmission delay of the GTS frames for a set of transmissions for different arrival rates (25, 50, 100 Kbps). With the change in the topology of the network (addition of nodes), RPL updates a new set of transmissions to be scheduled in the following multisuperframe.

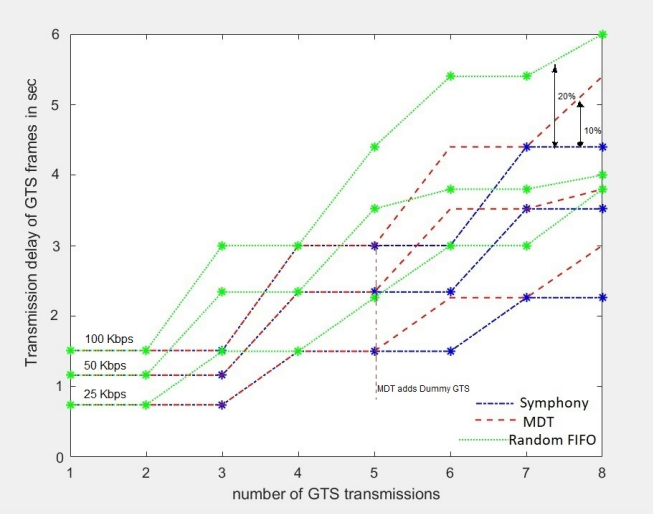


Figure 6: number of transmissions vs GTS delay (Analytical)

MDT under-performs because it spares timeslots aiming better reliability of the network. Certain amount of dummy timeslots are allocated for reliability purposes, contributing to the delay. These dummy packets result in more wasted bandwidth, eventually contributing to delay. The *Random FIFO* technique works based on best

effort. In case of any conflict, the transmission is scheduled the eventual superframe to send the data. *Symphony* fills all the timeslots stringently on the basis of channels available, thus eventually leading to lesser transmission delays and also increased robustness. Unlike Random FIFO and MDT, the Symphony schedules did not wait until another Multisuperframe timeperiod to accommodate its transmissions. Hence, Symphony was stringently able to achieve lesser delay comparatively.

To complement our analytical results, we carried out simulations for Symphony using the OpenDSME platform. We conducted experiments for delay over several GTS transmissions. We simulated our experiments at a 100 Kbps traffic rate for varying number of transmissions. In our simulations, we pitted Symphony against MDT, standard DSME and CSMA/CA.

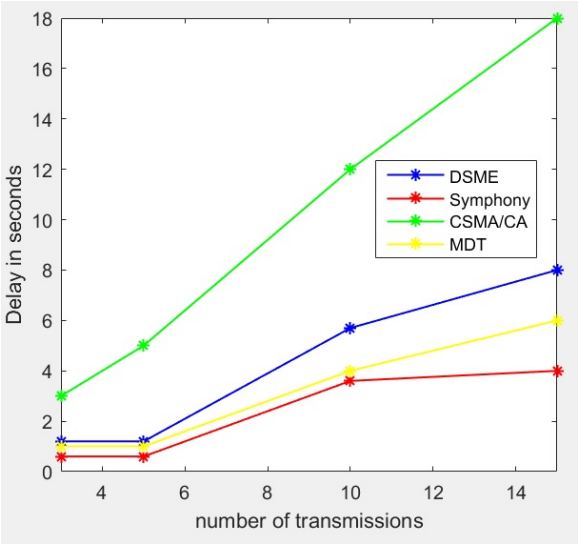


Figure 7: number of transmissions vs GTS delay

From our performance analysis and simulations, we learn that Symphony is able to achieve 10-15 % reduction delay when compared to many state of the art algorithms for DSME. As the number of transmissions increase Symphony is able to provide a schedule in such a way it is optimal to achieve a lesser latency. It also must be noticed that the transmissions that are provided onto Symphony is derived through RPL, which in-turn can improve the overall Quality of Service of the network manifold. We believe that integrating RPL onto DSME and providing a routing aware algorithm like Symphony can push DSME to become a de-facto standard for seamless IoT communication.

7 FUTURE SCOPE

In this paper we introduce an approach to improve the overall Quality of Service in a periodically evolving real-time DSME network. We provide an architecture for the integration of RPL and DSME technologies through a routing-aware algorithm called Symphony. The key goal of this work is to provide dynamic optimal schedules for GTS allocation based upon the RPL topology information, while reducing the latency of the overall network.

Through our detailed mathematical and simulation analysis we compared Symphony to some of the state of the art algorithms to find that, Symphony with its stringent packing strategy, performs better in terms of latency. By adopting symphony, we can witness a a decrease in latency by 10-15 %. Our Simulation of RPL also provides us an insight that routing over a dynamically evolving DSME networks can improve its reliability manifold.

We aim at implementing our algorithm in a hardware platform which will enable us to compare with the existing analytical results. We also intend to develop an open-source implementation of this protocol for Commercially Off The Shelf WSN platforms (COTS) (e.g. TelosB devices), to validate the results over real WSN hardware.

REFERENCES

[1] Giuliana Alderisi, Gaetano Patti, Orazio Mirabella, and Lucia Lo Bello. 2015. Simulative assessments of the ieee 802.15. 4e dsme and tsch in realistic process automation scenarios. In *Industrial Informatics (INDIN), 2015 IEEE 13th International Conference on*. IEEE, 948–955.

[2] Marc Domingo-Prieto, Tengfei Chang, Xavier Vilajosana, and Thomas Watteyne. 2016. Distributed pid-based scheduling for 6tisch networks. *IEEE Communications Letters* 20, 5 (2016), 1006–1009.

[3] Peng Du and George Roussos. 2012. Adaptive time slotted channel hopping for wireless sensor networks. In *Computer Science and Electronic Engineering Conference (CEEC), 2012 4th*. IEEE, 29–34.

[4] Diego Dujovne, Thomas Watteyne, Xavier Vilajosana, and Pascal Thubert. 2014. 6TiSCH: deterministic IP-enabled industrial internet (of things). *IEEE Communications Magazine* 52, 12 (2014), 36–41.

