Discrete Event Dynamic Systems (2022) 32:573–610
https://doi.org/10.1007/s10626-022-00369-1

Synthesis of winning attacks on communication protocols using supervisory control theory: two case studies

Shoma Matsui[1] · **Stéphane Lafortune**[2]

Received: 9 January 2021 / Accepted: 22 September 2022 / Published online: 20 October 2022

Abstract
There is an increasing need to study the vulnerability of communication protocols in distributed systems to malicious attacks that attempt to violate properties such as safety or nonblockingness. In this paper, we propose a common methodology for formal synthesis of successful attacks against two well-known protocols, the Alternating Bit Protocol (ABP) and the Transmission Control Protocol (TCP), where the attacker can *always eventually* win, called For-all attacks. This extends previous work on the synthesis of There-exists attacks for TCP, where the attacker can *sometimes* win. We model the ABP and TCP protocols and system architecture by finite-state automata and employ the supervisory control theory of discrete event systems to pose and solve the synthesis of For-all attacks, where the attacker has partial observability and controllability of the system events. We consider several scenarios of person-in-the-middle attacks against ABP and TCP and present the results of attack synthesis using our methodology for each case.

Keywords Distributed protocols · Person-in-the-middle attacks · Supervisory control · Alternating bit protocol · transmission control protocol

1 Introduction

Keeping systems secure against attacks and preventing security incidents are challenging tasks due to the increasing complexity of modern system architectures, where a number of hardware and software components communicate over potentially heterogenous networks. To analyze systems which are too complex to be fully described

This article belongs to the Topical Collection: *Topical Collection on Cybersecurity*
Guest Editors: Rong Su and Carlos Basilio

✉ Shoma Matsui
s.matsui@queensu.ca

Stéphane Lafortune
stephane@umich.edu

[1] Department of Electrial and Computer Engineering, Queen's University, Kingston, Canada

[2] Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI, USA

monolithically, abstraction employing formal methods plays a key role and it has been studied in particular in the computer science literature (see, e.g., Baier and Katoen 2008; Kang et al. 2016). In networked systems, components with different architectures cooperate with each other using various pre-designed *protocols*. Due to the proliferation of communication using standardized protocols, vulnerabilities or misuses of protocols can result in serious security issues. As a concrete example, Bagheri et al. (2015) introduces a formal model and analysis of a protocol used in Android OS, one of the most popular operating systems for smart phones. In order for components to cooperate with each other without damaging systems and without data corruption, robustness of protocols against communication failures is essential in modern system architectures. To ensure such robustness of protocols, relevant properties, such as *safety* and *liveness*, should be satisfied even if packets are dropped for instance. However, the situation is different in the context of malicious attacks, where an attacker that has infiltrated part of the system (e.g., the network) may be able to induce a violation of the safety or liveness properties, thereby causing the protocol to enter an abnormal state.

The development of resilient protocols that satisfy requirements and are applicable to various systems requires formal methods for modelling, verification, and synthesis. These problems have a long history in computer science as well as in control engineering. The readers are referred to Baier and Katoen (2008) and Holzmann and Lieberman (1991) for a comprehensive treatment of modelling and verification by employing formal methods, such as temporal logic. To prevent systems from being damaged by attacks that exploit vulnerabilities of protocols, the recent work (Alur and Tripakis 2017) introduces the process of completing an incompletely specified protocol so that the completed protocol satisfies required properties and does not suffer from deadlock. Alur and Tripakis (2017) explains its methodology of protocol completion using the Alternating Bit Protocol (ABP).

In control engineering, the formalism of of discrete event systems (DES) (Cassandras and Lafortune 2021) and its supervisory control theory (SCT) (Wonham and Cai 2019) are useful tools to treat the problem of protocol verification as a supervisory control problem (Rudie and Wonham 1992), so as to determine whether a given protocol satisfies the required properties. Not only can SCT be used to analyze existing protocols, it can also be used to synthesize a desired protocol based on given requirements. For instance, Kumar et al. (1997) introduces a systematic approach to design a protocol converter for mismatched protocols so that the specifications of the entire system and protocols themselves are satisfied simultaneously. On the other hand, Rudie and Wonham (1990) considers protocols comprising local communicating processes, and formalizes protocol synthesis as the problem of controlling the local processes so that the global specification of the entire system is satisfied, employing the decentralized version of SCT. For a comprehensive survey of protocol synthesis, focusing on the formalization of the design of protocols, the readers are referred to Saleh (1996).

More generally, detection, mitigation, and prevention of attacks on supervisory control systems within the framework of SCT has been considered in several works, such as Carvalho et al. (2018), Wakaiki et al. (2019), Su (2018), and Meira-Góes et al. (2019). Carvalho et al. (2018) presents a methodology of designing intrusion detectors to mitigate online four types of attacks; actuator enablement/disablement and sensor erasure/insertion. Focusing on sensor deception attacks under which the attacker arbitrarily edits sensor readings by intervening between the target system and its control module to trick the supervisor to issue improper control commands, Wakaiki et al. (2019) and Su (2018) study how to synthesize robust supervisors against sensor deception attacks, while (Su 2018) also introduces the synthesis problem of attack strategies from the attacker's point of view.

Subsequently, a different technique from Su (2018) to compute a solution of the synthesis problem of robust supervisors was proposed in Meira-Góes et al. (2019).

As protection against attacks is one of the main subjects of systems security, methodologies for designing attack strategies against systems have been reported in the literature (Meira-Góes et al. 2020; Lin et al. 2019; Von Hippel et al. 2020a). Meira-Góes et al. (2020) presents how to synthesize an attacker in the context of stealthy deception attacks, modelled in the framework of SCT, which cannot be detected by the supervisor and cause damage to the system, as a counter weapon against intrusion detection modules as in Carvalho et al. (2018). While (Meira-Góes et al. 2020) considers sensor deception attacks as the attacker's weapon, Lin et al. (2019) introduces the synthesis of actuator attacks under which the attacker has the ability to hijack the control commands generated by the supervisor, to damage the system.

Formal synthesis of successful attacks against protocols is the problem considered in this paper, in the context of two case studies. The work in Von Hippel et al. (2020a) (and its conference version (Von Hippel et al. 2020b)) is of special relevance, as it introduces a methodology of attacker synthesis against systems whose components are modelled as finite-state automata (FSA). It presents how so-called "There-exists" attackers can be found (if they exist) using a formal methodology that has been implemented in the software tool Korg (Von Hippel 2020). In the terminology of Von Hippel et al. (2020a), "There-exists" refers to attackers that cannot always lead protocols to a violation of required properties, but *sometimes* succeed ("there exists" a winning run for the attacker). (Von Hippel et al. 2020a) formulates the properties that protocols must protect against as *threat models*, and it illustrates its methodology with the Transmission Control Protocol (TCP), specifically connnection establishment using three-way handshake, as standardized in Postel (1981). The formal model in Von Hippel et al. (2020a) was inspired by that in Jero et al. (2015) where automated attack discovery for TCP is performed using a state-machine-informed search.

In this paper, we revisit the respective ABP and TCP models of Alur and Tripakis (2017) and Von Hippel et al. (2020a) in the standard framework of DES modelled as FSA. In contrast to the feedback-loop control system architecture in the previously-mentioned works on sensor/actuator deception attacks in SCT, we consider a network system architecture in which two peers are sending and receiving packets through channels and/or networks, as explained in Section 3. We consider "person-in-the-middle" (PITM) attacks as in Von Hippel et al. (2020a) and Jero et al. (2015), in a manner reminiscent of deception attacks. Inspired by and complementary to the approach in Von Hippel et al. (2020a), we exploit results in SCT and develop a methodology to synthesize "For-all" attackers, that is, attackers that *can always eventually* cause a violation of required properties of the system, extending the previous work by Von Hippel et al. (2020a) on There-exists attackers. Section 4 will present the details of our methodology, and will state our main results as Theorem 1. We then apply this methodology to both ABP and TCP, using essentially the same models as in Alur and Tripakis (2017) and Von Hippel et al. (2020a). Thus, our results extend those in Von Hippel et al. (2020a) by formally considering the synthesis of "For-all" attackers on TCP, since For-all attacks are more powerful than There-exists attacks. In both of our case studies, we approach attack synthesis as a supervisory control problem under partial observation *from the attacker's viewpoint*, which is then solved using existing techniques (Cassandras and Lafortune 2021; Wonham and Cai 2019). As specifically discussed in Section 4.3, under the assumptions of our PITM attack model, a "For-all" attacker for a given threat model is obtained by building the realization of the (partial-observation) *nonblocking* supervisor that results in the supremal controllable and normal sublanguage (supCN) of the *threat model* language with respect to the system language and to the attacker's controllable and observable event sets. The supCN operation was first introduced in Cho and Marcus (1989), and several formulas to compute supCN were derived in Brandt et al. (1990). For each of the two protocols ABP and TCP, respectively in Sections 5 and 6, we analyze several setups capturing different PITM attacker capabilities.

The detailed case studies presented in this paper, based upon established models of ABP and TCP (three-way handshake part), show the various steps on how to build, in a systematic manner, successful PITM attacks (if they exist) on these two well-know protocols. We believe they can also serve as inspiration for similar case studies on other protocols.

The remainder of this paper is organized as follows. Section 2 provides a brief review of the DES framework and its Supervisory Control Theory employed in this paper. In Section 3, we introduce the context on modelling of communication protocols and give an overview of the PITM attack model under consideration, which is based on specifying a safety or nonblockingness property that the attacker is intent on violating in the context of SCT. Section 4 formulates the SCT-based synthesis problem of a For-all attacker (if it exists) and presents the features of the common methodology that is used in the subsequent sections on ABP and TCP, respectively. ABP is considered first in Section 5, and then TCP is considered in Section 6. Both sections contain sufficient details so that these case studies can be replicated. Finally, we conclude the paper in Section 7.

2 Preliminaries

In this section, we introduce several notions of the DES framework in Cassandras and Lafortune (2021), leveraged to build our models in this paper. The central definitions we need here are *automata*, *nonblockingness* of automata, *parallel composition*, *supervisory control theory* and *nonblocking supervisor*.

In DES, what happens in the system is explained by sequences of predefined *events* which discretely occur. Specifically, the system's behaviour is represented as a set of sequences of events, called a *language*, and each sequence is called a *string*. Namely, a language is a set of strings. Note that strings could be arbitrary long and languages could be infinite sets.

One of the intuitive methods to represent (regular) languages is *finite state automata* (FSA), or simply *automata*, represented as a quintuple

$$G = (X, E, f, x_0, X_m) \tag{1}$$

where X is the finite set of states, E is the finite set of events, $f : X \times E \rightarrow X$ is the (partial) transition function, x_0 is the initial state and $X_m \subseteq X$ is the set of marked states. The function f denotes the system's behaviour as state transitions defined in the automaton G, e.g., $f(x,e) = x^{'}$ represents a transition labelled by event $e \in E$ from state $x \in X$ to state $x' \in X$. From Eq. 1, the connection between languages and automata is formally defined as the *generated language* $\mathcal{L}(G) := \{s \in E^* \mid f(x_0, s) \text{ is defined}\}$.

From the perspective of system control, it makes sense to consider that several behaviours of the system are acceptable or desired. We call the strings denoting acceptable behaviours *marked strings*, and the language consisting of marked strings is called a *marked language*. To represent the marked language associated with G, the marked states in X_m come to play that role. Mathematically, the language marked by G is defined by $\mathcal{L}_m(G) := \{s \in \mathcal{L}(G) \,|\, f(x_0, s) \in X_m\}$. However, depending on the structure of G, it may not be guaranteed that the system G can always eventually reach its marked states. In particular, the existence of deadlock and livelock in G can prevent the marked states from being reached. Such a property in DES is called *nonblockingness*. Specifically, G is said to be blocking if $\overline{\mathcal{L}_m(G)} \subset \mathcal{L}(G)$ and nonblocking if $\overline{\mathcal{L}_m(G)} = \mathcal{L}(G)$. In other words, if G is blocking, then there exists deadlock or livelock in G, that is, there exists a state from which the marked states cannot be reached, and vice versa.

In many cases, the systems we analyze consist of several subcomponents, or one may want to examine at once the entire behaviour of multiple system models. The DES framework has an operation of automata called *parallel composition* to build models of entire systems from subsystem models. For example, the parallel composition G' of system G_1 and system G_2 is denoted by $G' = G_1 \parallel G_2$. Roughly speaking, a common event in G_1 and G_2 can only occur in G' if both G_1 and G_2 execute it simultaneously. The private (unshared) events, on the other hand, can be executed in G' whenever feasible in either G_1 or G_2. For the detailed definition and properties of parallel composition, readers are referred to (Cassandras and Lafortune 2021, pp. 81–87).

Considering that the given systems do not always follow their specifications, supervisory control is a concept to control the systems represented as DES, and its mathematical framework is called *supervisory control theory* (SCT), which is to synthesize a controller attached to the system so that the given specifications are satisfied. In the framework of SCT in DES, a system to be controlled is called a *plant*, and a plant is controlled by a *supervisor* that enables or disables particular (controllable) events so that the plant satisfies a given specification for safety or nonblockingness for instance. The control actions of the supervisor are determined by observation of the strings generated by the plant; thus the plant and supervisor from a feedback loop as depicted in Fig. 1.

Technically speaking, a supervisor S is defined as a function

$$S : \mathcal{L}(G) \to 2^E \tag{2}$$

which takes a string generated by G and returns a set of events permitted to occur in G. In other words, $S(s)$ is a control action for a string $s \in \mathcal{L}(G)$. Note that supervisor S is prohibited from disabling a feasible uncontrollable event at any state. Namely, letting $E_{uc} \subseteq E$ be a set of uncontrollable events in G, for each $s \in \mathcal{L}(G)$, it always holds that $E_{uc} \cap \{e \in E \,|\, f(f(x_0, s), e) \text{ is defined}\} \subseteq S(s)$.

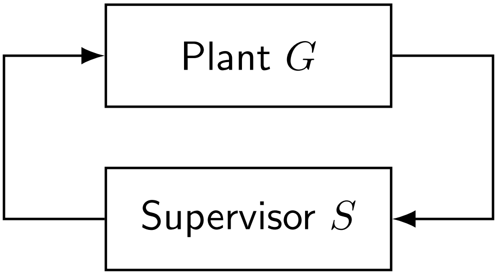


Fig. 1 The feedback loop of supervisory control

In the framework of SCT, it is also considered that the supervisor has a limited observability of events generated by the plant. This limitation is represented by partitioning the set of events E into two disjoint subsets: the sets of observable events E_o and of unobservable events E_{uo}, namely $E = E_o \cup E_{uo}$. To implement this property, the supervisor in Eq. 2 is extended to the *partial-observation supervisorS_P* defined by

$$S_P : P[\mathcal{L}(G)] \to 2^E \tag{3}$$

where P is the natural projection from domain E^* to codomain E_o^*, removing unobservable events from a string generated by G. Note that in this scheme, the control action by S_P is supposed to always take effect before any unobservable event occurs.

