# Byzantine Resilient Protocol for the IoT

Antônio A. Fröhlich, Roberto M. Scheffel, David Kozhaya and Paulo E. Veríssimo

***Abstract*—Wireless sensor networks, often adhering to a single gateway architecture, constitute the communication backbone for many modern cyber-physical systems. Consequently, fault-tolerance in CPS becomes a challenging task, especially when accounting for failures (potentially malicious) that incapacitate the gateway or disrupt the nodes-gateway communication, not to mention the energy, timeliness, and security constraints demanded by CPS domains. This paper aims at ameliorating the fault-tolerance of WSN based CPS to increase system and data availability. To this end, we propose a replicated gateway architecture augmented with energy-efficient real-time Byzantine-resilient data communication protocols. At the sensors level, we introduce FT-TSTP, a geographic routing protocol capable of delivering messages in an energy-efficient and timely manner to multiple gateways, even in the presence of voids caused by faulty and malicious sensor nodes. At the gateway level, we propose a multi-gateway synchronization protocol, which we call *ByzCast*, that delivers timely correct data to CPS applications, despite the failure or maliciousness of a number of gateways. We show, through extensive simulations, that our protocols provide better system robustness yielding an increased system and data availability while meeting CPS energy, timeliness, and security demands.**

***Index Terms*—Fault Tolerance. Algorithm/protocol design and analysis. Routing Protocols. Wireless Sensor Networks.**

## I. Introduction

More and more Cyber-Physical Systems (CPS) are being interconnected, particularly in the realm of Wireless Sensor Networks (WSN), Industry 4.0, and the Internet of Things (IoT). However, due to cost constraints these systems are often being built around old Internet technology that was not designed considering requirements such as timeliness, positioning, security, fault-tolerance and trustfulness, which are essential for the CPS domain. In this paper, we address the system and data availability problem relevant to CPS built on WSN. We aim at providing higher data availability for CPS applications by specifically enhancing fault and intrusion tolerance of WSN architectures. We elaborate further on this in what follows.

WSN architectures can exhibit failures of benign or malicious nature, occurring either at the level of individual devices (e.g. sensor, actuator, machine) or at the level of gateways that connect such devices to the traditional IT infrastructure (e.g. servers, Cloud, Internet). For example, many battery-operated sensors in a WSN are deployed in areas that are physically hard to reach and maintain. As a result, many sensors and actuators e.g., in a smart building, are often left unattended. Moreover, many devices in an industrial environment are subjected to extreme conditions. Consequently, such devices are subject to failures, and even some are likely to be exploited by motivated attackers. Similar threats can also jeopardize gateways. In fact, gateways are typically implemented on computers running an ordinary operating system such as Linux and they are usually connected to the Internet. Hence, a gateway is at an even higher risk of being attacked compared to individual devices.

Adding to that, typical WSN architectures, on top of which CPS applications are established, suffer from centralization. Namely, such architectures often rely on a single gateway (or sink) that connects all system devices or part of them to the IT infrastructure. A single gateway is not only a performance bottleneck but more importantly is a single point of failure that hinders the system's availability. A failure or a spurious behavior of this gateway relative to malicious attacks may compromise the data from the entire sensor network.

In this paper we address the above threats by presenting a fully established solution, encompassing architectural and algorithmic contributions. First, we propose redundancy at the deployment level of both sensor nodes (devices in general) as well as gateways/sinks[1]. Hence, we favor increased distribution of all network components to eliminate centralization.

Previous attempts, like [1], [2], [3], [4], have also suggested using multiple gateways, to achieve performance scalability, minimized energy consumption and to deal with network/gateway failures. However, in the proposed solutions, messages are delivered to just **one** of the available gateways, for example considering the route that provides better energy balance or the fastest delivery. In case of communication interruptions or gateway unavailability these solutions re-route their traffic to an alternative gateway.

In brief, these solutions fall short in terms of resiliency: they do not provide any data guarantees when a gateway is compromised by an attacker, in which case an attacker can tamper data before delivering it to the application. Our objective, unlike these existing solutions, is to enhance the system's resiliency to faults and intrusions which yields in turn better system and data availability.

In order to provide system robustness and resilience, we devise protocols that makes use of our proposed sensor and gateway architectural redundancy. First, at the level of the WSN[2] devices (sensor nodes, actuators, etc.), we devise a rout-

A. A. Fröhlich and R. M. Scheffel where with the Software/Hardware Integration Lab, Federal University of Santa Catarina, Florianópolis, SC, Brazil. E-mail: {guto,scheffel}@lisha.ufsc.br

D. Kozhaya and P. E. Veríssimo where with the Interdisciplinary Centre for Security, Reliability and Trust - University of Luxembourg, Luxembourg. E-mail: david.kozhaya@gmail.com, paulo.verissimo@uni.lu

[1] We use the terms sink and gateway interchangeably in this paper, designating a particular kind of network node that is connected at the same time to both the device's network and the IT infrastructure network.

[2] We use the term WSN to designate any kind of device's network, including industrial networks and the portion of the IoT concerning CPS.

ing algorithm, which we call *Fault-Tolerant Trustful Space-Time Protocol* (FT-TSTP), that uses SmartData to achieve data transfer resiliency to black hole attacks and to voids formed by malfunctioning or displaced nodes in the WSN. FT-TSTP utilizes geographic routing to forward packets towards multiple sinks, without relying on route building techniques, announcement packets, or routing tables in the node's memory. Our FT-TSTP algorithm builds on top of the Trustful Space-Time Protocol (TSTP) [5] that achieves energy-efficient and timely data transfers in single-gateway WSN architectures.

Second at level of gateways, which might receive different data, we devise an intrusion-tolerant synchronization protocol, which we call *ByzCast*, that allows correct and timely actuation signals to be dispatched to actuators despite having some gateways crashed or even compromised by malicious attackers.

We illustrate the effectiveness of our solution through extensive simulations, showcasing that our approach indeed provides high data availability and reliability. Compared to TSTP, a single-sink geographic routing protocol that loses many messages in the presence of certain network voids, our results show that FT-TSTP can circumvent such limitations and achieve high delivery rates to a number of sinks.

In short the main contributions of this paper are:

1) Architectural and algorithmic solutions for achieving better fault and intrusion tolerance in WSN networks.
2) FT-TSTP, a routing algorithm based on SmartData that transfers data between WSN devices and gateways in an energy-efficient and timely manner. FT-TSTP can tolerate black holes attacks as well as voids formed by malfunctioning or displaced nodes.
3) ByzCast, an intrusion-tolerant protocol of synchronizing data received by different gateways.
4) Extensive simulations illustrating the increased data availability achieved by our solution.

The rest of the paper is organized as follows. Section II presents background information explaining the concept of SmartData and the TSTP protocol. Section III presents our FT-TSTP protocol, which implements multi-sink message delivery and void detouring. We also present *ByzCast*, our inter-gateway synchronization algorithm. In Section IV we discuss how the proposed protocols provide resiliency against different faults and attacks. Section V presents our simulations evaluating the transmission delay and energy consumption overhead of our protocols. Finally, Section VI discusses related work and Section VII concludes the paper.

## II. SmartData and Trustful Space-Time Protocol

In this section, we describe the Trustful Space-Time Protocol (TSTP) [5] and the *SmartData* construct [6] that lay the basis for the ideas proposed in this paper. TSTP is an application-oriented, cross-layer communication protocol for CPSs on a WSN or on the IoT. TSTP delivers trusted, timed, geo-referenced, SI-compliant data to applications through the *SmartData* construct, which promotes a data-centric view of the network. We first start by explaining what *SmartData* is and what are the different parts constituting it.

### A. SmartData: Semantics, Interface, and Security

*SmartData* can be viewed as a piece of information, which is enriched with enough metadata to become self-contained in terms of semantics, spatial location, timing, and trustfulness. *SmartData* is meant to be the only application-visible construct in the sensing/actuating platform. Therefore, it implicitly mediates all system-level services, including communication, synchronization, scheduling and the interaction with transducers and actuators. An application can read a sensor simply by accessing the information contained within the SmartData, via an interface. The information within the SmartData is automatically updated from the network according to the parameters specified at instantiation time. Actuation happens through the same interface: changing a SmartData causes messages to be propagated over the network to command actuators accordingly.

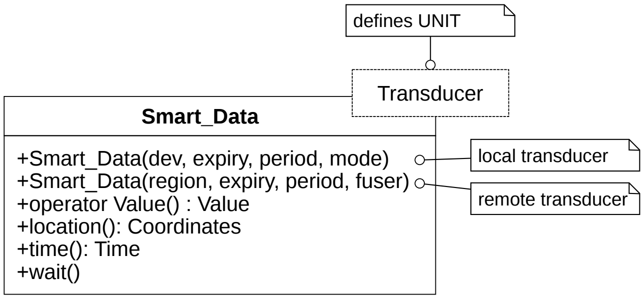

Figure 1. SmartData interface.