[5] Simon Duquennoy, Beshr Al Nahas, Olaf Landsiedel, and Thomas Watteyne. 2015. Orchestra: Robust mesh networks through autonomously scheduled tsch. In *Proceedings of the 13th ACM conference on embedded networked sensor systems*. ACM, 337–350.

[6] Robert W Floyd. 1967. Nondeterministic algorithms. *Journal of the ACM (JACM)* 14, 4 (1967), 636–644.

[7] Ted Herman and Sébastien Tixeuil. 2004. A distributed TDMA slot assignment algorithm for wireless sensor networks. In *Algosensors*, Vol. 3121. Springer, 45–58.

[8] Florian Kauer, Maximilian Köstler, Tobias Lübkert, and Volker Turau. 2017. OpenDSME-A portable framework for reliable wireless sensor and actuator networks. In *Networked Systems (NetSys), 2017 International Conference on*. IEEE, 1–2.

[9] Harrison Kurunathan, Ricardo Severino, Anis Koubâa, and Eduardo Tovar. 2017. Worst-case bound analysis for the time-critical MAC behaviors of IEEE 802.15. 4e. In *Factory Communication Systems (WFCS), 2017 IEEE 13th International Workshop on*. IEEE, 1–9.

[10] Harrison Kurunathan, Ricardo Severino, Anis Koubaa, and Eduardo Tovar. 2018. IEEE 802.15. 4e in a Nutshell: Survey and Performance Evaluation. *IEEE Communications Surveys & Tutorials* (2018).

[11] Harrison Kurunathan, Ricardo Severino, Anis Koubâa, Eduardo Tovar, et al. 2018. RPL over DSME: A Technical Report. CISTER.

[12] Nandakishore Kushalnagar, Gabriel Montenegro, and Christian Schumacher. 2007. *IPv6 over low-power wireless personal area networks (6LoWPANs): overview, assumptions, problem statement, and goals*. Technical Report.

[13] Junhee Lee and Wun-Cheol Jeong. 2012. Performance analysis of ieee 802.15. 4e dsme mac protocol under wlan interference. In *ICT Convergence (ICTC), 2012 International Conference on*. IEEE, 741–746.

[14] Junhee Lee, Wun-Cheol Jeong, and Byeong-Cheol Choi. 2016. A multi-channel timeslot scheduling algorithm for link recovery in wireless multi-hop sensor networks. In *Information and Communication Technology Convergence (ICTC), 2016 International Conference on*. IEEE, 871–876.

[15] Tuomas Paso, Jussi Haapola, and Jari Iinatti. 2013. Feasibility study of ieee 802.15. 4e dsme utilizing ir-uwb and s-aloha. In *Personal Indoor and Mobile Radio Communications (PIMRC), 2013 IEEE 24th International Symposium on*. IEEE, 1863–1867.

[16] Prasan Kumar Sahoo, Sudhir Ranjan Pattanaik, and Shih-Lin Wu. 2017. A Reliable Data Transmission Model for IEEE 802.15. 4e Enabled Wireless Sensor Network under WiFi Interference. *Sensors* 17, 6 (2017), 1320.

[17] Thomas Watteyne, Antonella Molinaro, Maria Grazia Richichi, and Mischa Dohler. 2011. From manet to ietf roll standardization: A paradigm shift in wsn routing protocols. *IEEE Communications Surveys & Tutorials* 13, 4 (2011), 688–707.

SlotSwapper: A Schedule Randomization protocol for Real-Time WirelessHART Networks

Ankita Samaddar, Arvind Easwaran, Rui Tan
Nanyang Technological University, Singapore
{ankita003,arvinde,tanrui}@ntu.edu.sg

ABSTRACT

Industrial process control systems are time-critical systems where reliable communications between sensors and actuators need to be guaranteed within strict deadlines to maintain safe operation of all the components of the system. WirelessHART is the most widely adopted standard which serves as the medium of communication in industrial setups due to its support for Time Division Multiple Access (TDMA) based communication, multiple channels, channel hopping, centralized architecture, redundant routes and avoidance of spatial re-use of channels. However, the communication schedule in WirelessHART network is decided by a centralized network manager at the time of network initialization and the same communication schedule repeats every hyper-period. Due to predictability in the time slots of the communication schedule, these systems are vulnerable to timing attacks which eventually can disrupt the safety of the system. In this work, we present a moving target defense mechanism, the SlotSwapper, which uses schedule randomization techniques to randomize the time slots over a hyper-period schedule, while still preserving all the feasibility constraints of a real-time WirelessHART network and makes the schedule uncertain every hyper-period. We tested the feasibility of the generated schedules on random topologies with 100 simulated motes in Cooja simulator. We use schedule entropy to measure the confidentiality of our algorithm in terms of randomness in the time slots of the generated schedules.

Keywords

WirelessHART, schedule, randomization, entropy

1. INTRODUCTION

Time-critical systems such as the industrial process control systems are real-time cyber-physical systems (CPS) that monitor and control the production lines in a manufacturing plant. The number of devices in such setup keeps increasing. To support more devices and to cope up with frequent changes in the network topology due to addition (removal) of devices to (from) the network, a switch of the communication infrastructure from wired networks to wireless networks is desirable. Among the existing wireless sensor network (WSN) standards, WirelessHART is best suited for the industrial process control systems due to its reliable TDMA-based schedule, centralized architecture, multi-channel support, channel hopping, redundancy in routes, and avoidance of spatial re-use of channels.

RTN'19, July 2019, Stuttgart, Germany
©2019 Copyright retained by the authors.

