CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Improving the resilience of a CACC controller against jamming attacks

A simulation study and assessment of the impact of barrage jamming attacks conducted on existing and proposed models

Master's thesis in Computer science and engineering

LUDVIG OHLSSON

KARTHIK SHARMA

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2024

Master's thesis 2024

Improving the resilience of a CACC controller against jamming attacks

A simulation study and assessment of the impact of barrage jamming attacks conducted on existing and proposed models

LUDVIG OHLSSON

KARTHIK SHARMA

UNIVERSITY OF GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2024

Improving the resilience of a CACC controller against jamming attacks
A simulation study and assessment of the impact of barrage jamming attacks conducted on existing and proposed models
LUDVIG OHLSSON
KARTHIK SHARMA

Supervisor: Johan Karlsson, Department of Computer Science and Engineering
Advisor: Mateen Malik, Research Institutes of Sweden
Examiner: Johan Karlsson, Department of Computer Science and Engineering

Master's Thesis 2024
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2024

Improving the resilience of a CACC controller against jamming attacks
A simulation study and assessment of the impact of barrage jamming attacks conducted on existing and proposed models
LUDVIG OHLSSON
KARTHIK SHARMA
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

This research explores how the resilience of a particular Cooperative Adaptive Cruise Controller (CACC) algorithm "P1" can be improved, specifically examining the impact of wireless jamming attacks on Vehicle-to-Vehicle (V2V) communication and implementing rudimentary countermeasures. Previous research has shown that there are vulnerabilities in this P1 algorithm and based on this research we propose and implement various methods to enhance its resilience and finally compare these results to the original algorithm. We utilize the ComFASE simulation environment, which integrates existing simulation frameworks such as Plexe, Veins, SUMO and OmNET++ to perform vehicular and network dynamics simulations. ComFASE also adds on a layer of fault and attack injection tools which we use in conjunction with the frameworks to inject barrage jamming attacks, design countermeasures and finally carry out another round of attack injections to gauge their efficacy. The P1 controller algorithm described further in our thesis was found to cause collisions when under attack in 1475 out of the 3575 experiments we ran. In order to improve its resilience, we propose fallback models as countermeasures. Our best proposed fallback called Model 4c managed to reduce the number of collisions to 637 out of the same 3575 experiments. With this simple yet effective fallback mechanism, we have managed to get a collision reduction of 56.81% when covering a broad range of noise values. If we only consider noise levels greater than 0.66 mW, we have a close to 100% reduction in number of collisions. These results show that the resilience of the P1 controller against barrage jamming attacks can be significantly improved even with just relatively simple modifications to existing algorithms.

Keywords: Computer, science, computer science, engineering, project, thesis, cybersecurity, platooning, resilience, CACC.

Acknowledgements

We wish to express our deepest gratitude to our supervisor Mateen Malik, and our examiner Johan Karlsson, whose invaluable support made this thesis possible. We extend sincere appreciation to RISE and our team member Christos Profentzas for their assistance throughout the project. Lastly, we would like to acknowledge the unwavering support of our families and well-wishers.

Ludvig Ohlsson, Karthik Sharma, Gothenburg, 2024-12-29

Contents

List of Figures

List of Tables

1

Introduction

In an era when vehicular technology is rapidly advancing, the concept of an interconnected vehicle system emerges as a promising alternative in transportation. At the heart of this shift lie specialized controllers known as Cooperative Adaptive Cruise Controllers (CACC) which are commonly facilitated by Vehicle-to-Vehicle (V2V) communication. These sophisticated systems enable vehicles to collaborate through wireless networks, offering a range of benefits, from enhanced traffic flow to reduced fuel consumption and increased safety [1].

CACCs have the ability to devise what is known as platooning – a strategic formation wherein a convoy of vehicles follows the lead of a designated lead vehicle. The leading vehicle can be either self-driven or human-driven depending on the implementation. By maintaining precise proximity and synchronized movement, these controllers effectively diminish aerodynamic drag by up to 35% [2], thereby optimizing fuel burn while navigating through traffic.

However, with cyber-attacks becoming increasingly widespread in the digital landscape, the integrity of the V2V communication system becomes a huge concern. The need to increase the resilience of such CACC algorithms by integrating top-notch security measures into their design framework against such attacks becomes apparent.

In this thesis, we will be looking closely at a CACC model referred to henceforth as the "P1" model which stands for the first CACC model described in the Plexe paper [3]. It is based on the equations for longitudinal control derived from Rajamani et al. [4]. This model will be analyzed to discover its vulnerabilities to attacks on a V2V network and improved upon to strengthen its resilience.

1.1 Aim

This thesis seeks to explore in detail how CACC controller algorithms can be made robust and resilient against a specific type of communication threats, namely barrage jamming attacks. These network interference attacks, when employed against a V2V network, have the potential to disrupt the communication between vehicles and thereby affect the CACC controllers. This may lead to controller malfunction due to loss of packets or due to erratic values, and in the case of platooning, a potential failure to maintain formation can cause vehicles to collide. In order to address these critical vulnerabilities, this report will attempt to answer the following research questions in detail.

Research Question 1 - Why is the P1 controller so sensitive to jamming attacks?

Research Question 2 - Can we come up with simple modifications of the P1 controller to reduce or even prevent collisions?

Research Question 3 - What is the role of the vehicle's radar in the P1 controller? Would it be possible for us to take advantage of it to prevent crashes in our fallbacks?

1.2 Related works

Rajamani et al. [4] lay down the foundational equations to form the basis of the work done in our thesis and these were crucial to understand before we began making changes in the model to strengthen its resilience against jamming attacks. The unmodified P1 controller model assumes ideal communication parameters, which is very unlikely in the real world, as the threat of malicious actors is a very well known fact. Mateen et al. [5] present a fault and attack simulation engine ComFASE, which they have used to model and gain insight into attacks against the platooning model described by Segata et al. [3]. The longitudinal control equations in Segata et al. are directly inspired by Rajamani et al. while the latitudinal control is their own simplified implementation. This paper [5] is the main inspiration as well as the baseline for conducting our experiments and we will compare our results with theirs to see if we get an improvement.

Looking at the results of Mateen et al., we came up with a new set of test campaigns by performing post injection analysis, specifically targeting known vulnerable areas to limit the number of experiments; not only shortening the time taken for each campaign to be simulated, but also dive deeper into more relevant analysis to our Research Questions. Mateen et al. [6] further elaborate on various different types of jamming attacks such as barrage jamming, destructive interference and deceptive jamming and their effects. The paper makes a comparison of the results obtained while each of these attacks are injected into the first CACC model (named P1 in our thesis) described in Segata et al. [3] as-is without any modifications or fallbacks.

1.3 Report structure

This report is organized into six chapters. Following this introduction, *Chapter 2: Background* provides a detailed background on the tools and systems used as well as the theory behind. *Chapter 3: Proposed Fallback Mechanisms* lists the fallback mechanisms developed for strengthening the resilience of the P1 controller introduced in the previous chapter. Following that, we describe the methods used for testing the aforementioned fallbacks in detail in *Chapter 4: Experimental Setup*. Finally, the results of evaluating the fallback mechanisms is presented in *Chapter 5: Experimental Results*, and then discussed in *Chapter 6: Conclusion*.

2

Background

This chapter gives an overview of the tools and systems used for the thesis. We mention the simulators themselves and how each one plays a role in our test-bench setup and how they interact with each other. We write about different kinds of jamming attacks and concentrate especially on Barrage Jamming. We also mention the different severity levels for the jamming attacks.

2.1 ComFASE simulation environment

We have used several simulators to conduct our experiments and every one of them is crucial in obtaining the desired output of our thesis work. ComFASE [5] is an open source fault and attack injection environment designed to inject communication faults and simulate various attack vectors, such as wireless jamming, within networked systems. Developed as an extension to Veins by RISE Sweden, this tool is essential for evaluating the resilience of communication protocols and systems in the presence of intentional disruptions, such as cyber-attacks. These attacks can range from packet loss to severe signal degradation, mimicking real-world scenarios where networks encounter interference or deliberate attacks. By injecting such faults and attack vectors into our simulations, ComFASE helps us test the resilience and robustness of vehicular communication systems and driver support functions.

In our experiments, we focused particularly on Barrage Jamming since it is much easier to implement compared to other types of jamming and hence also easy for an attacker to conduct in the real world. We simulated it on the communication link between vehicles in a platoon, forcing the P1 model to operate under conditions of severely limited or blocked communication. ComFASE's ability to inject realistic communication faults allowed us to add and test attack tolerance to this particular CACC model. ComFASE is built on top of the following simulators:

1. **Veins** :
   Veins (Vehicles In Network Simulation) [7] is an extension to OMNeT++ and integrates SUMO. It bridges the gap between network simulation and traffic simulation, giving us a complete view of how vehicular networks function in real-world conditions.

   Veins is essential for tying together the traffic and network simulation elements of our work. It allows us to study how communication protocols perform in real vehicular environments, factoring in dynamic traffic patterns and vehicle interactions.

2. **Plexe** :
   Plexe (Platooning Extension) [3] is an open-source software developed as an extension to Veins to allow the simulation of platooning, which is when vehicles drive in tightly coordinated formations, on OMNeT++. It supports the necessary communication protocols for platooning, allowing vehicles to exchange real-time data about their speed, position, and trajectory, ensuring

they can stay in formation. Furthermore, like Veins, Plexe uses OMNeT++ for the network side of things and SUMO for the traffic side. This allows us to simulate complex platooning scenarios that include both vehicle coordination and network communication.

Plexe implements several models for automatic control of vehicles in an extension to SUMO. The models relevant for this thesis are the CACC and ACC models found in `sumo/src/microsim/cfmodels/MSCFModel_CC.cpp`, which implement longitudinal control based on the equations by Rajamani et al[4]. We refer to these as *P1* ("Plexe 1") and *PA* ("Plexe ACC") for the CACC and ACC respectively.

Plexe is critical for simulating advanced vehicular communication scenarios involving vehicle platooning. With it, we were able to evaluate how communication between closely coordinated vehicles impacts traffic efficiency, safety, and network performance.

3. **SUMO** :
SUMO (Simulation of Urban MObility) [8] is an open-source traffic simulator designed for high-performance, microscopic traffic simulations. Developed by the German Aerospace Center, it is a vital tool for simulating vehicle dynamics and traffic flow at an individual level, which is crucial for studying vehicular networks. One of the most useful features of SUMO is its ability to simulate vehicle-to-everything (V2X) communication. This allows us to explore how vehicles communicate with each other and with traffic infrastructure in real-time. The TraCI (Traffic Control Interface) API allows us to link SUMO with other simulators, like OMNeT++, enabling dynamic interaction between vehicle movement and network communication.

   SUMO handles the traffic simulation component of our experiments, providing the physical context for the vehicular networks we are studying. It allows us to accurately simulate the movement of vehicles and their communication patterns within the network, feeding real-time traffic data into OMNeT++ for further analysis.

4. **OMNeT++** :
OMNeT++ [9] is a component based C++ simulation library and framework with an Eclipse based IDE used for building network simulators. Domain specific functionality is provided by independently developed frameworks. It is widely used in academic research, particularly for building network simulations involving communication networks, distributed systems, and queuing models.

   OMNeT++ serves as the foundation for all our network simulations, coordinating the communication protocols and other simulations through its flexible framework. Its ability to integrate with traffic simulators like SUMO (via Veins) makes it essential for our vehicular network experiments.

   By combining these powerful simulation tools, we were able to build a comprehensive experimental setup that mimics real-world vehicular communication scenarios. OMNeT++ handles the core network simulation, SUMO takes care

of the traffic dynamics, Veins integrates the two, and Plexe allows us to simulate platooning scenarios. This multi-simulator approach is essential for getting realistic results that are crucial to the success of our thesis.

2.2 Types of jamming

Based upon the insights gained by the investigations conducted by Mateen et al.[6], we briefly explain the three main types of jamming they have looked into and the impact on the safety of the platooning application:

1. **Barrage Jamming**: Barrage jamming [10] is a basic form of jamming where a device transmits noise across all frequencies used by the target, continuously until the jamming stops. It doesn't take into account the specific signals or protocols being used by the target. According to game theory and information theory, when a jammer has no information about the target's signal, barrage jamming is usually the most effective approach [10].
2. **Destructive Interference**: Destructive interference [11] is a more advanced version of wireless jamming where, specially orchestrated signals designed to cancel the legitimate signal out or in other words, the overlap results in zero to low power(depending on the phase shift angle), are used to block information exchange. This attack is harder to carry out since the inverted signal has to arrive at the receiver with extremely accurate timing, phase, frequency and amplitude, but is still achievable especially with predictable forms of signals like the GPS although harder in case of V2V communications due to its varying nature. The end result of the interference is weakened or entirely canceled communication signals [11].
3. **Deceptive Jamming**: Deceptive Jamming [12] is when the attacker attempts to mimic legitimate signals and constantly bombard these to the receiver so as to induce a faulty state of the target system. For example, packets could be sent which appear legitimate but contain no meaningful information other than setting a system to receiving state, causing the system to not send out any data while having no useful data to work on [12].

In our thesis, we limit our scope to just Barrage Jamming, as designing fallbacks for each type of jamming requires a time-frame which is not feasible for a master thesis. According to [10], Barrage Jamming is a non-protocol-aware attack, which means no prior knowledge of the communication system is necessary and the entire spectrum of radio frequencies is equally jammed [6]. Destructive Interference and Deceptive Jamming are instead both considered to be protocol-aware jamming [6] [10], and therefore require in-depth knowledge of either the physical properties of the signal or the bit-level packet information within the communication system. These latter two jamming types being much harder to use both as attacks and to design fallbacks for played a major role in choosing Barrage Jamming as the main aim of our thesis.

2.3 Outcome classification

We use the five severity levels mentioned in [6] for classifying our experiments:

1. **Non-effective**: Outcomes where vehicles are unaffected by the attack and behave exactly the same as in the golden run.
2. **Negligible**: Outcomes where vehicles have a max. deceleration of 1.53 m/s^2.
3. **Benign**: Outcomes where vehicles have a maximum deceleration of 5 m/s^2.
4. **Severe_braking**: Outcomes where vehicles have a maximum deceleration *greater* than 5 m/s^2, but are still able to avoid collisions.
5. **Severe_collision**: Outcomes where vehicles collide.