Given G and S_P, the closed-loop behaviour of G controlled by S_P is denoted by a DES S_P/G, formalized in the following definition.

Definition 1 (Languages generated and marked by S_P/G) (cf. (Cassandras and Lafortune 2021, p. 151)) The generated language $\mathcal{L}(S_P/G)$ is recursively defined as

1. $\varepsilon \in \mathcal{L}(S_P/G)$
2. $[s \in \mathcal{L}(S_P/G) \wedge s\sigma \in \mathcal{L}(G) \wedge \sigma \in S_P[P(s)]] \iff [s\sigma \in \mathcal{L}(S_P/G)]$

and the marked language $\mathcal{L}_m(S_P/G)$ is defined as

$$\mathcal{L}_m(S_P/G) := \mathcal{L}(S_P/G) \cap \mathcal{L}_m(G). \tag{4}$$

We can also examine the blockingness of S_P/G as a meaningful characteristic of the controlled system. Similarly to the blockingness of G, the DES S_P/G is said to be *blocking* if $\mathcal{L}(S_P/G) \neq \overline{\mathcal{L}_m(S_P/G)}$ and blocking if $\mathcal{L}(S_P/G) = \overline{\mathcal{L}_m(S_P/G)}$. Since these properties depend on the synthesis result of S_P, S_P is said to be blocking if S_P/G is blocking and to be nonblocking if S_P/G is nonblocking.

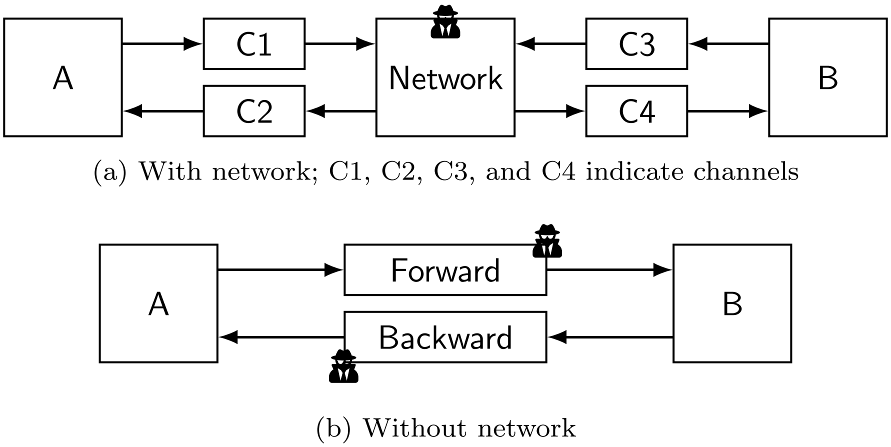
The specification that the plant should obey is given as a specification language $L^{spec} \subseteq \mathcal{L}(G)$, or its automaton representation H such that $\mathcal{L}_m(H) = L^{spec}$. It is an important point that L^{spec} may not be $\mathcal{L}_m(G)$-closed, namely $L^{spec} \neq \overline{L^{spec}} \cap \mathcal{L}_m(G)$, and we may want the supervisor S_P to "mark" strings in $\mathcal{L}(S_P/G)$ based on L^{spec}, rather than $\mathcal{L}_m(G)$. Therefore, the SCT framework provides an alternative version of S_P, called a *marking supervisor*, defined as

$$\mathcal{L}_m(S_P/G) := \mathcal{L}(S_P/G) \cap L^{spec}. \tag{5}$$

For the technical details of marking supervisors, the readers are referred to Section 3.9 in Cassandras and Lafortune (2021). In the rest of this paper, nonblockingness of S_P will be defined by either Eqs. 4 or 5, depending on the properties of the considered specification L^{spec} (namely, L^{spec} being $\mathcal{L}_m(G)$-closed or not).

3 System and attack models

Before proceeding to the specific ABP and TCP protocols, we highlight in this section and in the next one the common elements of our two case studies.



(a) With network; C1, C2, C3, and C4 indicate channels

(b) Without network

Fig. 2 Communication overview

3.1 System architecture

When modelling communication protocols such as ABP and TCP, we consider a "system" that consists of peers communicating with each other, channels, and networks. For clarity of presentation, we suppose the system comprises two peers, two or four channels, and one network. If peers form a small network using channels, e.g., a local area network (LAN), then networks can be omitted and we consider two channels connecting each peer, namely, the forward and backward channels.

Figure 2 illustrates an overview of the flow of packets between two peers through channels. Peers A and B exchange packets using communication protocols through the channels and network. In this paper, we consider "person-in-the-middle" (PITM) as the attack model on the system. In this model, the attacker *infiltrates the network or channels*, and afterwards sends fake packets and/or discards genuine ones, exploiting vulnerabilities of the protocol (as captured by the peer automata), to damage the system. The system may contain other processes for exogenous events, e.g., timers, called environment processes, which are not depicted in Fig. 2. Channels work as interfaces between the peers and the network, relaying packets to their destinations. Each component of the system is modelled by a finite-state automaton, and denoted as follows:

G_{PA}: Peer A; G_{PB}: Peer B; G_C: Channel; G_N: Network; and G_e: Environment processes.

Each channel is represented by one finite-state automaton, thus G_C is the parallel composition of the channel automata. For example, if the system architecture is that in Fig. 2a, then $G_C = G_{C1} \parallel G_{C2} \parallel G_{C3} \parallel G_{C4}$ where G_{Ci} ($i = [1,4]$) are the respective automata modelling each channel. If the system architecture is that in Fig. 2b, then $G_C = G_{FC} \parallel G_{BC}$ where G_{FC} and G_{BC} are the forward and backward channels, respectively, and G_N is empty since there is no network in such an architecture. In the case where there exist more than two environment processes in the system, G_e is also constructed as the parallel composition of all environment processes.

To capture PITM attacks on the above system, we create new versions of the channels and network automata when they are infiltrated by the attacker and denote them by $G_{C,a}$ and $G_{N,a}$, respectively. We consider that the attacker cannot directly tamper the internal codes of peers in our model of PITM attacks, meaning that the attacker cannot disable nor enable the private events of the peers. Instead, in the infiltrated channels or network, the attacker intercepts packets and can delete them, and can also insert new packets to impersonate the sender or receiver, as similarly considered in Jero et al. (2015). Thus, we construct $G_{C,a}$ and $G_{N,a}$ by the *addition of new transitions and events* that represent the feasible actions of the attacker, as the addition can capture insertion and replacement of packets, and packet deletion by the attacker can be captured by disabling transitions which indicate packet transfer. Concrete examples of $G_{C,a}$ and $G_{N,a}$ will be presented in the case studies in Sections 5 and 6.

Let us define a nominal system model (i.e., without attacker) by

$$G_{nom} := (X_{nom}, E_{nom}, f_{nom}, x_{nom,0}, X_{nom,m}). \tag{6}$$

G_{nom} is the parallel composition of the peers, channels, network, and environment processes, namely

$$G_{nom} = G_{PA} \parallel G_{PB} \parallel G_C \parallel G_N \parallel G_e \tag{7}$$

As we consider PITM attacks on the system, we enhance G_{nom} to the new model of the system under attack

$$G_a := (X_a, E_a, f_a, x_{a,0}, X_{a,m}) \tag{8}$$

where possible new transitions and events representing the actions of the attacker come from the enhanced $G_{C,a}$ and $G_{N,a}$ automata described above. The other compoents of G_{nom}, namely the peer automata G_{PA} and G_{PB}, as well as G_e, remain unchanged. In our case studies, the plant G_a is acted upon by the attacker; hence, the plant consists of the entire system under attack:

$$G_a = G_{PA} \parallel G_{PB} \parallel G_{C,a} \parallel G_{N,a} \parallel G_e$$

The sending and receiving of packets are represented by events. As we consider PITM attacks, it is reasonable to assume that an attacker infiltrating the network or channels can only monitor incoming and outgoing packets at the infiltrated component. In other words, the attacker cannot observe the private events of the peers. Therefore, we consider that the events in our system model are partitioned into *observable events* and *unobservable events*, based on the system structure and the capability of the attacker. It is also natural to assume that the attacker cannot prevent the peers from sending packets to the network or channels, although the attacker can discard their packets. That is, the attacker cannot control the receiving of packets by the network or channels.

Example 1 Let us consider PITM attacks on the Alternating Bit Protocol (ABP). ABP is a protocol which defines the communication mechanism between two peers depicted in Fig. 2b. Each peer sends and receives packets from its counterpart through the forward and backward channels using first-in-first-out (FIFO) semantics. Inspired by (Alur and Tripakis 2017), we consider G_{nom} as the parallel composition of the following 7 automata.

- $G_S = (X_S, E_S, f_S, x_{S,0}, X_{S,m})$: ABP sender
- $G_R = (X_R, E_R, f_R, x_{R,0}, X_{R,m})$: ABP receiver
- $G_{FC} = (X_{FC}, E_{FC}, f_{FC}, x_{FC,0}, X_{FC,m})$: Forward channel
- $G_{BC} = (X_{BC}, E_{BC}, f_{BC}, x_{BC,0}, X_{BC,m})$: Backward channel
- $G_{SC} = (X_{SC}, E_{SC}, f_{SC}, x_{SC,0}, X_{SC,m})$: Sending client
- $G_{RC} = (X_{RC}, E_{RC}, f_{RC}, x_{RC,0}, X_{RC,m})$: Receiving client
- $G_T = (X_T, E_T, f_T, x_{T,0}, X_{T,m})$: Timer

Therefore, we have

$$G_{nom} = G_S \parallel G_R \parallel G_{FC} \parallel G_{BC} \parallel G_{SC} \parallel G_{RC} \parallel G_T \tag{9}$$

We also consider that Peer A first sends packets to Peer B, and afterwards Peer B sends an acknowledgement to Peer A. Since Peer A plays a role of the sender side and Peer B is at the receiver side, $G_{PA} = G_S$, $G_{PB} = G_R$, $G_C = G_{FC} \parallel G_{BC}$, and $G_e = G_{SC} \parallel G_{RC} \parallel G_T$, thus Eq. 9 reduces to Eq. 7. Note that G_N in Eq. 7 will be empty in this case.

The various event sets are defined as follows, where synchronization in $\parallel$ will be achieved by common events:

$$E_S = \{send, done, timeout, p_0, p_1, a'_0, a'_1\} \tag{10}$$

$$E_R = \{deliver, p'_0, p'_1, a_0, a_1\} \tag{11}$$

$$E_{FC} = \{p_0, p_1, p'_0, p'_1\} \tag{12}$$

$$E_{BC} = \{a_0, a_1, a'_0, a'_1\} \tag{13}$$

$$E_{SC} = \{send, done\} \tag{14}$$

$$E_{RC} = \{deliver\} \tag{15}$$

$$E_T = \{timeout\} \tag{16}$$

Hence

$$E_{nom} = E_S \cup E_R \cup E_{FC} \cup E_{BC} \cup E_{SC} \cup E_{RC} \cup E_T \tag{17}$$

$$= \{send, done, timeout, deliver, p_0, p_1, p'_0, p'_1, a_0, a_1, a'_0, a'_1\}. \tag{18}$$

The events with prefix “p” indicate that a packet with indicator bit “0” or “1” has been sent from the ABP sender to the ABP receiver (i.e., from Peer A to Peer B), and prefix “a” indicates an acknowledgement sent from the ABP receiver to the ABP sender, corresponding to which “0” or “1” has been received by the ABP receiver. The prime symbol is attached to the events of packets and acknowledgement to distinguish those before going through the channel from the corresponding ones after the channels, as is done in Alur and Tripakis (2017).

Figure 4 shows the models of the ABP components. G_S and G_R are example solutions of the distributed protocol completion problem in Alur and Tripakis (2017). Note that we have removed from the models in Fig. 4 “dead” transitions which are never executed by the system when the attacker is not present. The terminology “dead” is from (Alur and Tripakis 2017). In addition, we mark all the states of the ABP components, for reasons that will become clear later. Namely,

$$X_{S,m} = X_S,\ X_{R,m} = X_R,\ X_{FC,m} = X_{FC},\ X_{BC,m} = X_{BC},\ X_{SC,m} = X_{SC},\ X_{RC,m} = X_{RC},\ X_{T,m} = X_T$$

In (Alur and Tripakis 2017), the forward and backward channels are modelled as nondeterministic finite-state automata as shown in Fig. 4b and d. That nondeterminism is introduced to model nonadversarial errors in communication channels, such as packet drop and duplication (see Section 4.2 in Alur and Tripakis (2017)). To construct the system model in Eq. 9, we need deterministic finite-state automata as factors of the parallel composition. Thus, we construct G_{FC} and G_{BC} as observer automata of G_{FC}^{nd} and G_{BC}^{nd}, depicted in Fig. 5:

$$G_{FC} = Obs(G_{FC}^{nd}) \tag{19}$$

$$G_{BC} = Obs(G_{BC}^{nd}) \tag{20}$$

where "observers" are as defined in Cassandras and Lafortune (2021) and capture the standard conversion of a nondeterministic automaton to a deterministic one (often referred to as subset construction). Observe that G_{FC} and G_{BC} generate exactly the same languages as G_{FC}^{nd} and G_{BC}^{nd}, respectively.

Let us consider one example case of PITM attacks where a powerful attacker infiltrates the forward channel. To construct the plant under attack G_a capturing the attacker's actions, we enhance G_{FC}^{nd} to $G_{FC,a}^{nd}$ as depicted in Fig. 8a by adding the new transitions shown as the red arrows. This enhanced channel model represents the attacker's capability that can send packets to the recipient with whichever bit 0 or 1, regardless of the incoming packets from the sender. Letting $G_{FC,a} = Obs(G_{FC,a}^{nd})$ in the same way as Eq. 19, G_a is hereby given by

$$G_a = G_S \parallel G_R \parallel G_{FC,a} \parallel G_{BC} \parallel G_e \tag{21}$$

Section 5 describes in detail the procedure to model the PITM attack against ABP.

In our case studies, G_a is the plant and the attacker plays a role of the supervisor; in this context, the specification represents what damage the attacker wants to cause to the system. In other words, the specification should capture *violations* of a desired property of the communication protocol, such as absence of deadlock or proper delivery of packets. Therefore, using SCT to synthesize a supervisor that enforces the violation of a desired property of the communication protocol under consideration means that we have actually synthesized an attack strategy that indeed causes a violation of that property.