Figure 1 depicts the SmartData interface. Instances of SmartData can abstract local transducers, as well as create local proxies of remote transducers. In either case, the binding of a SmartData object with the corresponding transducer is done via the `Transducer` class parameter. The constant named `UNIT` is used to personify the corresponding SI quantity (a 32-bit type identifier [7]).

The initial value of parameter `expiry` is refreshed on every SmartData update, so it is taken as a relative value, just like the `period` parameter. Specifying an expiry smaller than the period is invalid. The parameter `mode` designates SmartData's visibility: *Private* to the process that created it, inaccessible from the network; *Advertised* on the network for remote sensing; or *Commanded* remotely over the network, which is used by actuators. The `region` parameter, used for remote transducers, specifies a Space-Time region of interest for a given SI Quantity as $Space\text{-}Time\ Region(x, y, z, r, t_0, t_f)$, where $x$, $y$, $z$ designates the center of the interest region, $r$ designates its radius, $[t_0, t_f]$ designates the interest's time interval. SmartData objects (created with the first constructor) matching the criteria will reply to the interest in accordance with parameters `period` and `fuser`.

Figure 2 depicts a SmartData from the perspective of the communication protocols, encapsulated in a network packet. Such packet carries includes spatial coordinates where the data was produced (`Origin(x,y,z)`), a high-resolution timestamp identifying when data was produced (`Origin(t)`), the validity of the data expressed as an absolute timestamp (`Expiry`), and a Message Authentication Code (`MAC`). SmartData are signed and encrypted using the Poly1305-

| Origin (x,y,z,t) | Unit | Value | Expiry | MAC |
|---|---|---|---|---|

Figure 2. SmartData encapsulated in a network packet.

AES [8] cryptographic MAC. Explicit error-detecting codes are not needed, as corrupted messages simply fail decryption. The key used in this algorithm is derived using Elliptic Curve Diffie–Hellman (ECDH) [9]. The precisely synchronized clocks used to timestamp data can be used along with both algorithms: for Poly1305-AES, timestamps can serve as one of the input parameters; for ECDH, they can define a narrow time window for sensor deployment. Individual node serial numbers or UUIDs also serve as input to ECDH.

This symmetric cipher scheme is meant to be used only in the WSN. Therefore, SmartData objects must be reciphered at the gateways, so the keys used in the WSN do not have to be shared with the Cloud. Gateways keep logs of every symmetric key they have used to communicate with nodes.Thus, it is possible to audit a SmartData from the Cloud, presenting it and its MAC to the gateway, which recalculates the MAC using the received $Data\ Point$, the key $K$ and the node's $Id$ and compares the result with the provided MAC.

### *B. Trustful Space-Time Protocol (TSTP)*

*1) TSTP Space and Time Synchronization:* SmartData relies on nodes being synchronized in time and in space. TSTP implements the mechanisms needed for these synchronizations relying on its cross-layer design and on some control information carried within its messages (see Figure 3). TSTP achieves time synchronization via the *Speculative Precision Time Protocol* [10], which achieves sub-microsecond precision on a IEEE 802.15.4 WSN. Space synchronization within TSTP is achieved using a speculative version of the *Heuristic Cooperative Calibration Positioning System* (HeCoPs) [11].

*2) TSTP Medium Access Control and Routing:* TSTP MAC [12] is the component within TSTP responsible for interfacing the protocol with the physical network in an energy-aware manner. Its design follows the general principles of RB-MAC [13]: a long preamble composed of *microframes* (see figure 4) is sent before each message, such that just one sender occupies the channel at every full period $S$; sensor nodes sleep for most of the time and, when they receive a message, nodes closer to the destination become relay candidates, using their distances to the destination to derive the time offset $\delta(m)$ for *Clear Channel Assessment* (CCA) and retransmission. The relay candidate closest to the destination accesses the channel earlier and wins the contention, resulting in a greedy, fully-reactive, geographic routing.

Messages being routed by node $i$ are kept in a queue $Q_i$. Each entry $e_Q \in Q_i$ represents a message $m$ that is scheduled for transmission or retransmission. In addition to the message, $e_Q$ also holds $m$'s *Id* (extracted from the microframes), its *Expiry* $T_\varepsilon$ (extracted from its header), its *Destination* and the message's offset $\delta(m)$ previously calculated.

TSTP uses implicit acknowledgment (ACK) to confirm the routing of messages. A node $i$ *only* removes a message $m$ from its queue $Q_i$ when another message $m\prime$ with the same *Id* is overheard in the network, transmitted by a node that is closer to the destination. An explicit ACK is only used is when the message reaches its final destination: that node must retransmit the same message (`Last Hop` = destination), just to acknowledge the last forwarder and any neighbors that might still have that message queued. TSTP considers that an unacknowledged message either suffered a collision or reached a geographic void. To handle this, the traditional random exponential backoff scheme is used, retransmitting the message until an ACK is received or the message expires.

TSTP's transmission time offset $\delta(m)$ is made sensitive to other routing metrics by *distorting* space. A node running out of memory or consuming too much energy can *stretch* space, increasing its distance to the destination so that other nodes become more likely to win the contention to retransmit a message. Conversely, a node with a message close to expiring can *shrink* space, increasing the chances of winning the contention and transmit it earlier. This distortion can be introduced by coefficient $\alpha \in [0, 1]$ that multiplies $\delta(m)$ and defines how much any other metric influences the perceived distance, and the offset used for contention. A value of $\alpha = 0.5$ produces an offset equivalent to the real distance, an $\alpha \in [0, 0.5)$ shrinks space, while an $\alpha \in (0.5, 1]$ stretches space.

For the discussions in this paper, the *Expiry* of messages (see Figure 3) is the metric associated with the distortion coefficient $\alpha$, expediting messages which are closer to expire. Noticeably, the timeliness of the whole WSN requires careful planing and deployment, specially on dense networks, with many sensors and high sampling rates. The queue $Q_i$ is also sorted by the slack time of messages in this scheme.

*3) TSTP Greedy Forwarding Algorithm:* To prevent the distortion coefficient from causing messages to be forwarded to an incorrect destination, the TSTP Greedy Forwarding Algorithm (Algorithm 1) ensures that all messages queued on a node for transmission satisfy the Progress Property: there must be a positive spatial progress towards the destination. This property can be written as $\forall j \forall i \{m_j i \in Q_i | D_i < D_j\}$, meaning that each message $m_j i$ from node $j$ overheard by node $i$ will be stored in node $i$'s transmission queue $Q_i$ if and only if the distance $D_i$ from node $i$ to the message's destination is smaller than the distance $D_j$ from node $j$ to the same destination. The property is implemented by Algorithm 1, which handles four possible cases:

- Case 1: If $m$ is already enqueued (line 4) and is coming from a node closer to the destination (line 6), then this means that $m$ has already made positive progress to the destination. Therefore, $m$ is an ACK, and $e_Q$ is removed from $Q_i$ (line 7) while $m$ is simply discarded (line 9).
- Case 2: If $m$ is already enqueued (line 4) and $m$ is coming from a node farther from the destination (line 6), then $m$ is a retransmission attempt and can be ignored. The local copy of $m$ in $e_Q$ remains in $Q_i$ for later retransmission.
- Case 3: If $m$ is a new message (line 13) that came from a node more distant from the destination (line 14), then this node becomes a relay candidate for $m$. Therefore, $m$ is inserted into $Q_i$ (creating an $e_Q$ - line 15).
- Case 4: If $m$ is a new message (line 13) that came from a node closer to the destination (line 14), then it means

| Bits: | 3 | 1 | 2 | 2 | 8 | 3*sb | 64 | 3*sb + tb | 64 | 0 or 32 |
|---|---|---|---|---|---|---|---|---|---|---|
| | Message Type | Time Request | Spatial Scale | Temporal Scale | Location Confidence | Last Hop $(x, y, z)$ | Last Hop Timestamp | Origin $(x, y, z, t)$ | Expiry | Location Deviation |

Figure 3. TSTP message header format.

| Bits: | 1 | 11 | 12 | 32 | 16 |
|---|---|---|---|---|---|
| | All Listen | Count | Id | Distance | CRC |

Figure 4. TSTP microframe format.

