PCSPOOF: Compromising the Safety of Time-Triggered Ethernet

Andrew Loveless*‡ Linh Thi Xuan Phan† Ronald Dreslinski* Baris Kasikci*
*University of Michigan †University of Pennsylvania ‡NASA Johnson Space Center
*{loveless, rdreslin, barisk}@umich.edu †linhphan@seas.upenn.edu

***Abstract*—Designers are increasingly using mixed-criticality networks in embedded systems to reduce size, weight, power, and cost. Perhaps the most successful of these technologies is Time-Triggered Ethernet (TTE), which lets critical time-triggered (TT) traffic and non-critical best-effort (BE) traffic share the same switches and cabling. A key aspect of TTE is that the TT part of the system is *isolated* from the BE part, and thus BE devices have no way to disrupt the operation of the TTE devices. This isolation allows designers to: (1) use untrusted, but low cost, BE hardware, (2) lower BE security requirements, and (3) ignore BE devices during safety reviews and certification procedures.**

We present PCSPOOF, the first attack to break TTE's isolation guarantees. PCSPOOF is based on two key observations. First, it is possible for a BE device to infer private information about the TT part of the network that can be used to craft malicious synchronization messages. Second, by injecting electrical noise into a TTE switch over an Ethernet cable, a BE device can trick the switch into sending these malicious synchronization messages to other TTE devices. Our evaluation shows that successful attacks are possible in seconds, and that each successful attack can cause TTE devices to lose synchronization for up to a second and drop tens of TT messages — both of which can result in the failure of critical systems like aircraft or automobiles. We also show that, in a simulated spaceflight mission, PCSPOOF causes uncontrolled maneuvers that threaten safety and mission success. We disclosed PCSPOOF to aerospace companies using TTE, and several are implementing mitigations from this paper.

***Index Terms*—Time-Triggered Ethernet, packet-in-packet attacks, electromagnetic interference, embedded systems**

I. INTRODUCTION

Increasingly, embedded systems are using *mixed-criticality* network technologies that allow traffic with different timing and fault tolerance requirements to coexist in the same physical network [1]–[4]. These technologies let designers reduce size, weight, power, and cost by sharing the same network between critical and non-critical parts of the system. For example, aircraft can share one network between vehicle control systems and passenger Wi-Fi and entertainment systems [5], [6]; spacecraft can share one network between life support systems and onboard experiments [7], [8]; and manufacturing plants can share one network between robot control systems and data collection systems [9].

One of the most successful mixed-criticality network technologies is *Time-Triggered Ethernet (TTE)* [2], Today, TTE serves as the network backbone for several spacecraft, including NASA's Orion capsule [10], NASA's Lunar Gateway space station [7], and ESA's Ariane 6 launcher [11]. TTE is also widely used in aircraft [12]–[14], energy generation systems [15], and industrial control systems [16], [17], and is a leading contender to replace CAN bus and FlexRay as the standard network technology in future automobiles [18], [19].

TTE has several properties that make it attractive for safety and mission-critical applications. Most notably, TTE follows a *time-triggered (TT)* paradigm, in which devices are tightly synchronized, and they send messages and execute software according to a predetermined schedule. This TT approach reduces message latencies to hundreds of microseconds and jitter to near-zero [20], [21], making TTE appropriate for even the tightest control loops. TTE also provides fault tolerance by replicating the whole network to form multiple *planes*, and by forwarding messages over all planes simultaneously [22].

In addition, TTE enables mixed-criticality architectures by being 100% compatible with standard Ethernet [23]. This means that *non-critical* systems, which typically use standard Ethernet hardware to lower costs [24], can send messages over the same cabling as the critical TTE devices. Unlike TT traffic, standard Ethernet traffic is forwarded on a *best-effort (BE)* basis, filling in space *around* the TT traffic [23]. Also, standard Ethernet traffic typically only travels over a single network plane, so does not have any fault tolerance guarantees [7].

A key aspect of TTE's mixed-criticality design is that the TT part of the system is *isolated* from the BE part. In other words, no matter how the BE devices behave, they should not be able to disrupt synchronization between TTE devices, or the timely or successful delivery of TT traffic [25]. This isolation is commonly used as justification for several cost-cutting measures, including: (1) procuring BE devices from relatively untrusted (but low cost) suppliers [26], [27]; (2) relaxing security requirements for BE devices [28]; and (3) reducing the scope of analysis and certification of a system to focus solely on the TTE devices [29]. For example, on NASA spacecraft, onboard experiments are often provided by university research groups, are operated by the university students with minimal NASA involvement, and are not considered in safety reviews or the certification process of the overall vehicle [30], [31].

In this paper, we present PCSPOOF, a new attack that breaks TTE's isolation guarantees for the first time — allowing a single malicious BE device on a single plane to disrupt synchronization and communication between TTE devices on all planes. PCSPOOF is based on two key observations:

First, it is possible for a malicious BE device to *infer* private information about the TTE network that is needed to construct valid TTE synchronization messages, called *protocol control*

frames (PCFs). For example, an attacker can exploit the fact that (1) all PCFs in the network contain a common identifier, and that (2) BE devices are *not allowed* to send messages containing this identifier. Such messages are simply dropped by the switches. Therefore, by issuing phony ARP [32] requests to other BE devices (e.g., routers), tricking them into sending messages containing possible identifiers, then checking which of the messages are dropped, an attacker can quickly determine the *actual* identifier used in the PCFs (see §IV-A).

Second, using a few extra circuit components, a malicious BE device can conduct electromagnetic interference (EMI) into a TTE switch and trick the switch into forwarding PCFs that the BE device is not allowed to send. In particular, by conducting EMI into the switch over an Ethernet cable, resulting in radiated EMI *inside the switch*, a BE device can cut the header off of a BE message in flight, revealing a malicious PCF in the message's payload (a type of packet-in-packet attack [33]). Since the EMI radiates from inside the switch, the attack cannot be prevented by conventional switch and cable shielding. Also, since the source of the radiated EMI (the port connected to the attacker) is close to the internal switch components (1–10 cm), the EMI requires relatively little power to be effective, and therefore can be generated with a small circuit (e.g., a 2.5 cm $\times$ 2.5 cm square, see §IV-B). As we show in §III, such a circuit could reasonably be hidden in a BE device and integrated into a TTE system without detection.

Finally, our work reveals a flaw in modern TTE devices that makes them especially susceptible to PCSPOOF's EMI injection. In particular, while modern devices verify the *contents* of the preamble that precedes each message, they do not verify the preamble *length*. An attacker can exploit this by sending very large BE messages, which are more likely to reveal PCFs when corrupted by EMI, without the PCFs being rejected by downstream TTE devices (see §IV-B).

We evaluated PCSPOOF on a real TTE testbed. Our results show that PCSPOOF can successfully inject a malicious PCF in 10–20 s. A single injection can cause TTE devices to lose synchronization for up to a second and fail to transmit tens of TT messages — both of which can cause the failure of critical systems [34], [35]. Moreover, in the worst case, PCSPOOF causes these outcomes simultaneously for *all TTE devices in the network* (see §VI-B). We also evaluated PCSPOOF on an avionics testbed for a real spaceflight mission; our results show that PCSPOOF can threaten mission success and safety from a single BE device, such as those used in an onboard research experiment developed by a university.

In summary, we make the following contributions:

- PCSPOOF: the first attack to break TTE's isolation guarantees; PCSPOOF can disrupt critical TT systems from a single malicious BE device (§IV).
- An extensive study of the susceptibility of TTE hardware to PCSPOOF, which reveals a security flaw in the implementation of modern devices (§IV-B).
- A detailed experimental evaluation of PCSPOOF on a real TTE testbed that assesses the probability and impact of successful attacks (§VI).
- A case study demonstrating the effect of PCSPOOF on a simulated spaceflight mission (§VI-D).
- A detailed description of methods to make TTE systems more resilient to PCSPOOF (§VII).

Responsible Disclosure. We disclosed our attack to several organizations using TTE for critical applications, including NASA, ESA, Northrop Grumman Space Systems, and Airbus Defense and Space. All organizations acknowledged the seriousness of the attack and several are implementing mitigations we suggest in this paper. Our work is also making NASA reconsider the way that onboard experiments and commercial-off-the-shelf devices are verified to be safe.

We also disclosed our attack to TTTech Computertechnik AG, the leading provider of TTE equipment and chip-IP. TTTech acknowledged the attack and is working on hardware, configuration, and tooling updates to mitigate it.

Also, the SAE AS-2D2 committee is working to mitigate our attack by revising the TTE standard (SAE AS6802) to allow PCFs up to 1518 bytes (the max Ethernet frame size). The use of max-sized PCFs would prevent the PCF injection method we use (cutting the header off a frame in flight) from producing a PCF that is accepted by TTE devices.

II. Background

In this section, we describe Time-Triggered Ethernet and the synchronization protocol it is based on.

A. Time-Triggered Ethernet (TTE)

TTE networks contain two types of devices, *switches* and *end systems*, where each end system consists of a host processor (which runs user software) and a TTE network interface card (NIC) [36]. Like in standard Ethernet, the switches forward messages, or *frames*, between the end systems. For redundancy, the entire network is replicated, creating multiple paths between each end system [22]. We refer to each redundant network as a *plane*. End systems send frames simultaneously through all planes, and receivers accept the first frame that arrives. This approach allows the system to continue operating even after multiple failures. An example of a typical TTE network is shown in Figure 1.

TTE networks utilize a *time-triggered* design, in which all TTE devices are tightly synchronized, and the behavior of the network is determined by a global schedule [2]. The schedule is built offline and loaded onto each TTE device before the system is deployed. The schedule specifies when TT frames are forwarded and expected to arrive. In addition, it specifies the timing of interrupts that software running on the end systems use to coordinate their actions [7]. This design reduces network latency and jitter to a minimum, resulting in very predictable system performance [20], [21].

Additionally, TTE networks are compatible with standard Ethernet [23]. This allows designers to use (inexpensive) standard Ethernet hardware for devices without strict timing or fault tolerance requirements, like passenger entertainment systems in airplanes or monitoring systems in power plants [37],

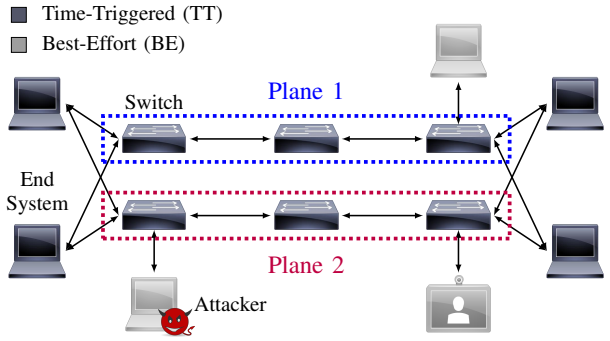

Fig. 1: Example of a typical fault-tolerant TTE network. The attacker controls a single BE device on a single plane.

[38]. These devices can plug directly into TTE switch ports and treat the TTE network exactly like a standard Ethernet network. We refer to these standard Ethernet devices as *best-effort (BE)* devices, since the switches forward their traffic around the pre-scheduled TT traffic only as bandwidth allows.

TTE uses several mechanisms to *isolate* the TT traffic from the BE traffic, including not allowing BE traffic to be transferred in windows reserved for TT traffic, and storing TT and BE frames in separate switch buffers [23]. Together, these mechanisms aim to ensure malicious BE devices have no way to interfere with the TT part of the system [25].

On the surface, attacks that break TTE's isolation guarantees seem impossible. For example, since the TTE switches reserve bandwidth for TT traffic and store TT and BE traffic separately, flooding the network from a BE device cannot cause TT traffic to be delayed or dropped [39]. The switches will simply drop the excess BE traffic to allow TT traffic to flow. Also, a BE device cannot generate its own TT traffic, since the switches ignore any TT traffic that is not defined in the pre-loaded schedule [40]. Even in the extreme case where a malicious BE device somehow kills the switch it is connected to, the TTE devices will continue to operate without disruption over the redundant planes.

B. The TTE Synchronization Protocol

TTE networks rely on a synchronization protocol to enable communication between devices [2], and PCSPOOF works by *disrupting* this protocol. Below, we describe the synchronization protocol and why it is susceptible to PCSPOOF.

