RomLibEmu: Network Interface Stress Tests for the CERN RadiatiOn Monitoring Electronics (CROME)

Katharina Ceesay-Seitz, Hamza Boukabache, Marvin Leveneur, Daniel Perrin

Further developers: Amitabh Yadav, Jonas Bodingbauer

20th October 2021

Agenda

- CROME – CERN RadiatiOn Monitoring Electronics
- ROMULUS communication library
- Need for robustness testing
- RomLibEmu – ROMULUS Library Emulation and Test Tool
- Robustness test results
- Conclusion/Outlook

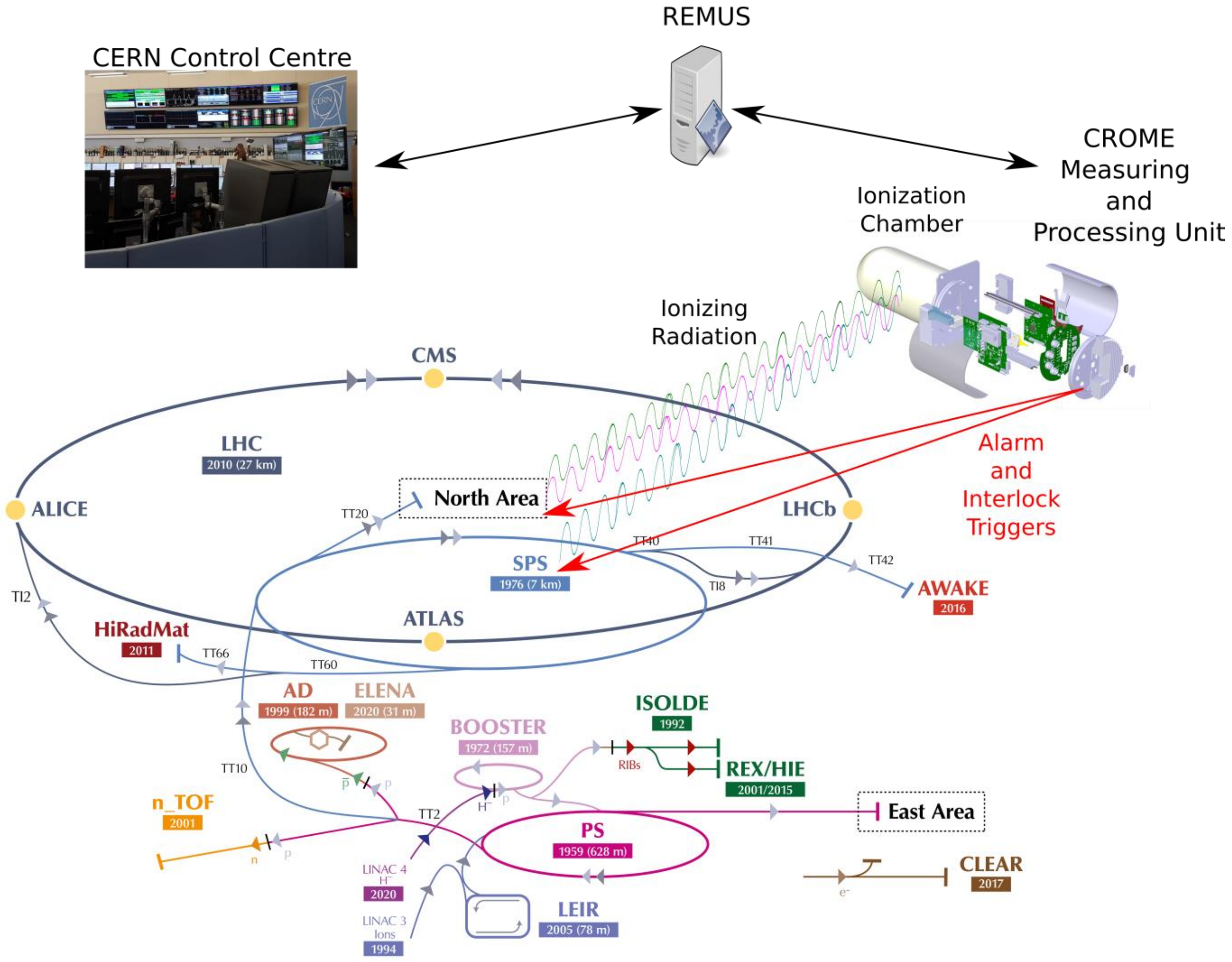
Introduction

Radiation Protection Group

- Responsible for radiological safety of people at and near CERN sites

CROME – CERN RadiatiOn Monitoring Electronics

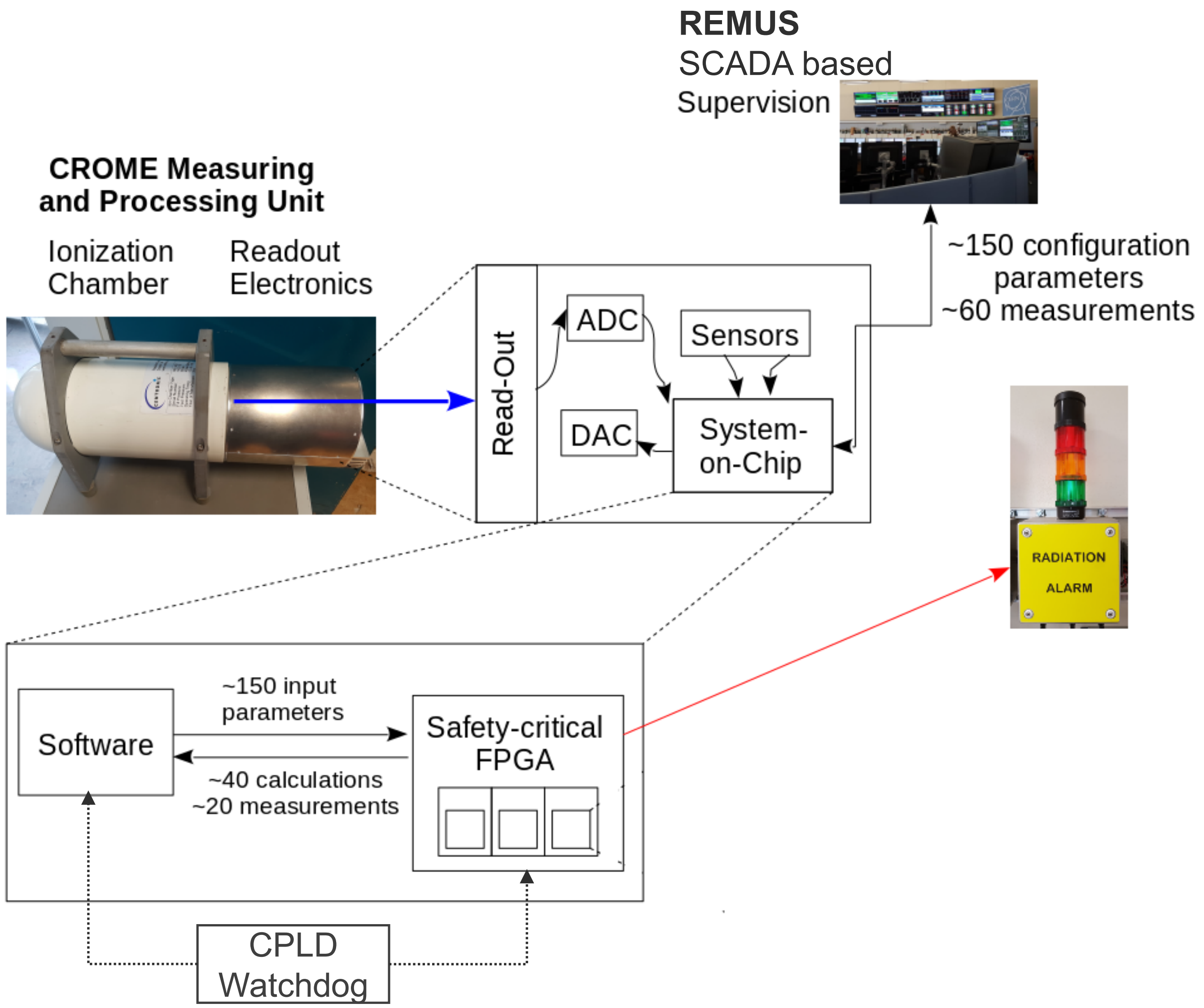
- Measure ionizing radiation in human accessible areas
- Triggers alarms and interlocks in dangerous conditions
- 150 units installed, more to come



[1] Based on E. Mobs, The CERN accelerator complex – 2019, cds.cern.ch/record/2684277

