TCPconal: A Prototype of Specification Language for TCP Connections Data Processing

Dmitri G. Korzoun, Dr. Iouri A. Bogoiavlenski

Department of Computer Science, University of Petrozavodsk

Lenin St., 33, Petrozavodsk, Republic of Karelia, 185640, Russia

E-mail: {ybgv, dkorzun}@mainpgu.karelia.ru

Abstract

It is substantiated in the article necessity of formal language development for specification of TCP connections data processing. We introduce description of the first realized prototype of TCPconal language. The prototype provides extracting of specified substreams from a total TCP connections stream. These substreams correspond to given combinations of hosts, group of hosts and subnets. Frequency analysis of hosts activity and TCP/IP application protocols usage is also supported. Besides, TCPconal provides flexible specification of reports on processing results. The prototype of the language is implemented using `flex` and `bison` tools. The implementation is a subsystem of TCP traffic analyzer TCPconan. An example of TCPconal program and examples of the reports are given as well.

1 Introduction

The network management model of ISO [1] consists of five conceptual areas:

The research was partially supported by grant N 96–42–36 of Ministry of Secondary and Professional Education of Russian Federation.

- Performance Management;
- Configuration Management;
- Accounting Management;
- Fault Management;
- Security Management.

The effective solution of any task arising in these areas is possible only on the base of results of monitoring and analysis of network traffic data.

Selection of the protocol layer to be monitored is one of the key problems, because it defines elementary units of traffic data to be fixed.

Modern network technologies provide very high data transfer speed, that leads to huge volume of traffic data on some layers. For instance, day traffic of Ethernet segment containing 200 workstations may reach 18 Mpackets, 6 Gbytes on Ethernet packets (frames) layer [2]. Storage and analytical processing of such volumes of data become highly expensive and difficult.

The more effective is approach, where protocol connections' data are chosen as elementary units. This allows to store data on dozens Mbytes in compressed form. For example this approach is accepted by *Cisco*, which builds connections monitoring modules into OS IOS (subsystem NetFlow [3]). The subsystem allows to concentrate on Cisco routers primary forming of the connections data.

In modern distributed systems overwhelming majority of connections are produced with TCP. It means that problems of TCP connections data analysis are of uttermost importance for the network management.

It is necessary for the problems solution to process the monitoring data by various ways. The processing methods depend on concrete research purposes. Some examples of them are:

- Investigation of traffic substreams connected with separate hosts, their groups or subnets.
- Frequency analysis of usage of distributed systems various elements, such are for example certain servers or application protocols.
- Investigation of TCP inner algorithms and/or of TCP implementations functional correctness.

- Investigation of behavior of connection–oriented protocols like MDCS for wireless links [4].
- Investigation of properties of TCP connections structures, which can arise in a network.
- Obtaining of various characteristics, which are necessary for traffic mathematical models and their parameters estimation.

The list may be prolonged far enough.

It is evident, that the methods of the traffic data processing cannot be fixed, because they strongly depend on analysis needs. Hence, it has sense to develop approaches providing flexible management of the processing. We propose to solve the problem by development of specification language, which allows to a user to formulate necessary processing. We have of experience of design this kind of languages for filtering and processing of low level network traffic [5]. In the paper we describe the prototype of new language TCPconal, which is aimed for specification of TCP connections data processing.

The rest of the paper is organized as follows. Section 2 contains review of existing software for TCP traffic data monitoring and analysis. Section 3 is devoted to TCPconal prototype description. Section 4 describes the prototype implementation as a subsystem of TCPconan traffic analyzer. Some results of experimental measurements using TCPconal specifications are presented in section 5. The appendix contains an example of TCPconal program for hosts activity analysis.

2 Software for TCP Traffic Analysis

2.1 General Scheme of a Traffic Analyzer

The general scheme is presented in the Figure 1.

'Data Acquisition' unit captures the traffic of investigated system. The traffic may be as real (native or artificial) so simulated. Well known package `tcpdump` [16] may be considered as an example of this unit.