Although the use of wireless brings flexibility and adaptability to the communication infrastructure, it increases the threats of cyber attacks. Some recent sophisticated attacks against critical infrastructures such as Stuxnet [1] and Dragonfly [2] have alerted us to the shaky protection of the conventional air gap solution. The main components of a WirelessHART network are the sensors, actuators, Gateway, a network manager, and multiple access points (AP). Each communication between these devices are real-time flows with fixed periods and deadlines. To make the flows schedulable, the schedule in a WirelessHART network is predetermined by the centralized network manager at the time of network initialization. The same schedule is repeated over every hyper-period (*i.e.*, lowest common multiple of the periods of all the flows in the network), until there is any change in the network topology, such as addition/removal of new/existing devices to/from the network. The repetitive execution of the deterministic flow schedule in a WirelessHART network over every hyper-period makes these systems vulnerable to timing attacks. Such repetition greatly helps the attacker to analyze the eavesdropped traces and infer the schedule. With the inferred schedule, the attacker can further launch various strategic destructive attack steps. For instance, the attacker can selectively jam the transmissions from/to a certain critical sensor/actuator which can eventually breach the safety of the system.

In this work, we aim at reducing the predictability of the time slots in the communication schedule of a real-time WirelessHART network. We propose a moving target defense (MTD) mechanism, the *SlotSwapper*, that randomizes the time slots in the communication schedule over every hyper-period, satisfying all the feasibility constraints of a real-time WirelessHART network as follows— (1) deadlines of all real-time flows in the network are to be satisfied, (2) the hop sequences associated with each flow are to be preserved and (3) no conflicting transmissions in the network are allowed. From our analysis, the attacker who can monitor the wireless transmissions needs at least two hyper-periods to infer the schedule. Randomizing the schedule over every hyper-period renders the attacker's inference futile, thereby greatly improving the confidentiality of the WirelessHART network's operations. More varied are the slots in a schedule, more difficult it is for the attacker to predict them. Hence, the measure of uncertainty in the time slots of a schedule can be expressed in terms of the amount of randomness in the time slots over the hyper-period schedules generated by our algorithm. We re-defined *schedule entropy* [3] as a metric to measure the uncertainty in predicting the time slots. We illustrated the feasibility of our proposed algorithm on random topologies with 100 simulated nodes in Contiki Cooja [4]. To the best of our knowledge, this

is the first work on randomization to reduce the determinism of the time slots of a hyper-period schedule in real-time WirelessHART networks.

2. RELATED WORK

Two notable works in the literature which adopt randomization techniques in the context of real-time processor scheduling are *taskshuffler* [3] and SPARTA [5]. [3] presents a schedule randomization protocol, the *taskshuffler*, that shuffles a set of fixed priority real-time tasks on a uniprocessor system. [5] proposes SPARTA, a scheduler to randomize the leakage points in the schedule protecting the system from Differential Power Analysis (DPA) attacks. However, both of these works are on uniprocessor system. Our problem is even harder than multi-processor scheduling. m channels and n real-time flows of our network can be mapped to m processors and n real-time tasks respectively. However, the conflicting transmissions among the flows impose additional constraint in our network which makes our problem even harder than multi-processor scheduling.

Due to support for TDMA schedule in WirelessHART networks, these networks are vulnerable to selective jamming attacks [6]. [7,8] survey various possible jamming attacks and the key ideas of existing security mechanisms against such attacks in WSNs. [9] proposes various types of side-channel attacks and their respective countermeasures in WSN. The countermeasures against jamming attacks can be provided from physical-layer solutions as in [7, 10] or cyber-space solutions such as [11, 12]. [13] presents the steps of an attacker to launch jamming attacks in industrial process control systems. Recent works such as [14] and [15] provide countermeasures against timing attacks in single and multi-channel WSN respectively by permuting the slot utilization pattern at the node level over a super-frame to randomize the schedule. However, the flows considered in these works are not associated with deadlines, hence, randomization of slot utilization pattern at the node level makes the flows schedulable. Our problem is more complex. Each flow in our network is a real-time flow with a strict deadline. Permuting the time-slots at each node does not guarantee deadline satisfaction of all the real-time flows in our network, hence, existing solutions in [14] and [15] are not applicable.

3. WIRELESSHART BACKGROUND

The WirelessHART protocol, being compliant with IEEE 802.15.4, is the first open wireless communication standard for measurement and control in network and process industry [16]. A WirelessHART network consists of a Gateway, multiple field devices, APs and a centralized network manager which are connected via wireless mesh networks. The network manager, connected to the Gateway, is responsible for managing the devices, scheduling, creating the routes and optimizing the network. The field devices are wireless sensors and actuators which can either transmit or receive in a particular time slot. Also, in a time slot, a receiver can receive from exactly one sender. Multiple APs are connected to the Gateway via wired connections to provide redundant paths between the Gateway and the network devices. The key features of the WirelessHART network for which it is suitable for process industries include

TDMA: For reliable collision-free communications in a WirelessHART network, time is globally synchronized and slotted into 10ms time slots within which a network device sends a packet and receives its corresponding acknowledgment.

Channel and route diversity: WirelessHART supports a maximum of 16 channels [17] at a frequency band of 2.4 GHz. To avoid interference from neighboring wireless systems, it adopts channel hopping in every time slot. A channel is blacklisted if it suffers from external interference. WirelessHART allows route diversity by transmitting a packet multiple times via multiple paths over different channels.

Avoidance of spatial re-use of channels: To avoid interference and to increase reliability, WirelessHART avoids spatial re-use of channels [17]. The physical channel assigned to a link in a particular time slot is given by [17], $Ch_p = (ASN + Ch_l) \bmod m$, where ASN represents Absolute Slot Number and increases at every slot, Ch_l and Ch_p are the logical and physical channels assigned to a node, m denotes the number of channels in the network.

A WirelessHART network is represented as a graph $G = (V, E)$, where V is the set of nodes which are the sensors, actuators and Gateway; E is the set of edges or links between the devices. An edge $e = u \rightarrow v$, $u, v \in V$, is part of G, if and only if device u can reliably communicate with device v. In a transmission along an edge $u \rightarrow v$, the transmitting node, u, is the *sender* and the receiving node, v, is the *receiver* of the transmission.