Introduction CROME - CMPU

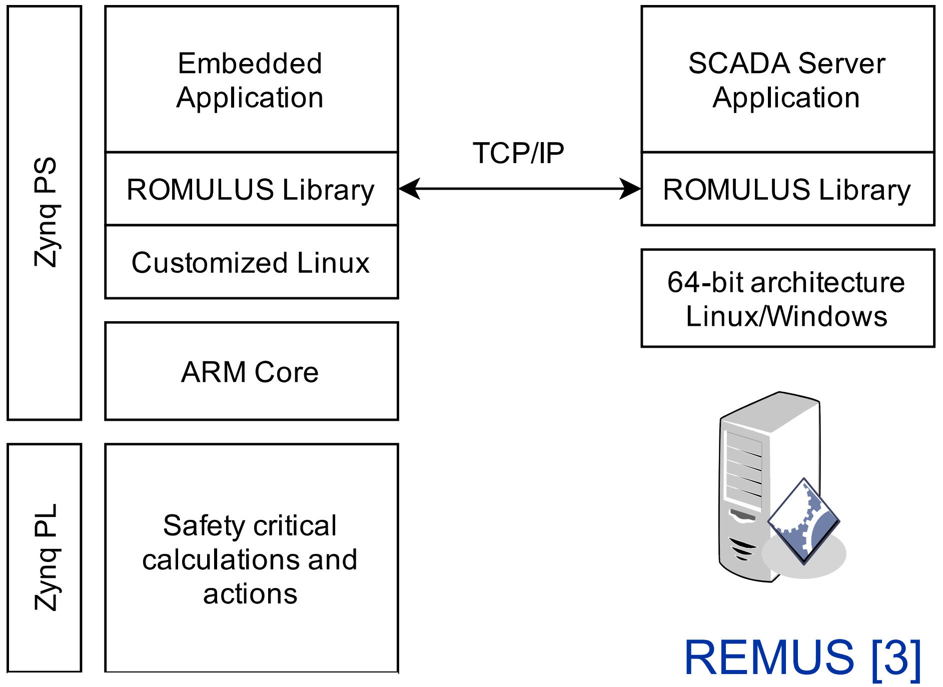
- **Zynq-7000 System-on-Chip**
  - Processing System :
    - 2 32-bit ARMv7 cores
    - PetaLinux based OS (CROMiX)
    - Userspace application in C
    - Custom communication library in C (ROMULUSlib) [2]
  - Programmable Logic
    - Safety critical measurements, calculations and decisions
    - Fault tolerant architecture

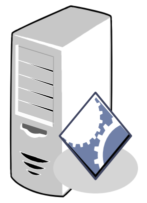


ROMULUS – REMUS communication protocol

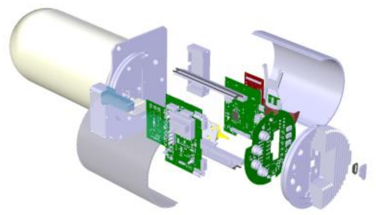
- **ROMULUS library [2]**
  - Used by CMPU application and REMUS server
  - Custom protocol on top of TCP/IP
  - Real time measurement streaming
  - Historical data retrieval
  - Commands to read and **write safety-critical parameter configuration**

→ **Needs to be safe and robust**





REMUS [3]



[2] **A. Yadav**, H. Boukabache, K. Ceesay-Seitz, D. Perrin, “ROMULUSlib: An autonomous, TCP/IP-based, multi-architecture C networking library for DAQ and Control applications”, **MOBR01, this conference**.

[3] A. Ledeul, G. Segura Millan, A. Savulescu, B. Styczen, and D. Vasques Ribeira, “CERN Supervision, Control and Data Acquisition System for Radiation and Environmental Protection”, in Proc. 12th International Workshop on Personal Computers and Particle Accelerator Controls

Robustness Testing

- **Robustness:**
  - According to 610.12-1990 IEEE Standard Glossary of Software Engineering:

  Robustness is “the degree to which a system or component can function correctly in the presence of invalid inputs or stressful environmental conditions”

- **Robust systems:**
  - are stable in unforeseen operating conditions,
  - do not accept invalid input,
  - do not produce faulty output when presented with unexpected input [4]

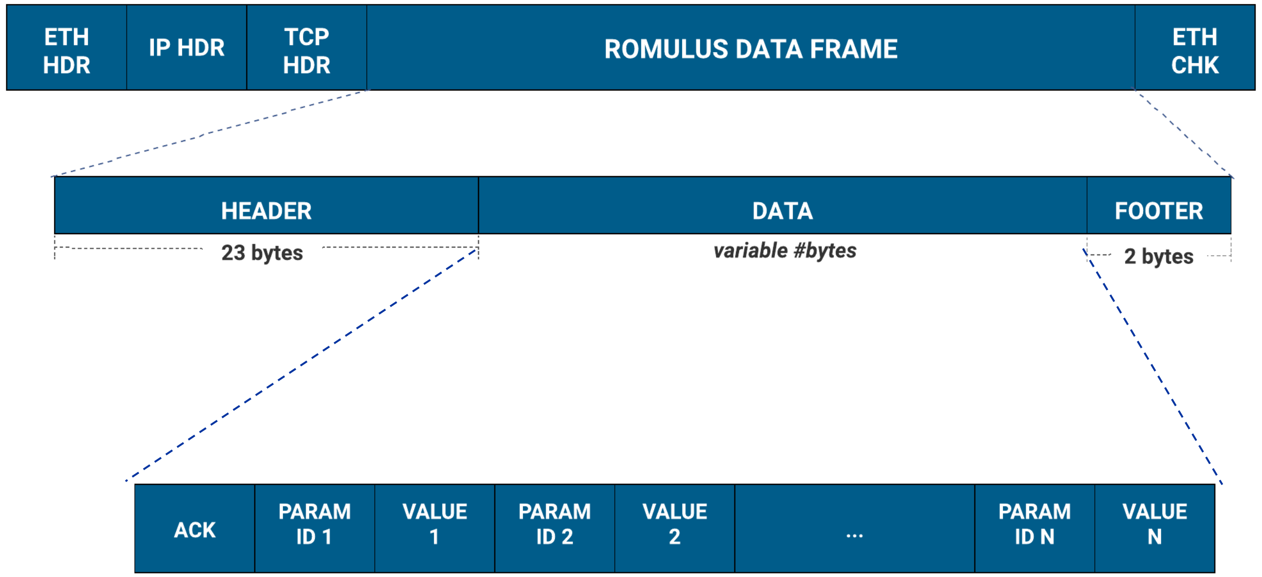
- **Robustness Testing: Evaluate the system’s response to unexpected/invalid inputs.**
  - Common techniques: model-based techniques, fault injection, fuzzing, interception, code changes, injection and mutation testing [5]

[4] S. Shah, D. Sundmark, B. Lindström, S. Andler, “Robustness Testing of Embedded Software Systems: An Industrial Interview Study”, in IEEE Access 4: 1-1, pp. 1859-1871, 2016, 10.1109/ACCESS.2016.2544951.

[5] N. Laranjeiro, J. Agnelo, J. Bernardino, “A Systematic Review on Software Robustness Assessment”, in ACM Computing Surveys, vol. 54, issue 4, nr. 89, pp. 1-65, 2021, doi:10.1145/3448977

RomLibEmu –
ROMULUS Library Emulation and Test Tool

- Functionality tested with various tools
- **RomLibEmu:** Test robustness of
  - ROMULUS library
  - Embedded userspace application
- **Motivation:**
  - Availability
    - Reduce risk of application crashes
  - Reliability/Safety
    - Avoid unpredictable system behaviour due to invalid inputs

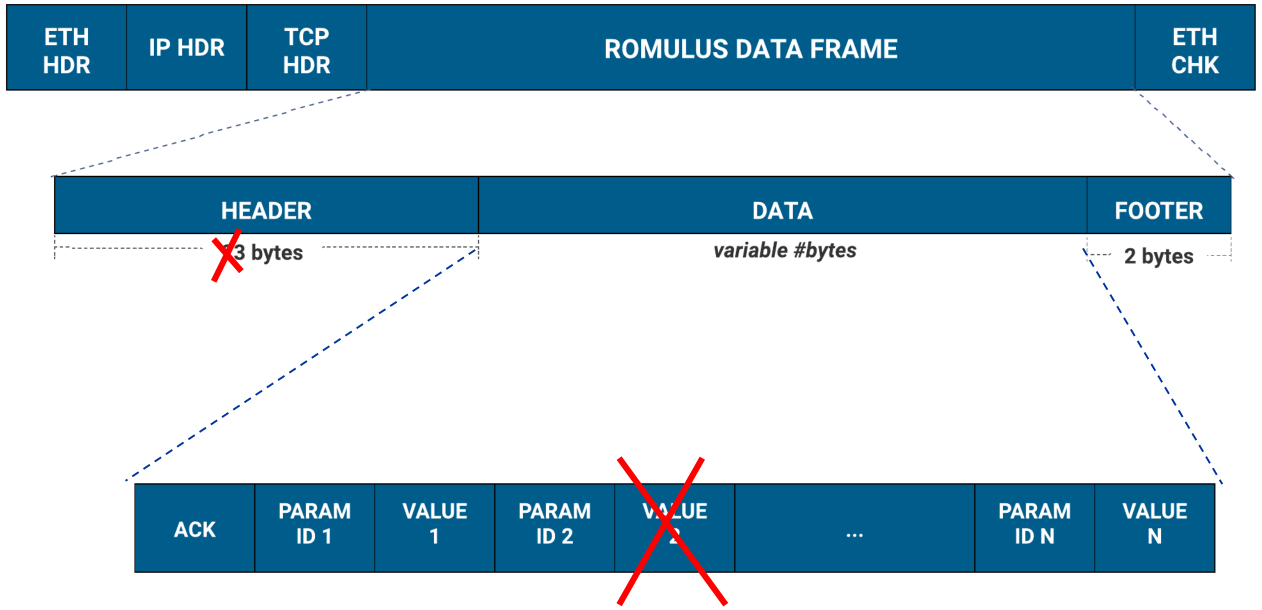


[2]

