Analyzer test plan

DaCoPAn

Helsinki 11th April 2005

Software Engineering Project

UNIVERSITY OF HELSINKI
Department of Computer Science

UNIVERSITY OF PETROZAVODSK
Department of Computer Science

Course

581260 Software Engineering Project (6 cr)

Project Group

Carlos Arrastia Aparicio
Jari Aarniala
Alejandro Fernandez Rey
Vesa Vainio
Jarkko Laine
Jonathan Brown

Kirill Kulakov
Andrey Salo
Andrey Ananin
Mikhail Kryshen
Viktor Surikov

Customer

Markku Kojo

Project Masters

Juha Taina (Supervisor)
Yury Bogoyavlenskiy (Supervisor)

Turjo Tuohiniemi (Instructor)
Dmitry Korzun (Instructor)

Homepage

`http://www.cs.helsinki.fi/group/dacopan`

Change Log

Version	Date	Modifications
1.0	Put the date here	First version

Contents

1 Introduction

This document is a Test Plan for the Analyzer system of the DaCoPAn project. It describes the testing approach and test cases to be used for verification and validation of the Analyzer system. The testing is mainly focused on features and functions, which are listed in the Requirements specification document.

Used testing approach is presented in section 2. Test cases for the unit testing are described in section 3. The section 4 presents test cases for the integration testing. The validation testing is described in section 5.

2 Approach

Three phases for testing are used: unit, integration and validation testing.

The unit testing based on while-box methods. Unit tests exersice that each individual unit works as it is supposed. These tests are executed by the developers in parallel with the implementation. Unit tests includes the following cases:

- basic tests. It is a simple tests and must be accepted.
- common tests. It is a admissible domain for unit work. The tester may select some cases in this domain for test execution. All tests must be accepted.
- boundary tests. It is a quantity of tests for unit work. If unit's domain does not have known bounds then tester gets maximum meaning. All tests must be accepted.
- wrong tests. This tests are used for check correctly unit work on wrong input data. The unit work result should be concur with due result.
- special tests. This tests checks some unit-specific work cases. These cases are developed on design phase or implementation phase. Also these cases may be included into the user requirements. All tests must be accepted or received result should be concur with due result.

The integration testing based on interface testing approach. Integration tests exersice systematically all key combinations of subsystems to uncover errors and defects associated with subsystem interfaces. Bottom-up incremental integration is used. The integration sequence is following:

- integrating objects to module;
- integrating two neighboring modules;
- integrating modules to application.

The cases for Integration testing are following:

- basic tests. It is a simple tests for checking interface between units. These tests must be accepted.
- common tests. It is a admissible domain for interface work. The tester may select some cases in this domain for test execution. All tests must be accepted.
- boundary tests. It is a quantity of tests for interface work. If interface's domain does not have known bounds then tester gets maximum meaning. All tests must be accepted.
- wrong tests. This tests are used for check correctly interface work on wrong input data. The interface work result should be concur with due result.
- special tests. This tests checks some interface-specific work cases. These cases are developed on design phase or implementation phase. Also these cases may be included into the user requirements. All tests must be accepted or received result should be concur with due result.

The validation testing use black-box approach. The validation testing is based on User requirements, use cases and behavioral models [3, 1]. Validation tests exersice on the behavior and the advanced functionality of the whole Analyzer system.

3 Unit testing

3.1 Command line parser

Author: Andrew Salo

Description: This module processes command line arguments of the Analyzer.

3.1.1 parse_args()

Description: This function parses command line arguments and sets global variables to user specified values. The arguments of the function should be exactly `main` function arguments.

Parameters:

1. `int argc`: the number of command line arguments.
   Restrictions: should be a valid number of command line arguments.
2. `char **argv`: a vector of strings; its elements are individual command line argument strings.
   Restrictions: should contain a valid number of non-NULL strings.

Return value: `void`.

List of tests:

Test case 1: (general) Process valid command line arguments.

Input data:
argc, argv - `main` function arguments containing valid command line data.

Expected outcome:
The global variables are set to user specified values.

Notes:
The command line should have not only valid syntax, but also valid data (valid number of mandatory arguments, valid IP address format, etc).

Test case 2: (positive) Show program information (version or documentation).

Input data:
argc, argv - `main` function arguments containing `--help` and/or `--version` options.

Expected outcome:
Show version information or brief documentation depending on specified option and exit successfully.

