Requirements specification

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
1.00	02.03.2004	First published version of the document. To be revised with the customer.

Contents

1 Introduction

This document describes the requirements for the DaCoPAn project. This document has been written as a contract between the customer of the final product and the teams planning and developing the project. It is also a guideline for designers and developers about the functionalities performed by the final product and the problems that it solves. This will be the first version of the final product, what means that some requirements are also focused in the extension of the project by other groups in the future. Also in this document are prioritized some of the features for the system that will be implemented by this software engineering group showing which could be left for future groups to be implemented.

The DaCoPAn project addresses playing back animations of packet trace information captured from real data communication traffic. Serving as a tool for different users for self-learning, teaching and researching.

The following contents of the document are structured in this way:

User requirements gathering requirements from each part of the system by telling what that entity should do.

Next comes the problem model domain with explanations about the system scope, a high-level division model explaining the subsystems to be developed and their problems, data model presenting the requirements about the data used and processed by the system and the constraints gathering requirements for expandability, performance, environments of operation and non-functional requirements.

Another section collects networking scenarios that are going to be used as base to develop and test the program. In these scenarios is presented an outline of the behaviour of the network protocols and the values of header fields and implicit variables that are needed to visualize. All these scenarios are prioritized in three categories the last one of lower priority just present cases to be implemented in future versions of the product and are written to be a reference for a design that will allow future extensions.

The document continues with a list of use cases representing the different uses and actions that the different actors can have over the system. The main actors are teachers, researchers and students and they use the functions offered by the product in a different way and with different intersts.

In the next section the system requirements are presented for the subsystems composing the product and the intermediate file that serves as bridge between them, explaining in detail the different functions that should perform.

Finally a section showing the high level architecture of the system and making a presentation of the different components that should be develop for each subsystem from the point of view of their functionality .

2 User requirements

According to the discussions with the customer, the product is divided in two subproducts, the Analyzer and the Animator. In this section the requirements given explicitly by the customer are listed.

2.1 Analyzer

This part of the product will be a program that takes two source logs collected by a tool called *tcpdump*, merges and processes the data and then stores it in the protocol events file readable by the animator part. The basic user requirements for the analyzer are presented in the list below.

Tcpdump logs. The analyzer requires two tcpdump log files. It also needs two IP addresses for selecting two hosts from the tcpdump logs. All possible information which is relevant to any of these hosts should be stored in the file created by the Analyzer. Other information will be ignored. User of the analyzer can filter unwanted messages out of the logs for example by using tcpdump filters.

Calculations. Not all data is explicitly stored in tcpdump logs, e.g. TCP states or protocol variables. Special calculations, perhaps with additional logs, are needed to get this data. The basic version of the Analyzer will support these calculations on a minimal required level, but the output file format must be designed to be able to store all this data.

Supported protocols. The analyzer should support HTTP, DNS and perhaps another UDP-based application on the application layer.

- In the case of HTTP, the analyzer should be able to figure out which TCP segments make up a HTTP request (or response) and output this information to the protocol events file. All HTTP headers should also be included in the file.
- For DNS, the analyzer should extract all relevant DNS protocol fields from the UDP packets that contain the DNS request and output the field values to the protocol events file.

The analyzer should also support TCP and UDP on the transport layer and IP on the network layer. The UDP-based application doesn't have to be specified — the idea is that a teacher user can use whatever UDP-based program to produce tcpdump logs presenting large packets and IP fragmentation. ARP communication should be present in the protocol events file format in some way even though the link level in general is not required.

Time difference. If there is time difference between the two tcpdump logs, the user must enter the difference manually. If no time difference is set, the analyzer assumes that

the times are synchronized. The order of protocol exchange should be determined based on the timestamps in the logs.

IP reordering. In the design phase, attention should be paid to the fact that IP packets might not arrive in the same order they were sent in. Packets arriving in wrong order should thus not be discarded.

Message delay. The delays between messages should be included in the output file. In case the time difference is not set correctly, the data in the file can be incorrect.

IP version. The analyzer supports IP version 4. A possible expansion to version 6 should be taken into account at the design for the protocol events file format.

2.2 Animator

The animator part of the product reads in protocol events files written by the analyzer and animates the communication and other data present in the file. Alternatively, the Animator may read in a Scenario file that contains the animation data but also contains information about a sequenced presentation of the data and possibly textual notes to be presented to the user.

In the following list some basic requirements for the animator are presented.

Animation. The Animator should be able to visualize the protocol exchange between the two end hosts using suitable animations that clearly show the details of the exchange. MSC (message sequence chart) is a traditional example (used extensively in literature) of such an animation.

Encapsulation. The encapsulation of protocol data from higher level units to lower level units should be visualized using a suitable animation type.

Mute. The user should be able to start and pause the animation and rewind and forward it by steps of one or more animation units.

Animation speed. The user should be able to select the speed of the animation. The speed can be based in "real time" (i.e. real time scaled in proportion to fit the needs of the animation) or in the units transferred (i.e. animation ticks).

Animation flow. The user should be able to select the network traffic flows to be shown in the animation. The flows should be presented some way so that they are easily distinguished, for example in different colors. (Note: This requirement will be removed if we decide to limit the number of simultaneous flows to one flow)

Animation layer. The user should be able to select a specific layer (application, transport or network) for viewing on the fly. Encapsulation of the layers should be present in the animation in some way refined later.

Animation configuration. The user should be able to configure the exact set of host variables and protocol header fields to be shown. The selection is done out of the information present in the loaded animation data file and thus depends on what information the Analyzer saves in the file. The configuration can be saved and later loaded into the program as part of the Scenario file. The user may be able to view detailed information on a packet for example by clicking it. Basic (relevant to current scenario) information is visible all the time.

Relevant information. Relevant information of both endpoints should be shown upon user request. This includes port number, ip address, hostname etc. (Note: This information is shown only if it has been provided by the user running the analyzer. Neither the Analyzer or the Animator is expected to make network connections to find out this information by e.g. making DNS queries.)

Breakpoints. Optional: The user should be able to specify breakpoints in the animation, i.e. specific times when the animation automatically pauses for input from the user. The breakpoints should be specified by using the Animator user interface and saved in the Scenario file format.

Notes. Optional: The author of the presentation should be able to add notes (comments) to the presentation. The notes should then be shown by the animator to the user.

ARP animation. Optional: The occurrence of ARP communication should be shown in the presentation in some way.

3 Problem domain model

3.1 System scope model

The system scope model is used to represent relations between the product and external entities. It reflects product purposes, input data requirements and restrictions, user abilities, and possible directions of product further development. The system scope model diagram is shown at Figure 1.

3.2 High-level division model

For distributed development, the problem must be divided into two parts. As per the customer's requirements, the product should be divided into separate software programs: an analyzer, which takes the source logs, merges the data (message mapping) and stores it in the intermediate format (protocol events file), and the animation program (animator), which takes the protocol events file as an input source and performs the animation.

Possible problems for the SE subsystems development:

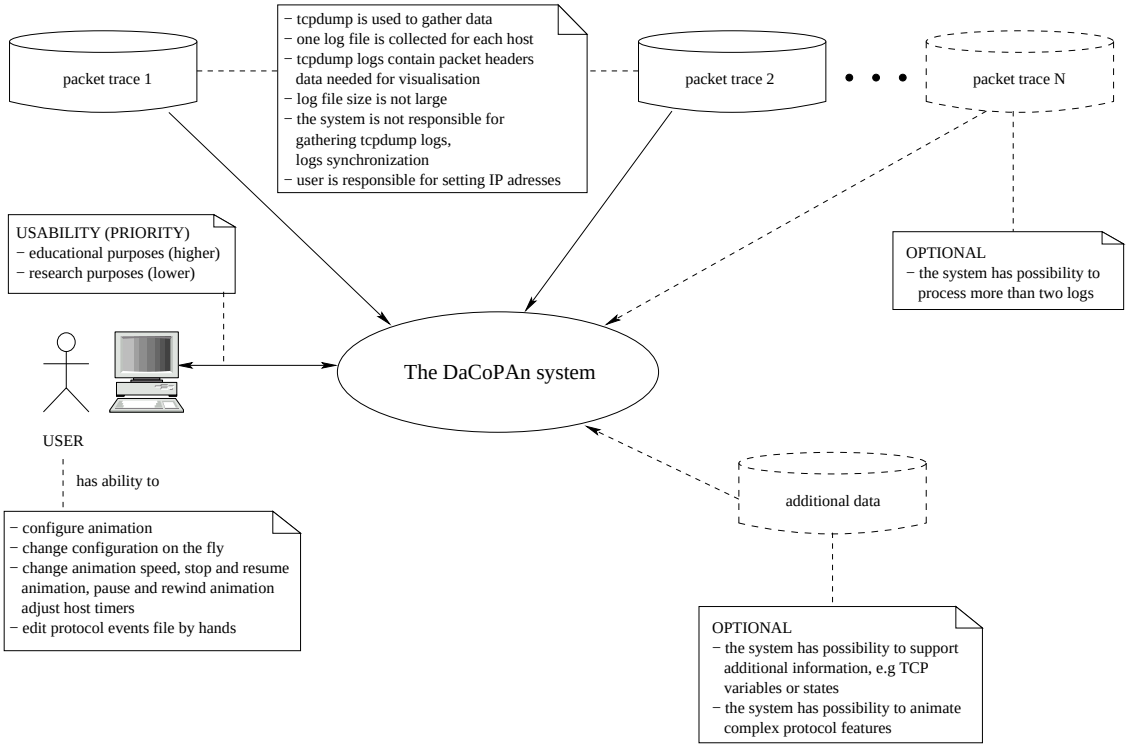

Figure 1: System scope model

Analyzer: Hard to test; requires comprehensive collection of different network traffic logs and access to gather the real network traffic. Developers have to have good enough skills in system programming, algorithms and data structures. Deep knowledge of internet protocols is also required to construct the messages mapping algorithm and to design an expandable file format for protocol events.