There are two main roles that TTE devices can take in the synchronization protocol: (1) *sync master* and (2) *compression master* [2]. Typically a subset of the end systems act as sync masters (based on the required fault tolerance), and 1–2 switches per plane act as compression masters [22]. The remaining devices act as *sync clients*, which use the synchronized time base, but do not help maintain it [2].

In general, synchronization works by continuously exchanging special messages, called *protocol control frames (PCFs)*, between the devices [2]. This exchange is repeated at regular periods called *integration cycles* [2]. At the start of each integration cycle, each sync master sends a PCF with its local clock value to the compression masters. The compression masters average the received clock values, then send out the resulting clock value in a new PCF to the sync masters and clients, which use it to correct their local clocks.

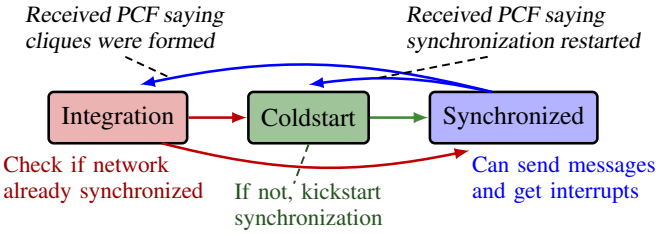

Fig. 2: Simplified version of the state machine that sync masters execute in the TTE synchronization protocol.

In addition, receiving certain PCFs from a compression master can cause sync masters and clients *to lose synchronization*. Specifically, a special "coldstart acknowledgement" PCF tells a sync master that another sync master detected synchronization was lost and is reestablishing it [2]. Similarly, the contents of a normal (i.e., integration) PCF can tell a sync master/client that *cliques* — multiple groups not synchronized to each other — have formed [41]. In either case, the sync master/client briefly loses synchronization and attempts to resynchronize with the network. Figure 2 shows how these PCFs impact the sync master state machine.

Because a single PCF can knock devices out of synchronization, significant effort has been spent to ensure all PCFs generated by compression masters can be trusted. For example, the TTE standard requires each compression master to be a self-checking pair — i.e., it only produces a PCF if two independent processors agree on the contents [2].

In PCSPOOF, we exploit the trust the TTE protocol puts in compression masters. By injecting PCFs into the network that look like they came from real compression masters, an attacker can make sync masters/clients repeatedly lose synchronization. We note that because synchronization loss between non-faulty devices is so rare in practice (requiring multiple specific failures), systems are often not designed to tolerate it [36], [42], [43]. Also, even systems that do tolerate synchronization loss are not designed to tolerate the repeated synchronization loss caused by PCSPOOF (see §VI-D).

III. Threat Model

We assume a standard multi-plane TTE network like those used today in spacecraft [7], [10], [11], aircraft [14], and energy generation systems [44] (see §II-A). The network includes both TTE and BE devices. For fault tolerance, the TTE end systems are connected to and communicate over all redundant planes simultaneously. In contrast, BE devices typically do not have any fault tolerance requirements, so often connect to only a *single* switch in a *single* plane in order to save wiring mass and cost [7].

We assume the attacker has the ability to execute malicious software on a *single* BE device, including sending and receiving *standard* Ethernet messages. The connectivity of the attacker's BE device is shown in Figure 1. In addition, we

assume the BE device includes additional circuit components that allow it to conduct electromagnetic interference (EMI) through its Ethernet cable and into the switch. As we show in §IV-B, such a circuit can be constructed from as little as 5 circuit components, and can take up as little as 2.5 cm $\times$ 2.5 cm on a single-layer printed circuit board.

There are two realistic ways these assumptions can be satisfied in practice: (1) the BE device is supplied by a malicious third-party and integrated into the TTE network at *design time*, or (2) the BE device is connected to the TTE network *after* the network is deployed.

First, the system integrator could obtain the BE device from a malicious third-party and integrate it into the system at design time. In TTE networks, non-critical functions are commonly performed using commercial-off-the-shelf (COTS) devices to reduce costs [7]. This is true even in critical industries like spaceflight and aviation [45], [46]. Unlike critical TTE devices, which come from secure supply chains, COTS devices come from unsecured supply chains that are susceptible to tampering [45]. Also, the companies that design COTS devices are often relatively untrusted, and do not typically follow any formal development process to ensure safety and security (e.g., RTCA DO-254) [45]. In addition to COTS suppliers, BE devices in spaceflight commonly come from university research groups and laboratories [30]. In any of these organizations, a rogue employee, student, or team could alter the device with the malicious circuit and software [47], [48]. A simple *ticking timebomb* [47] trigger, which enables malicious behavior after a configurable amount of time, could be used to activate the circuit and software after the network is deployed — without requiring any input from the attacker.

Even in critical industries like spaceflight and aviation, such malicious hardware and software is not likely to be caught by the system integrator. The reason is that, besides through well-known means like causing an explosion or fire, there has been no known way for non-critical BE devices in a TTE network to disrupt the operation of critical TT devices. As a result, verification of these BE devices is limited to ensuring they do not contain dangerous substances, will survive the operating environment (e.g., vibration and thermal qualification), and perform their intended function [31], [45]. For example, explosives are detected by swabbing, and other dangerous materials are detected through outgassing tests [49]. However, no detailed analysis of the circuit components, circuit layout, or software is performed [31], [45], [50], [51]. The malicious circuit and software needed for PCSPOOF cannot be detected by such basic safety testing. Moreover, a ticking timebomb trigger could simply delay activating the malicious circuit and software until after all functional testing is completed [47].

Second, instead of compromising the system at design time, an attacker could connect a malicious BE device to the network after the network is deployed. For example, TTE allows a factory to share switches between the assembly line and non-critical hardware, like laptops used for monitoring and analysis [17], [52]. If an employee could be tricked into plugging a malicious device (e.g., a USB to Ethernet dongle) into one of these switches, for example, through a supply chain attack (as above) or social engineering, they could inadvertently disrupt the control of critical plant processes and halt production of the entire facility. Alternatively, consider a future commercial airplane that shares a TTE network between the passenger cabin and vehicle control systems.[1] Modern airplanes contain exposed seat electronics boxes under the seats for connecting entertainment units to the passenger network [38]. If a passenger has knowledge of the connectors used, they could secretly disconnect one of these electronics boxes during a flight, plug in a malicious device [38], and interfere with the safe operation of the aircraft — even if the vehicle control data is all encrypted.

We stress that in all the above cases, an attacker with a connection to only a *single* network plane can disrupt TTE devices *throughout* the network and on *all* planes (see §VI). We also note that the connection from the attacker to the TTE network could span more than a single Ethernet cable. The attack works even if the connection is made via a series of several cables, patch panels, and Ethernet jacks.

Lastly, we stress that, besides the single BE device, the attacker has no access to or knowledge of any part of the TTE network. In particular, the attacker has no information about the TT network schedule or the position of devices within the network. Also, the attacker cannot receive any TT messages, or access any telemetry or diagnostic information from the TTE switches or end systems.

IV. Design

This section describes PCSPOOF, the first attack capable of disrupting critical TTE traffic flows and interrupts from a BE device. PCSPOOF achieves this goal by disrupting the TTE synchronization protocol [2]. Disrupting synchronization lets PCSPOOF potentially disrupt *any TT traffic flow*, without needing the attacker to know what traffic flows exist or what they are used for. It also makes PCSPOOF broadly applicable, since all TTE networks use the same synchronization protocol [2].

Of course, disrupting the TTE synchronization protocol from a BE device should be impossible. The protocol is formally verified to work correctly in spite of a malicious TTE end system and any number of malicious BE devices [15].

To overcome this challenge, we use two key observations: (1) an attacker can deduce secret information, known only to TTE devices, in order to create malicious protocol control frames (PCFs), and (2) an attacker can use EMI generated from a BE device to inject these malicious PCFs into the network and get them accepted by TTE devices.

Figure 3 gives an overview of PCSPOOF. The attack proceeds in two stages. In the first stage, the attacker learns *how* to craft authentic-looking PCFs, which requires two pieces of information. The first is the *critical traffic marker*, a special bit pattern found at the start of every PCF. The second is

[1] While, to our knowledge, no commercial airplanes *currently* share switches between the passenger cabin and critical devices, device manufacturers have advocated that TTE's isolation guarantees would make such sharing safe, while reducing size, weight, power, and cost [5], [6].

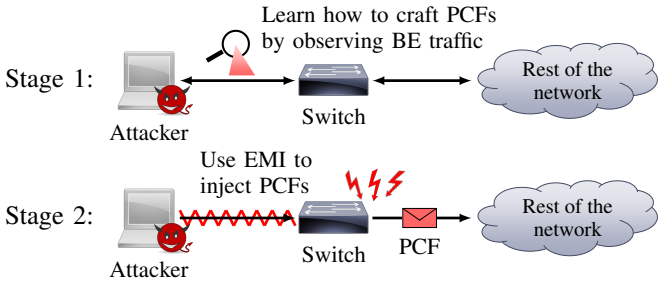

Fig. 3: High-level overview of PCSPOOF.

a *virtual link ID*, which identifies the switch that sends a given PCF. As we will show in §IV-A, the first value is found *indirectly* by observing how the switches respond when forwarding different types of BE traffic. The second value is inferred using knowledge of common network scheduling practices and public hardware documentation.

In the second stage, the attacker injects malicious PCFs into the network. However, since switches block PCFs from BE devices, this requires somehow *bypassing* the switch. To accomplish this, we leverage the fact that, by conducting EMI through the Ethernet cable and into the switch, it is possible to induce link resets *in different switch ports*. These link resets can be used to "transform" BE traffic, which the attacker is allowed to send, into PCFs *as they leave the switch*. Since the transformation happens downstream of the switch logic, it cannot be prevented by extra switch error checking or self-checking pair processors [53].

Next, we describe in detail how to craft malicious PCFs (§IV-A) and inject them into the network (§IV-B).

A. Stage 1: Crafting Malicious PCFs

In the first stage of PCSPOOF, the attacker learns how to craft authentic-looking PCFs. Below, we describe the structure of a PCF, as well as how to obtain the information necessary to make injected PCFs look legitimate.

Anatomy of a PCF. In general, the structure of a PCF is the same as a standard minimum-sized IEEE 802.3 Ethernet frame [2], [54]. However, unlike standard Ethernet frames, PCFs are not forwarded according to a destination media access control (MAC) address. Instead, the first 6 bytes of the frame, which would normally contain the destination MAC address, are replaced with the following two fields:

- *Critical Traffic Marker* — A special value used to identify all PCFs and TT traffic in the network.
- *Virtual Link ID* — Identifies the source of the PCF. For our purposes, it typically identifies a switch.

In order for a PCF to be seen as legitimate, these fields must both match values specified in the network schedule loaded onto the TTE devices when the network was deployed.

PCFs also contain several other fields. However, unlike the critical traffic marker and virtual link ID, it is easy for an attacker to pick suitable values for these fields. This is because either (1) the range of acceptable values is very small or (2) the fields are simply not checked by the TTE hardware. For example, since the virtual link ID identifies the source of a frame, the source MAC address field in the Ethernet header is not checked in practice. We tested a large array of modern and legacy TTE hardware (listed later in Table I), and found that it can be set to any value.

In Appendix A, we list all the other fields found in a PCF, as well as suitable values for these fields for the purpose of the PCSPOOF attack. Since the critical traffic marker and virtual link ID are the only fields that are difficult for an attacker to select, we focus the rest of this section on how an attacker can determine them.

Finding the critical traffic marker. Generating authentic-looking PCFs requires the attacker to find the critical traffic marker used in the network schedule. To accomplish this, they can take advantage of the following rules, which TTE switches use when determining how to forward frames [2].

- If the destination MAC address contains the critical traffic marker, the virtual link ID is valid, and the frame comes from a known TTE device, the frame is forwarded according to the TTE schedule.
- If the destination MAC address contains the critical traffic marker but the virtual link ID is invalid, or the frame comes from a BE device, the frame is dropped.
- If the destination MAC address does *not* contain the critical traffic marker, the frame is forwarded according to the rules of IEEE 802.3 (standard Ethernet) [54].