3.2 For-all attack

One of the contributions of this paper as compared to previous work is that we consider that the attacker wants to attack the system in a "For-all" manner, to be interpreted in the following sense: *the attacker can always eventually cause a violation of the given property*. Such specifications are naturally captured in SCT using the notion of marked states and nonblockingness. When the marked states capture the violation of the given property, then a *nonblocking supervisory* in SCT will exactly achieve the goal of a For-all attacker, since it will always be possible to eventually reach a marked state. Specifically, consider an attacker's marked (i.e., non-prefix-closed) specification language $L_a^{spec} \subset \mathcal{L}(G_a)$ which consists of strings that are illegal but feasible in the system under attack. Let S_a be a supervisor (aka attacker) for G_a that achieves as much of L_a^{spec} as possible in the controlled system S_a/G_a. We denote this marked language by K, namely, $K \subseteq L_a^{spec}$ and the attacker wants K to be as *large* as possible. In order to achieve a For-all attack, the attacker wants S_a to be *nonblocking*, namely, $\mathcal{L}_m(S_a/G_a) = K$ and $\mathcal{L}(S_a/G_a) = \overline{K}$. Thus, nonblockingness of the system under attack implies that the attacker can always eventually win; thus, we have indeed obtained a For-all attack strategy. This is how For-all attacks are defined in this paper.

The above definition of For-all attacks is formalized in Definition 2.

Definition 2 (For-all Attack-Supervisor) Given $L_a^{spec} \subset \mathcal{L}(G)$, let $K \subseteq L_a^{spec}$ be a nonempty sublanguage. S_a is said to be a For-all attack-supervisor with respect to G_a and K if

1. $\mathcal{L}_m(S_a/G_a) = K$; and
2. $\mathcal{L}(S_a/G_a) = \overline{K}$.

3.3 There-exists attack

If there exists a supervisor S_a not satisfying the condition in Definition 2 but $\mathcal{L}(S_a/G_a) \cap K \neq \emptyset$, then we say that such an S_a achieves a There-exists attack, because in that case the controlled system (under the actions of the attacker) S_a/G_a will contain deadlocks and/or livelocks (i.e., the system under attack is blocking in the terminology of SCT); this prohibits the attacker from always being able to eventually win. Still, the nonemptyness of $\mathcal{L}_m(S_a/G_a)$ means that the attacker can sometimes win. This is how There-exists attacks are defined in this paper.

The above definition of There-exists attacks is formalized in Definition 3.

Definition 3 (There-exists Attack-Supervisor) Given $L_a^{spec} \subset \mathcal{L}(G)$, let $K \subseteq L_a^{spec}$ be a nonempty sublanguage. S_a is said to be a There-exists attack-supervisor with respect to G_a and K if

1. $\mathcal{L}(S_a/G_a) \cap K \neq \emptyset$; and
2. S_a is not a For-all attack-supervisor.

Now that we have shown how to build the plant model G_a, we address in the next section the construction of an automaton representation for the (non-prefix-closed) language L_a^{spec}, which will be the "specification automaton" for the attacker that is needed in the context of SCT algorithmic procedures.

Remark 1 In the prior work (Von Hippel et al. 2020a), mostly analogous definitions of There-exists and For-all attackers are given, but in the framework of reactive synthesis with infinite strings and temporal logic (LTL) specifications (see Definition 6 in Von Hippel et al. (2020a)). The technical difference comes from requiring "can always eventually win" instead of requiring "will always eventually win" (as is typically done in LTL and is done in Von Hippel et al. (2020a)). The latter is expressible in LTL, but not the former. The reactive synthesis setting is formally compared to that of SCT in Ehlers et al. (2017), where it is shown that nonblockingness in SCT is not expressible in LTL but instead corresponds to "AGEF(marked)" in CTL. In this paper, since we use SCT, we match the notion of "AGEF(marked)", i.e., "can always eventually win". Moreover, since we are working in the context of SCT, we will use the term "nonblockingness" for the class of "liveness" properties that will be considered in this paper.

4 Procedure for synthesis of For-all attacks on communication protocols

In this section, we discuss the modelling procedure to construct a specification automaton for the attacker based on the considered properties (instances of safety or nonblockingness) of the communication protocol that are to be violated by actions of the attacker. Then, we formulate the problem of finding For-all feasible attacks on the system as a supervisor synthesis problem in SCT which has a known solution. The SCT-based methodology presented in this section will be applied to ABP and TCP in the next two sections.

4.1 Safety properties

As in Alur and Tripakis (2017), consider a safety property whose violation is modelled by an automaton, termed a *safety monitor* G_{sm}. G_{sm} captures the violation of the given safety property in terms of *illegal* states in its structure. Since the specification for attackers represents a violation of the property, the illegal states are represented by *marked* states in G_{sm}. In other words, G_{sm} captures the *violation* of the safety property of interest when it reaches its *marked states*. (Note that in our problem context, we do require marked states to capture violation of safety properties.)

G_{sm} can be derived from automata composing G_{nom}, namely, the peers, channels, or network, by modifying state marking for instance. One can also independently design G_{sm} as a new automaton that we call a *dedicated automaton* in this paper. Both instances will occur in our case studies. For example, in Section 5, the safety monitors G_{sm} for ABP are given as dedicated automata in Fig. 6. Let G_{other} be the parallel composition of the automata in G_{nom} which are not used to construct G_{sm}. For example, from Eq. 7, if G_{sm} is built by modifying $G_{PA} \parallel G_{PB}$, then $G_{other} = G_C \parallel G_N \parallel G_e$. In Section 6, we will construct G_{sm} for the TCP case study using TCP peer models G_{PA} and G_{PB} in Fig. 16 later on.

The specification automaton will in our case studies be the parallel composition of G_{other} and G_{sm}, as is commonly done in SCT. Letting H_{nom} be the specification automaton with respect to G_{nom} (system without attacker), we have that $H_{nom} = G_{other} \parallel G_{sm}$. Note that since we want marking in H_{nom} to be determined by marking in G_{sm}, all the states of G_{other} are to be marked. In the absence of attackers, the communication protocol should ensure the safety property under consideration, which means that its violation should never occur. This can be verified by confirming that H_{nom} has no reachable marked states, i.e., H_{nom} captures no violations of the given safety property with respect to G_{nom}.

To represent the specification automaton with respect to the system under attack, namely G_a, we construct $G_{other,a}$ based on G_a in the same manner as G_{other}. For instance, if G_{sm} is a dedicated automaton and the attacker infiltrates the network, then $G_{other,a} = G_{PA} \parallel G_{PB} \parallel G_C \parallel G_{N,a} \parallel G_e$. Let $H_a = G_{other,a} \parallel G_{sm}$ be the specification automaton under attack. Similarly to marking in G_{other}, we want G_{sm} to determine marking in H_a, thus all the states of $G_{other,a}$ are to be marked. If there exist no marked states in H_a, then the attacker is not powerful enough to cause a violation of the safety property. Even if H_a has marked states, there may not exist For-all attacks (but possibly only There-exists attacks), depending on whether a nonblocking supervisor can be synthesized with respect to plant G_a and specification automaton H_a; this will be addressed in the solution of the SCT problem discussed below.

In summary, the procedure to build H_a for a given safety property is presented in Algorithm 1.

```
Input: $G_{nom}$, $G_a$, $G_{sm}$
Output: $H_a$
 1: if $G_{sm}$ is a dedicated automaton then
 2:     $G_{other} = G_{nom}$
 3:     $G_{other,a} = G_a$
 4: else
 5:     $\Phi = \{G_{PA}, G_{PB}, G_C, G_N, G_e\}$
 6:     $\Phi_a = \{G_{PA}, G_{PB}, G_{C,a}, G_{N,a}, G_e\}$
 7:     $G_{other} = ||\{G \in \Phi \mid G$ is not used to construct $G_{sm}\}$
 8:     $G_{other,a} = ||\{G \in \Phi_a \mid G$ is not used to construct $G_{sm}\}$
 9: end if
10: $H_{nom} = (Y_{nom}, E_{nom}, g_{nom}, y_{nom,0}, Y_{nom,m}) = G_{other} \parallel G_{sm}$
11: if $Y_{nom,m} \neq \varnothing$ then
12:     Terminate with empty solution   ▷ The given model is incorrect as the safety property is violated even if no attacker is present.
13: end if
14: Mark all the states in $G_{other,a}$
15: $H_a = (Y_a, E_a, g_a, y_{a,0}, Y_{a,m}) = Trim(G_{other,a} \parallel G_{sm})$ ▷ $H_a$ should be trim because we want the attacker to always be able to eventually win, i.e., there should not be any deadlocks/livelocks in the controlled $G_a$.
16: if $Y_{a,m} = \varnothing$ then
17:     Terminate with empty solution   ▷ The attacker's actions can never cause a violation of the given safety property.
18: end if
19: return $H_a$
```

Algorithm 1 Attack Specification against Safety (SafeSpec).

Proposition 1 Suppose that $Y_{nom,m} = \varnothing$ in Algorithm 1, that is, the given system model is correct in terms of the safety properties. If $Y_{a,m}$ on line 16 is empty, then no For-all attack exists and no There-exists attack exists.

Proof By construction, G_{sm} captures a violation of the given safety property by reaching its marked states. Let $X_{other,a}$ and X_{sm} be the sets of states of $G_{other,a}$ and G_{sm}, respectively. Note that $Y_a \subseteq X_{other,a} \times X_{sm}$ from line 15 of Algorithm 1. Since all the states in $X_{other,a}$ are marked, it holds that $Y_{a,m} = \varnothing$ iff for every $(x_{other,a}, x_{sm}) \in Y_a$, x_{sm} is not marked. This means that the safety monitor G_{sm} never captures the violation iff H_a has no marked states. In other words, the attacker can never cause a violation of the given safety property. Therefore, if $Y_{a,m} = \varnothing$, then no For-all attack exists and no There-exists attack exists.

We build several instances of H_a for ABP in Section 5.4 and for TCP in Section 6.5. The safety monitors for ABP are given as dedicated automata in Fig. 6, as will be explained in Section 5.1, while those for TCP are derived from G_a based on the given safety property, as will be explained in Section 6.2.

4.2 Nonblockingness properties

We examine a "limited" liveness property, called *nonblockingness*, as expressible in SCT for *-languages, namely, languages of finite strings. Nonblockingness is an adequate tool in many applications, such as in software systems; see, e.g.: deadlock in database concurrency control (Lafortune 1988); deadlock in multithreaded programs (Gadara project) (Liao et al. 2013). Since our approach is based on SCT, nonblockingness is the only type of liveness property that we consider in our case studies. Thus, the set of marked states used for nonblockingness will be the "parameter" that captures the desired instance of liveness. In our setting, in For-all attacks the attacker *wants* to cause a violation of nonblockingness with respect to the chosen marked states. First of all, G_{nom} in Eq. 7 should be trim for correctness of the system without attacker, as otherwise G_{nom} would contain deadlocks or livelocks. However, G_a should *not* be trim, meaning that the system under attack should contain deadlock or livelock states, i.e., be blocking.

As for the case of safety monitors previously considered, in several instances the violation of the nonblockingness property of interest will be modelled using a dedicated automaton, the *nonblockingness monitor* G_{nm}; one such example is shown in Fig. 7, inspired by Alur and Tripakis (2017) and considered in in Section 5.2. The *marked* states of G_{nm} will record the *violation* of the given nonblockingness property.

On the other hand, if G_{nm} is not given *a priori*, then violations of nonblockingness will be captured as follows: starting from G_a, unmark all states and mark instead the desired (from the viewpoint of the attacker) deadlock and livelock states in G_a, resulting in a suitable G_{nm} model. This is done because deadlock and livelock states are illegal, and the attacker wants the system to reach those illegal states (some or all of them, depending on the type of attack). This is the approach that we will follow in our case study on TCP, as will be explained in Section 6.5.3 and 6.5.4.

Next, we construct $G_{other,a}$ in the same way as in Section 4.1. That is, we model $G_{other,a}$ as the parallel composition of the automata in G_a which are not used to build G_{nm}, and ensure that all the states in $G_{other,a}$ are marked. Note that if G_{nm} is not given as a dedicated automaton and we derive G_{nm} from G_a, then $G_{other,a}$ is empty.

Finally, we define $H_a = Trim(G_{other,a} \parallel G_{nm})$, to represent the specification for the attacker which leads the plant to deadlock or livelock states. As a result, we introduce the algorithm to construct H_a in the case of the nonblockingness properties in Algorithm 2.

```
Input: $G_{nom}$, $G_a$, $G_{nm}$
Output: $H_a$
if $G_{nom}$ is not trim then
    Terminate with empty solution    ▷ The given model is incorrect as the nonblockingness property is violated even if no attacker is present.
end if
if $G_{nm}$ is empty then    ▷ $G_{nm}$ is not given a priori.
    if $G_a$ is trim then
        Terminate with empty solution ▷ The attacker's actions in $G_{C,a}$ and/or $G_{N,a}$ cannot cause a violation of the nonblockingness properties.
    else
        $X_{a,m} = \varnothing$
        Add target deadlock/livelock states in $X_a$ to $X_{a,m}$    ▷ Pick the desired (from the viewpoint of the attacker) deadlock and livelock states.
        $G_{nm} = G_a$
        Let $G_{other,a}$ be empty
    end if
else
    $\Phi_a = \{G_{PA}, G_{PB}, G_{C,a}, G_{N,a}, G_e\}$
    $G_{other,a} = \|\{G \in \Phi_a \mid G$ is not used to construct $G_{nm}\}$
end if
Mark all the states in $G_{other,a}$
$H_a = (Y_a, E_a, g_a, y_{a,0}, Y_{a,m}) = Trim(G_{other,a} \parallel G_{nm})$
if $Y_{a,m} = \varnothing$ then
    Terminate with empty solution    ▷ The attacker's actions can never cause a violation.
end if
return $H_a$
```

Algorithm 2 Attack Specification against Nonblockingness (NonblockSpec).

Proposition 2 Suppose that G_{nom} is trim in Algorithm 2, that is, the given system model is correct in terms of the nonblockingness properties. If $Y_{a,m}$ on line 19 is empty, then no For-all attack exists and no There-exists attack exists.

Proof The proof can be done in the same manner as of Proposition 1, replacing G_{sm} by G_{nm}.

We will discuss several instances of H_a for ABP in Section 5.4 and TCP in Section 6.5.

4.3 Problem formulation

In this section, we formulate the Attack-Supervisor Synthesis Problem (ASSP), which is an instance of a standard SCT partial-observation supervisory control problem, but where the attacker plays the role of "supervisor" and the specification is a *violation* of a given communication protocol property. ASSP is the formal statement of the For-all attack synthesis problem that is solved in our case studies on ABP and TCP.

Attacked-plant As was described earlier, G_C and/or G_N are modified to represent the attacker's ability of inserting and/or discarding packets, resulting in new automata denoted by $G_{C,a}$ and $G_{N,a}$. Next, we form the plant G_a for ASSP as the parallel composition of nominal and infiltrated automata. For example, if the network is infiltrated by the attacker, then $G_a = G_{PA} \parallel G_{PB} \parallel G_C \parallel G_{N,a} \parallel G_e$.