Notes:
When the function finds a `--version` or `--help` option it prints corresponding information and immediately terminates the program. The rest of command line is not processed.

Test case 3: (negative) Process command line which has invalid syntax.

Input data:
argc, argv - `main` function arguments which have invalid syntax.

Expected outcome:
Descriptive error string is displayed and the program is terminated with non-zero status.

Notes:
The error message itself is printed by `getopt` function. The short tip (something like “Try ‘--help’ for more information.”) is also displayed.

Test case 4: (negative) Process command line with correct syntax but invalid data (invalid number of mandatory arguments, invalid IP address format, etc).

Input data:
argc, argv - `main` function arguments containing invalid data.

Expected outcome:
Descriptive error string is displayed and the program is terminated with non-zero status.

Notes:
The error message is printed by error processing module.

3.2 Log reader

Author: Andrew Salo

Description: This module reads binary tcpdump log files and produces packet trace presentations.

3.2.1 read_log()

Description: This function reads packet trace file, which is binary file captured by tcpdump tool, and produces packet trace presentation.

Parameters:

1. `struct host *h`: the structure containing packet trace file name.
   Restrictions: non-NULL pointer.
2. `struct ptu **begin`: pointer to the beginning of packet trace presentation.
   Restrictions: none.

Return value: `void`.

List of tests:

Test case 1: (general) Read valid binary tcpdump log file.

Input data:
h - valid pointer to structure host.

Expected outcome:
Packet trace presentation is produced.

Test case 2: (negative) File reading error.

Input data:
h - valid pointer to structure host.

Expected outcome:
Descriptive error message is displayed and the program is terminated with non-zero exit status.

3.3 Events Calculator

Author: Ananin Andrey

Description: This module splits PTU sequence and calculate necessary protocol variables.

3.3.1 split()

Description: This function splits each element of the PTU sequence on events and create Events sequence.

Parameters:

1. `PTU *PTUSeqeunce` the pointer to the first element of PTU Sequence
   Restrictions: non-NULL pointer.
2. `event **sequence`: the pointer to the first element of the linked list (empty default) of the analyzed protocol events.
   Restrictions: First value is NULL pointer.
3. `host *hosts`: the pointer to the first element of the linked list (empty default) of the used hosts.
   Restrictions: non-NULL pointer.
4. `link **links`: the pointer to the first element of the linked list (empty default) of the used links.
   Restrictions: First value is NULL pointer.
5. `flow **flows`: the pointer to the first element of the linked list (empty default) of the used flows.
   Restrictions: First value is NULL pointer.

Return value: `void`.

List of tests:

Test case 1: (general|positive) Create right Events sequence.

Input data:
PTUSequence - pointer to the first element of the PTU sequence,
events - pointer to the first element of the Events sequence (NULL at first),
hosts - pointer to the first element of the host list,
links - pointer to the first element of the link list (NULL at first),
flows - pointer to the first element of the flow list (NULL at first).

Expected outcome:
Events sequence is successfully created. The data represented in the Events sequence corresponds to the protocol events in right order. Links and flows sequence are suddessfully created.

Notes:
This test should be repeated for each scenario (providing corresponding PTU sequence as an input).

Test case 2: (negative) Create wrong Events sequence.

Input data:
PTUSequence - pointer to the first element of the PTU sequence,
events - pointer to the first element of the Events sequence (NULL at first). *hosts - pointer to the first element of the host list,
links - pointer to the first element of the link list (NULL at first),
flows - pointer to the first element of the flow list (NULL at first).

Expected outcome:
Events sequence is successfully created. But the data are not represented in the Events sequence (corresponds to the protocol events) in right order. Links and flows sequence are suddessfully created.

Notes:
none.

3.3.2 calculate()

Description: This function builds list of flows, list of links, calculates necessary protocol variables and adds dropped events in the events sequence.

Parameters:

1. `PTU *PTUSeqeunce` the pointer to the first element of PTU Sequence
   Restrictions: non-NULL pointer.
2. `event **events_sequence`: Events sequence which should be calculated.
   Restrictions: First value is NULL pointer.
3. `link **links`: List of links which should be calculated.
   Restrictions: First value is NULL pointer.
4. `link **flows`: List of flows which should be calculated.
   Restrictions: First value is NULL pointer.
5. `char *add_log_var`: Used additional log file.
   Restrictions: non-NULL pointer.

Return value: `void`.