**Algorithm 1** TSTP Greedy Forwarding

```
1: procedure GreedyForwardToSink($m$)
2:     $queued \leftarrow false$
3:     for each $e_Q \in Q_i$ do
4:         if $m.id = e_Q.id$ then
5:             $queued \leftarrow true$
6:             if $Distance(m.LastHop, m.Destination) \leq$
                   $Distance(here(), m.Destination)$ then
7:                 $Q_i$.remove($e_Q$)      // $m$ made progress
8:             end if
9:             delete $m$               // $m$ is a retransmission
10:                                     // from farther away
11:        end if
12:    end for
13:    if $queued = false$ then
14:        if $Distance(m.LastHop, m.Destination) >$
               $Distance(here(), m.Destination)$ then
15:            $Q_i$.insert($m$)           // enqueue $m$ for relay
16:        else
17:            delete $m$               // $m$ already made progress
18:        end if
19:    end if
20: end procedure
```

that the current node would not make positive progress towards the destination. The message is ignored (line 17).

## III. Byzantine Resilient SmartData

In this section, we describe our approach for increasing the robustness as well as the resiliency of CPS built around the SmartData concept. As explained in Section I, these systems can exhibit failures of benign or malicious nature occurring either at the level of individual devices in the WSN or at the level of gateways that interface the WSN with the Internet. For the discussion in this Section, we consider that gateways run the network-wide CPS applications. We address these threats by an architectural enhancement that deploys redundancy at both levels, devices and gateways, and by a protocol that makes use of this architectural redundancy to provide system robustness and resilience. In this way, we guarantee that whenever an application receives a SmartData, the following properties are preserved:

1) **Confidentiality:** SmartData are not disclosed to unauthorized nodes while they are being transported.
2) **Integrity:** SmartData are not modified while they are being transported;
3) **Authenticity:** The origin of SmartData can be confirmed.
4) **Non-repudiation:** A node cannot repudiate the SmartData it has previously produced.
5) **No duplication:** SmartData updates are delivered to applications only once.
6) **Availability:** An application that declares interest for some SmartData, eventually receives them.
7) **Agreement:** If a SmartData update is propagated to an application running on any correct gateway, then every correct gateway eventually propagates the same update. This update (e.g. an actuation command) only reaches the target region of space-time if at least $N$ gateways perform the same update.
8) **Timeliness:** SmartData are delivered to applications before they expire.

To provide the properties above, we designed, implemented and evaluated a fault-tolerant version of TSTP for the WSN, which we call *Fault-Tolerant Trustful Space-Time Protocol* (FT-TSTP), augmented with a reliable broadcast protocol for the gateways' network. Our algorithms adhere to the interface and the semantics defined by the SmartData construct. Thus, transporting SmartData using our protocols does not require any modifications on applications. Indeed, some of these properties are already ensured by the original TSTP as part of the support to SmartData. The Poly1305-AES required by SmartData ensures Confidentiality, Integrity, Authenticity, and Non-repudiation. Therefore, this work focuses preserving these three properties and on providing in addition *No duplication*, *Availability* and *Timeliness*.

A crashed (Byzantine) node can behave in an arbitrary manner. For example, it can refuse to forward messages, compromising Availability, duplicate a message to promote a *Replay* attack, and can even artificially inject messages on the network, limiting the bandwidth and energy available for useful communication in a sort of *Distributed Denial-of-Service* (DDoS) attack. The subsequent sections describe the design of our protocols, explaining how we handle these kinds of attacks.

### A. Fault-Tolerant Trustful Space-Time Protocol (FT-TSTP)

FT-TSTP is a communication protocol that transports data from devices in the WSN (e.g. sensors, actuators, machines) to gateways connected to an IT infrastructure. FT-TSTP is Byzantine-resilient, i.e., it can transport data despite device and gateway failures (benign and malicious). Redundancy of nodes in a WSN is a major strategy to achieve high data availability and FT-TSTP certainly also depends on node redundancy in this sense. Nevertheless, node redundancy is not enough if the data produced on the WSN is meant to be used for network-wide CPS applications running on gateways or in the Cloud or over the Internet. WSN sinks (or IoT gateways) must also be replicated to avoid being single points of failure. Therefore, FT-TSTP assumes multiple sinks and defines a novel algorithm to forward messages containing SmartData.

FT-TSTP's *Multi-Sink Greedy Forwarding* Algorithm (2) requires a bootstrapping slightly different from that of the original TSTP. Instead of providing newcomer nodes with the coordinates of a single sink, an array of sinks $S$ is given. The number of sinks in this array is configurable and depends on the desired level of fault tolerance. For example at least $3f+1$ sinks are needed to be able to tolerate $f$ compromised sinks; $f$ designates the maximum number of faulty (Byzantine) sinks that can be tolerated while still being able to reach any form of agreement on the data delivered by sinks [14]. The sinks are ordered at deployment-time and that order is not modified during a cycle of operation, so all nodes agree on the order of the sinks in the array.

Besides modifying the bootstrap procedures, we also modified the format of microframes and messages. Microframes now carry the coordinates of the last hop, and therefore messages no longer need to carry the distance from the last hop to the sink. They now also feature a bitmap (`Sinks`) designating the sinks to which the following message is to be delivered to. This bitmap has one bit for each of the $N_S$ sinks defined at deployment-time[3]. The resulting formats are depicted in Figures 6 and 5.

With these modifications, Algorithm 2 is able to implement a greedy, fully-reactive, geographic routing policy similar to the one in the original TSTP, but with multiple destination sinks. Each node that hears any of the microframes that precedes a message $m$ decides whether or not to wake up to listen to it after calculating its distance to ALL the destination sinks whose bits in `Sinks` are set. If it can make the message progress toward ANY of those sinks, then it wakes up to receive $m$ and becomes one of its potential relay nodes. The queues of messages waiting to be forwarded by each node have now a second dimension, corresponding to each of the sinks $\in S$ and is designated $Q_i^s$. When a new message $m$ is produced, its `Sinks` attribute is initialized at the origin node with all bits set, so Algorithm 2 (line 2) initially tries to forward the message to all sinks. Subsequently, relay nodes only include $m$ in the $Q_i^s$ corresponding to the sink for which they are relaying the message (lines 35-42), clearing the bits of $m$.`Sinks` associated with the other sinks. This way, messages get forwarded directionally, avoiding flooding the network with unnecessary replicas of $m$. The behavior of the `All Listen` bit is unaltered, so control messages and messages whose destination is not a sink are routed using Algorithm 1.

In order to avoid bounces, which may occur when a previous relay node hears a messaged that was already forwarded to a sink in the past and therefore was removed from the corresponding $Q_i^s$ queue, a second bidimensional queue of recently forwarded messages, $F_i^s$, is kept at each node $i$. Messages are kept in this queue until they expire. These queues are updated at the beginning of each transmission cycle by Algorithm 3.

**Algorithm 2** FT-TSTP Multi-Sink Greedy Forwarding

```
 1: procedure MultiSink_Greedy_Forward($m$)
 2:     for each $s \in S \cap m.Sinks$ do
 3:         $queued \leftarrow false$
 4:         $isNew \leftarrow true$
 5:         $toACK \leftarrow false$
 6:         for each $e_F \in F_i^s$ do
 7:             if $e_F.id = m.id$ and $e_F.Origin = m.Origin$ then
 8:                 if $m.LastHop = \infty$ and $e_F.LastHop \neq \infty$ then
 9:                     // $m$ entered recovery mode and must be
10:                     // removed from $F_i^s$ and reinserted
11:                     // into $Q_i^s$ (line 36)
12:                     delete $F_i^s$.remove($e_F$)
13:                     $isNew \leftarrow false$
14:                 else
15:                     // $m$ was already relayed
16:                     $queued \leftarrow true$
17:                     if $m.LastHop \neq \infty$ then
18:                         $A_i^s$.insert(m)
19:                         $toACK \leftarrow true$
20:                     end if
21:                 end if
22:             end if
23:         end for
24:         if not $queued$ then
25:             for each $e_Q \in Q_i^s$ do
26:                 if $e_Q.id = m.id$ and $e_Q.Origin = m.Origin$ then
27:                     // $m$ is being relayed
28:                     $queued \leftarrow true$
29:                     if distance($m.LastHop, s$) $\leq$ distance($here(), s$)
                          then
30:                         // $m$ already made progress in this direction
31:                         $F_i^s$.insert($Q_i^s$.remove($e_Q$))
32:                     end if
33:                 end if
34:             end for
35:         end if
36:         if not $queued$ and distance($m.LastHop, s$) >
             distance($here(), s$) then
37:             // enqueue $m$ for relay in this direction
38:             if $m.LastHop \neq \infty$ or isNew then
39:                 // $m$ is not in recovery mode
40:                 $m.LastHop \leftarrow$ here()
41:             end if
42:             $m.Retries = \beta$
43:             $Q_i^s$.insert($m$)
44:         else
45:             // $m$ will not make progress in this direction
46:             // or is already in $Q_i$ or $F_i$
47:             if not $toACK$ then
48:                 delete $m$
49:             end if
50:         end if
51:     end for
52: end procedure
```

TSTP does not handle geographic voids. A message $m$ is retransmitted towards its destinations until it expires. Voids can result from poor sensors placement, from sensor failures, or from sensor nodes that get compromised and refuse to forward messages. Therefore, in order to achieve the availability property, FT-TSTP must handle voids natively. The acknowledgement mechanism in Algorithm 2 enables voids to be easily detected by Algorithm 3: If $m$ is not overheard after a certain number of transmission slots ($m$.`Retries`, Algorithm 3, lines 23-24), then it is safe to assume that node $i$ has no (sane) neighbors that are closer to the destination than itself. The $\beta$ coefficient is also applied in this sense, so $m$ is expected to be acknowledged (i.e. retransmitted) by a neighbor in each of the designated directions after $\beta$ transmissions.

