In-Vehicle Security

CPS and IoT Security

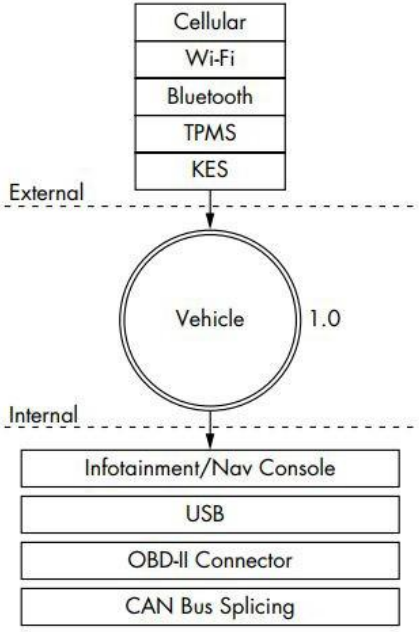
Alessandro Brighente

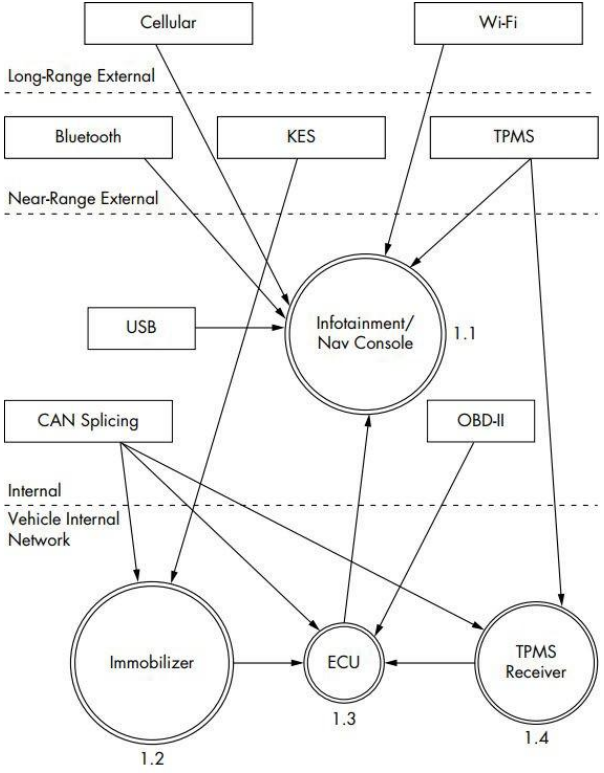
Master Degree in Cybersecurity

UNIVERSITÀ DEGLI STUDI DI PADOVA

SPRITZ SECURITY & PRIVACY RESEARCH GROUP

What is there in a Car?





Vehicle Components and Network

SPRITZ
SECURITY & PRIVACY
RESEARCH GROUP

UNIVERSITÀ
DEGLI STUDI
DI PADOVA

- Modern vehicles contain a large number of Electronic Control Units (ECUs)
- ECUs are embedded systems that control one or more (sub)system in a car or other vehicles
- Examples: engine control module, powertrain control module, transmission control module, suspension control module,...
- ECUs are nodes in a rather complex in-vehicle network, where they should communicate one another to report many different types of information

What is CAN Bus

- Controller Area Network (CAN) is an in-vehicle network bus-based standard developed in 1986 by Bosch
- It allows in-vehicle components (ECUs) to communicate one another
- It has broad application in automotive systems, including power train, suspension, and braking
- In 2003, it became a standard series with ISO 11898
- The Society of Automotive Engineers (SAE) standardized CAN bus communications to have asynchronous data rate up to 1 Mbps at a 40 m distance

Why CAN Bus

- **Robustness:** ideal for safety applications thanks to its durability and reliability (bit stuffing, bit monitoring, frame check, ack and CRC checks)
- **Low cost:** objective is to reduce errors, weight, wiring and costs
- **Flexibility:** it is a message-based protocol, so nodes can be added or removed without updating the system
- **Efficiency:** messages with high priority are clearly marked and have prioritized access to the bus

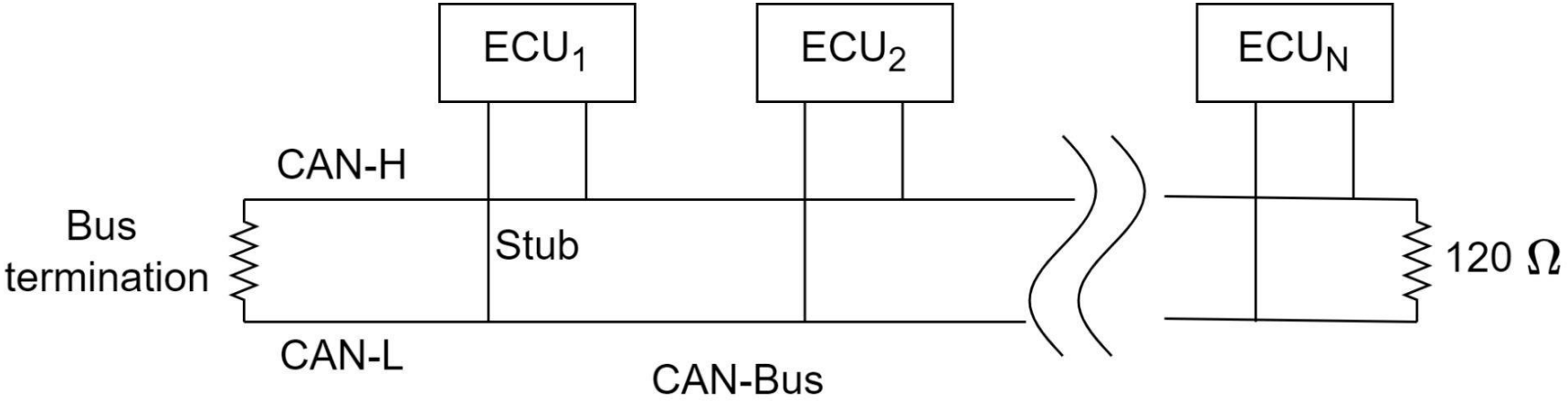
CAN Bus Wiring

SPRITZ
SECURITY & PRIVACY
RESEARCH GROUP

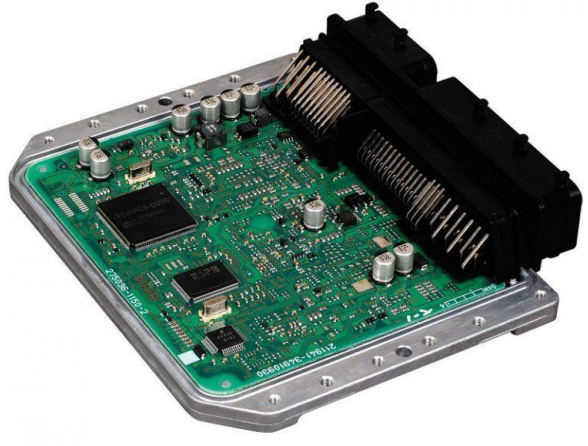
UNIVERSITÀ
DEGLI STUDI
DI PADOVA

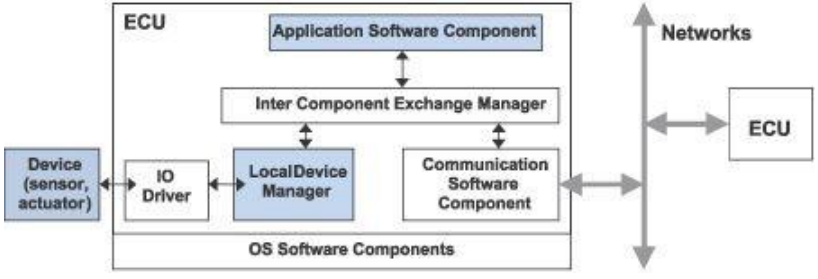
- To achieve reliability, it is important to prevent message collision such that no data gets lost
- Thanks to the shared bus and the message handling system, CAN allows for a reduced number of wires to achieve reliability
- Redundancy is the best way to achieve reliability, however it comes with a higher communication implementation cost (huge number of wires)
- Point-to-point communications were used before CAN

CAN Bus Architecture



Electronic Control Unit

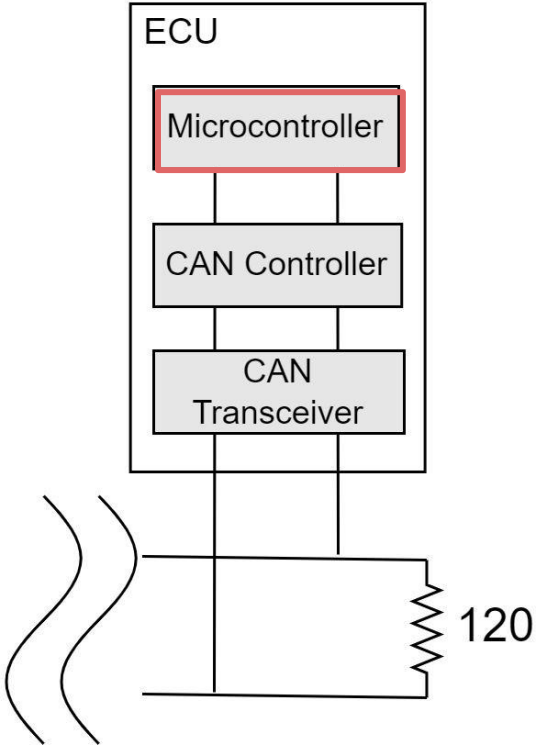




ECU Architecture

Microcontroller:

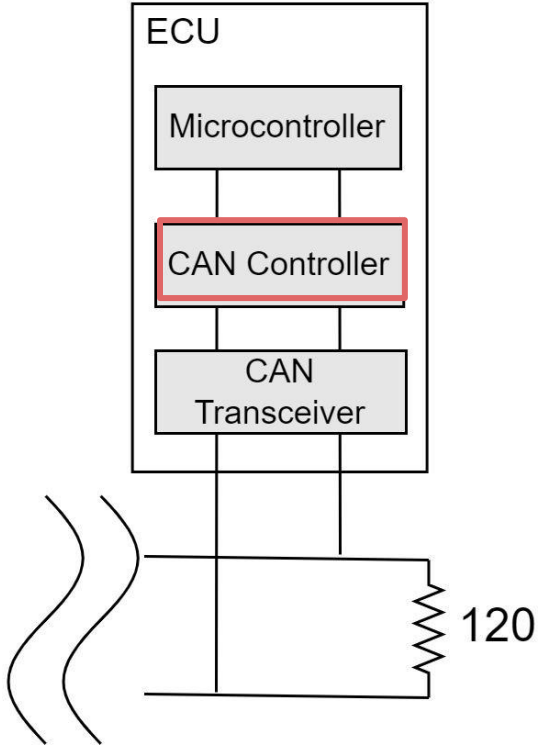
- central processing unit of the ECU to decide what the received signal means and what messages it wants to transmit
- Allows for the connection of other devices, such as sensors and actuators