Definition 1: *Two transmissions along edges $u \rightarrow v$ and $w \rightarrow x$, where $u, v, w, x \in V$, are said to be **conflicting transmissions**, if both of them have the same sender or the same receiver, i.e., if $(u = w) \vee (v = w) \vee (u = x) \vee (v = x)$. For each edge $u \rightarrow v \in E$, there exists a set of conflicting transmissions in G. To keep track of the conflicting transmissions in G, we store an adjacency list known as the **Conflict List**. Each index i in the list corresponds to an edge in E and the list corresponding to i stores the list of edges which generate conflicting transmissions with i.*

An end-to-end communication between a sensor and an actuator occurs in two phases: a *sensing phase* and a *control phase* during which the communications are between the sensors and the Gateway and between the Gateway and the actuators respectively.

4. SYSTEM MODEL

Our system model consists of a WirelessHART network $G = (V, E)$ and n end-to-end flows $\mathcal{F} = \{\mathcal{F}_1, \mathcal{F}_2, \ldots \mathcal{F}_n\}$. Each flow $\mathcal{F}_i \in \mathcal{F}$ periodically generates a packet at the source node $s_i \in V$ with period p_i. The packet passes via Gateway and reaches the destination node $d_i \in V \setminus \{s_i\}$ within deadline δ_i. We assume that our flows are of implicit deadline, *i.e.*, $\delta_i \leq p_i$. A packet is scheduled in more than one routes between the source and destination for reliability.

Definition 2: *The **release time** (r_{ij}) of the j^{th} instance of flow $\mathcal{F}_i$ ($j \geq 1$) is the time at which the j^{th} instance of $\mathcal{F}_i$ is released at the source node s_i. r_{ij} is defined as*

$$r_{ij} = (j - 1) \cdot p_i. \tag{1}$$

Definition 3: *The **number of hops** in a route of a flow $\mathcal{F}_i$ is the number of intermediate devices between the source (s_i) and the destination (d_i) in the route of $\mathcal{F}_i$.*

Definition 4: *Given a graph G with m channels and a set of flows $\mathcal{F}$, a **feasible schedule** $\mathcal{S}$ is a sequence of transmissions over the slots in $\mathcal{S}$ along the edges in G. Each transmission is a mapping of a flow to a channel in a slot satisfying the following conditions:*

1. No transmission conflict: *Two transmissions along $u \rightarrow v$ and $w \rightarrow x$ can be scheduled in the same time slot t, if $u \rightarrow v$ and $w \rightarrow x$ are non-conflicting transmissions;*

2. No collision: *If $u \rightarrow v$ uses channel y and $w \rightarrow x$ uses channel z in the same time slot t, then $y \neq z$, $\forall y, z \in [1, m]$;*

3. No deadline violation: *If a flow $\mathcal{F}_j$, $1 \leq j \leq n$, has h hops, then all the h hops of $\mathcal{F}_j$ are to be scheduled within the deadline δ_j;*

4. Flow sequence preservation: *If a flow $\mathcal{F}_j$ has h hops, then the k^{th} hop $(1 < k \leq h)$ cannot be scheduled until all the previous $k-1$ hops are scheduled.*

We assume that the network manager blacklists those channels from the network in which the probability of successful transmission is less than a certain threshold [18]. Therefore, the number of packet drops in the network can be neglected. At the time of network initialization, the network manager decides the *schedule* depending on the number of available channels, the topology of the network and available routes for each flow [17], [19]. Given a graph G, a set of n flows $\mathcal{F}$ over G and m channels, the network manager runs any scheduling algorithm $\mathbb{A}$ that generates a schedule $\mathcal{S}$ satisfying all the conditions of Definition 4. The network manager then informs all the network devices about the allocated slots in which they can transmit (receive) messages from specific neighbors. The network devices become active only in those slots in which they can transmit (receive) messages. The same schedule repeats every hyper-period.

5. THREAT MODEL

The main objective of the adversary is to select a critical sensor or an actuator as the victim node in the network and predict the time slots in which the victim node sends (receives) packets to (from) its neighboring nodes by observing the traffic in the network. Our adversary model is based on the following assumptions:-

1. The adversary is aware of the network parameters such as the number of channels adopted by the network.
2. The adversary is equipped with multiple antennae, hence, he is capable of listening to all 16 channels in 2.4 GHz ISM band in the network.

Based on the above assumptions, the adversary has the following capabilities:

Capability 1: The adversary can target a specific node (sensor or actuator) as the victim node in the network and monitor all communications associated with that node. After analyzing the traffic for a sufficiently long period of time, the adversary can predict the time slots in which the victim node communicates with its neighbors.

Capability 2: Due to repetitive nature of the communication schedule, the adversary can estimate the hyper-period of the schedule. The adversary can use this estimate in the subsequent hyper-periods to infer the communication time slots of the victim node.

Capability 3: The adversary can reverse engineer the channel hopping sequences by silently observing the channel activities in the network [20].

With the above three capabilities, the adversary can execute further destructive attack steps. For instance, the adversary can target specific transmissions from (to) certain critical sensors (actuators) and can selectively jam the targeted transmissions in specific time slots, thereby causing disruptive effect on the system. Due to repetitive nature of the hyper-period schedules, same flow gets transmitted in the same time slot over every hyper-period. Hence, selectively jamming the predicted channel in specific time slots over every hyper-period results in jamming the targeted flow with probability 1. Different from the constant jamming attack that jams all the transmissions, selective jamming is more stealthy as it allows the attacker to strategically target certain critical sensors and/or actuators within their proximity with much lower radio transmission power. This reduces the overhead and cost for the attacker to implement the jamming attack [21]. In contrast, random jamming that does not infer the schedule and jams in randomly selected slots is much less effective [22].