Attack specification Next, we construct H_a using Algorithm 1 or Algorithm 2 based on the given safety or nonblockingness property to be violated, as discussed in Section 4.1 and Section 4.2. Since marking of states in H_a is determined by marking in G_{sm} or G_{nm}, the language marked by H_a, $\mathcal{L}_m(H_a)$, represents strings where the attacker wins, because

(i) These strings are feasible in G_a by construction.
(ii) These strings lead the safety or nonblockingness monitor to a marked state.

As we discussed in Section 3.1, it is reasonable to assume that in PITM attacks the attacker cannot disable or enable the events in the nominal (non-infiltrated) automata, and also that the attacker only observes the events in the automata of the infiltrated components. Thus we define the two partitions of E_a in Eq. 8, from the viewpoint of the attacker (which plays the role of supervisor):

(i) *Controllable events* $E_{a,c}$ and *uncontrollable events* $E_{a,uc}$ for controllability.
(ii) *Obsevable events* $E_{a,o}$ and *unobservable events* $E_{a,uo}$ for observability.

Consequently, we have the following supervisory control problem, under partial observation, for the attacker.

Problem 1 (Attack-Supervisor Synthesis Problem, or ASSP) Let G_a be a plant automaton, under attack, as in Eq. 8; $E_{a,c}$ be a set of controllable events; $E_{a,o}$ be a set of observable events; and $\mathcal{L}_m(H_a) \subset \mathcal{L}(G_a)$ be a marked (non-prefix-closed) specification language. Find a *maximal controllable and observable* sublanguage of $\mathcal{L}_m(H_a)$ with respect to $\mathcal{L}(G_a)$, $E_{a,c}$, and $E_{a,o}$, if a non-empty one exists.

The following theorem states that a non-empty output of ASSP will be the controlled behaviour under a successful For-all attack, highlighting our main results in this paper.

Theorem 1 Let K be a solution of ASSP. Then there exists a For-all attack-supervisor with respect to G_a and K. Conversely, if ASSP has no non-empty solution, then there does not exist a For-all attack-supervisor for $L_a^{spec} = \mathcal{L}_m(H_a)$, with the given controllable and observable event sets for the attacker.

Proof Since K is a controllable and observable sublanguage of $\mathcal{L}_m(H_a) \subset \mathcal{L}(G_a)$, from the "controllability and observability theorem" (Cassandras and Lafortune 2021, p. 197), there exists a supervisor S_P such that $\mathcal{L}_m(S_P/G_a) = K$ and $\mathcal{L}(S_P/G_a) = \overline{K}$. From Definition 2, S_P here is a For-all attack-supervisor with respect to G_a and K. If $\mathcal{L}_m(H_a)$ is not $\mathcal{L}_m(G)$ -closed, we consider S_P to be a marking supervisor, as mentioned in Section 2. Conversely, if the empty set is the only solution to ASSP, then there is no For-all attacker: this is because there is no non-empty language satisfying conditions 1 and 2 in Definition 2.

The *realization* (using standard SCT terminology) of the corresponding (nonblocking) supervisor will encode the control actions of the attacker. By taking the parallel composition of the supervisor's realization with the plant, we obtain an automaton that is language equivalent (generated and marked) to the plant under supervision. Namely, letting R_a be the realization of S_P, it holds that $R_a \parallel G_a$ is language equivalent to the controlled plant S_P/G_a; see (Cassandras and Lafortune 2021; Wonham and Cai 2019). R_a therefore corresponds to a TM-attacker as defined in Von Hippel et al. (2020a). In ASSP, we require maximalty of the controllable and observable sublanguage, since this problem is known to be solvable (Yin and Lafortune 2015).

In the PITM attack model, the assumption of $E_{a,c} \subseteq E_{a,o}$ usually holds. In fact, in all of the scenarios considered in Sections 5 and 6, the condition $E_{a,c} \subseteq E_{a,o}$ will hold. In this important special case, the supremal controllable and observable sublanguage of $\mathcal{L}_m(H_a)$ with respect to $\mathcal{L}(G_a)$, $E_{a,c}$, and $E_{a,o}$ exists and is equal to the supremal controllable and normal sublanguage of $\mathcal{L}_m(H_a)$, denoted by $\mathcal{L}_m(H_a)^{\uparrow CN}$, with respect to $\mathcal{L}(G_a)$, $E_{a,c}$, and $E_{a,o}$. If it is empty, then no For-all attack exists for the given safety or nonblockingness property.

If $\mathcal{L}_m(H_a)^{\uparrow CN} \neq \emptyset$, then this language represents the largest attacked behaviour which is possible in the context of a For-all attack against the safety or nonblockingness property. Any marked string in that language provides an example of a successful attack, which is feasible in G_a and steers G_{nm} or G_{sm} to its marked (illegal) state. Let H_a^{CN} be the trim automaton output by the algorithm for the supremal controllable and normal sublanguage, namely

$$\mathcal{L}_m(H_a^{CN}) = \mathcal{L}_m(H_a)^{\uparrow CN} \tag{22}$$

and

$$\mathcal{L}(H_a^{CN}) = \overline{\mathcal{L}_m(H_a)^{\uparrow CN}} \tag{23}$$

From the controllability and observability theorem of SCT, there exists a partial-observation nonblocking supervisor S_P such that

$$\mathcal{L}(S_P/G_a) = \overline{\mathcal{L}_m(H_a)^{\uparrow CN}} = \mathcal{L}(H_a^{CN}) \tag{24}$$

S_P corresponds to a FOR-ALL attack-supervisor since every string in the controlled behaviour, S_P/G_a, can be extended to a marked string, by nonblockingness of S_P. In other words, it is always eventually possible for the system under attack by S_P to violate the given property.

In the above formulation, $\mathcal{L}_m(H_a)$ may not be $\mathcal{L}_m(G_a)$-closed, since it is possible that $G_a = G_{other,a}$ and all the states in G_a are marked. Therefore, according to the use of G_{sm} and G_{nm}, whenever necessary we define S_P as a marking supervisor by following (5), namely

$$\mathcal{L}_m(S_P/G_a) := \mathcal{L}(S_P/G_a) \cap \mathcal{L}_m(H_a^{CN}) = \mathcal{L}_m(H_a)^{\uparrow CN} \tag{25}$$

As a last step, we need to build a realization of S_P as an automaton that: (i) only changes its state upon the occurrence of observable events, since H_a^{CN} contains transitions with unobservable events; and (ii) whose active event set at each state of the realization is equal to the events *enabled* by the supervisor (attacker) at that state. Noting that marking of states may be relevant in the case of a marking supervisor, the standard process for automaton realization of a partial-observation supervisor (see Section 3.7.2 in Cassandras and Lafortune (2021)) can be followed. From Eqs. 24 and 25, we build an automaton realization of S_P using H_a^{CN}, where S_P is such that

$$\mathcal{L}_m(S_P/G_a) = \mathcal{L}_m(H_a)^{\uparrow CN} \tag{26}$$

and

$$\mathcal{L}(S_P/G_a) = \overline{\mathcal{L}_m(H_a)^{\uparrow CN}} \tag{27}$$

First, we build the observer of H_a^{CN}, $Obs(H_a^{CN})$, with respect to $E_{a,o}$, using the standard process of observer construction (Cassandras and Lafortune 2021). Next, we add self loops for all events in $E_{a,c} \cap E_{a,uo}$ that need to be enabled at each state of $Obs(H_a^{CN})$, obtained by examining the corresponding states of H_a^{CN}. The attack strategy of the successful FOR-ALL attacker is encoded in this realization, as desired.

Based on the above discussion, we introduce Algorithm 3 to synthesize FOR-ALL attacks with respect to the given G_{nom}, G_a and G_m (either a safety or nonblockingness monitor). We also state in Proposition 3 that Algorithm 3 returns the realization of a FOR-ALL attack-supervisor, if it exists, which encodes the attack strategy in order for the attacker to lead the plant to a violation of the given safety/nonblockingness monitor.

Proposition 3 Suppose that H_a on line 2 or line 4 in Algorithm 3 is non-empty, i.e., Algorithm 1 or Algorithm 2 returns a non-empty solution. If ASSP (Problem 1) is solvable, then Algorithm 3 returns the realization of a FOR-ALL attack-supervisor.

```
Require: $G_{nom}, G_a, G_m$
Ensure: $R$
if $G_m$ is a safety monitor then
    $H_a = \text{SafeSpec}(G_{nom}, G_a, G_m)$
else
    $H_a = \text{NonblockSpec}(G_{nom}, G_a, G_m)$
end if
Compute $\mathcal{L}_m(H_a)^{\uparrow CN} = \mathcal{L}_m(H_a^{CN})$ from $G_a$ and $H_a$    ▷ $H_a^{CN}$ is the trim automaton output by the standard algorithm (Cassandras and Lafortune 2021) for the supremal controllable and normal sublanguage.
if $\mathcal{L}_m(H_a)^{\uparrow CN}$ is empty then
    Terminate with empty solution
end if
Compute the realization $R$ of $S_P$ from $H_a^{CN}$ such that $\mathcal{L}_m(S_P/G_a) = \mathcal{L}_m(H_a)^{\uparrow CN}$ and $\mathcal{L}(S_P/G_a) = \overline{\mathcal{L}_m(H_a)^{\uparrow CN}}$
return $R$
```

Algorithm 3 For-all attack synthesis.

Proof Since $E_{a,c} \subseteq E_{a,o}$, if there exists a solution of ASSP, then the supremal controllable and observable sublanguage of $\mathcal{L}_m(H_a)$ exists and is equal to $\mathcal{L}_m(H_a)^{\uparrow CN}$, which is a solution of ASSP. Thus from the proof of Theorem 1, a supervisor S_P such that $\mathcal{L}_m(S_P/G_a) = \mathcal{L}_m(H_a)^{\uparrow CN}$ and $\mathcal{L}(S_P/G_a) = \overline{\mathcal{L}_m(H_a)^{\uparrow CN}}$ is a For-all attack-supervisor. Therefore, if ASSP is solvable, then Algorithm 3 returns the realization of a For-all attack-supervisor.

As long as Algorithm 3 returns a non-empty automaton, from Proposition 3, the above methodology results in a closed-loop system that produces For-all attacks, in the presence of the attacker. Since H_a^{CN} in Algorithm 3 is a trim automaton, we know that at any state in H_a^{CN}, it is possible to reach a marked state, resulting in a violation of the monitor. Therefore, it is always possible for the attacker to eventually win.

Remark 2 When H_a output by Algorithm 1 or Algorithm 2 is not empty (i.e., when it has at least one marked state) but there is no For-all attack-supervisor (i.e., Algorithm 3 returns the empty solution), then we can conclude that there exists at least one There-exists attack-supervisor, according to Definition 3. For instance, one can take the supervisor S_{all} that always enables all events. Then $\mathcal{L}(S_{all}/G_a) = \mathcal{L}(G_a)$ and $\mathcal{L}(G_a) \cap \mathcal{L}_m(H_a) = \mathcal{L}_m(H_a)$ by construction of H_a. Hence, this attack-supervisor can reach any of the marked states in H_a where it "wins", but the closed-loop system will be blocking. Techniques in SCT for synthesizing blocking supervisors, as described in Section 3.5.5 of (Cassandras and Lafortune 2021) for instance, can be employed to guide the design of There-exists attack-supervisors when no For-all attack-supervisor exists. Further investigation of There-exists attack-supervisors is beyond the scope of this paper.

5 ABP Case study

Our first case study for synthesis of For-all attacks is for the Alternating Bit Protocol (ABP), as studied and modelled in Alur and Tripakis (2017). The models of ABP components we use in this section are described in Example 1.

5.1 Safety property models

As introduced in Section 4.1, safety properties are represented by safety monitor automata which define what states in the system must not be reached, i.e., define illegal states. (Alur and Tripakis 2017) provides two safety monitor automata, G^1_{sm} and G^2_{sm}, capturing the violation of safety properties for ABP, depicted in Fig. 6. The marked state q_2 in G^1_{sm} and G^2_{sm} indicates the illegal state, namely, the safety property is violated if the monitor reaches this state from the initial state. G^1_{sm} expresses that:

- *deliver* should happen after *send*, meaning that *deliver* of the ABP receiver and the Receiving client should not happen before the Sending client tells the ABP sender to send a bit to the forward channel.
- After *send* happens, the next *send* should not occur before *deliver* occurs, meaning that the Sending client should wait for the acknowledgement signal from the ABP receiver.

On the other hand, G^2_{sm} expresses that:

- *done* should happen after *deliver*, meaning that *done* of the ABP sender and the Sending client should not happen before the ABP receiver receives the signal and sends the acknowledgement to the ABP sender.
- After *deliver* happens, the next *deliver* should not occur before *done* occurs, meaning that *deliver* cannot happen before the Sending client tells the ABP sender to send the next signal to the forward channel.

Since the safety monitors are provided as dedicated automata, G^1_{sm} and G^2_{sm}, G_{other} in Algorithm 1 is equal to G_{nom}. In our ABP system model, H_{nom} on line 10 in Algorithm 1 has no marked states, thus we state that our ABP model is correct in terms of the safety properties. Namely, the nominal system (without attacker) does not violate the given safety properties.

5.2 Nonblockingness property models

The nonblockingness monitor in Fig. 7, G_{nm}, captures a violation of the nonblockingness property that the entire system should not get stuck, and should not keep invoking *send*. Namely, the first *send* should eventually be followed by a *deliver*. G_{nm} in Fig. 7 is a simplified version of a monitor provided by (Alur and Tripakis 2017) so that our nonblockingness monitor G_{nm} captures that the first transmission is never completed, which is adequate for our case study.

5.3 Attack model

As we consider the system architecture in Fig. 2b for ABP, the attacker infiltrates the forward and/or backward channels. To follow Algorithms 1 and 2, we first construct a modified model of the plant G_a in Eq. 8 under attack. Since the channels of ABP are under attack, we enhance G_{FC} and G_{BC} to those under attack, $G_{FC,a}$ and $G_{BC,a}$, by adding new transitions to represent capabilities of the attacker. Note that if we keep either of the channels nominal, then $G_{FC,a} = G_{FC}$ or $G_{BC,a} = G_{BC}$ accordingly. Therefore, $G_{C,a} = G_{FC,a} \parallel G_{BC,a}$.

The PITM attacker is represented by a modified forward or backward channel that can send the recipient a different packet from the incoming packet. For example, if the attacker has infiltrated the forward channel, then the attacker can send either p'_0 or p'_1 to the ABP receiver regardless of which p_0 or p_1 occurs. Figure 8 shows the attacked forward and backward channels. Red transitions are added to the original channel models in Fig. 4c and d. These new transitions enable the attacker to send whichever packet they want. To construct G_a, we model $G_{FC,a}$ and $G_{BC,a}$ as observer automata of $G^{nd}_{FC,a}$ and $G^{nd}_{BC,a}$, as was done for G_{nom}. Figure 9 depicts $G_{FC,a}$ and $G_{BC,a}$, representing new transitions compared to Fig. 5 as red transitions.

