Testing the security and reliability of Automotive Ethernet using fuzzing

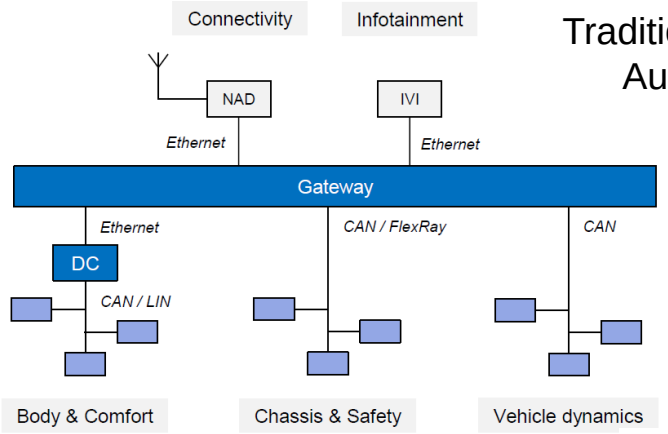
Razvan Petre (razvan.petre@spirent.com)

Detroit, September 24th, 2019

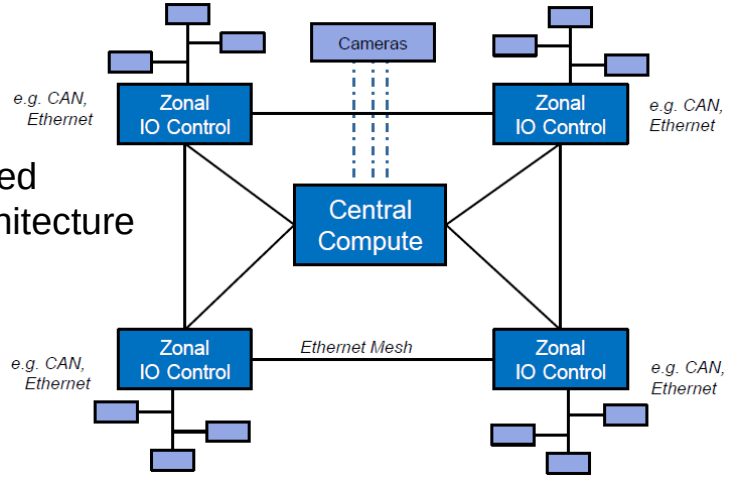
Trends in Automotive Networks

Past, Present and Future of Automotive Architectures

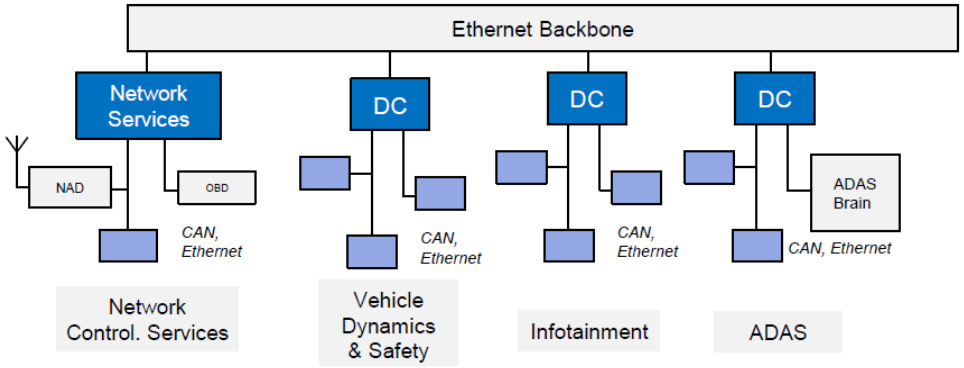
Traditional or Central Gateway Automotive Architecture