Animator: Developers need to interact with the customer to discuss prototypes of possible user-interfaces and scenarios of animations. User interface expertise is required to design and implement this subsystem. The educational and research aspects should be combined in a balanced proportion. The animator developers need to prepare a list of requirements which the data protocol events file format should fulfil.

Protocol events file: The protocol event file is an interface between the analyzer and the animator. It has to store all the necessary (implicit and explicit) information for the animator. The current consensus for the file format is an XML solution. This will make it more simple for the animator to parse, and the storage of the information is well structured and sufficiently extensible. In the planning and design phases each group needs to maintain sufficient communication with the other in order to facilitate the development of a good file format. An initial list of information and requirements for the animator will be needed in the development of the protocol events file format.

3.3 Data model

This section presents different data files used in the DaCoPAn system.

3.3.1 End host log

The end host log is a packet trace file for internet protocols gathered using the tcpdump tool. Tcpdump catches network traffic which is then stored in a log file in the format specified. All necessary data for network protocol animation (e.g identifiers for IP, sequence numbers for TCP, source and destination ports for UDP, etc.) can be extracted from corresponding packet headers. It is enough to have complete packet header information for basic animation. The log is in binary format. Real or simulated data communication traffic can be a source for this log.

End host logs are always gathered in such a way that the problem of their analysis is solvable. Built in filters of tcpdump should be used to leave unwanted data out of the log files. IP identifiers should be used for constructing the mapping between end host logs. The tcpdump log file can contain network traffic between the two selected hosts and between either one of them and a third party (e.g. ARP lookup, DNS).

A collection of such logs should be available which cover all required scenarios.

3.3.2 Additional input logs

The basic version of the DaCoPAn system does not support additional input logs in the analysis process. For later extensions, this possibility must be taken into account in the design (e.g. TCP variables, states, etc.).

3.3.3 Protocol events log

It is a consolidated sequence of protocol events in chronological order produced by the analyzer. All relevant information from end host logs and additional input logs must be stored. The basic case is two end hosts but the design should support later extensions for a greater number of hosts.

Although the analyzer does not support additional input logs in the scope of this project, the format of analyzed data must allow adding certain information manually.

Time stamps for each end host are stored as adjusted in the analyzer.

3.4 System constraints

3.4.1 Basic cases and expandability requirements

The software animates the data communication between two hosts. Possible networking scenarios are represented in [1]. The software must support at least the scenarios that have

the highest priority, and optionally may support other scenarios.

File formats should be designed so that they are easily compatible with future versions of the software, and with different versions of the product during the project's life cycle. The software doesn't need to be tolerant of formatting errors and the software doesn't need to give specific and verbose error messages if there are unexpected errors in the files as a result of manual editing.

3.4.2 Non-functional requirements for the Animator

Usability
The Teacher user is expected to be able to take the required time to study the documentation so that he can learn to prepare scenario files. A teacher user should not have to spend more than two hours studying user documentation and the program before able to prepare custom presentations.

The student user is the main usability concern. This user must be able to load and view animations without any extra help files and without spending time learning the basic functions of the program. He/she may need to spend some time learning the more advanced functions of the program, e.g. changing the animation settings or saving his own modified scenario files. Modification should be possible without additional help files, but a small amount of trial and error is acceptable. If the student wants to save modified scenario files, the same expectations apply to him as does to the teacher user.

Download size
The Animator software should be easily accessible by students using any computer connected to the internet, such as a home workstation. Ideally the time it takes to download and start the application should be no more than 2-3 minutes (provided that the workstation already has the Java runtime environment installed). This in turn implies that the total download size of the Animator application should preferably be 500 kilobytes or less.

Localization
The Animator should be localized to be usable in Finnish, Russian and Spanish in addition to the English version. This implies that the software should be designed so that localizing can be done in fast and simple way, using a separate XML or other textual file interface. (Note: this is an optional requirement)

Reliability
The worst damage from program failure should be nothing more than user frustration. However, the language properties of Java facilitate building errorfree and reliable software, so the program should be free of errors to the highest feasible degree.

Security
The program does not process private or confidential data, so there are no security issues involved in the program.

Maintainability
In order to allow for future enhancements to the animator, the code base should be written using clear and consistent coding conventions and thoroughly tested. Any design

decisions made in the design phase should be well documented: for example, using well established design patterns is the aim of the project, as they will make the code easily approachable for any teams assigned to add new features to the application in the future.

3.4.3 Performance requirements

Analyzer has no strict performance requirements, but it must not take an unreasonable amount of time or system resources to produce an intermediate file from the source logs with at most a few hundred of packets (a typical scenario is expected to consist of tens of packets).

Animator must produce the animation at the requested speed (no more than 2 packet events per second) without any delays that would be disturbing on a 500 MHz CPU with 128 MB RAM. (Java 1.4)

3.4.4 Operational requirements

Operational requirements are different for the analyzer and animator parts of the software:

Analyzer must work in the Linux operating system, must be invokable from the command-line and support arguments for specifying two log files and two IP addresses (required arguments). Some additional (optional) arguments may be supported as well (at least the time difference argument).

Animator must work on any operating system platform supporting Java 1.4 as a stand-alone application. The possibility of launching the software from inside a web browser (as a web startable application) is considered optional.

4 Networking scenario descriptions

The primary goal for the product pair created in this project is to teach networking principles to students at different level of knowledge. For this use it is useful to know what kind of thing they should be able to learn with the tool. In this section the most important animation scenarios for the student are listed in order to give the reader some understanding about the teaching goals for the product but to present one of the bases for requirement gathering. More information about the networking scenarios can be found in another document titled Networking scenarios.

The networking scenarios are divided into 3 priority groups. Priority 1 scenarios are the scenarios that the product will support in its first version. Priority 2 scenarios will probably be supported, but can be dropped in case of time critical situations. Priority 3 scenarios could be implemented in further extensions of the program. One aim of the project is that the program could be extendable to other network protocols or behaviours later on, so these scenarios are a key point for achieving this purpose.

4.1 List of variables

List of variables, layers of use and their scenarios:

- MTU value - Network layer: 201, 310
- TCP states - Transport layer: 321, 341, 322, 351
- Congestion window cwnd - Transport layer: 361, 362
- Slow start threshold size ssthresh - Transport layer: 362
- Retransmission Timeout (RTO) - Transport layer: 362, 333
- Header fields: questions and answers - Application layer: 421
- Duplicated ACK (dupack) Threshold - Transport Layer: 362 (The value of this variable is difficult to calculate; therefore it will be up to the user to introduce its value manually).

List of variables per priority and scenario:

- Priority 1:
  - 201 IP Packet delivery and content - Network layer
    MTU value
  - 310 IP Fragmentation and simple UDP data transfer - Network layer
    MTU value
  - 321 Three-way TCP connection establishment - Transport layer
    TCP states
  - 341 TCP normal FIN termination - Transport layer
    TCP states
  - 361 Slow start and sending data with TCP - Transport layer
    Congestion window (cwnd)
  - 362 TCP Recovery through fast retransmit (happening in slow start with packet loss) - Transport layer
    Congestion window (cwnd), slow start threshold size (ssthresh), Retransmission TimeOut (RTO), Duplicated ACK (dupack) Threshold
- Priority 2:
  - 333 TCP packet loss and recovery through RTO - Transport layer
    Retransmission TimeOut (RTO)
  - 411 HTTP: Simple request with reply - Application layer
    None particular
  - 101 ARP Lookup - ARP layer
    None particular
  - 322 TCP Timeout of Connection Establishment - Transport layer
    TCP states
- Priority 3:
  - 351 TCP Connection reset (RST) - Transport layer
    TCP states
  - 412 HTTP: Retrieval with multiple connections - Application layer
    None particular
  - 421 DNS request for A record - Application layer
    Header fields: requests and answers

4.2 Notes

- All the following networking scenarios need as variables timestamps, so it will not be present in the following description, but this variable is needed.