As discussed in Section 3.1, we suppose that the attacker cannot control and observe events outside the channels. Therefore, the event set E_a is partitioned as follows:

- Controllable events: $E_{a,c} = \{p'_0, p'_1, a'_0, a'_1\}$
- Uncontrollable events: $E_{a,uc} = \{send, done, timeout, deliver, p_0, p_1, a_0, a_1\}$
- Observable events: $E_{a,o} = \{p_0, p_1, p'_0, p'_1, a_0, a_1, a'_0, a'_1\}$
- Unobservable events: $E_{a,uo} = \{send, done, timeout, deliver\}$.

We consider that in our attack model, the attacker controls the output packets from the channels so that each safety or nonblockingness monitor in Sections 5.1 and 5.2 reaches its marked state, if possible.

5.4 Examination of the PITM attack for ABP

In this section, we examine the PITM attack for the above safety and nonblockingness properties of ABP according to the following steps:

1. Construct the plant under attack G_a as the parallel composition of the component models of ABP under attack, namely
$$G_a = G_S \parallel G_R \parallel G_{C,a} \parallel G_e \tag{28}$$
where $G_{C,a} = G_{FC,a} \parallel G_{BC,a}$ and $G_e = G_{SC} \parallel G_{RC} \parallel G_T$.
2. Using Algorithm 3, compute the realization of a For-all attack-supervisor with respect to G_{nom}, G_a and the safety/nonblockingness monitor for ABP.

For illustration purposes, if Algorithm 3 returns the realization of an attack-supervisor, we pick one example string from the initial state to one marked state in $\mathcal{L}_m(H_a)^{\uparrow CN}$, which represents one system behaviour under attack that reaches a marked state in the monitor.

G_a varies depending on $G_{C,a}$, namely which channel is under the PITM attack, so we consider the following three cases in each setup:

1. The forward channel is under the PITM attack (i.e. $G_{BC,a} = G_{BC}$):

$$G_a = G_S \parallel G_R \parallel G_{FC,a} \parallel G_{BC} \parallel G_e \tag{29}$$

2. The backward channel is under the PITM attack (i.e. $G_{FC,a} = G_{FC}$):

$$G_a = G_S \parallel G_R \parallel G_{FC} \parallel G_{BC,a} \parallel G_e \tag{30}$$

3. Both channels are under the PITM attack:

$$G_a = G_S \parallel G_R \parallel G_{FC,a} \parallel G_{BC,a} \parallel G_e \tag{31}$$

For clarity of presentation, we henceforth focus on the use of the safety monitor 1 and G_a in Eq. 29 in which the forward channel is under attack, as presented in Example 1. In other words, we consider H_a as the parallel composition of G_a in Eq. 29 and the safety monitor 1 G^1_{sm}. The other cases of Eqs. 30 and 31 and the safety monitor 2 can be examined using the same procedure.

5.4.1 Attack against safety properties

Setup 1 Consider the PITM channels in Fig. 8 which represent a powerful attacker that can send packets to the recipient with whichever bit 0 or 1, regardless of the incoming packets.

Following our procedure, we found that H_a has 168 marked states out of 265 states and H_a^{CN} is non-empty. Here, $\mathcal{L}_m(H_a^{CN}) = \mathcal{L}_m(H_a)$ and $\mathcal{L}(H_a^{CN}) = \mathcal{L}(G_a)$, so H_a is already controllable and normal with respect to G_a, thus the attacker issues no disablement actions. Let us pick the example string $send.p_0.p'_0.deliver.a_0.p'_1.deliver$, which means that the attacker sends the correct packet with bit 0 first, and afterwards sends a fake packet with bit 1 to the ABP receiver when it observes a_0. In other words, the attacker inserts q'_1 soon after it observes a_0. Consequently, G^1_{sm} captures the violation by reaching q_2 with $send.deliver.deliver$.

Setup 2 Let us represent a less-powerful attacker by removing additional transitions from the PITM channels in Fig. 8. First, we remove all red transitions except p'_1 from f_1 to f_0 in Fig. 8a, so that the attacker can send packets with bit 1 at the particular timing. Let $G^{nd}_{FC,wa}$ be the less powerful forward PITM channel derived from $G^{nd}_{FC,a}$. Figure 10 shows $G^{nd}_{FC,wa}$ and $G_{FC,wa} = Obs(G^{nd}_{FC,wa})$. The red transitions are new ones compared to G^{nd}_{FC} and G_{FC}.

Next, we compute G_a, H_a, and H_a^{CN} by following the steps at the beginning of Section 5.4. $G_a = G'_S \parallel G_R \parallel G_{FC,wa} \parallel G_{BC} \parallel G'_e$ has 248 states, and $H_a = G_a \parallel G^1_{sm}$ has 370 states and 228 marked states. H_a^{CN} is non-empty and consists of 1099 states and 771 marked states. In every case, $\mathcal{L}(H_a^{CN}) = \mathcal{L}(G_a)$, so no disabling happens. As the example string in H_a^{CN}, we pick $send.p_0.p'_0.deliver.a_0.p'_1.deliver$ which is the same as that in Setup 1, but H_a^{CN} here is not equivalent. Let $(H_a^{CN})_2$ be H_a^{CN} here and $(H_a^{CN})_1$ be H_a^{CN} in Setup 1. Since $(H_a^{CN})_2^{comp} \times (H_a^{CN})_1$ is non-empty, we conclude that $(H_a^{CN})_2$ lacks some attack strategies, but one additional p'_1 in $G^{nd}_{FC,wa}$ is enough to cause the violation of the safety property.

Setup 3 Let us make the attacker much less powerful than in Setup 2, by building a new automaton of the infiltrated forward channel and changing the sets of controllable and observable events.

Consider the new automaton of the infiltrated forward channel, depicted in Fig. 11. We denote this new automaton by $G^{oneshot,nd}_{FC,a}$ and its observer by $G^{oneshot}_{FC,a}$, namely $G^{oneshot}_{FC,a} = Obs(G^{oneshot,nd}_{FC,a})$. This forward channel means that the attacker can send a fake packet with bit 1 to the ABP receiver only once (one-shot attacker). After the fake packet, the channel's behaviour will get back to normal. Moreover, we consider the following controllable and observable event sets:

- Controllable events: $E_{a,c} = \{p'_1\}$
- Observable events: $E_{a,o} = \{p_0, p_1, p'_0, p'_1, a_0, a_1, a'_0, a'_1\}$

meaning that the attacker can observe events in both of the channels, but can only control p'_1 in the (infiltrated) forward channel. By following the procedure as we have done, G_a in Eq. 29, where $G_{FC,a} = G^{oneshot}_{FC,a}$, has 334 states. Also, $H_a = G_a \parallel G_{sm1}$ has 190 marked states out of 431 states, and H^{CN}_a is non-empty. Moreover, $\mathcal{L}_m(H^{CN}_a) \neq \mathcal{L}_m(H_a)$ and $\mathcal{L}(H^{CN}_a) \neq \mathcal{L}(G_a)$, thus the attacker issues event disablement actions during its attack on the system. For illustration, we pick the following example string in H^{CN}_a:

$$send.p_0.p'_0.deliver.a_0.a'_0.done.send.p_1.p'_1.deliver.a_1.a'_1.done.send.p_0.p'_0.deliver.a_0.a'_1.p'_1.deliver.a_1$$

By observation, the second a'_1 is of nonadversarial error packets which are sent mistakenly, and the second p'_1 is inserted by the attacker. Note that the attacker can observe p'_1 and a'_1 here. Accordingly, this string means that the attacker can lead the system to the undesired state by sending the fake packet p'_1 only once after the observation of one error packet. Moreover, the attacker disables p'_1 several times before sending the fake p'_1. Therefore, in this case, the violation is caused "by chance", since the attacker exploits errors, but that violation is enabled by the attacker's intervention. It is worth mentioning that if we remove the events in the backward channel (i.e., a_0, a_1, a'_0 and a'_1) from $E_{a,o}$, then H^{CN}_a is empty. This means that the attacker needs to observe the behaviour of the backward channel so as to exploit nonadversarial errors to attack. Moreover, if we set $E_{a,c} = \emptyset$ and $E_{a,o} = \{p_0, p_1, p'_0, p'_1, a_0, a_1, a'_0, a'_1\}$, then H^{CN}_a is empty again, meaning that the attacker needs to have the controllability of p'_1 to attack successfully.

5.4.2 Attack against nonblockingness properties

Setup 4 Consider that the attacker wants the system to violate the nonblockingness property represented by the nonblockingness monitor G_{nm} in Fig. 7. Let us examine the system under attack where the forward channels are infiltrated by the attacker, namely G_a in Eq. 29. Note that the forward PITM channel here is that in Fig. 8a which is quite powerful. Since G_{nm} is given as a dedicated automaton, we build $H_a = Trim(G_{other,a} \parallel G_{nm})$ where $G_{other,a} = G_a$.

In this case, G_a consists of 174 states, and H_a comprises 14 states and 13 marked states. H^{CN}_a is non-empty and consists of 10 states and 9 marked states. As the example string in H^{CN}_a, we pick string $send.p0.p1'.a1.timeout$ which means that the attacker sends a fake packet with bit 1 to the ABP receiver after it observes $p0$, and expects the system to suffer from timeout. Moreover, from H^{CN}_a, the attacker-supervisor disables p'_0 to prevent *deliver*, resulting in $\mathcal{L}(H^{CN}_a) \neq \mathcal{L}(G_a)$. Therefore, there exist no *deliver* transitions in H^{CN}_a. This result shows that the attacker successfully leads the system to violate the nonblockingness property that *send* should eventually be followed by *deliver*.

6 TCP case study

Our second case study concerns one of the major protocols in the Internet, the Transmission Control Protocol (TCP) (Postel 1981). TCP is widely used to communicate through unreliable paths. We consider a communication architecture as in Fig. 2a. Each peer sends and receives packets to and from channels, and the network interconnects channels to relay the incoming packets to their destinations. As in Von Hippel et al. (2020a), we consider the connection establishment phase of TCP, based on three-way handshake, and do not model the congestion control part of that protocol.

6.1 Component models of TCP

Let G_{nom} in Eq. 6 be the entire connection establishment part of TCP without an attacker. Based on the architecture of TCP introduced in Von Hippel et al. (2020a), we consider G_{nom} as the parallel composition of the following components:

- $G_{PA} = (X_{PA}, E_{PA}, f_{PA}, x_{PA,0}, X_{PA,m})$: Peer A
- $G_{PB} = (X_{PB}, E_{PB}, f_{PB}, x_{PB,0}, X_{PB,m})$: Peer B
- $G_{C1} = (X_{C1}, E_{C1}, f_{C1}, x_{C1,0}, X_{C1,m})$: Channel 1
- $G_{C2} = (X_{C2}, E_{C2}, f_{C2}, x_{C2,0}, X_{C2,m})$: Channel 2
- $G_{C3} = (X_{C3}, E_{C3}, f_{C3}, x_{C3,0}, X_{C3,m})$: Channel 3
- $G_{C4} = (X_{C4}, E_{C4}, f_{C4}, x_{C4,0}, X_{C4,m})$: Channel 4
- $G_N = (X_N, E_N, f_N, x_{N,0}, X_{N,m})$: Network

namely

$$G_{nom} = G_{PA} \parallel G_{PB} \parallel G_{C1} \parallel G_{C2} \parallel G_{C3} \parallel G_{C4} \parallel G_N \tag{32}$$

Hence, $G_C = G_{C1} \parallel G_{C2} \parallel G_{C3} \parallel G_{C4}$ and $G_e = G_N$, so Eq. 32 reduces to Eq. 7. fvThe event sets are defined as follows:

$$\begin{aligned} E_{PA} = \{ & listen_A, timeout_A, deleteTCB_A, SYN_{AC1}, SYN_{C2A}, \\ & ACK_{AC1}, ACK_{C2A}, FIN_{AC1}, FIN_{C2A}, SYN\_ACK_{AC1}, SYN\_ACK_{C2A} \} \end{aligned} \tag{33}$$

$$\begin{aligned} E_{PB} = \{ & listen_B, timeout_B, deleteTCB_B, SYN_{BC3}, SYN_{C4B}, \\ & ACK_{BC3}, ACK_{C4B}, FIN_{BC3}, FIN_{C4B}, SYN\_ACK_{BC3}, SYN\_ACK_{C4B} \} \end{aligned} \tag{34}$$

$$E_{C1} = \{SYN_{AC1}, SYN_{C1N}, ACK_{AC1}, ACK_{C1N}, FIN_{AC1}, FIN_{C1N}, SYN\_ACK_{AC1}, SYN\_ACK_{C1N}\} \tag{35}$$

$$E_{C2} = \{SYN_{NC2}, SYN_{C2A}, ACK_{NC2}, ACK_{C2A}, FIN_{NC2}, FIN_{C2A}, SYN\_ACK_{NC2}, SYN\_ACK_{C2A}\} \tag{36}$$

$$E_{C3} = \{SYN_{BC3}, SYN_{C3N}, ACK_{BC3}, ACK_{C3N}, FIN_{BC3}, FIN_{C3N}, SYN\_ACK_{BC3}, SYN\_ACK_{C3N}\} \tag{37}$$

$$E_{C4} = \{SYN_{NC4}, SYN_{C4B}, ACK_{NC4}, ACK_{C4B}, FIN_{NC4}, FIN_{C4B}, SYN\_ACK_{NC4}, SYN\_ACK_{C4B}\} \tag{38}$$

$$\begin{aligned} E_N = \{&SYN_{C1N}, SYN_{C3N}, SYN_{NC2}, SYN_{NC4},\\ &ACK_{C1N}, ACK_{C3N}, ACK_{NC2}, ACK_{NC4},\\ &FIN_{C1N}, FIN_{C3N}, FIN_{NC2}, FIN_{NC4},\\ &SYN\_ACK_{C1N}, SYN\_ACK_{C3N}, SYN\_ACK_{NC2}, SYN\_ACK_{NC4}\} \end{aligned} \tag{39}$$

Hence

$$E_{nom} = E_{PA} \cup E_{PB} \cup E_{C1} \cup E_{C2} \cup E_{C3} \cup E_{C4} \cup E_N \tag{40}$$

The subscripts in the event names indicate the directions of packets. For example, “AC1” means packets from Peer A to Channel 1. Note that the subscripts “A” and “B” are added to “listen” and “deleteTCB” to make these events private.

Figures 12, 13, 14, 15, and 16 depict the models of the above TCP components. G_{PA} and G_{PB} illustrate the sequence of three-way handshake and cleanup. We mark the states “closed”, “listen”, and “established” in the automata of the peers, because the peer should not stay in other states during communication, based on Postel (1981). We also mark all states in the automata of the channels and network, to prevent these automata from marking the system. Namely,

$$X_{C1,m} = X_{C1}, \quad X_{C2,m} = X_{C2}, \quad X_{C3,m} = X_{C3}, \quad X_{C4,m} = X_{C4}, \quad X_N = X_{N,m}.$$

6.2 Safety property models

In Von Hippel et al. (2020a), the safety/liveness property of interest is defined as a threat model (TM). TM explains the property using Linear Temporal Logic (LTL) (Baier and Katoen 2008). In this paper, we represent the required properties in Von Hippel et al. (2020a) as finite-state automata.