2.4 Test bench setup

Figure 2.1
The Test Bench Computer

2.4.1 Hardware specifications

- **Processor**: AMD® Ryzen 7 5800x 8 Cores 16 Threads

- **Volatile Memory**: 94.2 GiB DDR4 @ 3200 MHz
- **Non Volatile Memory**: 25 TB SSD
- **Graphics Card**: Nvidia® RTX 3090 @ 2.1 GHz w 24GB GDDR5 @ 19.5 Ghz

2.4.2 Software versions

- **Operating System**: Ubuntu 20.04.4 LTS 64-bit @ Linux Kernel Version 5.15.0-124-generic
- **OMNeT++**: Version 5.6.2
- **SUMO**: Our own fork [13] based on Version 1.9.2
- **Plexe**: Version 3.0a2
- **ComFASE**: Our own fork [14] based on Version 2024-03-12

2.5 PA controller

Before we get to explaining the CACC, we need to have a basic overview of the Adaptive Cruise Controller (ACC), which we refer to henceforth as *PA* for "Plexe ACC", due to the fact that it is the first mentioned ACC in Plexe by Segata et al., see equation 4 in [3] and chapter 6 in [4]. This controller is used by the lead vehicle at all times, but we will additionally use it for other vehicles as part of our proposed fallback models in this thesis.

2.5.1 ACC control equation

$$\ddot{x}_{i\_des} = -\frac{1}{T}(\dot{\epsilon}_i + \lambda\delta_i) \tag{2.1}$$

$$\begin{aligned} &where \\ &\quad \delta_i = x_i - x_{i-1} + l_{i-1} + T\dot{x}_i \\ &\quad and \\ &\quad \dot{\epsilon}_i = \dot{x}_i - \dot{x}_{i-1} \end{aligned}$$

In the above equation, i is the controlled vehicle, T is the time headway in seconds, $\dot{\epsilon}_i$ represents the speed difference between the current vehicle and the preceding one, which helps in determining whether to accelerate or decelerate, δ_i is the difference between the current distance to the preceding vehicle and the desired following distance and λ is a design parameter always greater than zero [3]. We get the desired acceleration for vehicle i $\ddot{x}_{i\_des}$ by plugging in the values for the parameters into the above equation. The ACC runs in conjunction with the regular Cruise Controller (CC) and the desired acceleration is always the minimum of the CC and ACC as seen in Eq. 2.2. $\ddot{X}$ represents the desired acceleration, $\ddot{x}_{\mathrm{CC}}$ represents the acceleration mandated by the CC and $\ddot{x}_{\mathrm{ACC}}$ is the acceleration mandated by the ACC.

$$\ddot{X}des = \min(\ddot{x}_{\mathrm{CC}}, \ddot{x}_{\mathrm{ACC}}) \tag{2.2}$$

Up until now, we have only mentioned the CC and ACC which, in the context of a platoon, are more suitable for leader vehicles [3] but now, we are going to discuss more in depth about controller models suitable for follower vehicles.

2.6 The P1 controller - Baseline

The primary controller that has been analyzed in this thesis is the one we call P1, short for the first CACC described by Segata et al. [3]. This is a CACC algorithm that tries to maintain a constant spacing between cars in a platoon and has longitudinal control inspired by Rajamani et al. [4].

2.6.1 P1 controller equation

$$\begin{aligned} &\ddot{x}_{i\_des} = \alpha_1\ddot{x}_{i-1} + \alpha_2\ddot{x}_0 + \alpha_3\dot{\varepsilon}_i + \alpha_4(\dot{x}_i - \dot{x}_0) + \alpha_5\varepsilon_i \\ &\text{where,} \\ &\varepsilon_i = x_i - x_{i-1} + l_{i-1} + \mathrm{gap}_{\mathrm{des}} \\ &\dot{\varepsilon}_i = \dot{x}_i - \dot{x}_{i-1} \end{aligned} \tag{2.3}$$

$\ddot{x}_{i\_des}$ is the desired acceleration. $\ddot{x}_0$ and $\dot{x}_0$ are the acceleration and speed of the leader respectively, while $\ddot{x}_{i-1}$ is the acceleration of the preceding vehicle. The distance error ε_i is based on a constant desired distance [3], while $\mathrm{gap}_{\mathrm{des}}$ is the desired spacing gap between the vehicles. l_{i-1} is the length of the preceding vehicle, x_i is the position of the vehicle i and finally x_{i-1} is the position of the preceding vehicle. Similar to the ACC, the Desired Acceleration is given by the minimum of the mandated acceleration of the CC and CACC, $\ddot{X}des = \min(\ddot{x}_{\mathrm{CC}}, \ddot{x}_{\mathrm{CACC}})$

The alpha terms in Equation 2.3 are as follows,

$$\begin{aligned} &\alpha_1 = 1 - C_1, \alpha_2 = C_1, \alpha_5 = -\omega_n^2 \\ &\alpha_3 = -(2\xi - C_1(\xi + \sqrt{\xi^2 - 1}))\omega_n \\ &\alpha_4 = -C_1(\xi + \sqrt{\xi^2 - 1})\omega_n \end{aligned} \tag{2.4}$$

From [3], we see that C_1 is a weighting factor between the accelerations of the leader and the preceding vehicle (default set to 0.5), ξ is the damping ratio (default set to 1), and ω_n is the bandwidth of the controller (default set to 0.2 Hz) and the defaults are taken from [15].

The first four variables in Equation 2.3, $\ddot{x}_{i-1}$, $\ddot{x}_0$, $\dot{\varepsilon}_i$ and $(\dot{x}_i - \dot{x}_0)$ were found to depend mainly on the information received from the V2V network communication (aside from the constant alpha factors). Upon analysis, it was further discovered that the first two variables $\ddot{x}_{i-1}, \ddot{x}_0$ simply repeat the values received prior to losing communication until a new `PlatooningBeacon` arrives, see Sec. 2.7. The subsequent

two variables $\ddot{x}_0$ and $\dot{\varepsilon}_i$ behaved similarly, but changed slightly based on missing data prediction based on interpolation [16]. These predictions are on by default but can be turned off by setting `vars->usePrediction` to `false`. The final variable ε_i solely depends on the radar and is explained in Section 2.8.

2.6.2 P1 controller code implementation in SUMO

```
double
MSCFModel_CC::_cacc(const MSVehicle* veh, [...]) const {
    CC_VehicleVariables* vars = veh->getCarFollowVariables();
    //compute epsilon, i.e., the desired distance error
    double epsilon = -gap2pred + spacing;
    //compute epsilon_dot, i.e., the desired speed error
    double epsilon_dot = egoSpeed - predSpeed;

    return vars->caccAlpha1 * predAcceleration
         + vars->caccAlpha2 * leaderAcceleration
         + vars->caccAlpha3 * epsilon_dot
         + vars->caccAlpha4 * (egoSpeed - leaderSpeed)
         + vars->caccAlpha5 * epsilon;
}
```

Source code in function `_cacc` in the SUMO file `MSCFModel_CC.cppSource` showing the P1 controller code.

Listing 1

In the above code, see listing 1, we get a clearer picture of the parts of the equation with respect to the implemented code in SUMO. Each of the `caccAlpha` terms correspond to the alpha equations in Equation 2.4. The first term `predAcceleration`, uses the predecessor vehicle's acceleration. The second term `leaderAcceleration` uses the received leader acceleration. The third term `epsilon_dot` calculates the desired speed error. The fourth term `(egoSpeed - leaderSpeed)` uses the difference in speed between the vehicle and the leader. The fifth and final term `epsilon` is the weight-adjusted desired distance error.

2.7 Network communication

The vehicles of the platoon communicate over the V2V network by regularly broadcasting `PlatooningBeacon` messages to all vehicles in range. This is done every 0.1 s by each vehicle in order, looping around the platoon in a forward direction. The information contained in these packets is shown in listing 2.

```
packet PlatooningBeacon {
    //id of the originator
    int vehicleId = 0;
    double controllerAcceleration = 0;
    double acceleration = 0;
    double speed = 0;
    double positionX = 0;
    double positionY = 0;
    double time = 0;
    int sequenceNumber = 0;
    double length = 0;
    double speedX = 0;
    double speedY = 0;
    double angle = 0;
}
```

Message Structure of the V2V communication as seen in plexe/src/plexe/messages/PlatooningBeacon.msg

Listing 2

Upon receiving messages in the form of wireless signals, vehicles attempt to decode the signals, checking for sufficient signal strength and a lack of errors. When successful, the `BaseApp::onPlatoonBeacon` function in the application layer, see listing 3, will eventually handle the `PlatooningBeacon`. Otherwise, the information/noise is discarded. We can thus know there is some form of communication loss or delay if nothing reaches this function for some time.

```
void BaseApp::onPlatoonBeacon(const PlatooningBeacon* pb)
{
    if (positionHelper->isInSamePlatoon(pb->getVehicleId())) {
        // if the message comes from the leader
        if (pb->getVehicleId() == positionHelper->getLeaderId()) {
            plexeTraciVehicle->setLeaderVehicleData([pb data ...]);
        }
        // if the message comes from the vehicle in front
        if (pb->getVehicleId() == positionHelper->getFrontId()) {
            plexeTraciVehicle->setFrontVehicleData([pb data ...]);
        }

        // send data about every vehicle to the CACC.
        struct VEHICLE_DATA vehicleData;
        [... save general vehicle data - omitted ...]

        // send information to CACC
        plexeTraciVehicle->setVehicleData(&vehicleData);
    }
    delete pb;
}
```

Function `onPlatoonBeacon` located in plexe/src/plexe/apps/BaseApp.cc

Listing 3

Inside the `onPlatoonBeacon` function the message is deconstructed and handled differently depending on if it is coming from the leader, front or other vehicle. `setLeaderVehicleData` and `setFrontVehicleData` are called with acceleration, speed, position and time received from the leader and front vehicle respectively. The time variable is a beacon timestamp and can be used to determine if communication has been lost or not. Information from any other vehicle is sent to the CACC controller using `setVehicleData`, but is not used by the P1 and PA controllers.

2.8 Vehicle radar

Every vehicle is equipped with a radar which is used to measure the distance to the vehicle in front of it (called the *predecessor* vehicle). Looking at Equation 2.3, we notice that out of the 5 different terms in the equation and their respective constant multipliers, the last term $\omega_n^2 \epsilon_i$ is completely unaffected by jamming signals due to the fact that `double epsilon = -gap2pred + spacing;` where `gap2pred` clearly comes from the radar function `getRadarMeasurements` as seen in Fig. 2.2.

Similarly, the variable `spacing` is read from a collection of variables `vars` representing the current state of each vehicle, see Fig. 2.3, but the parameter `caccSpacing` itself is set as a part of the configuration file `"omnetpp.ini"`, see Fig. 2.5, and has no

```
double MSCFModel_CC::freeSpeed(const MSVehicle* const veh, double speed, double seen, double maxSpeed,
    CC_VehicleVariables* vars = (CC_VehicleVariables*)veh->getCarFollowVariables();
    if (vars->activeController != Plexe::DRIVER) {
        double gap2pred, relSpeed;
        getRadarMeasurements(veh, gap2pred, relSpeed);
        if (gap2pred == -1) {
            gap2pred = std::numeric_limits<double>().max();
        }
        return _v(veh, gap2pred, speed, speed + relSpeed); // predSpeed = speed + relSpeed
    } else {
        return MSCFModel::freeSpeed(veh, speed, seen, maxSpeed, onInsertion);
    }
}
```

Figure 2.2
Source code in SUMO showing `gap2pred` variable being fetched using the function `getRadarMeasurements`

```
double
MSCFModel_CC::getCACCConstantSpacing(const MSVehicle* veh) const {
    CC_VehicleVariables* vars = (CC_VehicleVariables*) veh->getCarFollowVariables();
    return vars->caccSpacing;
}
```

Figure 2.3
Source code in SUMO showing the variable `spacing` being fetched using the function `getCarFollowVariables`

discernible intervention from the V2V communication system. While `relSpeed` is also fetched from the radar and used to calculate `predSpeed = speed + relSpeed`, predSpeed is then overwritten in other parts of the code with the data fetched from the wireless communication as seen in Fig. 2.4.

```
//overwrite pred speed using data obtained through wireless communication
predSpeed = vars->frontSpeed;
leaderSpeed = vars->leaderSpeed;
if (vars->usePrediction) {
    predSpeed += (currentTime - vars->frontDataReadTime) * vars->frontAcceleration;
    leaderSpeed += (currentTime - vars->leaderDataReadTime) * vars->leaderAcceleration;
}
```

Figure 2.4
`predSpeed` overwritten by V2V Communication in the function `MSCFModel_CC::_v` in the file sumo/src/microsim/cfmodels/MSCFModel_CC.cpp

Looking at the multiplication factors for the parameters set in Equation 2.3 and comparing them against the values we found in Figure 2.5, we determined that the radar part of the equation, i.e. the fifth term alone is extremely inefficient in holding the vehicles in formation during a jamming attack affecting every other part of the equation. This gave us a significant insight into the reason behind the consistent crashing of vehicles into each other in the paper written by Mateen et al. [5]. Ploeg et al. [17] explain why the radar alone is not sufficient to maintain string stability especially at short spacing distances. Since our fallbacks are mainly dependent on the radar, the need to have a larger spacing became evident in our testing phases, as we describe in Chapter 5.

```
##########################################################
#              Scenario common parameters                #
##########################################################

#controller and engine related parameters
*.node[*].scenario.caccC1 = 0.5
*.node[*].scenario.caccXi = 1
*.node[*].scenario.caccOmegaN = 0.2 Hz # originally `0.2 Hz`
*.node[*].scenario.caccSpacing = 5 m # originally `5 m`
*.node[*].scenario.engineTau = 0.5 s
*.node[*].scenario.ploegH = ${ploegH = 0.5}s
*.node[*].scenario.ploegKp = 0.2
*.node[*].scenario.ploegKd = 0.7
*.node[*].scenario.useRealisticEngine = false
#via wireless send acceleration computed by the controller, not the actual one
*.node[*].scenario.useControllerAcceleration = true
*.node[*].scenario.nLanes = ${nLanes}
```

Figure 2.5
Engine and Controller Parameters found in omnetpp.ini

3

Proposed Fallback Mechanisms

In this chapter we propose three alternative controller algorithms [13] to improve the resilience of the P1 controller algorithm in cases of communication malfunction. These fallbacks can be turned on as per a switchover criterion; in this thesis we have tested activating them using different durations of communication loss and determined suitable durations to be further analyzed using a broader set of noise values. We start the chapter by first explaining the fallback mechanism used in all of our alternatives, and then move on to describing the specific alternatives one-by-one. We end by additionally proposing a timer extension applicable to all three models as well as three model variants for distinguishing models using different parameters.