From these rules, we see that all frames sent by BE devices should be delivered (as bandwidth allows), *except* those containing the critical traffic marker. Thus, an attacker can *infer* the critical traffic marker by tricking other BE devices into sending the attacker frames containing *possible* critical traffic markers and checking which frames *do not arrive*. Below, we describe one method for accomplishing this by abusing the Address Resolution Protocol (ARP) [32], which is used by nearly all BE Ethernet devices.

To start, the attacker must find the IP address of another BE device in the network, which we refer to as the *target*. Any device can be used, such as a router used for passenger Wi-Fi in an airplane, or an inventory management computer in a factory. To get the target's IP address, the attacker sends Internet Control Message Protocol (ICMP) echo requests to all IP addresses in the subnetwork and sees who responds. The standard `fping` utility can do this out of the box, and the process takes tens of seconds even in large networks.

Next, the attacker cycles through a list of possible critical traffic markers. For each one, the attacker sends an ARP request to the target saying "Which MAC address goes with IP X? Tell MAC Y," where X is the IP address of the target, and Y is the MAC address containing the critical traffic marker to test. Upon receiving this message, the target replies to MAC address Y with the target's MAC address.

Assuming the attacker spoofs their source MAC address as Y in each ARP request, each reply for which Y *does not* contain the critical traffic marker is forwarded to the attacker. This is because, with each ARP request, the switch learns to associate MAC address Y with the attacker's port. Otherwise,

the reply is dropped. Thus, the attacker can identify the critical traffic marker by sending an ARP request for each possible critical traffic marker and checking which request gets no reply. To handle the fact that BE messages can be dropped for reasons unrelated to the critical traffic marker (e.g., buffer overflows), the attacker repeats this process in phases; in each phase, only testing critical traffic markers for which no reply was received previously.

There are only around 1 billion possible critical traffic markers [1], [2], so brute forcing the critical traffic marker is fast in practice. We used a Raspberry Pi 4 to find the critical traffic marker in multiple representative spacecraft networks with real surrogate spaceflight hardware. It took only 6–7 hours on average when sending ARP requests at 100 Mbps, and 24 hours when sending requests at 25 Mbps.

We note that, since the critical traffic marker is part of the TTE schedule, it typically does not change over a system's lifetime [55]. The reason is that the schedule typically undergoes a thorough verification and validation process [56]–[58]. Changes to the schedule can require repeating this process, which is expensive and time consuming [55], [56]. This means the attacker does not need to determine the critical traffic marker all at once, or at the same time as they execute the rest of the attack.

Finally, we acknowledge that if all BE devices in the network were configured to use static MAC/IP mappings and drop ARP requests, the method described above would not work. However, when testing on real COTS devices used in flight (e.g., routers), we found that these devices respond to ARP requests. Also, our discussions with avionic designers have revealed industry is explicitly embracing ARP to avoid the complexity of managing static MAC/IP mappings.

However, we note that even if ARP is disabled on all BE devices, an attacker can still easily find the critical traffic marker by using two malicious devices. One device simply sends frames with every possible critical traffic marker to the other device. Since the switch does not know the identity of every BE device, it will flood each frame out of all ports. The second device then tracks which frames are not received. This method cannot be prevented in any TTE switch that has default routes enabled, and we have successfully tested it on a variety of real spaceflight switches.

Finding the virtual link ID. The last piece of information the attacker needs to generate PCFs is the virtual link ID corresponding to a *real* compression master (i.e., switch) that generates PCFs in the network. That way, once a PCF is injected on a given network segment, downstream TTE devices cannot tell that the injected PCF is illegitimate.

Theoretically, the virtual link ID could be any 16-bit number, so there are 65536 possibilities. However, there are two pieces of information an attacker can use to reduce the number of possible virtual link IDs to 2 or fewer.

First, even though there are *theoretically* 65536 possible virtual link IDs, existing switches do not support that many. Also, the number of IDs that switches *do* support is public information. For example, TTTech's Space ASIC, which is used in NASA's Gateway and ESA's Ariane 6 launcher, only supports 4096 virtual link IDs [59]. TTTech's aircraft switches are limited to the same number [60].

Second, existing TTE scheduling tools use *extremely* predictable rules for assigning virtual link IDs to PCFs. For example, the most popular scheduling tools assign virtual link IDs in *reverse order* from the maximum value supported by the hardware (i.e., 4096) [61]. Virtual link IDs for sync masters (i.e., end systems) are assigned first, with a different ID used for each of three PCF types [2]. Virtual link IDs for compression masters (i.e., switches) are assigned next, with each switch using the same ID for all PCF types.

As the number of sync masters and compression masters in a system is predictable, so is the virtual link ID the attacker needs. For example, existing switches support at most 8 sync masters [62]. Also, most TTE systems have one compression master per plane. We are not aware of any existing system with more than two compression masters per plane. Thus, in a large system like a spacecraft or aircraft, the virtual link ID needed by the attacker is likely $4095 - (8 \times 3) = 4071$, and more rarely 4070.

As we show in §VI, PCF injection is so fast that, even if the attacker cannot determine the virtual link ID with certainty, they can simply try all possible values until injection succeeds. Also, we note that, like the critical traffic marker, virtual link IDs are part of the TTE schedule [61], so are unlikely to change once a system is deployed [55].

B. Stage 2: Injecting PCFs into the Network

Now that the attacker knows how to construct a PCF, they need a way to inject the PCF into the network. The attacker cannot simply send the PCF directly, since all PCFs sent from BE devices will be dropped by the switch. To overcome this challenge, PCSPOOF uses EMI to "transform" a BE frame, which the BE device is allowed to send, into a PCF.

In order to perform this transformation, the attacker stores a PCF *inside* the payload of a benign BE frame. By carefully corrupting the BE frame in transit, it is possible to then trick the switch into sending the PCF. Attacks that use this general approach, hiding a malicious message inside a benign message, are called *packet-in-packet attacks* [33].

Below, we describe what makes Ethernet susceptible to packet-in-packet attacks, how TTE hardware can prevent these attacks, and how PCSPOOF defeats these defenses.

Packet-in-packet attacks on Ethernet. To understand why Ethernet is susceptible to packet-in-packet attacks, it is first necessary to understand how Ethernet frames are generated and interpreted by Ethernet devices.

Two types of integrated circuits are needed for a device to send and receive Ethernet frames — the *media access controller (MAC)* and the *physical layer transceiver (PHY)*. The MAC is responsible for assembling and validating Ethernet frames, and for passing them between the host processor and PHY. The PHY is responsible for translating these frames between bytes understood by the MAC, and special symbols

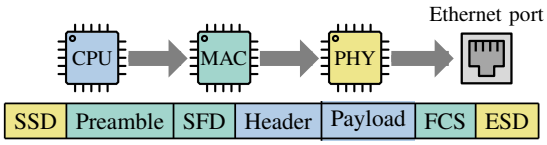

Fig. 4: Frame encapsulation by the MAC and PHY.

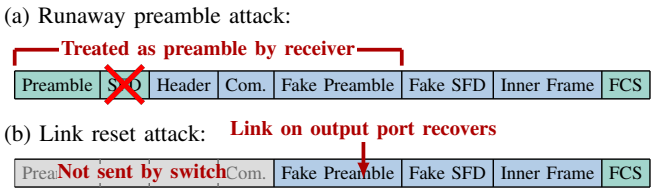

Fig. 5: Packet-in-packet attacks on wired Ethernet.

used at the physical layer, and for writing and reading these symbols to and from the Ethernet wiring.

Figure 4 shows the path of a frame through the MAC and PHY. Each circuit adds additional information to the frame. Specifically, the MAC adds the *preamble* (7 bytes of 0x55), which allows a receiver to "lock on" to the incoming frame, as well as the *start frame delimiter (SFD)* (1 byte of 0xd5), which signals the start of the Ethernet header. The MAC also adds the frame check sequence (FCS) at the end of the frame. The PHY adds a *start-of-stream delimiter (SSD)* to signal the start of the transmission, as well as an *end-of-stream delimiter (ESD)* to signal the end of the transmission.

When receiving a frame, the PHY waits until it sees the SSD, at which point it tells the MAC the preamble is starting. The MAC then reads the preamble until it gets to the SFD byte, at which point it reads in the Ethernet frame. When the PHY receives the ESD symbol, it again signals the MAC, at which point the MAC knows the frame is complete. The last 4 bytes read by the MAC are treated as the FCS.

This design has been shown to be susceptible to two types of packet-in-packet attacks, which we show in Figure 5. In the figure, assume an attacker wants to send a malicious frame (the "inner frame") past a switch to some receiver. However, the switch is configured to drop this frame.

The first type of packet-in-packet attack is a *runaway preamble* attack. Here an attacker exploits the fact that, if a frame's SFD byte is corrupted after the frame is forwarded by the switch, the receiver's MAC will treat this SFD byte (and any following bytes) as preamble [63]. Many MACs do not check that the preamble matches the expected pattern (all 0x55). Thus, by placing a *fake* SFD byte in the frame's payload, immediately before the inner frame, an attacker can trick a receiver into reading the inner frame [63].

The second type of packet-in-packet attack uses *link resets* [64]. In Ethernet, the PHY continuously checks for link pulses and idle symbols produced by the device on the other side of the cable [54]. If these indicators are disrupted, the link is "lost," and the PHY stops transmitting frames. However, the MAC is *not aware* of link status changes and will continue transmitting [64]. An attacker can exploit this by sending a frame (the same structure as above) through the switch while the link is down on the outgoing port. If the attacker is lucky, the link will recover in the middle of the fake preamble being transmitted by the MAC, resulting in only the inner frame actually being sent by the switch.

Device	Preamble		
	Too Long	Too Short	Non-0x55
TTTech Dev. 1G SW	Y, ≤11 bytes	N	N
TTTech PMC 1G NIC	Y, ≤11 bytes	N	N
TTTech A664 Lab SW	Y, ≤1451 bytes	Y, ≥3 bytes	Y, 1st two bytes
TTTech OBC HiRel SW	Y, ≤1451 bytes	Y, ≥3 bytes	Y, 1st byte
TTTech Space Lab SW	Y, ≤1451 bytes	Y, ≥3 bytes	Y, 1st byte
TTTech A664 Lab NIC	Y, ≤1451 bytes	Y, ≥3 bytes	Y, 1st two bytes

TABLE I: Indicates whether TTE switches (SWs) and network cards (NICs) accept PCFs with non-standard preambles. "Too Long, Y, ≤ 11 bytes" means the device accepts PCFs with longer-than-normal preambles up to 11 bytes. Devices labeled "1G" are an older generation of devices.

Note that for either approach to work, the FCS of the original frame must be made to match that of the inner frame. For this, an attacker can exploit the fact that, by adding a 4-byte FCS complement to a frame's payload, it is possible to force the frame's FCS to any value [63]. For more details on calculating and using the FCS complement, see past work on packet-in-packet attacks [63], [64].

Susceptibility of TTE hardware. We tested a wide variety of modern and legacy TTE devices to determine how susceptible they are to both types of packet-in-packet attacks. For each device, we used an XMOS XCORE-200 [65] to generate PCFs with various non-standard preamble and SFD patterns, and determined whether the PCFs were accepted.

Our results are shown in Table I. In all cases, a PCF is only accepted if either (1) the preamble contains only 0x55 bytes, or (2) the preamble *starts* with one or two non-0x55 bytes, but the rest of the preamble is all 0x55 bytes.

These results show that, unlike standard Ethernet hardware [63], TTE hardware can *completely prevent runaway preamble attacks*. There are two reasons. First, the original frame's header, as well as the FCS complement, *will not* be treated as preamble by the receiver unless they both only contain 0x55 bytes. Second, the switch can prevent the frame from ever being forwarded to the receiver by filtering all BE traffic with a destination MAC starting with a valid preamble/SFD pattern (e.g. 0xd5, 0x55d5, 0x5555d5). This is a capability of all TTE switches we have tested.