Zonal-based Automotive Architecture



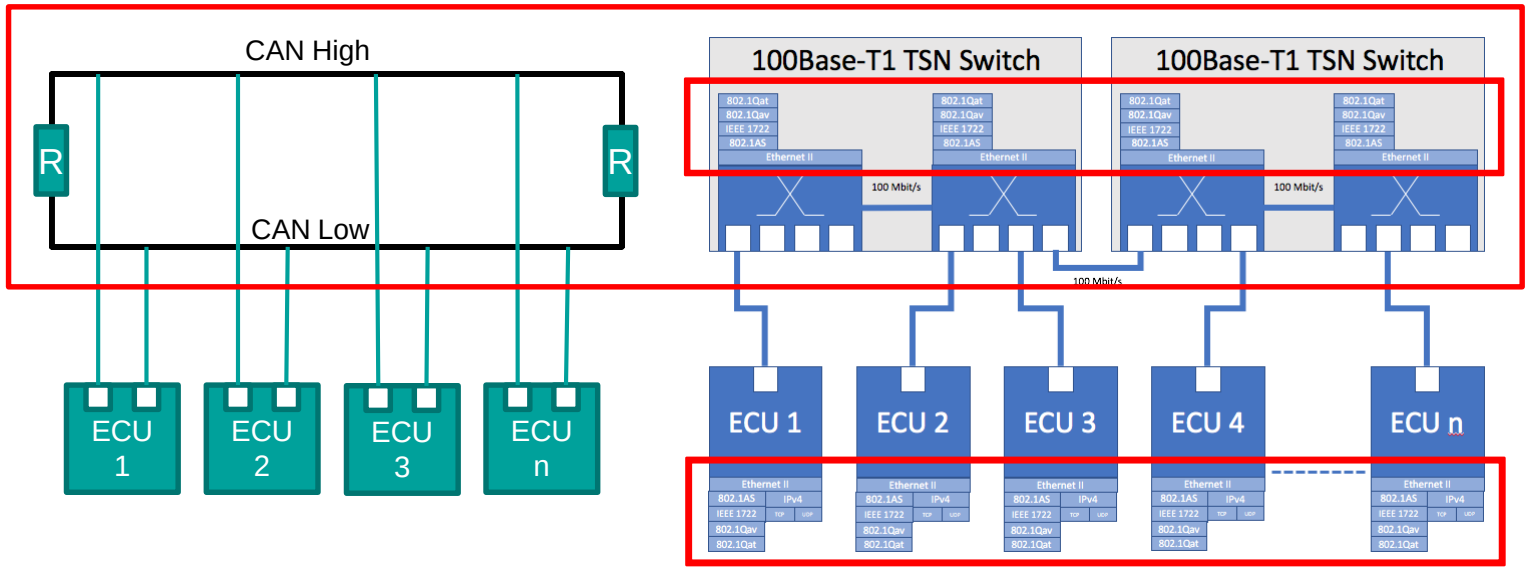
Domain-based Automotive Architecture



* text and diagrams taken from IEEE P802.1DG v0.3, July 2018

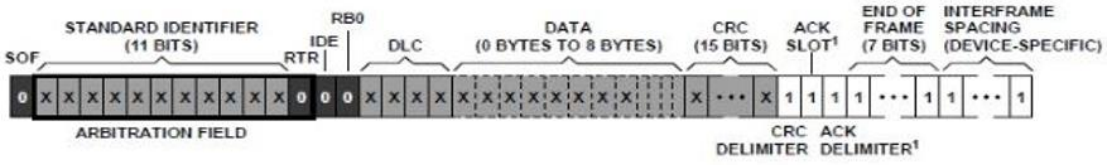
CAN vs Ethernet

Key differentiator: The Transport Network

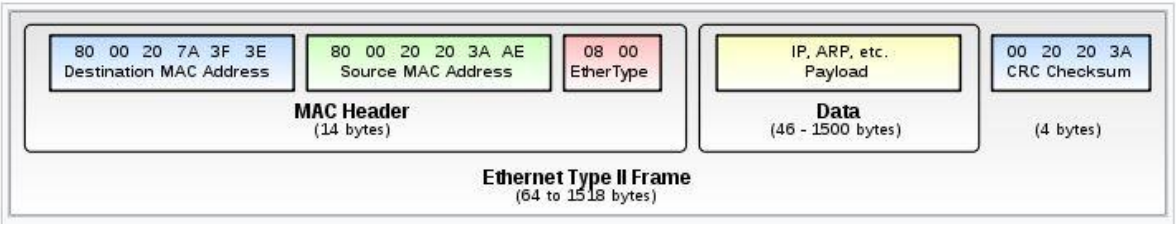


- Standard Data Frame

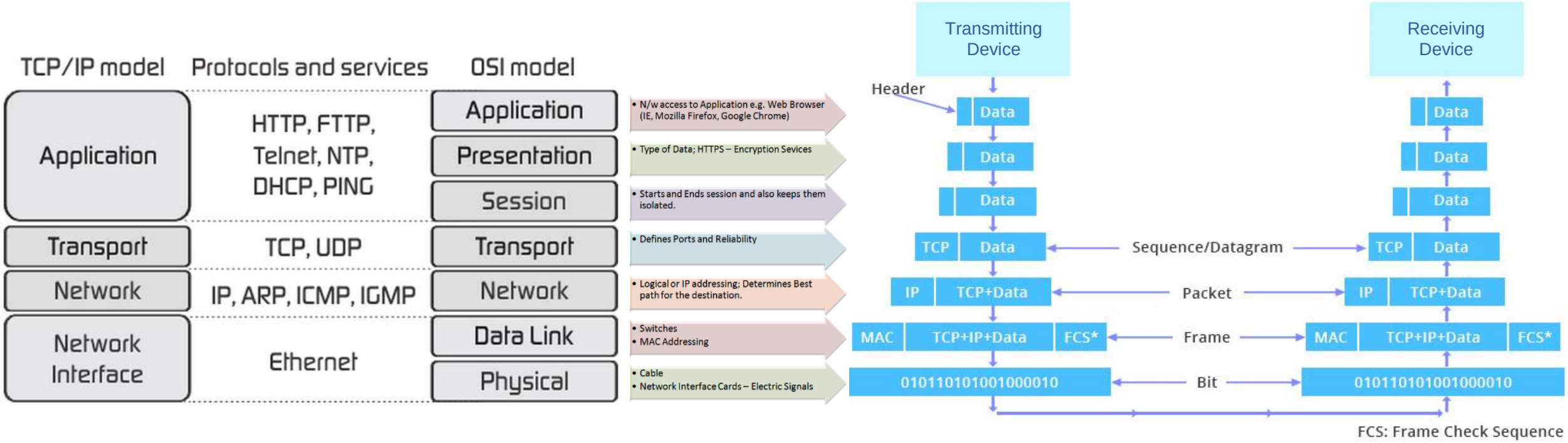
CAN 2.0A Frame Format



Ethernet Frame Format



TCP/IP and OSI Models



- Each layers communicates with its peer layer only
- Data encapsulation

Testing types, techniques and tactics

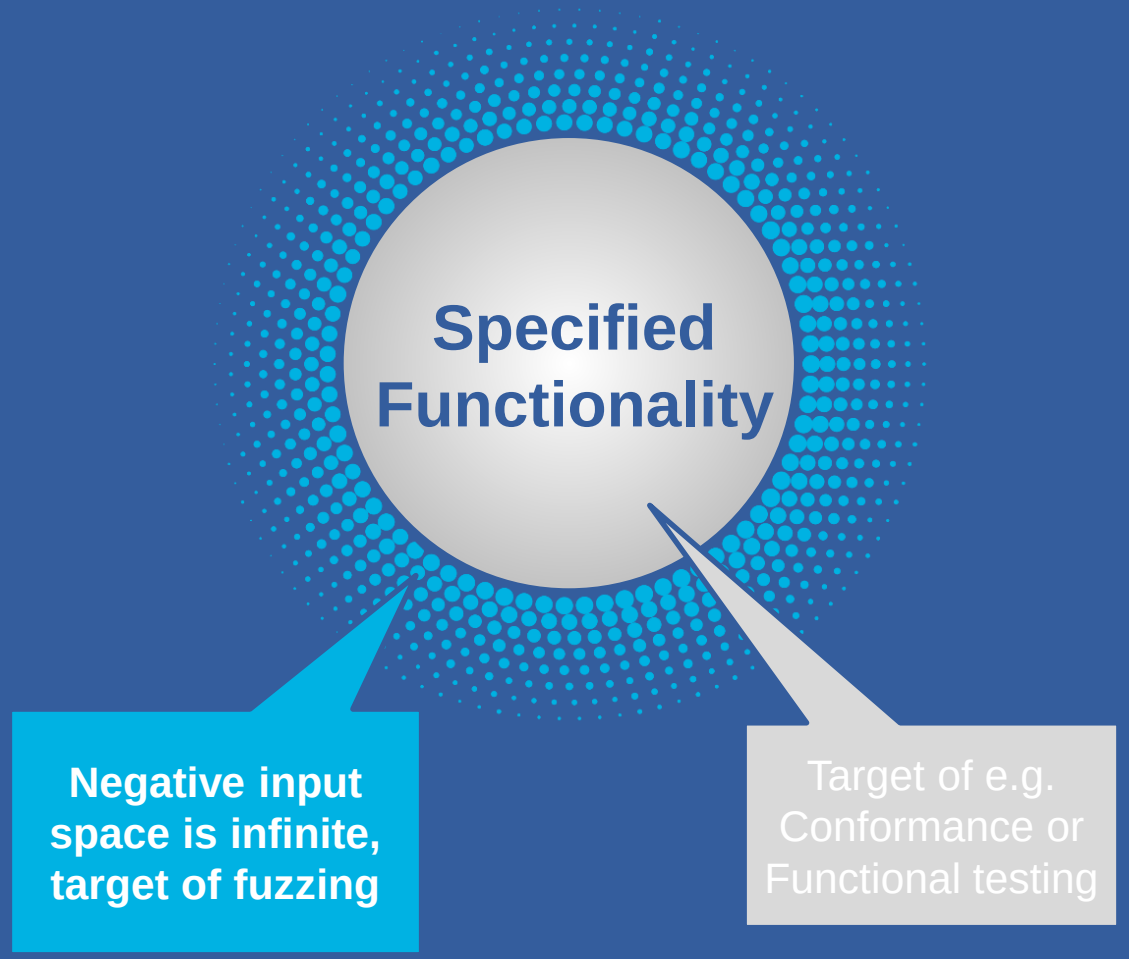
- Conformance testing
- Positive vs Negative testing
- Device vs System level testing
- Functional testing
- Performance: load, stress, robustness testing
- Security testing
- Black-box vs White-box testing

e.g. OPEN TC8 v2.0, Aug 2017, 823 Tests

49	ARP
22	ICMPv4
53	IPv4
56	Network Auto Configuration
53	UDP
222	TCP
90	DHCPv4
105	SOME/IP Server
144	SOME/IP ETS

What is fuzzing

- It is an **automated** software **testing technique**
- It involves providing **invalid, unexpected, or random data** as inputs to a computer program.
- An effective fuzzer generates semi-valid inputs that:
  - are "**valid enough**" to pass the parser, but
  - are "**invalid enough**" to expose corner cases



Who is using Fuzzing?

A brief history of fuzzing

1950s

- Testing programs with random inputs started in the **1950s,** when data was still stored on punched cards.
- In **1983**, "The Monkey“ was developed, a tool that would generate random inputs for classic Mac OS applications, such as MacPaint.
- The term "fuzzing" originates from a **1988** class project, taught by Barton Miller at the University of Wisconsin.
- In April **2012**, Google announced ClusterFuzz, a cloud-based fuzzing infrastructure for security-critical components of the Chromium web browser.
- In September **2016**, Microsoft announced Project Springfield, a cloud-based fuzz testing service for finding security critical bugs in software.
- In November **2018** Google revealed that OSS-Fuzz, the company's automated fuzzing service/bot, has identified and reported over 9,000 vulnerabilities in widely used open source projects since 2017.

2019

Types of Fuzzing

From dumb to smart fuzzers

Dumb

Smart

- Random-based fuzzing
  - No protocol knowledge
- Data structure aware fuzzing
  - A data model, a grammar, or a protocol description is required
- Mutation-based fuzzing
  - It generates inputs by modifying existing data (e.g. utilize a pcap capture as seed)
- Generation-based fuzzing
  - It uses fuzzing heuristics to generate data based on a specified model