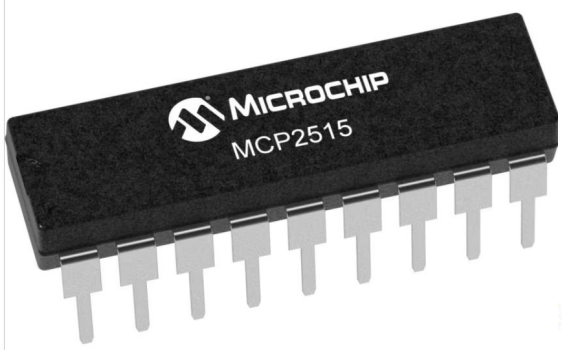
ECU Architecture

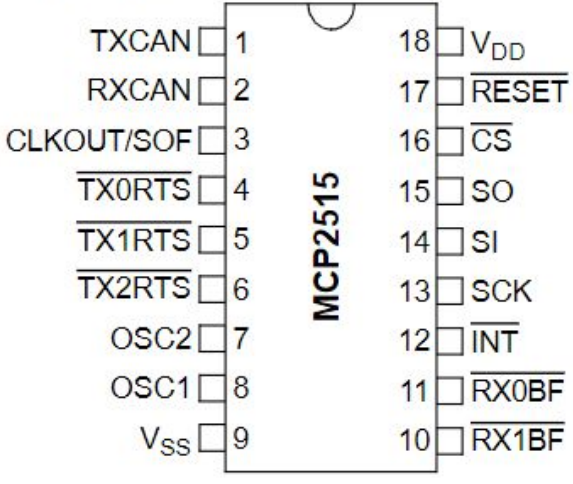
CAN controller:

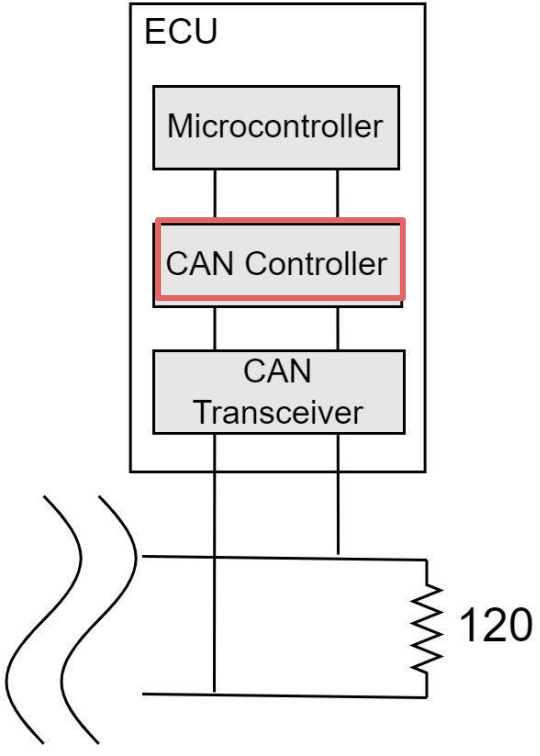
- Stores the received serial bits and passes messages to the processor
- Transmits bits serially on the bus upon reception from the processor



ECU Architecture

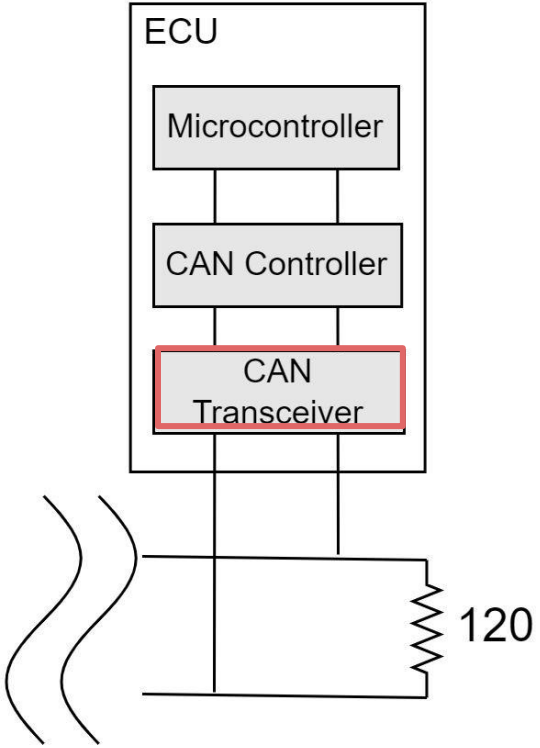






CAN transceiver:

- Protects the CAN controller from overvoltage
- Converts CAN bus levels to levels that the CAN controller can use
- Converts the data stream from the CAN controller to CAN bus levels

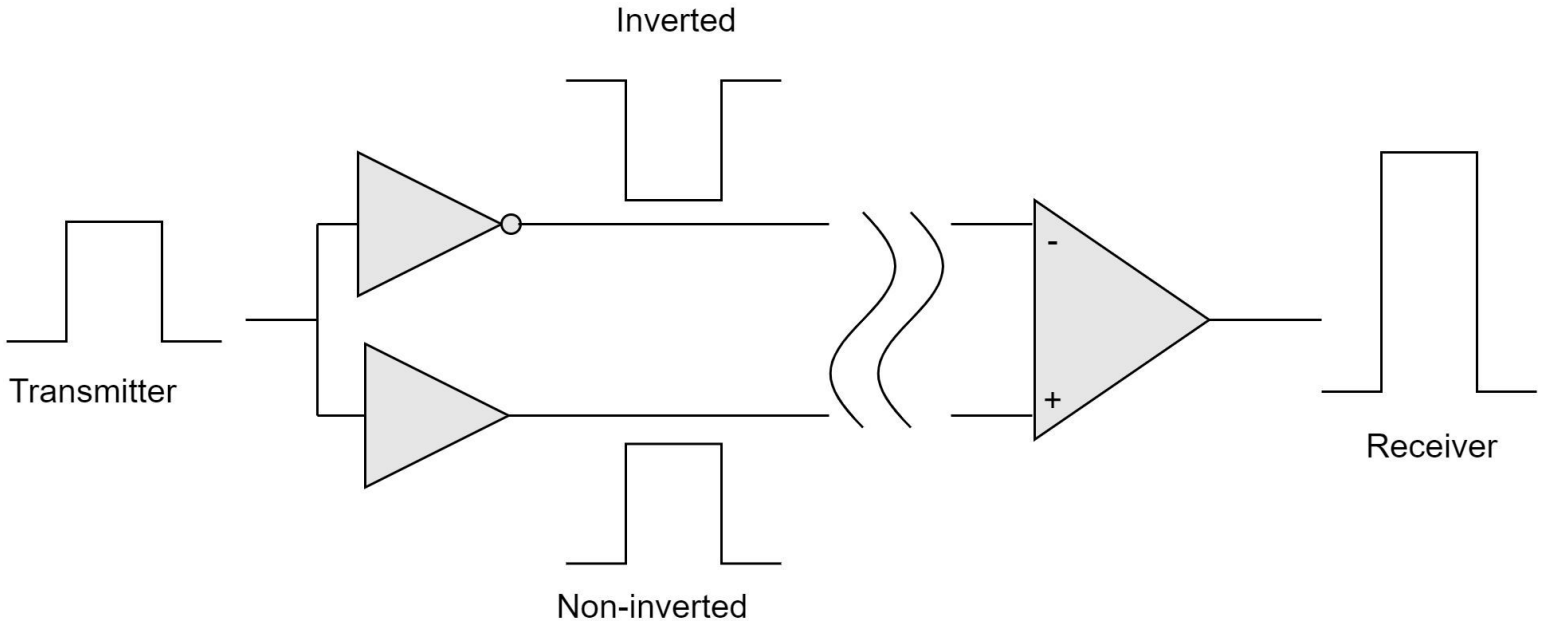


CAN Bus Communication

- CAN bus uses **differential** **wired-AND signals**
- It uses two signals, CAN high (CAN-H) and CAN low (CAN-L)
- They are either driven to a dominant state with CAN-H > CAN-L
- Or driven to a recessive state, with CAN-H $\leq$ CAN-L
- Dominant state = 0 bit
- Recessive state = 1 bit

CAN Bus Communication

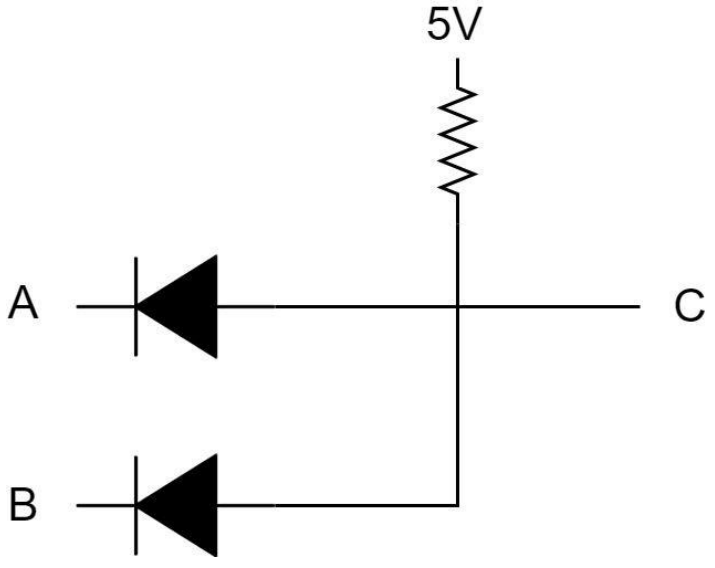
- CAN bus uses **differential** wired-AND signals