Von Hippel et al. (2020a) provides one threat model, TM1, for one relevant safety property of TCP. TM1 defines the safety property that if Peer A is at state “closed”, then Peer B should not be at state “established”, because both peers should consecutively reach their “established” states after beginning the connection handshake. Let G_{sm}^{TM1} be the safety monitor to capture the violation of TM1. We represent G_{sm}^{TM1} as the parallel composition of the automata in Fig. 16 where the marked states are only “closed” in Peer A and “established” in Peer B, namely $G_{sm}^{TM1} = G_{PA}^{TM1} \parallel G_{PB}^{TM1}$, where $G_{PA}^{TM1} = (X_{PA}^{TM1}, E_{PA}^{TM1}, f_{PA}^{TM1}, x_{PA,0}^{TM1}, X_{PA,m}^{TM1})$ and $G_{PB}^{TM1} = (X_{PB}^{TM1}, E_{PB}^{TM1}, f_{PB}^{TM1}, x_{PB,0}^{TM1}, X_{PB,m}^{TM1})$. Note that

$$\begin{aligned} &X_{PA}^{TM1} = X_{PA}, \quad E_{PA}^{TM1} = E_{PA}, \quad x_{PA,0}^{TM1} = x_{PA,0}, \quad X_{PA,m}^{TM1} = \{closed\} \neq X_{PA,m},\\ &X_{PB}^{TM1} = X_{PB}, \quad E_{PB}^{TM1} = E_{PB}, \quad x_{PB,0}^{TM1} = x_{PB,0}, \quad X_{PB,m}^{TM1} = \{established\} \neq X_{PB,m} \end{aligned}$$

Hence, the marked states in G_{sm}^{TM1} are illegal states, capturing that Peer A is at “closed” and Peer B is at “established” simultaneously.

Since the safety monitor for TM1, G_{sm}^{TM1}, is derived from G_{PA} and G_{PB}, G_{other} in Algorithm 1 is the parallel composition of the automata of the channels and network, namely $G_{other} = G_{C1} \parallel G_{C2} \parallel G_{C3} \parallel G_{C4} \parallel G_N$. Let H_{nom} in Algorithm 1 be a nominal specification automaton (without attacker) for TM1. In our system model of TCP, $H_{nom} = Trim(G_{nm}^{TM1} \parallel G_{other})$ has no marked states, thus we conclude that our TCP model, without attackers, is correct in terms of TM1.

6.3 Nonblockingness property models

Von Hippel et al. (2020a) also provides two liveness properties denoted as TM2 and TM3. TM2 defines the liveness property that Peer 2 should eventually reach the "established" state. TM3 requires that both peers should not get stuck except at "closed" state, that is, no deadlocks except at "closed" state are allowed. Both TM2 and TM3 requires the system to remain alive during the communication process. In our case study, we translate TM2 and TM3 into "equivalent" nonblockingness properties as expressible representations in the SCT framework, thus slightly abusing the notations "TM2" and "TM3" in Von Hippel et al. (2020a).

We construct the nonblockingness monitors of TM2 and TM3, G_{nm}^{TM2} and G_{nm}^{TM3}, by following Section 4.2. In this case, the nonblockingness monitors are not given as dedicated automata, thus we construct G_{nm}^{TM2} and G_{nm}^{TM3} based on G_{nom} and G_a. We discuss the construction of G_{nm}^{TM2} and G_{nm}^{TM3} in Section 6.5, because to build these automata, we rebuild G_{nom} and G_a as new automata according to TM2 and TM3.

6.4 Attack model

In this section, we explain the attack model for TCP. As we consider the system architecture in Fig. 2a for TCP, the attacker infiltrates the network. First, we construct a modified model of the plant G_a in Eq. 8 under attack. Since the network of TCP is under attack, we enhance G_N to that under attack, $G_{N,a} = (X_{N,a}, E_{N,a}, f_{N,a}, x_{N,a,0}, X_{N,a,m})$, by adding new transitions and events to represent the capabilities of the attacker. Thus,

$$G_a = G_{PA} \parallel G_{PB} \parallel G_{C1} \parallel G_{C2} \parallel G_{C3} \parallel G_{C4} \parallel G_{N,a} \tag{41}$$

Figure 14 depicts the PITM attacked model of the network, $G_{N,a}$, where "$ATTK$" is the set of events of outgoing packets from the network, namely

$$ATTK = \{SYN_{NC2}, ACK_{NC2}, FIN_{NC2}, SYN\_ACK_{NC2}, SYN_{NC4}, ACK_{NC4}, FIN_{NC4}, SYN\_ACK_{NC4}\}, \tag{42}$$

representing multiple transitions, illustrated as the red transitions, by events in $ATTK$. Hence, the event set of $G_{N,a}$, $E_{N,a}$, is as follows:

$$E_{N,\,a} = ATTK \cup E_N \tag{43}$$

where E_N is in Eq. 40. This allows the attacker to be flexible so that the attacker can send any packets and freely choose the destination of packets. As in the discussion in Section 3.1 and in the ABP model, we suppose that the attacker cannot control and observe events outside the network. Hence, the event set of G_a, E_a, is partitioned for controllability and observability of the attacker as follows:

- Controllable events: $E_{a,c} = ATT\,K$ in Eq. 42
- Uncontrollable events: $E_{a,uc} = E_{nom} \setminus E_{a,c}$
- Observable events: $E_{a,o} = E_{N,a}$ in Eq. 43
- Unobservable events: $E_{a,uo} = E_{nom} \setminus E_{a,o}$

In our attack model, the attacker controls the outgoing packets from the network, to lead the safety/nonblockingness monitor to reach its marked (illegal) state.

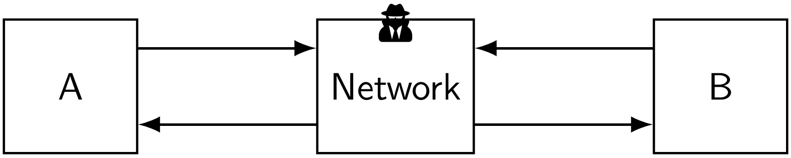


Fig. 3 Communication overview without channels

6.5 Examination of the PITM attack for TCP

In this section, we examine whether a For-all attack exists in terms of TM1, TM2, and TM3. As in Section 5.4, we try to synthesize a For-all attack by the following procedure:

1. Construct the plant under attack G_a in Eq. 41.
2. Using Algorithm 3, compute the realization of a For-all attack-supervisor with respect to G_{nom}, G_a and the safety/nonblockingness monitor for TCP.

As done in Section 5.4, if Algorithm 3 returns the realization, we pick one example string from the initial state to one marked state in $\mathcal{L}_m(H_a)^{\uparrow CN}$, which represents one system behaviour under attack that reaches the marked state in the monitor.

6.5.1 Threat model 1 with channels

Setup 1 Let us consider a powerful attacker represented by $G_{N,a}$ in Fig. 14. By following the above procedure, G_a has 118761 states and 6307 marked states, and H_a has 38270 states and 704 marked states.

Next, we compute H_a^{CN} with respect to G_a and H_a by following the procedure for TM1. As a result, H_a^{CN} is non-empty, having 52783 states and 626 marked states, and $\mathcal{L}_m(H_a^{CN})$ contains the string

$$SYN_{BC3}.SYN_{C3N}.SYN\_ACK_{NC4}.SYN\_ACK_{C4B}.ACK_{BC3}$$

which steers G_{sm}^{TM1} to its marked states. Therefore, we conclude that there exists a For-all attacker S_P defined in Eq. 25 with respect to G_a and H_a in this setup. From $\mathcal{L}(G_a) \neq \mathcal{L}(H_a^{CN})$, the attacker disables some transitions by controllable events in G_a, to always eventually win.

6.5.2 Threat model 1 without channels

Setup 2 One may find that in our TCP model, the channels just relay the incoming packets to their destinations, without any deletion or manipulation of packets. Since we assume ideal channels, we can reduce the communication architecture in Fig. 2a to that without channels, namely the architecture in Fig. 3.

Due to the removal of the channels, to assure the synchronization of the peers and network in the parallel composition, we rename the subscripts of the events in E_{PA} in Eq. 33, E_{PB} in Eq. 34, E_N in Eq. 40, and $E_{N,a}$ in Eq. 43, as follows:

$$\begin{array}{cccc} AC1 \to AN, & C2A \to NA, & BC3 \to BN, & C4B \to NB, \\ C1N \to AN, & NC2 \to NA, & C3N \to BN, & NC4 \to NB \end{array} \tag{44}$$

According to this change, the new G_{nom} and G_a are as follows:

$$G_{nom} = G_{PA} \parallel G_{PB} \parallel G_N \tag{45}$$

$$G_a = G_{PA} \parallel G_{PB} \parallel G_{N,a} \tag{46}$$

G_{nom} in Eq. 45 is trim, consisting of 41 states and 5 marked states, and G_a in Eq. 46 comprises 580 states and 27 marked states, and is not trim. Since we removed the automata of the channels from our system model, G_{other} and $G_{other,a}$ in Algorithm 1 are equal to G_N and $G_{N,a}$, respectively. Even after the removal of the channels, H_{nom} has no marked states.

Noting that $E_{a,c} \subseteq E_{a,o}$ still holds after renaming, let us revisit the procedure at the beginning of Section 6.5 for the construction of H_a and the computation of H_a^{CN} with the new G_a. In this setup, H_a consists of 547 states and 3 marked states, H_a^{CN} with respect to G_a and H_a is non-empty with 513 states and 3 marked states. $\mathcal{L}_m(H_a^{CN})$ contains the following string:

$$SYN_{BN}.SYN_{NB}.ACK_{BN}.ACK_{NB} \tag{47}$$

where $SY\,N_{NB}$ and ACK_{NB} are fake packets inserted by the attacker, tricking Peer B into reaching “established” whereas Peer A does not move out from “closed”. Finally, from $\mathcal{L}(H_a^{CN}) \neq \mathcal{L}(G_a)$ and non-trim G_a, the attack-supervisor disables several transitions in G_a.

Setup 3 As we have done in the ABP case study, let us consider a less-powerful attacker than the previous setups. First, we change the controllable events for the attacker, $E_{a,c}$, as follows:

$$E_{a,c} = \{SYN_{AN}, SYN\_ACK_{NB}\} \tag{48}$$

$$E_{a,uc} = E_a \setminus E_{a,c} \tag{49}$$

whereas $E_{a,o}$ and $E_{a,uo}$ do not change. Note that $E_{a,c} \subseteq E_{a,o}$ still holds. $SY\,N_{AN}$ in $E_{a,c}$ means that the attacker can discard SYN packets coming from Peer A. Next, we redesign the infiltrated network by the attacker, $G_{N,a}$, to represent the reduced capability of the attacker. Figure 15 indicates the model of an infiltrated network by a less powerful attacker, $G^w_{N,a}$. The red transitions are where the attacker can take action.

From the change of $G_{N,a}$ to $G^w_{N,a}$, we change G_a to the entire system under the less powerful PITM attack, namely $G_a = G_{PA} \parallel G_{PB} \parallel G^w_{N,a}$, in this setup. As a result, the new G_a is not trim, consisting of 48 states, 7 marked states, and 1 deadlock state. Because G^{TM1}_{sm} is not different from Setup 2, $G_{other,a} = G^w_{N,a}$ here. Therefore by following the same procedure as above, H_a comprises 47 states and 1 marked state, and H_a^{CN} with respect to G_a and H_a here is non-empty with 63 states and 2 marked states, containing the following string leading G^{TM1}_{sm} to its marked state:

$$SYN_{BN}.SYN\_ACK_{NB}.ACK_{BN} \tag{50}$$

In conclusion, there still exists a For-all attacker with the less-powerful PITM model.

From $G^w_{N,a}$ in Fig. 15, the attacker can send a fake SYN_ACK packet to Peer B only when Peer B enters “SYN sent” state, and the attacker must keep Peer A at “closed” state. Hence, the attacker must disable $SY\,N_{AN}$ at “closed” state in G_{PA} shown in Fig. 16 where the subscripts of events are changed as in Eq. 44, and $\mathcal{L}(H_a^{CN}) \neq \mathcal{L}(G_a)$ reflects this disablement action. Therefore, if SYN_{AN} is uncontrollable, then H_a^{CN} is empty.

6.5.3 Threat model 2

Consider G_{PA}, G_{PB}, G_N, and $G_{N,a}$ in Setup 2. Recall that Threat Model 2 (TM2) requires Peer A to reach its "established" state eventually. To design the nonblockingness monitor which captures the violation of TM2, we first unmark all states of G_{PA} and mark its "established" state. Let G_{PA}^{TM2} be a new automaton derived from G_{PA} in Fig. 16a by this marking and renaming as in Eq. 44. In contrast to the construction of safety monitors, G_{PA}^{TM2} captures the desired behaviour where Peer A reaches its "established" state eventually. Thus we construct G_{nom} and G_a as follows:

$$G_{nom} = G_{PA}^{TM2} \parallel G_{PB} \parallel G_N \tag{51}$$

$$G_a = G_{PA}^{TM2} \parallel G_{PB} \parallel G_{N,a} \tag{52}$$

To prevent it from marking G_{nom} and G_a, we mark all states in G_{PB}, so the marked states of G_{nom} and G_a are determined by the "established" state in G_{PA}^{TM2}.

Setup 4 Let us construct H_a by following Algorithm 2. First of all, G_{nom} in Eq. 51 is trim, thus the system model without attacker is correct in terms of TM2, meaning that Peer A eventually reaches its "established" state. So, let us proceed to the next step. From the additional transitions of $G_{N,a}$ in Fig. 14, G_a in Eq. 52 is not trim, thus G_a contains several deadlock and/or livelock states. In this scenario, we build G_{nm}^{TM2} for TM2 based on G_a and not as a separate automaton. In G_a, there are 25 deadlock states. These deadlock states are those the attacker wants G_a to reach so that Peer A cannot always reach its "established" state. To design G_{nm}^{TM2} representing the violation of TM2, namely reaching the deadlock states, we unmark all states in G_a and then mark all the deadlock states. Hence, let G_{nm}^{TM2} be the new automaton built by the marking of deadlock states in G_a, so that every string in $\mathcal{L}_m(G_{nm}^{TM2})$ ends with one of the deadlock states in G_a. Finally, the specification automaton for the attacker is $H_a = Trim(G_{nm}^{TM2})$.