List of tests:

Test case 1: (general|positive) Create right Events sequence with right dropped events, lists of flows and links and calculated variables.

Input data:
PTUSequence - pointer to the first element of the PTU sequence,
*events_sequence - pointer to the first element of the Events sequence (NULL at first),
*links - pointer to the first element of the link sequence (NULL at first),
*flows - pointer to the first element of the flow sequence (NULL at first),
add_log_var - pointer to the additional log file name.

Expected outcome:
Events sequence is successfully created. The data represented in the Events sequence corresponds to the protocol events in right order. Each event should have address of the flow and link in the correspondings lists. Also dropped events should be added in the events sequence in the right order.

Notes:
This test should be repeated for each scenario (providing corresponding PTU sequence as an input).

Test case 2: (specific) Can't add dropped event in the events sequence.

Input data:
PTUSequence - pointer to the first element of the PTU sequence,
events_sequence - pointer to the first element of the Events sequence (NULL at first),
emphiric timestamp of dropped event is the same as timestamp of corresponding event.

Expected outcome:
Dropped event was not added in the events sequence. Exit with error.

Notes:
none.

3.4 PEF Writer

Author: Mikhail Kryshen

Description: This module writes the analyzed communication data into the *protocol events file*.

3.4.1 pef_write()

Description: This function writes the analyzed protocol communication data to the specified output stream using the Protocol Events Format.

Parameters:

1. `FILE *out`: specifies where to output the data
   Restrictions: non-NULL pointer.
2. `event *sequence`: the pointer to the first element of the linked list of the analyzed protocol events.
   Restrictions: non-NULL pointer.

3. `host *hosts`: the pointer to the first element of the linked list of the hosts.
   Restrictions: non-NULL pointer.
4. `link *links`: the pointer to the first element of the linked list of the links.
   Restrictions: non-NULL pointer.
5. `flow *flows`: the pointer to the first element of the linked list of the flows.
   Restrictions: non-NULL pointer.

Return value: `void`.

List of tests:

Test case 1: (general) Write a PEF file.

Input data:
out - pointer to the file stream opened for writing,
events - pointer to the first element of the *events sequence*,
hosts - pointer to the first element of the host sequence,
links - pointer to the first element of the link sequence,
flows - pointer to the first element of the flow sequence.

Expected outcome:
PEF file is successfully written. The data represented in the file corresponds to the data in the source events sequence.

Notes:
This test should be repeated for each scenario (providing corresponding events sequence as an input).

Test case 2: (negative) I/O error.

Input data:
out - pointer to the file stream, which is not writable for some reason,
events - pointer to the first element of the *events sequence*,
hosts - pointer to the first element of the host sequence,
links - pointer to the first element of the link sequence,
flows - pointer to the first element of the flow sequence.

Expected outcome:
Error processing function is called. Program terminates with non-zero exit code.

3.4.2 pef_writef()

Description: This function writes the analyzed protocol communication data to the specified file using the Protocol Events Format.

Parameters:

1. `char *filename`: specifies where to output the data
   Restrictions: non-NULL pointer.
2. `event *sequence`: the pointer to the first element of the linked list of the analyzed protocol events.
   Restrictions: non-NULL pointer.
3. `host *hosts`: the pointer to the first element of the linked list of the hosts.
   Restrictions: non-NULL pointer.
4. `link *links`: the pointer to the first element of the linked list of the links.
   Restrictions: non-NULL pointer.
5. `flow *flows`: the pointer to the first element of the linked list of the flows.
   Restrictions: non-NULL pointer.

Return value: `void`.

List of tests:

Test case 1: (general) Write a PEF file.

Input data:
filename - pointer to the file name (file may be opened for writing),
events - pointer to the first element of the *events sequence*,
hosts - pointer to the first element of the host sequence,
links - pointer to the first element of the link sequence,
flows - pointer to the first element of the flow sequence.

Expected outcome:
pef_write() function is called, PEF file is successfully written.

Test case 2: (negative) Failed to open the file.

Input data:
filename - pointer to the file name (file could not be opened for writing for some reason: permission denied, bad filename etc.),
events - pointer to the first element of the *events sequence*,
hosts - pointer to the first element of the host sequence,
links - pointer to the first element of the link sequence,
flows - pointer to the first element of the flow sequence.

Expected outcome:
Error processing function is called. Program terminates with non-zero exit code.