- The names for protocol header fields are the used in their respective RFCs
- The protocol header fields in the layers description contain the header fields which are relevant for educational purposes, but the protocol events file should contain, if possible, all the header fields of the protocol.
- By default the following scenarios try to be as simplified as possible and do not support delayed ACKs nor DNS lookups nor connection establishment. Unless that it is explicitly written.
- Some of the scenarios presented with priority 3 do not present list of variables nor protocol header fields. They should be filled in revisions of the document for future improvements.

4.3 Implementation priority #1

4.3.1 201 - IP packet delivery and content

Description: A sends an IP datagram to B.

Relevant layers: Network layer

Network layer.

Protocol header fields	Total Length, Datagram Identification, Flags (DF=1, MF), Offset, Source IP Address, Destination IP Address, *Version, Header Length, Time To Live, Checksum*
Variables	MTU

4.3.2 310 - IP fragmentation and simple UDP data transfer

Description: This scenario combines IP fragmentation and simple UDP transfer. A sends an UDP packet to B. This is sent using IP and the size of the data to be sent exceeds the MTU value of the network. The data is fragmented into some IP datagrams and sent to B. B receives them and proceeds to their defragmentation. At this point the original UDP packet sent is received in B.

Relevant layers: Network layer, Transport layer

Transport layer.

Protocol header fields	Source Port, Destination Port, Length
Variables	-

Network layer.

Protocol header fields	Total Length, Datagram Identificator, Flags (MF, DF), Offset
Variables	MTU value

4.3.3 321 - Three-way TCP connection establishment

Description: Endpoint A (client) wants to start a communication with B (server) to transfer information using the TCP protocol. Then uses the three-way TCP connection establishment mechanism to assure that the server can attend its request. Three TCP packets are sent to the network in this process. The first one from A to B specifying that A wants to connect to B and the port to do it. In the second the availability of B is confirmed to A by an acknowledgement. Finally A acknowledges B the reception of its last packet sent. At this point the TCP connection is established. Maximum segment size negotiation between both hosts should be visualized.

Relevant layers: Transport layer

Transport layer.

Protocol header fields	Sequence Number, Acknowledgment Number, Flags (SYN, ACK), Maximum Segment Size (Options with Kind=3)
Variables	TCP states

4.3.4 341 - TCP normal FIN termination

Description: A client host wants to end the TCP connection existing between it and a server host. Four packets are needed to close the connection. Client sends FIN packet to server (sequence number x), which acknowledges (sequence number x+1). Then server sends FIN packet to client (sequence number y), then client acknowledges (sequence number y+1). The connection is then closed.

Relevant layers: Transport layer

Transport layer.

Protocol header fields	Sequence Number, Acknowledgment Number, Flags (FIN, ACK)
Variables	TCP states

4.3.5 361 - Slow start and sending data with TCP

Description: A TCP connection has been established between 2 hosts. The slow start algorithm controls TCP flow. It operates by observing that the rate at which new packets should be injected into the network is the rate at which the acknowledgments are returned by the other end, using the congestion window, called cwnd.

Relevant layers: Transport layer

Transport layer.

Protocol header fields	Sequence Number, Acknowledgment Number, Flags (ACK), Maximum Segment Size (Options with Kind=3)
Variables	Congestion window (cwnd)

4.3.6 362 - TCP Recovery through fast retransmit (happening in slow start with packet loss)

Description: This scenario starts with the normal slow start defined in the scenario number 361 and once a packet is lost the TCP fast retransmit algorithm starts. TCP is required to generate an immediate acknowledgment when an out-of-order segment is received. This duplicate ACK should not be delayed. The purpose of this duplicate ACK is to let the other end know that a segment was received out of order, and to tell it what sequence number is expected. From A point of view, duplicate ACK is caused by a lost segment or just a reordering of segments, so it waits for a small number of duplicate ACKs (dupack threshold) to be received. If three or more duplicate ACKs are received in a row, it is a strong indication that a segment has been lost. The client then performs a retransmission of what appears to be the missing segment, without waiting for a retransmission timer to expire. Slow start is not performed after the three duplicate ACKs are received because the receipt of the duplicate ACKs tells the client that a packet has been lost. Since the receiver can only generate the duplicate ACK when another segment is received, that segment has left the network and is in the receiver's buffer. That is, there is still data flowing between the two ends, and we don't want to reduce the flow abruptly by going into slow start. Next, congestion avoidance, but not slow start is performed.

Relevant layers: Transport layer

Transport layer.

Protocol header fields	Sequence Number, Acknowledgment Number, Flags (ACK), Maximum Segment Size (Options with Kind=3)
Variables	Congestion Window (cwnd), slow start threshold size (ssthresh), Retransmission TimeOut (RTO), Duplicated ACK (dupack) Threshold

4.4 Implementation priority #2

4.4.1 333 - TCP packet loss and recovery through RTO

Description: The idea is to show a simple timeout and retransmission example. A (the client) sends a TCP packet in an established TCP connection to B (server). If after a given timeot expires (RTO: Retransmission Timeout) the acknowledgement for the packet has not been received, the packet is sent again. In the meanwhile B is waiting for more packets to arrive, if a new packet is received it answers with an acknowledgement packet.

Relevant layers: Transport layer

Transport layer.

Protocol header fields	Sequence Number, Acknowledgment Number, Flags (ACK)
Variables	Retransmission TimeOut (RTO)

4.4.2 411 - HTTP: Simple request with reply

Description: A client (web browser) requests an HTML page from a server and receives it using HTTP protocol. No embedded objects will be included in the transfered HTML page (such as images, ...).

Relevant layers: Application layer

Application layer.

Protocol header fields	**Request** = Request line + general header + request header + entity header + message body **Response** = Status line + general header + entity header + message body
Variables	-

Transport layer.

Protocol header fields	-
Variables	-

4.4.3 101 - ARP Lookup

Description: A has the IP address of B and wants to know B's physical ethernet address. A sends broadcast ARP request for the address of B, B recognizes its IP address and answers to A.

Relevant layers: ARP layer

ARP layer.

Protocol header fields	Type of message (request/response), physical and network addresses of source and destination
Variables	-

4.4.4 322 - TCP Timeout of Connection Establishment

Description: During an attempt of connection establishment it can happen that the server doesn't answer after beginning a connection establishment request. The client has to handle this sutuation and set a timeout to exit if the reply to its request is not received.

Relevant layers: Transport layer

Transport layer.

Protocol header fields	
Variables	TCP states

4.5 Implementation priority #3 (Postponed networking scenarios)

4.5.1 351 - TCP Connection reset (RST)

Description: In general, a reset is sent by TCP whenever a segment arrives that doesn't appear correct for the referenced connection. This can happen in the following three situations:
* Connection Request to Nonexistent Port
* Aborting a connection
* Detecting Half Open Connections

Relevant layers: Transport layer

Transport layer.

Protocol header fields	
Variables	TCP states

4.5.2 412 - HTTP: Retrieval with multiple connections

Description: An HTML page with a couple of images is downloaded from the server. The data is downloaded using several connections, one for each file.

Relevant layers: Application layer, Transport layer

Application layer.

Protocol header fields	**Request** = Request line + general header + request header + entity header + message body **Response** = Status line + general header + entity header + message body
Variables	-

Transport layer.

Protocol header fields	-
Variables	-

4.5.3 421 - DNS request for A record

Description: Some application wants to find out the IP address for some domain name. It sends the address to a DNS server that then replies with the correct address record.

Relevant layers: Application layer

Application layer.

Protocol header fields	Identification, flags, number of questions, number of answer RRs, number of authority RRs, number of additional RRs.
Variables	Header fields: questions and answers

5 Use cases

In this section the people using the software and their main goals are presented.

In the DaCoPAn project there are some different actors that the final product is meant for. Each of this actors have different interests in using the system and all the detail

In the next section of the document use cases capture who (actor) does what (action) with the system, for what purpose, without dealing with system internals. A complete set of use cases specifies all the different ways to use the system, and therefore defines all behaviour required of the system. In these use cases some variations are introduced, they are special situations that can happen and a reaction from the system is needed. Sometimes the reaction of the system is out of the scope of the project and this variations are marked as optional to prioritize them for their development.

5.1 Interest groups

People who will be using the DaCoPAn product can be divided in three groups: teachers, students and researchers.

The teachers' main interest is to use the software as a helper tool in teaching network protocols. They might want to use the product in a class room situation as a substitute for presentation slides or to give exercises to the students to solve.

The students will be using the software to learn how the network protocols function. They will be able to download the software along with intermediate files from a web site so that they can watch the animations on a computer either at home or in a classroom.