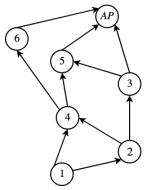


Figure 1: A network graph with six nodes and one AP

		1	2	3	4	5	6	7	8
$\mathcal{S}_1$		$1-2$	—	—	$2-3$	$4-5$	—	—	$3-AP$
	ch_1	(F_1)			(F_3)	(F_2)			(F_3)
		$4-5$	—	$5-AP$	—	$2-3$	$3-AP$	$5-AP$	—
	ch_2	(F_2)		(F_2)		(F_1)	(F_1)	(F_2)	
$\mathcal{S}_2$		—	$1-2$	—	$5-AP$	$3-AP$	$4-5$	—	$5-AP$
	ch_1		(F_1)		(F_2)	(F_1)	(F_2)		(F_2)
		$2-3$	$4-5$	$2-3$	—	—		$3-AP$	—
	ch_2	(F_3)	(F_2)	(F_1)				(F_3)	

Table 1: Two schedules $\mathcal{S}_1$ and $\mathcal{S}_2$ over 8 time slots with three flows $\mathcal{F}_1$, $\mathcal{F}_2$, $\mathcal{F}_3$ where $s_1 = 1$, $s_2 = 4$, $s_3 = 2$ and $d_1 = d_2 = d_3 = AP$.

Attack consequences: Selectively jamming the transmissions from a critical sensor node results in blocking the sensor data to reach the Gateway. As a result, proper control commands cannot be delivered to the actuators which in turn may result in degraded performance of the system. Also, selectively jamming the control commands to reach the actuators may hamper the safety of the system.

Motivation of our work: The main objective of our work is to develop a MTD technique, the *SlotSwapper*, that randomizes the communication time slots over every hyper-period schedule such that the schedule changes before the attacker can estimate it. We present a motivating example to illustrate how the threat can be addressed by randomizing the time slots in every hyper-period schedule.

Example 1: *Consider the network graph shown in Figure 1 with two channels, three flows, F_1, F_2 and F_3 where the sources are $s_1 = 1$, $s_2 = 4$, $s_3 = 2$; the destinations are $d_1 = d_2 = d_3 = AP$; the periods and the dealines are $p_1 = p_3 = \delta_1 = \delta_3 = 8$, $p_2 = \delta_2 = 4$ respectively. Consider $\mathcal{S}_1$ in Table 1 to be the hyper-period schedule over the flows. Consider node* 1 *to be the victim node. In the traditional TDMA-based real-time WirelessHART network, the network starts with schedule $\mathcal{S}_1$ which repeats every* 8 *time slots. An attacker listening to the channels in the network will find nodes* 1 *and* 2 *communicating every* 8 *time slots. In particular, to identify this repetitive pattern, the attacker needs to listen to the network for at least two hyper-periods, i.e.,* 16 *time slots. The attacker can launch selective jamming*

attack earliest in the 17^{th} *slot. With our proposed MTD technique, a new schedule is followed in each hyper-period,* i.e.*, if* $\mathcal{S}_1$ *is followed in the first eight slots, then* $\mathcal{S}_2$ *will be followed in the next eight slots and so on. However, there is no communication between nodes* 1 *and* 2 *in slot 1 in* $\mathcal{S}_2$*,* i.e.*, the communicating time slots in two consecutive hyper-periods are different. To identify the repetitive patterns in the schedule, the attacker needs to monitor the communications for at least two hyper-periods. Hence, by changing the schedule every hyper-period, the system will change at a faster pace compared to the learning pace of the attacker, rendering further strategic destructive attack steps (e.g., selective jamming) infeasible.*

6. PROPOSED MTD TECHNIQUE

Our proposed MTD technique, the *SlotSwapper*, consists of two main phases— (1) An offline schedule generation phase (2) an online schedule selection phase. *Sched_Gen*() considers an initial hyper-period schedule $\mathcal{B}$ for a set of n flows $\mathcal{F}$ over a graph G, and generates a new feasible schedule $\mathcal{S}'$ by randomizing the slots in $\mathcal{B}$. However, randomization of time slots in $\mathcal{B}$ is to be done in such a way that all the conditions of generating a *feasible schedule* (Definition 4) are obeyed. To reduce the repeatability of time slots in $\mathcal{B}$, we propose to run *Sched_Gen*() K times (K is a large number) in offline mode and generate a set of feasible hyper-period schedules $\mathbb{S}$. We suggest to select a schedule uniformly at random every hyper-period from $\mathbb{S}$ and execute that schedule over that hyper-period.

Algorithm 1: *SlotSwapper*

```
1 S = {B};// a base scehdule
2 for i=1,2 upto K do
3     S = S ∪ Sched_Gen();
4 S = Select a random schedule from S every hyper-period ;
```

Offline Randomized Schedule Generator : Algorithm 2 presents an overview of *Sched_Gen*(). Table 2 summarizes the notations used in the algorithm. We present an example to illustrate the steps of *Sched_Gen*().

G	a network graph over V nodes and $\|E\|$ edges
$\mathcal{F}$	a set of n flows defined over G
m	number of channels in the network
hp	hyper-period of n flows
$\mathcal{B}$	a base schedule consisting of mapping of a channel in a slot to a flow over one hp
C	Conflict List corresponding to the network graph G
$\mathcal{S}'$	a copy of the base schedule $\mathcal{B}$
hop_list	a dictionary to store hop number to slot mapping of all the flow instances in $\mathcal{F}$
edge_list	a dictionary to map channel to edge in a particular slot in $\mathcal{S}'$.

Table 2: List of notations used in the algorithm.