3.1 Controller model implementation

All of our proposed models make use of a fallback mechanism and assume that the timestamp has not been tampered with. This mechanism works by regularly comparing the current time with the timestamp of the last received network packet from the front vehicle. If the difference is low, meaning the packet was received recently, it assumes communication is still functioning and uses the base P1 model. If the difference is high, communication is assumed to have been lost and the vehicle switches to using one of the fallbacks. Additionally, some models make use of several fallback steps for different durations of communication loss, and some have a 1 second minimum time for a fallback to stay on. The complete code used for all of the models is shown in listing 16 as part of the Appendix.

To switch between the different models there are four parameters that can be set, as shown in listing 4. The two first parameters can only be `true` or `false`, while the remaining parameters in theory can be anything ≥ 0 but have only been tested for a few values. The way the parameters combine determine the specific model used.

```
// Set parameters to use here
const bool   useDegraded = true;
const bool   useFallback = true;
const double fbMinOnTime = 1.0;
const double fbActivTime = 2.0;
```

Listing 4: Parameters to set for choosing what fallback model to use.

3.2 Model 1: P1 controller

Model 1 is the baseline P1 controller we have used in our comparison and analysis against the models with our fallbacks. It is this model that we have based our fallback models upon by building on top of it. For more in-depth information about this controller, see Sec. 2.6 of the Background chapter.

3.3 Model 2: Fallback to degraded P1

One possible way to improve the resilience of the base model is to extend it with a fallback to a degraded CACC, which is what our Model 2 does. In this model, after a period of communication loss the CACC switches over to getting its `predSpeed` from the radar rather than from communication. Additionally, it increases the desired spacing between vehicles tenfold. These two simple changes work in combination to strengthen the model's resilience against communication disturbances.

Looking back at the original P1 CACC equation, see Eq. 2.3 in the previous chapter, these two changes result in the $\dot{\epsilon}_i$ of the third term being changed to only use information acquired from the radar, and the L used to calculate ϵ_i of the fourth term being multiplied by 10. In the corresponding code, this equates to giving `predSpeed = radarPredSpeed` and `spacing = caccSpacing*10` as input to the `_cacc` function rather than `predSpeed` and `caccSpacing`.

3.4 Model 3: Fallback to PA

Another option is to switch to the normal ACC controller when communication is lost, which has the advantage of reusing already existing code. Here we have simply copied over the ACC controller code to the fallback handler with only a small modification of replacing `predSpeed` with `radarPredSpeed`, as shown in listing 5. In reality, this modification is not really even a real one, since it is just there to ensure that the predecessor speed will be acquired from the radar and not the network - something which already is the case when running ACC normally. The modification only so happens to be necessary here because of how the CACC normally overwrites the `predSpeed` variable with data received through V2V communication. In the case of running ACC normally (i.e. not as a fallback), this overwrite never happens.

```
double ccAcceleration = _cc(veh, egoSpeed, vars->ccDesiredSpeed);
double accAcceleration = _acc(veh, egoSpeed, radarPredSpeed, [...]);
if (gap2pred > 250 || ccAcceleration < accAcceleration) {
    controllerAcceleration = ccAcceleration;
} else {
    controllerAcceleration = accAcceleration;
}
```

Listing 5: ACC fallback implementation

3.5 Model 4: Degraded fallback multimodel

Finally, we have also developed a model combining the degraded CACC fallback with the ACC fallback. This model works by first switching to the degraded CACC after an initial, short communication loss of 0.1 s - followed by falling back to the ACC

after a longer `fbActivTime` second duration of continued communication loss. For clarity, listing 6 shows pseudocode for how these two models are combined into one.

```
if msgLostTime < 0.1 {
    useCACC()
} else if msgLostTime < fbActivTime {
    useDegraded()
} else {
    useACC()
}
```

Listing 6: Pseudocode implementation of the degraded fallback multimodel.

3.6 Extension: minimum fallback on-time

In addition to the three models mentioned above, we propose a simple modification that can be made to all of them and potentially improve their resilience to certain types of attacks. This modification is the inclusion of a one second timer that needs to run out before any model switches back to the CACC. Normally, as soon as communication resumes between vehicles (i.e. a new message is received) all three models return to using the normal CACC. With the timer they will instead always use their respective fallbacks for a minimum of 1 second before switching back. This prevents intermittent attacks from making the models switch quickly back and forth between fallback and no fallback.

3.7 Model variants

Since our proposed fallback models can use different parameters and extensions we have split them up into multiple variants distinguished by postfix letters:

1. Variant A: Default parameters. Labeled with a postfix “a” (e.g. Model 3a).
2. Variant B: Minimum fallback on-time extension. Labeled with a postfix “b”.
3. Variant C: Lowered time to activate fallback. Labeled with a postfix “c”.

4

Experimental Setup

This chapter describes and explains in detail the methods used for conducting the experiments performed on our proposed fallback models. It gives an account of the setup, specific parameters used and all other details needed for replication.

Prior to doing any experiments, the environment had to be set up. This was done by first downloading and installing all of the required software – see [Ch. 2, Sec. 2.1] – as instructed by the available documentation. Next, we modified the SUMO and ComFASE source code in several iterations as part of implementing the models and automating the attack campaign process. Our final implementations are available as forks on GitHub, which are listed in [Ch. 2, Sec. 2.4.2] Software Versions and should be compiled from source as the last step of the setup process.

4.1 Methodology

Experiments testing and measuring the resilience of the P1 controller were performed for each of the fallbacks, as well as once without any fallbacks. This was done by using ComFASE to inject barrage jamming attacks on the V2V network during simulation and then analyzing the braking and collision data collected. First of all, a few initial configuration steps were done once before any model was tested:

1. In Plexe, the configuration file `examples/platooning_comfase/omnetpp.ini` was modified with the correct settings to use. These settings remained the same for every experiment that was later conducted.
2. A single *golden run* was performed on the P1 base model, i.e. Model 1, for later comparison; meaning, a run under ideal conditions without any barrage jamming. Data collected from this run was saved in a folder for later use.

After that, the following steps were performed (in order) for every model:

1. **SUMO code preparation and compilation.** The model parameters located in the `MSCFModel_CC.cpp` file were set to the values corresponding to the model to be tested, see Sec. 4.3. SUMO was then recompiled by running `./compile.sh` in the SUMO root in order to use the updated code.
2. **Attack campaign configuration.** The attack parameters were specified in the `configure_campaign.xml` file located in the same folder. The attack type `Barrage_jamming` was set to `true` in all cases, while the specific barrage jamming attack parameters were set to one of two configurations depending on if a variable or maximum noise attack was desired, see Sec. 4.2. Moreover, `comfase_run.py` (once again located in the same folder) was modified to set the attack output data to be stored in a sub-folder designated for the model.
3. **Attack campaign execution.** Finally, the attack campaign was executed for the selected model by running `comfase_run.py`. Once finished, all data

collected and graphs generated by the script could be found in the previously specified sub-folder within the `ComFASE_data` folder.

4.2 Attack parameters

In order to execute the attack campaigns, attack parameters for the barrage jamming first had to be set. These parameters included start times, durations and attack values. The latter is what we call the attack *noise* and warrants a short explanation before we go further. Noise is a value between 0.0 and 1.0 used for attack simulation and represents the signal noise that would be in a real barrage jamming attack. It is measured in milliwatts (mW) and depending on its value it will disturb V2V communication to different extents, where higher noise means higher disturbances.

Two sets of configurations were used for the attack campaigns – one for maximum noise attacks and one for variable noise attacks. All campaigns used 13 attack start times (range: $[17.0\ s, 21.9\ s)$) and 11 attack durations (range: $[1\ s, 12\ s)$), resulting in $13 * 11 = 143$ experiments per model and attack noise value. For the maximum noise configuration, only the maximum attack noise value was used and hence campaigns using that configuration only have 143 experiments. For campaigns using the variable noise configuration, the total experiments were instead $143 * 25 = 3575$ per model since 25 attack values in the range $[0.04, 1.0]$ were used.

Setting of the configuration variables is done in the file `configure_campaign.xml` as described in section 4.1 Methodology. The values to set in general and for the two different configurations respectively are shown below.

configure_campaign.xml - all attack scenarios

```
<!-- ################ Attack Model Selection ################ -->
<Attack_type
Delay="false"
DoS="false"
Destructive_interference="false"
Barrage_jamming="true"
Deceptive_jamming="false"
/>
```

configure_campaign.xml - maximum noise attack scenario

```
<!-- ########### Barrage jamming Attack Parameters ########### -->
<Barrage_jamming
attackInitiationStartTime="17.0" attackInitiationEndTime="21.9"
attackInitiationTimeStep="0.4" attackStartValue="1.0"
attackEndValue="1.01" attackValueStep="0.04" attackMinDuration="1"
attackMaxDuration="12" attackDurationStep="1"
attackOnSender="true" attackOnReceiver="true"
/>
```

configure_campaign.xml - variable noise attack scenario

```
<!-- ########### Barrage jamming Attack Parameters ########### -->
<Barrage_jamming
attackInitiationStartTime="17.0" attackInitiationEndTime="21.9"
attackInitiationTimeStep="0.4" attackStartValue="0.04"
attackEndValue="1.01" attackValueStep="0.04" attackMinDuration="1"
attackMaxDuration="12" attackDurationStep="1"
attackOnSender="true" attackOnReceiver="true"
/>
```

4.3 Model parameters

Below we list the parameters used for each of the fallback models. The parameters modified in `omnetpp.ini` existed prior to our implementation but have been changed. The other parameters specific to each model have been added by us to `MSCFModel_CC.cpp` as part of the implementation.

4.3.1 Applicable to all models

Modified parameters used for all models are listed below.

omnetpp.ini

```
sim-time-limit = 45 s

[...]

# Changed 3rd item: accHeadwayTime=0.2s for CACC->ACC fallback
**.headway = ${headway = 0.3, 1.2, 0.2, 0.1, 0.1, 0.1 ! controller}s
```

4.3.2 Model 1: P1 base model

```
const bool   useDegraded = false;
const bool   useFallbACC = false;
const double fbMinOnTime = 0.0;
const double fbActivTime = 2.0;
```

Listing 7: Parameter values used for Model 1.

4.3.3 Model 2a: Fallback to degraded P1

```
const bool   useDegraded = true;
const bool   useFallbACC = false;
const double fbMinOnTime = 0.0;
const double fbActivTime = 2.0;
```

Listing 8: Parameter values used for Model 2a.

4.3.4 Model 2b: Fallback to degraded P1 + timer

```
const bool   useDegraded = true;
const bool   useFallbACC = false;
const double fbMinOnTime = 1.0;
const double fbActivTime = 2.0;
```

Listing 9: Parameter values used for Model 2b.

4.3.5 Model 3a: Fallback to PA (2 s)

```
const bool   useDegraded = false;
const bool   useFallbACC = true;
const double fbMinOnTime = 0.0;
const double fbActivTime = 2.0;
```

Listing 10: Parameter values used for Model 3a.

4.3.6 Model 3b: Fallback to PA (2 s) + timer

```
const bool   useDegraded = false;
const bool   useFallbACC = true;
const double fbMinOnTime = 1.0;
const double fbActivTime = 2.0;
```

Listing 11: Parameter values used for Model 3b.

4.3.7 Model 3c: Fallback to PA (1 s)

```
const bool   useDegraded = false;
const bool   useFallbACC = true;
const double fbMinOnTime = 0.0;
const double fbActivTime = 1.0;
```

Listing 12: Parameter values used for Model 3c.

4.3.8 Model 4a: Degraded fallback multimodel (2 s)

```
const bool   useDegraded = true;
const bool   useFallbACC = true;
const double fbMinOnTime = 0.0;
const double fbActivTime = 2.0;
```

Listing 13: Parameter values used for Model 4a.

4.3.9 Model 4b: Degraded fallback multimodel (2 s) + timer

```
const bool   useDegraded = true;
const bool   useFallbACC = true;
const double fbMinOnTime = 1.0;
const double fbActivTime = 2.0;
```

Listing 14: Parameter values used for Model 4b.

4.3.10 Model 4c: Degraded fallback multimodel (1 s)

```
const bool   useDegraded = true;
const bool   useFallbACC = true;
const double fbMinOnTime = 0.0;
const double fbActivTime = 1.0;
```

Listing 15: Parameter values used for Model 4c.

4.4 Miscellaneous experiments

Beyond the experiments described above, we additionally conducted some miscellaneous experiments for improvement and analysis of our proposed models. These are briefly accounted for in the two sections below.

4.4.1 ACC headway time

Model 3 and 4 both make use of an ACC fallback of which an `accHeadwayTime` parameter needs to be set for. To find out an optimal value for this parameter, 143 experiments using the maximum noise configuration were conducted on Model 3c for each of the following headway times: 0.1 s, 0.15 s, 0.2 s and 0.3 s. The headway times were chosen iteratively (trial and error) in order to find the threshold value resulting in no collisions and no severe braking.

4.4.2 Message loss per noise level

Since we conducted our experiments with varying levels of barrage jamming noise, it was of interest to know how much message loss each of these levels corresponded to. Thus, a "message loss per noise level" analysis was conducted by barrage jamming from 0 to 60 seconds, resulting in a total of 7065 messages being sent per noise level. The noise levels tested were the following (all in mW): 0.2, 0.4, 0.6, 0.62, 0.63, 0.66, 0.8, 1.0 and applied to both the transmitter and receivers. They were chosen iteratively in order to find the specific threshold value for when communication stopped working entirely (i.e. complete message loss) due to the fact that our fallbacks do not work unless there is a complete message loss for a specified duration.

5

Experimental Results

In this chapter we present the results of evaluating the three fallbacks proposed in this thesis. We start by presenting the performance under attack of the default P1 model without any modifications. Next, we go through each fallback one-by-one in their own sections and show how they performed in comparison to the base model. After that, we also shortly present some miscellaneous results that, while not specific to any fallback, are still of relevance. Finally, we end the chapter with a summary of the results and a comparison of the fallback alternatives.