Types of Fuzzing

…continued

Dumb

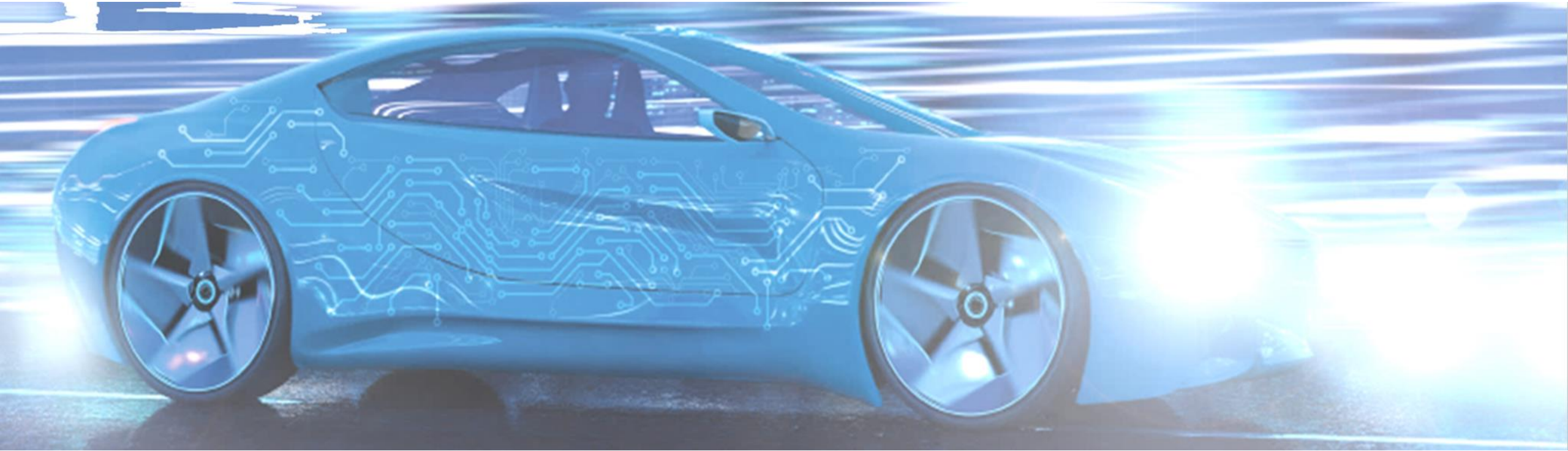
Smart

- Black-box fuzzing
  - The fuzzer is unaware of internal program structure
- White-box fuzzing
  - The fuzzer leverages program analysis to systematically increase code coverage or to reach certain critical program locations
- Data fuzzing
  - Only the input data is modified, the order of the messages is always correct
- Behavioral fuzzing
  - Order or the messages is modified
  - Timing constraints can be tested as well
  - Illegal transitions in the protocol state machine

Fuzzing Automotive Ethernet Protocols

Choosing the right fuzzing ingredients for IVN testing

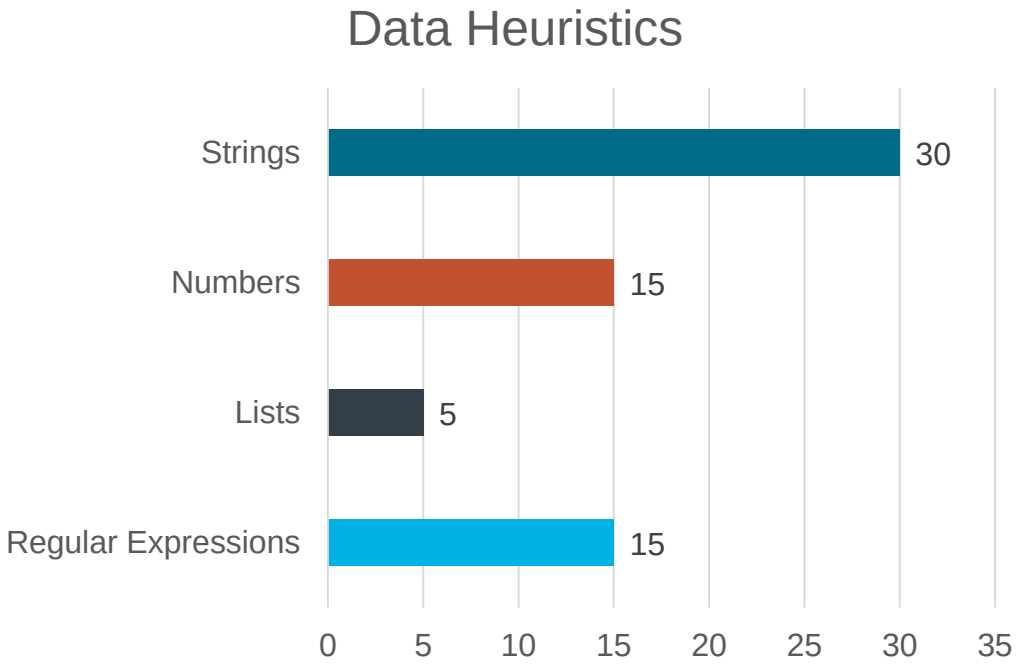
- Data aware fuzzing
- Both mutation- and generation- based fuzzing
- Black-box fuzzing
- Both data and behavioral fuzzing