In this case, H_a consists of 580 states and 25 deadlock states which are determined by G_a, and H_a^{CN} with respect to G_a and H_a is non-empty, where $\mathcal{L}_m(H_a^{CN})$ contains the following string:

$$SYN_{AN}.SYN\_ACK_{NA}.ACK_{AN}.FIN_{NA}.SYN_{BN}.SYN_{NB}.ACK_{AN} \tag{53}$$

SYN_ACK_{NA}, FIN_{NA}, and SYN_{NB} in Eq. 53 are fake packets inserted by the attacker. This string makes Peer A and Peer B stuck at "close wait" state and at "i1" state, respectively. Here, $\mathcal{L}(H_a^{CN}) = \mathcal{L}(G_a)$, thus the attacker just inserts fake packets and does not disable any controllable events. In conclusion, there exists a For-all attack for TM2 in this setup.

6.5.4 Threat model 3

In this section, we examine whether any For-all attacks against the Threat Model 3 (TM3) exist. TM3 captures the following nonblockingness requirement for the system: the peers should not suffer from any deadlocks if they leave "closed" state.

Consider G_{PA}, G_{PB}, G_N, and $G_{N,a}$ in Setup 2 again. Since TM3 is defined by a nonblockingness property, we design a nonblockingness monitor for TM3 similarly as a monitor for TM2, discussed in Section 6.5.3. According to TM3, we first unmark all states and mark "closed" state in G_{PA} and G_{PB}. Let G_{PA}^{TM3} and G_{PB}^{TM3} be the new automata derived from G_{PA} and G_{PB} in Fig. 16 by this marking and renaming as in Eq. 44, respectively. Since G_{PA}^{TM3} and G_{PB}^{TM3} capture the desired behaviour of the system model, we construct G_{nom} and G_a as follows:

$$G_{nom} = G_{PA}^{TM3} \parallel G_{PB}^{TM3} \parallel G_N \tag{54}$$

$$G_a = G_{PA}^{TM3} \parallel G_{PB}^{TM3} \parallel G_{N,a} \tag{55}$$

Since all states in G_N and $G_{N,a}$ are marked, the marked states in G_{nom} and G_a are determined by "closed" state of G_{PA}^{TM3} and G_{PB}^{TM3}.

Setup 5 We construct H_a using Algorithm 2. First, G_{nom} in Eq. 54 consisting of 41 states and 1 marked state is trim, thus our system model without attacker is correct in terms of TM3. This means that neither Peer A nor Peer B suffers from deadlocks and/or livelocks when they are not at "closed" state. In the next step, due to $G_{N,a}$, G_a in Eq. 55 comprising 580 states and 3 marked states is not trim, thus G_a contains deadlock and/or livelock states. In particular, G_a has 25 deadlock states and no livelock states. Since the nonblockingness monitor for TM3, G_{nm}^{TM3}, is not given as a dedicated automaton, G_{nm}^{TM3} is derived from G_a by unmarking all states and marking the 25 deadlock states in G_a. Finally, we have $H_a = Trim(G_{nm}^{TM3})$.

As a result, H_a in this setup consists of 580 states and 25 marked (deadlock in G_a) states, and H_a^{CN} with respect to G_a and H_a is non-empty with 660 states and 25 marked states. To see a behaviour of the system under the attack, we pick the following example string in $\mathcal{L}_m(H_a^{CN})$:

$$listen_A.SYN_{BN}.SYN_{NA}.SYN\_ACK_{AN}.ACK_{NA}.FIN_{AN}.ACK_{NA} \tag{56}$$

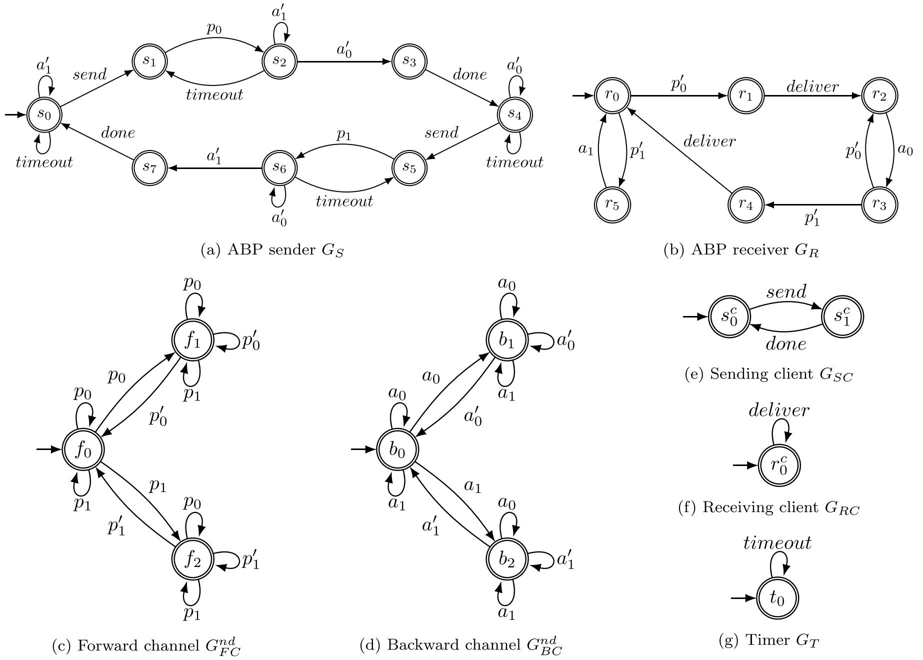
where the fifth and seventh ACK_{NA} are fake packets sent from the attacker to Peer A. This string makes Peer A and Peer B stuck at "FIN wait 2" and "SYN sent", respectively. Here, $\mathcal{L}(H_a^{CN}) = \mathcal{L}(G_a)$, thus the attacker inserts fake packets and does not disable any controllable events. To sum up, there exists a For-all attack for TM3 in this setup.

7 Conclusion

We investigated the synthesis problem of For-all attacks under which the attacker can always eventually win, in the specific context of person-in-the-middle attacks on two well-known communication protocols, ABP and TCP, where in each case a sender and a receiver communicate over channels and a network. We formulated this problem in the framework of discrete event systems in order to leverage its supervisory control theory for attacker synthesis. We showed that the synthesis of a For-all attack can be formulated as the problem of finding a maximal controllable and observable sublanguage of the specification language for the attacker with respect to the given plant and the capabilities of the attacker in terms of controllable and observable events. The plant is the combination of the models of the sender, receiver, channels, and network. The specification language for the attacker is derived from a suitable specification automaton; we described in Sections 4.1 and 4.2 how to construct that automaton for various examples of safety properties and nonblockingness properties, respectively. The goal of the attacker is to force a violation of the given safety or nonblockingness property of the communication protocol. We formally derived in Sections 5 and 6, when they existed, several For-all person-in-the-middle attacks for ABP and TCP under different scenarios of attacker capabilities and safety or nonblockingness property to be violated. We are not aware of any prior work where formal methods are used to synthesize attacks on ABP. For the case of TCP, our results extend the results in Von Hippel et al. (2020a), where the authors considered the synthesis of There-exists attacks under which the attacker may not always win, but will sometimes win. In total, we presented four setups for ABP and five setups for TCP, where the plant, specification, and event partitions vary. Further setups are discussed in the expanded version of this paper available at Matsui and Lafortune (2022).

In the PITM attack setups we considered, it was reasonable to assume that the attacker observes all the events it controls. Hence, the synthesis of a For-all attack reduced to the computation of the supremal controllable and normal sublanguage in supervisory control theory of discrete event systems. This means that the methodology that we employed for ABP and TCP could be applied to other protocols and other types of attacks that can be modelled as additional transitions in the transition structure of the protocol. This shows that formulating attacker synthesis as a supervisory control problem is a powerful approach in the study of vulnerabilities of distributed protocols. In the future, it would be of interest to investigate how to make distributed protocols more resilient to both There-exists and For-all attacks.

Appendix : A: Figures of ABP



(a) ABP sender G_S

(b) ABP receiver G_R

(c) Forward channel G_{FC}^{nd}

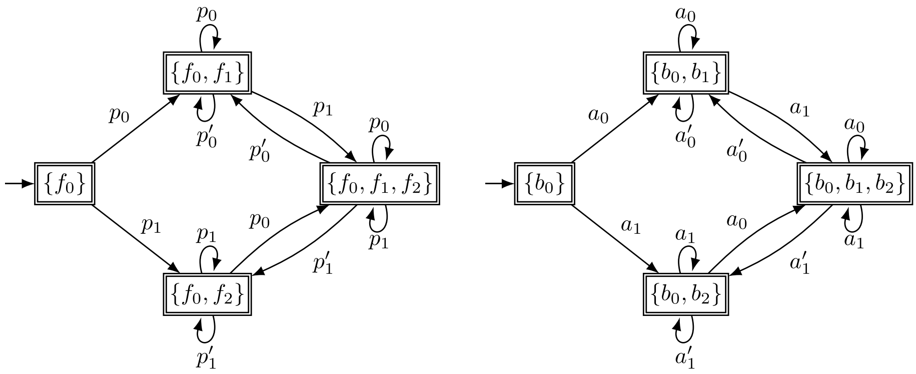
(d) Backward channel G_{BC}^{nd}

(e) Sending client G_{SC}

(f) Receiving client G_{RC}

(g) Timer G_T

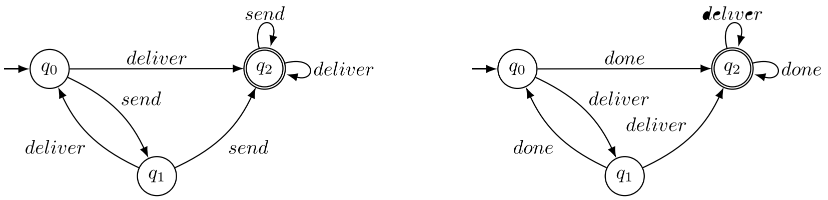
Fig. 4 Models of ABP components adopted from Alur and Tripakis (2017)



(a) Forward channel G_{FC}

(b) Backward channel G_{BC}

Fig. 5 Observer automata of channels



(a) Safety monitor 1 G^1_{sm}; *send* and *deliver* should happen in the right order.

(b) Safety monitor 2 G^2_{sm}; *deliver* and *done* should happen in the right order.

Fig. 6 Safety monitors from Alur and Tripakis (2017)

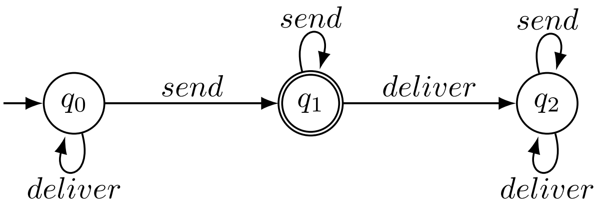
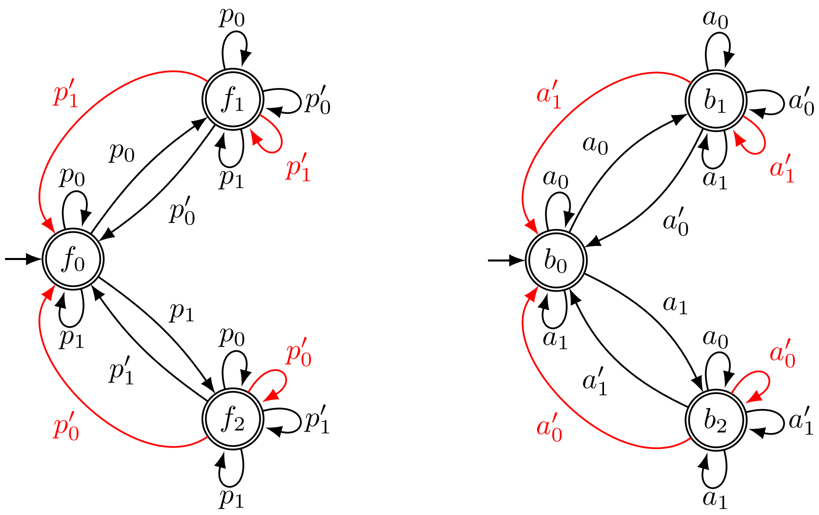


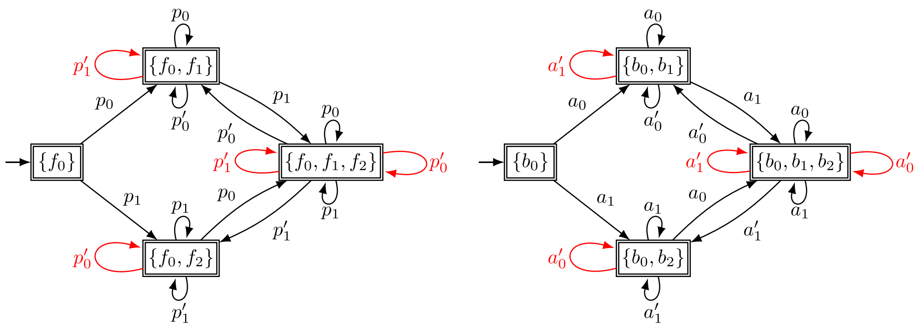
Fig. 7 Nonblockingness monitor G_{nm} inspired by Alur and Tripakis (2017); the first *send* should eventually be followed by a *deliver*



(a) Forward MITM channel $G^{nd}_{FC,a}$

(b) Backward MITM channel $G^{nd}_{BC,a}$

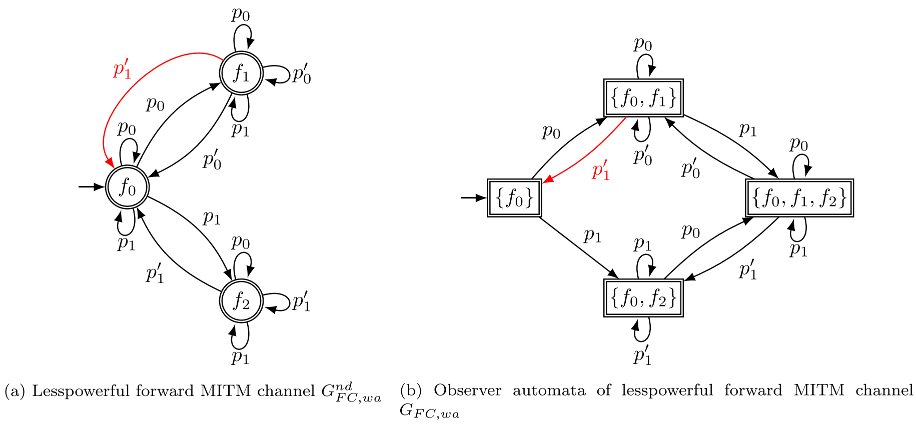
Fig. 8 Channel models under the MITM attack