If a message $m$ is not acknowledged after $m$.`Retries` retransmissions (Algorithm 3, lines 25-37), then the recovery

[3] For alignment reasons, the field `Sinks` is padded to have a size that is multiple of 8.

| Bits: 3 | 1 | 2 | 8 | 64 | 3**sb* + *tb* | 64 | 0 or 32 |
|---|---|---|---|---|---|---|---|
| Message Type | Time Request | Temporal Scale | Location Confidence | Last Hop Timestamp | Origin (x,y,z,t) | Expiry | Location Deviation |

Figure 5. FT-TSTP message header format (the number of bits in spatial (*sb*) and temporal (*tb*) coordinates are defined by the `Spatial Scale` field in the microframe and the `Temporal Scale` field).

| Bits: 1 | 11 | 12 | $N_S$ | 2 | 3**sb* | 16 |
|---|---|---|---|---|---|---|
| All Listen | Count | Id | Sinks | Spatial Scale | Last Hop (x,y,z) | CRC |

Figure 6. FT-TSTP microframe format (the number of bits in spatial coordinates (*sb*) is defined by the `Spatial Scale` field.

routing mode is enabled by making $m.LastHop \leftarrow \infty$ and reseting the retransmission counter (lines 30-32). By putting the last hop position at $\infty$, the node triggers a reverse flooding routing strategy, since it will always be more distant to the destination than any other node. This reverse routing is exited when a node that has not heard $m$ before (i.e. the conditions of lines 7 and 25 of Algorithm 2 do not verify) restores a real last hop (Algorithm 2, line 40).

**Algorithm 3** FT-TSTP Multi-Sink Queue Update

```
 1: procedure Update_Queues
 2:     for each $s \in S$ do
 3:         for each $e_Q \in Q_i^s$ do
 4:             if $(e_Q.Expiry - \tau(e_Q)) <$ now() then
 5:                 // $e_Q$ expired without making progress
 6:                 // in this direction
 7:                 $m \leftarrow Q_i^s$.remove($e_Q$)
 8:                 delete $m$
 9:             end if
10:         end for
11:         for each $e_F \in F_i^s$ do
12:             if $e_F.Expiry <$ now() then
13:                 // $e_F$ was relayed long ago and can be forgotten
14:                 $m \leftarrow F_i^s$.remove($e_F$)
15:                 delete m
16:             end if
17:         end for
18:     end for
19:     $done \leftarrow false$
20:     while $|Q_i| \neq 0$ and not $done$ do
21:         // the element at the head of $Q_i$ is the next to be transmitted
22:         $s \leftarrow Q_i$.next_to_expire()
23:         $m \leftarrow Q_i^s$.head()
24:         $m.Retries \leftarrow m.Retries - 1$
25:         if $m.Retries < 0$ then
26:             if $m.LastHop = \infty$ then
27:                 $m \leftarrow Q_i^s$.remove($m$)
28:                 $F_i^s$.insert($m$)
29:             else
30:                 // $m$ in $e_Q$ reached a void, set recovery mode
31:                 $m.LastHop \leftarrow \infty$
32:                 $m.Retries = \beta - 1$
33:                 $done \leftarrow true$
34:             end if
35:         else
36:             $done \leftarrow true$
37:         end if
38:     end while
39: end procedure
```

*False voids* can be detected if a message takes two different paths simultaneously, e.g., on the edge of a previous void. The message will later converge to nodes close to a sink, and the last arriving copy will be ignored, as the algorithm doesn't forward previously seen messages. The lack of an implicit ACK will initiate the void detour algorithm, making the message be retransmitted in recovery mode and start an unnecessary reverse flooding. To handle this, when a node receives a message already forwarded, the preamble of microframes will be transmitted in order to send the ACK. For that, the message is inserted in another queue $A_i$ (Algorithm 2, line 16-18), that must be processed in the transmission cycle. This is an exception to the rule of TSTP to not have explicit ACK, except for the sink nodes.

### *B. A Byzantine Reliable Broadcast Protocol*

Having only a real-time Byzantine-resilient protocol that transports data from WSN devices to gateways is not sufficient. First, gateways might not receive the same set of messages, as messages can be lost. Second, gateways themselves possess the keys relative to all WSN devices. Thus, a gateway that is compromised by an attacker can fake messages relative to any sensor node. Such a compromised gateway can issue as well inconsistent actuation signals that render system behavior incorrect and sometimes even hazardous.

We consider gateways to be connected by a separate network, different than the WSN one. If a sink sends a message pretending to be some sensor node (this is possible since a gateways has the private key of all sensor nodes), it can do so in two ways, by disseminating the information over the WSN or by sending this information over gateway network. If it does so over the WSN other sensor nodes detect this false personalization (identity theft) and do not propagate that information further. While if sent over the gateway network, then we need a protocol that prevents correct gateways from delivering such values.

In short, an inter-gateway protocol is needed to ensure two main things: (i) all correct gateways receive the same information (set of messages), upon which they issue actuation signals and (ii) no compromised gateway can affect the system's correct and timely overall behavior.

The inter-gateway protocol we devise is a Byzantine reliable broadcast algorithm that utilizes the network connecting the gateways (this protocol does not transmit any information on the WSN). Roughly speaking, the protocol works as follows. Any sink that receives a message from a WSN device (via the WSN) broadcasts this message to all other sinks. A sink that receives a message from some other sink, echoes this message only if that same message is received from at least $f+1$ sinks. A sink delivers a message (and issues actuation corresponding to that message) whenever that sink receives the same message from at least $2f+1$ sinks.

*1) ByzCast Algorithm:* We now describe in more details our Byzantine reliable broadcast, detailed in Algorithm 4.

We assume that any message arriving over the WSN is issued by a unique WSN device $n_i$ and has a unique time-stamp $t_{stamp}$ unforgeable by other WSN devices. As such the source WSN device combined with the time-stamp of a message constitute a unique identifier of that message.

Every gateway $g_i$ that receives data, e.g., a message $m$, from a WSN device (i.e., via the WSN) issues a broadcast on the gateway network. It does so by sending RBcast $(g_i, (n_i, t_{stamp}, m))$ to all gateways (lines 4-10). This happens under the condition that the received data's expiry time is larger than the current time (at the which the broadcast was issued) plus some slack, denoted by $\Delta$. The calculation of $\Delta$ will be explained later in this section.

A gateway $g_j$ receiving RBcast $(g_i, (n_i, t_{stamp}, m))$ (for the first time) from at least $f + 1$ gateways, sends Echo $(g_i, (n_i, t_{stamp}, m))$ to all other gateways, via the gateway network, if $g_j$ has not already sent some Echo $(..., (n_i, t_{stamp}, m))$ (lines 12-17).

When a gateway $g_j$ receives Echo $(g_i, (n_i, t_{stamp}, m))$ from at least $f + 1$ gateways, and $g_i$ has not yet sent Echo $(..., (n_i, t_{stamp}, m))$, $g_j$ sends Echo $(g_i, (n_i, t_{stamp}, m))$ to all gateways (lines 19-24). If a gateway $g_j$ receives Echo $(g_i, (n_i, t_{stamp}, m))$ from at least $2f + 1$ gateways, $g_j$ delivers message $m$ (lines 25-27).