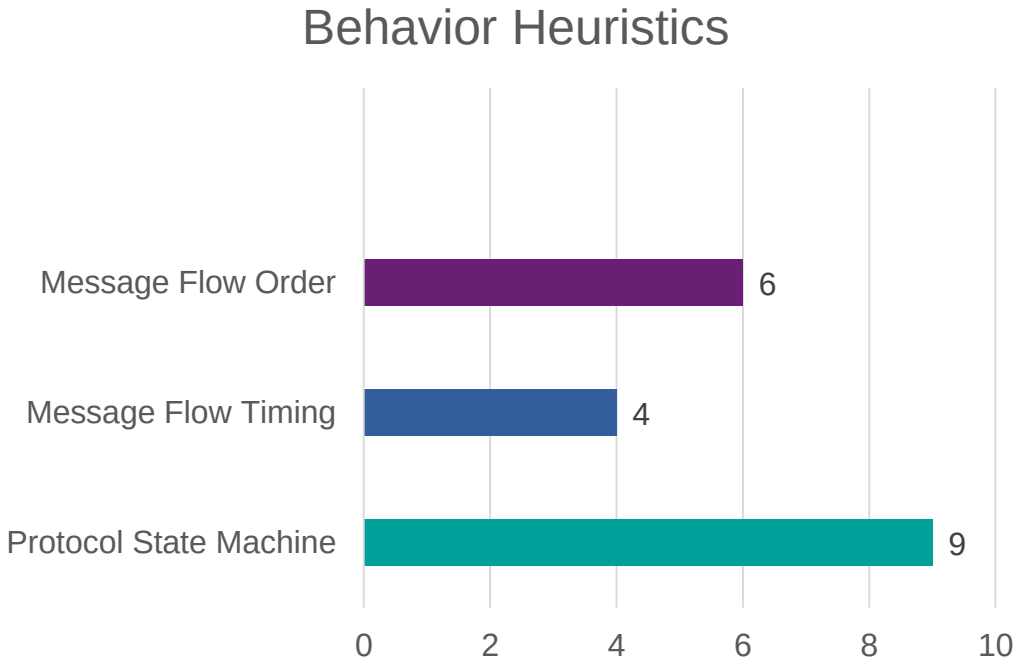


Fuzzing Heuristics

The power of a smart fuzzer

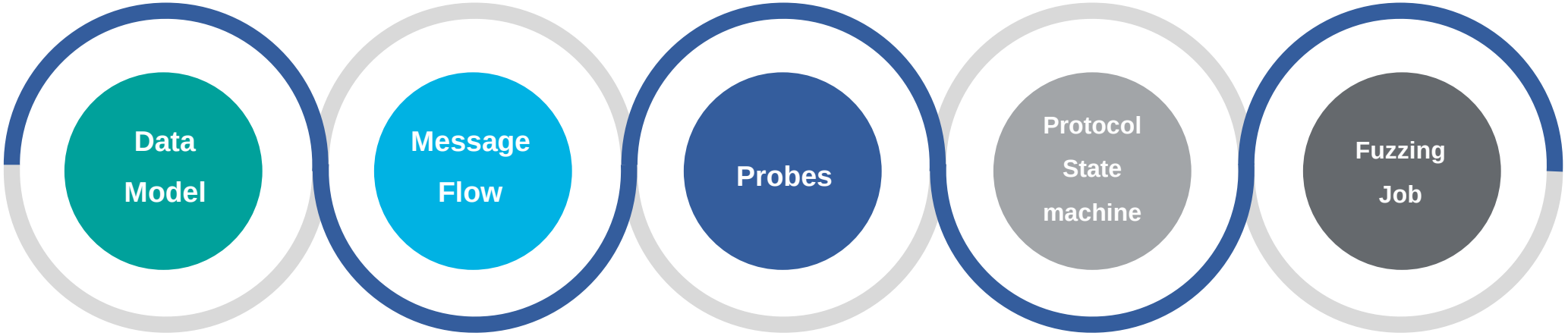
Spirent Automotive Fuzzer offers:





Spirent Automotive Fuzzer

A modularized solution for data and behavior fuzzing



1. Data Model

Contains the definition of the protocol messages

For each field, the fuzzing heuristic can be specified

Fields can be omitted from being fuzzed

2. Message Flow

Describes the message order and any timing restrictions between messages

Used for basic behavior fuzzing

3. Probes

How to check if the DUT is still alive

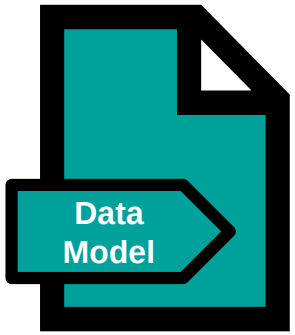
4. State machine

Used for advanced behavior fuzzing

5. Fuzzing Job

Uses all previous 4 components to glue together a fuzzing job

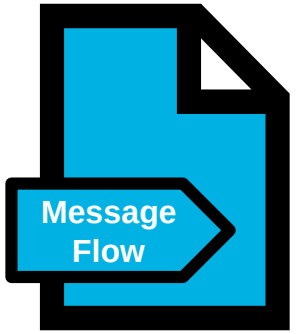
Data model



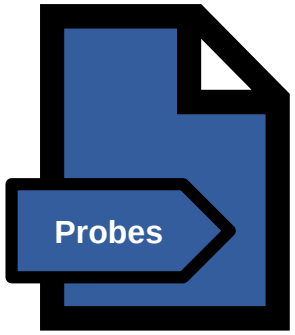
- Describes the structure of the protocol messages
- It is mandatory for the fuzzing engine
- Defines the rules for data fuzzing
  - “none”
  - “full”
  - “default”
  - “comma separated heuristic list”

Message flow

- A file describing:
  - order of the messages
  - any state information (relation between request / responses)
  - timing restrictions between messages
- It is optional and only required for behavior fuzzing



Probes



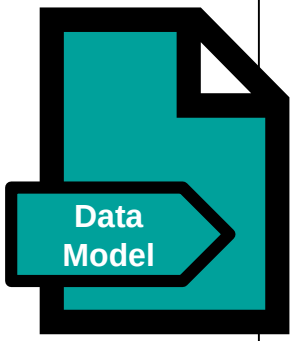
- Black-box testing requires a way to know if the DUT has crashed or not
- Sample probes available:
  - Ping
  - Create TCP connection
  - Terminate TCP connection
  - Receive [template] [after T1] [before T2]
  - Send [Eth Packet] [after T1] & Receive [template] [after T1] [before T2]
  - Run scripts

IPv4 fuzzing

Data fuzzing targeting the protocol stack

IPv4 Header Format

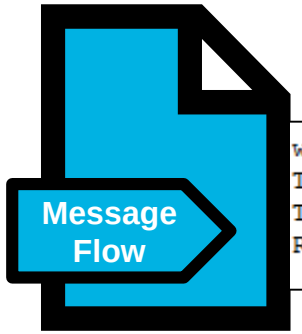
Offsets	Octet	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Version				IHL				DSCP						ECN		Total Length															
4	32	Identification																Flags			Fragment Offset												
8	64	Time To Live								Protocol								Header Checksum															
12	96	Source IP Address																															
16	128	Destination IP Address																															
20	160	Options (if IHL > 5)																															
24	192																																
28	224																																
32	256																																