The 'Initial processing and primary data storage' unit is destined for filtering and primary data processing. For example, the unit might eliminate non TCP packets, extract connections data and store them.

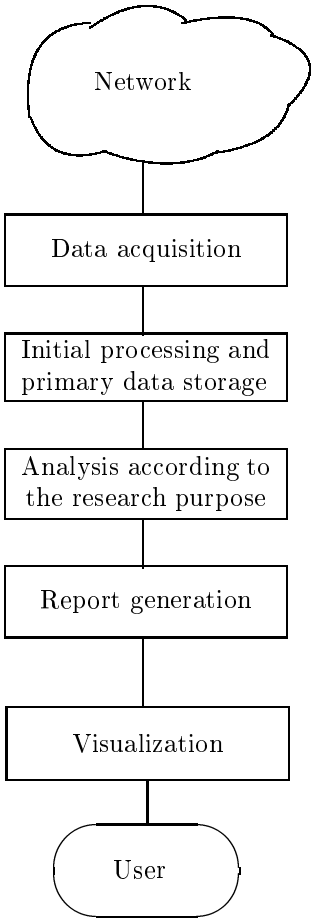

Figure 1. The general scheme of a traffic analyzer

'Analysis' unit performs full processing of traffic data according to the research purposes. As a rule, this unit is most complex and consumes the most part of resources. 'Report generation' unit forms certain types of reports about performed traffic data processing.

The problem of the processing specification is related to two units: 'Analysis in according to the research purpose' and 'Report generation'.

Our language TCPconal allows to specify working procedures of these two units.

2.2 Some Existing Software Tools for TCP Traffic Analysis

The existing software tools may be divided on three following groups:

1. **Active Software.** This kind of tools generates artificial traffic, which serves as a base for measurements. Real networks and links are used for the traffic transfer.

2. **Passive Software.** The tools only measure native traffic existing in a network.

3. **Simulation Software.** The tools simulate behavior of a distributed system. It means as artificial traffic generation so simulation of a network. Real links are not used.

Each of the group has own advantages and drawbacks. Choice of the appropriate tool depends on a purpose of an investigation. Moreover, combinations of the tools of different groups may be useful in some situations. However, the traffic data processing problem should be solved for each of the group listed above.

There exists a number of tools (free and commercial) for TCP traffic investigation. Let us consider some well known packages. Essentially more comprehensive, but also incomplete lists can be found in [6, 7].

TCPreduce. It reduces the `tcpdump` trace file to one ASCII record per a connection, containing its short description [8].

TCPtrace. It reads output dump files in the format of several popular packet capturing program (`tcpdump`, `snoop`, `etherpeek`, and `netm`). For each connection it keeps track of main important set of characteristics. Its output format ranges from *Simple* to *Long* to *Very Detailed* [9].

TCPanaly. It is a tool for automatically analyzing a TCP implementation behavior by inspecting packet traces of the TCP's activity: problems discovering, standard deviations, inefficient actions. More detailed description can be found in [10]. Theoretical background of this tool is stated in [11].

TReno. It was designed to measure the single stream bulk transfer capacity over Internet path. It is an amalgam of two existing algorithms: `traceroute` and an idealized version of the flow control algorithms presented in RENO TCP. It generates and measure artificial traffic on the given path [12].

DBS. It gives performance index with multi–point configuration and also measures changes of throughput. Traffic between elements of the configuration is given by special way and it is artificially generated [13].

Netperf. It is a benchmark that can be used to measure various aspects of networking performance. Its primary focus is on bulk data transfer and request/response performance. It measures most important characteristics of TCP traffic on the given path by its artificial generation. All measurements are performed at the packet level, not at the connection one [14].

tcplib. It is a library for network simulation. It is based on a number of empirical models of TCP application protocols behavior [15].