**Algorithm 4** ByzCast: A Byzantine Reliable Broadcast

```
 1: Uses: Synchronous Reliable Links
 2:     Send(...) if sending and receiving nodes are correct message is received
        successfully by destination within delay $d$.
 3:
 4: upon event <receive $(n_i, t_{stamp}, m)$ from WSN at time $t$> do
 5: // every message $m$ has a unique unforgeable time-stamp $t_s$ and is issued
    by a unique WSN device $n_i$
 6:     if message expires at time $> t + \Delta$ then
 7:         for every gateway do
 8:             Send RBcast $(g_i, (n_i, t_{stamp}, m))$
 9:         end for
10:     end if
11:
12: upon event <receive RBcast $(g_i, (n_i, t_{stamp}, m))$ from $g_i$> do
13:     if RBcast $(..., (n_i, t_{stamp}, m))$ is received from at least $f + 1$
    gateways $\wedge$ no Echo $(..., (n_i, t_{stamp}, m))$ has been sent then
14:         for every gateway do
15:             Send Echo $(g_i, (n_i, t_{stamp}, m))$
16:         end for
17:     end if
18:
19: upon event <receive Echo $(g_i, (n_i, t_{stamp}, m))$> do
20:     if Echo $(..., (n_i, t_{stamp}, m))$ is received from at least $f + 1$
    gateways $\wedge$ no Echo $(..., (n_i, t_{stamp}, m))$ has been sent then
21:         for every gateway do
22:             Send Echo $(g_i, (n_i, t_{stamp}, m))$
23:         end for
24:     end if
25:     if Echo $(..., (n_i, t_{stamp}, m))$ is received from at least $2f + 1$
    gateways $\wedge$ message $(n_i, t_{stamp}, m)$ has not been delivered then
26:         Deliver $(n_i, t_{stamp}, m)$
27:     end if
28:
```

Having a gateway deliver a message $m$, means that it is safe for the gateway to issue actuation based on $m$. However, in order to ensure that all gateways issue actuation signals in the same order, gateways need to see messages in the same order. Let $t_{arr}$ be the time at which some gateway receives a message $m$ from the WSN. If $m$ expires before $t_{arr} + \Delta$ then $m$ is ignored; otherwise, it sent via RBcast $(...)$. Despite being delivered, gateways will only use a delivered message for actuation in its last time instant before expiring (assuming a discrete time scale).

*2) Determining $\Delta$:* Given the algorithm above for synchronizing the information between gateways, we assess next the value of $\Delta$, the upper bound on the time needed to disseminate reliably and securely a valid sensor data to all gateways via our Byzantine reliable broadcast protocol.

Let us denote by $d$ the upper bound on the delay for sending a message successfully between any two gateways (via the gateway network). $d$ includes the overall time needed from the moment a source gateways sends a value until the destination gateways receives that value[4]. Our ByzCast protocol (Algorithm 4) becomes active when at least $f + 1$ gateways broadcast a received message relative to some sensor node (lines 12-17 of Algorithm 4). In other words, in the worst case, when $f$ gateways are Byzantine and decide not to broadcast a message received relative to a WSN device, a message (sent by a WSN device) needs to be heard by $2f + 1$ gateways before the broadcast is active. From the moment that $2f + 1$ gateways receive a message two communication rounds are needed for message delivery, i.e., $2d$. Assuming that the difference in time between having a message from a WSN reach one gateway and reach at least $2f + 1$ is $d_{2f+1}$, then we have:

$$\Delta = d_{2f+1} + 2d.$$

Hence $\Delta$ is an upper bound on the delay from the moment that some gateway receives a message $m$ (from the WSN) until $m$ is delivered. Notice that $\Delta$ is partially dependent on FT-TSTP, namely the $d_{2f+1}$ factor within $\Delta$. As shown in Algorithm 4 (lines 6-10), the value of $\Delta$ is used to make sure that no message is delivered after getting expired.

## IV. Protocols' Properties & Resiliency to Attacks

In this section we explain how our proposed protocols, namely the FT-TSTP augmented with the ByzCast, provide the required properties (listed in Section III). We discuss as well the resiliency of our protocols to attacks, such as the *Replay* attack and the *Distributed Denial of Service* attack.

### A. No Duplication

Both protocols described in the previous Sections, FT-TSTP and ByzCast, cooperate to ensure the aimed *No Duplication* property for SmartData. At the WSN level, FT-TSTP identifies messages with a hash of the space-time coordinates where the SmartData was created. These ids are used in Algorithm 2 along with the full coordinates to identify previously seen messages for the sake of acknowledging them and also to prevent the re-injection of duplicates into the network during routing. The Algorithm can be enriched with an additional clause to check for messages being injected at the origin

[4] We assume that the queuing time at the destination side before a message is ready to be processed is included within $d$

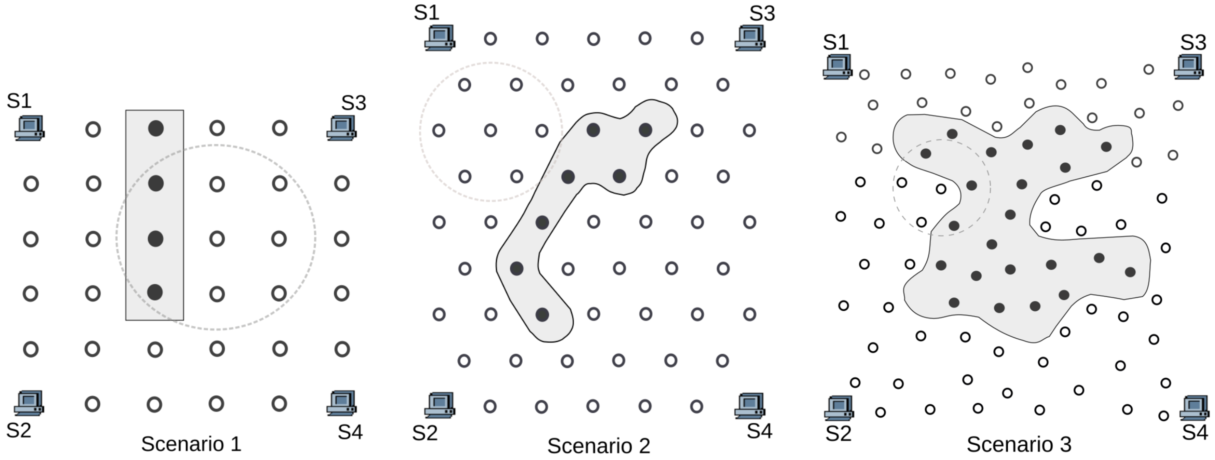

Figure 7. Simulated scenarios. Voids are represented by the gray area. Dashed circles represent radio range.

Table I
SIMULATION PARAMETERS.

| Scenario | Grid Size (m) | Node Placement | Radio Range (m) | Duty Cycle (%) | TX Power | Period (s) | Expiry (s) |
|---|---|---|---|---|---|---|---|
| **1 - Building** | 70x70 | 32 nodes, 6x6, regular squares | 20 | 5 | -10 dBm | 60 | 60 |
| **2 - Building** | 150x175 | 55 nodes, 7x9, regular hexagons | 45 | 5 | -5 dBm | 60 | 60 |
| **3 - Field** | 500x500 | 81 nodes, irregular placement | 80 | 5 | 0 dBm | 120 | 120 |

containing SmartData that does not match the region of space-time of the injecting node:

```
if distance(here(), $m.Origin$) $\leq R$ then
    if round(Region(here(), now())) $\neq$ round($m.Origin$) then
        delete $m$
    end if
end if
```

When a message reaches a gateway, ByzCast is initiated to propagate the message to all other gateways. Messages are uniquely identified by their source WSN device combined with their unforgeable time-stamp. As such, ByzCast eliminates delivering any message duplicates by constantly checking if that particular message has been already delivered (line 25).

### B. Availability

Compared to TSTP, SmartData availability is improved using our protocols. FT-TSTP forwards messages containing SmartData to multiple gateways, hence the failure of a gateway no longer hinders availability. Moreover, due to the recovery routing mode in FT-TSTP, reverse flooding is promoted whenever a void (or a Byzantine node) prevents the ordinary geographic routing. If there is at least one route to the destination and if the SmartData expiry allows, there is a very high probability that messages are delivered, as it is shown in Section V.

Our ByzCast protocol as well helps in improving availability. Precisely, in unfortunate situations where a gateway is unreachable by any WSN device (due to device failures or attacks), our ByzCast protocol guarantees that such alienated gateways still receive the desired SmartData (via the gateway interconnect now).

### C. Agreement

Given that some gateways can be compromised by malicious attackers, wrong control and actuation signals might be issued by individual gateways. In order to be resilient to such false signals, WSN actuators ignore actuation signals that are not received from at least $f + 1$ gateways. In other words, an actuation signal is considered valid *if and only if* that same signal is issued by at least $f+1$ distinct gateways. This ensures that at least one correct gateway approves of issuing such a control signal.