In contrast, the fact that modern TTE devices accept frames with such long preambles makes them *very susceptible* to link reset attacks. The reason is the attacker can send packet-in-packet frames with 1000+ bytes of fake preamble, maximizing the chance that the link recovers while this fake preamble is being forwarded, *while still* ensuring the resulting PCF is accepted by downstream TTE devices.

Enabling PCF injection. In the past, link reset packet-in-packet attacks on wired Ethernet have been considered impractical, since there was *no known way* for an attacker to cause link resets without physically manipulating the network — e.g., unplugging a cable or rebooting a switch [64].

In contrast, PCSPOOF allows a *networked device*, controlled

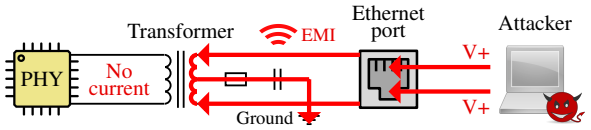

Fig. 6: Effect that a common mode surge on an Ethernet twisted pair has on the inside of a switch.

by the attacker, to cause link resets between the switch and other devices. In general, it accomplishes this by conducting electrical noise into the switch over the Ethernet cable, which results in radiated EMI inside the switch and disrupts the operation of the PHYs on other switch ports.

Figure 6 shows how this EMI is generated in more detail. The figure depicts a malicious BE device connected to a TTE switch. For simplicity, we assume the connection uses a twisted-pair Ethernet cable (e.g., 100BASE-TX), which is the most popular choice today due to cost and reliability [66], [67]. When the cable is plugged in, the wires in the cable are electrically connected to copper traces on the switch printed circuit board (PCB) inside the switch chassis.

Faraday's Law tells us that, by causing rapid high-voltage surges on the wires in the cable, and thus on the above traces, it is possible to generate a changing magnetic field that induces errors in *different* traces and chips on the switch PCB through inductive coupling [68]. Similarly, it is possible to generate strong electric fields that induce errors in parallel traces on the switch PCB through capacitive coupling [68]. Both fields are examples of EMI.

Due to the proximity and parallel orientation of traces and circuitry related to different switch ports on the switch PCB, it is common for EMI generated from one port to cause link resets on other ports. This is due to the EMI directly causing glitches in other PHYs, or causing noise on traces between other PHYs and their respective switch ports.

Of course, it is not possible to cause surges on wires while they are being used for communication. Instead, the attacker has two options. First, they can cause surges on *unused* wires in the cable. For example, a Cat 5/6 cable has 4 twisted pairs, but only two are used for 100BASE-TX communication. Second, the attacker can alternate between using the same pairs for inducing link resets and sending BE frames.

We note that, in addition to causing link resets in *other* ports, PCSPOOF causes link resets in the port connected to the attacker. This is fine; as long as the attacker's link recovers before the outgoing port, meaning the outgoing port could wake up while a frame is in flight, PCF injection is possible. In our tests, this happened about half the time.

Avoid killing the switch. The challenge of using high voltage to induce link resets is that we must be careful not to kill the switch or PHY connected to the attacker. Doing so would close our attack vector and result in the network continuing to operate normally over the redundant planes.

To accomplish this, PCSPOOF takes advantage of the fact that IEEE 802.3 requires galvanic isolation to protect the PHY from large voltage surges on the Ethernet cable, such as from lightning [54]. In twisted-pair Ethernet, this isolation

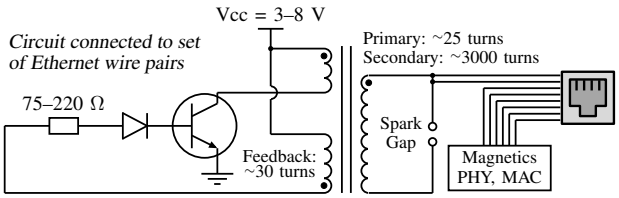

Fig. 7: Simple transformer driver circuit that, when placed inside a BE device, generates EMI inside a switch.

is performed using small transformers, which in TTE switches are packaged in magnetics chips on the switch PCB [69] or on a connected daughter card [60].

Figure 6 shows the design of an Ethernet transformer. A different transformer is connected to each twisted pair in the Ethernet cable. In normal operation, opposite voltages are applied to each wire in a pair, causing current to flow through the primary winding of the transformer. As the current changes, it creates a changing magnetic field, inducing a voltage across the secondary that is seen by the PHY.

This means that, if PCSPOOF caused high-voltage surges on only one wire in a twisted pair, high voltages would be induced in the PHY and kill it. To avoid this, PCSPOOF generates *common mode* surges [70] in which the same voltage is applied to *both* wires in a pair. In this case, a "center tap" in the primary winding allows the current to return to ground. Since current flows in opposite directions from each wire towards the tap, the magnetic fields cancel and high voltage is not directly induced in the PHY.

Example attack circuit. Figure 7 shows a simple circuit capable of generating the common mode surges needed for PCSPOOF. *Disclaimer: The high voltages produced by this circuit can be extremely dangerous.*

The circuit briefly works as follows. When power is applied, current flows to the base of the transistor, turning it on and letting current flow through the primary winding of the transformer. As current in the primary rises, the magnetic field increases in the transformer core, decreasing current in the feedback winding and turning the transistor back off.

When this happens, current abruptly stops flowing through the primary, generating a high-voltage spike known as an *inductive kick*, which we measured as around 100 V.

The kick induces a voltage across the secondary, causing current to rush into the Ethernet cable. Since the secondary has so many turns, this voltage is very large (10–20 kV). Eventually, the voltage gets so large that current arcs across the spark gap, and the secondary voltage drops back to zero.

Meanwhile, the power supply turns the transistor back on, and the process repeats (i.e., the circuit oscillates on its own). This means that, as long the circuit is powered, it will generate EMI in the switch that can cause link resets.

The design of the circuit makes it easy to hide inside another device. Even with large through-hole parts, the whole circuit fits on a 2.5 cm × 2.5 cm PCB. There exist suitable transformers (the largest part) that are just 2.5 cm × 1.25 cm [71] and look like those in typical embedded computers

and power supplies. Similarly, glass-enclosed spark gaps that look like small light emitting diodes are available [72]. Finally, we note that the circuit oscillates so fast that, with small gap sizes, it makes almost no audible noise.

V. Implementation

To evaluate our attack, we implemented PCspooF and executed it on a real TTE testbed used to verify real-life avionic systems. The testbed was designed to mimic a typical fault-tolerant network in a crewed spacecraft or aircraft [7], [10], [11]. Four switches acted as compression masters, and four end systems acted as sync masters. Also, a fifth end system acted as a sync client [2]. The end systems communicated over two redundant planes, each containing half of the switches.

The end systems were implemented on a Dell PowerEdge T620 running CentOS 7.9 with kernel 3.10.0-1160.11. We used TTTech A664 Lab NICs — lab versions of real TTE NICs used in flight. Due to limited hardware availability, we also used older TTTech 1G NICs for some network-level experiments. However, in these cases, the older and newer NICs were verified to behave the same.

For switches, we used modern TTTech OBC HiRel switches [73], as well as older TTTech Development 1G switches in cases where behavior differences were not relevant. We selected the OBC HiRel switch because it is an engineering development unit of a real radiation-hardened spaceflight switch. It also uses TTTech's Space ASIC, which is currently being used in real space vehicles [11], [74].

For scheduling the TTE network, we used TTTech's TTE Tools v5.4 [61], which were the most up-to-date scheduling tools for our hardware, and the same tools used in real systems. We stress that we used the same configurations and settings as are used for real spacecraft avionic systems.

We created an attack device that connected to one TTE switch in one plane. The device used a Raspberry Pi (RPi) 4B with Ubuntu 20.04 LTS and kernel 5.4.0-1041 to run the PCspooF code. The device used a high-voltage circuit, based on Figure 7, to induce link resets and enable PCF injections. The RPi communicated using 100BASE-TX, and the two unused cable pairs carried the high voltage signal.

We used SF/FTP shielded Cat 6A cables for all the connections between the various devices. The connection from the attack device to the switch consisted of a 10 m cable, an Ethernet coupler, and a 3 m cable.

VI. Evaluation

As we showed in §IV-A, an attacker can reasonably determine how to craft a legitimate PCF in a matter of hours, even with modest embedded hardware and limited network bandwidth. Therefore, we focused our evaluation on assessing the *likelihood* and *impact* of successful PCF injections.

Specifically, we conducted a series of experiments on our testbed to answer four key questions: (1) What is the probability of successfully injecting a PCF? (2) How much does a PCF injection disrupt synchronization between TTE devices? (3) How much does a PCF injection disrupt the delivery of TT messages? and (4) How much damage do PCF injections cause in a real spaceflight application?

A. Probability

Experimental setup. To determine the probability of successfully injecting a PCF, we needed to answer two questions. Question 1: How often is an attacker able to inject a PCF — i.e., transform a BE frame into a PCF that gets forwarded by the switch connected to the attack device? Question 2: Given a PCF is injected, how often does a *downstream* TTE device, which receives the PCF from the switch, accept that PCF? Moreover, we wanted to determine how the network settings (e.g., transmission rate, background traffic load, drop rate) impacted the answers to these questions.

To answer Question 1, we varied the distance between the switch port connected to the attack device (attack port) and the nearest switch port connected to a TTE device (target port) from 4 ports away (14 cm) to 7 ports away (21 cm), where 7 ports is the maximum separation between two ports on the OBC HiRel switch. For each distance, we continuously sent a 1500-byte BE frame containing a PCF from the attack device to the switch for 5 minutes. To induce link resets in the target port, we enabled/disabled the high-voltage circuit approximately every 1.5–2 s. We used a Fluke OptiView XG analyzer to capture frames forwarded out of the target port and identify the injected PCF.

We repeated the above experiments in four different setups. In the first, there was no background traffic, and we varied the attacker's transmission rate from $\{25, 50, 100\}$ Mbps, where 100 Mbps is the maximum rate of the network. In the second, we configured background BE traffic flowing through the switch and out the target port to consume all but $\{20, 50, 80\}$ Mbps of the bandwidth, and the attacker to send at the maximum rate. This range was chosen to span the network's total bandwidth, while reflecting the fact that real systems leave bandwidth margin for performance reasons and to enable future expansion [75]. In the third, we repeated the second setup except with background TT traffic instead of BE traffic. In the fourth, we configured background TT traffic to flow at 20 Mbps, background BE traffic to flow at 70 Mbps, the attacker to send at the maximum rate, and the network to drop $\{0.01, 0.1, 1\}$% of all frames. The fourth setup is a realistic representation of real systems, where TT traffic is commonly limited to 10–20 Mbps [76], [77], and a bandwidth margin of at least 10% is typical [78].

To answer Question 2, we varied the integration cycle of the network from $\{0.5, 1, 2, 4, 8\}$ ms, where a smaller integration cycle causes tighter synchronization. This range reflects values used in real avionic systems [79]–[81]. For each cycle, we used a hardware tap to inject 1000 PCFs on the link between two switches, and recorded the number of injections that caused at least one end system to lose synchronization. We then repeated the process for a link between a switch and end system.

We performed the above experiment in four different setups representing the most extreme configurations from Question 1: (1) no background traffic, (2) 80 Mbps of background BE

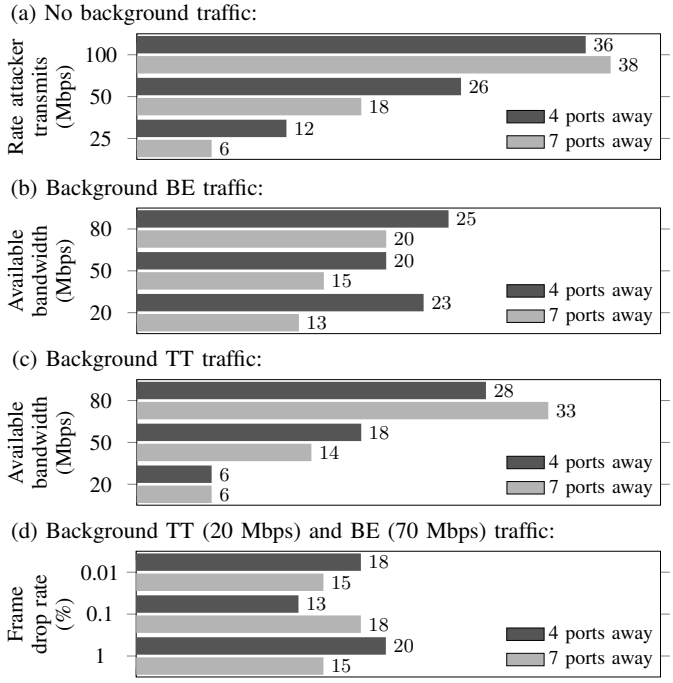

Fig. 8: Number of PCF injections in 5 minutes under different conditions (e.g., background traffic, drop rate) and with different distances between attack and target ports.

traffic, (3) 80 Mbps of background TT traffic, and (4) 20 Mbps of background TT traffic, 70 Mbps of background BE traffic, and a 1% drop rate. We report results for coldstart acknowledgement (CA) PCFs only, as the results for integration (IN) PCFs are nearly identical.