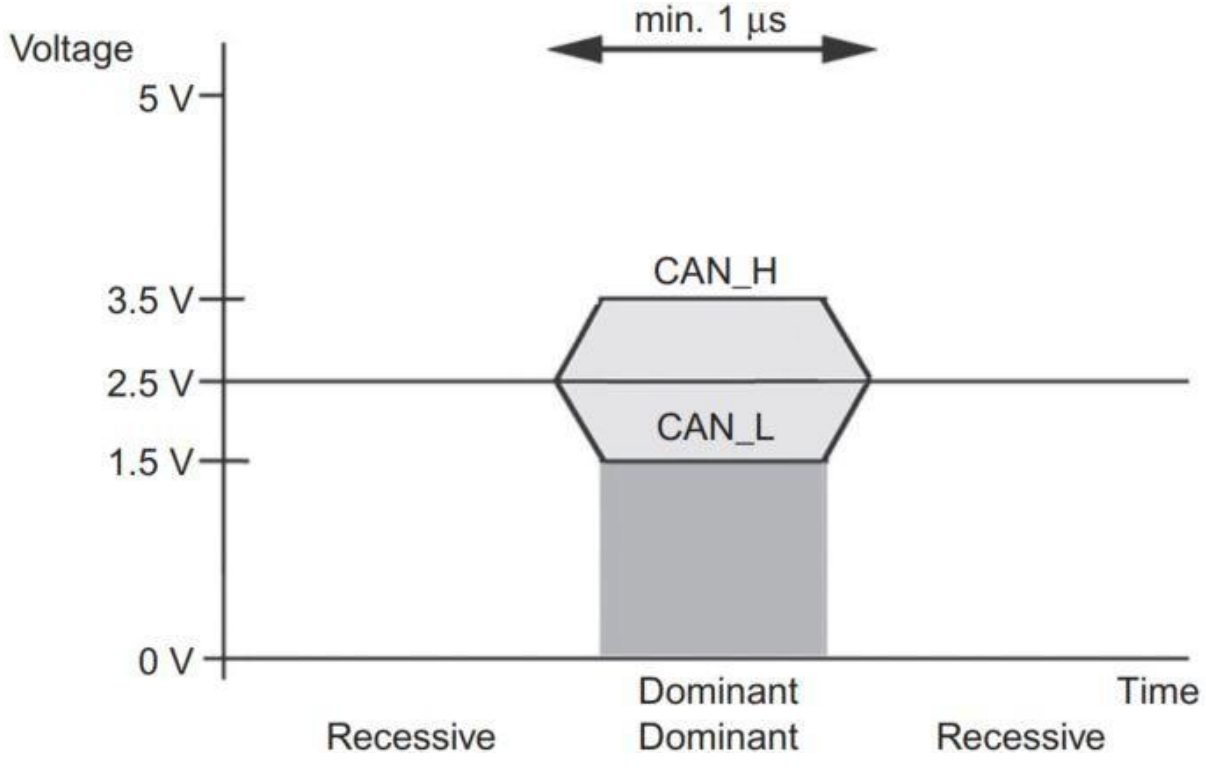
CAN Bus Communication

- CAN bus uses differential **wired-AND signals**

Input		Output
A	**B**	**A AND B**
0	0	0
0	1	0
1	0	0
1	1	1

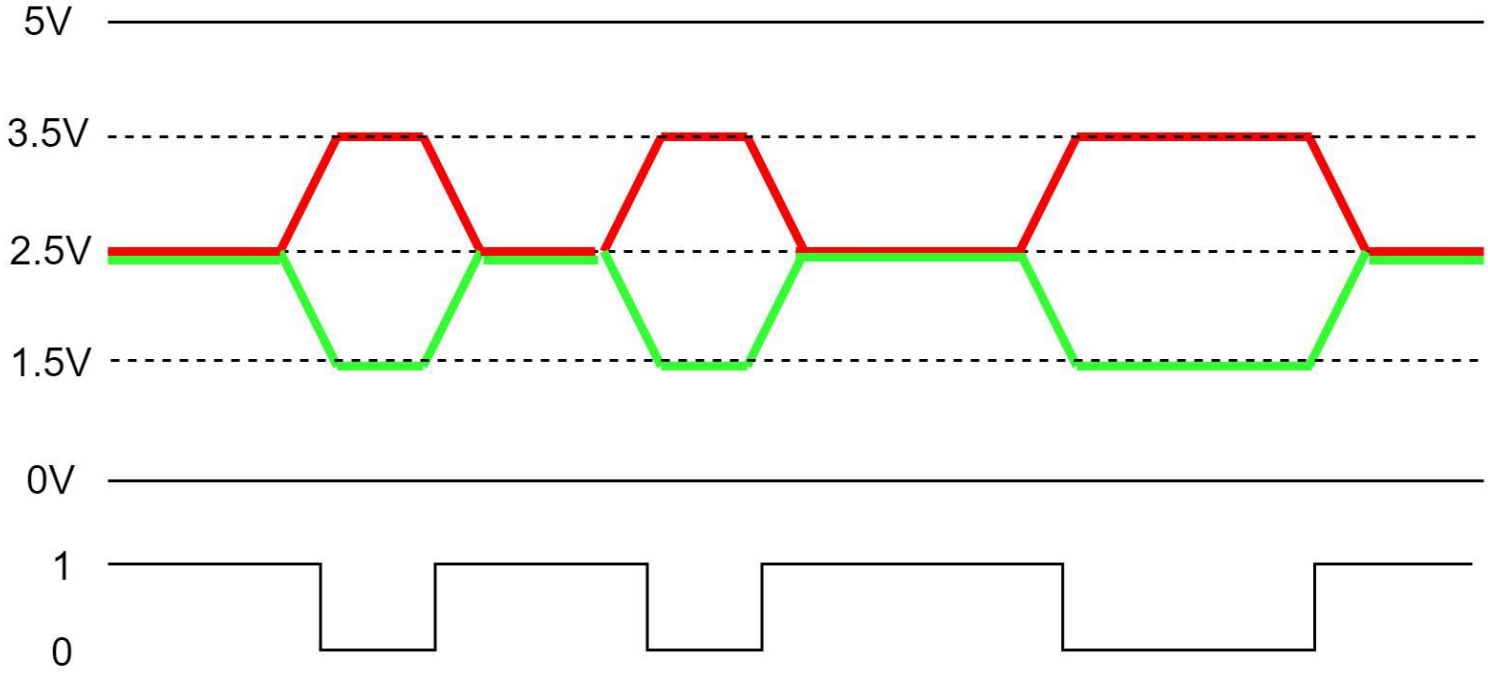


- Either dominant or recessive



CAN Bus Communication

- Dominant state = 0 bit
- Recessive state = 1 bit



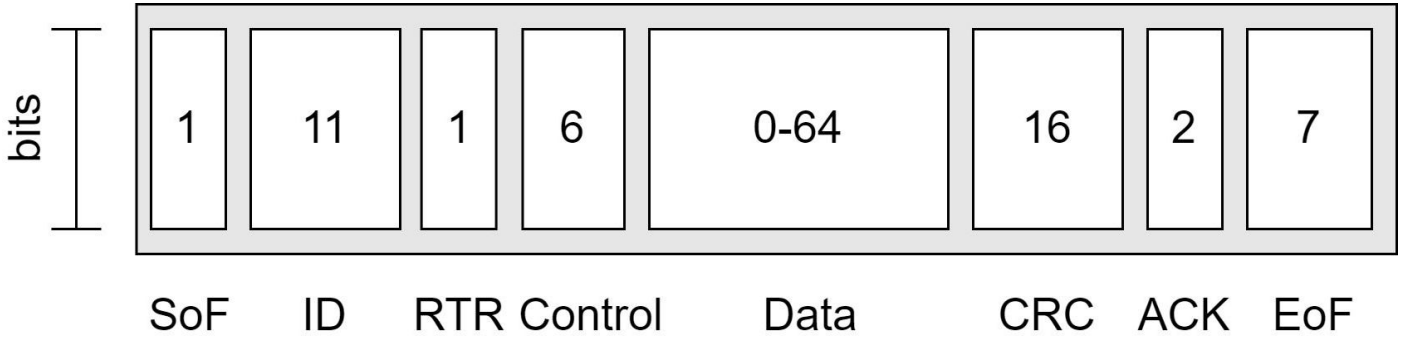
Physical and Data Link Layer

- The CAN bus is described by a data link and physical layer
- ISO 11898-1 describes the data link layer, while ISO 11898-2 describes the physical layer

- From ISO 11898-2:
  - CAN nodes must be connected via a two-wire bus with baud rates up to 1 Mbit/s or 5 Mbit/s (CAN FD)
  - Maximal cable lengths should be between 500 m (125 kbit/s) and 40 m (1 Mbit/s)
  - The CAN bus termination must be a 120 Ohms resistor

CAN Frames

- The standard CAN frame is composed by 8 message fields
- CAN 2.0A and 2.0B differ in the length of the ID field



Message fields of the CAN frame

CAN Frame Message Fields

- **Start of Frame (SoF):** dominant 0 to tell the other a node wants to talk
- **ID:** frame identifier. The lower the ID, the higher the priority. It is not an identifier of the sender
- **Remote Transmission Request (RTR):** indicates whether a node sends data or requests data from another node
- **Control:** contains the Identifier Extension Bit (IEB) which is dominant 0 for 11 bits ID. It also contains the 4 bit Data Length Code (DLC) that specifies the length of the data bytes to be transmitted (0 to 8 bytes)

CAN Frame Message Fields

- **Data:** payload in common network terminology. CAN signals that can be extracted and decoded for information
- **CRC:** check for data integrity
- **ACK:** indicates whether a node has acknowledged and received data correctly
- **End of Frame (EoF):** end of the CAN frame

Validity of the CAN Frame

- CAN frames need to satisfy certain conditions to be valid
- If a node detects its transmission of an erroneous message, it takes actions accordingly
- This is the CAN bus error handling, and each node has its own error counter and state
- The state is in the set (active, passive, bus off) depending on the counter value

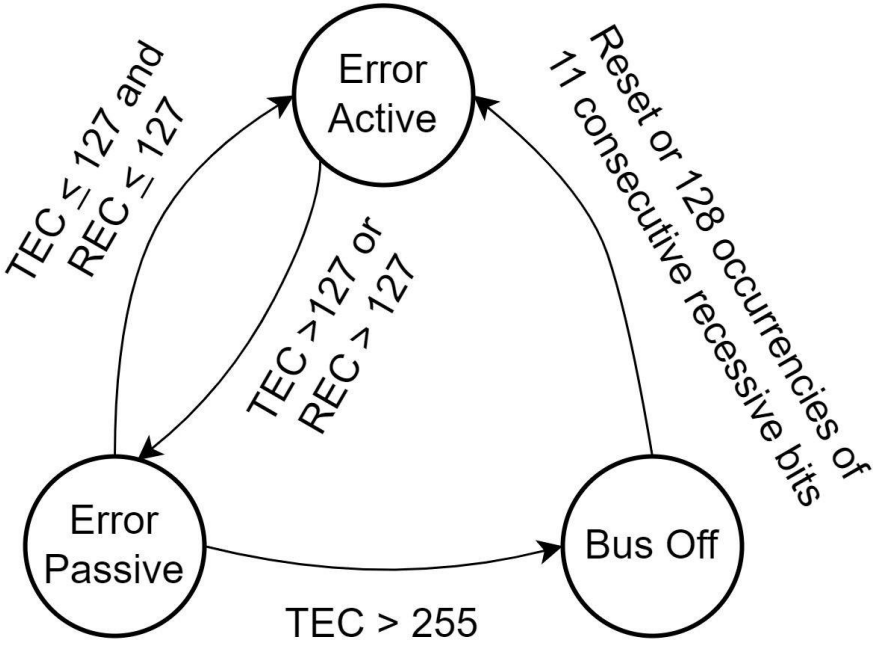
Validity of the CAN Frame

- Counters are Transmit Error Counter (TEC) and Receive Error Counter (REC)
- TEC = TEC+ 8 if an error occurs during transmission
- REC = REC + 1 if an error in the reception
- Success decreases the counter by one in both cases
- Nodes start at the error active frame
- A node transitions to the error passive state if the value of the counter exceeds 127

Validity of the CAN Frame

- In the error passive state the node can only write recessive error flags, thus not affecting the bus traffic
- The node transitions to the Bus off state if the counter exceeds 255
- In this case, the node no longer takes part to the bus traffic
- To get back to error active, the node needs to be either reset or should count 128 occurrences of 11 consecutive recessive bits

Validity of the CAN Frame



Error detection and fault confinement states

Bus Off Mode

- When an ECU transitions to the bus off mode it means that something serious is happening
- To prevent accidents but still allow the driver to reach a safe place, the ECU in bus off usually runs with predefined parameters and reduced functionality (e.g., limited RPM)
- We call an ECU in this state as *limp home*
- In this state, warning limps are turned on on the driver's dashboard
- Depending on the severity, the limp mode is eventually disabled

Errors in CAN Bus Frames

- In CAN bus transmissions we can have no less than five types of errors

1. *Bit error*: occurs when an ECU, after comparing its transmitted bits with those on the CAN bus detects a mismatch (does not hold in arbitration mode)
2. *Stuff error*: after transmitting five consecutive same polarity-bits, the ECU sends an opposite bit to maintain soft synchronization. An error occurs if stuffing does not happen
3. *CRC error*: computed one different from the received one