Researchers might be interested in using the tool to visualize the behavior of real networks in some interesting cases. They are not interested in the same simple cases as the other interest groups but instead have their own tcpdump file pairs that they want to analyze.

5.2 Use case definitions

There are three different types of actors: teacher, student, researcher. In each use case the primary actor is emphasized. The identification of the use cases is mainly based on discussion with the customer.

5.2.1 Use case: Produce a protocol events file

Actors: *Teacher*, researcher

Description: The user creates a protocol events file (intermediate file) using the analyzer tool. He/she gives two packet traces along with other additional information to the analyzer which then combines the data and produces a protocol events file.

Pre-conditions:

- User should gather a pair of packet traces files to be passed as input to the analyzer.

Steps:

1. The user invokes the analyzer program from the command line (or any another interface) passing the filenames of these packet trace files and some other mandatory and/or optional parameters (as IP identifiers and ports of the end hosts...).
2. The analyzer validates the syntax of the packet traces files.
3. The analyzer merges the packet trace files using the data that belong to connections specified by parameters.
4. The analyzer produces and output file with all the information acquired from the packet traces. The format of this file is suitable to be played in the animator.

Variations:

#1: The analyzer may be able to detect errors in the parameters passed as input and report about it to the user.

#2: The analyzer may detect syntax errors in the packet trace files passed as input and return a detailed report to the user about the errors found in the files.

5.2.2 Use case: Load an animation file in the animator

Actors: *Student*, researcher, teacher

Description: The actor provides the animator with a file which has data in a specified file format. The file format can be Protocol events file or Scenario file.

Pre-conditions:

- The actor should get the animation file from some source. In general, a student should get it from a teacher, and researcher and teacher should know how to get it from the analyzer.

Steps:

1. The actor selects the file to be loaded.
2. The animator might succeed in loading the file or not. This data may contain scenario information.
3. The result of the loading should be shown to the actor within a reasonable delay (some seconds).

Variations:

#1: The loading could be erroneous and the protocol events file is rejected by the animator.

5.2.3 Use case: Play an animation

Actors: *Student*, teacher, researcher

Description: After loading an animation, the user watches it by pressing the play button.

Pre-conditions:

- The user has loaded an animation file into the animator.

Steps:

1. User presses play.
2. The animation starts running on the screen. In the animation information about the data transfer is shown.
3. The animation ends when all the data has been shown or the user presses the stop button.

Variations:

#1: The user can pause and then continue the animation while watching it.

#2: The user can interrupt watching an animation by pressing the stop button.

5.2.4 Use case: Step forward in an animation

Actors: *Student*, teacher, researcher

Description: The user selects how many steps he/she wants to step forward. The animation is then moved to that point and continued from there in stop/scroll mode.

Pre-conditions:

- The user has loaded the animation file into the animator.
- The animator is in stop/scroll mode

Steps:

1. The user selects amount of steps to move in animation.
2. The system then moves the current position in the animation to that location and pauses there. It does not animate the skipped frames but draws their results.
3. When the user has inspected the information at that animation location he/she continues the animation by playing it, stepping forward again, or he/she can also stop the animation there.

Variations:

#1: The user can also step backwards in an animation.

5.2.5 Use case: Add breakpoints to an animation (optional)

Actors: *Teacher*, student, researcher

Description: The user has found an interesting spot in the animation where he/she wants to pause the animation for example when using the animation as a lecture presentation. He/she then marks that point in the animation as a breakpoint.

Pre-conditions:

- The user has loaded the animation file into the animator.

Steps:

1. The user scrolls the animation to the place where he/she wants to add the breakpoint.
2. The user presses the "add breakpoint" button.
3. The system adds a breakpoint to that point in the current layer/animation type of the animation.

Variations:

#1: The user can also remove breakpoints from an animation.

5.2.6 Use case: Add comments to an animation (optional)

Actors: *Teacher*, researcher, student

Description: In some interesting point in the animation the user adds a comment to explain the behavior of the viewed protocol.

Pre-conditions:

- The user has loaded the animation file into the animator.

Steps:

1. The user scrolls to the place where he/she wants to add the note.
2. The user uses some means provided by the animator to add the note to the animation.
3. The note is added to the current view at the selected point.

Variations:

#1: User edits comments in an animation.

#2: User removes comments from an animation.

5.2.7 Use case: Change settings of an animation

Actors: , Teacher, student, researcher

Description: The actor can select levels of animation. He/She selects the set of variables for animation specially concerning different available header information or available protocol variables.
It should be possible for the actor to change the set of variables and some views of the animation even during the animation process as well. This set of variables contain protocol information in the different layers of the animation. If the animation takes place in several layers at a time, the actor should be able to select in which layer he wants to visualize the information.

Pre-conditions:

- The actor has loaded an animation.
- The file used for the animation may contain default visualization settings.

Steps:

1. If the file loaded for the animation contain predefined settings for the animation those values are analyzed and presented in the animation.
2. The user tunes the animation through an interface (selecting options through clicking on them for instance, using buttons, checkboxes,...). The main things to change here are the animations to be presented, layers to be shown and protocol headers fields and variables for each layer.
3. The animator changes the visualization according to the user choices and to the network data available. The program keeps this information and it can be saved later.

Variations:

#1: If there are not predefined settings in the animation file a default view for it is presented.

#2: The actor can configure the animation at any time. (Optional)

#3: Some of the values selected by the actor are not avilable. The system warns about it and do not display that infotmation.

5.2.8 Use case: Save tunable settings for the animator

Actors: *Teacher*, researcher, student

Description: After using the software and deciding a custom set of tunable options, the user saves this configuration for future sessions.

Pre-conditions:

- The actor has selected and changed the default or previous values of configuration for the animator.

Steps:

1. The user selects the option for saving the current configuration.
2. The animator gathers all the information about the changing values in configuration and selects which can be saved for the next session (maybe some options are not available to the user before loading the files and the animator should be aware of this)
3. The animator saves this information into the protocol events file, into the section reserved for the scenario values.

Variations:

#2: It is possible that saving the current configuration is impossible. An error message could be shown to the user.

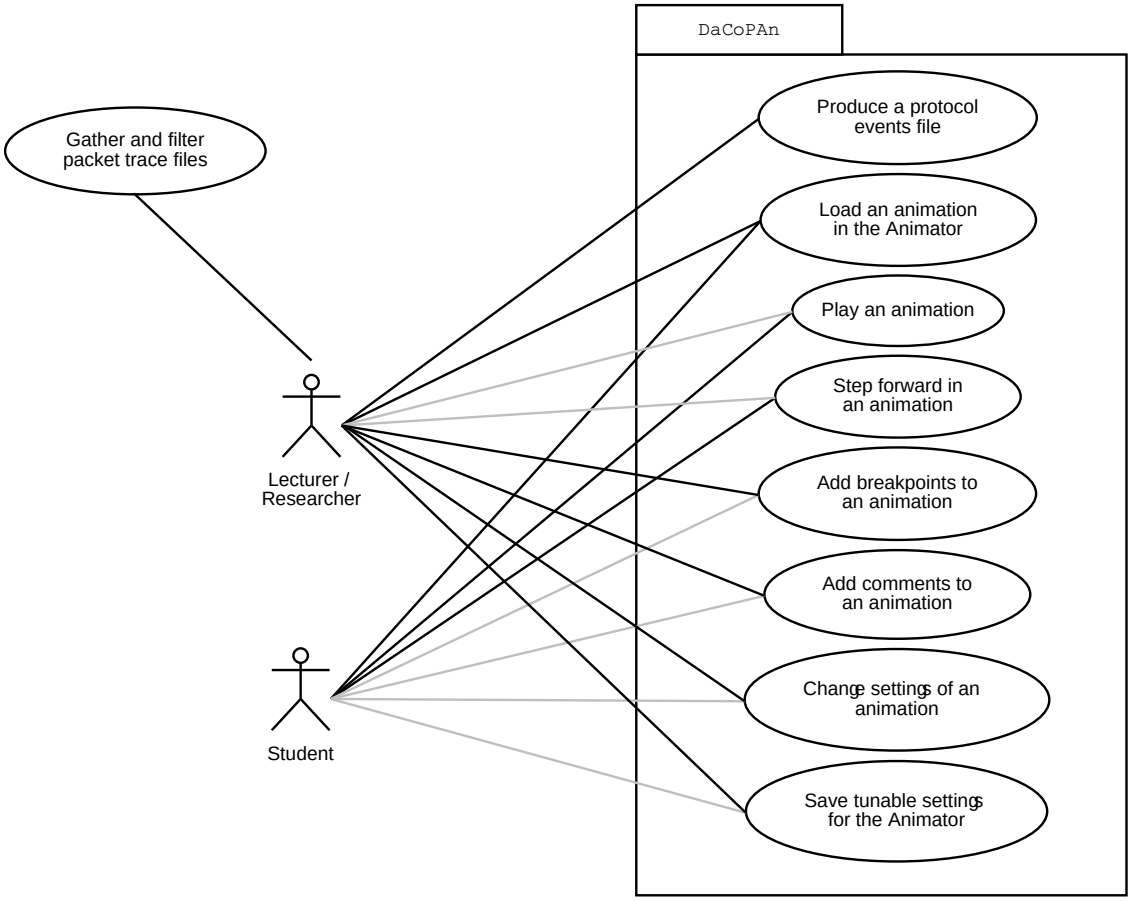

Figure 2: The use cases

Summary of the use case model

The high-level use cases model of the system is shown in Figure 2.