3.5 Error processing module

Author: Andrew Salo

Description: This module processes different types of errors occurred while running the Analyzer.

3.5.1 anlz_strerror()

Description: This function maps the error code specified by its argument to a descriptive error message string.

Parameters:

1. `int err`: specifies the error code. `ERR_*` constant should be used instead of explicit integer value.
   Restrictions: none.

Return value: `const char *`.

List of tests:

Test case 1: (general) Get descriptive error string using `ERR_*` constant.

Input data:
err - one of `ERR_*` constants.

Expected outcome:
The descriptive error message string corresponding to specified constant is returned.

Test case 2: (specific) Get descriptive error message using explicit integer value.

Input data:
err - integer value.

Expected outcome:
The descriptive error message string, if err >= 0 and err < err_num, or “unknown error” string, otherwise.

Notes:
This feature should not be used by the Analyzer modules while calling the function on errors. Only symbolic error names `ERR_*` should be used.

3.5.2 error()

Description: This function displays the error message according to error message format and terminates the program with the exit status specifies.

Parameters:

1. `int status`: the exit status of the program.
   Restrictions: none.
2. `const char *cause`: the cause in the error message string.
   Restrictions: none.
3. `const char *descr`: the description of the error.
   Restrictions: none.

Return value: `void`.

List of tests:

Test case 1: (general) Display error message and exit with non-zero status.

Input data:
status - non-zero integer,
cause - string,
descr - string.

Expected outcome:
The function should display error message and exit with non-zero status. If descr is a NULL string, the "unknown error" message is displayed.

Notes:
The error message format may vary depending on descr value (NULL or non-NULL).

Test case 2: (general) Display error message and return.

Input data:
status - zero integer,
cause - string,
descr - string.

Expected outcome:
The function should display error message and return. The program should not be terminated. If descr is a NULL string, the "unknown error" message is displayed.

Notes:
The error message format may vary depending on descr value (NULL or non-NULL).

4 Integration testing

4.1 Command line parser

Test 1_1 (basic)
Description. The simplest way to invoke the Analyzer

Input data. two IP addresses, two file names
Expected outcome. Command line parser sets global variables

Test 1_2 (common)
Description. Analyzer invocation with more options
Input data. two IP addresses, two file names, time alignment, output file name
Expected outcome. Command line parser sets global variables

Test 1_3 (common)
Description. Analyzer invocation with two time alignments
Input data. two IP addresses, two file names, two time alignments
Expected outcome. Command line parser sets global variables

Test 1_4 (common)
Description. Analyzer invocation with dns and http ports specified
Input data. two IP addresses, two file names, http and dns port numbers
Expected outcome. Command line parser sets global variables

Test 1_5 (special)
Description. Analyzer invocation with duplicate ports specified
Input data. two IP addresses, two file names, http and dns port numbers with duplicated ports
Expected outcome. Command line parser sets global variables

Test 2_1 (basic)
Description. Show program version information
Input data. version option
Expected outcome. Version information

Test 2_2 (basic)
Description. Show program brief documentation
Input data. help option
Expected outcome. brief documentation

Test 2_3 (special)
Description. Show program version ignoring other options
Input data. version option and other options
Expected outcome. Version information

Test 3_1 (negative)
Description. Invoke program with non-existent option
Input data. non-existent option
Expected outcome. Error message

Test 3_2 (negative)

Description. Invoke program with non-existent short option
Input data. non-existent short option
Expected outcome. Error message

Test 3_3 (negative)
Description. Invoke program with option without required argument
Input data. Analyzer options without required IP address(es) and/or file name(s)
Expected outcome. Error message

Test 4_1 (negative)
Description. Invoke program without IP addresses and packet trace files
Input data. Analyzer options without required IP addresses and file names
Expected outcome. Error message

Test 4_2 (negative)
Description. Invoke program with large number of IP addresses
Input data. Analyzer options with a large number of IP addresses
Expected outcome. Error message

Test 4_3 (negative)
Description. Invoke program with invalid number of packet trace files
Input data. Analyzer options with wrong number of file names
Expected outcome. Error message

Test 4_4 (negative)
Description. Invoke program with invalid IP address format
Input data. Analyzer options with IP address(es) in invalid format
Expected outcome. Error message

Test 4_5 (negative)
Description. Invoke program with invalid time alignment
Input data. Analyzer options with invalid time alignment
Expected outcome. Error message