[2] **A. Yadav**, H. Boukabache, K. Ceesay-Seitz, D. Perrin, “ROMULUSlib: An autonomous, TCP/IP-based, multi-architecture C networking library for DAQ and Control applications”, **MOBR01, this conference**.

RomLibEmu – ROMULUS Library Emulation and Test Tool

- **Invalid packets may arrive due to**
  - A fault in the ROMULUS library
  - A fault in the CMPU application/REMUS driver
  - Incompatible ROMULUS library versions used by CMPU and REMUS
  - Unexpected network traffic
  - Network overload
  - Intentional attacks



[2]

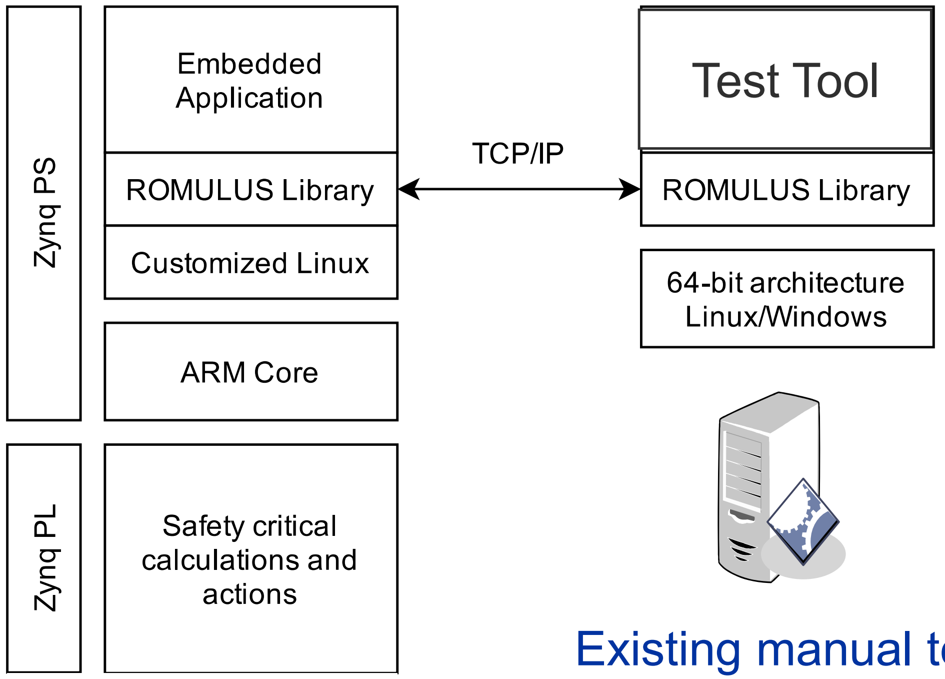
[2] **A. Yadav**, H. Boukabache, K. Ceesay-Seitz, D. Perrin, “ROMULUSlib: An autonomous, TCP/IP-based, multi-architecture C networking library for DAQ and Control applications”, **MOBR01, this conference**.

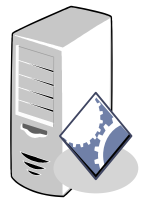
RomLibEmu –
ROMULUS Library Emulation and Test Tool

- **Motivation for RomLibEmu**
  - Existing test tools/kit use ROMULUS library for communication
  - ROMULUS library was built to create protocol conforming messages
  - Cannot create malformed packets intentionally

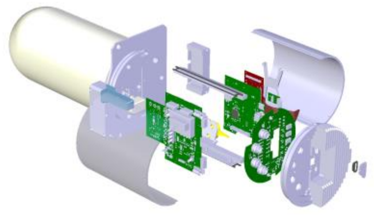
    → No means to test what happens if invalid/unexpected packets are received by CMPU

  - Automated regression testing





Existing manual test tools

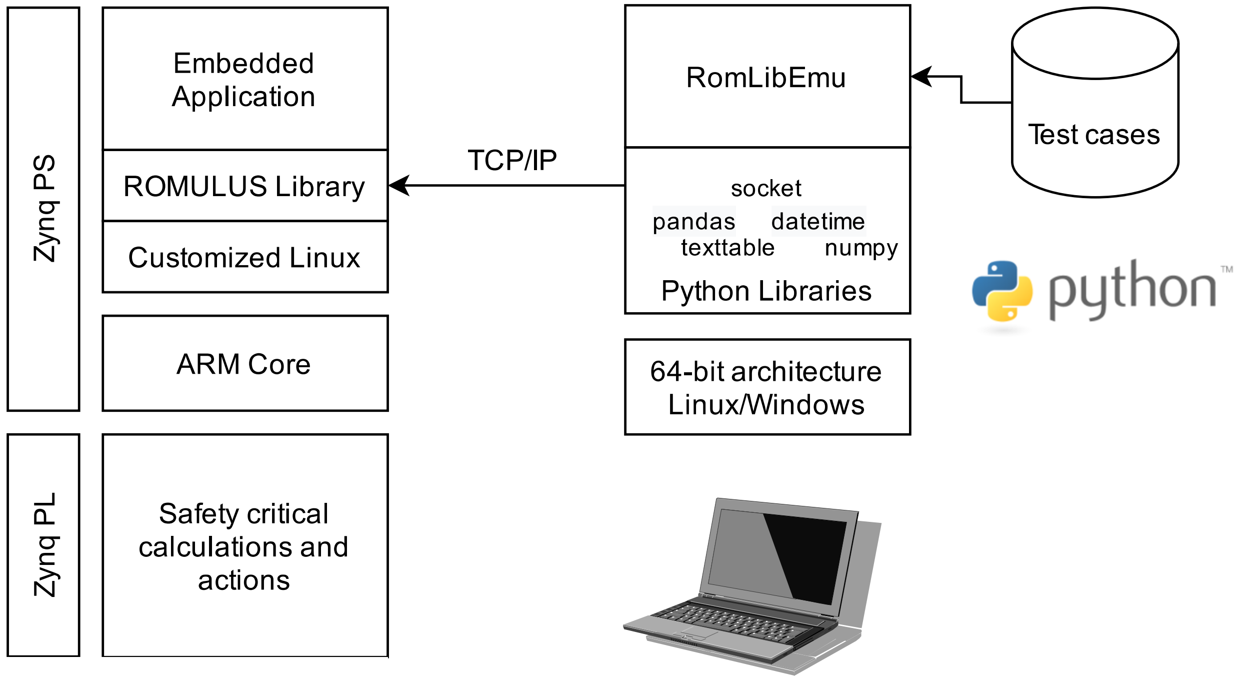


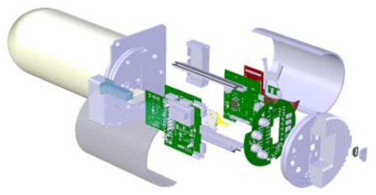
RomLibEmu –
ROMULUS Library Emulation and Test Tool

- Independently developed
  based on protocol specification
- Python 3.7

- Send and receive packets via TCP/IP
- Check for expected response

- Test cases
  - Dictionaries of messages
  - Customized modules/functions
    using RomLibEmu's features