Results. Figure 8 shows our results for Question 1. As the figure shows, PCSPOOF can inject PCFs in *a matter of seconds*. This is true even in configurations with heavy background traffic. For example, in a realistic configuration with 90% of the bandwidth consumed by TT and BE traffic, and with an extremely high 1% drop rate [82], we observed 20 injections in 5 minutes (roughly one every 15 seconds). The injection rate decreases as the amount of background TT traffic increases, since the attacker is limited to a smaller portion of the bandwidth, resulting in more of the attacker's frames being dropped. However, even in the unlikely case of TT traffic consuming 80% of the bandwidth [76], we still observed more than one injection per minute on average. The figure also shows that PCF injections are possible when the attack and target ports are far apart from each other.

Tables II and III show our results for Question 2. With a 1 ms integration cycle, which is common in practice [81], switches accepted 3.9% of injected PCFs under realistic traffic loads and a 1% drop rate. When combined with the results from Figure 8, this means PCSPOOF is likely to inject a PCF *and* get it accepted by a switch *in 6–7 minutes*. Even with a larger 4 ms integration cycle, PCSPOOF is likely to inject a PCF and get it accepted by a switch in 30–40 minutes. Unlike switches, end systems accept all PCFs they receive (i.e., all injections succeed). Thus, PCSPOOF is likely to inject a PCF and get it accepted by an end system *in tens of seconds*, regardless of the integration cycle.

Integration Cycle	Background Traffic			
	None	BE	TT	TT+BE (1% drop)
8 ms	0.4%	0.3%	0.3%	0.5%
4 ms	0.7%	0.7%	0.5%	0.8%
2 ms	2.0%	1.7%	1.8%	1.6%
1 ms	4.4%	4.2%	3.3%	3.9%
500 μs	8.0%	7.1%	7.2%	6.3%

TABLE II: Percentage of injected PCFs that were accepted on links to switches.

Integration Cycle	Background Traffic			
	None	BE	TT	TT+BE (1% drop)
8 ms	100%	100%	100%	99.0%
4 ms	100%	100%	100%	99.1%
2 ms	100%	100%	100%	99.0%
1 ms	100%	100%	100%	99.1%
500 μs	100%	100%	100%	98.8%

TABLE III: Percentage of injected PCFs that were accepted on links to end systems.

B. Interrupts

Experimental setup. TTE uses synchronized periodic interrupts to coordinate the execution of software on the end systems [7]. Often, two interrupts are used, a *major* interrupt and a *minor* interrupt, where the major interrupt period divides evenly by the minor interrupt period [20], [83], [84].

By interfering with these interrupts, an attacker could cause significant problems that systems are not designed to handle, such as end systems performing computations on old information, sending data when it is not expected, or failing to generate outputs when needed [42], [85], [86].

To determine how PCSPOOF affects these interrupts, we configured our testbed with a 1 s major interrupt, 25 ms minor interrupt, and 4 ms integration cycle. These values are commonly used in real systems [84], [87], and match those in our case study of a real spaceflight mission (§VI-D). We note that 4 ms is the smallest integration cycle allowed with our interrupt configuration [61], and minimizes the time it takes the network to recover from PCF injections.

We used a hardware tap to perform 250 successful PCF injections on both inter-switch links and links between switches and end systems. We repeated this process for both CA and IN PCFs, and report the time between the interrupts immediately before and after each injection. We also report interrupt timing for a 5 minute control case, in which no PCFs were injected.

Results. Figure 9 shows our results; a single PCF injection can significantly disrupt interrupt timing. For example, a single PCF injection on an end system link can delay the major interrupt by *more than a second* and the minor interrupt by *more than 25 ms*. An inter-switch link injection can delay the minor interrupt by *more than 40 ms*.

In addition, PCF injections on inter-switch links cause interrupt delays *on multiple end systems at once*. This happens because the injected PCF is forwarded to multiple end systems. Importantly, this means that N-modular redundancy [88], a fault tolerance technique where the same function is performed on multiple end systems, cannot protect systems

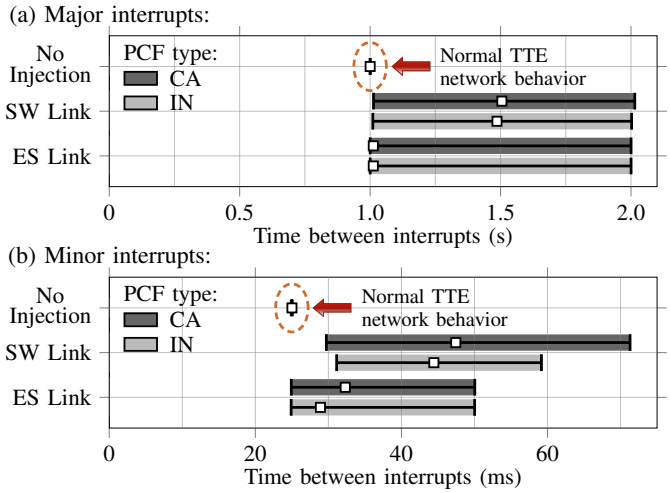

Fig. 9: Average-Max-Min charts showing the time between interrupts following successful PCF injections on links to switches (SWs) and end systems (ESs).

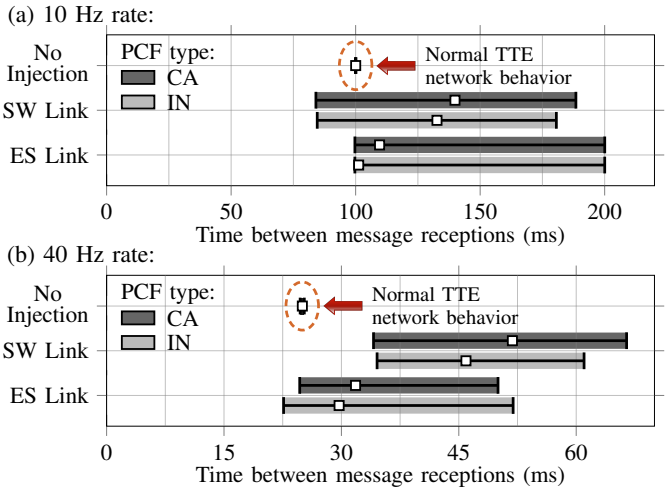

Fig. 10: Average-Max-Min charts showing the message delays following successful PCF injections on links to switches (SWs) and the sender end system (ES).

Data Rate:	10 Hz				40 Hz			
PCF Type:	CA		IN		CA		IN	
Link Type:	SW	ES	SW	ES	SW	ES	SW	ES
Min Drops:	20	20	20	20	20	19	20	20
Max Drops:	21	20	21	20	21	20	21	20

TABLE IV: Message drops following successful PCF injections on links to switches and the sender end system.

from PCSPOOF. A single injection could simply delay the interrupts on *all* redundant end systems simultaneously.

In safety-critical systems, where N-modular redundancy is widely trusted for important functions [76], [89], [90], these delays could be disastrous. For example, in automobiles, steering outages exceeding 50 ms can be non-recoverable [34]. Similarly, aircraft can require inputs as often as every 40 ms to avoid failures [35]. Moreover, as we show in §VI-D, even in cases where a single widespread outage (such as from an inter-switch injection) does not cause system failure, repeated isolated outages (such as from end system injections) can cause redundant systems to fail.

Moreover, we observed that, in the worst case, a single PCF injection can disrupt the interrupt timing of *all end systems in the network*. This is because, when used in a fault-tolerant configuration, TTE requires 3 sync masters to be operational for synchronization between *any end systems* to be possible [61]. Thus, if enough end systems lose synchronization, all end systems do — even ones that never receive the injected PCF.

C. Messaging

Experimental setup. To determine the effect PCSPOOF has on TT messaging, we repeated the experiment from §VI-B, with one end system configured as the *sender*, and the others as *receivers*. The sender continuously wrote messages with 100-byte payloads to its NIC, representing typical traffic in embedded systems [91]. The messages were stored in a queuing buffer of default size for our hardware [61]. The network was scheduled to continuously transmit the oldest message in the queue to the receivers at a rate from $\{5, 40\}$ Hz, representing the minimum and maximum data rates typically found in real systems [42], [87].

For each rate, we used a hardware tap to perform 250 successful PCF injections on both inter-switch links and links between switches and end systems. We repeated the process for both CA and IN PCFs, and report the time between when an end system last received a message before each injection and next received a message after the injection. We also report the number of message drops caused by each injection — i.e., the number of times the sender successfully stored a message in its NIC, but the message was not received. Finally, we report results for a 5 minute control case with no PCF injections.

Results. Figure 10 shows our results. As expected, by disrupting synchronization, PCF injections can cause large message delays. For example, a single PCF injection caused a message expected every 25 ms to not arrive for *up to 65 ms*. As discussed in §VI-B, such delays can be disastrous for critical applications like steering and engine control, where delays beyond 40–50 ms can be nonrecoverable [34], [35].

Table IV shows the number of message drops caused by each PCF injection. In summary, each successful injection resulted in approximately 20 message drops in a row for all receivers. Thus, successful PCF injections do not only result in message delays but also cause TT messages *to be permanently lost*. Interestingly, we observed a similar number of drops regardless of the rate at which messages were transmitted, seemingly due to messages being purged from NIC and switch buffers when synchronization is disrupted.

We stress that PCF injections caused these message drops even though TT traffic *travels over multiple planes simultaneously*, and PCFs were only injected *on a single plane*. Therefore, redundant communication paths are not an effective way of mitigating PCSPOOF. The reason is that, since PCSPOOF disrupts the synchronization protocol, it disrupts communication on all planes simultaneously.

Fig. 11: NASA spaceflight simulation without (left) and with (right) PCSPOOF. PCSPOOF caused a significant deviation in the vehicle's flight path, which prevented docking.

D. Case Study: NASA Asteroid Redirect Mission

To determine how much damage PCSPOOF causes in a real spaceflight application, we conducted a case study based on NASA's planned Asteroid Redirect Mission [92], in which a robotic spacecraft would move an asteroid into a stable orbit around the Moon. A crewed spacecraft, such as NASA's Orion, would then carry astronauts to the asteroid in order to study it, take samples, and return the samples to Earth.

We executed a subset of the mission on a real avionics testbed, during which a representative Orion capsule approached and attempted to dock with the robotic spacecraft. The Orion guidance software, which included several genuine Orion flight software components (e.g., for optical navigation) [20], ran against NASA's Trick Simulation [93], which modeled the vehicles in space, as well as Orion's sensors and actuators. The Orion subsystems and simulation communicated over a fault-tolerant TTE network, similar to Figure 1. We used network settings from the real mission.

We executed the mission twice. In the first trial, no PCFs were injected. In the second trial, we executed the full end-to-end attack, including finding the critical traffic marker and injecting PCFs with the attack circuit. After determining the rate at which PCFs were injected on links to the flight computers (roughly every 16 seconds), we switched to injecting PCFs on those links with a hardware tap. This let us assess the impact of PCSPOOF over a long mission (hours), without risk of damaging the switch.

Our results are shown in Figure 11. As expected, in the absence of PCF injections, the mission completed successfully. Orion approached the robotic spacecraft at a relative velocity of 2–3 m/s until it was approximately 300 m away, aligned itself with the robotic spacecraft, and proceeded straight at 0.1–0.5 m/s until docking was complete.