Test 4_6 (negative)
Description. Invoke program with invalid time alignment format
Input data. Analyzer options with time alignment in invalid format
Expected outcome. Error message

Test 4_7 (negative)
Description. Invoke program with more than MAX_PORTS ports specified
Input data. Analyzer options with more than MAX_PORTS ports specified
Expected outcome. Error message

Test 4_8 (negative)

Description. Invoke program with invalid port specified
Input data. Analyzer options with invalid port number
Expected outcome. Error message

Test 4_9 (negative)
Description. Invoke program with port which doesn’t fit in 16 bit integer
Input data. Analyzer options with invalid port number
Expected outcome. Error message

4.2 Log reader

Test 1_1 (common)
Description. Read example packet trace files for scenario 201
Input data. Analyzer options with defined IP addresses and packet trace logs for scenario 201
Expected outcome. Analyzer reads packet trace files and forms array of ptu sequences

---

Test 1_2 (common)
Description. Read example packet trace files for scenario 310
Input data. Analyzer options with defined IP addresses and packet trace logs for scenario 310
Expected outcome. Analyzer reads packet trace files and forms array of ptu sequences

---

Test 1_3 (common)
Description. Read example packet trace files for scenarios 321, 341, 361
Input data. Analyzer options with defined IP addresses and packet trace logs for scenarios 321, 341, 361
Expected outcome. Analyzer reads packet trace files and forms array of ptu sequences

---

Test 1_4 (common)
Description. Read example packet trace files for scenario 362
Input data. Analyzer options with defined IP addresses and packet trace logs for scenario 362
Expected outcome. Analyzer reads packet trace files and forms array of ptu sequences

---

Test 1_5 (common)
Description. Read example packet trace files for scenario 411
Input data. Analyzer options with defined IP addresses and packet trace logs for scenario 411
Expected outcome. Analyzer reads packet trace files and forms array of ptu sequences

---

Test 1_6 (common)
Description. Read example packet trace files for scenario 101
Input data. Analyzer options with defined IP addresses and packet trace logs for sce-

nario 101
Expected outcome. Analyzer reads packet trace files and forms array of ptu sequences

Test 1_7 (common)
Description. Read example packet trace files for scenario 333
Input data. Analyzer options with defined IP addresses and packet trace logs for scenario 333
Expected outcome. Analyzer reads packet trace files and forms array of ptu sequences

4.3 Message mapper

Test 1_1 (common)
Description. Read example packet trace files for scenario 201
Input data. Analyzer options with defined IP addresses and packet trace logs for scenario 201
Expected outcome. Analyzer reads packet trace files and forms ptp list with established links

Test 1_2 (negative)
Description. Read example packet trace files for scenario 201 with wrong sequence of IP addresses
Input data. Analyzer options with packet trace logs for scenario 201 and wrong sequence of IP addresses
Expected outcome. Error message or empty ptp list

Test 1_3 (common)
Description. Read example packet trace files for scenario 201 with wrong sequence of IP address
Input data. Analyzer options with packet trace logs for scenario 201 and wrong sequence of IP addresses
Expected outcome. Analyzer reads packet trace files and forms ptp list without established links

Test 2_1 (negative)
Description. Read example packet trace files for scenario 310 with wrong sequence of IP addresses
Input data. Analyzer options with packet trace logs for scenario 310 and wrong sequence of IP addresses
Expected outcome. Error message or empty ptp list

Test 2_2 (common)
Description. Read example packet trace files for scenario 310
Input data. Analyzer options with defined IP addresses and packet trace logs for scenario 201

Expected outcome. Analyzer reads packet trace files and forms ptp list with established links

---

Test 3_1 (negative)
Description. Read example packet trace files for scenario 321, 341, 361 with wrong sequence of IP addresses
Input data. Analyzer options with packet trace logs for scenario 321, 341, 361 and wrong sequence of IP addresses
Expected outcome. Error message or empty ptp list

Test 3_2 (common)
Description. Read example packet trace files for scenario 321, 341, 361
Input data. Analyzer options with defined IP addresses and packet trace logs for scenario 321, 341, 361
Expected outcome. Analyzer reads packet trace files and forms ptp list with established links

---

Test 4_1 (negative)
Description. Read example packet trace files for scenario 362 with wrong sequence of IP addresses
Input data. Analyzer options with packet trace logs for scenario 362 and wrong sequence of IP addresses
Expected outcome. Error message or empty ptp list