Example 2: *Consider the same setting as in Figure 1 and Example 1. Let* $\mathcal{S}_1$ *in Table 1 be the base schedule. Let us consider the* 1^{st} *hop of* $\mathcal{F}_3$ *in* $\mathcal{S}_1$ *with* $\sigma_t = 4$ *and c_ch* $= 1$*. The window corresponding to* 1^{st} *hop of* $\mathcal{F}_3$ *is* $[1, 7]$*. For every slot* $\sigma'_t \in [1, 7]$ *and every channel* $ch \in [1, 2]$*, we call* $trConf()$ *and check for conflicting transmission.* $2 \rightarrow 3$ *has conflicting transmission with* $[1 \rightarrow 2, 2 \rightarrow 4, 3 \rightarrow AP]$ *in* $\mathcal{S}_1$*. Therefore, (slot,channel) pairs such as,* $(1, 1)$*,* $(5, 2)$ *and* $(6, 2)$ *are rejected due to transmission conflict with* $(4, 1)$*. Similarly, (slot,channel) pairs such as* $(5, 1)$ *and* $(7, 2)$ *are also rejected by function* $deadPr()$ *due to violation of deadlines of the flow instances. (slot,channel) pairs* $(5, 1)$ *and* $(7, 2)$ *correspond to the second instance of* $\mathcal{F}_2$ *with release time at* 5^{th} *slot and deadline at* 8^{th} *slot. Hence, the second instance of* $\mathcal{F}_2$ *cannot be swapped with any other slot before slot* 5 *or after slot* 8*. Similarly,* $flowPr()$ *does not allow (slot,channel) pairs* $(1, 2)$*,* $(6, 2)$ *and* $(7, 2)$ *in the eligible list in order to preserve the hop sequences of flows. If the transmission corresponding to* 1^{st} *hop of* 1^{st} *instance of* $\mathcal{F}_2$ *(via edge* $4 \rightarrow 5$*) of (slot,channel) pair* $(1, 2)$ *is allowed to swap with* $(4, 1)$*, then the second hop of that instance of* $\mathcal{F}_2$ *would have been scheduled before the first hop, violating the hop sequences of the flow instances. Finally, the list of eligible (slot,channel) pairs are —* $[(2, 1), (2, 2), (3, 1), (3, 2), (4, 2), (6, 1), (7, 1)]$*. Let* $(3, 2)$ *be the randomly selected element. Swapping the transmissions and the flow instances between* $(3, 2)$ *and* $(4, 1)$ *and iterating the same procedure over all the flow instances generates a completely new feasible schedule.*

Algorithm 2: *Sched_Gen*

```
1  for tick = 1,2, ..., hp do
2      for j = 1,2, ..., |F| do
3          if tick == F_j.deadline then
4              inst = tick/F_j.deadline;
5              for p = 1,2, ...,F_j.n_hops do
6                  σ_t = slot of p^th hop of inst;
7                  elig_list = {};// empty list
8                  if m == 1 // single-channel
9                   then
10                     lb = inst * F_j.release_time;
11                     ub = inst * F_j.deadline;
12                     for σ'_t = lb, lb+1,..., ub do
13                         if S'[σ'_t] ≠ F_j then
14                             Add σ'_t to the elig_list;
15                     σ_random = random(elig_slots);
16                     swap (σ_t, σ_random);
17                 else
18                     c_ch = channel of p^th hop of inst;
19                     if p == 1// first hop
20                      then
21                         lb = inst * F_j.period;
22                     else
23                         lb = slot of (p − 1)^th hop of inst + 1;
24                     if p == F_j.n_hops // last hop
25                      then
26                         ub = inst * F_j.deadline;
27                     else
28                         ub = slot of (p + 1)^th hop of inst - 1;
29                     for σ'_t = lb, lb+1,..., ub do
30                         for ch = 1,2,...,m do
31                             b_1 = trConf(σ_t, c_ch, σ'_t, ch, C);
32                             b_2 = deadPr(σ_t, c_ch, σ'_t, ch);
33                             b_3 = flowPr(σ_t, c_ch, σ'_t, ch);
34                             if b_1 && b_2 && b3 == 1 then
35                                 Add (σ'_t, ch) to elig_list;
36                     (σ, c) = random(elig_list);
37                     swap(σ_t, c_ch, σ, c);
38                 update hop_list, edge_list and S';
39 return S';
```

Online Selection of Schedules: On executing *Sched_Gen*() K times in offline mode, we get a set of feasible schedules $\mathbb{S}$. At the time of network initialization, each node is informed about the time-slots in which it can send/receive messages in each of these K hyper-period schedules. The online schedule selector runs at each node once in every hyper-period, selects a schedule $\mathcal{S}$ from $\mathbb{S}$ uniformly at random and executes $\mathcal{S}$ over that hyper-period. To ensure that the same schedule

is selected at each node, we propose to use a pseudo-random number generator (PRNG) [23] (assumed to be secure) initialized with the same seed at each node. This allows each node to select the same schedule every hyper-period without any additional communication.

7. MEASURE OF UNCERTAINTY

Given a set of schedules $\mathbb{S}$ generated by *Sched_Gen*(), we need to quantify the amount of uncertainty in the schedules in $\mathbb{S}$. In [3], *schedule entropy* is used to measure the uncertainty of a given schedule for a uniprocessor system. We have redefined *schedule entropy* as a function of the slot and channel entropy to measure the randomness in the schedules in $\mathbb{S}$. In a multi-channel WirelessHART network, each of the slots σ_i in a schedule $\mathbb{S}$ consists of m channels which can be represented as $\sigma_i = \{c_{i1}, c_{i2}, \ldots, c_{im}\}$. Given a hyper-period schedule $\mathcal{S}$ over l slots and m channels for a set of flows $\mathcal{F}$, the occurrence of the j^{th} flow $\mathcal{F}_j$ in the k^{th} channel of i^{th} slot is a discrete random variable with possible outcomes from 0 to n, where 0 represents idle flow, n is the total number of flows in $\mathcal{F}$. Let $c_{ik} = j$ denotes the j^{th} flow occurring in the k^{th} channel of i^{th} slot of $\mathcal{S}$. However, the occurrence of the j^{th} flow in the k^{th} channel of the i^{th} slot restricts the occurrence of some other flow $\mathcal{F}'_j$ in the same channel of the same slot. Also, if a flow $\mathcal{F}_j$ completes its hops in the i^{th} slot in the schedule, it cannot occur in the subsequent slots until the arrival of its next instance. We therefore, define *Schedule entropy* as

Definition 5: ***Schedule entropy*** *over a set of flows $\mathcal{F}$ for a WirelessHART network with m channels is the conditional entropy of $\mathcal{F}_j$ occurring in the k^{th} channel of the i^{th} slot, given the entropy of all the slots from 1 to $i-1$. It is represented as*

$$H(\mathcal{S}) = \sum_{i=1}^{l} H(\sigma_i | \sigma_1, \sigma_2, \ldots, \sigma_{i-1}) \tag{2}$$

$$H(\sigma_i) = -\sum_{c_{i1}=0}^{n} \sum_{c_{i2}=0}^{n} \cdots \sum_{c_{im}=0}^{n} \Pr(c_{i1}, c_{i2}, \ldots, c_{im}) \log_2 \Pr(c_{i1}, c_{i2}, \ldots, c_{im}) \tag{3}$$

For a multi-channel WirelessHART network with n flows ($n > 16$), the number of possible permutations in the calculation of the joint probability for each slot is exponential. Hence, we consider the empirical probability distribution of the flows across all the channels in each slot which is an upper-approximated value of *slot entropy* as the joint probability is always less than or equal to the sum of individual probabilities [24]. Further, calculation of conditional entropy in Equation (2) involves joint probability distribution of slots in $\mathcal{S}$, which is exponential in nature. So, we consider the empirical probability distribution of the slots in $\mathcal{S}$.