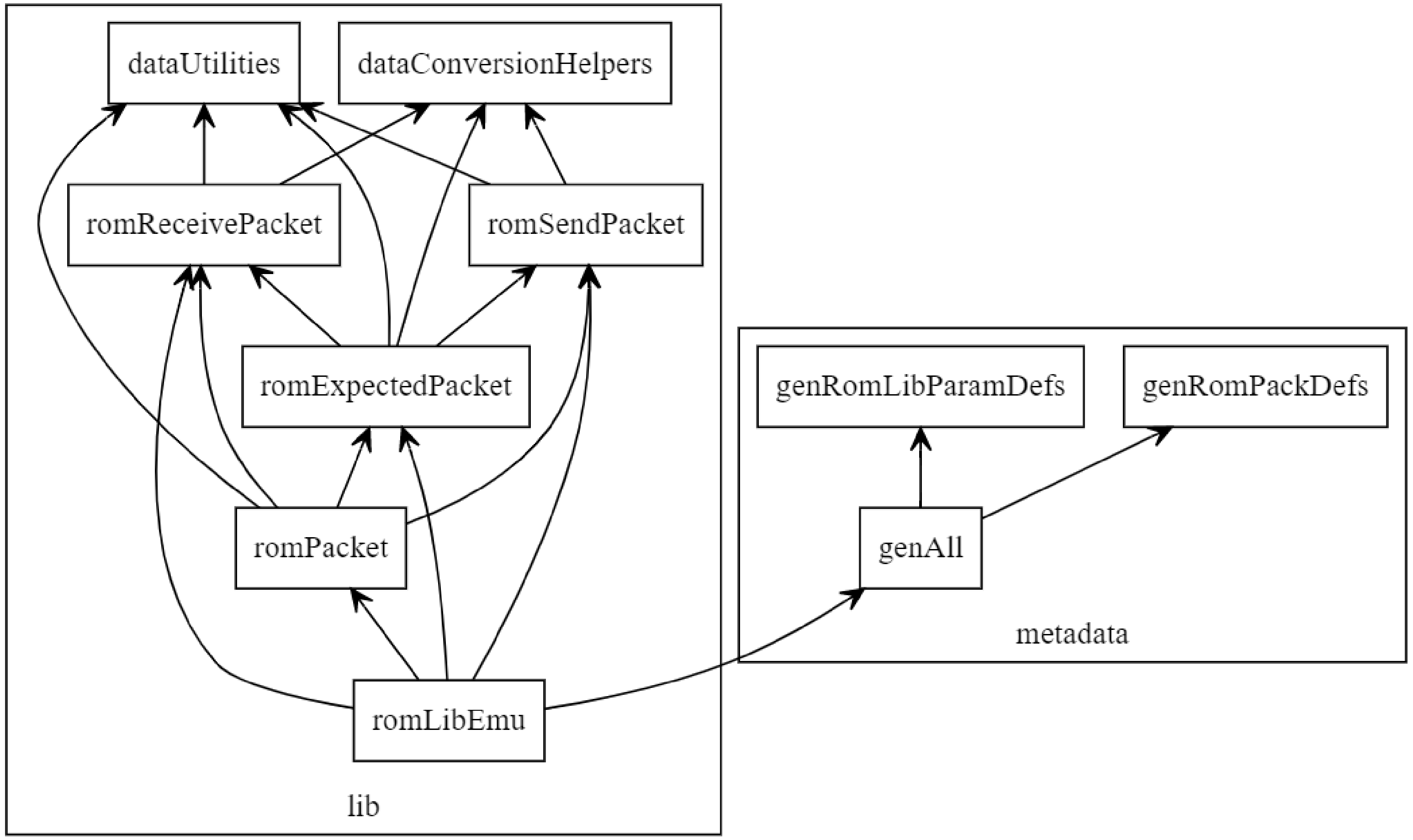




RomLibEmu –
ROMULUS Library Emulation and Test Tool

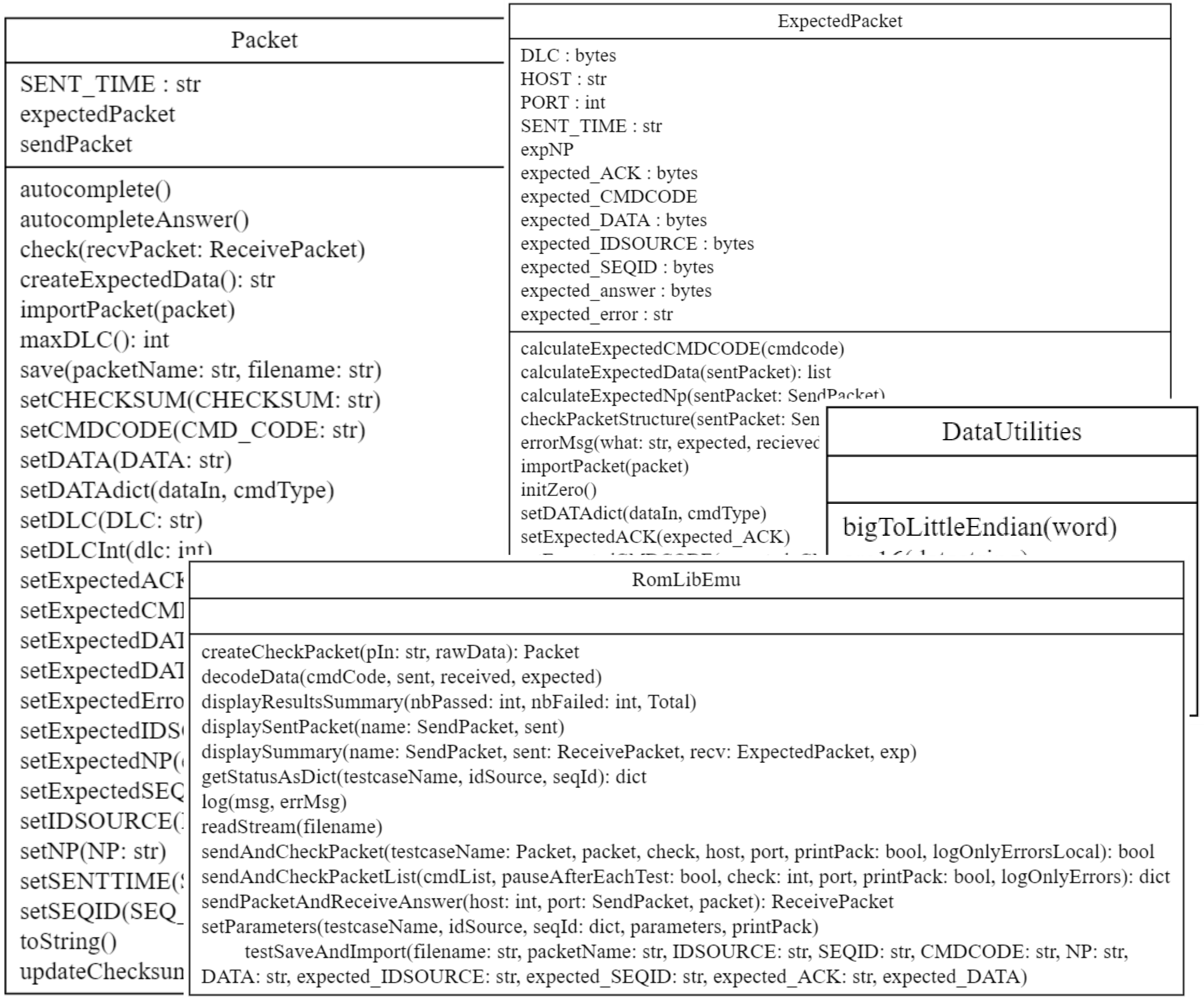
- Object-oriented – easily adaptable to future versions of protocol or application

- Protocol messages and application parameters are generated from input based on specification and stored into dictionaries



RomLibEmu

- Classes and Utility functions for constructing well-formed ROMULUS packets
- Auto-completion of
  - ROMULUS packets (checksum, data length field)
  - Expected response (response code, data length, checksum - if value is known)
- **Robustness:**
  - Possibility to construct malformed packets from raw byte string data