### D. Timeliness

SmartData defines two modes of operation: time-triggered and event-driven (see Section II-A). However, the event-driven mode, in which sensor nodes produce messages whenever they observe a measurable variation in a physical quantity, does not bear any notion of temporal guarantees. The network is operated on a best-effort scheme driven by the expire of messages. Modeling network congestions and the expiration of messages in this scenario depends of a deep knowledge of the CPS into which it is integrated. Therefore, in order to achieve the timeliness property, FT-TSTP only supports the time-triggered mode. In this mode, sensor nodes periodically produce messages containing SmartData in response to, *and only* in response to, Interest messages propagated by the gateways. Tools are available to optimally schedule a network of this nature off-line [15], [16], which could be used to plan the deployment of a FT-TSTP network. Alternatively, schedulability analysis and/or statistical prediction can be used to implement an admission control mechanism at the gateways, thus coordinating the insertion of Interest messages coming from the IT infrastructure.

However, even if the network is proved to be schedulable at deployment-time, that is, the placement of nodes, the periods of messages and their expires are all in tune with the requirements of applications using the WSN, there are still run-time phenomena that can cause such a network to fail. FT-TSTP addresses two categories of such phenomena to improve timeliness: Refusal-to-Forward and Distributed Denial-of-Service attacks. The counter-measures to these attacks will be described, however, if after all steps a message expires, then it is simply dropped (Algorithm 3, line 4).

### *E. Handling of Refusal-to-Forward Attacks*

Compromised nodes in a WNS can refuse to forward messages toward their destinations, causing an impact that is comparable to a lack of connectivity resulting from node failures. FT-TSTP's *Multi-Sink Greedy Forwarding* Algorithm (2) handles this of connectivity trough its void-detour features (i.e. recovery mode). So, if there is a minimum set of sane nodes that can be connected to deliver a message to its destination, and the WSN is not congested, then Algorithm (2) will eventually find it using reverse flooding. Nevertheless, Byzantine nodes refusing to forward a message can still cause a big impact on application by deliberately refusing to forward messages and thus increasing the latency of the network, eventually to beyond the expiry of many messages. Since FT-TSTP defines a fully reactive routing scheme for the sake of fault-tolerance, the preference for geographic routing is restored for each individual message. However, entering and leaving the rescue routing mode is an expensive process.

### *F. Handling Distributed Denial-of-Service (DDoS) Attacks*

Compromised nodes in a WSN can promote DDoS attacks by flooding the network with unsolicited messages (i.e. with messages that are not Responses to any Interest message sent by the sinks). Eventually, the attack can escalate to levels comparable to jamming the radio channel in a given region. If stopping a sensor node from transmitting is out-of-reach to communication protocols above the physical-level, confining the attack to the region close the the attacker is done by FT-TSTP following the same principal adopted on gateways in the original TSTP protocol: messages carrying SmartData that is not a response to any valid Interest messages are simply dropped. For this, in addition to sinks, all nodes that hear an Interest message keep a copy of it. Before accepting to become a relay for any message, a node first examines its list of Interest messages. In this way, DoS attacks are confined to one hop from the source.

## V. Protocol Evaluation

In order to evaluate the Fault-Tolerant Trustful Space-Time Protocol (FT-TSTP) and the Byzantine Reliable Broadcast Protocol (ByzCast) described in this paper and therefore corroborate the claims made about their ability to extend the SmartData concept with Byzantine-resiliency, we have devised a set of experiments focused on the protocols impact on the WSN message delivery rate, latency and energy consumption.

We evaluate the communication performance of the protocols proposed here through a set of simulations on the OMNet++ [17] (version 4.6) simulator using the Castalia (version 3.3) framework [18]. We assessed the quality of our simulation models by comparing the simulation results with those obtained in field on our Solar Smart Building, which is automated using the Embedded Parallel Operating System (EPOS) [19] and the original TSTP [12] on EPOSMote III devices [19]. Indeed, the source code used for the simulations is basically the same used in the real deployment, but the comparison between the real scenario and the simulated model allowed us to adjust the model's parameters in order to obtain realistic results.

### *A. Simulation Parameters*

We simulated a set of WSNs, each with different nodes placements and void configurations, in order to evaluate the impact of delivering messages simultaneously to multiple sinks and void detouring over the delivery rate, the end-to-end transmission time and energy consumption. Three WSNs were modeled on top of the CC2420 IEEE 802.15.4 physical layer available in Castalia. Table I summarizes the parameters used in these simulations. Each WSN was simulated using TSTP and FT-TSTP for a simulation period of 3600s, with the presence of the voids represented in Figure 7. For FT-TSTP, the $\beta$ parameter (retries) was set to 1. FT-TSTP was also simulated with different numbers of sinks, from 1 to 4. Each simulation was executed 10 times with different random number seeds, and the results presented are the average values.

### *B. Scenarios*

The scenarios where modeled in order to evaluate the FT-TSTP behavior under different aspects of WSN deployments, such as nodes distance and placement, radio TX power and void configurations. The objective was to verify that the void detour algorithm is able to handle different voids types, while measuring the latency and energy overheads, since there is a trade-off between reliability obtained through multi-sink transmissions and latency/power consumption. Scenarios 1 and 2 can be applied in monitoring air quality in large indoor areas, as the temperature in shopping malls or dust in big industrial sheds. Scenario 3 is typical of environmental monitoring or smart agriculture, like soil moisture in irrigation systems, or temperature/humidity in field monitoring. In all scenarios, the data periods are not too tight, but fault tolerance is justified by economic loss on the lack of correct measurements and actuation. The voids are modeled as concave regions (Scenarios 2 and 3), and a section on the border, causing a semi-partition in the field (Scenario 1). These layouts are similar to the presented in other works [2] [20]. In the simulations, one SmartData update was produced by every operational node at each data period, and the message had the same time to reach the sinks before expiry. The data generation instant was randomly chosen in the first period, and from that on a new SmartData was generated in exact periodic cycles.

Table II
DELIVERY RATE

| | TSTP | FT-TSTP | | | |
|---|---|---|---|---|---|
| Scenario | 1 Sink | 1 Sink | 2 Sinks | 3 Sinks | 4 Sinks |
| Scenario 1 | 46.8434% | 0.0000% | 0.0000% | 0.3921% | 99.6079% |
| Scenario 2 | 66.4279% | 0.0036% | 0.5624% | 9.3647% | 90.0692% |
| Scenario 3 | 82.7258% | 0.0119% | 0.0655% | 0.7917% | 99.1310% |

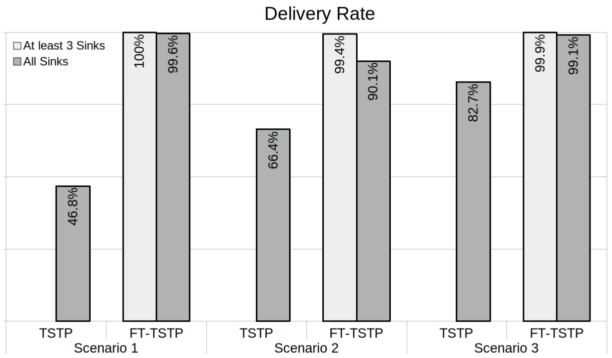

Figure 8. FT-TSTP vs TSTP delivery ratio to at least three and to all sinks, with 4 sinks and void present.

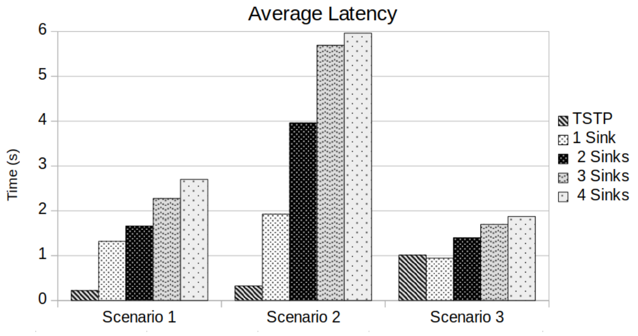

Figure 9. FT-TSTP vs TSTP **average** end-to-end message delivery latency, TSTP with 1 sink and FT-TSTP with 1 to 4 sinks.

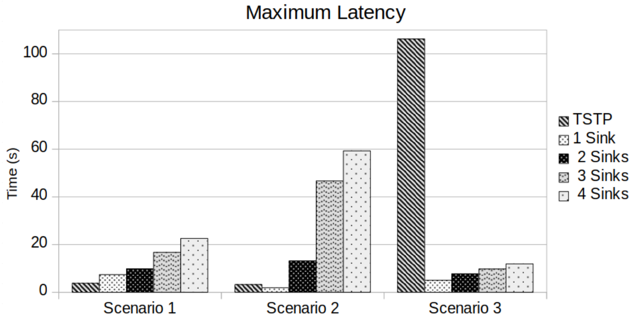

Figure 10. FT-TSTP vs TSTP **maximum** end-to-end message delivery latency, TSTP with 1 sink and FT-TSTP with 1 to 4 sinks.