Definition 6: ***Upper-approximated slot entropy*** $\widetilde{H(\sigma_i)}$ *and* ***Upper-approximated schedule entropy*** $\widetilde{H(\mathcal{S})}$ *are defined respectively as follows*

$$\widetilde{H(\sigma_i)} = -\sum_{k=1}^{m} \sum_{j=0}^{n} \Pr(c_{ik} = j) \log_2 \Pr(c_{ik} = j) \tag{4}$$

$$\widetilde{H(\mathcal{S})} = \sum_{i=1}^{l} \widetilde{H(\sigma_i)}. \tag{5}$$

where $\Pr(c_{ik} = j)$ is the probability mass function of the j^{th} flow occurring in the k^{th} channel of the i^{th} slot.

8. EVALUATION

Simulation setup: We use Cooja simulator [4] of Contiki 3.0 to test the feasibility of our schedules. We generated three random topologies with 100 simulated Tmote Sky motes by varying the degree of nodes (θ) or the number of incoming and outgoing edges incident on a node — (1) Graph A (θ between 2 to 4) (2) Graph B (θ between 3 to 6) (3) Graph C (θ between 3 to 8). More the degree of a node, more are the chances of conflicting transmissions and less is the number of available flows for a particular time-slot. Nodes with highest number of neighbors are considered to be the APs.

Flow Generation: A fraction (α) percent of the nodes are randomly selected as the source and destination nodes. The source and destination nodes are disjoint. In our experiments we varied α between 20-80%. We selected the number of hops of each flow to be between 2 to 8 [25] and considered the shortest path as the primary path. The flows have implicit-deadline with periods varying randomly in the range of 2^7 to 2^{10}.

Experiments: We fixed the hyper-period at 2^{10} time slots and ran experiments upto 10000 hyper-periods with the number of flows and the number of channels varying between 10 to 40 and 1 to 4 respectively. For each condition, we generated 100 random instances and measured the upper approximated schedule entropy ($\widetilde{H(\mathcal{S})}$) for each of these instances. Figure 2 shows $\widetilde{H(\mathcal{S})}$ for all the tested scenarios. It has been observed that $\widetilde{H(\mathcal{S})}$ is maximum for single-channel WirelessHART network for all three graphs. This is because in single-channel WirelessHART networks, there is no conflicting transmissions among the flows in the network. As a result, a flow can be scheduled at any slot within its release time and deadline. For a fixed number of channels, $\widetilde{H(\mathcal{S})}$ increases significantly with increase in the number of flows upto 30. After that, there is no significant increase in the value of $\widetilde{H(\mathcal{S})}$ with increase in the number of flows. This is because, with increase in the number of flows more flows can appear in a slot. However, as the number of flows increase, the number of conflicting transmissions among the flows increase which in turn restricts the number of available flows to be scheduled in a particular slot. $\widetilde{H(\mathcal{S})}$ also increases with increase in the number of channels between 2 to 4, as the number of available positions for a flow to be scheduled get increased. However, it has been observed that with increase in the number of channels, the increase in $\widetilde{H(\mathcal{S})}$ is significantly less for Graph C. Among all the three graphs, the number of edges is maximum in Graph C resulting in more conflicting transmissions among the flows thereby restricting the number of available positions to schedule a flow.

Although we ran our algorithm upto 10000 hyper-periods to measure the randomness in the generated schedules, the amount of memory available to each Tmote sky mote is not sufficiently large to store large number of schedules. We measured that each mote can only support a maximum of 2000 time slot information. We observed that, if a node is in the path of all the 40 flows, then it requires to store at-least 80 time slot information per schedule (40 for transmissions and 40 for re-transmissions). With this specification, we were able to store 25 schedules in each node. We can man-

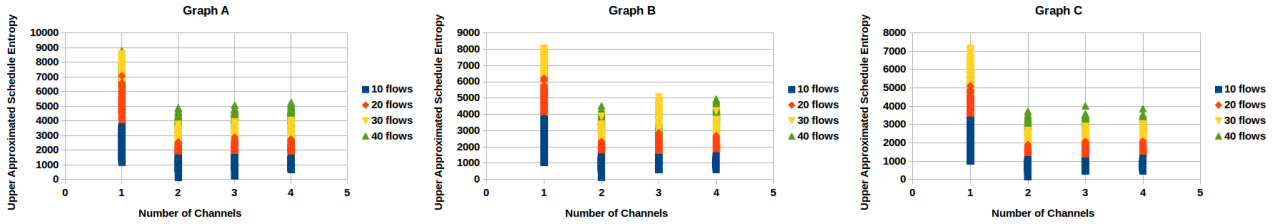


Figure 2: Upper Approximated Schedule Entropy over Graph A,Graph B and Graph C, with number of flows varying between 10 to 40 and number of channels between 1 to 4 with a hyper-period of 1024 time slots

ually tune the nodes with different sets of schedules after several hyper-periods to further reduce the chance of predicting the schedules. Our MTD technique only involves an additional random number generation in each node once in every hyper-period, the power consumption of which is negligibly small.

9. CONCLUSION

In this work, we presented an MTD mechanism, the *SlotSwapper*, to reduce the predictability of TDMA slots in a real-time WirelessHART network. We used schedule entropy to measure the uncertainty of the schedules generated by our algorithm. We illustrated the feasibility of the schedules on simulated networks in Cooja with 100 Tmote sky motes.