Test 4_2 (common)
Description. Read example packet trace files for scenario 362
Input data. Analyzer options with defined IP addresses and packet trace logs for scenario 362
Expected outcome. Analyzer reads packet trace files and forms ptp list with established links

---

Test 5_1 (negative)
Description. Read example packet trace files for scenario 101 with wrong sequence of IP addresses
Input data. Analyzer options with packet trace logs for scenario 101 and wrong sequence of IP addresses
Expected outcome. Error message or empty ptp list

Test 5_2 (special)
Description. Read example packet trace files for scenario 101 with wrong IP addresses
Input data. Analyzer options with wrong IP addresses and packet trace logs for scenario 101
Expected outcome. empty ptp list

Test 5_3 (common)

Description. Read example packet trace files for scenario 101
Input data. Analyzer options with defined IP addresses and packet trace logs for scenario 101
Expected outcome. Analyzer reads packet trace files and forms ptp list with established links

---

4.4 Events calculator

Test 1_1 (common)
Description. Read example packet trace files for scenario 201 without defined port numbers for application level
Input data. Analyzer options with packet trace files for scenario 201 without defined ports
Expected outcome. Analyzer reads packet trace logs and forms events sequence with unknown application protocol if it is not used predefined ports

---

Test 1_2 (special)
Description. Read example packet trace files for scenario 201 with defined application ports
Input data. Analyzer options with packet trace files for scenario 201 and port numbers
Expected outcome. Analyzer reads packet trace logs and forms events sequence

---

Test 2_1 (common)
Description. Read example packet trace files for scenario 310 without defined port numbers for application level
Input data. Analyzer options with packet trace files for scenario 310 without defined ports
Expected outcome. Analyzer reads packet trace logs and forms events sequence with unknown application protocol if it is not used predefined ports

---

Test 2_2 (special)
Description. Read example packet trace files for scenario 310 with defined application ports
Input data. Analyzer options with packet trace files for scenario 310 and port numbers
Expected outcome. Analyzer reads packet trace logs and forms events sequence

---

Test 3_1 (common)
Description. Read example packet trace files for scenario 321, 341, 361 without defined port numbers for application level
Input data. Analyzer options with packet trace files for scenario 321, 341, 361 without defined ports
Expected outcome. Analyzer reads packet trace logs and forms events sequence with unknown application protocol if it is not used predefined ports

---

Test 3_2 (special)
Description. Read example packet trace files for scenario 321, 341, 361 with defined application ports
Input data. Analyzer options with packet trace files for scenario 321, 341, 361 and port numbers
Expected outcome. Analyzer reads packet trace logs and forms events sequence

---

Test 4_1 (common)
Description. Read example packet trace files for scenario 362 without defined port numbers for application level
Input data. Analyzer options with packet trace files for scenario 362 without defined ports
Expected outcome. Analyzer reads packet trace logs and forms events sequence with unknown application protocol if it is not used predefined ports

---

Test 4_2 (special)
Description. Read example packet trace files for scenario 362 with defined application ports
Input data. Analyzer options with packet trace files for scenario 362 and port numbers
Expected outcome. Analyzer reads packet trace logs and forms events sequence

---

Test 5_1 (common)
Description. Read example packet trace files for scenario 333 without defined port numbers for application level
Input data. Analyzer options with packet trace files for scenario 333 without defined ports
Expected outcome. Analyzer reads packet trace logs and forms events sequence with unknown application protocol if it is not used predefined ports

---

Test 5_2 (special)
Description. Read example packet trace files for scenario 333 with defined application ports
Input data. Analyzer options with packet trace files for scenario 333 and port numbers
Expected outcome. Analyzer reads packet trace logs and forms events sequence

---

Test 6_1 (common)
Description. Read example packet trace files for scenario 411 without defined port numbers for application level
Input data. Analyzer options with packet trace files for scenario 411 without defined ports
Expected outcome. Analyzer reads packet trace logs and forms events sequence with unknown application protocol if it is not used predefined ports

---

Test 6_2 (special)
Description. Read example packet trace files for scenario 411 with defined application

ports
Input data. Analyzer options with packet trace files for scenario 411 and port numbers
Expected outcome. Analyzer reads packet trace logs and forms events sequence

---

Test 7_1 (common)
Description. Read example packet trace files for scenario 101 without defined port numbers for application level
Input data. Analyzer options with packet trace files for scenario 101 without defined port numbers
Expected outcome. Analyzer reads packet trace logs and forms events sequence with unknown application protocol if it is not used predefined ports