5.1 Model 1: P1 base model

Starting with the unmodified base model, we have found that it is highly susceptible to attacks, confirming previous findings from Mateen et al. [5][6]. Attacks conducted with maximum noise resulted in collisions (`Severe_collision`) occurring already at 2 seconds attack duration and increasing in frequency until 4 seconds where it plateaued at a maximum frequency of 53.8%. A graph demonstrating these results, with 143 experiments categorized by their severity levels, is shown in figure 5.1.

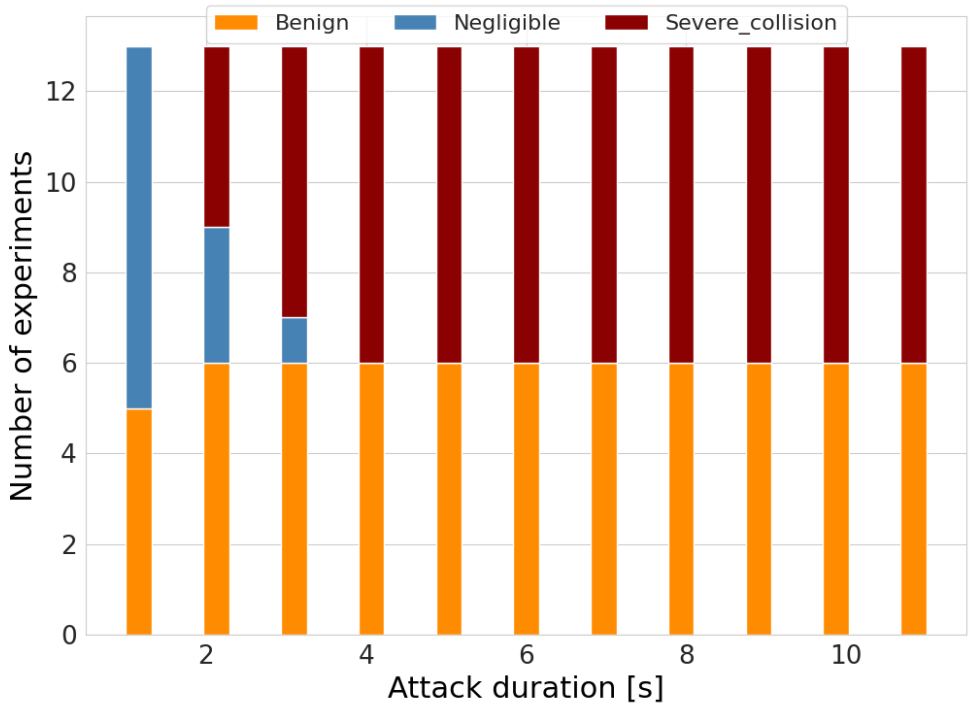


Figure 5.1: Outcome distribution for *maximum* noise experiments on Model 1.

Similar results were found for the more comprehensive set of 3575 experiments with variable noise levels, as shown in figure 5.2. One notable difference though is the

presence of a few `Non_effective` experiments, which after further investigation were discovered to be limited to noise levels at 0.04 and below, as shown in figure 5.3.

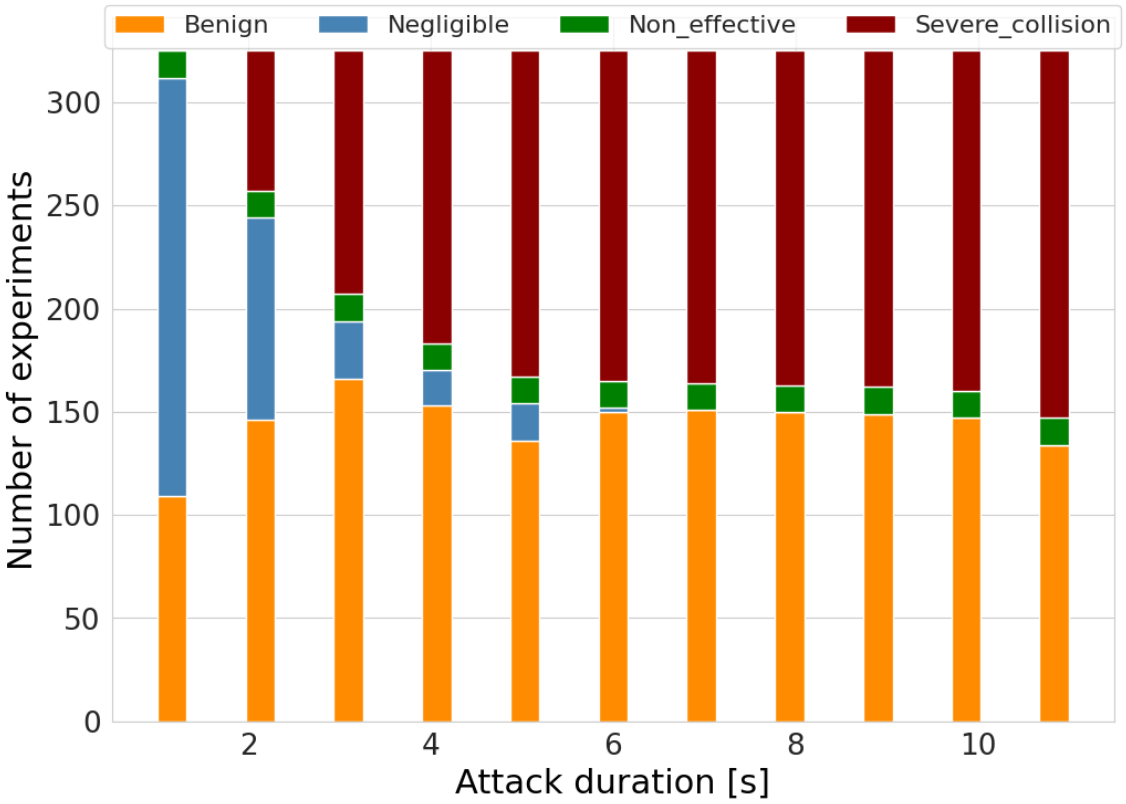


Figure 5.2: Outcome distribution for *variable* noise experiments on Model 1.

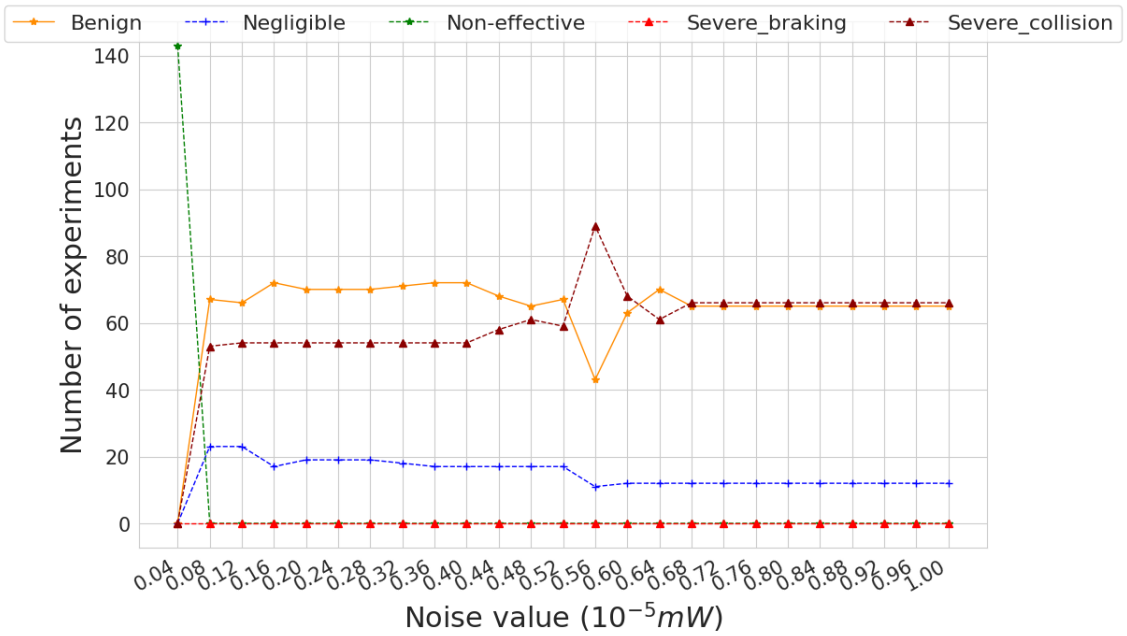


Figure 5.3: Outcome distribution as a function of noise value for Model 1.

5.2 Model 2a: Fallback to degraded P1

The first of our proposed fallback mechanisms shows a modest yet significant improvement to controller resilience. While collisions are still present even in the max noise attack scenario, they have been reduced by 72.7% in comparison to the base model and been completely eliminated for attack durations of 2 and 4-6 seconds. These results can be seen in figure 5.4.

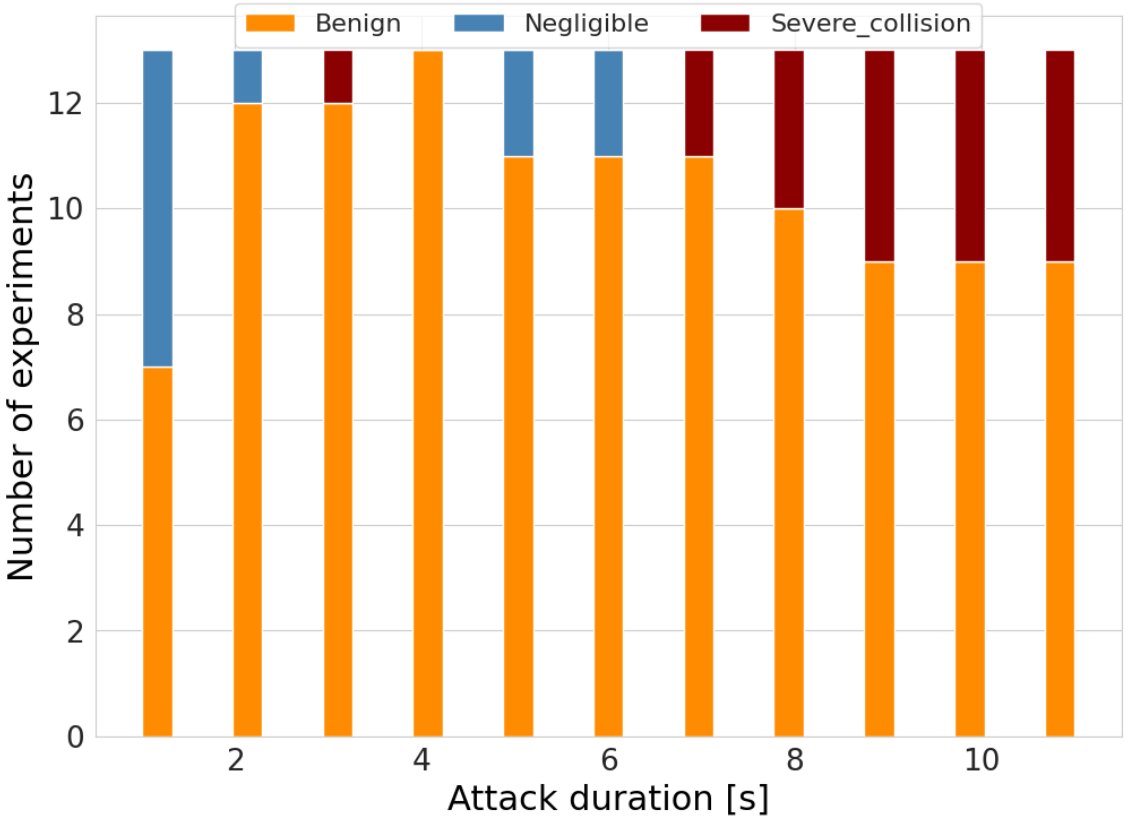


Figure 5.4: Outcome distribution for *maximum* noise experiments on Model 2a.

Looking at the results for the variable noise attack scenario shown in figure 5.5, we can see a similar but smaller improvement. Collisions are here present for the same attack durations as for the base model, but in smaller quantities. In total, 889 collisions were recorded for this model, which can be compared to the notably higher 1475 registered for the base model.

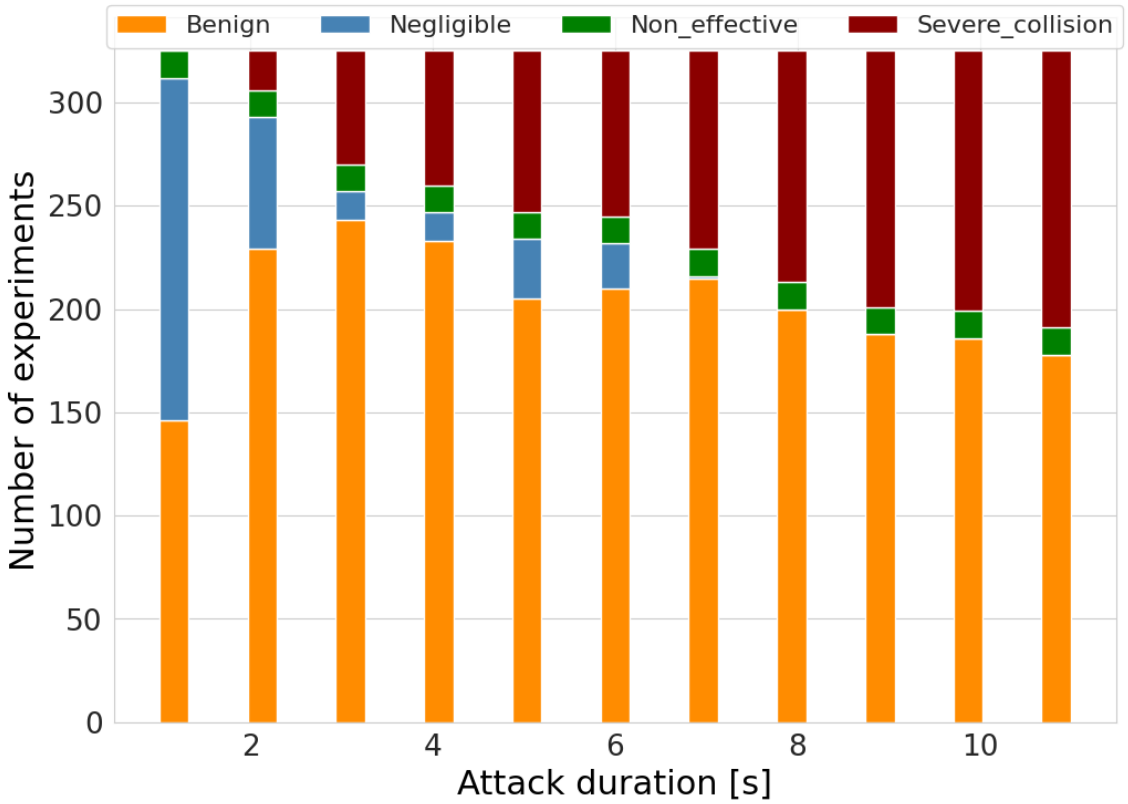


Figure 5.5: Outcome distribution for *variable* noise experiments on Model 2a.