6 System requirements: Analyzer

Expanded use cases describe the functional structure of the user requirements in more detail. They define a typical course of events in the analyzer. There is one main use case and four main actions performed by the analyzer: reading tcpdump log files, mapping messages, calculating events and writing this information to a file in some format. See Figure 3.

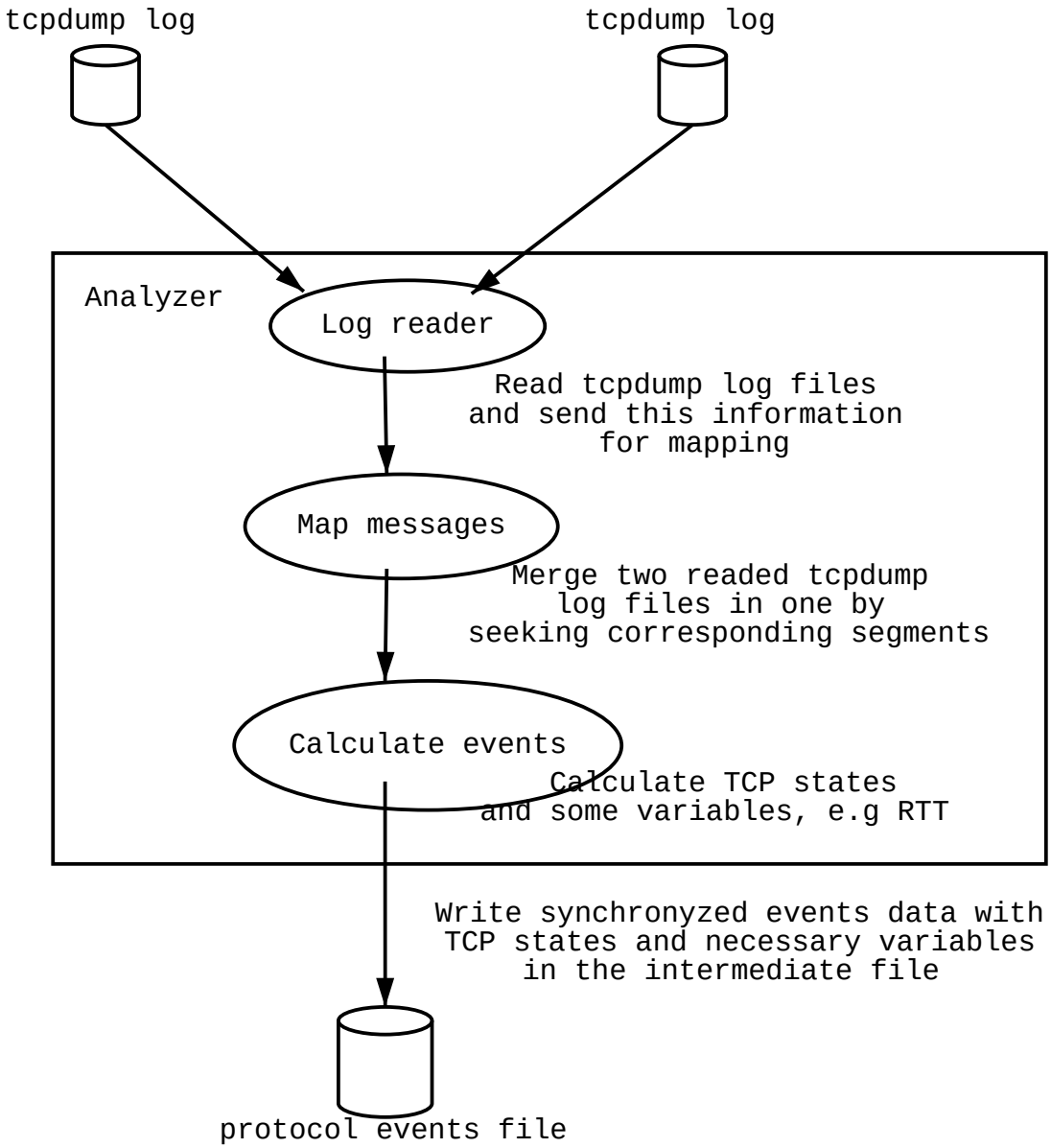

Figure 3: Functions of the analyzer

6.1 Produce protocol events file

Use case	Produce protocol events file
Actors	Teacher, researcher
Objective	Create protocol events file using two tcpdump log files
Description	A user can create a protocol events file (sometimes referred to as an intermediate file). The analyzer combines two packet trace files (tcpdump logs) and produces a protocol events file. The analyzer also requires some additional information (IP identifiers of end hosts, time differences, etc.) for producing a protocol events file and for calculating certain events.
Type	Main
Typical course of events	
User actions	System response
1. User selects two tcpdump files and gives necessary parameters to the analyzer (IP addresses of end hosts, etc)	
	2. Reads tcpdump files
	3. Maps messages
	4. Calculates events
	5. Writes all necessary information in the protocol events file
6. User gets protocol events file	

6.2 Log Reader

Log Reader is the Analyzer component, that is responsible for reading binary tcpdump log files and filling internal memory data structures with the information.

- **Input data**
  Log Reader input data consists of following items:
  - Two packet trace files.
    The packet trace file corresponds to some host. The packet trace file is obtained using tcpdump tool running on the corresponding host. The packet trace file is in binary format.
  - Two IP addresses.
    Each IP address corresponds to one packet trace file and is a valid IP address of the corresponding host.

- **Output data**
  Log Reader output data consists of internal memory data structures. The data structures specification will be developed during design phase.

- **Functionality**
  This section contains scenarios, which describe the functionality of Log Reader:

Description	All mandatory arguments are specified by the user. Each packet trace file exists, is readable by user, and not corrupted. This is the basic case.
Action	Read the packet trace files and fill the internal memory data structures.
Description	Mandatory parameter is missing. Packet trace file does not exist, is not readable by the user, or corrupted.
Action	Show a message with error description, possibly short usage information, and terminate the Analyzer.

6.3 Message mapping

Message mapping module in the Analyzer merges data from two tcpdump files

- **Input data** Tcpdump files as some data structures from the log reader output, IP addresses of end hosts.

- **Output data** Information in chronological order about packets where each one packet corresponds to other

- **Functionality**

Description	All input information is right
Action	normal finding of corresponding packets(segments) according to IP addresses of end hosts
Description	IP addresses of end hosts can’t be found in tcpdump data
Action	an error will be returned
Description	any correspondence of packets can’t be found
Action	an error will be returned
Description	for one packet corresponding packet can’t be found
Action	this packet will be written in protocol exchange sequence
Description	chronological order can’t be presented
Action	an error will be returned

6.4 Events calculator

Events calculator module in the Analyzer calculates possible variables and TCP states. First version of Analyzer will support these calculation on the minimal level, but intermediate file should be designed to store these data.

- **Input data** Some necessary data for calculating TCP states (mapping data) and variables; this data is based on network scenarios.
- **Output data** List of calculating possible variables and TCP states.
- **Functionality**

Description	All necessary preliminary data for calculating is correct
Action	Normal calculation of variables and TCP states
Description	All required data isn't passed to the module
Action	An error will be returned
Description	Necessary data for calculations is incorrect
Action	An error will be returned

7 System requirements: Animator

The purpose of this section is to outline some system requirements for the animation types. To be able to do this meaningfully, it is necessary to introduce the animation types conceptually. The rationale for including the animation types in the Requirements Specifications is that they have been conceived to be able to effectively and meaningfully visualize the mechanisms involved in the Networking Scenarios.

The main animation types are the Message Sequence Chart (MSC) and the Encapsulation. There are also two auxiliary animation types: the Unit Flow Orchestation and the Transfer Progress Indicator. They are meant to be used together and simultaneously with the MSC, never alone or with the Encapsulation.

7.1 Message Sequence Chart animation

7.1.1 Overview

Figure 4 is a schematic diagram of how the Message Sequence Chart may look. The bracket symbols and associated text are just explanations for the figure - they will not appear in the software.