Main disadvantage of the systems like those mentioned above is that all of them does not allow to process data of traffic in a way flexible enough. All calculated characteristics are fixed in a package and user has no opportunity to define explicitly the processing features. However, some of the systems have very simple means for the processing specification. For example `TCPtrace` supports three types of the report generation, `DBS` allows to a user to specify traffic in a network configuration, and `tcplib` allows to construct simulation models by using of the library functions. As a rule, the processing is controlled by command line, which is rather poor mechanism for these purposes.

3 A Prototype of TCPconal Language

3.1 The requirements to the language

1. *Independence of an input data acquisition method.* An input data sources and format should not be strictly fixed. Traffic can be real, it can be simulated, it can have different formats, but all these factors must not hinder processing. The requirement may be provided on the lexical analysis level of input data streams. Besides, the language should provide processing of the input containing data

with different level of details. This requirement allows easily to implement the language in different traffic analysis systems and for different platforms.

2. *Flexible mechanism of processing type selection.* The user should be allowed to choose processing from a wide set of possible types and those and only those ones which are really necessary for him. This allows to eliminate excessive work (computing of unnecessary characteristics and parameters). In the case of TCP it is especially important, because the protocol is rather complex and the TCP connections data actually have an immense number of variants of possible analysis and of corresponding processing.

3. *Orientation to the structural processing.* TCPconal should allow to treat the traffic on the following structural levels:

   - TCP connections level. A connection is considered independently of any other one.
   - TCP connection group level. All connections belonging to a given group are considered as correlated in some sense. For example, it is pointed in [17] that a client using a collection of parallel connections to connect to a server is more aggressive user of the network that one that uses a single TCP connection.
   - Hosts level. An activity of a host is investigated in comparison with other hosts.
   - Host groups level. An activity of the host group is investigated and compared with traffic of other hosts.
   - Application protocols level. Fractions due to the protocols are investigated.

   The requirement is useful for common network management, where the problem is to clarify activity of the resources usage by a given host or by a group of them, or to determine the most and the least active elements of the network.

4. *Selection of output data presentation.* As a rule, result of processing is an intermediate stage of a research. Further it can be stored or visualized for comprehensive analysis. The user should have an

opportunity to select suitable for him kind of presentation, appropriate content, format, and volume of the results. It allows to use TCPconal in a combination with arbitrary software of data analysis and/or visualization.

5. *Extendibility of the language.* Modern network technologies are intensively developed and continuously improved. TCPconal should have ability to support the changes by adding of new features without loss of the compatibility with the previous versions.

One important direction of the language development has its origin in the fact that TCP is rather complex protocol. It uses a number of various algorithms and techniques. This means that there exist large amount of the TCP traffic behavior characteristics, which can be investigated. Thus, it is desirable to extend the language with tools to support investigation of some basic problems of TCP implementations testing, its current performance and its further developments. These problems examples are listed below.

- *Congestion Establishment and Termination.* Correct, economical and effective way to establish and to terminate TCP connection.
- *Congestion Avoidance and Control.* TCP tries to determine current network workload and using that it synchronizes intensity of TCP packet injection with network throughput.
- *Acknowledgment Strategy.* TCP uses acknowledgment mechanism in data transferring to ensure guaranteed packet delivery. The problem is to design and to implement this algorithm with minimal network resources usage.
- *Packet Reordering.* It is well known that packets can be delivered in a different order then they had initially. It is one of Internet key characteristics. This phenomenon implicitly influences to behavior of some TCP parts.
- *Packet Loss.* IP does not guarantee packet delivery. So TCP is forced to discover itself all lost packets.

- *Deadlocks.* There are possible some situations, where network is not overload, members of TCP connection have data to exchange, but TCP does not allow to transfer data. It is a result of non efficient performing of some TCP algorithms. The problem is in discovering and elimination of such situations.
- *Slide Window Strategy.* Slide window mechanism is a part of TCP to control and regulate network workload.
- *Segment Sizes.* TCP should know what sizes of TCP segments it would be better to use. Right choice between small and large segments may reduce network workload.
- *Round–trip Time.* TCP estimates RTT and intensively uses the estimation in some algorithms. The good approximation of the value leads to more efficient network resources usage.