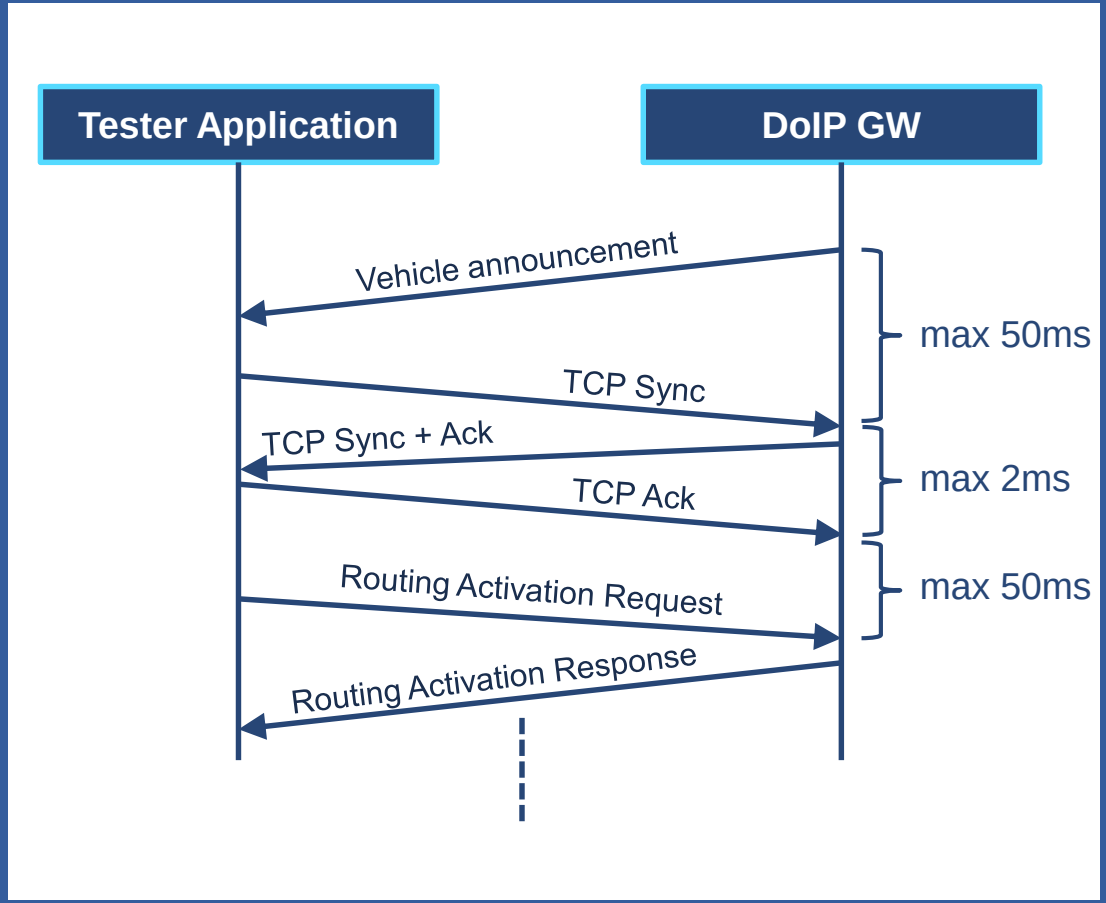

```
Ipv4Header {
   version type="uint4"; fuzz="none",
   headerLength type="uint4"; fuzz="default",
   typeOfService {
      precedence type="uint3",
      delay type="bool1",
      throughput type="bool1",
      reliability type="bool1",
      reserved type="uint2"
   } fuzz="none",
   totalLength type="uint16"; fuzz="default",
   identification type="oct2"; fuzz="none",
   flags {
      reserved type="bool1"; fuzz="full",
      doNotFragFlag type="bool1"; fuzz="full",
      moreFragsFlag type="bool1"; fuzz="full",
   },
   fragmentOffset type="uint13"; fuzz="default",
   ....
```

DoIP fuzzing

Behavior fuzzing targeting the protocol stack & application




```
wait="VehicleAnnouncement"
TcpConnectionStart maxTime="50ms"; fuzz="default"
TcpConnectionDone maxTime="2ms"; fuzz="default"
RoutingActivationReq maxTime="50ms"; fuzz="default"
```