A comparison between TSTP and FT-TSTP delivery ratio in the three modeled scenarios is shown in Table II and Figure 8. In the presence of faulty nodes, creating a void region, TSTP's delivery rate suffers a considerable impact, delivering under 50% of the messages in Scenario 1. FT-TSTP was able to deliver more than 99% messages to at least three sinks in all scenarios, which is enough for the ByzCast algorithm come to an agreement about the received values.

Figure 9 shows the average latency time to a message to reach all the sinks. Scenario 2 shows a greater latency because more messages are being transmitted over the WSN with the same data period, but there are more nodes than in Scenario 1. Figure 10 shows the maximum latency observed in the simulations, and shows that Scenario 2 is close to it's saturation point, as some messages are reaching the sinks very close to the expiry time (60s), and about 10% don't reached all sinks (figure 8). On Scenario 3, figure 10 shows that TSTP imposes long delays to some messages due to retransmissions on void borders.

Figure 11 shows the energy consumed by active nodes and sinks during the simulation period (1 hour). The additional energy consumption arises from different aspects. First, the increase in the microframe's size, from 9 to 28 bytes demands more energy to transmit the message's preamble. In conjunction with the retransmissions in *recovery mode* to circumvent the voids, it explains the energy increasing from TSTP to FT-TSTP with a single sink. The increasing from 1 to 2 sinks is due to the duplication of the messages, as they have to travel in opposite directions most of the time. The energy increasing for 3 and 4 sinks is smaller, as messages make progress to more than one sink on most transmissions. If a fifth sink were added, the energy increasing would be even smaller than the increase from 3 to 4. The outliers values, represented by the circles, show that when TSTP reaches a *local minimum* - on a void border - it consumes much more power than the nodes in FT-TSTP with one sink. This happens because these nodes will continuously retransmit messages until its expiration. So, the deployment of a resilient WSN requires a careful dimension of data periods and duty cycle values in order to meet the resilience requirements, as mentioned in section II-A.

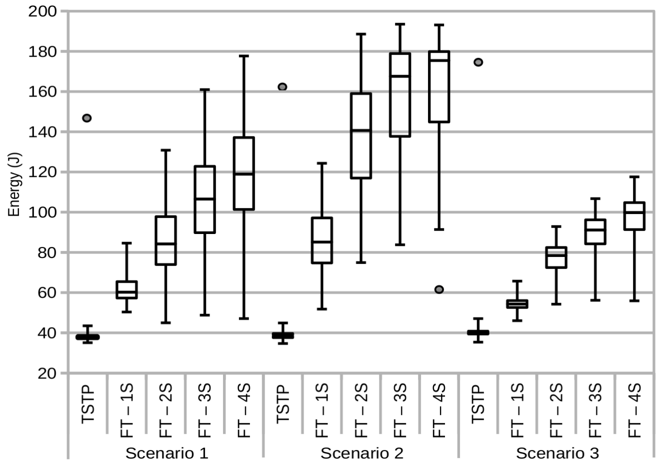

Figure 11. FT-TSTP vs TSTP energy consumption at one node in 1 hour. Circles represent outliers. Energy for FT-TSTP with 1 to 4 sinks.

## VI. RELATED WORK

The void detour problem is addressed by several researches. In geographic routing protocols, these algorithms can be classified in two groups: right-hand rule and back-pressure rule. The first makes the packet be routed through the border of the void, as in GPSR [21]. They have the side effect of high power consumption on the border nodes. The back-pressure rule, as in SPEED [22] and FT-SPEED [23], sends messages back to the upstream nodes, notifying the presence of a void and demanding the use of alternative paths. In [24] three schemes are presented in order to merge these two approaches, and the authors claim that they can be used with any geographic routing. Each node has to know the neighbors' position, and must be able to send messages to a specific node. Special packets for *Void Detection* and *Void Maintenance* are employed in order to map the voids. The schemes are not applicable to *reactive* geographic protocols like FT-TSTP, as nodes do not keep information about neighbors' position. These protocols also have the advantage of re-routing packets without void discovery and maintenance packets exchange. They are also able to instantly restore routing when transient failures occur, as well as fast response when additional node failures occur, changing voids' configuration. The *recovery mode* in FT-TSTP makes nodes send the packets backwards (or aside), until a route that detours the void is found, acting like a composition of back-pressure and border routing.

The REACT algorithm presented in [25] assumes that the sink's signal can reach every sensor node, doing the sink-to-nodes communication in an one-hop way. The nodes don't know its exact location, but they calculate the distance to the sink by using RSSI strength, assuming that a stronger signal means that the node is closer to the sink. The sensors readings are transmitted to the sink in a multi-hop manner, where each node closer to the sink than the transmitter becomes a potential relay, in a self-voting mechanism similar to the used in TSTP/FT-TSTP. Data aggregation is used to reduce the number of messages. If a hole is detected the packed is marked as *hole-packet* and broadcasted until it reaches a node nearer than the *hole node* that initiates the process. The protocol doesn't considers multiple sinks and there are no authentication or privacy mechanisms.

Several works have employed multi-sink approaches. The authors of [1] propose the use o multiple mobile sinks, which move to WSN regions where nodes have the most remaining energy. Nodes always try to send data to the nearest sink. In [2], each node tries to send packets to the closest sink. If a void is detected, a *flexible set* flag is set in the packets to enlarge the forwarder set, to circumvent the void or to transmit packets to another sink. In [3] an algorithm builds clusters and backbones, connection nodes to trees with a sink as root. There are N trees, one for each sink, and each node belongs to a unique tree. Trees are rebuilt on nodes or sinks failures. Pheromone levels (ant colony algorithm) are used in [20] to determine multiple routes from nodes to sinks and from sinks to nodes, based on QoS parameters. Again, each node delivers its messages to the sink whose path offers the best QoS parameters. The BAMBi protocol proposed by [26] sends a copy to every known sink separately. Routing trees with sinks as roots are built with each node belonging at least to N trees (where N is the number of sinks). Trees are rebuilt on node or sink failure.

All the mentioned solutions use multiple sinks try to minimize the impact of node and sink failures, or to get a better energy balance or lower delivery times. All consider a message delivered if it reaches one sink. None of them proposes integrity checking for messages, so they are susceptible to attacks that can modify the message content on the network or the sinks. The FT-TSTP has the objective to deliver messages to all sinks with the purpose to allow ByzCast to verify the message integrity.

The work in [27] addresses the relay node placement problem from a fault-tolerance perspective. The proposed solution aims at minimizing the number of relays needed to connect a set of points of interest to a sink by placing them on the vertexes of a triangular lattice inscribed in minimal rectangle that encompasses all the points. Their placement outperforms those based on straight lines or steiner-points and could be used along with the protocol proposed here whenever placement can be planed beforehand.

LiveOS [28] approaches fault-tolerance in WSN from the perspective of individual nodes. It offers applications a mechanism to store and roll-back memory regions similar to a transaction manager. Such memory blocks can be processed in parallel by threads running in multiple cores in a voting-like scheme. If computations are considered correct, then the modifications to the memory block are committed, otherwise they get rolled back. The system does not provide the voting mechanism and the assumption that a multicore sensor node can have one of its core to malfunction while other components continue to operate normally limits the applicability of the proposed scheme.

## VII. Conclusion

The use of the *SmartData* concept, a high level abstraction of sensing data, on an infrastructure that is resilient to byzantine faults in the sensors network and in the gateways level, provides a more reliable infrastructure for the IoT. At the sensors net level, the FT-TSTP protocol was presented, that adds *availability* to the TSTP protocol, as it can handle voids generated by compromised nodes. Availability is also increased by delivering the message to multiple sinks (gateways), reducing the probability that byzantine nodes can isolate the sink from the rest of the WSN. At the gateways level, besides providing availability through redundancy, the *agreement* property achieved by the ByzCast algorithm avoids that a compromised gateway prevents the entire WSN from delivering reliable sensed data. Other properties of the SmartData, as *timeliness*, *no duplication*, *non-repudiation*, *confidentiality*, *integrity* and *authenticity* are preserved by the proposed solution.

The availability obtained by the sinks redundancy has a trade-off with higher energy consumption and end-to-end communication time. In networks where the resilience to byzantine faults is mandatory, these cost increments are inevitable. The simulation results have shown a linear growth of energy and time consumption, influenced by the network traffic load and the size and configuration of void regions. So, it is possible to make reasonable predictions and previous dimensioning of the WSN in order to obtain bounded delivery time, even in a given level of nodes failures.