---

Test 7_2 (special)
Description. Read example packet trace files for scenario 101 with defined application ports
Input data. Analyzer options with packet trace files for scenario 101 and port numbers
Expected outcome. Analyzer reads packet trace logs and forms events sequence

---

Test 7_3 (negative)
Description. Read example packet trace files for scenario 101 with wrong IP addresses
Input data. Analyzer options with packet trace files for scenario 101 and wrong IP addresses
Expected outcome. Analyzer reads packet trace logs and forms empty events sequence

---

4.5 PEF writer

Test 0_1 (basic)
Description. write pef without data
Input data. none
Expected outcome. Analyzer writes empty pef

Test 1_1 (common)
Description. Read example packet trace files for scenario 201 without defined port numbers for application level
Input data. Analyzer options with packet trace files for scenario 201 without defined ports
Expected outcome. Analyzer reads packet trace logs and writes pef with unknown application protocol if it is not used predefined ports

---

Test 1_2 (special)
Description. Read example packet trace files for scenario 201 with defined application ports
Input data. Analyzer options with packet trace files for scenario 201 and port numbers
Expected outcome. Analyzer reads packet trace logs and writes pef

Test 1_3 (special)
Description. Read example packet trace files for scenario 201 with wrong sender IP address
Input data. Analyzer options with packet trace files for scenario 201 and wrong sender IP address
Expected outcome. Analyzer reads packet trace logs and writes empty pef

Test 1_4 (special)
Description. Read example packet trace files for scenario 201 with wrong receiver IP address
Input data. Analyzer options with packet trace files for scenario 201 and wrong receiver IP address
Expected outcome. Analyzer reads packet trace logs and writes pef with dropped packets

Test 2_1 (common)
Description. Read example packet trace files for scenario 310 without defined port numbers for application level
Input data. Analyzer options with packet trace files for scenario 310 without defined ports
Expected outcome. Analyzer reads packet trace logs and writes pef with unknown application protocol if it is not used predefined ports

Test 2_2 (special)
Description. Read example packet trace files for scenario 310 with defined application ports
Input data. Analyzer options with packet trace files for scenario 310 and port numbers
Expected outcome. Analyzer reads packet trace logs and writes pef

Test 3_1 (common)
Description. Read example packet trace files for scenario 321, 341, 361 without defined port numbers for application level
Input data. Analyzer options with packet trace files for scenario 321, 341, 361 without defined ports
Expected outcome. Analyzer reads packet trace logs and writes pef with unknown application protocol if it is not used predefined ports

Test 3_2 (special)
Description. Read example packet trace files for scenario 321, 341, 361 with defined application ports
Input data. Analyzer options with packet trace files for scenario 321, 341, 361 and port numbers
Expected outcome. Analyzer reads packet trace logs and writes pef

Test 4_1 (common)
Description. Read example packet trace files for scenario 362 without defined port numbers for application level
Input data. Analyzer options with packet trace files for scenario 362 without defined ports
Expected outcome. Analyzer reads packet trace logs and writes pef with unknown application protocol if it is not used predefined ports

---

Test 4_2 (special)
Description. Read example packet trace files for scenario 362 with defined application ports
Input data. Analyzer options with packet trace files for scenario 362 and port numbers
Expected outcome. Analyzer reads packet trace logs and writes pef

---

Test 5_1 (common)
Description. Read example packet trace files for scenario 333 without defined port numbers for application level
Input data. Analyzer options with packet trace files for scenario 333 without defined ports
Expected outcome. Analyzer reads packet trace logs and writes pef with unknown application protocol if it is not used predefined ports

---

Test 5_2 (special)
Description. Read example packet trace files for scenario 333 with defined application ports
Input data. Analyzer options with packet trace files for scenario 333 and port numbers
Expected outcome. Analyzer reads packet trace logs and writes pef

---

Test 6_1 (common)
Description. Read example packet trace files for scenario 411 without defined port numbers for application level
Input data. Analyzer options with packet trace files for scenario 411 without defined ports
Expected outcome. Analyzer reads packet trace logs and writes pef with unknown application protocol if it is not used predefined ports

---

Test 6_2 (special)
Description. Read example packet trace files for scenario 411 with defined application ports
Input data. Analyzer options with packet trace files for scenario 411 and port numbers
Expected outcome. Analyzer reads packet trace logs and writes pef

