Passive Approach for Robustness Testing of Communication Protocols

Eliane Martins, Anderson Morais
IC- Unicamp

Ana Cavalli
Institut Telecom & Management Sud-Paris

IC-Unicamp – Set/2008
Topics

- Robustness Testing
  - Why
  - What
- Proposed Approaches
- An Hybrid Approach
- Results
Why robustness testing?

Testing software to ensure that the functional requirements were met …

… is necessary but not enough

- How does the system behave in presence of
  - erroneous or unexpected user inputs?
  - internal or external failures?
  - stressful environmental conditions?
Robustness

**Definition**

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

IEEE Std 610.12-1990 - Glossary of Software Engineering Terminology
Robustness testing

Definition [CW03] :

“aimed to determine whether a system or component can have an acceptable behavior in the presence of faults or stressful environmental conditions”
Robustness testing approaches

- Ad-hoc approaches
  - Hard to automate
- Based on models
- Based on fault injection
Model-based approaches

- Formalization of robustness testing is inspired on that of conformance testing
  - Conformance testing:
    - Goal: determining whether an implementation conforms to its specification
  - The specification is represented by a (behavior) model from which:
    - Test cases can be derived
    - Observed results can be analyzed
Robustness test cases generation

Specification (nominal) -> Specification (modified) -> Test case generation -> Robustness test cases

Faults + Extra outputs
Illustrative example

Nominal model

Diagram showing nodes 1, 2, 3, and 4 with connections labeled 'a', 'b', and 'x'.
Illustrative example

Nominal model

Augmented model

Degraded state

Unhandled invalid input

Handled invalid input

Exceptional end state
Illustrative example

Nominal model

Augmented model

Nominal model

Augmented model

inopportune inputs
Difficulties with the model-based approaches

- Model size is too big for use
  - Need to carefully define test objectives
- Tester has limited control of faults
  - Faults to consider may depend on the application domain and on the system architecture
  - Environmental (context) faults (memory, processor, communication channel, device drivers) are not considered
- System behavior in the presence of faults cannot always be completely specified
Fault injection

- Definition

*Deliberate introduction of faults into a system to observe its behavior*

- Applicability
  - To verify whether the error detection and recovery mechanisms behave as expected.
  - To evaluate dependability measures such as reliability for a given mission time, availability, performance degradation due to fault handling.
  - To understand the effects of real faults.
Fault injection approaches

- Faults can be injected:
  - Into a model: Simulation-based fault injection
  - Into a prototype or final system:
    - Hardware level: Hw-implemented fault injection (HWIFI)
    - Software level: Sw-implemented fault injection (SWIFI)
Robustness testing and fault injection

- Interface fault injection:
  - affects functions input/output parameters or protocol messages fields
  - Invalid values produced according to input/output domains or formats

- Some approaches and tools:
  - Ballista/Piranha, Mafalda, Fuzz, Riddle, PROTOS, Jaca
Limitations of interface fault injection approaches

- Oracle is generally not based on the specification
  - “golden run” or reference implementation
  - Crash or not crash
- Knowledge about the system structure or behavior is not frequent
Proposed approach

- Hybrid approach combining
  - Fault injection
  - Passive testing
Passive testing approaches

- Based on trace acceptation
  - determines whether the observed trace satisfies the specification model

- Based on invariants
Abstract test architecture

PO: Point of Observation
SAP: Service Access Point

Fault Injector
Fault set

Implementation under test (IUT)

Test context

Execution trace

Pass
Fail
Inconclusive

Tester

Specification

Robustness properties

Invariant
Invariants analysis approach

Behavior model

Invariants = properties of interest

Invariants in the form of regular expressions

\[ I_1 = RcvInvoke(TID = N) /?, *, TR-Invoke.res / \{ Ack (TID = N) \} \]

\[ I_2 = RcvInvoke(TID = N) / TR.Invoke.ind, *, TR-Invoke.res / \{ Ack (TID = N) \} \]
Test configuration
The WAP stack

User defined inputs

WAP Terminal
- WAE
- WSP
- WTP - Initiator
- WDP/UDP
- IP

WAP Gateway
- WAE
- WSP
- WTP - Responder
- WDP/UDP
- Fault Injector
- IP

HTTP server

Faultlets
Client Terminal (simulator)        Gateway (Kannel)

WSP                                    WTP (Initiator)          WTP (Responder)          WSP

Tr-Invoke.req → Invoke → Tr-Invoke.ind
Tr-Result.ind → Result → Tr-Result.req
Tr-Result.res → Ack → Tr-Result.cnf