## REFERENCES

[1] S. Yasotha, V. Gopalakrishnan, and M. Mohankumar, "Multi-sink optimal repositioning for energy and power optimization in wireless sensor networks," *Wireless Personal Comms.*, vol. 87, no. 2, pp. 335–348, 2016.

[2] S. Ozen and S. Oktug, "Adaptive sink selection for wsns using forwarder set based dynamic duty cycling," in *Sensing, Communication, and Networking Workshops (SECON Workshops), 2014 Eleventh Annual IEEE International Conference on*. IEEE, 2014, pp. 7–12.

[3] M. Carlos-Mancilla, E. Lopez-Mellado, and M. Siller-Gonzalez, "A localized multi-sink multi-hop algorithm for wireless sensor networking," in *Global Information Infrastructure and Networking Symposium (GIIS), 2015*. IEEE, 2015, pp. 1–6.

[4] J.-q. Zhang and R.-c. Wang, "Qos-aware routing for multi-sink wmsns," *DEStech Trans. on Computer Science and Engineering*, no. icte, 2016.

[5] D. Resner and A. A. Fröhlich, "Design Rationale of a Cross-layer, Trustful Space-Time Protocol for Wireless Sensor Networks," in *20th IEEE Intl. Conference on Emerging Technologies and Factory Automation (ETFA).*, Luxembourg, Luxembourg, 2015, pp. 1–8.

[6] A. A. Frohlich, A. M. Okazaki, R. V. Steiner, P. Oliveira, and J. E. Martina, "A Cross-layer Approach to Trustfulness in the Internet of Things," in *9th Workshop on Software Technologies for Embedded and Ubiquitous Systems (SEUS)*, Paderborn, Germany, Jun. 2013, pp. 1–8.

[7] IEEE 1451.0, *IEEE Standard for a Smart Transducer Interface for Sensors and Actuators - Common Functions, Comm. Protocols, and Transducer Electronic Data Sheet (TEDS) Formats*, September 2007.

[8] D. J. Bernstein, "The poly1305-aes message authentication code," in *Proc. of Fast Software Encryption*, Paris, France, Feb 2005, pp. 32–49.

[9] O. Raso, P. Mlynek, R. Fujdiak, L. Pospichal, and P. Kubicek, "Implementation of elliptic curve diffie hellman in ultra-low power microcontroller," in *2015 38th International Conference on Telecommunications and Signal Processing (TSP)*. IEEE, jul 2015. [Online]. Available: https://doi.org/10.1109%2Ftsp.2015.7296346

[10] D. Resner, A. A. Fröhlich, and L. F. Wanner, "Speculative Precision Time Protocol: submicrosecond clock synchronization for the IoT," in *21th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, Berlin, Germany, Sep. 2016.

[11] R. Reghelin and A. A. Fröhlich, "A Decentralized Location System for Sensor Networks Using Cooperative Calibration and Heuristics," in *9th ACM/IEEE International Symposium on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, Torremolinos, Malaga, Spain., Oct. 2006, pp. 139–146.

[12] D. Resner and A. A. Frohlich, "TSTP MAC: A Foundation for the Trustful Space-Time Protocol," in *14th IEEE/IFIP International Conference on Embedded and Ubiquitous Computing (EUC). To appear*, Paris, France, Aug. 2016.

[13] M. R. Akhavan, T. Watteyne, and A. H. Aghvami, "Enhancing the performance of RPL using a Receiver-Based MAC protocol in lossy WSNs," in *IEEE ICT*, Ayia Napa, Cyprus, May 2011, pp. 191–194.

[14] D. Dolev, "The byzantine generals strike again," Stanford, CA, USA, Tech. Rep., 1981.

[15] P. Pop, P. Eles, and Z. Peng, "Schedulability-driven communication synthesis for time triggered embedded systems," *Real-Time Systems*, vol. 26, no. 3, pp. 297–325, apr 2004.

[16] S. S. Craciunas and R. S. Oliver, "SMT-based task- and network-level static schedule generation for time-triggered networked systems," in *Proceedings of the 22nd Intl. Conference on Real-Time Networks and Systems - RTNS'14*. ACM Press, 2014.

[17] OpenSim, "OMNeT++ - Objective Modular Network Testbed in C++," 2017. [Online]. Available: https://omnetpp.org/

[18] A. Boulis, "Castalia A simulator for Wireless Sensor Networks and Body Area Networks," 2017. [Online]. Available: https://github.com/boulis/Castalia

[19] S. I. Lab, "EPOS - Embedded Parallel Operating System," 2017. [Online]. Available: https://epos.lisha.ufsc.br/

[20] D. Zhang and E. Dong, "A virtual coordinate-based bypassing void routing for wireless sensor networks," *IEEE sensors journal*, vol. 15, no. 7, pp. 3853–3862, 2015.

[21] B. Karp and H. T. Kung, "Gpsr: Greedy perimeter stateless routing for wireless networks," in *Proceedings of the 6th Annual International Conference on Mobile Computing and Networking*, ser. MobiCom '00. New York, NY, USA: ACM, 2000, pp. 243–254. [Online]. Available: http://doi.acm.org/10.1145/345910.345953

[22] T. He, J. A. Stankovic, C. Lu, and T. Abdelzaher, "Speed: A stateless protocol for real-time communication in sensor networks," in *Distributed Computing Systems, 2003. Proceedings. 23rd International Conference on*. IEEE, 2003, pp. 46–55.

[23] L. Zhao, B. Kan, Y. Xu, and X. Li, "Ft-speed: A fault-tolerant, real-time routing protocol for wireless sensor networks," in *Wireless Communications, Networking and Mobile Computing, 2007. WiCom 2007. Intl. Conf. on*. IEEE, 2007, pp. 2531–2534.

[24] M. Aissani, A. Mellouk, N. Badache, and M. Boumaza, "A novel approach for void avoidance in wireless sensor networks," *International Journal of Communication Systems*, vol. 23, no. 8, pp. 945–962, 2010.

[25] M. M. Lima, H. A. Oliveira, D. L. Guidoni, and A. A. Loureiro, "Geographic routing and hole bypass using long range sinks for wireless sensor networks," *Ad Hoc Networks*, 2017.

[26] S. Misra, K. Bhattarai, and G. Xue, "Bambi: Blackhole attacks mitigation with multiple base stations in wireless sensor networks," in *Communications (ICC), Intl. Conf. on*. IEEE, 2011, pp. 1–5.

[27] I. Khoufi, P. Minet, and A. Laouiti, "Fault-tolerant and constrained relay node placement in wireless sensor networks," in *2016 IEEE 13th Intl. Conf. on Mobile Ad Hoc and Sensor Systems (MASS)*. IEEE, oct 2016, pp. 127–135.

[28] X. Liu, H. Zhou, S. Xiong, K. M. Hou, C. de Vaulx, and H. Shi, "Development of a resource-efficient and fault-tolerant wireless sensor network system," in *2015 2nd Intl. Symposium on Dependable Computing and Internet of Things (DCIT)*. IEEE, nov 2015, pp. 122–127.

**Antônio Augusto Fröhlich** received his PhD in Computer Science from the Technical University of Berlin in 2001. He has been a professor in the Computer Science Department, Federal University of Santa Catarina, Brazil since 1995 and head of the Software/Hardware Integration Lab since 2001. His current research interests include embedded systems and operating systems.

**Roberto Milton Scheffel** received his MSc in Computer Science from the Federal University of Santa Catarina, Brazil in 1997. He has been a professor at the Federal University of Technology - Paraná (UTFPR) since 2015. He is currently working towards the PhD degree at the Federal University of Santa Catarina under the supervision of A. A. Fröhlich. His research interests include distributed embedded systems and fault tolerance.

**David Kozhaya** is a Research Scientist at ABB Corporate Research, Switzerland. He received his PhD degree in Computer Science in December 2016, from EPFL, Switzerland. His research interests include reliable distributed computing, real-time distributed systems, and fault- and intrusion-tolerant distributed algorithms. His past work experiences span across interdisciplinary domains ranging from research, teaching, financial and market analysis, and the management of various non-profit organizations.

**Paulo Esteves Veríssimo** is a Professor and FNR PEARL Chair at the Faculty of Science, Technology and Communication (FSTC), University of Luxembourg (UL); and head of the CritiX research group at UL's Interdisciplinary Centre for Security, Reliability and Trust. He is interested in secure and dependable distributed architectures, middleware and algorithms for: resilience of large-scale systems and critical infrastructures, privacy and integrity of highly sensitive data, and adaptability and safety of real-time networked embedded systems.