Errors in CAN Bus Frames

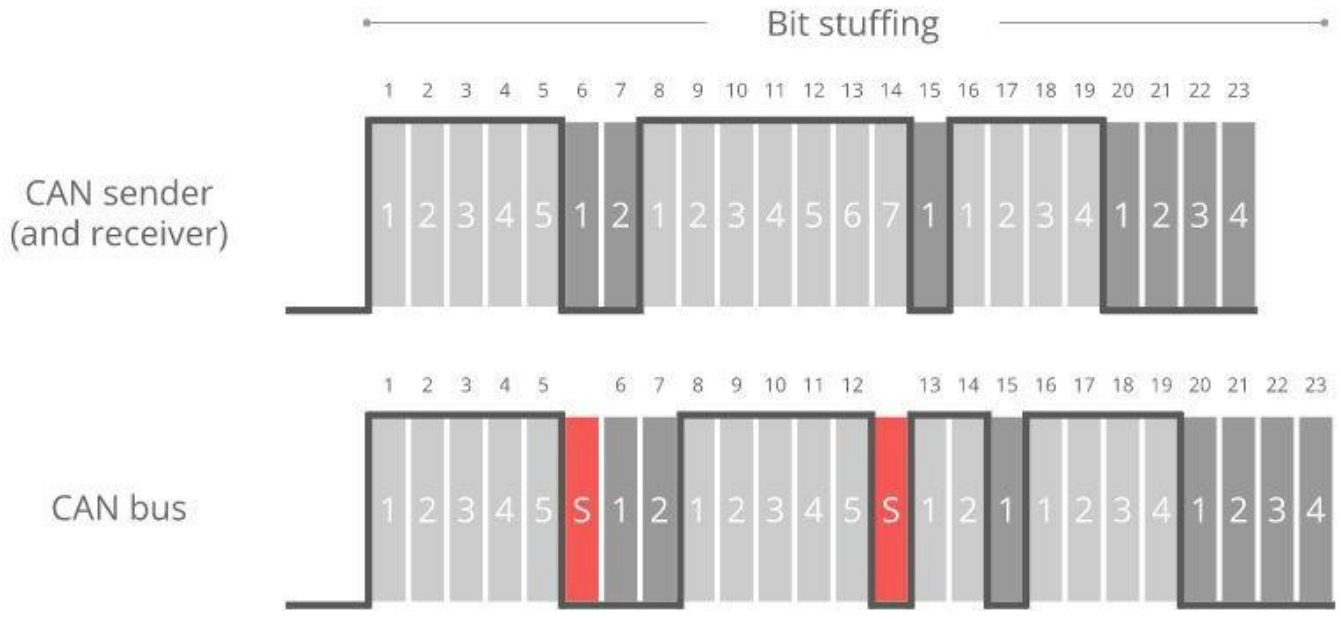
- In CAN bus transmissions we can have no less than five types of errors

4. *Form error*: if a fixed-form bit field (e.g. EoF) has a bit error the ECU raises a form error
5. *ACK error*: when a node receives a frame, it responds with a dominant bit in the ACK frame. If none respond, then ACK error

Error Flags

- When a node detects an error it needs to warn the others on the bus
- The node raises an *error flag* which may come in two forms: active and passive
- Nodes that are in **error-active mode** issue an *active error flag* which consists of 6 dominant bits
- The transmitted frame causes other nodes to violate the bit-stuffing rule, transmit their own error frame caused by the stuff error, and terminate any on-going transmissions or reception

Something More on Errors



Every 5 consecutive bits of same polarity, add one for synch

See source on: https://www.csselectronics.com/pages/can-bus-errors-intro-tutorial

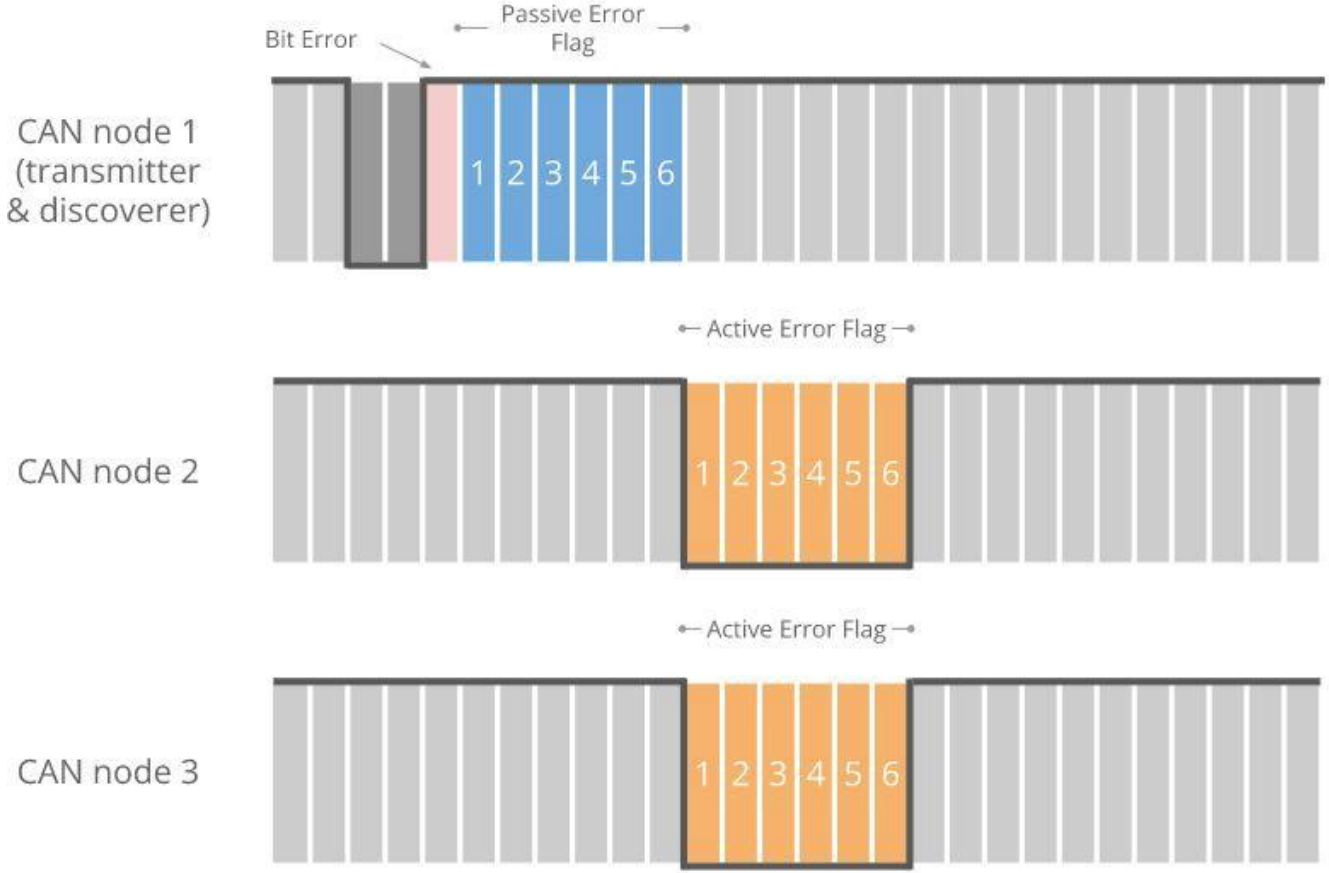
Error Flags

- Bit stuffing error is a violation of this rule and is visible to all CAN nodes
- Hence, we can have a bit error that makes the transmitter raise this flag, and all other nodes raise this error to notify the overall CAN
- We call them as *primary Active Error Flag* and *secondary Active Error flags*

Passive Error Flags

- When the node is in passive error mode, and is transmitting, it can only transmit recessive bits as error flag
- This is in turn detected as a bit stuffing error by all other nodes
- If all other nodes are in error active, they will create an active error flag
- If instead the error passive node raises an error passive flag while receiving, this will not be detectable by other nodes in the network

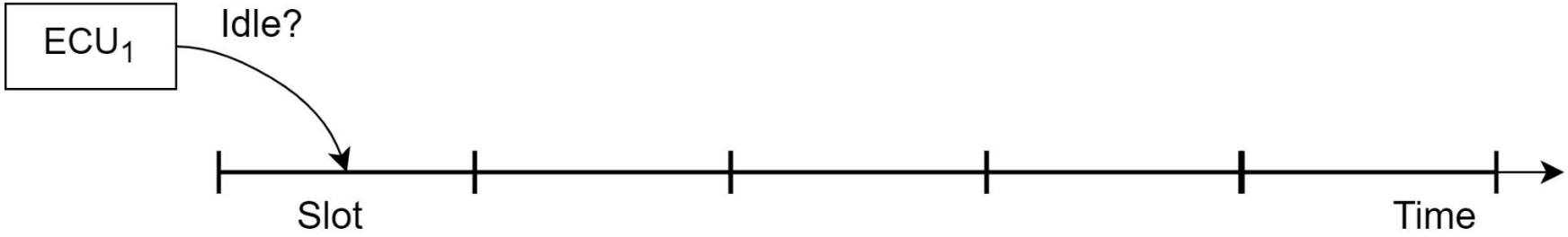
Passive Error Flags



Bus Arbitration

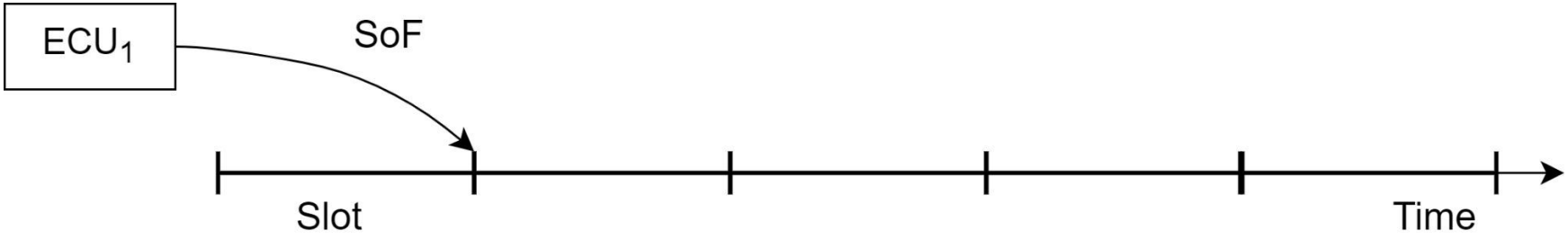
- It refers to the process by which nodes get a share of the bus resources to transmit
- CAN arbitration protocol is both priority-based and non-preemptive
- Non-preemptive: a message cannot be preempted by a higher priority one if this was queued after the transmission began
- The media access protocol alternates contention and transmission phases