ARXML – AUTOSAR XML

Description of communication networks

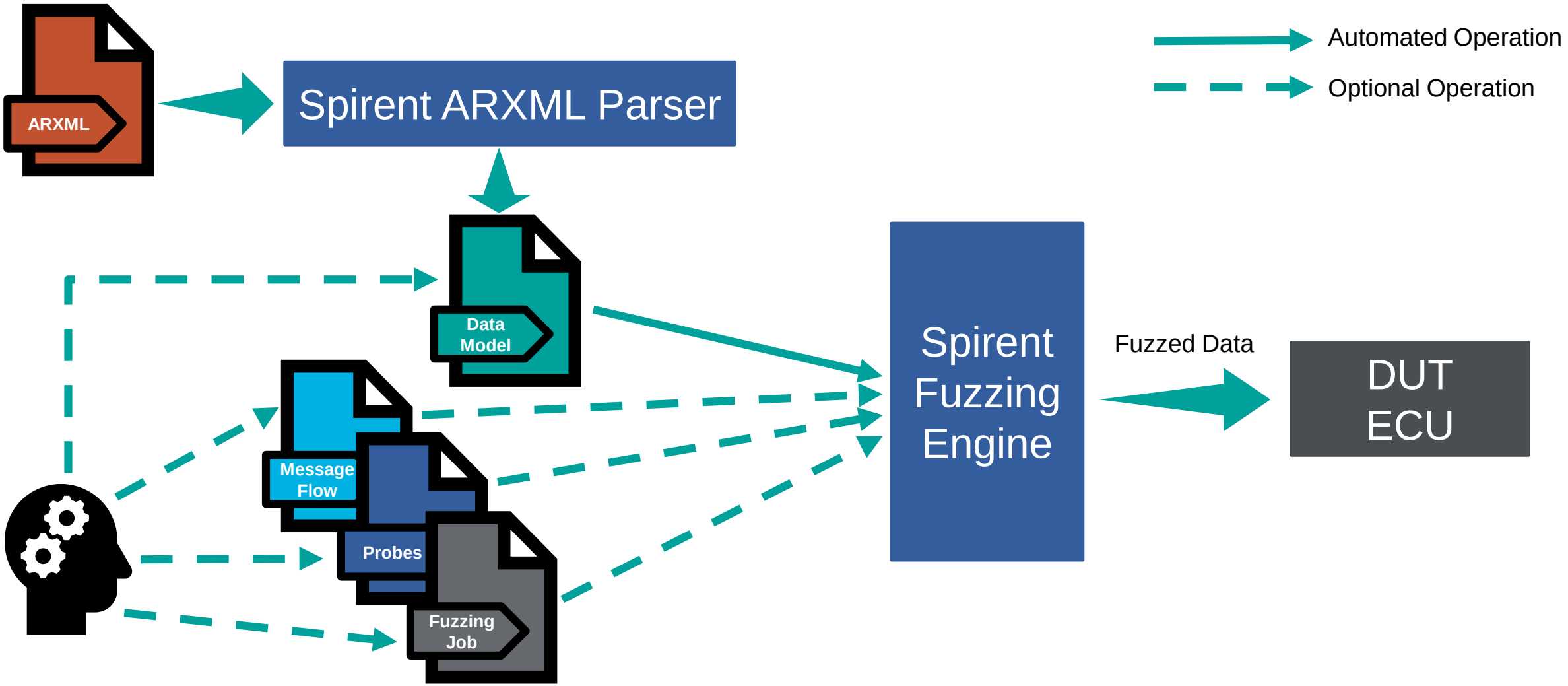
- A file used to represent the configuration of AUTOSAR based ECUs
- It contains description for
  - **Clusters**
  - **ECUs**
  - **Frames**
  - **PDUs**
  - **Signals**
  - **Data Types**
- Similar to DBC, LDF or FIBEX files


```xml
<ETHERNET-COMMUNICATION-CONTROLLER UUID="11111-aaa-aaa-aaa">
  <SHORT-NAME>EthernetCnt_Ctrl</SHORT-NAME>
  <ETHERNET-COMMUNICATION-CONTROLLER-VARIANTS>
    <ETHERNET-COMMUNICATION-CONTROLLER-CONDITIONAL>
      <COUPLING-PORTS>
        <COUPLING-PORT UUID="11111-aaa-aaa-aaa">
          <SHORT-NAME>AAM_EthernetCnt_Ctrl</SHORT-NAME>
          <COUPLING-PORT-SPEED>100000000</COUPLING-PORT-SPEED>
          <PHYSICAL-LAYER-TYPE>BROAD-R-REACH</PHYSICAL-LAYER-TYPE>
          <VLAN-MEMBERSHIPS>
            <VLAN-MEMBERSHIP>
              <DEFAULT-PRIORITY>100</DEFAULT-PRIORITY>
              <VLAN-REF DEST="ETHERNET-PHYSICAL-CHANNEL">/VehicleTopology/Ethernet/Command_and_Control</VLAN-REF>
            </VLAN-MEMBERSHIP>
            <VLAN-MEMBERSHIP>
              <DEFAULT-PRIORITY>101</DEFAULT-PRIORITY>
              <VLAN-REF DEST="ETHERNET-PHYSICAL-CHANNEL">/VehicleTopology/Ethernet/Diagnostics</VLAN-REF>
```

```xml
<SOCKET-ADDRESSS>
  <SOCKET-ADDRESS UUID="11111-aaa-aaa-aaa">
    <SHORT-NAME>Command_and_Control_AppServer</SHORT-NAME>
    <APPLICATION-ENDPOINT UUID="11111-aaa-aaa-aaa">
      <SHORT-NAME>Command_and_Control_AppServer</SHORT-NAME>
      <NETWORK-ENDPOINT-REF DEST="NETWORK-ENDPOINT">/VehicleTopology/Ethernet/..</NETWORK-ENDPOINT-REF>
      <TP-CONFIGURATION>
        <UDP-TP>
          <UDP-TP-PORT>
            <PORT-NUMBER>256</PORT-NUMBER>
          </UDP-TP-PORT>
        </UDP-TP>
      </TP-CONFIGURATION>
    </APPLICATION-ENDPOINT>
    <CONNECTOR-REF DEST="ETHERNET-COMMUNICATION-CONNECTOR">/ECUSystem/..</CONNECTOR-REF>
  </SOCKET-ADDRESS>
```