The Message Sequence Chart is a traditional chart format used in networking protocol literature. The idea is to present packet interchange between two hosts (marked as A and B in the figure). Vertical dimension in the figure is the time axis, with the positive

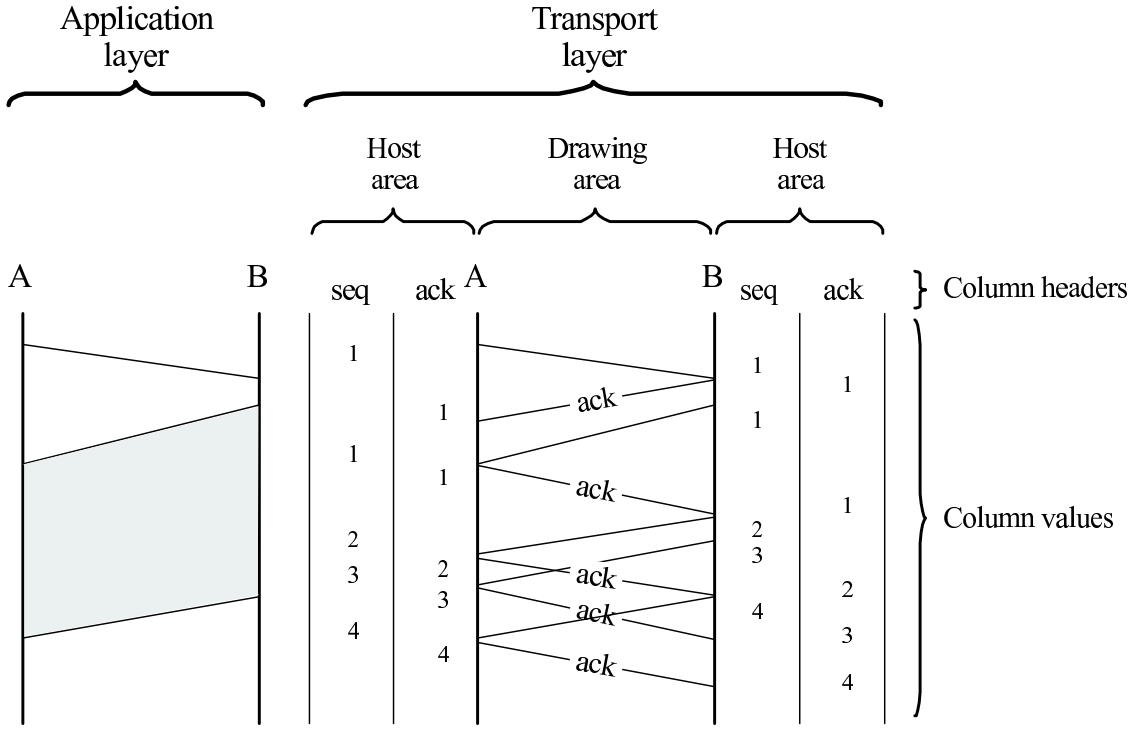

Figure 4: Schematic diagram of Message Sequence Chart animation display. Columns are shown in left-to-right order.

direction being downwards. The figure illustrates also how symbolic information can be plotted inside the drawing areas and on the sides of the drawing areas, in the so-called host areas. The figure also illustrates how exchanges on both the application layer and the transport layer can be orchestrated side-by-side.

7.1.2 Features of the Message Sequence Chart (MSC) animation

Mandatory

- MSC needs to be able to present packet exchanges in the drawing area.
- The length (measured in time) may be greater than can be accommodated on screen at once, so some kind of scrolling mechanism is required.
- The two operational modes (play/now & stop/scroll) must be implemented.
- MSC needs to be able to visualize a packet loss.
- At least one protocol header field may be selected to be shown symbolically inside the drawing area. (see section Presenting numerical information about events)
- At least four protocol header fields or host variables may be selected to be shown symbolically inside the host areas. (see section Presenting numerical information about events)

Optional

- More than one stack layer may be orchestrated side-by-side on screen with the same time scale. The Animator does not need to be able to show simultaneously multiple MSC's with different time scales.
- TPI animation type may be orchestrated with the MSC, simultaneously on screen with the same time scale.
- UFO animation type may be orchestrated with the MSC, simultaneously on screen with the same time scale.
- "Show data below now line" setting may be implemented (see below).
- The user may be able to augment the presentation with breakpoints and notes.
- The user may be able to view detailed (all available) information about a packet by clicking on a packet, or a line representing the transfer of a packet.

7.1.3 Application layer on MSC

Events including just one segment on transport layer (e.g. HTTP request that fits in one TCP segment) are visualized with a line.

Events including many segments on transport layer (e.g. HTTP reply that takes >1 TCP segments) are visualized with a coloured area. The area represents an abstraction of what happens on transport layer.

The left and right sides of the area (for visualizing application layer event) are vertical and go exactly along the corresponding sides of the MSC drawing area. The upper end of the vertical line corresponds to the first data segment (on transport layer) of the event being sent/received. Likewise, the lower end corresponds to the last data segment for the event. ACK-only segments are ignored for this presentation (because including them would make the visual presentation logically contradictory - reply could begin before request had ended). Thus, the left and right sides of the area mark the duration in time of the event from the viewpoint of both of the hosts. The upper and lower sides of the area are straight lines connecting the lower ends and upper ends of the vertical lines. The upper and lower sides of the area also correspond exactly to the first and last segment of data that belong to the event.

The area described above is an abstraction of what goes on on the lower level. The area geometrically covers the data transfer that goes on in transport layer, but doesn't show any details, that are not relevant for the application. From the application's point of view, the event is just a duration in time. That's what the area shows.

Optionally, there may be vertical lines shown outside the actual drawing area (inside "host" areas) to make it more clear that the duration of the event is related to the states of the host.

7.1.4 Operating modes for the MSC animation

There are main operating modes for the MSC animation, the "play/now" mode and the "stop/scroll" mode. (The names reflect analysis rationale for the modes. Different names will be presented to the end user.)

"Play/now" mode

In this mode the time passes automatically. Conceptually, there is a "now line" that represents "now" time for the animation. This now line may (probably will be, but is not mandatorily required) be visualised as a horizontal line in the visualization. The user is not allowed to scroll the view, scrolling is controlled automatically. Optionally, there may be an animation of packets moving between the host. These may be simple objects moving along the now line, or a larger and more detailed presentation on a separate panel (this presentation format is described in a separate description).

"Stop/scroll" mode

In this mode, the time stands still, unless the user takes some actions. The user can use controls to step the time forward or back. It is possible to scroll the presentation vertically to show all the data that has been plotted up to the "now" moment in time. Scrolling is a separate action from controlling the flow of time. Controlling time implies changes in the state of animation and the hosts, scrolling just manipulates the visual display.

7.1.5 "Show data below now line" setting

Optionally, there may be a setting in the program that allows the user to control if "future" events should be drawn or not. The default setting is "off", which is the only available mode, if the setting is not implemented.

- When "on": All the animation data for the scenario is always drawn. If there is a now line, the now line just glides over data that has already been drawn. This mode would be useful to allow the user to explore the scenarios. This is NOT preferable for the initial (first time for the user) presentation of a scenario.
- When "off": Only animation data up to "now" moment is drawn. When time passes (either is "play" mode or by stepping), more data is drawn. This mode is preferable for initial presentation of the scenarios, as this mode directs the user’s attention to how the actions evolve over time. The user may also be encouraged to construct his/her own predictions of what will happen in "future", based on information that is available "now". This would be very useful for education.

7.1.6 Presenting numerical information about events

Presenting numerical information means showing numbers (1, 2, 3, ...) on the screen, as opposed to graphical presentation in the form of lines and other geometrical shapes.

On the MSC, a very limited amount of information about the packets can be presented in the drawing area. This is due to limited space on the screen and also the necessity to avoid clutter on screen. See figure 4 for what exactly is the drawing area and what are the host areas. The figure also presents an example of the presenting numerical information in MSC.

Numerical information can be shown either in the drawing area (along the lines representing packet exchanges) or outside of the drawing area, in the so-called host areas. The conceptual distinction is that preferably information about the packets (protocol header fields) are presenting in the drawing are and information about the hosts (internal variables) are presented in the host areas. However, to avoid clutter, it will often be necessary to present also the header fields away from the drawing area.

There is no reason to technically limit the choice of fields and variables. The user should be able to select ANY field or variable (available in the protocol events log) to be shown in either of the areas. Because of screen space limitations it will be necessary to limit the NUMBER of fields and variables shown at once. It is suggested that zero to two fields could be selected for the drawing area and zero to four fields for the host areas. (The limit of four is an arbitrary choice, but seems reasonable because it allows presentation of any mechanism in the networking scenarios, and it is not too large to necessitate the use of horizontal scrolling mechanisms on the screen.)

From an educational point of view, the MEANINGFUL selection of fields and variables is dependent on the particular scenario that is being taught. Most often the Lecturer user would make the selection of fields and variables while preparing the scenario files for the students. However, the students are not restricted from selecting any available fields and variables for presentation, and thus exploring different aspects of the network exchanges.