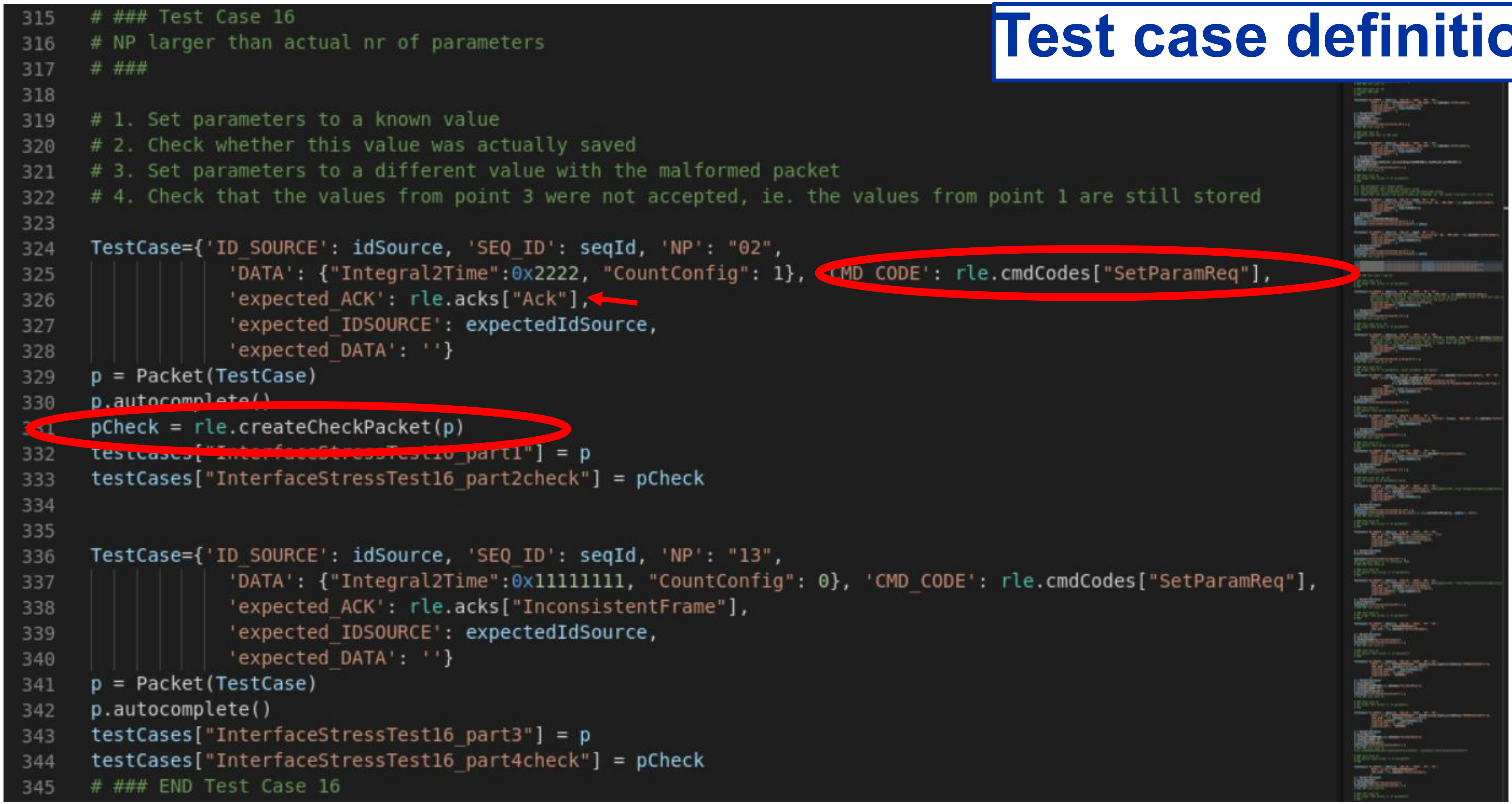
```
Packet
SENT_TIME : str
expectedPacket
sendPacket
autocomplete()
autocompleteAnswer()
check(recvPacket: ReceivePacket)
createExpectedData(): str
importPacket(packet)
maxDLC(): int
save(packetName: str, filename: str)
setCHECKSUM(CHECKSUM: str)
setCMDCODE(CMD_CODE: str)
setDATA(DATA: str)
setDATAdict(dataIn, cmdType)
setDLC(DLC: str)
setDLCInt(dlc: int)
setExpectedAC
setExpectedCM
setExpectedDAT
setExpectedDAT
setExpectedErro
setExpectedIDS
setExpectedNP(
setExpectedSEQ
setIDSOURCE(
setNP(NP: str)
setSENTTIME(
setSEQID(SEQ_
toString()
updateChecksun
```

```
ExpectedPacket
DLC : bytes
HOST : str
PORT : int
SENT_TIME : str
expNP
expected_ACK : bytes
expected_CMDCODE
expected_DATA : bytes
expected_IDSOURCE : bytes
expected_SEQID : bytes
expected_answer : bytes
expected_error : str
calculateExpectedCMDCODE(cmdcode)
calculateExpectedData(sentPacket): list
calculateExpectedNp(sentPacket: SendPacket)
checkPacketStructure(sentPacket: Sen
errorMsg(what: str, expected, recievec
importPacket(packet)
initZero()
setDATAdict(dataIn, cmdType)
setExpectedACK(expected_ACK)
```

```
DataUtilities
bigToLittleEndian(word)
```

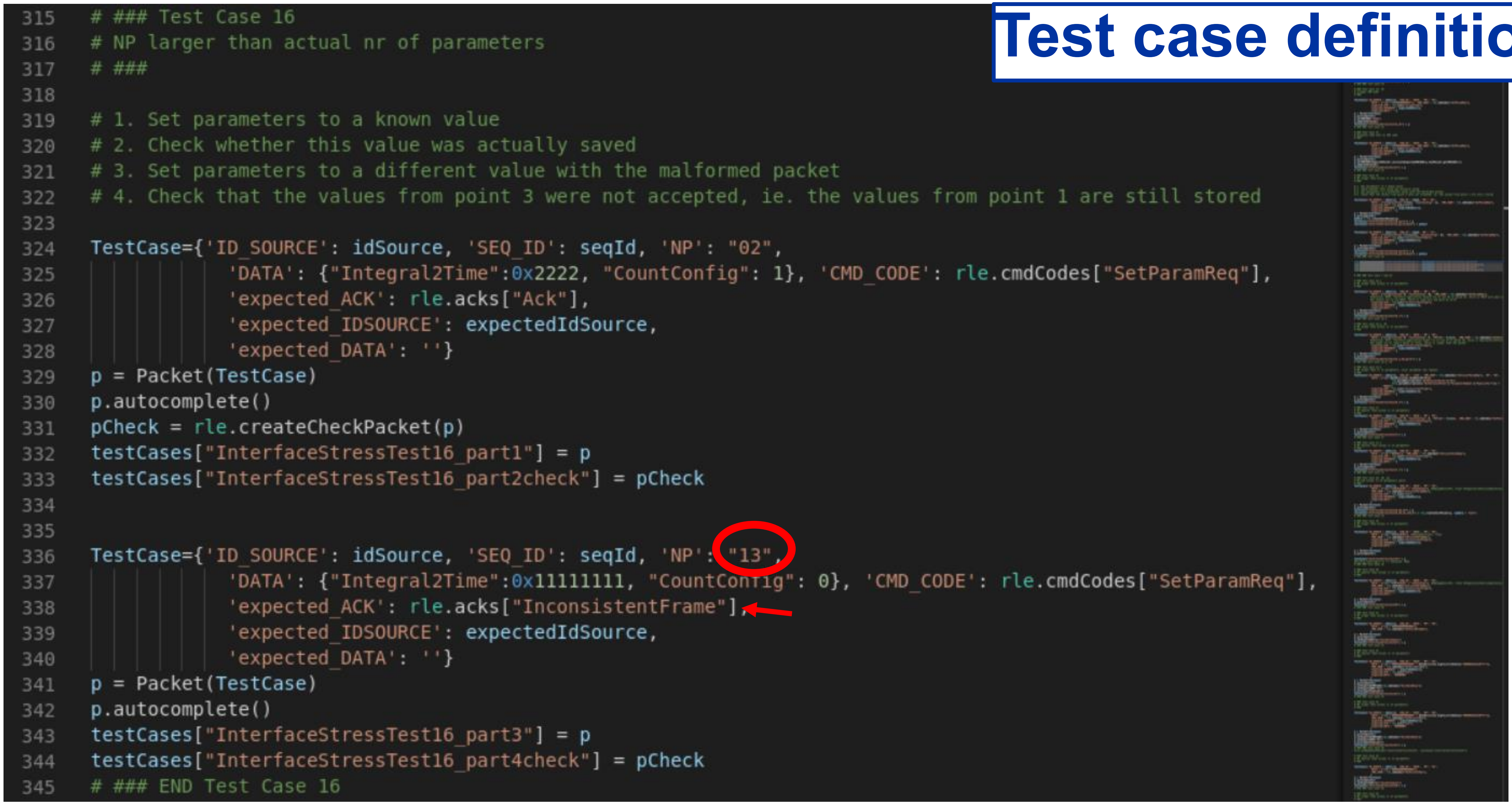
```
RomLibEmu
createCheckPacket(pIn: str, rawData): Packet
decodeData(cmdCode, sent, received, expected)
displayResultsSummary(nbPassed: int, nbFailed: int, Total)
displaySentPacket(name: SendPacket, sent)
displaySummary(name: SendPacket, sent: ReceivePacket, recv: ExpectedPacket, exp)
getStatusAsDict(testcaseName, idSource, seqId): dict
log(msg, errMsg)
readStream(filename)
sendAndCheckPacket(testcaseName: Packet, packet, check, host, port, printPack: bool, logOnlyErrorsLocal): bool
sendAndCheckPacketList(cmdList, pauseAfterEachTest: bool, check: int, port, printPack: bool, logOnlyErrors): dict
sendPacketAndReceiveAnswer(host: int, port: SendPacket, packet): ReceivePacket
setParameters(testcaseName, idSource, seqId: dict, parameters, printPack)
    testSaveAndImport(filename: str, packetName: str, IDSOURCE: str, SEQID: str, CMDCODE: str, NP: str,
DATA: str, expected_IDSOURCE: str, expected_SEQID: str, expected_ACK: str, expected_DATA)
```