Bus Arbitration



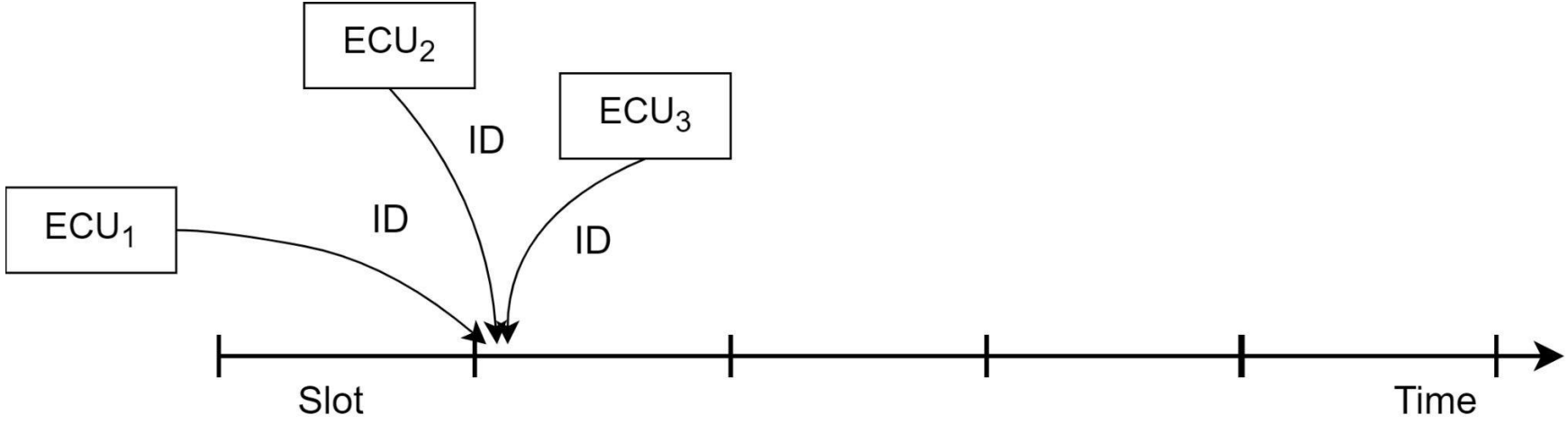
- Time is divided into slots, each with a length sufficient for a message to go back and forth in the whole bus
- A node wishing to transmit checks whether the channel is idle

Bus Arbitration



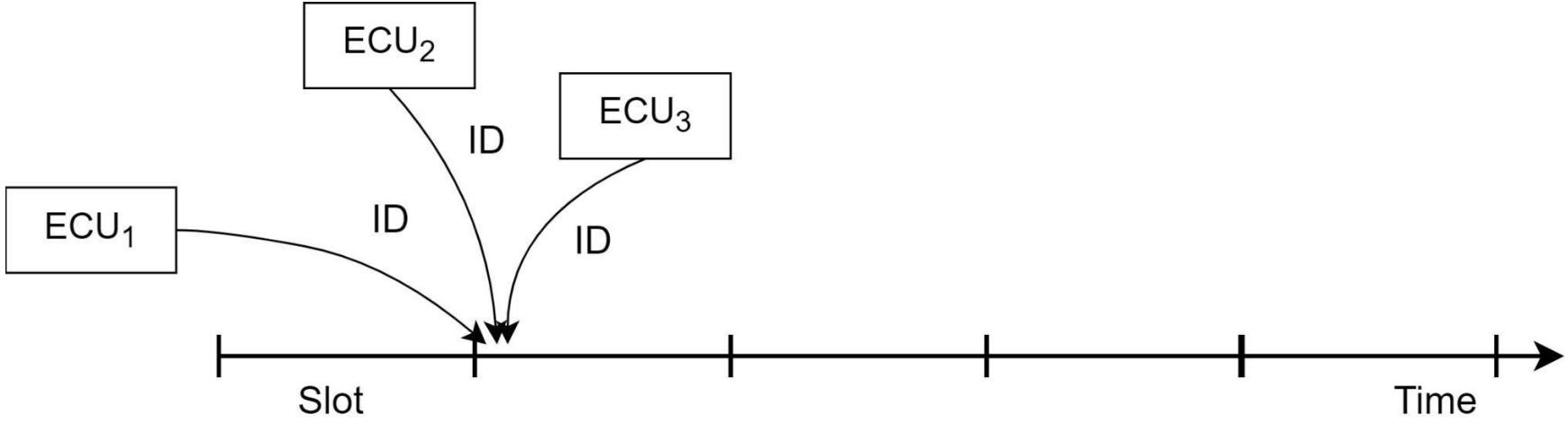
- If idle, it waits the next slot and start a contention phase by transmitting a SoF bit

Bus Arbitration



- All other nodes wishing to transmit send their ID
- The CAN controller compares its output with the actual bus level at the end of each bit cycle

Bus Arbitration



- The node loses contention in case it submitted a recessive level (high) and detects a dominant level (low)
- Error is dominant transmitted and recessive detected

Bus Arbitration Example

S1	1	0	0	0	1	0	0	0
S2	0	0	1	1	1	0	0	1
S3	0	0	1	1	1	0	1	0
Bus	0	0	1	1	1	0	0	1

S2 wins the arbitration at the 7th bit

Bus Arbitration

- After the node that wins the competition terminates its transmission with an EoF, there should be new transmissions
- The CAN bus uses a 3 bit inter-frame symbols to separate packets
- After this, the bus becomes idle again and arbitration shall commence again

Accessing CAN Bus

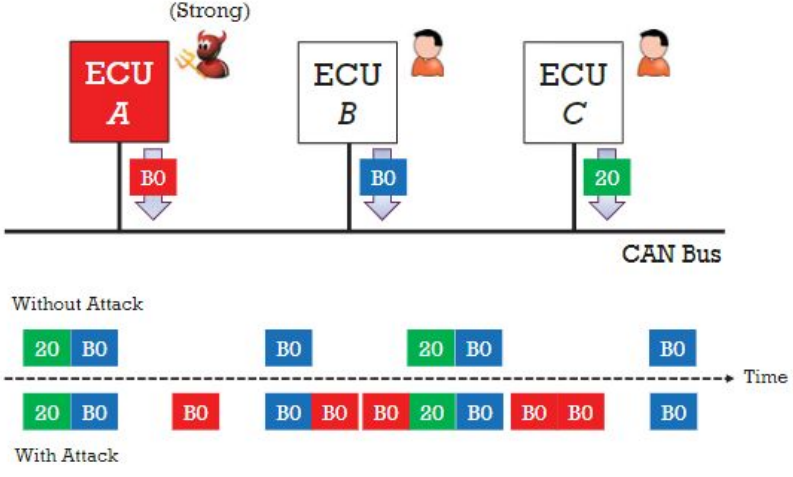
- To access CAN bus you need to connect to the On Board Diagnostic (OBD) port
- Basically all cars nowadays use the OBD-II standard port
- It is usually located near the driver's or passenger's seat
- To talk with the CAN bus you need a converter, such as that in the figure
- This converts CAN bus data to something readable via a USB port

Attacker Model

- Attacker can physically and remotely compromise more than one in-vehicle ECU in different means
- The attacker objective is to manipulate or impair in-vehicle functions
- Two means:
  - inject (attack) messages with a spoofed ID
  - stop/suspend message transmission of the compromised ECU
- Furthermore, ECUs can be weakly or fully compromised
- Weakly compromised means an attacker cannot inject *any* fabricated message

Fabrication Attack

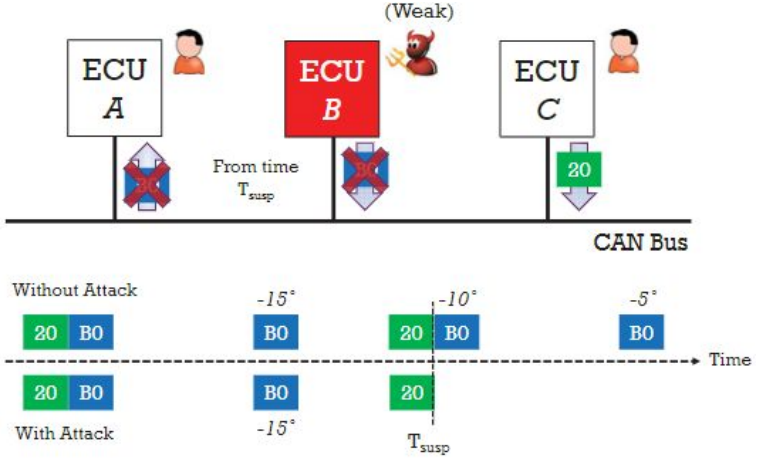
- Requires a compromised ECU as a strong attacker to fabricate and inject messages with forged ID, DLC, and data
- **Objective:** override any periodic messages sent by a legitimate safety-critical ECU so that the receiving ECU gets distracted or become inoperable



(a) Fabrication attack.

Suspension Attack

- Requires a single weekly compromised ECU
- **Objective:** stop/suspend the weakly compromised ECU's message transmission, thus preventing the propagation of info to other ECUs
- Detrimental also for ECUs that rely on the reception of such messages

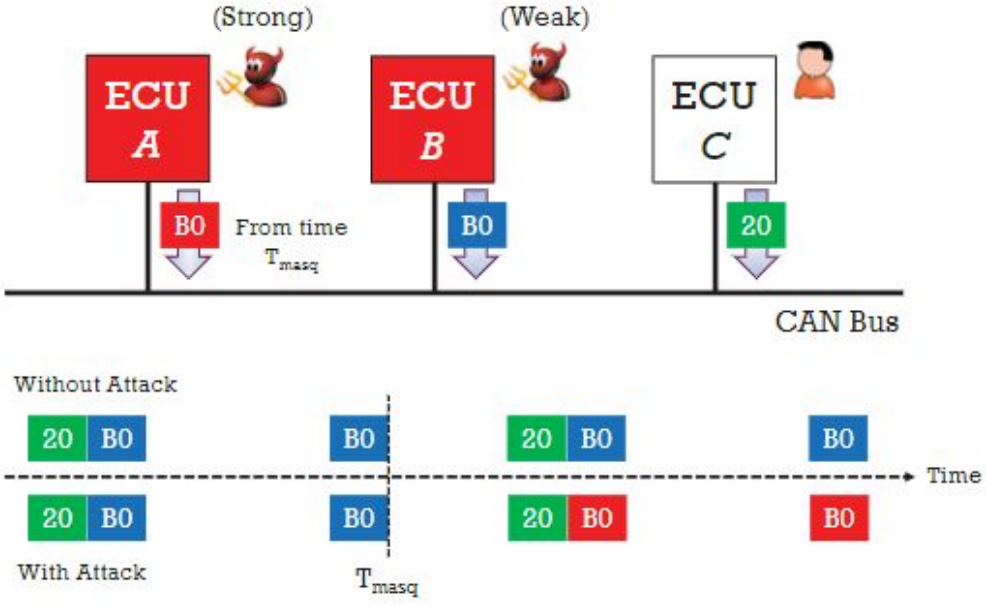


(b) Suspension attack.