In contrast, PCSPOOF caused message drops and delays that caused Orion to deviate from its intended flight path. Rather than aligning with the robotic spacecraft, Orion swung underneath it at a distance of approximately 115 m, missed the docking opportunity, and floated away at a rate of 1–2 m/s. These results show that PCSPOOF can significantly disrupt the operation of critical systems that rely on TTE. PCSPOOF also threatens safety, as the uncontrolled maneuvers we observed could easily cause collisions with other objects or vehicles.

VII. MITIGATIONS

In this section, we discuss potential mitigations to PCSPOOF.

Block EMI. PCSPOOF enables PCF injections by conducting EMI into the switch over an Ethernet cable. This interference can be prevented by using optocouplers or surge protector devices between the Ethernet cables and TTE switch ports. However, such devices often suffer from performance limitations [94], decrease system reliability by introducing new points of failure, and can increase size, weight, and power due to the inclusion of new hardware [95].

Another option is to use fiber-optic cables, which are incapable of conducting EMI into the switch. However, such cables have several downsides compared to copper, including higher cost, worse durability, and decreased compatibility with commercial hardware [67]. For these reasons, most TTE systems use copper physical layers [8], [96]–[98].

Compression master placement. PCSPOOF requires injected PCFs to look like they came from real compression masters (CMs). By carefully placing the CMs, designers can ensure injected PCFs will not be used. For example, if CMs and BE devices are placed on opposite sides of the network, injected PCFs will come from paths with no CMs, and thus be ignored. However, this separation may not be possible in networks with few switches, and can increase size, weight, and power by requiring long cable runs between where BE devices are needed and allowed to connect to the network.

Use link-layer security. One way to mitigate PCSPOOF is to use a link-layer authentication protocol, like IEEE 802.1AE [99]. Unless the attacker knows the cryptographic key used in the network, they cannot inject PCFs that are accepted by TTE devices. Unfortunately, link-layer security is not implemented in existing TTE devices. Adding authentication would require updates to the TTE hardware, as well as impact compatibility with existing TTE systems.

Check the source MAC address. As shown in §IV-A, TTE devices do not check the source MAC address field in PCFs they receive. This means attackers do not need to determine the MAC address of a real compression master in the network in order to craft authentic-looking PCFs. TTE devices could improve security by checking the source MAC address in received PCFs against known correct values. However, doing so would require changes to the TTE hardware.

Check the preamble length. PCSPOOF cuts the headers off frames in flight, causing receivers to get injected PCFs with potentially very long preambles. TTE devices would be less likely to accept these PCFs if they rejected frames with long preambles. This would force attackers to send smaller BE frames, making the link less likely to recover in the region required for successful injections. However, adding preamble length checks would require updates to the TTE hardware.

Hide key PCF fields. In PCSPOOF, attackers determine the critical traffic marker used in PCFs by sending BE frames and seeing which frames are dropped by the switches. Thus, designers can prevent attackers from finding the critical traffic marker by configuring switches to drop additional BE traffic

that does not contain the critical traffic marker. However, this prevents BE devices whose MAC addresses overlap with the blacklisted addresses from receiving messages, and causes certain BE multicast addresses to become unusable.

Similarly, attackers determine the virtual link IDs used in PCFs by exploiting predictable patterns used by existing schedulers (see §IV-A). Randomly assigning these virtual link IDs, or regularly changing them, would improve security.

Use more sync masters. If PCSPOOF disrupts enough sync masters, it causes the whole network to lose synchronization, regardless of whether all devices received an injected PCF or not (see §VI-B). Increasing the number of sync masters can reduce the probability of this happening, but may not be possible in small systems like automobiles. Also, even if a network has many sync masters, care must be taken in choosing their locations in the network. Otherwise, a single injected PCF could still take out all the sync masters.

Disable dangerous state transitions. PCSPOOF exploits the fact that end systems that receive a coldstart acknowledgement PCF will temporarily lose synchronization (see §II-B). Thus, one way to combat PCSPOOF is to disable this state transition in the configurations loaded on the end systems [2]. Unfortunately, removing this transition also impacts the ability for the network to detect cliques at system startup [2]. It also does nothing to prevent an attacker from injecting integration PCFs, which can also disrupt synchronization (see §II-B).

VIII. Related Work

Attacks on TTE's isolation guarantees. Because of its use in critical applications, much effort has been spent trying to break TTE's isolation guarantees [21], [38], [100], [101]. For example, the Aviation Cyber Security Study [100] analyzed the ability of BE devices to interfere with TT traffic via denial-of-service or MAC flooding attacks. To our knowledge, no successful attacks have ever been reported. TTE's security has also been studied in much less restricted threat models, such as when an attacker controls critical TTE devices [102], [103], or has physical access to the system and can thus unplug cables and intercept messages [40], [104]. In contrast, PCSPOOF does not require the attacker to have physical access to the system, and it can succeed from a single BE device connected to a single network plane.

Ethernet packet-in-packet attacks. Several packet-in-packet attacks have been developed [33], [63], [64], [105]. Recently, EtherOops [63] showed that runaway preamble attacks on wired Ethernet were possible by exploiting data corruptions, like those caused by faulty cables. However, as shown in §IV-B, this attack can easily be prevented in modern TTE networks. Other researchers showed that Ethernet packet-in-packet attacks could be accomplished by exploiting link resets, like those caused by unplugging and replugging cables [64]. However, the attacker had no ability to cause these link resets to occur. PCSPOOF also uses link resets to let an attacker inject malicious PCFs into the network. However, importantly, PCSPOOF lets the attacker induce those link resets at will from a networked BE device.

EMI attacks on Ethernet. Several studies have explored methods for inducing errors in Ethernet networks by exposing switches and cables to EMI [63], [106]–[109]. For example, [107] studied the susceptibility of office networks to nearby electromagnetic pulse devices. However, such attacks are mostly effective only on networks with unshielded cables, require the attacker to be in close proximity to the network (e.g., a few meters), and require large antennas to radiate the EMI [63], [107]. In contrast, PCSPOOF induces switch errors by conducting EMI from a networked device, *through* an Ethernet cable, and *into* the switch. This means PCSPOOF works on networks with shielded cables, works from any distance — provided the attack device connects to a switch — and takes up little physical space (see §IV-B).

Timing attacks on real-time systems. There exist several timing-based attacks on TT and other real-time systems [110]–[117]. Generally, these attacks work by (1) using side channels to infer the task or communication schedule [110]–[115], then (2) interfering with critical tasks or traffic by consuming shared resources when they are needed [116], [117]. PCSPOOF is orthogonal to these methods. In particular, PCSPOOF does not rely on knowledge of timing to be successful. Also, while designers can mitigate interference attacks by planning for the worst-case contention that critical TT tasks and messages may experience [116], [117], PCSPOOF can disrupt all TT tasks and messages regardless of temporal overprovisioning.

Destructive high-voltage circuits. Several devices use high voltage to damage electronic equipment [118]–[120]. For example, the venerable EtherKiller [119] destroys Ethernet switches by shorting a switch port to a wall socket. USBKill [118] devices destroy computers by discharging high-voltage capacitors into the computer's USB port. Unlike these devices, PCSPOOF is not intended to damage a TTE switch; doing so would prevent successful attacks. Rather, it is designed to induce controlled errors in the switch that cause link drops on other ports. Also, PCSPOOF is designed to introduce these errors from a fully-functioning Ethernet device.

IX. Conclusion

TTE is a popular choice for mixed-criticality systems because of its ability to share the network between critical TT and non-critical BE devices. However, this design requires that the critical TT services be completely isolated from the BE devices. We presented PCSPOOF, the first attack capable of breaking TTE's isolation guarantees. Our results show that PCSPOOF threatens the safety of critical TTE systems, like spacecraft and aircraft. We hope the description of our attack, as well as the mitigations we identified, will influence the deployment of current TTE systems, as well as the designs of future mixed-criticality network technologies.

Acknowledgments

We thank the anonymous reviewers for their helpful feedback on improving the paper. This research was supported in part by the NSF Graduate Research Fellowship (award DGE 1256260) and NSF grants CNS-1750158 and CNS-1703936.

References

[1] "ARINC 664 P7: Aircraft Data Network Part 7 – Avionics Full-Duplex Switched Ethernet Network," Aeronautical Radio, Incorporated (ARINC), Sep. 2009.

[2] "SAE AS6802: Time-Triggered Ethernet," SAE International, Nov. 2016.

[3] S. Parkes, C. McClements, D. McLaren, A. F. Florit, and A. G. Villafranca, "SpaceFibre Networks: SpaceFibre, Long Paper," in *Proc. International SpaceWire Conference*, Yokohama, Japan, Oct. 2016.

[4] "IEEE 802.1Qbv-2015: IEEE Standard for Local and Metropolitan Area Networks – Bridges and Bridged Networks – Amendment 25: Enhancements for Scheduled Traffic," Institute of Electrical and Electronics Engineers, Mar. 2016.

[5] I. Safiulin and A. Geissler, "Deterministic Ethernet and IEEE TSN for Aerospace," Aug. 2019. http://www.modern-avionics.com/Files/22-TTTech-Safiulin-29.08.2019.pdf.

[6] W. Zischka and M. Jakovljevic, "IEEE TSN in Aerospace: Capabilities and Future Outlook," in *Proc. International Conference on Prospects of Civil Avionics Development*, Moscow, Russia, Jul. 2021.

[7] A. Loveless, "On Time-Triggered Ethernet in NASA's Lunar Gateway," NASA Avionics Architectures Community of Practice, Houston, TX, USA, Jul. 2020. https://ntrs.nasa.gov/citations/20205005104.

[8] "International Deep Space Interoperability Standards – Avionics Standard," National Aeronautics and Space Administration, Mar. 2019. https://www.internationaldeepspacestandards.com/.

[9] M. Felser, "Industrial Ethernet – Real-Time Ethernet for Automation Applications," in *Industrial Communication Technology Handbook*. CRC Press, Aug. 2014.

[10] M. Fletcher, "Progression of an Open Architecture: From Orion to Altair and LSS," Honeywell, International, Tech. Rep. S65-5000-20-0, May 2009.

[11] C. Fidi, I. Masar, J.-F. Dufour, and M. Jakovljevic, "Radiation-Tolerant System-On-Chip (SOC) with Deterministic Ethernet Switching for Scalable Modular Launcher Avionics," in *Proc. ERTS*, Toulouse, France, Jan. 2018.

[12] "GE Fanuc Intelligent Platforms Announces TTEthernet for Safety-Critical Avionics Applications," Apr. 2009. https://www.ge.com/news/press-releases/ge-fanuc-intelligent-platforms-announces-ttethernet-safety-critical-avionics.

[13] "TTTech to Provide ARINC 664 p7 Products for Mission System on UK AW101 Merlin Mk4/4a Helicopters," Apr. 2015. https://www.tttech.com/press/tttech-to-provide-arinc-664-p7-products-for-mission-system-on-uk-aw101-merlin-mk4-4a-helicopters/.

[14] W. Bellamy, "TTEthernet Avionics Backbone a Technology Breakthrough for S-97 Raider," *Aviation Today*, Jul. 2015. https://www.aviationtoday.com/2015/07/20/ttethernet-avionics-backbone-a-technology-breakthrough-for-s-97-raider/.

[15] W. Steiner and B. Dutertre, "The TTEthernet Synchronization Protocols and Their Formal Verification," *Int. J. Crit. Comput.*, 2013.

[16] B. Hall, M. Paulitsch, D. Benson, and A. Behbahani, "Jet Engine Control Using Ethernet with a BRAIN," in *Proc. Joint Propulsion Conference and Exhibit*, Hartford, CT, USA, 2008.

[17] "PCIe Card from TTTech Enables Deterministic Ethernet Connectivity for Industrial PCs," Feb. 2015. https://www.tttech.com/press/pcie-card-from-tttech-enables-deterministic-ethernet-connectivity-for-industrial-pcs/.