Test case definition



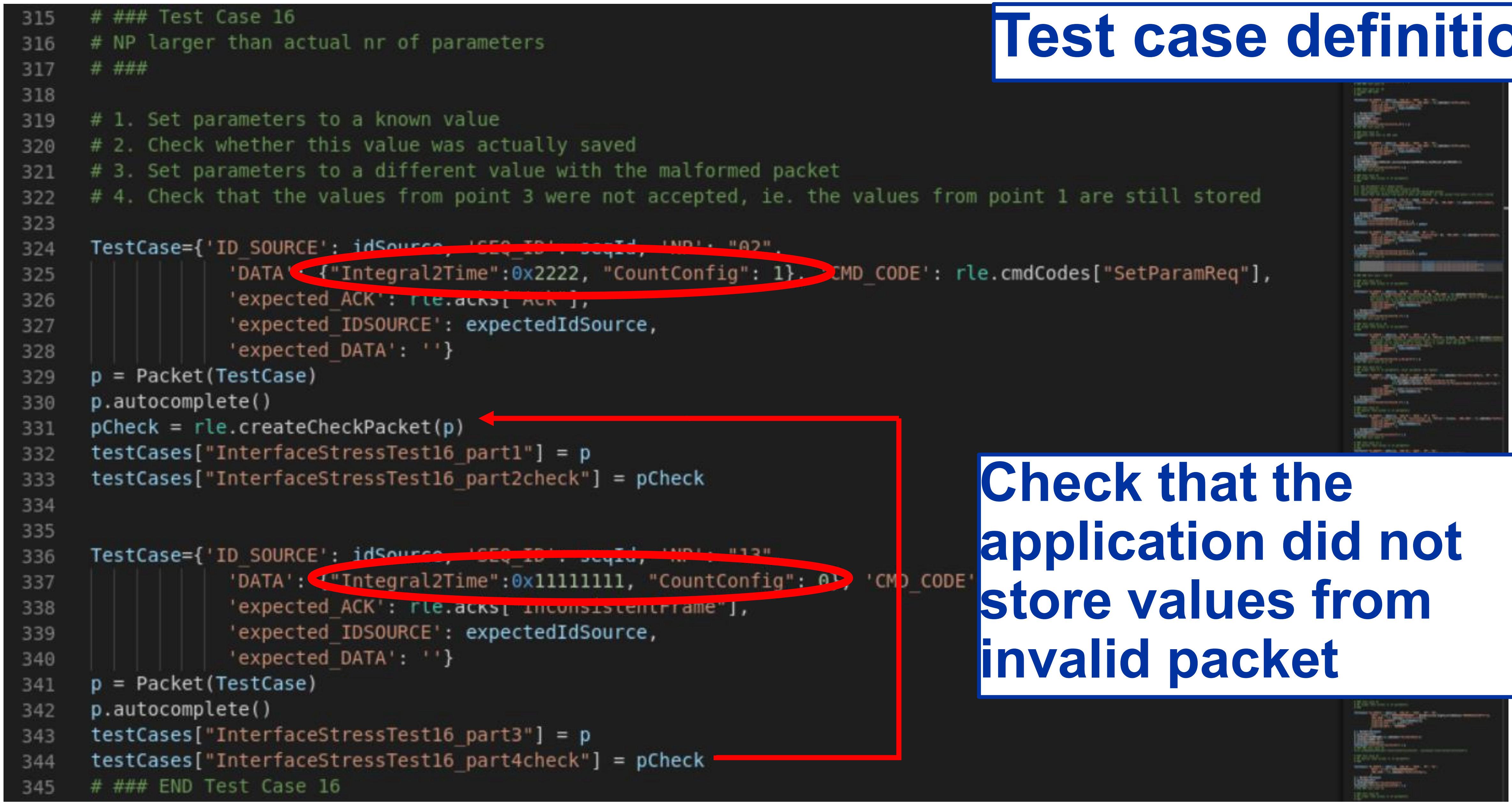
```
# ### Test Case 16
# NP larger than actual nr of parameters
# ###

# 1. Set parameters to a known value
# 2. Check whether this value was actually saved
# 3. Set parameters to a different value with the malformed packet
# 4. Check that the values from point 3 were not accepted, ie. the values from point 1 are still stored

TestCase={'ID_SOURCE': idSource, 'SEQ_ID': seqId, 'NP': "02",
          'DATA': {"Integral2Time":0x2222, "CountConfig": 1}, 'CMD_CODE': rle.cmdCodes["SetParamReq"],
          'expected_ACK': rle.acks["Ack"],
          'expected_IDSOURCE': expectedIdSource,
          'expected_DATA': ''}
p = Packet(TestCase)
p.autocomplete()
pCheck = rle.createCheckPacket(p)
testCases["InterfaceStressTest16_part1"] = p
testCases["InterfaceStressTest16_part2check"] = pCheck


TestCase={'ID_SOURCE': idSource, 'SEQ_ID': seqId, 'NP': "13",
          'DATA': {"Integral2Time":0x11111111, "CountConfig": 0}, 'CMD_CODE': rle.cmdCodes["SetParamReq"],
          'expected_ACK': rle.acks["InconsistentFrame"],
          'expected_IDSOURCE': expectedIdSource,
          'expected_DATA': ''}
p = Packet(TestCase)
p.autocomplete()
testCases["InterfaceStressTest16_part3"] = p
testCases["InterfaceStressTest16_part4check"] = pCheck
# ### END Test Case 16
```

Test case definition



```
# ### Test Case 16
# NP larger than actual nr of parameters
# ###

# 1. Set parameters to a known value
# 2. Check whether this value was actually saved
# 3. Set parameters to a different value with the malformed packet
# 4. Check that the values from point 3 were not accepted, ie. the values from point 1 are still stored

TestCase={'ID_SOURCE': idSource, 'SEQ_ID': seqId, 'NP': "02",
            'DATA': {"Integral2Time":0x2222, "CountConfig": 1}, 'CMD_CODE': rle.cmdCodes["SetParamReq"],
            'expected_ACK': rle.acks["Ack"],
            'expected_IDSOURCE': expectedIdSource,
            'expected_DATA': ''}
p = Packet(TestCase)
p.autocomplete()
pCheck = rle.createCheckPacket(p)
testCases["InterfaceStressTest16_part1"] = p
testCases["InterfaceStressTest16_part2check"] = pCheck


TestCase={'ID_SOURCE': idSource, 'SEQ_ID': seqId, 'NP': "13",
            'DATA': {"Integral2Time":0x11111111, "CountConfig": 0}, 'CMD_CODE': rle.cmdCodes["SetParamReq"],
            'expected_ACK': rle.acks["InconsistentFrame"],
            'expected_IDSOURCE': expectedIdSource,
            'expected_DATA': ''}
p = Packet(TestCase)
p.autocomplete()
testCases["InterfaceStressTest16_part3"] = p
testCases["InterfaceStressTest16_part4check"] = pCheck
# ### END Test Case 16
```