Masquerade Attack

- Need to compromise two ECUs, one as strong and the other as weak
- **Objective:** manipulate an ECU while shielding the fact that an ECU is compromised

- The attacker first monitors the traffic of the weak ECU
- Once obtained the transmission period, stop the weak and use the strong instead



The Bus-Off Attack

- The bus-off attacks is a denial of service attack that exploits how ECUs access the bus
- K.-T. Cho and K. G. Shin, “Error Handling of In-Vehicle Networks Makes Them Vulnerable,” in Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. ACM, 2016
- The goal of the attacker is to compromise the network and in particular multiple healthy (i.e., non compromised ECUs) with a minimal number of messages
- Notice that this is very different from flooding

Bus-Off Attack: Adversary Model

SPRITZ
SECURITY & PRIVACY
RESEARCH GROUP

UNIVERSITÀ
DEGLI STUDI
DI PADOVA

- The attacker can compromise an in-vehicle ECU either physically or remotely to gain its control
- We do not require the adversary to reverse engineer messages or checksums to be able to deliver the attack
- Once the ECU is compromised, the attacker can inject messages with any ID, DLC and data on the CAN bus

The Attack: Hint

- The error handling mechanism of CAN bus forces ECUs that measure a certain number of errors to go bus-off
- The attacker exploits this feature of CAN bus to iteratively isolate ECUs
- Injecting attack messages, the adversary coerces the TEC of an uncompromised/healthy victim ECU to continuously increase
- Eventually, the attacker forces the error confinement to force the victim or even the entire network to shutdown

The Attack

- Suppose that a victim V periodically sends a message M
- The attacker can deliver a successful bus-off attack if all conditions

  are fulfilled

  - **C1:** Using the same ID
  - **C2:** Transmitting at the same time as M
  - **C3:** Having at least a bit that is dominant in the attack message while being recessive in M (all preceding bits are equal)

The Attack

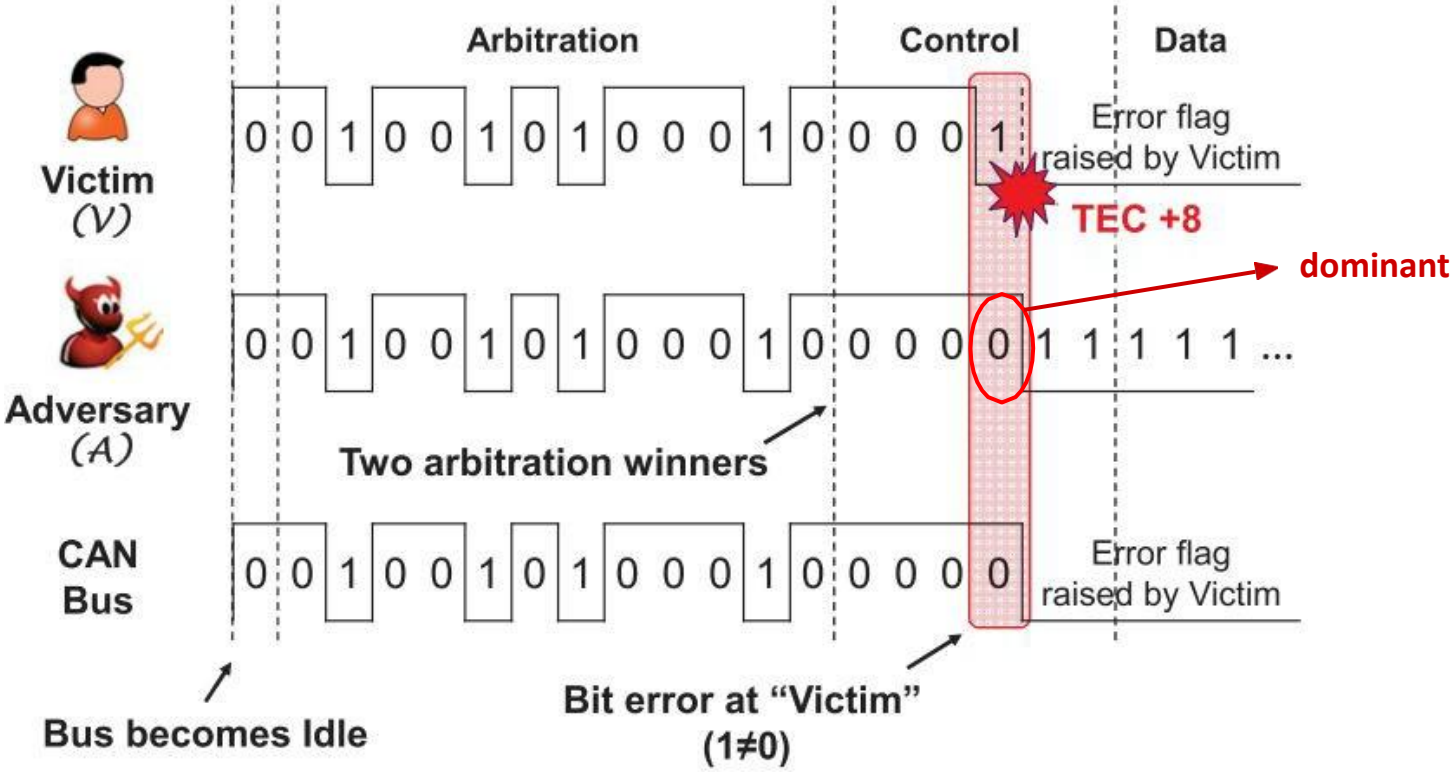
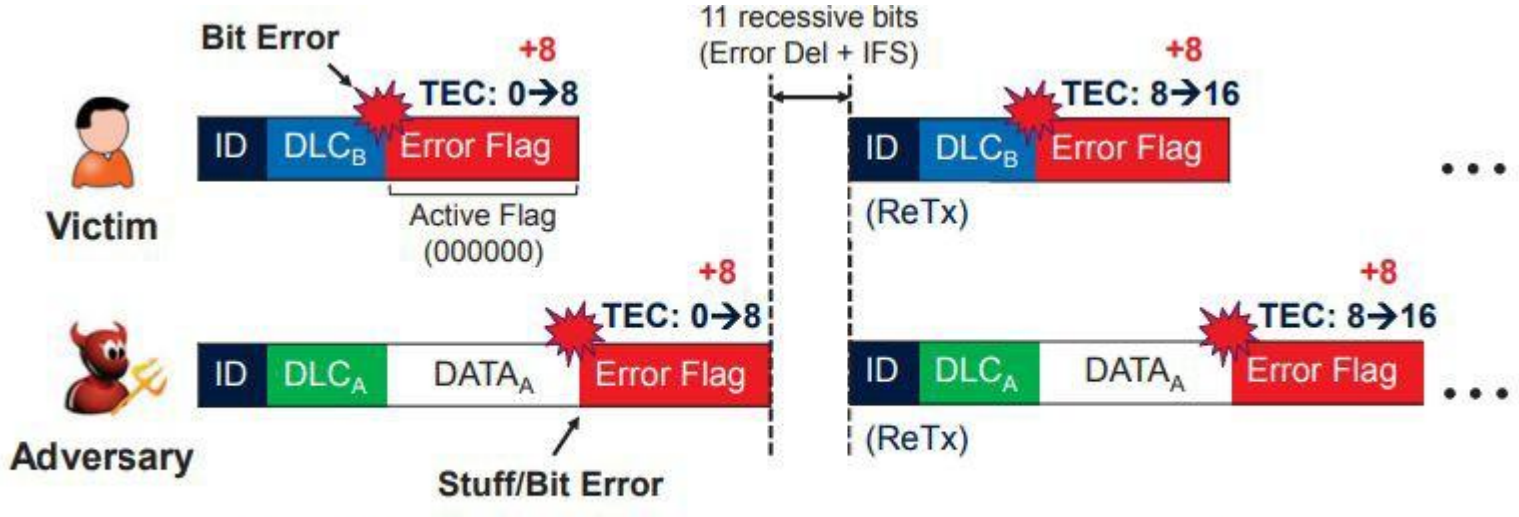


Figure from the original paper: "Error handling of in-vehicle networks makes them vulnerable"

Attack Phases

- **Phase 1: Victim in Error-Active**
- Both adversary and victim start in error-active mode, and the attacker targets one of the victim's periodical messages
- The attacker sends the malicious message and the victim's TEC increases
- The attacker's TEC also increases, due to the stuff or bit errors triggered at the adversary node
- Both nodes automatically retransmit the failed message

Phase 1



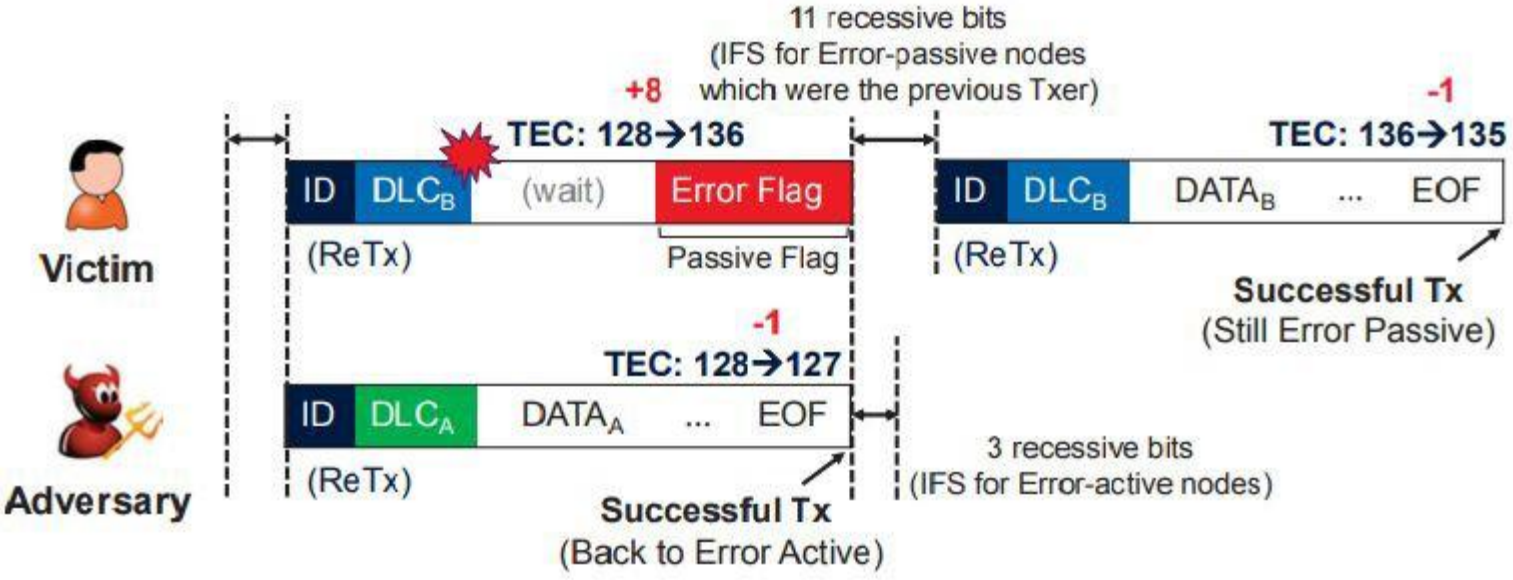
(a) Phase 1 – Victim in error-active mode.