---

Test 7_1 (common)
Description. Read example packet trace files for scenario 101 without defined port numbers for application level

Input data. Analyzer options with packet trace files for scenario 101 without defined port numbers
Expected outcome. Analyzer reads packet trace logs and writes pef with unknown application protocol if it is not used predefined ports

---

Test 7_2 (special)
Description. Read example packet trace files for scenario 101 with defined application ports
Input data. Analyzer options with packet trace files for scenario 101 and port numbers
Expected outcome. Analyzer reads packet trace logs and writes pef

---

Test 7_3 (special)
Description. Read example packet trace files for scenario 101 with wrong IP addresses
Input data. Analyzer options with packet trace files for scenario 101 and wrong IP addresses
Expected outcome. Analyzer reads packet trace logs and writes empty pef

---

Test 8_* (common)
Description. Read packet trace files produced by DaCoPAn team
Input data. Analyzer options with packet trace files
Expected outcome. Analyzer reads packet trace files and writes pef if it is possible

---

5 Validation testing

The testing strategy was described in section 2. Validation/acceptance testing is an essential part of the testing phase. The Analyzer system is tested as a result product of the DaCoPAn project. This testing is behavioral model based and ensures that all features of the DaCoPAn system meet their requirements. The basic input data for validation testing are the packet trace log files which are introduced by Customer. Also other packet trace log files are the input data for validation testing. Each network scenario [2] must be supported by Analyzer with corresponding priority.

5.1 Behaviors and Use cases

The following behaviors must be checked:

Produce a protocol events file
This behavior is described in section **5.2.1 "Use case: produce a protocol events file"** [3] and section **7.1 "Produce a protocol events file"** [1]. User gets two packet trace log files and Analyzer creates Protocol events file.

Get usage info
This behavior is described in section **7.2 “Get usage info”** [1]. User specified option `--help` and Analyzer prints short user’s manual.

Get program info
This behavior is described in section **7.3 “Get program info”** [1]. User specified option `--version` and Analyzer prints information about program version, etc.

5.2 Networking scenarios

The networking scenarios are described on [2]. The scenarios with implementation priority #1 must be supported by Analyzer. The scenarios with implementation priorities #2 and #3 may be supported by Analyzer.

There are following networking scenarios that mast be checked:

201 – IP packet delivery and content
The input packet trace files contains IP datagrams. The output Protocol events file contains events “IP datagram sent” and “IP datagram received”. Also PEF includes information about protocol headers and variables.

310 – IP fragmentation and UDP data transfer
The input packet trace files contains IP datagrams and fragmented UDP packets. The output Protocol events file contains events “IP datagram sent”, “IP datagram received”, “UDP packet sent”, “UDP packet received”. Also PEF includes information about protocol headers and variables, UDP packet fragmentation and protocol encapsulation.

321 – Three-way TCP connection establishment
The input packet trace files contains TCP packets, information about connection establishment. The output Protocol events file contains events “TCP packet sent” and “TCP packet received”. Also PEF includes information about protocol headers and variables. The important information are a SYN and ACK flags.

341 – TCP normal FIN termination
The input packet trace files contains TCP packets, information about connection termination. The output Protocol events file contains events “TCP packet sent” and “TCP packet received”. Also PEF includes information about protocol headers and variables. The important information are a FIN and ACK flags.

361 – Slow start and sending data with TCP
The input packet trace files contains TCP packets. The output Protocol events file contains

events “TCP packet sent” and “TCP packet received”. Also PEF includes information about protocol headers and variables.

362 – TCP recovery through fast retransmit
The input packet trace files contains TCP packets. Some of these packets are lost. The output Protocol events file contains events “TCP packet sent”, “TCP packet received” and “TCP packet lost”. Also PEF includes information about protocol headers and variables. The important information are a lost packets.

5.3 User requirements

The user requirements are described on [3]. These requirements must be taken into account during test execution.

References

1 DaCoPAn Software Engineering project, *Design: Analyzer*. Release 1.0. Universities of Helsinki and Petrozavodsk, April 2004.

2 DaCoPAn Software Engineering project, *Networking scenarios*. Release 1.0. Universities of Helsinki and Petrozavodsk, March 2004.

3 DaCoPAn Software Engineering project, *Requirements specification*. Release 1.0. Universities of Helsinki and Petrozavodsk, March 2004.