[18] T. Steinbach, H. Lim, F. Korf, T. C. Schmidt, D. Herrscher, and A. Wolisz, "Tomorrow's In-Car Interconnect? A Competitive Evaluation of IEEE 802.1 AVB and Time-Triggered Ethernet (AS6802)," in *Proc. VTC*, Quebec City, Canada, Sep. 2012.

[19] T. Steinbach, F. Korf, and T. C. Schmidt, "Comparing Time-Triggered Ethernet with FlexRay: An Evaluation of Competing Approaches to Real-Time for In-Vehicle Networks," in *Proc. WFCS*, Nancy, France, May 2010.

[20] A. Loveless, "On TTEthernet for Integrated Fault-Tolerant Spacecraft Networks," in *Proc. AIAA SPACE*, Pasadena, CA, USA, Aug. 2015.

[21] A. Starke, D. Kumar, M. Ford, J. McNair, and A. Bell, "A Test Bed Study of Network Determinism for Heterogeneous Traffic using Time-Triggered Ethernet," in *Proc. MILCOM*, Baltimore, MD, USA, Oct. 2017.

[22] W. Steiner and M. Paulitsch, "Time-Triggered Ethernet," in *Industrial Communication Technology Handbook*. CRC Press, Aug. 2014.

[23] W. Steiner, "TTEthernet: Time-Triggered Services for Ethernet Networks," in *Proc. DASC*, Orlando, FL, USA, Oct. 2009.

[24] "Time-Triggered Ethernet Slims Down Critical Data Systems," *NASA Spinoff*, 2018. https://spinoff.nasa.gov/Spinoff2018/t_4.html.

[25] R. Obermaisser, "Properties of Time-Triggered Communication Systems – Fault Containment and Error Containment," in *Time-Triggered Communication*. CRC Press, 2011.

[26] D. Escorial and M. Hann, "A Combined Dependability and Security Approach for Third Party Software in Space Systems," in *Proc. EDCC*, Gothenburg, Sweden, Sep. 2016.

[27] J. Fichuk, "Safe to Fly: Certifying COTS Hardware for Spaceflight," NASA Johnson Space Center, Tech. Rep. JSC-CN-22144, Aug. 2013.

[28] "Compositional Assurance Cases and Arguments for Distributed MILS," D-MILS Project Partners, Tech. Rep., Apr. 2015.

[29] R. Obermaisser, "Properties of Time-Triggered Communication Systems – Certifiability," in *Time-Triggered Communication*. CRC Press, 2011.

[30] M. Cichowicz, "From Schenley Place to Outer Space: Pitt Team Developing Computers for Space Station," Jul. 2017. https://news.engineering.pitt.edu/from-schenley-place-to-outer-space/.

[31] "Risk Classification for NASA Payloads," National Aeronautics and Space Administration, Tech. Rep. NPR 8705.4, Jun. 2004. https://nodis3.gsfc.nasa.gov/npg_img/N_PR_8705_0004_/N_PR_8705_0004_.pdf.

[32] "RFC 826: An Ethernet Address Resolution Protocol," Nov. 1982. https://datatracker.ietf.org/doc/html/rfc826.

[33] T. Goodspeed, S. Bratus, R. Melgares, R. Shapiro, and R. Speers, "Packets in Packets: Orson Welles' In-Band Signaling Attacks for Modern Radios," in *Proc. WOOT*, San Francisco, CA, USA, Aug. 2011.

[34] G. Heiner and T. Thurner, "Time-Triggered Architecture for Safety-Related Distributed Real-Time Systems in Transportation Systems," in *Proc. FTCS*, Washington, DC, USA, Jun. 1998.

[35] J. H. Lala and R. E. Harper, "Architectural Principles for Safety-Critical Real-Time Applications," *Proc. IEEE*, no. 1, Jan. 1994.

[36] A. Loveless, C. Fidi, and S. Wernitznigg, "A Proposed Byzantine Fault-Tolerant Voting Architecture using Time-Triggered Ethernet," in *Proc. SAE AeroTech*, Fort Worth, TX, USA, Sep. 2017.

[37] A. Loveless, "On TTEthernet for Integrated Fault-Tolerant Spacecraft Networks [Slides]," in *Proc. AIAA SPACE*, Pasadena, CA, USA, Aug. 2015. https://ntrs.nasa.gov/citations/20170009923.

[38] K. Kainrath, M. Fruhmann, K. Gebeshuber, E. Leitgeb, and M. Gruber, "Evaluation of Cyber Security in Digital Avionic Systems," in *Proc. VTC*, Antwerp, Begium, May 2020.

[39] W. Steiner, G. Bauer, B. Hall, and M. Paulitsch, "Time-Triggered Ethernet – Protocol Services, Communication Services, Media Access Control," in *Time-Triggered Communication*. CRC Press, 2011.

[40] W. Steiner, "Candidate Security Solutions for TTEthernet," in *Proc. DASC*. East Syracuse, NY, USA, Oct. 2013.

[41] W. Steiner, G. Bauer, B. Hall, and M. Paulitsch, "Time-Triggered Ethernet – Protocol Services, Startup and Restart, Clique Detection," in *Time-Triggered Communication*. CRC Press, 2011.

[42] A. Loveless, R. Dreslinski, B. Kasikci, and L. T. X. Phan, "IGOR: Accelerating Byzantine Fault Tolerance for Real-Time Systems with Eager Execution," in *Proc. RTAS*, Nashville, TN, USA, May 2021.

[43] T. Bonk, W. T. Smithgall, M. Fletcher, and G. Carlucci, "System and Method for a Cross Channel Data Link," US Patent US20 100 183 016A1, Jul., 2010.

[44] M. Plankensteiner, "TTTech Company Overview," TTTech Computertechnik AG, Mar. 2011. https://www.slideshare.net/TTTech/tttech-2011companyoverview.

[45] R. F. Hodson, Y. Chen, J. E. Pandolf, K. Ling, K. T. Boomer, C. M. Green, J. A. Leitner, P. Majewicz, S. H. Gore, C. S. Faller, E. C. Denson, R. E. Hodge, A. P. Thoren, and M. A. Defrancis, "Recommendations on Use of Commercial-Off-The-Shelf (COTS) Electrical, Electronic, and Electromechanical (EEE) Parts for NASA Missions," National Aeronautics and Space Administration, Tech. Rep. NESC-RP-19-01490, Dec. 2020.

[46] "Ensuring Successful Implementation of Commercial Items in Air Force Systems," United States Air Force Scientific Advisory Board, Tech. Rep. SAB-TR-99-03, Apr. 2000.

[47] T. Trippel, K. Shin, K. Bush, and M. Hicks, "Bomberman: Defining and Defeating Hardware Ticking Timebombs at Design-time," in *Proc. S&P*, Oakland, CA, USA, May 2021.

[48] S. King, J. Tucek, A. Cozzie, C. Grier, W. Jiang, and Y. Zhou, "Designing and Implementing Malicious Hardware," in *Proc. LEET*, San Francisco, CA, USA, Apr. 2008.

[49] M. Orlandi, T. Rohr, M. Stienstra, and C. Semprimoschnig, "The Role of ESA TEC-QTE in the ISS Safety Process," in *Proc. IAASS*, Montreal, Canada, May 2013.

[50] G. Johnson-Roth, "Mission Assurance Guidelines for A-D Mission Risk Classes," The Aerospace Corporation, Tech. Rep. TOR-2011(8591)-21, Jun. 2011.

[51] C. M. Wilson, J. Stewart, P. Gauvin, J. Mackinnon, J. Coole, J. Urriste, A. George, G. Crum, E. Timmons, J. Beck, T. Flatley, M. Wirthlin, A. Wison, and A. Stoddard, "CSP Hybrid Space Computing for STP-H5/ISEM on ISS," in *Proc. Conference on Small Satellites*, Logan, UT, USA, Aug. 2015.

[52] "New Semi-Rugged Laptop for Manufacturing and Automotive Professionals," Jan. 2021. https://www.logisticsbusiness.com/it-in-logistics/mobile-computing-rfid/new-semi-rugged-laptop/.

[53] A. Deutschinger, "TTE Controller," EtherSpace Consortium, Jun. 2021. https://www.tttech.com/wp-content/uploads/2021-04-22_P1b_Webinar-EtherSpace_TTTech_presentation.pdf.

[54] "IEEE 802.3-2018: IEEE Standard for Ethernet," Institute of Electrical and Electronics Engineers, Jun. 2018.

[55] B. Luksik, A. Loveless, and A. George, "Gatekeeper: A Reliable Reconfiguration Protocol for Real-Time Ethernet Systems," in *Proc. DASC*, San Antonio, TX, USA, Oct. 2021.

[56] "TTE Verify," TTTech Computertechnik AG, Aug. 2022. https://www.tttech.com/products/products/aerospace/development-test-vv/verification-tools/tte-verify/.

[57] A. Basu, S. Bensalem, M. Bozga, B. Delahaye, A. Legay, and E. Sifakis, "Verification of an AFDX Infrastructure Using Simulations and Probabilities," in *Proc. RV*, Berlin, Germany, Nov. 2010.

[58] "iSAFT Test Tool for Deterministic On-board Ethernet Networks," in *Proc. ESA Workshop on ADCSS*, Noordwijk, The Netherlands, Nov. 2019. https://indico.esa.int/event/323/contributions/5041/attachments/3743/5198/11.30b_-_Teletel_-_Ethernet_for_Space_with_TSN.pdf.

[59] "TT6802-1-SW: The TTE Switch Controller Space," TTTech Computertechnik AG, Aug. 2022. https://www.tttech.com/wp-content/uploads/TTTech_12792_TTE-Switch_Controller_Space.pdf.

[60] "TTE Switch Module A664 Pro," TTTech Computertechnik AG, Aug. 2022. https://www.tttech.com/wp-content/uploads/TTTech_TTE-Switch_Module_Core_2-1.pdf.

[61] "TTE-Plan User Manual," TTTech Computertechnik AG, Tech. Rep. D-TTE-G-01-014, version 5.4.6, Jul. 2020.

[62] "TTE Switch Space 3U cPCI," TTTech Computertechnik AG, Aug. 2022. https://www.tttech.com/products/products/space/flight-rugged-hardware/ttethernet-space-equipment/switch-space-3u-cpci/.

[63] B. Seri, G. Vishnepolsky, and Y. Yusepovsky, "EtherOops: Bypassing Firewalls and NATs by Exploiting Packet-in-Packet Attacks in Ethernet," Armis, Tech. Rep., Aug. 2020.

[64] A. Barisani and D. Bianco, "Fully Arbitrary 802.3 Packet Injection – Maximizing the Ethernet Attack Surface," in *Proc. Black Hat*, Las Vegas, NV, USA, Jul. 2013.

[65] "XCORE-200," XMOS, Aug. 2022. https://www.xmos.ai/xcore-200/.

[66] A. Neumann, M. J. Mytych, D. Wesemann, L. Wisniewski, and J. Jasperneite, "Approaches for In-vehicle Communication – An Analysis and Outlook," in *Proc. CN*, Landek, Poland, Jun. 2017.

[67] B. Rutemiller and J. Weber, "Communications and Connectivity – Ethernet Indepth, Wiring Types," in *Automation of Water Resource Recovery Facilities: WEF Manual of Practice No. 21*. Water Environment Federation, 2013.

[68] M. Baker, "Test Circuit Design Considerations – Printed Circuit Board Physics," in *Demystifying Mixed-Signal Test Methods*. Elsevier Science, 2003.

[69] "TTE Switch A664 Lab v2.0," TTTech Computertechnik AG, Aug. 2022. https://www.tttech.com/wp-content/uploads/TTTech_TTE-Switch_A664_Lab_v2.0.pdf.

[70] J. Randolph, "Designing Ethernet Cable Ports to Withstand Lightning Surges," *In Compliance Magazine*, Dec. 2016. https://incompliancemag.com/article/designing-ethernet-cable-ports-to-withstand-lightning-surges/.

[71] "Comidox 15KV Boost High Voltage Generator," Aug. 2022. https://www.amazon.com/dp/B07JG4K6S6?psc=1&ref=ppx_yo2_dt_b_product_details.

[72] "vacuum spark gap," Aug. 2022. https://www.ebay.com/sch/i.html?_from=R40&_trksid=p2380057.m570.l1313&_nkw=vacuum+spark+gap&_sacat=0.