5.3 Model 2b: Fallback to degraded P1 + timer

The b-variant of Model 2 performed similarly but slightly differently than the a-variant presented in the previous section, as can be seen in table 5.1. In the maximum noise attack scenario, it performed marginally worse with 1 more collision, and in the variable noise attack scenario it performed slightly better with 36 fewer collisions.

Table 5.1: Outcome classification for Model 2b in the different noise scenarios.

Scenario	**Non-effective**	**Negligible**	**Benign**	**Severe**		**Total**
				Braking	**Collision**	
Maximum noise	0	11	113	0	19	143
Variable noise	143	288	2291	0	853	3575

5.4 Model 3a: Fallback to PA (2 s)

Our second proposed fallback model showed varying results depending on the variant and attack scenario, but it consistently performed better than the base model. The variant presented in this section, Variant A, had only 2 collisions in the maximum noise scenario and therefore outperformed Model 2 – as can be seen in figure 5.6. However, for the variable noise attack campaign this model ended up having a slightly

lower resilience than Model 2, coming in at 936 collisions compared to 889 for Model 2a and 853 for Model 2b. The results for this latter scenario can be seen in figure 5.7.

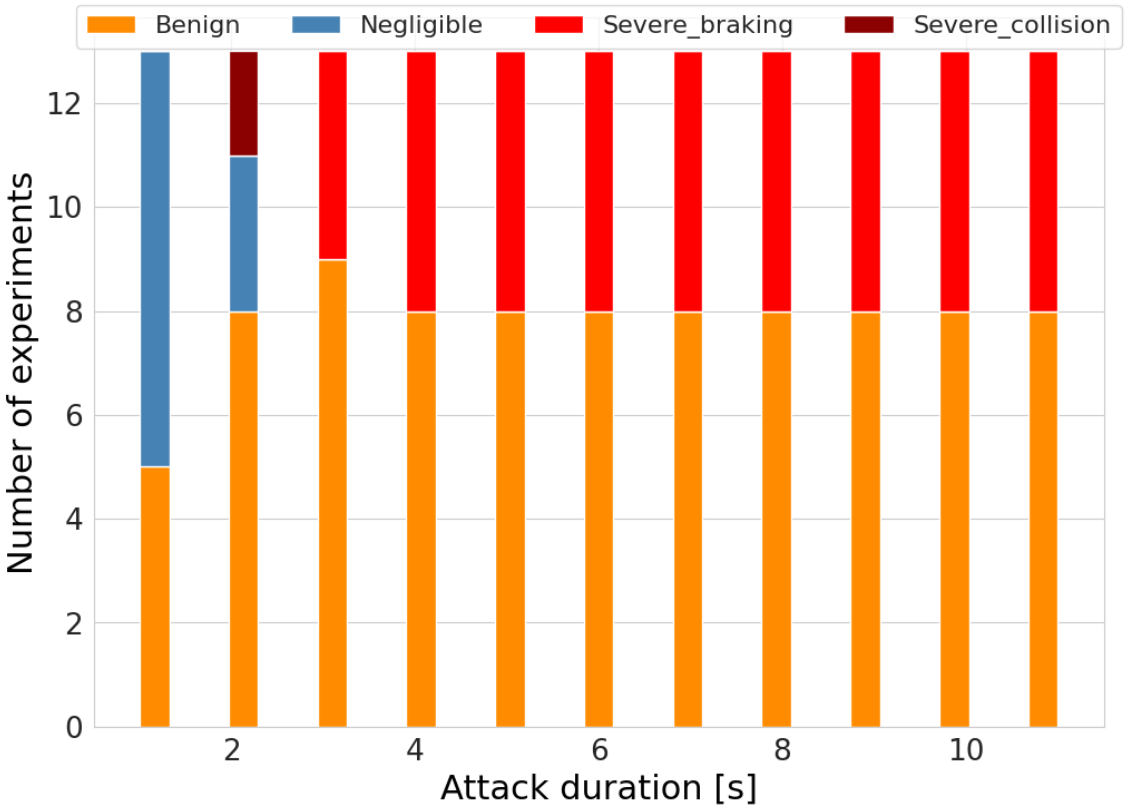


Figure 5.6: Outcome distribution for *maximum* noise experiments on Model 3a.

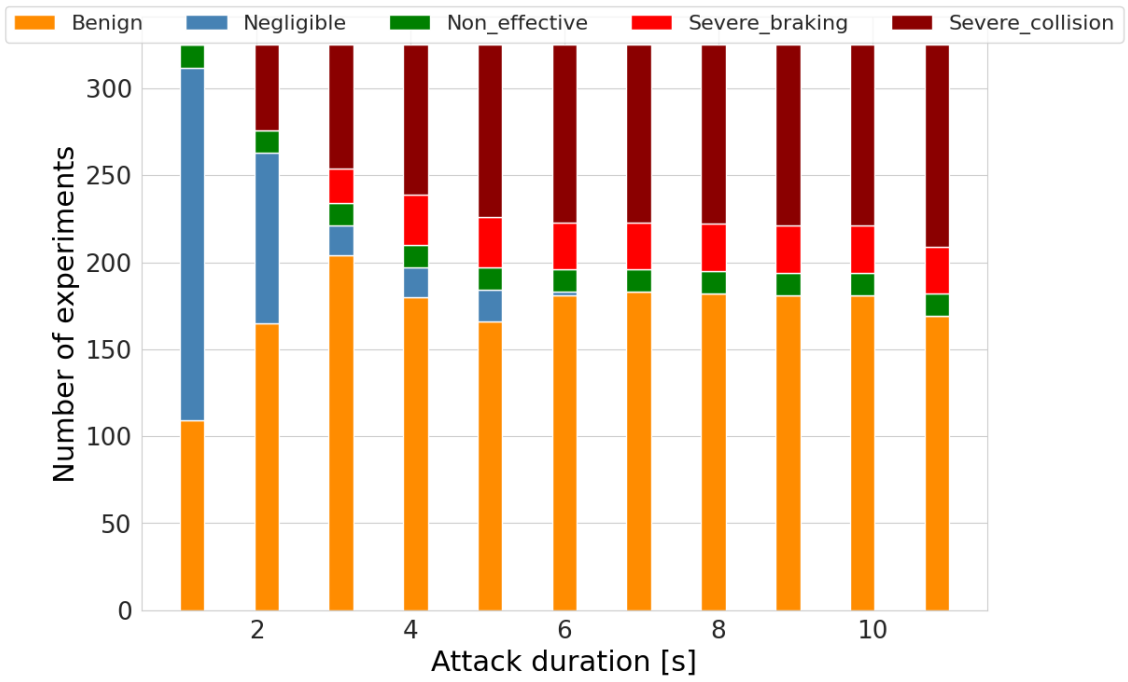


Figure 5.7: Outcome distribution for *variable* noise experiments on Model 3a.

5.5 Model 3b: Fallback to PA (2 s) + timer

The b-variant of Model 3 performed slightly worse than the a-variant – particularly in the maximum noise attack scenario – as can be seen in table 5.2. It otherwise follows the general pattern we have seen of there being a much higher share of collisions in the variable noise scenario compared to the maximum noise scenario. In this case, we see that there are 6.99% versus 26.29% collisions in the maximum and variable noise scenarios respectively. This compares to 1.40% and 26.18% for the a-variant.

Table 5.2: Outcome classification for Model 3b in the different noise scenarios.

Scenario	**Non-effective**	**Negligible**	**Benign**	**Severe**		**Total**
				Braking	**Collision**	
Maximum noise	0	8	87	38	10	143
Variable noise	143	325	1873	294	940	3575

5.6 Model 3c: Fallback to PA (1 s)

Variant C of our second fallback model – which uses a shorter 1 s switch-to-fallback time – was measured to successfully be able to neutralize all attacks with noise levels 0.68 and above. As can be seen in the results for the maximum noise attacks shown in figure 5.8, no collisions occurred nor any severe braking. Noise levels between 0.08 mW and 0.64 mW however still caused some collisions, see figures 5.9-5.10. Noise levels 0.04 and below were ineffective as attacks, but this was the case for all models.

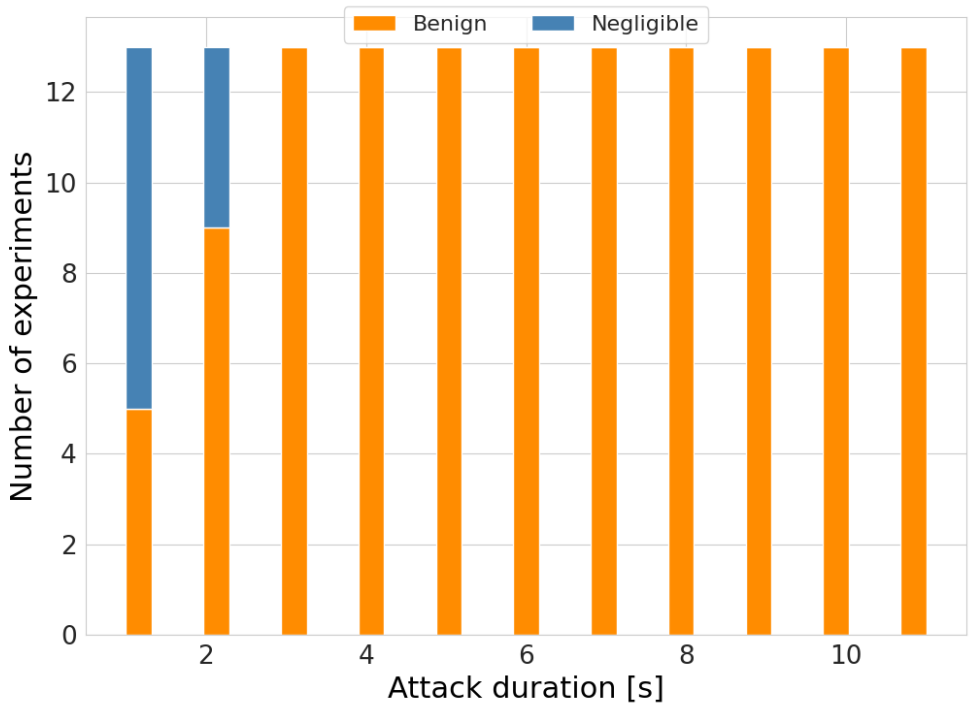


Figure 5.8: Outcome distribution for *maximum* noise experiments on Model 3c.

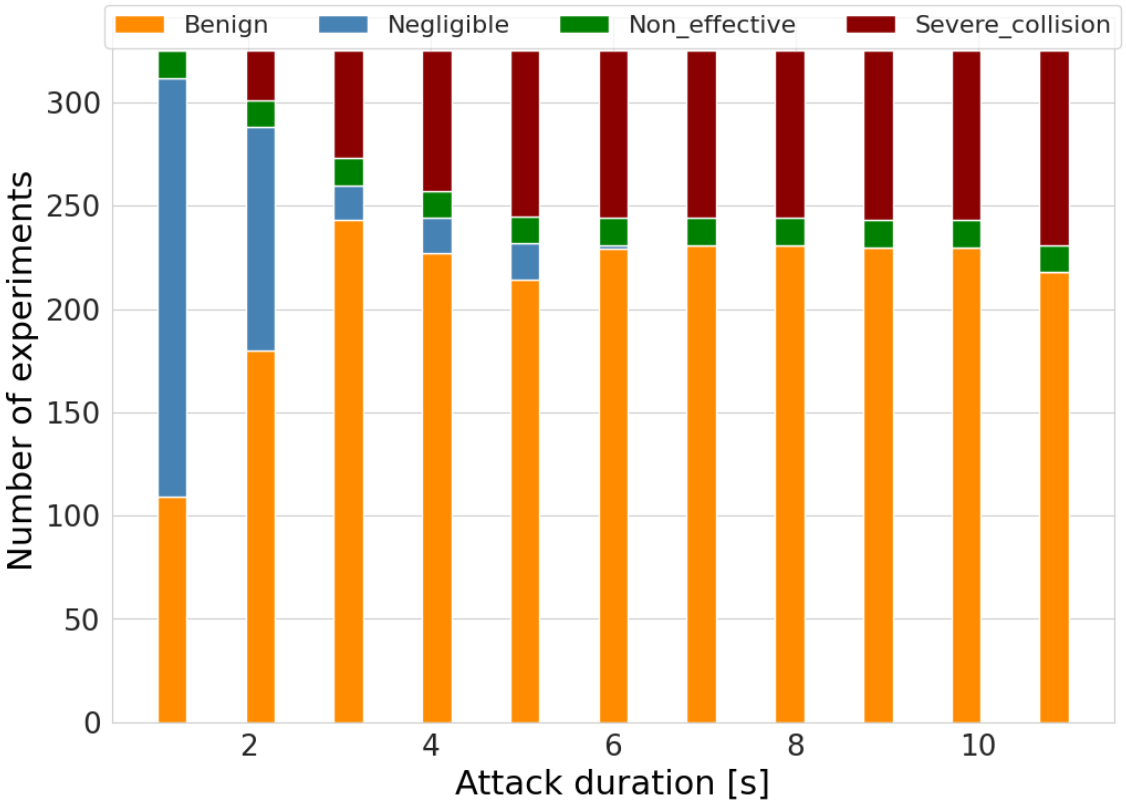


Figure 5.9: Outcome distribution for *variable* noise experiments on Model 3c.