In the host areas, there can be column-shaped areas reserved for plotting numerical data. One column is reserved for values of one variable (or header field, if necessary). The exact selection of columns is scenario-dependent and needs to be fully configurable. The ordering of columns can either be "left-to-right" or "centric". In "left-to-right" option the columns are in the same left-to-right order on both hosts. In "centric" option the same column is closest to drawing are on both sides.

Numerical values will be drawn in the columns next to events. To avoid overlap, the information should be plotted just above the time-coordinate of received events and just below sent event. This way, if receiving and sending happen very close to each other, the numerical values will not overlap on screen.

7.1.7 Breakpoints and notes

Breakpoints

A breakpoint marks a specific point in time. When MSC animation is run in "play/now" mode and a breakpoint is encountered, MSC switches to "stop/scroll" mode. Thus the function of breakpoints is that the presentation is stopped for user input. To continue presentation, the user can select a Play button again. The breakpoints have no effect when

the animation is run in "stop/scroll" mode.

Notes

The user can augment a scenario with textual notes. Notes are input and saved as ASCII (optionally, Unicode) text. Each note is mapped to a point in time and a host (either A or B). The notes may be presented in host areas, in a separate column. The column width may be configurable. Optionally, the font size for notes may be configurable (either per note or per scenario). There may also be an alternative for presenting notes.

7.2 Encapsulation animation

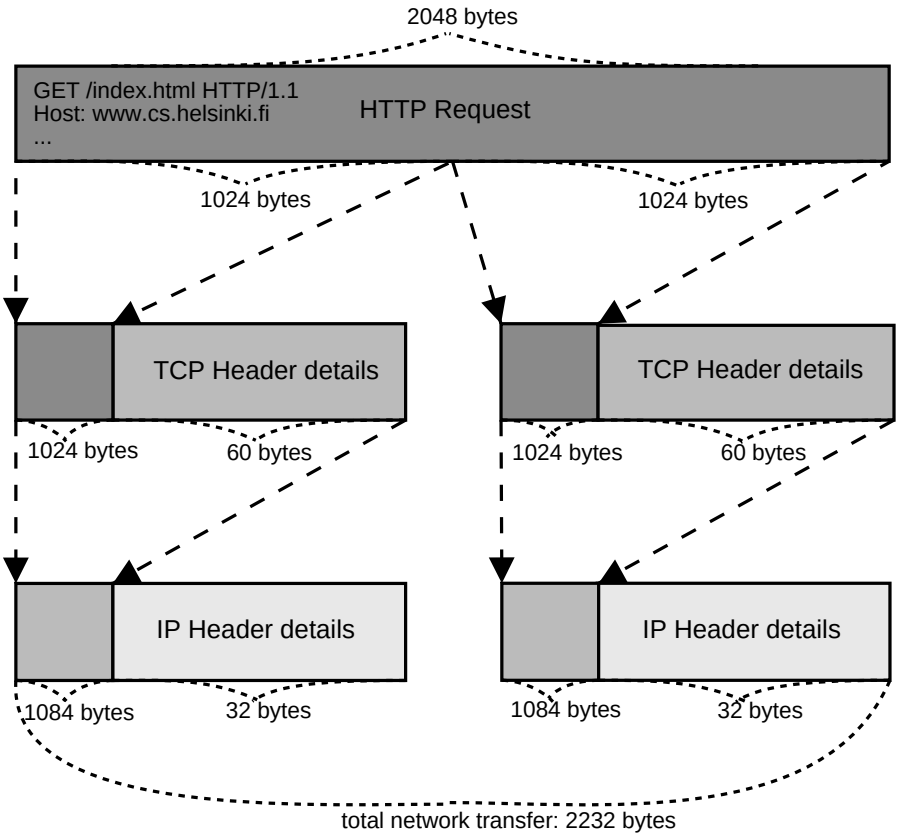

Figure 5: A sketch of the Encapsulation animation display.

The encapsulation and de-encapsulation of data in the hosts is inherently an "instantaneous" event from the point of view of the scenario in which it is happening. This is why the ENC animation type should be considered something that is displayed in a timeless "paused" or "stopped" state of the animator module. It should be possible to switch between the main MSC animation and the ENC modes in a way that the enc mode "stops" the MSC animation if it is not already stopped, and displays a particular encapsulation-deencapsulation pair.

There are two operating modes for the ENC animation:

- Encapsulation
- De-encapsulation (identical, in reverse)

7.2.1 Presenting numerical information about packets/encapsulation

The ENC panel is expected to function so that it takes up roughly half of the screen, vertically split from top to bottom, while the other half would be available for showing information about the packets, or the current state of the MSC, possibly.

The encapsulation would be animated so that it begins with a single application level unit, and proceeds to split it, if this is the case. After this, it is encapsulated in the transport layer as payload, and the header information is written into the transport unit header. (TCP or UDP header). Then this is encapsulated in a single IP datagram. (Splitting TCP segments is not yet supported by any scenario, but it could be easily visualized by just extending the same principle already in use to the network-transport layer.)

The deencapsulation should be visualized by playing the encapsulation in reverse, so that the animatino begins with one or more encapsulated network level units, and proceeds to "piece together" the higher level units from the payloads. This mode should replace the encapsulation, only one needs to be shown at a time.

Sizes of units should be shown with braces and real KByte size integers, displaying how the layer is shrunk to fit into the lower layer payload in the animation. The payload should be smaller than the header for each layer. Also possibly certain header information can be displayed in the header area fr each relevant unit, but only on it’s appropriate layer, on lower layers the header is shown, along with the payload, as being nothing more than payload for that lower layer. See associated image.

7.2.2 Mapping transfer units to other units in higher layers

In all scenarios it is not possible to reasonably map all lower level protocol entities to upper level entities. If it’s not possible to know what the application layer entity is, the animation only shows one app layer entity and shows the encapsulation into the lower layers. show only 2 TCP layer packets. In the case where an upper level entity (such as a large HTTP response) is present, the ENC is only able to show the encapslation of certain, predefined transport level entities, for example the first TCP packets. The ENC animation would thus be specific to a certain higher level entity, and although each and every TCP packet constituting the response will be present in the MSC animation, only a certain subset needs to be displayed for the user to grasp the concept at hand.

7.2.3 Mapping breakpoints and notes to encapsulation animation

If breakpoints and notes will be implemented for the encapsulation animation, they need to be mapped to specific phases of the animation. Within a scenario the breakpoints and notes will first need to be mapped to a particular protocol transfer unit (e.g. IP packet or a

TCP segment). Within the encapsulation animation the breakpoint or note will need to be mapped to mode of presentation (either encapsulation or de-encapsulation) and a specific layer.

7.3 Unit Flow Orchestration animation

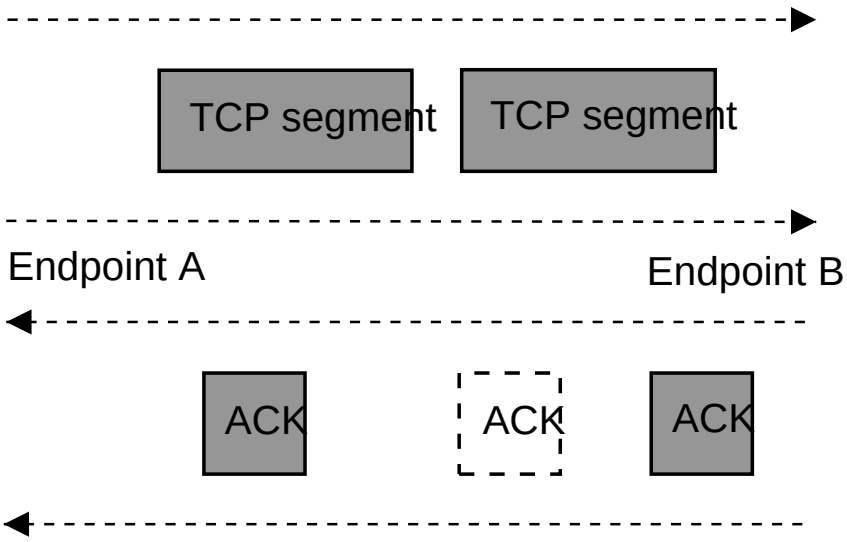

Figure 6: A sketch of the Unit Flow Orchestration animation display.

The Unit Flow Orchestration (UFO) is an animation type that is meant to be displayed in conjuction with the MSC animation. In the MSC, each unit transferred on a certain layer (application, transport or network) is simply visualized as an arrow. In the UFO, these units are visualized as "unit" objects moving across ’channels’ (one channel to each host, each channel is "one-way"). The sizes of the unit can be visualized by drawing the objects in their proportional sizes, or optionally in an adjusted logarithmic scale, to better allow for objects of vastly different size.

The main features of the UFO would be that it follows, at all times possible, the movement of the now-line. What the ufo shows in a paused state would be the same as the MSC, however ina stopped state (where no now line is relevant) the behaviour of the UFO also not relevant. The UFO panel can be used to display general host configuration data, or something else relevant in this state.