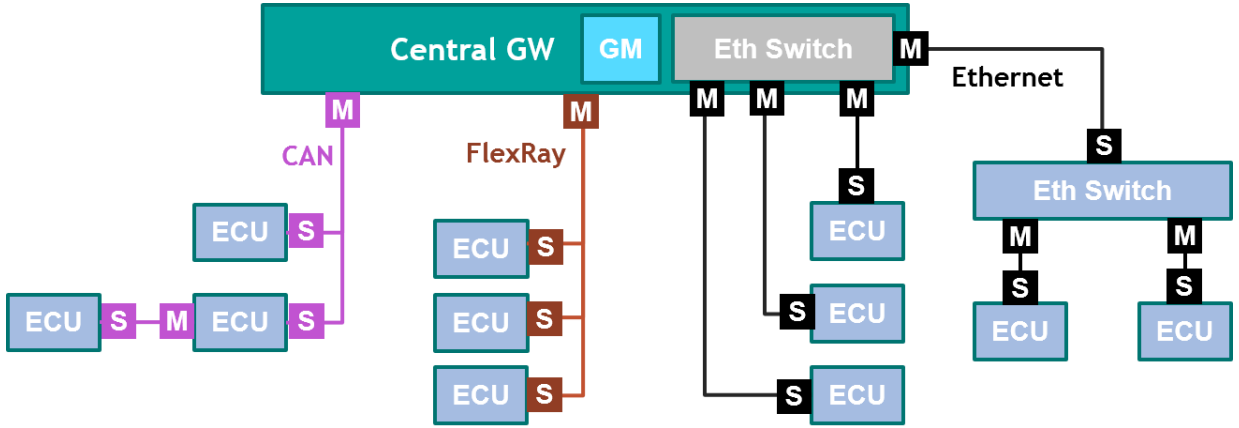
ARXML

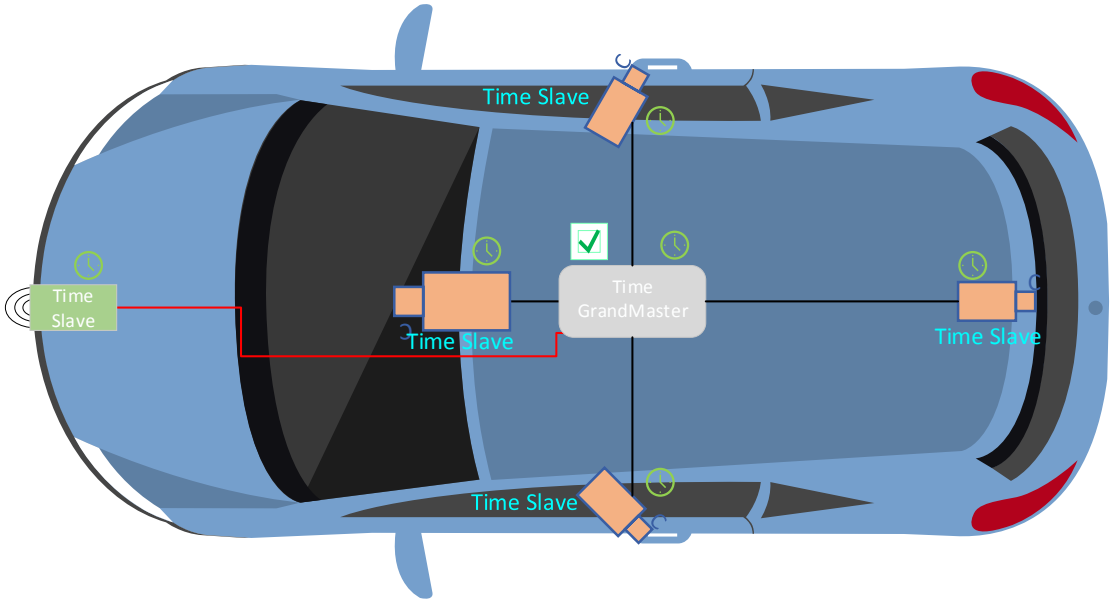
Fuzzing directly from ARXML



Fuzzing CAN based applications

AUTOSAR Time Sync over CAN

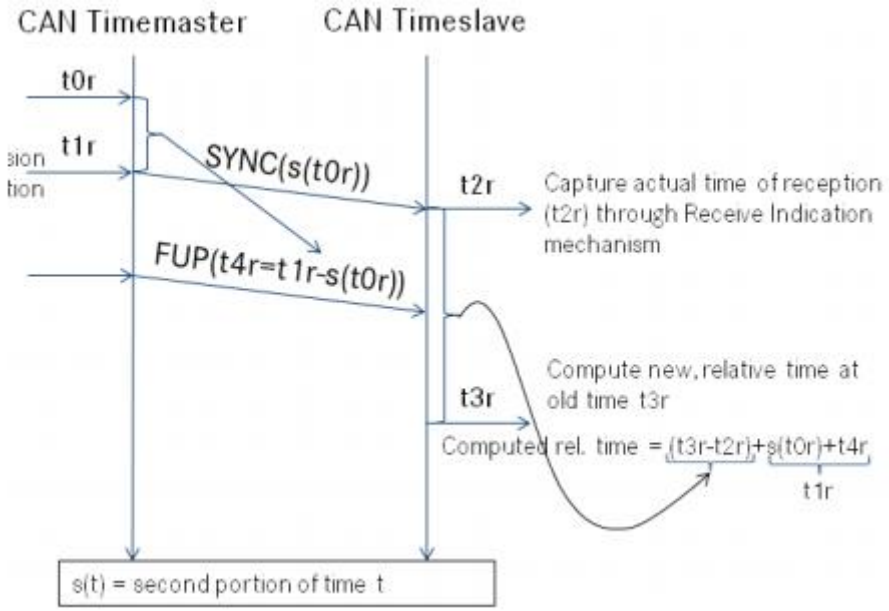




Fuzzing CAN

Data Model

```
SyncWithCrc {
   type type="oct1"; value="0x20"; fuzz="none",
   crc  type="oct1"; value="function_crc"; fuzz="none",
   timeDomain type="uint4"; fuzz="full",
   seqCounter type="uint4"; fuzz="default",
   userDefined type="oct1"; value="0x0"; fuzz="none",
   secondsPart type="uint32"; fuzz="default"
```



[SWS_CanTSyn_00017][
SYNC CRC secured message format:

Byte 0:	*Type* = 0x20
Byte 1:	*CRC*
Byte 2:	*D* = Time Domain 0 to 15 (Bit 7 to Bit 4) *SC* = Sequence Counter (Bit 3 to Bit 0)
Byte 3:	User Byte 0, default: 0
Byte 4-7:	*SyncTimeSec* = 32 bit LSB of the 48 bits seconds part of the time

If `CanTSynUseExtendedMsgFormat` = `TRUE`:

Byte 8-15:	reserved, always 0

AUTOSAR Specification of Time Synchronization over CAN
https://www.autosar.org/fileadmin/user_upload/standards/classic/4-3/AUTOSAR_SWS_TimeSyncOverCAN.pdf

Summary

Best practices & benefits of using fuzzing for automotive industry

- Fuzzing saves time & money, as minimal user input is required
- Fuzzing does not replace the need for conformance, performance or security testing; it rather complements them and brings much better coverage
- Same test methodology and tools can be used for fuzzing Ethernet protocols from L2 to L7, as well as legacy automotive busses
- Data Fuzzing can be used for any protocol
- Behavior Fuzzing for protocols with complex state machines is not recommended, since there is a considerable user input required
- Good Fuzzing Heuristics must be used to reduce the number of generated messages, since ECUs usually have limited resources