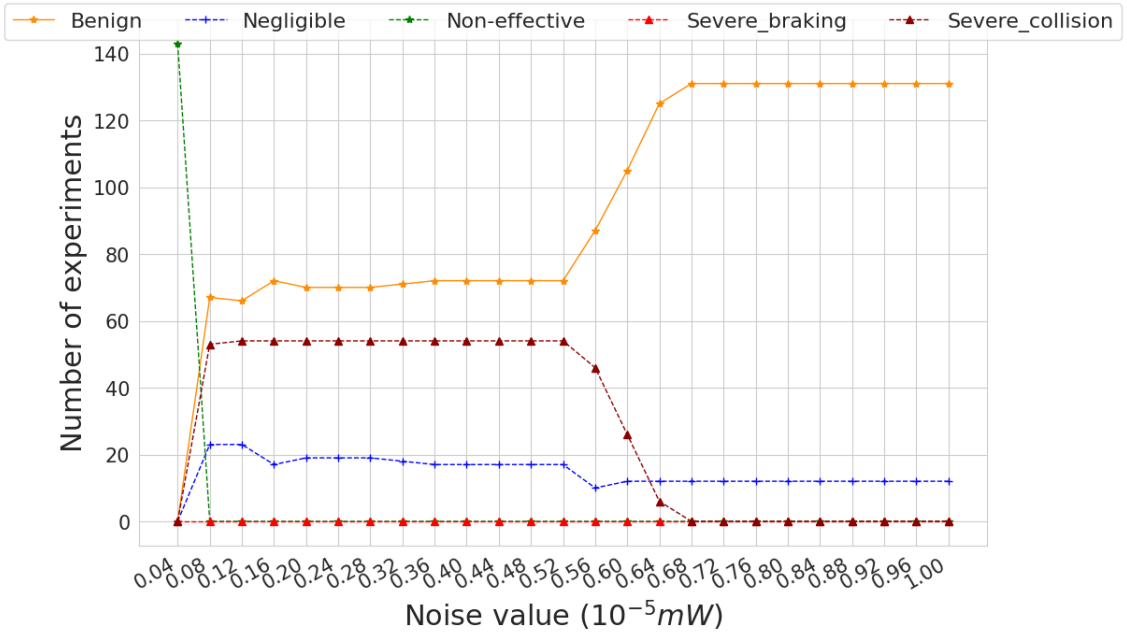


Figure 5.10: Outcome distribution as a function of noise value for Model 3c.

5.7 Model 4a: Degraded fallback multimodel (2 s)

Lastly, we found that the model combining the degraded P1 fallback with the PA fallback showed the most promising results. In the max noise attack scenario, there were only *Negligible* and *Benign* cases, as shown in figure 5.11. Moreover, the results from the variable noise testing (figure 5.12) showed the least amount of collisions out of all the models, coming in at only 667 – less than half of that of the base model.

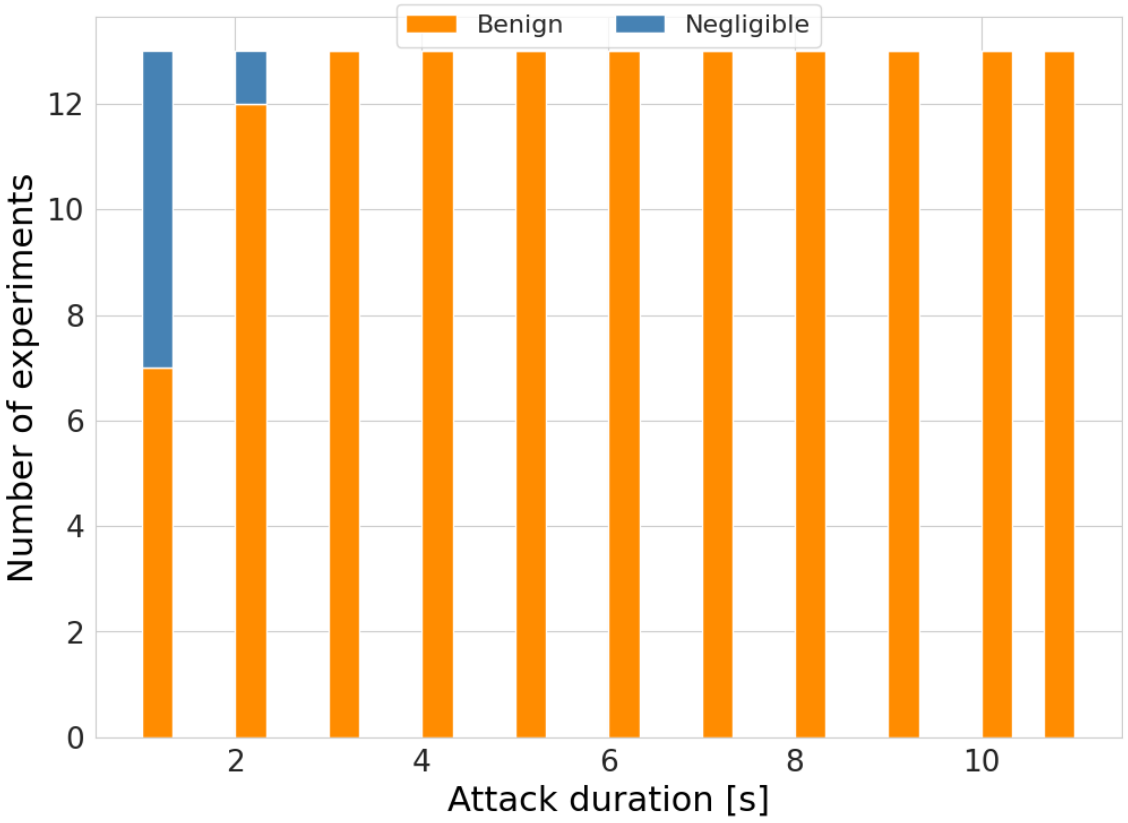


Figure 5.11: Outcome distribution for *maximum* noise experiments on Model 4a.

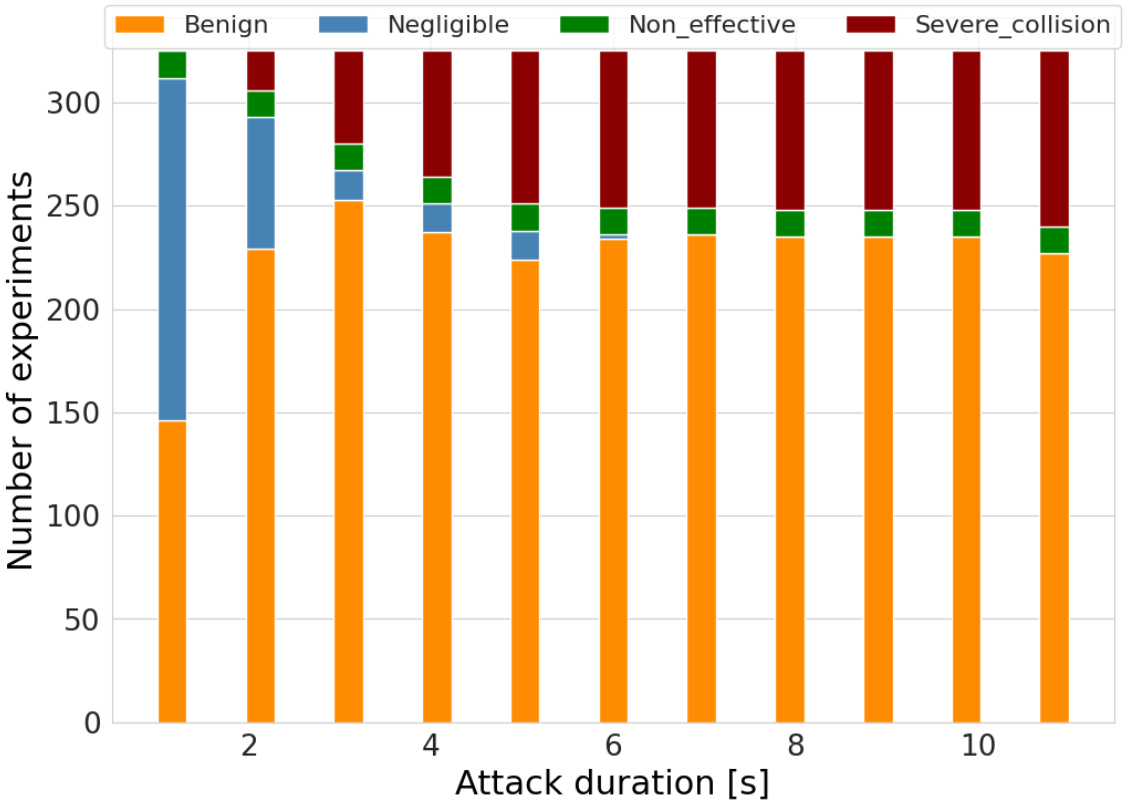


Figure 5.12: Outcome distribution for *variable* noise experiments on Model 4a.

5.8 Model 4b: Degraded fallback multimodel (2 s) + timer

The b-variant of Model 4 performed similarly but slightly differently than the a-variant presented in the previous section, as can be seen in table 5.3. In the maximum noise attack scenario, it performed marginally worse with one more *Benign* case, and in the variable noise scenario it performed slightly better with five fewer collisions.

Table 5.3: Outcome classification for Model 4b in the different noise scenarios.

Scenario	**Non-effective**	**Negligible**	**Benign**	**Severe**		**Total**
				Braking	**Collision**	
Maximum noise	0	6	137	0	0	143
Variable noise	143	265	2505	0	662	3575

5.9 Model 4c: Degraded fallback multimodel (1 s)

Model 4c performed identically to Model 4b in the maximum noise attack scenario, as can be seen in table 5.4. In the variable noise attack scenario it instead slightly outperformed the b-variant with 25 fewer experiments with collisions. Most of these

experiments no longer causing collisions were classified as *Benign*, however 7 of them were still classified as *Severe* but of the severe braking sort.

Table 5.4: Outcome classification for Model 4c in the different noise scenarios.

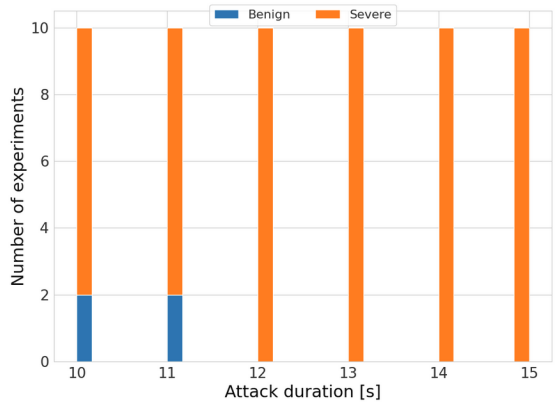
Scenario	Non-effective	Negligible	Benign	Severe		Total
				Braking	Collision	
Maximum noise	0	6	137	0	0	143
Variable noise	143	264	2524	7	637	3575

5.10 Miscellaneous results

In addition to the aforementioned fallbacks, we have also acquired some other miscellaneous results demonstrated below.

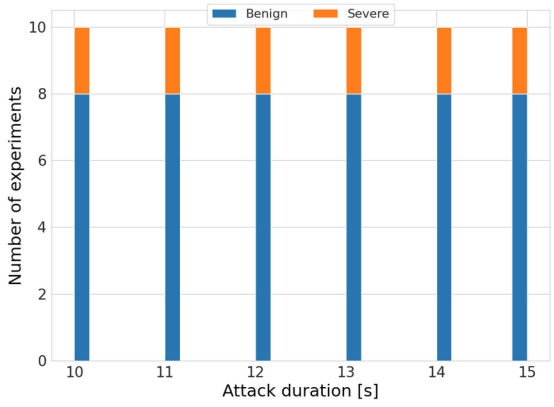
5.10.1 ACC headway times

The classification results for Model 3c using different headway times are shown through figures 5.13-5.16 as plots of number of experiments vs. attack duration.



Many severe cases due to deceleration briefly going above $5m/s^s$

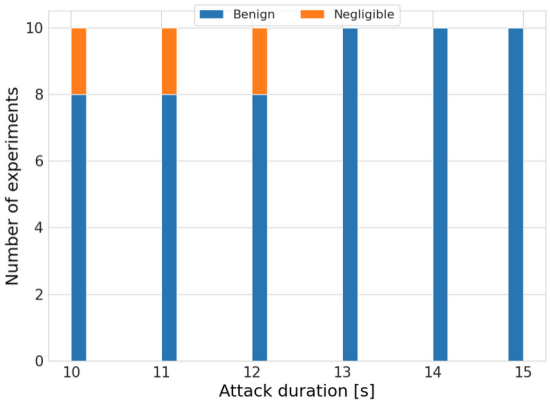
Figure 5.13: 3c with headway time 0.1 s



Not so many number of severe cases

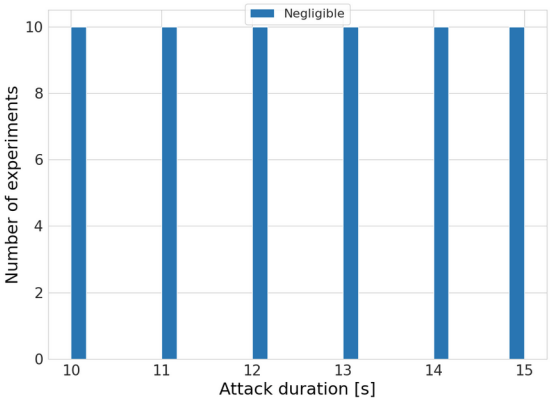
Figure 5.14: 3c with headway time 0.15 s

The results above and below were obtained from experiments run for 60 seconds each with attacks during 17 to 27 seconds and 18 to 33 seconds respectively. It can be seen that first at a headway time of 0.2 seconds (figure 5.15) there are no severe cases. Thus, this was the headway time chosen for this model's more thorough experiments described above in the previous section.



Best results at lowest possible headway time

Figure 5.15: 3c with headway time 0.2 s



Much better results with just slightly higher headway time

Figure 5.16: 3c with headway time 0.3 s

5.10.2 Message loss per noise level

The total recorded message loss per noise level is shown below in table 5.5. We found that a linear increase in injected noise results in an exponential increase in message loss. Roughly 50% message loss was observed already at a noise value of just 0.2 mW, and 100% message loss starting at noise level 0.63 mW. We additionally found that none of our fallbacks are able to prevent both collisions and severe braking below a threshold noise value of 0.66 mW.