10. ACKNOWLEDGEMENT

This work was conducted within the Delta-NTU Corporate Lab for Cyber-Physical Systems with funding support from Delta Electronics Inc. and the National Research Foundation (NRF) Singapore under the Corp Lab@University Scheme.

11. REFERENCES

[1] Ralph Langner. Stuxnet: Dissecting a cyberwarfare weapon. *IEEE Security & Privacy*, 2011.

[2] Dragonfly: Western energy sector targeted by sophisticated attack group, 2017. https://symc.ly/2Df3VTi.

[3] Man-Ki Yoon, Sibin Mohan, Chien-Ying Chen, and Lui Sha. Taskshuffler: A schedule randomization protocol for obfuscation against timing inference attacks in real-time systems. In *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2016 IEEE*. IEEE.

[4] Fredrik Osterlind, Adam Dunkels, Joakim Eriksson, Niclas Finne, and Thiemo Voigt. Cross-level sensor network simulation with cooja. In *Local computer networks, proceedings 2006 31st IEEE conference on*. IEEE.

[5] Ke Jiang, Petru Eles, Zebo Peng, Sudipta Chattopadhyay, and Lejla Batina. Sparta: A scheduling policy for thwarting differential power analysis attacks. In *Design Automation Conference (ASP-DAC), 2016 21st Asia and South Pacific*. IEEE.

[6] Alejandro Proano and Loukas Lazos. Selective jamming attacks in wireless networks. In *Communications (ICC), 2010 IEEE International Conference on*. IEEE.

[7] Aristides Mpitziopoulos, Damianos Gavalas, Charalampos Konstantopoulos, and Grammati Pantziou. A survey on jamming attacks and countermeasures in wsns. *IEEE Communications Surveys & Tutorials*, 2009.

[8] Deepali Virmani, Ankita Soni, Shringarica Chandel, and Manas Hemrajani. Routing attacks in wireless sensor networks: A survey. *arXiv preprint arXiv:1407.3987*, 2014.

[9] Kanthakumar Pongaliur, Zubin Abraham, Alex X Liu, Li Xiao, and Leo Kempel. Securing sensor nodes against side channel attacks. In *High Assurance Systems Engineering Symposium, 2008. HASE 2008. 11th IEEE*. IEEE.

[10] Raymond Pickholtz, Donald Schilling, and Laurence Milstein. Theory of spread-spectrum communications–a tutorial. *IEEE transactions on Communications*, 1982.

[11] Alejandro Proano and Loukas Lazos. Packet-hiding methods for preventing selective jamming attacks. *IEEE Transactions on dependable and secure computing*, 2012.

[12] Anthony D Wood, John A Stankovic, and Gang Zhou. Deejam: Defeating energy-efficient jamming in ieee 802.15. 4-based wireless networks. In *Sensor, Mesh and Ad Hoc Communications and Networks, 2007. SECON'07. 4th Annual IEEE Communications Society Conference on*. IEEE.

[13] Spase Stojanovski and Andrea Kulakov. Efficient attacks in industrial wireless sensor networks. In *ICT Innovations 2014*. Springer.

[14] Marco Tiloca, Domenico De Guglielmo, Gianluca Dini, Giuseppe Anastasi, and Sajal K Das. Jammy: a distributed and dynamic solution to selective jamming attack in tdma wsns. *IEEE Transactions on Dependable and Secure Computing*, 2017.

[15] Marco Tiloca, Domenico De Guglielmo, Gianluca Dini, Giuseppe Anastasi, and Sajal K Das. Dish: Distributed shuffling against selective jamming attack in ieee 802.15. 4e tsch networks. *ACM Transactions on Sensor Networks (TOSN)*, 2018.

[16] Chenyang Lu, Abusayeed Saifullah, Bo Li, Mo Sha, Humberto Gonzalez, Dolvara Gunatilaka, Chengjie Wu, Lanshun Nie, and Yixin Chen. Real-time wireless sensor-actuator networks for industrial cyber-physical systems. *Proceedings of the IEEE*, 2016.

[17] Deji Chen, Mark Nixon, and Aloysius Mok. *WirelessHART: Real-Time Mesh Network for Industrial Automation*. Springer Publishing Company, Incorporated, 2010.

[18] Jianping Song, Song Han, Al Mok, Deji Chen, Mike Lucas, Mark Nixon, and Wally Pratt. Wirelesshart: Applying wireless technology in real-time industrial process control. In *IEEE real-time and embedded technology and applications symposium*. IEEE, 2008.

[19] Jianping Song, Song Han, Xiuming Zhu, Aloysius K Mok, Deji Chen, and Mark Nixon. A complete wirelesshart network. In *Proceedings of the 6th ACM conference on Embedded network sensor systems*. ACM, 2008.

[20] Xia Cheng, Junyang Shi, and Mo Sha. Cracking the channel hopping sequences in ieee 802.15.4e-based industrial tsch networks. 2019.

[21] Kanika Grover, Alvin Lim, and Qing Yang. Jamming and anti-jamming techniques in wireless networks: a survey. *International Journal of Ad Hoc and Ubiquitous Computing*, 2014.

[22] Wenyuan Xu, Wade Trappe, Yanyong Zhang, and Timothy Wood. The feasibility of launching and detecting jamming attacks in wireless networks. In *Proceedings of the 6th ACM international symposium on Mobile ad hoc networking and computing*. ACM, 2005.

[23] Wikipedia contributors. Pseudorandom number generator — Wikipedia, the free encyclopedia, 2019.

[24] Claude Elwood Shannon. A mathematical theory of communication. *ACM SIGMOBILE mobile computing and communications review*, 2001.

[25] Rajeev Alur, Alessandro D'Innocenzo, Karl H Johansson, George J Pappas, and Gera Weiss. Modeling and analysis of multi-hop control networks. In *Real-Time and Embedded Technology and Applications Symposium, 2009. RTAS 2009. 15th IEEE*. IEEE, 2009.