... No faults injected
Fault injected
An experiment that failed

<table>
<thead>
<tr>
<th>Experiments</th>
<th>Runs</th>
<th>Result seen by Nokia browser</th>
<th>Observ.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>E1- Test packet corruption.</strong></td>
<td><strong>R1- Ack (0x3) →Invoke (0x1)</strong></td>
<td>Requested page</td>
<td></td>
</tr>
<tr>
<td><strong>Change PDU Type</strong></td>
<td>R2- Ack (0x3) →Invalid (0x00)</td>
<td>Requested page</td>
<td></td>
</tr>
<tr>
<td></td>
<td>R3- Ack (0x3) →Result (0x2)</td>
<td>Error message: “Server aborted connection”</td>
<td></td>
</tr>
<tr>
<td></td>
<td>R4- Ack (0x3) →Invalid (0xff)</td>
<td></td>
<td>Browser blocked</td>
</tr>
</tbody>
</table>
Example of observed trace with failure (1)


2007-10-11 01:21:50 [0] ERROR: pdu unpacking returned NULL

2007-10-11 [0] ERROR: SIGINT received, let's die.

Abort PDU
Wapbox hangs
Run aborted by the user


...
Another experiment that failed

- Experiment 5: wrong packet size.
  - Run 2: change PDU size to small value (=2)
    - Failure: no Abort message generated as was expected!


Crash of the wapbox

... 2007-10-11 03:53:21 [0] DEBUG: A too short PDU received
2007-10-11 03:53:21 [0] DEBUG: Dumping WAPEvent 0x820bad0
2007-10-11 03:53:21 [0] DEBUG: type = T-DUnitdata.ind
2007-10-11 03:53:21 [0] DEBUG: WAPAddrTuple 0x820bb40 = <127.0.1.1:32787> - <0.0.0.0:9201>
2007-10-11 03:53:21 [0] DEBUG: user_data = 2007-10-11 03:53:21 [0] DEBUG: Octet string at 0x820bd38:
2007-10-11 03:53:21 [0] DEBUG: len: 1
2007-10-11 03:53:21 [0] DEBUG: size: 2
2007-10-11 03:53:21 [0] DEBUG: immutable: 0
2007-10-11 03:53:21 [0] DEBUG: data: 18

Simple invariants used

S1. RcvInvoke/TR-Invoke.ind,*\{TR-Result.req\{Result\}\
S2. RcvInvoke/TR-Invoke.ind,*\{RcvAck\{TR-Result.cnf, NULL\}\
S3. RcvErrorPDU\{Abort, TR-Abort.ind\}\
S4. ?, *, RcvAbort\{TR-Abort.ind\}\
S5. ?, *, TimerTO_R\{Result,TR-Abort.ind\}\
S6. ?, *, TimerTO_A\{Ack,TR-Abort.ind, NULL\}\
S7. ?, *, TR-Abort.req/\{Abort\}\
S8. RcvInvoke/Ack, *, RcvAck/\{TR.Invoke.ind\}\
S9. RcvInvoke/Ack, *, RcvInvoke/\{Ack, NULL\}\
S10. ?, *, NULL/\{CRASH, HANG\}
Discussion about observed results

- Only control flow was considered in the invariant analysis

- Observed anomalous behavior:
  - Lack of resources created new sources of failures:
    - IUT did not tolerate some OS exceptions
  - Lack of information in the specification
    - Ex.: Initiator continues to send requests for new transactions even when the Responder keeps retransmitting the same results
Conclusions

- Hybrid approach for robustness testing, combining formal and fault injection techniques:
  - Fault injection:
    - Allows better coverage of environment faults than in traditional testing
  - Passive testing:
    - Allows more precise result analysis than simply observing crash or hangs, as is usual in FI
  - Possibility to test an IUT in its context → useful in later stages of system testing or even in the field
Current work

- Approach is in use for testing robustness against attacks:
  - Cryptographic protocol testing
  - Instead of communication faults, attacks are injected
  - Attack scenarios derived from real successful attacks reported in the literature
  - Attacker is implemented by a fault injector
  - Goal: reveal vulnerabilities in the protocol implementation
  - Invariants used to represent security properties
Future works

- Algorithm for the transformation of attack scenarios into executable scenarios for the fault injector (Attacker)
- Application of the approach to a case study
- Use of sequence alignment algorithms for results analysis
Thanks!

Email: eliane@ic.unicamp.br
    anderson.morais@ic.unicamp.br
References (1)

**Report about model-based robustness testing:**

**Ballista**

**Piranha**

**Mafalda**

**Fuzz**

**PROTOS**
References (2)

**RIDDLE**


**Jaca**


**About invariant testing:**


**Introduction to fault injection:**


**Hybrid approaches for active testing:**