- Thanks to the automatic retransmission, the attackers bring the victim to error-passive by sending a single message

Attack Phases

- **Phase 1 to 2**
- After 16 retransmissions, both victim's and attacker's TEC=128 → error-passive
- As a consequence, the victim attempts the delivery of a passive error flag with 6 recessive bits
- The attempt to transmit this flag persists until the end of the attacker's frame, after which it is successful (TEC = TEC -1)
- Due to the successful transmission, the adversary does not go to error-passive

Phase 1 to 2



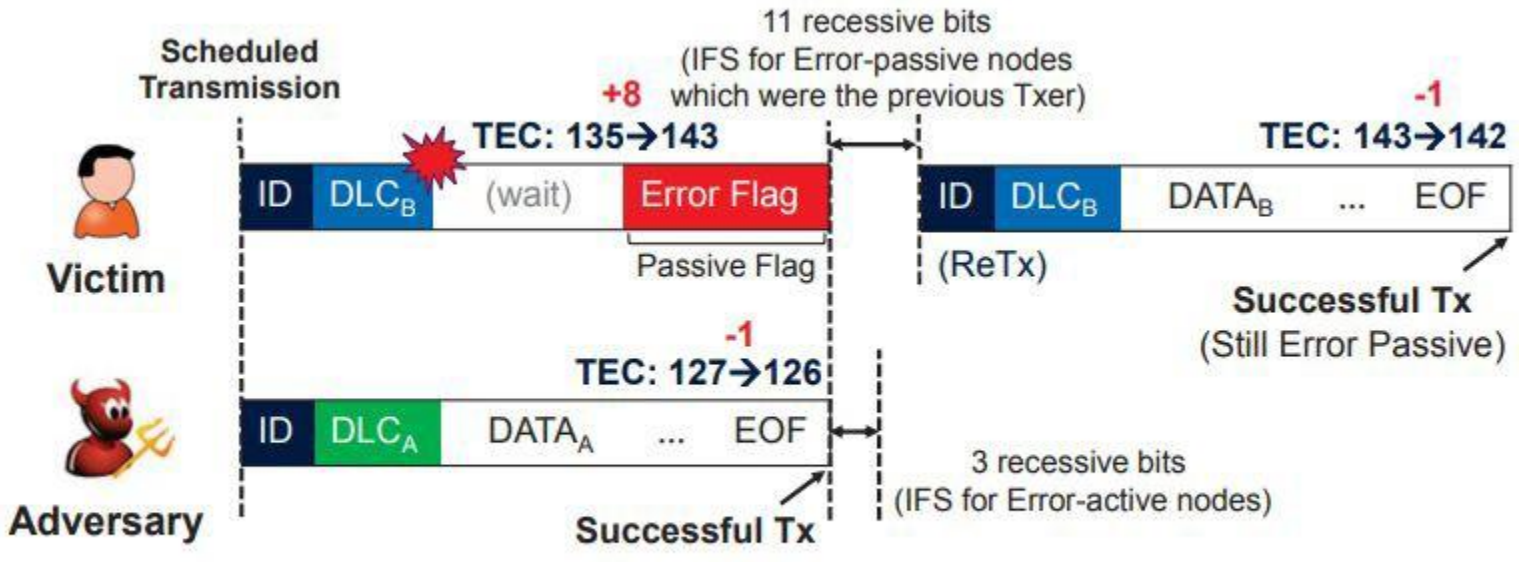
(b) Transition from Phase 1 to 2.

- TEC of victim and attacker differ

Attack Phases

- **Phase 2: Victim Error-Passive**
- Once the scheduled interval of the target message is elapsed, V retransmits M again
- At the same time, the adversary reinjects malicious M
- Since the victim is in error-passive, the attacker's TEC decreases by 1, whereas the victim's increases by 7 (+8 -1) thus maintaining the error-passive
  - Notice that error passive nodes have longer IFS than error active, thus difficult for the attacker to synchornize
- The attacker iterates until the victim's TEC>255, i.e., bus-off

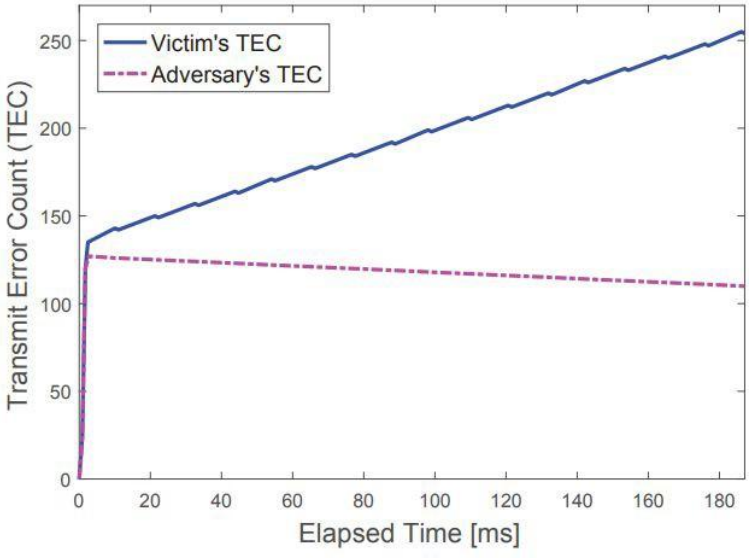
Phase 2



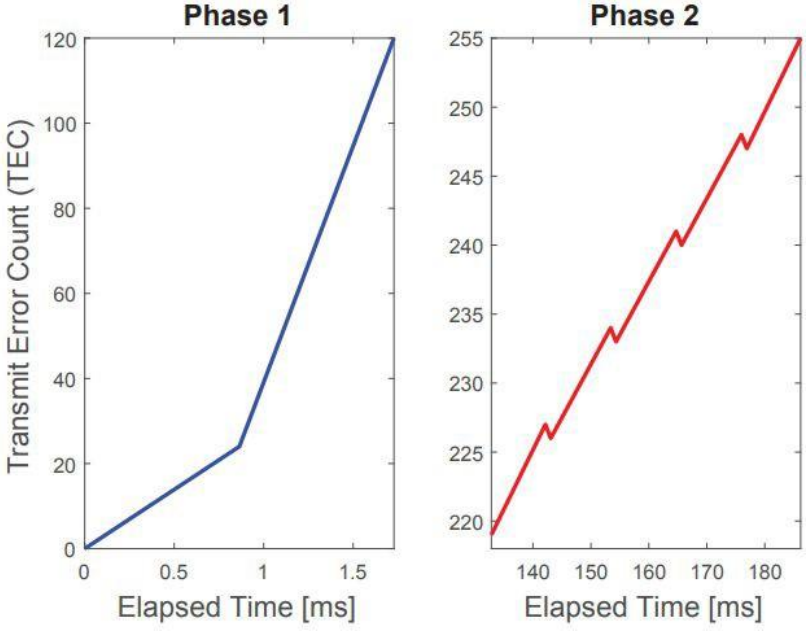
(c) Phase 2 – Victim in error-passive mode.

- TEC of victim and attacker differ

TECs Behavior



- TEC of victim and attacker



- TEC of victim

Important Note

- Remember that CAN ID do not contain actual information on the transmitter
- The ID contains information that define the time interval and priority (i.e., safety-critical) of messages
- The attacker should hence target low-value IDs (dominant) to disconnect possibly safety-critical ECUs
- Examples: acceleration, braking

Where to Modify Frames

- In order to satisfy **C3**, the attacker needs to have a dominant bit where the victim has a recessive one. All preceding bits are equal
- Since IDs should be the same (**C1**), the dominant bit should be either in control or data fields
- CRC and ACK are set by the CAN controller, and cannot be altered by the attacker
- As CAN messages usually have at least one and non-zero data values, modify either DLC or data values to all 0s
- Given that DLC is constant over that for each CAN ID, the attacker can learn the value and modify the DLC accordingly

Difficulties in Setting the ID

- Each ECU cannot acquire the IDs of all received messages
- Only of those that pass through its message filter
- We can distinguish hence between *received* message and *accepted* message
- The attacker can only read the ID from accepted messages, thus meeting the requirements on the same ID depends on the **filter** of the victim ECU
- If there is no filter then done

Non-Empty Message Filter

- The compromised ECU only accepts messages with a certain (range of) ID
- This however does not represent a restriction for the attacker in compromising it
- The attacker can directly modify the message filter of the compromised ECU via software commands
- For some controllers, the configuration mode can be triggered via user instruction through the Serial Peripheral Interface

Tx Synchronization

- Knowing everything about ID and how to modify the message, the attacker needs to know *when* to send it
- Precise synchronization is essential iteratively for the success of the attack
- If there is as little as one bit de-synch, there would be no two arbitration winners ⟶ attack fails
- The attacker may leverage the periodicity of CAN frames and send a message every *T* (estimated by observing the traffic) seconds

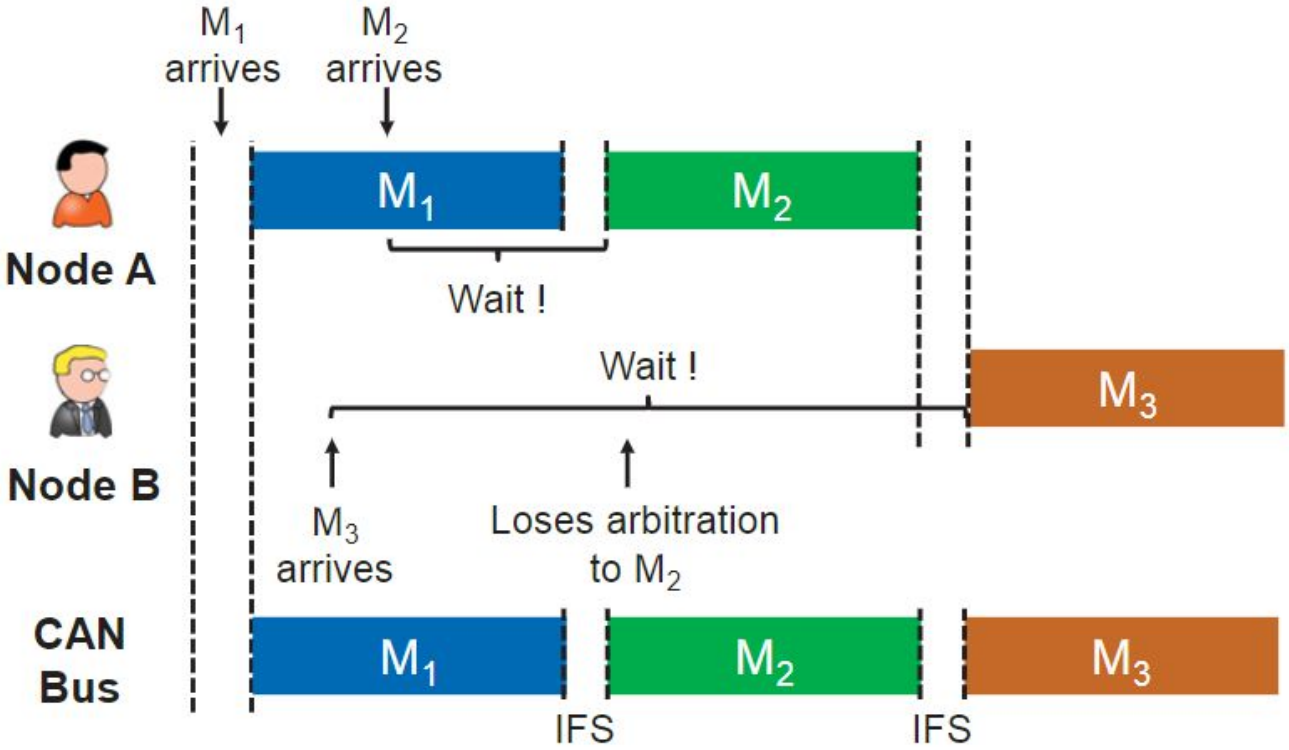
Preceded ID

- Although CAN messages are periodic, jitter may impair the effectiveness of the attack
- To overcome this issue the attacker can exploit the fact that *nodes which have either lost arbitration or had new messages buffered while the bus was busy attempt to transmit their messages as soon as the bus becomes idle*
- Given a node with ID M, we define its *preceded ID* as the ID of the message that completed its transmission right before the start of M

Preceded ID

- Although CAN messages are periodic, jitter may impair the effectiveness of the attack
- To overcome this issue the attacker can exploit the fact that *nodes which have either lost arbitration or had new messages buffered while the bus was busy attempt to transmit their messages as soon as the bus becomes idle*
- Given a node with ID M, we define its *preceded ID* as the ID of the message that completed its transmission right before the start of M

Preceded ID



M_3 always needs to wait for M_1 and M_2

Preceded ID

- Since message IDs and hence priority never changes, the preceded ID gives the attacker a way of predicting when to send the message
- Transmit time = 3 bits after the completion of the message with preceded ID
- If the target message has no preceded ID, the attacker can create one and send both the preceded ID and the attack message