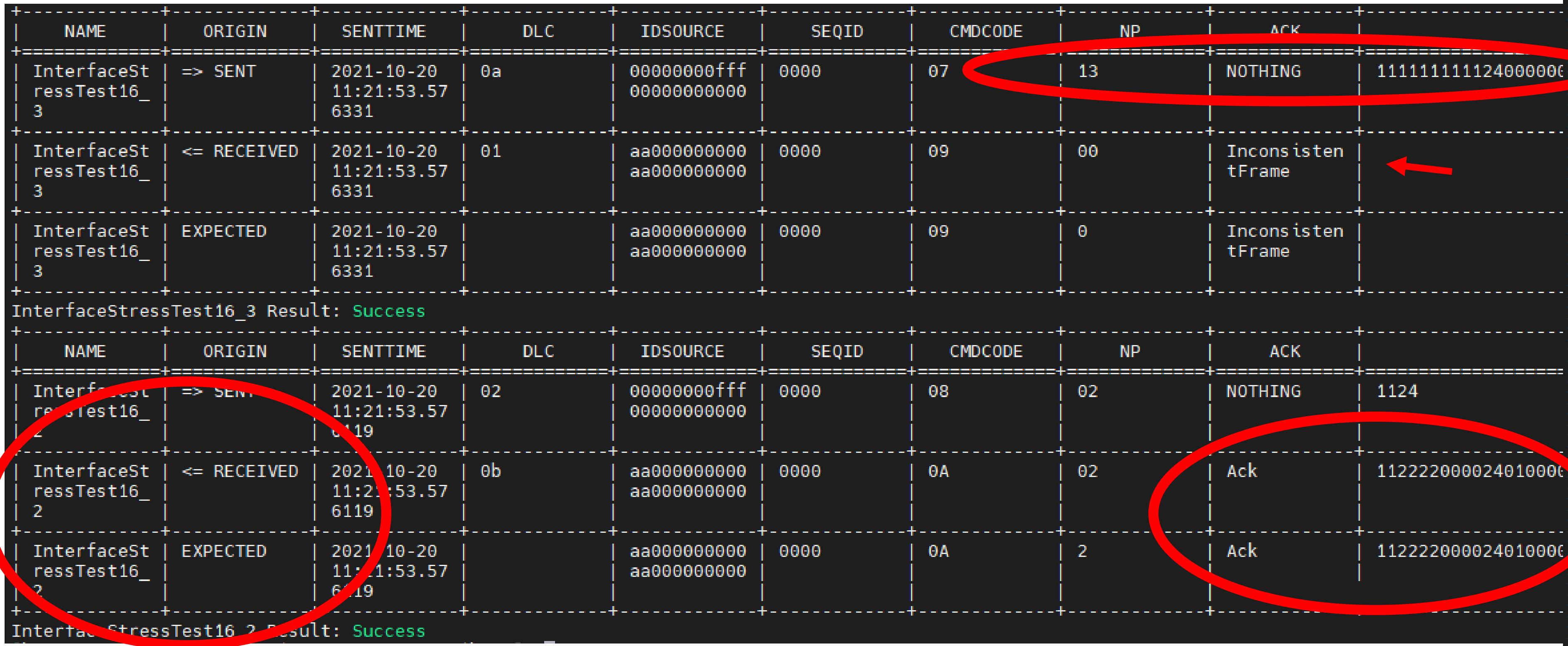
Test case definition



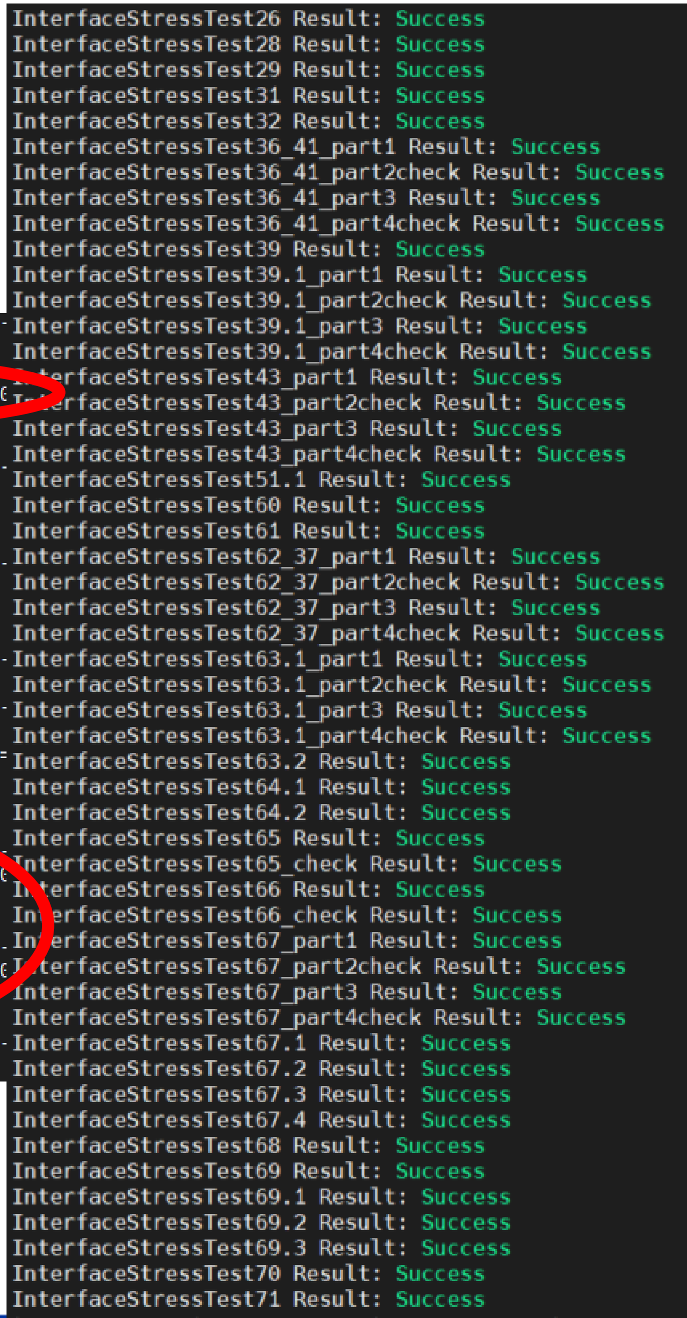
```
# ### Test Case 16
# NP larger than actual nr of parameters
# ###

# 1. Set parameters to a known value
# 2. Check whether this value was actually saved
# 3. Set parameters to a different value with the malformed packet
# 4. Check that the values from point 3 were not accepted, ie. the values from point 1 are still stored

TestCase={'ID_SOURCE': idSource, 'SEQ_ID': seqId, 'NP': "02",
            'DATA': {"Integral2Time":0x2222, "CountConfig": 1}, 'CMD_CODE': rle.cmdCodes["SetParamReq"],
            'expected_ACK': rle.acks["ACK"],
            'expected_IDSOURCE': expectedIdSource,
            'expected_DATA': ''}
p = Packet(TestCase)
p.autocomplete()
pCheck = rle.createCheckPacket(p)
testCases["InterfaceStressTest16_part1"] = p
testCases["InterfaceStressTest16_part2check"] = pCheck


TestCase={'ID_SOURCE': idSource, 'SEQ_ID': seqId, 'NP': "13",
            'DATA': {"Integral2Time":0x11111111, "CountConfig": 0}, 'CMD_CODE'
            'expected_ACK': rle.acks["InconsistentFrame"],
            'expected_IDSOURCE': expectedIdSource,
            'expected_DATA': ''}
p = Packet(TestCase)
p.autocomplete()
testCases["InterfaceStressTest16_part3"] = p
testCases["InterfaceStressTest16_part4check"] = pCheck
# ### END Test Case 16
```