Table 5.5: Noise values injected during attack vs. message loss.

injectedNoiseValue (mW)	**Message loss (%)**
0.2	50,52
0.4	65,41
0.6	99,77
0.62	99,99
0.63	**100**
0.66	**100**
0.8	100
1.0	100

5.11 Summary & comparison of important results

In this section we present a general summary of the results obtained. For easy comparison, the three fallback mechanisms and their timer variants have been given model IDs from 2-4, and the base model assigned ID 1. The classification results for these models under maximum noise attacks are first summarized in table 5.6. One full model and one model variant were successful in mitigating all collisions and fully avoiding severe braking: Model 4 and Model 3c. Out of these, Model 3c performed the best with the most *Negligible* cases, followed by Model 4a and Model 4b/c.

Table 5.6: Outcome classification for all models in the *maximum* noise scenario. Total experiments per model: 143.

Model ID	**Model Name**	**Non-effective**	**Negligible**	**Benign**	**Severe**	
					Braking	**Collision**
1	P1 base model	0	12	65	0	66
2*a*	Fallback to degraded P1	0	11	114	0	18
2*b*	Fallback to degraded P1 + timer	0	11	113	0	19
3*a*	Fallback to PA (2 s)	0	11	86	44	2
3*b*	Fallback to PA (2 s) + timer	0	8	87	38	10
3c	**Fallback to PA (1 s)**	**0**	**12**	**131**	**0**	**0**
4*a*	Degraded fallback multimodel (2 s)	0	7	136	0	0
4*b*	Degraded fallback multimodel (2 s) + timer	0	6	137	0	0
4*c*	Degraded fallback multimodel (1 s)	0	6	137	0	0

Next, table 5.7 shows the classifications for the same models but under variable noise attacks. We see that out of the four models, there were none without collisions. However, all of our models reduced the collisions in comparison to the already existing base model. Model 4c performed the best by reducing the collision count to 637, while Model 3b performed the worst with both 940 collisions and 294 cases of severe braking. 143 experiments were conducted per noise level, and the ones for the smallest noise level of 0.04 were non-effective on all models.

Table 5.7: Outcome classification for all models in the *variable* noise scenario. Total experiments per model: 3575.

Model ID	Model Name	Non-effective	Negligible	Benign	Severe	
					Braking	Collision
1	P1 base model	143	366	1591	0	1475
2*a*	Fallback to degraded P1	143	310	2233	0	889
2*b*	Fallback to degraded P1 + time	143	288	2291	0	853
3*a*	Fallback to PA (2 s)	143	355	1901	240	936
3*b*	Fallback to PA (2 s) + timer	143	325	1873	294	940
3*c*	Fallback to PA (1 s)	143	365	2342	0	725
4*a*	Degraded fallback multimodel (2 s)	143	274	2491	0	667
4*b*	Degraded fallback multimodel (2 s) + timer	143	265	2505	0	662
4c	**Degraded fallback multimodel (1 s)**	**143**	**264**	**2524**	**7**	**637**

Finally, the collision counts for each model are demonstrated in detail in tables 5.8 through 5.15. The tables show the total collisions along with the per-vehicle collision counts as a function of attack start time. To aide with analysis, the attack start times have additionally been divided into the following three regions:

- **Region 1:** Positive acceleration. attackStartTimes=(17.0, 17.4)
- **Region 2:** Negative acceleration. attackStartTimes=(17.8, 18.6, 19.0, 19.8)
- **Region 3:** Positive acceleration. attackStartTimes=(20.6, 21.0, 21.4)

Table 5.8: Collisions as a function of attack start time for Model 1.

Attack start time	Total Collisions	Collider vehicle			
		V1	V2	V3	V4
17.0	240	-	110	102	28
17.4	112	-	90	13	9
17.8	13	-	12	1	0
18.6	7	-	6	1	0
19.0	0	-	0	0	0
19.8	20	-	8	4	8
20.6	204	-	91	89	24
21.0	216	-	90	108	18
21.4	227	-	110	99	18

Table 5.9: Collisions as a function of attack start time for Model 2a.

Attack start time	Total Collisions	Collider vehicle			
		V1	V2	V3	V4
17.0	184	-	0	156	28
17.4	41	-	0	23	18
17.8	0	-	0	0	0
18.6	0	-	0	0	0
19.0	0	-	0	0	0
19.8	0	-	0	0	0
20.6	94	-	0	64	30
21.0	140	-	33	82	25
21.4	154	-	12	124	18

Table 5.10: Collisions as a function of attack start time for Model 2b.

Attack start time	Total Collisions	Collider vehicle			
		V1	V2	V3	V4
17.0	240	-	110	102	28
17.4	27	-	1	22	4
17.8	0	-	0	0	0
18.6	0	-	0	0	0
19.0	0	-	0	0	0
19.8	0	-	0	0	0
20.6	185	-	64	96	25
21.0	214	-	63	133	18
21.4	217	-	100	99	18

Table 5.11: Collisions as a function of attack start time for Model 3a.

Attack start time	Total Collisions	Collider vehicle			
		V1	V2	V3	V4
17.0	231	-	101	102	28
17.4	16	-	0	12	4
17.8	0	-	0	0	0
18.6	0	-	0	0	0
19.0	0	-	0	0	0
19.8	8	-	8	0	0
20.6	107	-	0	72	35
21.0	123	-	0	99	24
21.4	146	-	10	106	30

Table 5.12: Collisions as a function of attack start time for Model 3b.

Attack start time	Total Collisions	Collider vehicle			
		V1	V2	V3	V4
17.0	200	-	60	102	38
17.4	16	-	0	12	4
17.8	0	-	0	0	0
18.6	0	-	0	0	0
19.0	0	-	0	0	0
19.8	8	-	8	0	0
20.6	105	-	0	72	33
21.0	130	-	0	112	18
21.4	162	-	10	110	42

Table 5.13: Collisions as a function of attack start time for Model 3c.

Attack start time	Total Collisions	Collider vehicle			
		V1	V2	V3	V4
17.0	144	-	0	110	34
17.4	12	-	0	8	4
17.8	0	-	0	0	0
18.6	0	-	0	0	0
19.0	0	-	0	0	0
19.8	0	-	0	0	0
20.6	104	-	0	80	24
21.0	116	-	0	98	18
21.4	125	-	0	99	26

Table 5.14: Collisions as a function of attack start time for Model 4a.

Attack start time	Total Collisions	Collider vehicle			
		V1	V2	V3	V4
17.0	132	-	0	104	28
17.4	20	-	0	16	4
17.8	0	-	0	0	0
18.6	0	-	0	0	0
19.0	0	-	0	0	0
19.8	0	-	0	0	0
20.6	94	-	0	64	30
21.0	105	-	0	80	25
21.4	107	-	0	81	26

Table 5.15: Collisions as a function of attack start time for Model 4b.

Attack start time	Total Collisions	Collider vehicle			
		V1	V2	V3	V4
17.0	127	-	0	99	28
17.4	20	-	0	16	4
17.8	0	-	0	0	0
18.6	0	-	0	0	0
19.0	0	-	0	0	0
19.8	0	-	0	0	0
20.6	94	-	0	64	30
21.0	105	-	0	80	25
21.4	107	-	0	81	26

Table 5.16: Collisions as a function of attack start time for Model 4c.

Attack start time	**Total Collisions**	**Collider vehicle**			
		V1	V2	V3	V4
17.0	125	-	0	97	28
17.4	12	-	0	8	4
17.8	0	-	0	0	0
18.6	0	-	0	0	0
19.0	0	-	0	0	0
19.8	0	-	0	0	0
20.6	88	-	0	63	25
21.0	98	-	0	80	18
21.4	107	-	0	81	26

The tables above contain the results of 3575 experiments with varying noise values that were conducted for each of the respective models 1 to 4c.

Figure 5.17 shows the collision statistics per vehicle when using the best performing model, i.e., Model 4c, for each of the different injected noise values. The colliders are observed to always be either Car 3 or Car 4, where a car is counted as the collider when it crashes into another car in front of it. Note that per definition, Car 1 (V1) can never be a collider as it is the front vehicle and has nothing to collide into.

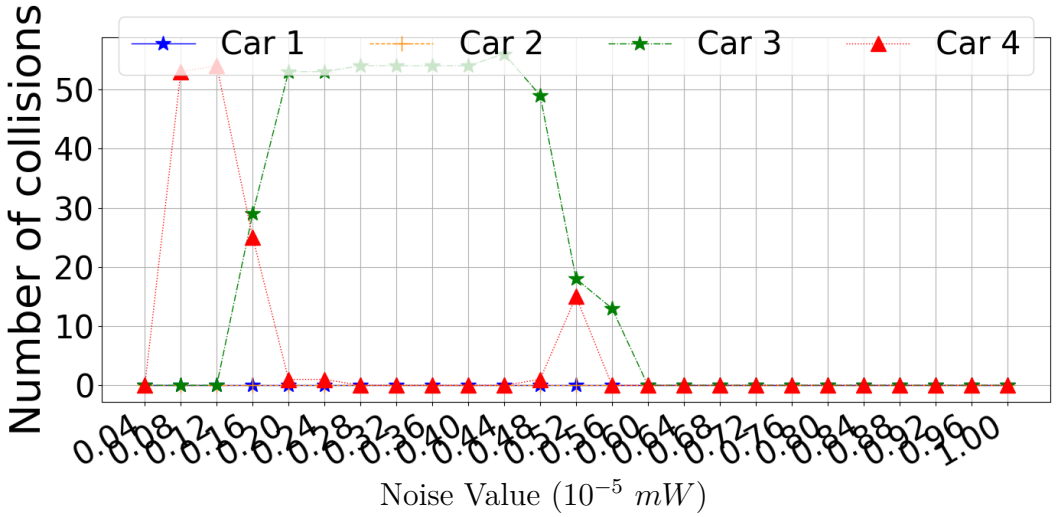


Figure 5.17: Plot of noise vs collisions per car for Model 4c.

6

Conclusion

In this final chapter we conclude with a discussion around the thesis work, its results and how it all relates to current and potential future research. We start with a discussion and interpretation of the results obtained, and then move on to discuss internal versus external validity. Next, we bring up potential future work that can be done building upon the work done in this thesis. Finally, we round off by summarizing the thesis and putting it into a general perspective.

6.1 Discussion

A summary of the model results obtained in this thesis is shown in table 6.1. Analyzing the results, we see that all of our proposed fallback models improve the controller resilience to barrage jamming attacks. The improvements range from a low of 39.73% fewer collisions for Model 2a to a high of 56.81% for Model 4c. The top 3 best performing model variants were the following:

1. **Model 4c.** The degraded fallback multimodel with 1 s fallback activation time performed the best. This was to be expected considering it combines the best aspects of different Model 2 and Model 3 variants. It makes use of the degraded P1 as an initial fallback that is quick to turn on, especially helping reduce collisions at lower noise levels. But it also then switches after 1 s communication loss to the otherwise more optimal PA fallback, gaining its benefits as well.
2. **Model 4b.** The degraded fallback multimodel extended with a minimum fallback on-time timer performed the second best. The reason for this variant performing better than the a-variant is that it was able to handle smaller noise values slightly better. This is likely because the 1 s minimum fallback on-time prevents the fallback from prematurely switching off upon just a few messages having been received when communication loss is still very high.
3. **Model 4a.** Coming in at third place is the a-variant of the degraded fallback multimodel. This variant had a few more collisions due to not handling smaller noise values as well.

Table 6.1: Summary of variable noise results showing how many experiments resulted in collision (Collisions) and the percentage reduction in those (Improvement) for each model in comparison to the P1 base model. Total experiments per model: 3575.

Model ID	Model Name	Collisions	Improvement (%)
1	P1 base model	1475	0
2*a*	Fallback to degraded P1	889	39.73
2*b*	Fallback to degraded P1 + timer	853	42.17
3*a*	Fallback to PA (2 s)	936	36.54
3*b*	Fallback to PA (2 s) + timer	940	36.27
3*c*	Fallback to PA (1 s)	725	50.85
4*a*	Degraded fallback multimodel (2 s)	667	54.78
4*b*	Degraded fallback multimodel (2 s) + timer	662	55.12
4c	**Degraded fallback multimodel (1 s)**	**637**	**56.81**

Next, we discuss the results for each specific model in the four first subsections below. Following that, we will use the remaining two sections to analyze the miscellaneous experiments conducted on ACC headway times and message loss per noise level.

6.1.1 Model 1

The results for the base model were as expected. Numerous collisions were recorded both in the maximum noise attack scenario and the variable noise scenario, as had been found in previous work by Mateen et al. [5][6]. While our experiments were not quite as extensive as theirs, we were still able to confirm their results as we used similar matching attack campaigns. This means that the P1 base controller is highly vulnerable to barrage jamming attacks, although only attacks lasting for more than 1 second. No collisions occurred at noise level 0.04 for this model – and all other models – indicating that such low noise levels do not impair communication in any notable way.

6.1.2 Model 2

The *Fallback to degraded P1* model was one of our worse models, yet it still significantly reduced collisions compared to the base model. This significant reduction in combination with the model's simplicity (simply changing some parameters) we think could prove it useful in certain situations. One interesting thing about this model is its "outlier" performance at an attack duration of 3 seconds. Both up until 3 seconds and from 4-6 seconds no collisions were recorded, but specifically at 3 seconds there was 1 collision. It is not obvious what might be causing this anomaly,

especially considering the collision occurred for both variants and the fallback is active already way before at 0.1 seconds into the attacks.

6.1.3 Model 3

The fallback to PA model (1 s) became the fourth best performing model variant, avoiding nearly as many collisions as the Model 4 variants. This is a bit interesting because its a- and b-variants were instead the worst performing model variants and shows that the fallback activation time (`fbActivTime`) is a very important parameter. We can draw the conclusion that activating the PA fallback first after 2 s is too late to avoid a lot of collisions, while 1 s is early enough to avoid a majority of collisions in the variable noise scenario and even all collisions in the maximum noise scenario.