(a) Foward MITM channel $G_{FC,a}$

(b) Backward MITM channel $G_{BC,a}$

Fig. 9 Observer automata of the MITM channels



(a) Lesspowerful forward MITM channel $G_{FC,wa}^{nd}$

(b) Observer automata of lesspowerful forward MITM channel $G_{FC,wa}$

Fig. 10 Lesspowerful forward MITM channel

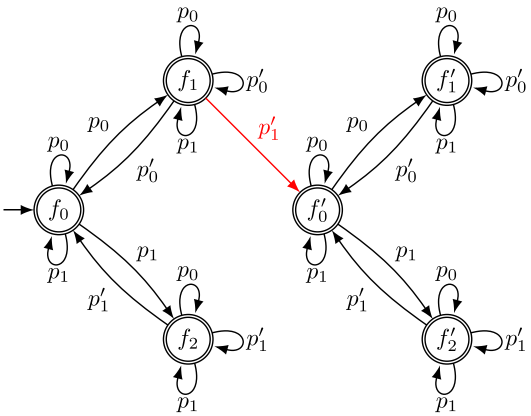
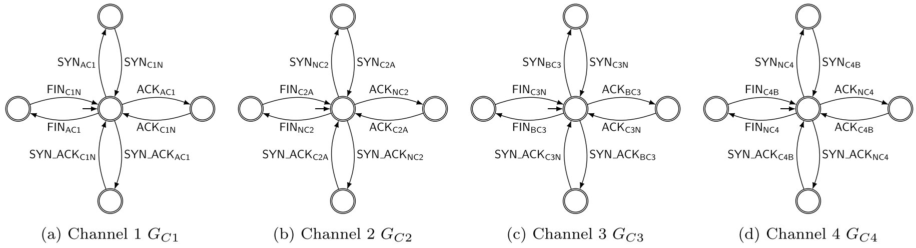


Fig. 11 One-shot forward MITM channel

Appendix : B: Figures of TCP



(a) Channel 1 G_{C1} (b) Channel 2 G_{C2} (c) Channel 3 G_{C3} (d) Channel 4 G_{C4}

Fig. 12 Channel models of TCP

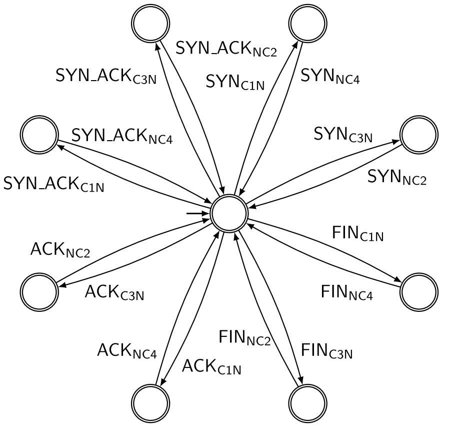


Fig. 13 Network model of TCP

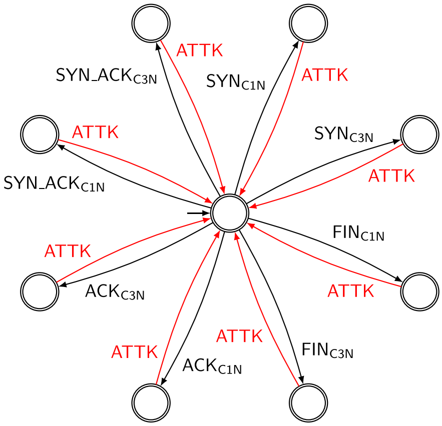


Fig. 14 Network model under the MITM attack $G_{N,a}$

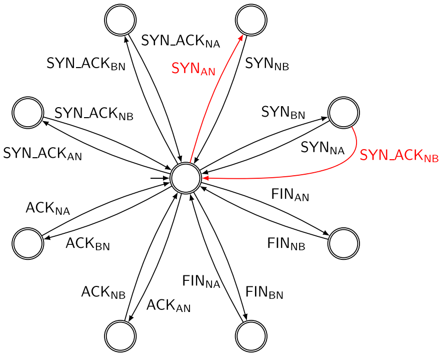
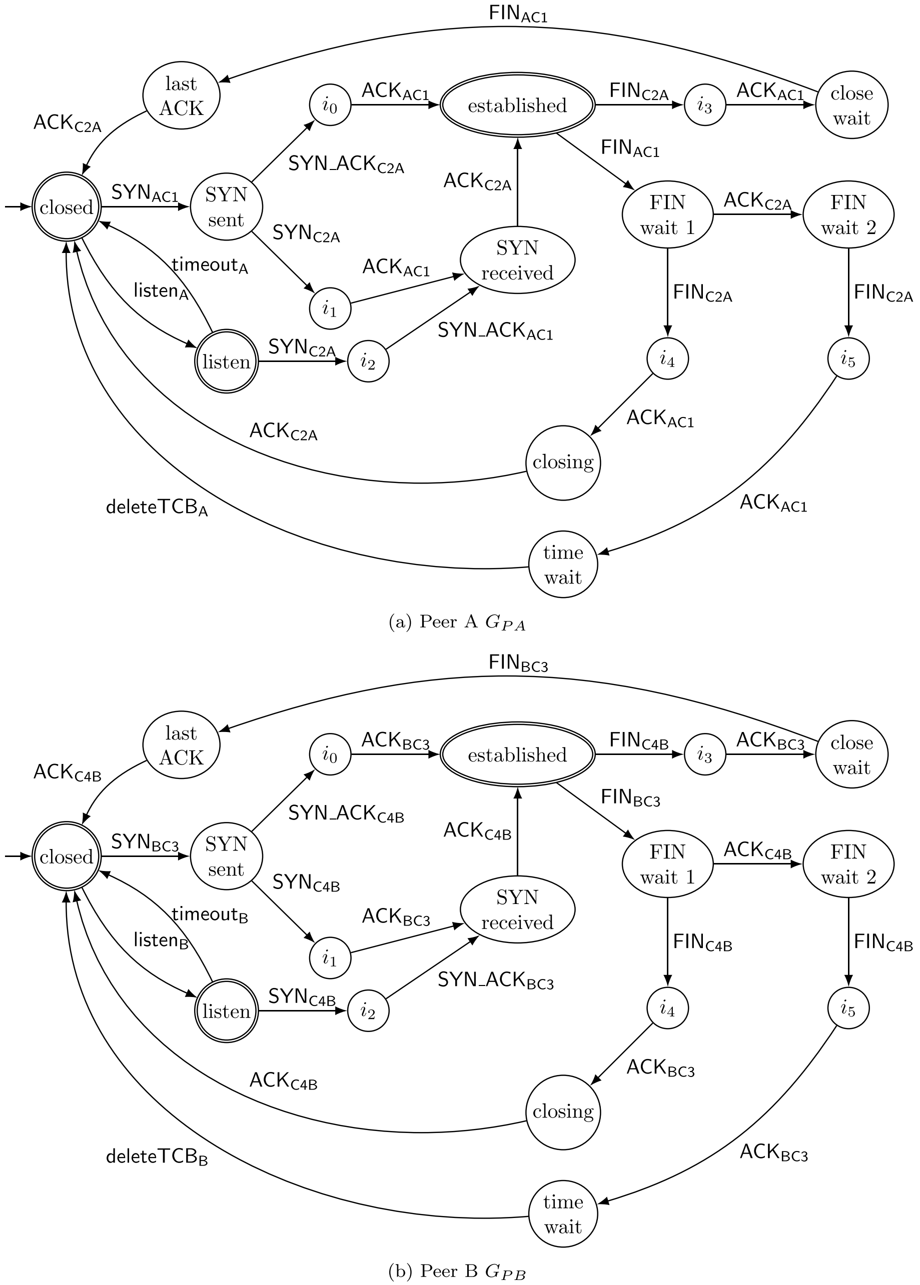


Fig. 15 Network model under the lesspowerful MITM attack $G^{w}_{N,a}$



(a) Peer A G_{PA}

(b) Peer B G_{PB}

Fig. 16 Peers with timeout

Acknowledgements This research was supported in part by the US NSF under grant CNS-1801342. We thank the reviewers for their pertinent comments that helped to improve the presentation of our results.

Declarations

Conflict of Interests The authors declare that they have no conflict of interest.

References

Alur R, Tripakis S (2017) Automatic synthesis of distributed protocols. ACM SIGACT News 48 (1):55–90. https://doi.org/10.1145/3061640.3061652

Bagheri H, Kang E, Malek S et al (2015) Detection of design flaws in the android permission protocol through bounded verification. In: International symposium on formal methods. https://doi.org/10.1007/978-3-319-19249-9_6. Springer, pp 73–89

Baier C, Katoen JP (2008) Principles of model checking. MIT Press, Cambridge

Brandt R, Garg V, Kumar R et al (1990) Formulas for calculating supremal controllable and normal sublanguages. Syst Control Lett 15(2):111–117. https://doi.org/10.1016/0167-6911(90)90004-E

Carvalho LK, Wu YC, Kwong R et al (2018) Detection and mitigation of classes of attacks in supervisory control systems. Automatica 97:121–133. https://doi.org/10.1016/j.automatica.2018.07.017

Cassandras CG, Lafortune S (2021) Introduction to discrete event systems, 3rd edn. Springer International Publishing AG, Cham. https://doi.org/10.1007/978-3-030-72274-6

Cho H, Marcus SI (1989) On supremal languages of classes of sublanguages that arise in supervisor synthesis problems with partial observation. Math Control Signals Syst 2(1):47–69. https://doi.org/10.1007/BF02551361

Ehlers R, Lafortune S, Tripakis S et al (2017) Supervisory control and reactive synthesis: a comparative introduction. Discrete Event Dynamic Systems 27(2):209–260. https://doi.org/10.1007/s10626-015-0223-0

Holzmann GJ, Lieberman WS (1991) Design and validation of computer protocols, vol 512. Prentice Hall, Englewood Cliffs

Jero S, Lee H, Nita-Rotaru C (2015) Leveraging state information for automated attack discovery in transport protocol implementations. In: 2015 45th Annual IEEE/IFIP international conference on dependable systems and networks. https://doi.org/10.1109/DSN.2015.22. IEEE, pp 1–12

Kang E, Milicevic A, Jackson D (2016) Multi-representational security analysis. In: Proceedings of the 2016 24th ACM SIGSOFT international symposium on foundations of software engineering. https://doi.org/10.1145/2950290.2950356, pp 181–192

Kumar R, Nelvagal S, Marcus SI (1997) A discrete event systems approach for protocol conversion. Discrete Event Dynamic Systems 7(3):295–315. https://doi.org/10.1023/A:1008258331497

Lafortune S (1988) Modeling and analysis of transaction execution in database systems. IEEE Trans Autom Control 33(5):439–447. https://doi.org/10.1109/9.1222

Liao H, Wang Y, Stanley J et al (2013) Eliminating concurrency bugs in multithreaded software: a new approach based on discrete-event control. IEEE Trans Control Syst Technol 21(6):2067–2082. https://doi.org/10.1109/TCST.2012.2226034

Lin L, Zhu Y, Su R (2019). Synthesis of covert actuator attackers for free. arXiv:190410159

Matsui S, Lafortune S (2022) Synthesis of winning attacks on communication protocols using supervisory control theory: Two case studies. arXiv:210206028

Meira-Góes R, Marchand H, Lafortune S (2019) Towards resilient supervisors against sensor deception attacks. In: 2019 IEEE 58Th conference on decision and control (CDC). https://doi.org/10.1109/CDC40024.2019.9029737. IEEE, pp 5144–5149

Meira-Góes R, Kang E, Kwong RH et al (2020) Synthesis of sensor deception attacks at the supervisory layer of cyber-physical systems. Automatica 121:109,172. https://doi.org/10.1016/j.automatica.2020.109172

Postel J (1981) Transmission control protocol. https://doi.org/10.17487/RFC0793

Rudie K, Wonham WM (1990) Supervisory control of communicating processes. In: Proceedings of the IFIP WG6 1 tenth international symposium on protocol specification testing and verification X, pp 243–257

Rudie K, Wonham WM (1992) Protocol verification using discrete-event systems. In: Proceedings of the 31st IEEE conference on decision and control. https://doi.org/10.1109/CDC.1992.370955. IEEE, pp 3770–3777

Saleh K (1996) Synthesis of communications protocols: an annotated bibliography. ACM SIGCOMM Computer Communication Review 26(5):40–59. https://doi.org/10.1145/242896.242900

Su R (2018) Supervisor synthesis to thwart cyber attack with bounded sensor reading alterations. Automatica 94:35–44. https://doi.org/10.1016/j.automatica.2018.04.006

Von Hippel M (2020) Korg. https://github.com/maxvonhippel/AttackerSynthesis

Von Hippel M, Vick C, Tripakis S et al (2020a) Automated attacker synthesis for distributed protocols. arXiv:200401220

Von Hippel M, Vick C, Tripakis S et al (2020b) Automated attacker synthesis for distributed protocols. In: International conference on computer safety, reliability, and security. https://doi.org/10.1007/978-3-030-54549-9_9. Springer, pp 133–149

Wakaiki M, Tabuada P, Hespanha JP (2019) Supervisory control of discrete-event systems under attacks. Dynamic Games and Applications 9(4):965–983. https://doi.org/10.1007/s13235-018-0285-3

Wonham WM, Cai K (2019) Supervisory control of discrete-event systems. Springer, Berlin. https://doi.org/10.1007/978-3-319-77452-7

Yin X, Lafortune S (2015) Synthesis of maximally permissive supervisors for partially-observed discrete-event systems. IEEE Trans Autom Control 61 (5):1239–1254. https://doi.org/10.1109/TAC.2015.2460391

Publisher's note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Shoma Matsui received the B. Eng. in Electrical and Information Engineering degree from Osaka City University in 2019, and the M. Sc. in Engineering degree from University of Michigan in 2020. He is currently pursuing his Ph.D. degree in the Department of Electrical and Computer Engineering at Queen's University, Kingston, Canada. His research interests include secrecy, privacy, and safety of Discrete-Event Systems, and applications to computer and network systems.

Stéphane Lafortune received the B.Eng degree from École Polytechnique de Montréal in 1980, the M.Eng degree from McGill University in 1982, and the Ph.D degree from the University of California at Berkeley in 1986, all in electrical engineering. Since September 1986, he has been with the University of Michigan, Ann Arbor, where he is a Professor of Electrical Engineering and Computer Science. In March 2018, he was appointed as the N. Harris McClamroch Collegiate Professor of Electrical Engineering and Computer Science. Lafortune is a Fellow of the IEEE (1999) and of IFAC (2017). He received the Presidential Young Investigator Award from the National Science Foundation in 1990 and the Axelby Outstanding Paper Award from the Control Systems Society of the IEEE in 1994 (for a paper co-authored with S.-L. Chung and F. Lin) and in 2001 (for a paper co-authored with G. Barrett). Lafortune's research interests are in discrete event systems and include multiple problem domains: modeling, diagnosis, control, optimization, and applications to computer and software systems. He served as Editor-in-Chief of the Journal of Discrete Event Dynamic Systems: Theory and Applications from 1/2015 to 12/2020.