Note: This animation type should be considered optional at this point

7.4 Transfer Progress Indicator animation

Transfer Progress Indicator (TPI) is an animation type whose purpose is to visualize the following things:

- total amount of data (bytes) sent between two endpoints

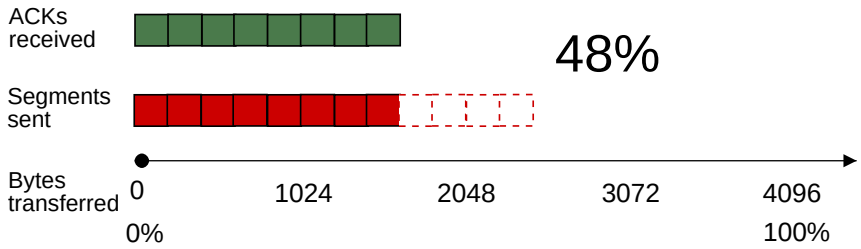

Figure 7: A sketch of the Transfer Progress Indicator animation display.

- progress (bytes, percentage) of the data transfer
- practical meaning of variables like cwnd

TPI is meant to be viewed in conjuction with the MSC animation. TPI is specific to the TCP protocol. TPI enhances conceptual understanding of the data transferred through the TCP connection. TPI shows the idea that the TCP segments together form a large stream of data. TPI can also be used to visually present the meaning of congestion window (cwnd) and possibly ssthresh or other related variables. As the bars in the indicator show the data that has been sent and the data that has been acknowledged, the difference between the bars is the amount of data that has been sent but not acknowledged. The size of this difference corresponds to the value of congestion window (cwnd).

Note: This animation type should be considered optional at this point

7.5 Scenario file format

The scenario file format needs to be able to express a sequence of more than one presentations. (In other words the same protocol exchange, viewed with different settings, with different animation types, or different variables.) Attributes for one presentation are the animation type, precise configuration (including free selection of variables) for animation type, starting time and ending time of animation (in the same scale as animation data), possible breakpoints and educational note texts.

8 Protocol events file

The protocol events file is the normal output of the Analyzer. It contains network traffic data obtained from two packet trace files and processed by the Analyzer. The protocol events file is read by the Animator.

The protocol events file should contain all the network traffic data, which is needed by the Animator to perform the animation. The restriction is that the information, which can't be extracted from the packet trace files, is not presented in the protocol events file. The protocol events file format should allow user to insert additional information, which can't be extracted by the Analyzer, but is used by the Animator, manually.

The basic format of the protocol events file should support three network layers: network layer, transport layer, and application layer. The basic format should support the following protocols on the corresponding layers:

- **network layer:** IP;
- **transport layer:** TCP, UDP;
- **application layer:** HTTP, DNS.

The protocol events file format should contain information on packet encapsulation.

The protocol events file format should be extensible, human-readable and editable. Current suggestion is to use XML for the protocol events file format.

9 High-level architecture

The high-level architecture model of the DaCoPAn system is presented in Figure 8. The figure presents the architecture from a very high level and is still open for changes in the design phase. It is an approximation on how the software might be created when real architectural design is started. The main purpose of this section (and the diagram) is to show concretely where the different requirements presented in this document fit in the actual product.

The diagram shows interaction between different components of the system and its subsystems, interaction between users and the system, and the input and output data formats used by the software.

The DaCoPAn software consists of two large subsystems, the Analyzer and the Animator, which are connected by a file format called *Protocol events file format*. The Animator also uses another file format for adding user setup and comments to the data produced by the analyzer. This file format is called the *Scenario file format*.

In the figure you can also see which different groups of users are using which components of the system. The analyzer is supposed to be only used by lecturers and researchers to create Protocol events files. Then this file can be given as input to the animator. The animator can be used by any user group, including the students, to observe the behaviour of protocols in a network.

9.1 Analyzer

High level model of subsystems of the analyzer will consist of 3 general components: Log reader, Message mapping and Events calculator. It is described in more detail in the subsection "Analyzer" of the section "System requirements".

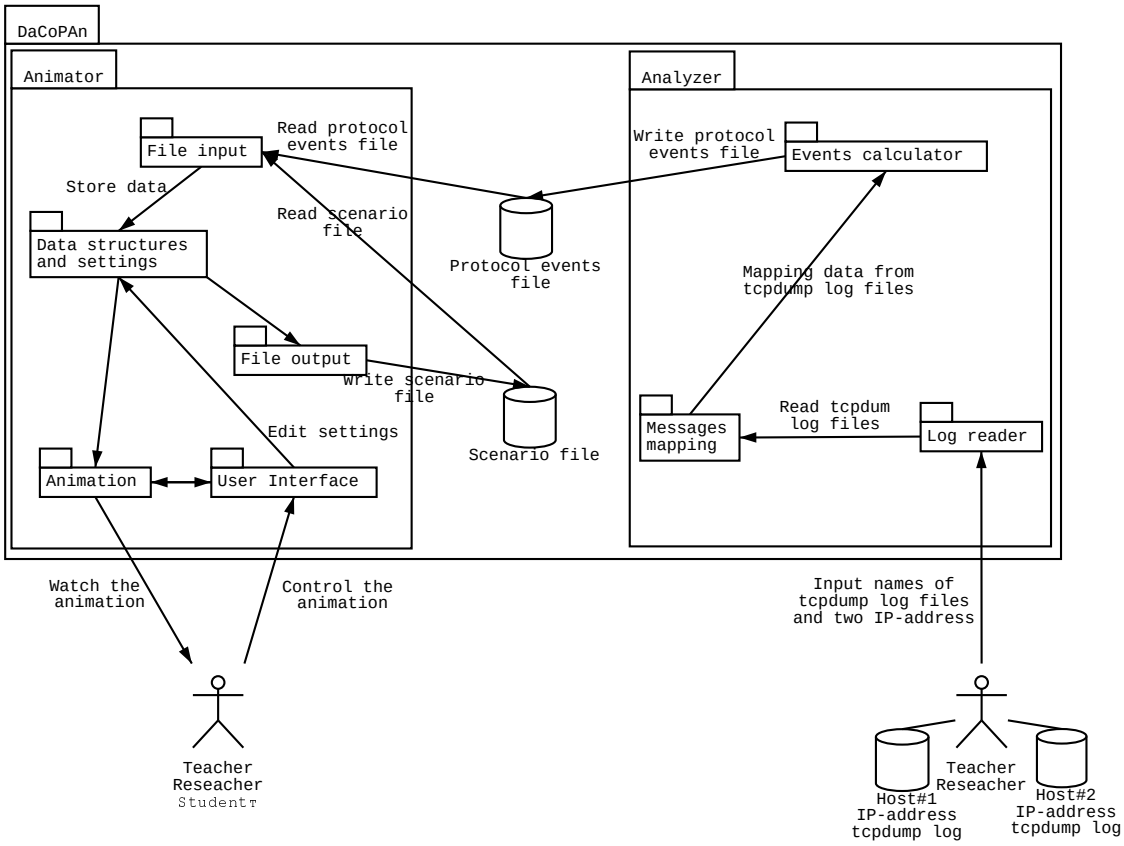

Figure 8: High-level architecture model of the DaCoPAn product

9.2 Animator

The actual implementation and design of the animator will be much more complex than what is seen in Figure 8, but by looking at that figure you can gain a basic understanding of the different parts of the work that the animator has to be able to do.

In addition to the diagram there are data structures for storing the data loaded from the animation file (either *protocol events file* or *scenario file*) and setup data from the user or the scenario file. Those are used by different components of the product.

9.2.1 File input and output

The animator subsystem has to be able to load networking data from protocol events files created by the analyzer as this file format is the bridge between these two modules of the DaCoPAn software. The animator can also use scenario files that have the same networking data as the protocol events files, but also include a section for adding animation setup (e.g. breakpoints and notes).

The data is loaded into data structures which in turn have some kind of class hierarchy of their own.

Another part of the software beside loading files is saving files. The animator will not be able to create and output network data - that is only created in the analyzer - but it will output animation setup data. That data will be appended to the files created by the analyzer so that setup related to some specific scenario can easily be later retrieved by just loading the file. This configuration data resides as a separate section in the file.

9.2.2 Animation

The animation module is large and actually encapsulates a lot of action in this diagram. It includes the common parts for showing animations (e.g. communication with the user interface) and all the different animation types supported by the software. More information about different animation types is presented in section 6.3 of this document.

9.2.3 User Interface

The user interface is the means for the user to communicate with the animator. It is therefor important that it is intuitive and easy to use in a powerful way. The user will be able to use it for controlling the animation viewing by changing the view modes and stepping in the animation. He/she can also use the user interface to edit comments, breakpoints, and other setup information presented in the earlier sections of this document.

References

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