What is harder to explain is why specifically during maximum noise attacks this model seemingly performs better even in its a- and b-variants than Model 2. However, we may note that fewer collisions than Model 2 in this scenario, it got a whole lot more cases of severe braking, making the sums of severe braking and collision cases go above the same sums for the Model 2 variants.

Another interesting thing to note about this model is that its a-variant actually got 2 collisions at 2 seconds attack duration in the maximum noise scenario, but at longer attack durations got *zero* collisions. This at first appears counterintuitive, since you would expect longer attack durations to mean higher severity and thus more collisions. However, we need to remember that this model switches to its ACC fallback only first after 2 seconds of communication loss. This means that for an attack duration of 2 seconds with maximum noise, there will be just 2 seconds of communication loss exactly, meaning the fallback will only be active very briefly before switching off. This brief activation will in practice have an almost negligible effect, meaning it is not surprising we get only a few less collisions than we got for the base model with the same attack duration. At 3 seconds and longer, we instead see more significant improvements since the fallback gets to be on for 1+ seconds.

6.1.4 Model 4

Considering Model 4 is a combination of Model 2 and 3, we would expect it to perform better than both, which it also generally speaking does. The only exception is in the maximum noise attack scenario where Model 3c performed slightly better than even the best Model 4 variant (4c). This suggests that switching to a degraded CACC before switching to the ACC is only helpful at lower noise levels where there is only partial communication loss, whereas at full communication loss it is better to just skip the degraded stage. Why would that be? Well, comparing the classifications vs. noise graphs for models 4c and 3c we see that 3c has fewer *Severe_collision* cases specifically within the noise range [0.48, 0.64]. These noise values result in high but not complete communication loss, meaning the ACC fallback is unlikely to stay on for long and be effective while the degraded fallback is since it activates even after just very brief loss. However, while the degraded fallback can be switched to quicker, it is also less effective at handling complete communication loss and is

therefore outperformed by the ACC-only 3c model in the maximum noise scenario.

6.1.5 ACC headway times

Our experiments testing different headway times for ACC fallbacks used in some models showed that headway times at 0.2 s and above enabled the fallback to avoid all collisions in the maximum noise attack scenario. In theory, one could therefore set the headway time to a value as high as possible to minimize severe cases. However, higher headway times come with the drawback of increasing the minimum distance maintained between vehicles in the platoon. This is undesirable as we want to keep small distances between the cars in order to save energy and thus presents us with a bit of a dilemma. We want to have as high of a headway time as possible in order to maximize platoon stability, but at the same time as low as possible of a value in order to maximize energy efficiency. We resolved this by choosing to go with the smallest possible headway time that still resulted in zero collisions – 0.2 s – giving us solid safety and still decent efficiency.

6.1.6 Message loss per noise level

The slight difference in the threshold noise values for when there is 100% message loss (at 0.63 mW) versus when our models are unable to prevent severe outcomes (at 0.66 mW) is worth discussing. Considering the two threshold values correspond to the same 100% message loss you would expect the models to be able to prevent severe outcomes already at the lower 0.63 mW noise value, but this is not the case. There is no reason to believe the models would behave differently for the exact same message loss, and thus these results can only be explained by the 100% message loss threshold varying slightly between different scenarios. Since we only conducted our message loss per noise level analysis specifically by jamming the entire duration from 0 to 60 seconds, we do not know how different the results might have been had we chosen other durations. It would make sense that when jamming for shorter durations the distance between cars varies differently, and this could affect message loss due to shorter distances increasing the likelihood of messages being received despite the noise present.

6.2 Validity of results

Here we discuss the validity of our results in terms of *internal* and *external* validity.

6.2.1 Internal validity

Threats to the internal validity include potential undiscovered bugs and suboptimal selection of attack campaign parameters. The former could invalidate results primarily through malfunctioning attack campaign scripts, but one could also view a broken model implementation as invalidating as then the results are in actuality not for the expected model but for a glitched variant of it. The latter could invalidate results by, e.g., not including sufficient attack duration values and thereby leaving out important

results. This is unlikely to have been the case though since previous research [5] conducted using the same simulators and base model showed no significant change in results beyond our chosen maximum attack duration of 11 s.

6.2.2 External validity

We judge that the external validity of our results is likely to be worse than the internal validity due to the highly specific simulation environment and configuration used for obtaining the results. For our experiments we have for example used a specific sinusoidal driving scenario with four vehicles in a platoon braking and accelerating in a fixed sinusoidal pattern. It is unlikely that vehicles would behave like this in a real-world scenario, complicating any potential generalization of our results. There are also many other simulation and configuration details that could change the results one way or the other. However, we do still believe the testing environment is similar enough to the real world for the results to still be meaningful, albeit with the caveat that they will only be roughly accurate in more general contexts.

6.3 Future work

In our initial investigation, we considered the possibility of using predictive models, such as acceleration predictions, to have some kind of a better degraded CACC system. The aim was to improve accuracy, similar to the approach used by Ploeg et al. [17]. However, we ultimately decided against this direction due to the increased complexity of such an implementation. That said, considering that it is likely such a model would be more effective at maintaining a platoon formation under attack, investigating it further might be worthwhile. Specifically, we could explore using AI to predict key parameters or identify a more robust algorithm that can better handle communication loss.

Another direction we explored was using Taylor series approximations to predict acceleration, but the first derivative prediction method did not yield significant improvements. However, better approximations might prove valuable in specific scenarios, such as dealing with lower levels of noise or refining control in more complex driving environments. The efficacy of such approximations may be evaluated through the use of similar experiments conducted in this thesis, but perhaps with the change of varying noise levels within every experiment instead of between experiments. Additionally, we considered the implementation of a Kalman filter to predict acceleration based on recent data, assuming sinusoidal driving conditions, which could offer a more accurate prediction model.

Finally, one could look into strategies for handling situations where sensors fail altogether, possibly by relying more heavily on communication and developing alternative fallback mechanisms to ensure safe platooning.

6.4 Summary

Nearing the end of this thesis report, we present one final outlook of the entire work conducted as a part of the thesis. We were successful in running the default P1 CACC model, design nine different models either as a modification or as an extension to the modifications, and quantify the results into visual data in form of both graphs and tables. The initial three models 2a, 3a and 4a are direct modifications of P1, while 2b, 3b and 4b are extensions of 2a, 3a and 4a respectively. 3c and 4c are the same as 3a and 4a but with different values of fallback activation time.

Coming to the a-variants, 2a was a degraded variant of P1 with the equation being simplified, modified to take radar values instead of values from the wireless V2V communication and distance between vehicles increased. Next was 3a, where we just switch to the PA controller which is an ACC. And finally, 4a is the model which combines 2a and 3a. The b-variants are based on the same principle as the a-variants but instead of switching back immediately to the regular P1 algorithm as soon as the attack concludes, we wait and make sure that the V2V incurs no further message loss for a specific period (1 s) in our case before switching back. This actually did not significantly change our results, at least not more than 5% in the best case scenario. The c-variants were different timings for the fallback activations and we noticed that the results got even better if we reduced the fallback to PA from 2 s to 1 s in both 3c and 4c in maximum noise and variable noise experiments respectively. But as we see the results, the difference between the b- and c-variants of Model 4 is relatively small and if repeated switching between the models adds an extra burden on the system in terms of computation power, we can very well use Model 4b with negligible trade-offs. The final decision rests on the implementer and making an informed decision is of utmost importance while considering safety according to us.

With Model 4c having 637 collisions out of the run 3575 experiments, instead of 1475 collisions on Model 1 or the baseline P1 controller model, we see that it is a 56.81% reduction in the number of collisions and only slightly ahead of Model 4b with 662 collisions. We also conclude that a simple switch to PA after 1 s of communication loss without increasing the distance between the vehicles like in 2c is also a promising result and worth exploring in case simplicity is of essence.

Bibliography

[1] D. Jia, K. Lu, J. Wang, X. Zhang, and X. Shen, "A survey on platoon-based vehicular cyber-physical systems," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 263–284, 2016. DOI: 10.1109/COMST.2015.2410831.

[2] J. Cerutti, G. Cafiero, and G. Iuso, "Aerodynamic drag reduction by means of platooning configurations of light commercial vehicles: A flow field analysis," *International Journal of Heat and Fluid Flow*, vol. 90, p. 108 823, 2021, ISSN: 0142-727X. DOI: https://doi.org/10.1016/j.ijheatfluidflow.2021.108823. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0142727X21000539.

[3] M. Segata, S. Joerer, B. Bloessl, C. Sommer, F. Dressler, and R. L. Cigno, "Plexe: A platooning extension for veins," in *2014 IEEE Vehicular Networking Conference (VNC)*, 2014, pp. 53–60. DOI: 10.1109/VNC.2014.7013309.

[4] R. Rajamani, H.-S. Tan, B. K. Law, and W.-B. Zhang, "Demonstration of integrated longitudinal and lateral control for the operation of automated vehicles in platoons," *IEEE Transactions on Control Systems Technology*, vol. 8, no. 4, pp. 695–708, 2000. DOI: 10.1109/87.852914.

[5] M. Malik, M. Maleki, P. Folkesson, B. Sangchoolie, and J. Karlsson, "Comfase: A tool for evaluating the effects of v2v communication faults and attacks on automated vehicles," in *2022 52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2022, pp. 185–192. DOI: 10.1109/DSN53405.2022.00029.

[6] M. Maleki, M. Malik, P. Folkesson, B. Sangchoolie, and J. Karlsson, "Modeling and evaluating the effects of jamming attacks on connected automated road vehicles," in *2022 IEEE 27th Pacific Rim International Symposium on Dependable Computing (PRDC)*, 2022, pp. 12–23. DOI: 10.1109/PRDC55274.2022.00016.

[7] C. Sommer, R. German, and F. Dressler, "Bidirectionally coupled network and road traffic simulation for improved ivc analysis," *IEEE Transactions on Mobile Computing*, vol. 10, no. 1, pp. 3–15, 2011. DOI: 10.1109/TMC.2010.133.

[8] M. Behrisch, L. Bieker-Walz, J. Erdmann, and D. Krajzewicz, "Sumo – simulation of urban mobility: An overview," vol. 2011, Oct. 2011, ISBN: 978-1-61208-169-4.

[9] A. Varga and R. Hornig, "An overview of the omnet++ simulation environment," Jan. 2008, p. 60. DOI: 10.1145/1416222.1416290.

[10] M. Lichtman, J. D. Poston, S. Amuru, *et al.*, "A communications jamming taxonomy," *IEEE Security & Privacy*, vol. 14, no. 1, pp. 47–54, 2016. DOI: 10.1109/MSP.2016.13.

[11] D. Moser, V. Lenders, and S. Capkun, "Digital radio signal cancellation attacks: An experimental evaluation," in *Proceedings of the 12th Conference on Security and Privacy in Wireless and Mobile Networks*, ser. WiSec '19, Miami, Florida: Association for Computing Machinery, 2019, pp. 23–33, ISBN: 9781450367264. DOI: 10.1145/3317549.3319720. [Online]. Available: https://doi.org/10.1145/3317549.3319720.

[12] W. Xu, W. Trappe, Y. Zhang, and T. Wood, "The feasibility of launching and detecting jamming attacks in wireless networks," in *Proceedings of the 6th*

ACM International Symposium on Mobile Ad Hoc Networking and Computing, ser. MobiHoc '05, Urbana-Champaign, IL, USA: Association for Computing Machinery, 2005, pp. 46–57, ISBN: 1595930043. DOI: 10.1145/1062689.1062697. [Online]. Available: https://doi.org/10.1145/1062689.1062697.

[13] *Our fork of sumo implementing the proposed fallback models*, https://github.com/TropicSapling/sumo, Nov. 2024.

[14] *Our fork of comfase automating the attack campaign process further*, https://github.com/karthikdgr8/ComFASE, Sep. 2024.

[15] P. Fernandes, "Platooning of ivc-enabled autonomous vehicles: Information and positioning management algorithms, for high traffic capacity and urban mobility improvement," AAI29194952, Ph.D. dissertation, 2012, ISBN: 9798841533177.

[16] Link to usePrediction description. [Online]. Available: https://sumo.dlr.de/doxygen/d9/d6a/class_c_c___vehicle_variables.html.

[17] J. Ploeg, E. Semsar-Kazerooni, G. Lijster, N. van de Wouw, and H. Nijmeijer, "Graceful degradation of cooperative adaptive cruise control," *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 1, pp. 488–497, 2015. DOI: 10.1109/TITS.2014.2349498.

A

Appendix - Implementation Details

```
// Set parameters to use here
const bool   useDegraded = <model-specific value>;
const bool   useFallbACC = <model-specific value>;
const double fbMinOnTime = <model-specific value>;
const double fbActivTime = <model-specific value>;

double msgLostTime    = realCurTime - vars->frontDataReadTime;
double fallbackOnTime = realCurTime - vars->fallbackSwitchTime;

if (!useFallbACC  (msgLostTime < fbActivTime && fallbackOnTime >
 ↪  fbMinOnTime)) {
    if (!useDegraded  (msgLostTime < 0.1 && fallbackOnTime >
     ↪  fbMinOnTime)) {
        vars->fallbackSwitchTime = -1.0;

        controllerAcceleration = _cacc(veh, egoSpeed, predSpeed,
         ↪  predAcceleration, gap2pred, leaderSpeed,
         ↪  leaderAcceleration, vars->caccSpacing);
    } else {
        if (!useFallbACC && vars->fallbackSwitchTime == -1.0) {
            vars->fallbackSwitchTime = realCurTime;
        }

        controllerAcceleration = _cacc(veh, egoSpeed,
         ↪  radarPredSpeed, predAcceleration, gap2pred, leaderSpeed,
         ↪  leaderAcceleration, vars->caccSpacing * 10);
    }
} else {
    if (vars->fallbackSwitchTime == -1.0) {
        vars->fallbackSwitchTime = realCurTime;
    }

    double ccAcceleration = _cc(veh, egoSpeed,
     ↪  vars->ccDesiredSpeed);
    double accAcceleration = _acc(veh, egoSpeed, radarPredSpeed,
     ↪  gap2pred, vars->accHeadwayTime);
    if (gap2pred > 250  ccAcceleration < accAcceleration) {
        controllerAcceleration = ccAcceleration;
    } else {
        controllerAcceleration = accAcceleration;
    }
}
```

Listing 16: Full code used for our proposed fallback models.