[73] "TTE Switch OBC HiRel," TTTech Computertechnik AG, Aug. 2022. https://www.tttech.com/wp-content/uploads/TTTech_TTE-Switch-OBC-HiRel-Box-1.pdf.

[74] N. Flaherty, "Time Triggered Ethernet for Space Gateway," *EE News Europe*, May 2021. https://www.eenewseurope.com/news/time-triggered-ethernet-space.

[75] "Data Network Evaluation Criteria Report," Federal Aviation Administration, Tech. Rep. DOT/FAA/AR-09/27, Jun. 2009.

[76] A. Loveless, "Notional 1FT Voting Architecture with Time-Triggered Ethernet," AES A&S Tag-Up Meeting, Houston, TX, USA, Nov. 2016. https://ntrs.nasa.gov/citations/20170001652.

[77] ——, "Overview of TTE Applications and Development at NASA/JSC," CCSDS SOIS SUBNET Working Group Meeting, Rome, Italy, Oct. 2016. https://ntrs.nasa.gov/citations/20160012363.

[78] J. Vila-Carbo, J. Tur-Massanet, and E. Hernandez-Orallo, "Analysis of Switched Ethernet for Real-Time Transmission," in *Factory Automation*. IntechOpen, Mar. 2010.

[79] M. Sandić, I. Velikić, and A. Jakovljević, "Calculation of Number of Integration Cycles for Systems Synchronized using the AS6802 Standard," in *Proc. ZINC*, Novi Sad, Serbia, May 2017.

[80] M. Abuteir, Z. Owda, C. Zubia, F. Casado, M. Coppola, L. Kohutka, A. Geven, J. Migge, and M. Muñoz, "Distributed Real-time Architecture for Mixed Criticality Systems (DREAMS)," University of Siegen, Tech. Rep. Fault Injection Framework D5.2.3, Jul. 2016.

[81] J.-B. Chaudron, "TTEthernet: Theory, Concepts, and Applications," in *Proc. ETR*, Rennes, France, Aug. 2015. http://etr2015.irisa.fr/images/presentations/TTEthernet_ETR_2015_Rennes.pdf.

[82] C. Plettner, "COTS for Space," in *Proc. RADECS*, Geneva, Switzerland, Oct. 2017. https://indico.cern.ch/event/649606/sessions/246265/attachments/1522111/2403688/COTS_plettner_v2.pdf.

[83] "Core Flight System Scheduler (SCH) Application Design Review," National Aeronautics and Space Administration, Oct. 2019. https://github.com/nasa/SCH.

[84] D. McComas, "NASA/GSFC's Flight Software Core Flight System," in *Proc. Flight Software Workshop*, San Antonio, TX, USA, Nov. 2012. https://ntrs.nasa.gov/citations/20130013412.

[85] G. Cohen, C. Lee, M. Strickland, and T. Torkelson, "Final Report: Design of an Integrated Airframe/Propulsion Control System Architecture," Boeing Advanced Systems, NASA Contractor Report 182007, Mar. 1992.

[86] R. Butler, "A Survey of Provably Correct Fault-Tolerant Clock Synchronization Techniques," NASA Langley Research Center, Tech. Rep. NASA-TM-100553, Feb. 1988.

[87] L. E. Prokop, R. L. Hirsh, and C. Pagan, "Requirements-Based Execution Time Prediction of a Partitioned Real-Time System Using I/O and SLOC Estimates," *Innov. Syst. Softw. Eng.*, 2012.

[88] F. B. Schneider, "Implementing Fault-Tolerant Services Using the State Machine Approach: A Tutorial," *ACM Computing Surveys*, vol. 22, no. 4, pp. 299–319, Dec. 1990.

[89] C. Kouba, D. Buscher, and J. Busa, "The X-38 Spacecraft Fault-Tolerant Avionics System," in *Proc. MAPLD*, Washington, DC, USA, Aug. 2003.

[90] K. Hoyme and K. Driscoll, "SAFEbus," in *Proc. DASC*, Seattle, WA, USA, Oct. 1992.

[91] J.-L. Scharbarg and C. Fraboul, "Dimensioning of Civilian Avionics Networks," in *Industrial Communication Technology Handbook*. CRC Press, Aug. 2014.

[92] "Asteroid Redirect Robotic Mission," NASA Jet Propulsion Laboratory, Aug. 2022. https://www.jpl.nasa.gov/missions/asteroid-redirect-robotic-mission-arrm.

[93] "Trick 13 Simulation Environment," NASA Johnson Space Center, Aug. 2022. https://software.nasa.gov/software/MSC-25665-1.

[94] R. Kliger, "Integrated Transformer-Coupled Isolation," *IEEE Instrumentation and Measurement Magazine*, Mar. 2003.

[95] "TP302 Ethernet Surge Protector PoE+ Gigabit RJ45 Lightning Suppressor," Tupavco, Aug. 2022. https://www.tupavco.com/products/ethernet-surge-protector.

[96] W. Steiner and M. Paulitsch, "TTEthernet in Orion," in *Industrial Communication Technology Handbook*. CRC Press, Aug. 2014.

[97] A. Breitenreiter, J. López, P. Reviriego, M. Krstic, U. Gutierro, M. Sánchez-Renedo, and D. González, "A Radiation Tolerant 10/100

Ethernet Transceiver for Space Applications," in *Proc. IOLTS*, Rhodes, Greece, Jul. 2019.
[98] "SEPHY: Space Ethernet Physical Layer Transceiver," European Union's Horizon 2020 Programme, Sep. 2019. https://www.tttech.com/innovation/research-projects/eu-h2020/sephy/.
[99] "IEEE 802.1AE-2018: IEEE Standard for Local and Metropolitan Area Networks – Media Access Control (MAC) Security," Institute of Electrical and Electronics Engineers, Dec. 2018.
[100] K. Kainrath, K. Gebeshuber, M. Fruhmann, and M. Gruber, "Aviation Cyber Security Study," Munich, Germany, Tech. Rep., 2017.
[101] "Generation of a Testbed for the Validation of the TTEthernet Technology," in *Proc. ESA TEC-ED/SW Presentation Days*, Airbus Defence and Space, Bremen, Germany, Dec. 2019.
[102] D. Onwuchekwa and R. Obermaisser, "Fault Injection Framework for Assessing Fault Containment of TTEthernet Against Babbling Idiot Failures," in *Proc. IWQoS*, Alberta, Canada, Jun. 2018.
[103] D. Onwuchekwa, "Fault Injection Framework for Time-Triggered Systems," Ph.D. dissertation, University of Siegen, Siegen, Germany, Oct. 2020.
[104] F. Skopik, A. Treytl, A. Geven, B. Hirschler, T. Bleier, A. Eckel, C. El-Salloum, and A. Wasicek, "Towards Secure Time-Triggered Systems," in *Proc. SAFECOMP*, Magdeburg, Germany, Sep. 2012.
[105] P. Robyns, P. Quax, and W. Lamotte, "Injection Attacks on 802.11n MAC Frame Aggregation," in *Proc. WiSec*, New York, NY, USA, Jun. 2015.
[106] E. B. Joffe, "Assessment of the Robustness of Commercial Data Communication Interfaces to a Military EMI Environment," in *Proc. EMC*, Detroit, MI, USA, Aug. 2008.
[107] M. Kreitlow, F. Sabath, and H. Garbe, "Analysis of IEMI Effects on a Computer Network in a Realistic Environment," in *Proc. EMC*, Dresden, Germany, Aug. 2015.
[108] S. Jeschke, J. Loos, M. Kleinen, O. Kurt, J. Bärenfänger, C. Hangmann, and I. Wüllner, "Susceptibility of 100Base-T1 Communication Lines to Coupled Fast Switching High-Voltage Pulses," in *Proc. EMC Europe*, Rome, Italy, Sep. 2020.
[109] A. Tsukada, K. Okamoto, Y. Okugawa, J. Kato, and M. Nagata, "System-Level Response of Ethernet Linkage to Bulk Current Injection into Cables," in *Proc. EMC Europe*, Rome, Italy, Sep. 2020.
[110] C. Chen, S. Mohan, R. Pellizzoni, R. B. Bobba, and N. Kiyavash, "A Novel Side-Channel in Real-Time Schedulers," in *Proc. RTAS*, Montreal, Canada, Apr. 2019.
[111] S. Liu and W. Yi, "Task Parameters Analysis in Schedule-Based Timing Side-Channel Attack," *IEEE Access*, vol. 8, Sep. 2020.
[112] M.-K. Yoon, S. Mohan, C.-Y. Chen, and L. Sha, "TaskShuffler: A Schedule Randomization Protocol for Obfuscation against Timing Inference Attacks in Real-Time Systems," in *Proc. RTAS*, Vienna, Austria, Apr. 2016.
[113] C.-Y. Chen, M. Hasan, A. Ghassami, S. Mohan, and N. Kiyavash, "REORDER: Securing Dynamic-Priority Real-Time Systems Using Schedule Obfuscation," Jun. 2018. http://arxiv.org/abs/1806.01393.
[114] N. Vreman, R. Pates, K. Krüger, G. Fohler, and M. Maggio, "Minimizing Side-Channel Attack Vulnerability via Schedule Randomization," in *Proc. CDC*, Nice, France, Dec. 2019.
[115] M. Nasri, T. Chantem, G. Bloom, and R. M. Gerdes, "On the Pitfalls and Vulnerabilities of Schedule Randomization Against Schedule-Based Attacks," in *Proc. RTAS*, Montreal, Canada, Apr. 2019.
[116] K. Krüger, G. Fohler, and M. Volp, "Improving Security for Time-Triggered Real-Time Systems against Timing Inference Based Attacks by Schedule Obfuscation," in *Proc. ECRTS*, Dubrovnik, Croatia, Jun. 2017.
[117] K. Krüger, M. Völp, and G. Fohler, "Vulnerability Analysis and Mitigation of Directed Timing Inference Based Attacks on Time-Triggered Systems," in *Proc. ECRTS*, Barcelona, Spain, Jul. 2018.
[118] "USBKill," Aug. 2022. https://usbkill.com/.
[119] "Etherkiller – Coming Soon to a NIC Near You," Aug. 2022. https://etherkiller.org/.
[120] G. Fragkos, "A Weapon for the Mass Destruction of Computer Infrastructures," Sep. 2015. https://gfragkos.blogspot.com/2015/09/a-weapon-for-mass-destruction-of.html.
[121] C. Fidi and A. Loveless, "A Modular, Scalable Avionics Architecture for Future Exploration Missions," in *Proc. AIAA SPACE*, Orlando, FL, USA, Sep. 2017.

APPENDIX A
DESCRIPTION OF FIELDS IN A PCF

As described in §IV-A, a PCF's header contains the critical traffic marker, the virtual link ID, and the source MAC address. In addition, the header contains the following field.

- *EtherType / Length* — Indicates that the frame is a PCF. It must be set to 0x891d [2].

The PCF's payload contains the following fields. For each field, we state what value to set the field to in order for injected PCFs to be accepted by TTE devices.

- *Integration Cycle* — Tracks the current synchronization period. It must fall within a range defined in the schedule [2]. However, a value of 0x0 is always valid.
- *Membership New* — Identifies which sync masters contribute to the synchronized time base. When injecting integration PCFs, setting this to a high enough value tricks devices into detecting a clique [41]. In our tests, a value of 0x1 was always sufficient.
- *Sync Priority* — Must match the compression master priority in the network schedule. Most TTE networks use only one priority [121], so the value 0x1 is usually correct. Otherwise, the hardware limits the possible values to a small range (e.g., 0x1–0x3) [60], [61].
- *Sync Domain* — Identifies a specific set of synchronized devices in the network. Most networks have only one sync domain [22], so the value 0x0 is usually correct. Otherwise, the hardware limits the possible values to a small range (e.g., 0x0–0x7) [61].
- *Type* — Must be set to 0x8 (for a coldstart acknowledgement PCF) or 0x2 (for a normal integration PCF).
- *Transparent Clock* — Tracks delay in the switches. It must fall within a range determined by the hardware. In our tests, a value of 0x0 was always valid.