Fabrication of Preceded ID

SPRITZ
SECURITY & PRIVACY
RESEARCH GROUP

UNIVERSITÀ
DEGLI STUDI
DI PADOVA

- The injection timing, quantity, and content of fabricated preceded ID messages is fundamental for success
- Regarding timing, it is fundamental to inject the message right before the target one and we still have the problem with jitter over periodicity
- The only source of randomness is jitter, which follows a Gaussian distribution with zero mean and sigma standard deviation

Fabrication of Preceded ID

- The successive times at which the attacker receives the victim message can be expressed as

victim transmit time

expected transmit time

delay

$$t_n^{adv} = t_n^{vic} + \mathbb{D} = t_{orig} + J_n + \mathbb{D}$$

$$t_{n+1}^{adv} = t_{n+1}^{vic} + \mathbb{D} = t_{orig} + T + J_{n+1} + \mathbb{D},$$

expected transmit time second message

Fabrication of Preceded ID

- We can then compute the next victim's transmit time as

only this is a random variable

$$t_{n+1}^{vic} = t_n^{adv} + T - \mathbb{D} + J_{n+1} - J_n = t_n^{adv} + T - \mathbb{D} + J^*,$$

- For fabrication of the preceded ID, the attacker has to start the transmission of its preceded ID messages(s) before the target and make the bus busy until it becomes sure that the attack and taget messages would synchronize

Fabrication of Preceded ID

- We can hence write the following constraints for the attacker

start of sending fab. msg.

$$1)\ t_{fab} < \min(t_{n+1}^{vic}) = t_n^{adv} + T - \mathbb{D} + \min(J^*)$$

$$2)\ t_{fab} + \mathbb{H} > \max(t_{n+1}^{vic}) = t_n^{adv} + T - \mathbb{D} + \max(J^*)$$

$$3)\ \mathbb{H} = \kappa\mathbb{F} > \max(J^*) - \min(J^*),$$

duration of bus holding

duration of preceded ID msg.

Fabrication of Preceded ID

- J* is a bounded random variable, with boundaries approximated as $|max(J^*)| = |min(J^*)| \simeq \mathbb{I}\sqrt{2}\sigma_v$

  attack parameter — measurable

- Setting I=3 provides 99.73% confidence interval and I=4 a 99.99% confidence interval
- To fully exploit the fabricated preceded IDs, the adversary needs to start injecting them prior to $t_n^{adv} + T - \mathbb{D} - \mathbb{I}\sqrt{2}\sigma_v$

Number of Messages

- To satisfy the third constraint, the adversary needs to occupy the bus for a duration of $\max(J^*) - \min(J^*) = 2\sqrt{2}\mathbb{I}\sigma_v$
- This can be done by sending 1 or more preceded ID messages
- An adversary aiming at minimizing the number of injections should maximize F
- To this aim, the adversary can exploit the bit stuffing rule and send messages with the most stuffed bits
- With L=DLC = 8, $\mathbb{F}^* = (8L + \boxed{44} + \lfloor 8L/4 \rfloor)/S_{bus} = 124/S_{bus}$

exterior to data

Number of Messages

- Based on this, the number of preceded ID need can be expressed as $\kappa = \left\lceil \frac{\max(J^*) - \min(J^*)}{\mathbb{F}^*} \right\rceil = \left\lceil \frac{2\sqrt{2}\mathbb{I}\sigma_v S_{bus}}{124} \right\rceil$

- For sigma = 0.025ms, S_{bus}=500Kbps an adversary only needs 1 message with I=3 and hence 99.73% confidence
- To raise confidence to 99.99% (i.e., I=4) the adversary needs two messages

Content of Preceding ID Messages

- The adversary needs to decide which ID to use for fabricating the preceded ID messages
- If only one preceded ID is needed, the attacker can use the next seemingly free ID
- To be as elusive as possible, such ID can be changed at every attack attempt choosing among the least frequently sent on the bus
- If two preceded IDs are needed, the adversary can choose one from the pool as before, but carefully select the second one to have higher priority than the target one

Preventing Bus-Off Attacks

SPRITZ
SECURITY & PRIVACY
RESEARCH GROUP

UNIVERSITÀ
DEGLI STUDI
DI PADOVA

- The defense mechanism can leverage two features of the bus-off
- **F1 (in phase 1):** at least two consecutive errors occur during the transmission of frames. Thus, we watch for consecutive error frames with an active error flag
- **F2 (in phase 2):** at the time when the error-passive victim's TEC increases, a message with the same ID will be successfully transmitted by some ECU on the bus

WeepingCAN

- The bus-off attack is easily detectable by a good intrusion detection system
- Other approaches include physical layer intrusion detection system due to differences in hardware between attacker and victim
- Secure hardware may prevent the attack by detecting and blocking the local transmission of the attacker
- CAN bus authentication may also prevent these kinds of attacker, however not likely to be implemented

WeepingCAN

- *WeepingCAN* is a variation of the original bus-off attack, stealthier than the original one
- Differences:
  - The attacker disables the retransmission of the attack message with the same ID as the victim
  - The attacker causes *recessive bit errors*
  - The attacker does not fabricate preceded ID messages
  - The attacker randomizes bit errors

WeepingCAN

- The attack message has the victim's ID, bit-time, and prefix bits until a random position where the victim is dominant but the attacker is recessive
- As for the original bus-off, both attacker and victim have TEC = 0 at the beginning
- The goal of weepingCAN is to avoid exhibiting temporal differences due to the error state transition → being stealthy

Attack Cycle

- The attacker synchronized with the victim using the same approach of the original bus-off
- The attacker however disables the retransmission of the attack packet
- The attacker injects the attack message
- The attacker' CAN controller sends an error-active flag due to the bit error in the attacker's packet, which increases both attacker's and victim's TEC by 8
- The victim retransmits the original packet and TEC=TEC-1
- Victim and attacker decrease their TEC for other transmissions

Disable Retransmissions

- Disabling retransmission allows to remove an easily detectable feature, i.e., the consecutive retransmissions of the same packet (F1)
- The attack is now composed by an active error flag + successful retransmission
- Two ways to avoid retransmissions
  - Disable automatic retransmission for all messages: the attacker writes to a control register that the ECU should never retransmit, neither with arbitration
  - Abort transmission on transmit error: most CAN controllers can raise an interrupt in transmit errors and allow to abort retransmission in the interrupt handler

Recessive Injection

- The attacker injects a recessive bit when the victim “contains” a dominant bit
- The attacker increases its TEC by 8, and sends an error active flag
- The victim increases its TEC, then successfully retransmits (TEC = TEC - 1)
- The attacker must identify additional messages it can transmit to keep its TEC lower than that of the victim (otherwise could go bus off!)
- The attacker needs to transmit at least 1 more message than the victim for each attack packet while the victim should transmit less than 7 other packets in this time frame , or its TEC will recover

Basic Attack Strategy

- Suppose that the attacker and victim both have a single message they can send
- Suppose A sends its packet with one fifth the period of V
- Thus, for every attack message, A transmits 5 messages while V transmits 1 $\rightarrow$ TEC_A=3, TEC_V=7
- V can be forced to error passive with 19 attack messages
- A needs to be careful not to reach error passive before V goes bus-off!
- A can decide to skip some injections

Randomized Bit Positions

- The second feature making bus-off identifiable is the presence of successful messages over the passive error flags of the victim
- Furthermore, the attacker may always use the same packet for the attack
- Therefore, it might be good for the attacker to have the possibility to randomize packets
- In particular, to randomize the position of the error bit

Randomized Bit Positions

- The DLC of most messages is a fixed constant that can be discovered by offline analysis of a CAN trace
- The number of dominant bits in DLCs vary between 1 and 3, so the entropy of a random injection is low → bad!

- A clever attacker may inject recessive error in the data field
- The data field often encodes a number that does not change quickly because of physical constraints or an event identifier that comes from a limited set of event

- If A can identify a deterministic pattern → inject error in the data

What About Bit Flipping?

- The idea of injecting errors via controlling CAN frames requires synchronization with the victim and carefully crafting attack frames
- In the end, the whole purpose of the attacker is to flip some bits on the can BUS to lead the victim ECU to bus off
- With minimum 32 injected fake frames (or bits?) we can force an ECU to bus off
- If the attacker can gain control over the bus, they can just set arbitrary bits and cause errors

Silently Disabling ECUs

- Suppose we include an IDS in our CAN bus that can detect arbitrary IDs appearing on the bus
- It contains a whitelist of all arbitration IDs the ECUs on the bus are programmed to transmit
- Such a whitelist allows us to detect but flip attackers attempting to cheat the arbitration process or transmit never seen before message (needed for preceded ID)

Attacker Model

- A *stealthy* attacker is able to read and write on the bus, with the goal of disabling and spoofing ECUs without being detected
- Flipping bits is a non-precise technology, so the attacker has a chance from 0-100% of flipping bits 0⟶1 and 0-100% of flipping 1⟶0
- Remember that 0 is dominant, so if anyone is transmitting a 0 the 0 wins
- We make no assumption about the attack vector, as long as it can flip bits
- We can also consider a blind attacker, which has no read access to the bus, only write

Stealthy Bus-Off

- The original bus-off attack produces errors on the bus by performing a single 1 to 0 flip
- Each attack increments the error counter by 8 → doing it 32 times over 32 messages forces the victim to bus-off
- We want to design an attacker with the same objective but that does not produce errors on the bus
- We also assume that attacker can arbitrarily flip bits rather than needing to win the arbitration process

Stealthy Bus-Off

- Main idea: when a CAN error frame is interrupted, it produces another error frame incrementing its TEC by 8 everytime it is interrupted
- This means that an attacker able to flip bits can force to victim to bus-off with a single message by flipping 32 bits
- Once the victim is in bus-off, it will stop sending error frames regardless of the fact that any of them have actually been delivered to the bus
- We need to ensure that no receive errors are produced and ensure the rest of the bus processes eventually sent malicious messages (spoofing the victim)

Probabilistic Attacker

- Flipping individual bits on a serial bus is inherently an hard problem, particularly changing the dominant state to a recessive state
- The attacker hence must be able to:
  - successfully complete a CAN packet (11 flips in a row – EOF, IFS,delim)
  - successfully interrupt an error frame (at least 1 bit flip in 6)
  - transmit a known arbitration ID (for blind attackers)

Completing a CAN packet

- The end of a CAN frame requires 11 recessive bits to be sent
- If a receiver sees a dominant bit in this sequence, it will send a frame check error frame
- This means that if the attacker cannot successfully perform 11 dominant to recessive bit flips in a row, the rest of the bus will stop functioning until an error frame completes or the bus-off state is reached → error from IDS
- If the victim is put onto bus-off or error passive before before the final 11 bits, then the victim will be either silent or transmitting a 1 → the attacker needs to do nothing to end a CAN frame

Completing a CAN packet

- Caveat: need to perform at least 16 bit flips before the CRC check
- The number of bit flips necessary to transmit an updated CRC depends on all prior successful bit flips
- Much like the final 11 bits, this is trivial if the bus is in a error passive state
- This translates in the ability to flip 16 to 64 bits

Interrupting an Error Frame

- The ability to interrupt an error frame represents the ability of the attacker to continue transmitting a message without letting an error frame complete and have the victim message be dropped
- This translates into flipping at least one every 6 bits
- First caveat: ensure that receiver errors (RCR, ACK, stuffing, frma echeck) do not occur
- Second caveat: interrupting another ECU attempting to transmit a message
- Both of them are avoidable by having the ECU in bus-off by the arbitration process or, failing that, in passive error