Check that the application did not store values from invalid packet

Test case output



```
+-------------+-------------+-------------+-------------+-------------+-------------+-------------+-------------+-------------+------------------
|    NAME     |   ORIGIN    |  SENTTIME   |     DLC     |  IDSOURCE   |    SEQID    |   CMDCODE   |     NP      |     ACK     |
+=============+=============+=============+=============+=============+=============+=============+=============+=============+==================
| InterfaceSt | => SENT     | 2021-10-20  | 0a          | 00000000fff | 0000        | 07          | 13          | NOTHING     | 111111111124000000
| ressTest16_ |             | 11:21:53.57 |             | 00000000000 |             |             |             |             |
| 3           |             | 6331        |             |             |             |             |             |             |
+-------------+-------------+-------------+-------------+-------------+-------------+-------------+-------------+-------------+------------------
| InterfaceSt | <= RECEIVED | 2021-10-20  | 01          | aa000000000 | 0000        | 09          | 00          | Inconsisten |
| ressTest16_ |             | 11:21:53.57 |             | aa000000000 |             |             |             | tFrame      |
| 3           |             | 6331        |             |             |             |             |             |             |
+-------------+-------------+-------------+-------------+-------------+-------------+-------------+-------------+-------------+------------------
| InterfaceSt | EXPECTED    | 2021-10-20  |             | aa000000000 | 0000        | 09          | 0           | Inconsisten |
| ressTest16_ |             | 11:21:53.57 |             | aa000000000 |             |             |             | tFrame      |
| 3           |             | 6331        |             |             |             |             |             |             |
+-------------+-------------+-------------+-------------+-------------+-------------+-------------+-------------+-------------+------------------
InterfaceStressTest16_3 Result: Success
+-------------+-------------+-------------+-------------+-------------+-------------+-------------+-------------+-------------+------------------
|    NAME     |   ORIGIN    |  SENTTIME   |     DLC     |  IDSOURCE   |    SEQID    |   CMDCODE   |     NP      |     ACK     |
+=============+=============+=============+=============+=============+=============+=============+=============+=============+==================
| InterfaceSt | => SENT     | 2021-10-20  | 02          | 00000000fff | 0000        | 08          | 02          | NOTHING     | 1124
| ressTest16_ |             | 11:21:53.57 |             | 00000000000 |             |             |             |             |
| 2           |             | 6119        |             |             |             |             |             |             |
+-------------+-------------+-------------+-------------+-------------+-------------+-------------+-------------+-------------+------------------
| InterfaceSt | <= RECEIVED | 2021-10-20  | 0b          | aa000000000 | 0000        | 0A          | 02          | Ack         | 112222000024010000
| ressTest16_ |             | 11:21:53.57 |             | aa000000000 |             |             |             |             |
| 2           |             | 6119        |             |             |             |             |             |             |
+-------------+-------------+-------------+-------------+-------------+-------------+-------------+-------------+-------------+------------------
| InterfaceSt | EXPECTED    | 2021-10-20  |             | aa000000000 | 0000        | 0A          | 2           | Ack         | 112222000024010000
| ressTest16_ |             | 11:21:53.57 |             | aa000000000 |             |             |             |             |
| 2           |             | 6119        |             |             |             |             |             |             |
+-------------+-------------+-------------+-------------+-------------+-------------+-------------+-------------+-------------+------------------
InterfaceStressTest16_2 Result: Success
```



```
InterfaceStressTest26 Result: Success
InterfaceStressTest28 Result: Success
InterfaceStressTest29 Result: Success
InterfaceStressTest31 Result: Success
InterfaceStressTest32 Result: Success
InterfaceStressTest36_41_part1 Result: Success
InterfaceStressTest36_41_part2check Result: Success
InterfaceStressTest36_41_part3 Result: Success
InterfaceStressTest36_41_part4check Result: Success
InterfaceStressTest39 Result: Success
InterfaceStressTest39.1_part1 Result: Success
InterfaceStressTest39.1_part2check Result: Success
InterfaceStressTest39.1_part3 Result: Success
InterfaceStressTest39.1_part4check Result: Success
InterfaceStressTest43_part1 Result: Success
InterfaceStressTest43_part2check Result: Success
InterfaceStressTest43_part3 Result: Success
InterfaceStressTest43_part4check Result: Success
InterfaceStressTest51.1 Result: Success
InterfaceStressTest60 Result: Success
InterfaceStressTest61 Result: Success
InterfaceStressTest62_37_part1 Result: Success
InterfaceStressTest62_37_part2check Result: Success
InterfaceStressTest62_37_part3 Result: Success
InterfaceStressTest62_37_part4check Result: Success
InterfaceStressTest63.1_part1 Result: Success
InterfaceStressTest63.1_part2check Result: Success
InterfaceStressTest63.1_part3 Result: Success
InterfaceStressTest63.1_part4check Result: Success
InterfaceStressTest63.2 Result: Success
InterfaceStressTest64.1 Result: Success
InterfaceStressTest64.2 Result: Success
InterfaceStressTest65 Result: Success
InterfaceStressTest65_check Result: Success
InterfaceStressTest66 Result: Success
InterfaceStressTest66_check Result: Success
InterfaceStressTest67_part1 Result: Success
InterfaceStressTest67_part2check Result: Success
InterfaceStressTest67_part3 Result: Success
InterfaceStressTest67_part4check Result: Success
InterfaceStressTest67.1 Result: Success
InterfaceStressTest67.2 Result: Success
InterfaceStressTest67.3 Result: Success
InterfaceStressTest67.4 Result: Success
InterfaceStressTest68 Result: Success
InterfaceStressTest69 Result: Success
InterfaceStressTest69.1 Result: Success
InterfaceStressTest69.2 Result: Success
InterfaceStressTest69.3 Result: Success
InterfaceStressTest70 Result: Success
InterfaceStressTest71 Result: Success
```

RomLibEmu - Results

- **Test goal / response classification according to CRASH scale [6]:**

  (originally developed to classify robustness test response of Operating Systems (OSs))

  Adapted to CROME:

• **C**atastrophic	– The OS crashes.	Availability
• **R**estart	– The CMPU Application hangs and requires a restart.	
• **A**bort	– The CMPU application crashes.	
• **S**ilent/**S**afety	– An error response is expected but does not get returned. <u>Unknown data may be used in safety critical decisions!</u>	**Safety**
• **H**indering/**H**armless	– A wrong error code is returned by the CMPU. Functionally wrong, but safe response.	Functionality
• Functional	– Expected error code is returned, protocol specification is not met.	

[6] P. Koopman, J. Sung, C. Dingman, D. Siewiorek and T. Marz, "Comparing operating systems using robustness benchmarks," in Proceedings of SRDS'97: 16th IEEE Symposium on Reliable Distributed Systems, 1997, pp. 72-79, doi: 10.1109/RELDIS.1997.632800.

Results

Regression test suite is executed before new software/firmware versions are released.

Failed test cases categorized according to adapted CRASH scale.

Test Run	Date of SW	Silent/Safety	Abort	Restart	Hindering/ Harmless	Functional
1	04/09/19	19	2	1	3	2
2	11/12/19	10	4	0	3	2
3	20/12/19	6	0	0	3	2
4	26/05/20	6	0	0	3	3
5	02/03/21	2	0	0	3	2
6	30/09/21	0	0	0	3	0

RomLibEmu – Results Examples

- **“Abort” – application crashes caused by:**
  - TCP frame with maximum data length, data all 0s
  - ROMULUS packet length > maximum, specified data length: 0, wrong ROMULUS checksum
  - A floating point parameter being set to a quiet NaN (Not a Number)
  - Many commands sent in parallel to the command port
- **“Silent/Safety”:**
  - Invalid floating point values (NaN, infinites) – quietly accepted by the application and passed to the safety critical calculations inside PL
  - Packets with invalid structure were processed:
    - Application read beyond field lengths. E.g. checksum was interpreted as parameter.

Results

Test Run	Date of SW	Nr. Test Cases	Passed	Failed
1	04/09/19	57	30	27
2	11/12/19	58	39	19
3	20/12/19	64	53	11
4	26/05/20	65	53	12
5	02/03/21	81	74	7
6	30/09/21	83	80	3

83 main test cases – 251 sub test cases

5 availability, 19 safety, 2 functional **bugs found and fixed**

Several **regressions detected before** software/firmware was **released into long-term test.**

Conclusion

- **Benefits of robustness tests:**
  - **Find faults that functional tests can’t find**
- **Benefit of automated test suite:**
- **Discover regressions early**

- **Outlook**
  - Functional tests with random inputs
  - HW/SW Co-Verification
  - Security
  - Automated regression runs with CI/CD

Thank you!

home.cern

Contact:
katharina.ceesay-seitz@cern.ch

References

[1] Based on E. Mobs, The CERN accelerator complex – 2019, cds.cern.ch/record/2684277

[2] A. Yadav, H. Boukabache, K. Ceesay-Seitz, D. Perrin, “ROMULUSlib: An autonomous, TCP/IP-based, multi-architecture C networking library for DAQ and Control applications”, in *Proc. 18th Int. Conf. on Accelerator and Large Experimental Physics Control Systems (ICALEPCS 2021)*, Shanghai, China, Oct. 2021, paper MOBR01, this conference.

[3] A. Ledeul, G. Segura Millan, A. Savulescu, B. Styczen, and D. Vasques Ribeira, “CERN Supervision, Control and Data Acquisition System for Radiation and Environmental Protection”, *in Proc. 12th International Workshop on Personal Computers and Particle Accelerator Controls (PCaPAC’18)*, Hsinchu City, Taiwan, Rep. of China, 2018, pp. 248-252. doi:0.18429/JACoW-PCaPAC2018-FRCC3

[4] S. Shah, D. Sundmark, B. Lindström, S. Andler, “Robustness Testing of Embedded Software Systems: An Industrial Interview Study”, in IEEE Access 4: 1-1, pp. 1859-1871, 2016, 10.1109/ACCESS.2016.2544951.

[5] N. Laranjeiro, J. Agnelo, J. Bernardino, “A Systematic Review on Software Robustness Assessment”, in ACM Computing Surveys, vol. 54, issue 4, nr. 89, pp. 1-65, 2021, doi:10.1145/3448977

[6] P. Koopman, J. Sung, C. Dingman, D. Siewiorek and T. Marz, "Comparing operating systems using robustness benchmarks," *Proceedings of SRDS'97: 16th IEEE Symposium on Reliable Distributed Systems*, 1997, pp. 72-79, doi: 10.1109/RELDIS.1997.632800.

RomLibEmu - Results

- **Test categories:**

  1. Good cases – Packets that conform to the ROMULUS-REMUS communication protocol.
  2. Application tests – Packets with correct ROMULUS packet structure, but bad parameter configuration.
  3. Bad network connection – Simulating the loss of packets, long delay, etc.
  4. ROMULUS library tests – Malformed ROMULUS packets that were generated due to bugs in the ROMULUS library; Network packets that were not targeted to the CMPU but arrive because of misconfiguration and are therefore treated by the application like legitimate traffic.
  5. Denial-of-Service – Overload of the network interface by sending too many messages in a short time or in parallel, unexpectedly long packets.
  6. Intentional attacks
  7. Regression tests – Test cases that have already revealed a bug in some version of the application or the ROMULUS library