3.2 Language Description

Today TCPconal is simple enough language, which allows to formulate three types of data processing, which are the most popular for TCP traffic. The types are connections analysis, hosts activity analysis, and analysis of TCP application protocols usage.

3.2.1 Main principles

It is accepted in the language that to solve a problem of the data processing a user should:

1. *Define a type of processing.*
2. *Select a substream of data to process.*
3. *Specify the processing results to be reported.*

All these actions are described using a set of built-in variables. The type of processing defines the corresponding set. The variables values present various characteristics of the traffic to have processed and the processing results. More detailed description of the current set of the variables is given in section 3.2.4.

A program to specify the traffic processing is defined as a list of statements, separated by semicolon ';'. Each statement belongs to one of the following group according to the language principles:

- Processing type specification
- Substreams selection
- Report specification

The statements may be repeated. If this leads to ambiguity then only the last one is taken into account.

3.2.2 Specification of the Processing Type

Statements of this group allow to a user to set a character of TCP traffic processing.

Analysis type setting.

```
analysis = { TYPE };
```

This statement defines a type of processing. In the current implementation `TYPE` can be one of the following terminals:

```
connections

hosts

protocols
```

They correspond to setting on connection analysis, hosts activity analysis and application protocols usage analysis respectively.

From the user point of view analysis type setting determines a set of built-in variables, available to the user. After that she/he has access to their values at the filters definition and the reports specification. This set diverse for different analysis types. Besides, the type implicitly defines the corresponding set of appropriate algorithms to evaluate the variables values, but the user have not an access to the algorithms.

In the connections analysis a substream, specified by a user, is extracted from the total data stream and the report for all connections of the substream is generated. The report format is also set by a user.

The aim of the host activity analysis is to estimate frequency characteristics of each member of the given host group in relation to the total group traffic and to the total traffic of the investigated network.

Application protocols analysis allows to estimate relative frequencies of the protocol usage by the specified group.

We suppose to expand the set of available analysis types and to improve its description style by implementing a hierarchical structure, for example:

```
analysis{Connections.LongTime.Pathology}
```

In the current implementation it is impossible to use several types of analysis in the same program. The reason of that is existence of such built-in variables, those are calculated with different algorithms depending on the chosen type of the analysis.

Setting of Data Aggregation Style. The time window of the investigation may be specified by means described in the next section. The window can be divided into intervals for data aggregation by three forms of `divide` statement:

```
divide into N connections;
```

or

```
divide into N intervals;
```

or

```
divide with N seconds;
```

This division will lead to the generation of corresponding time series, characterizing the traffic. The first form divides the time window into some number of intervals, containing exactly `N` connections. The second form gives exactly `N` time intervals in the time window. The last form specifies a division into intervals with length of `N` seconds.

This mechanism allows to apply well developed theory of time series analysis.

3.2.3 Substreams Selection

Statements of this group set filters, which allow to select some TCP connections from the total stream.

Time Window Setting.

```
time = { t1 , t2 };
```

This filter passes those and only those connections, which are established inside the given time window. Values of the constants `t1` and `t2` define bounds of the window and have the following format:

```
YYYY.MM.DD.hh:mm:ss
```

where `YYYY`—4 digits of the year, `MM`—2 digits of the month, `DD`—2 digits of the day, `hh`—2 digits of the hour, `mm`—2 digits of the minute, `ss`—2 digits of the second.

Possibilities to define the time window bounds with an exactitude of fractions of a second and to define a combination of the several windows will appear in the next release.

Setting of a hosts group to analyze.

```
group = { <list> };
```

This statement sets a host group and those and only those connections are passed through the filter, which has a member of the group as a participant of the connection. Nonterminal `<list>` defines a list of conditions, separated by commas '`,`' or ampersands '`&`'. These conditions describe the group and now may be specified as:

1. IPv4 address of a host.
2. Name of a host.
3. Subnet mask in the following form:

   ```
   subnet = <IPv4>/<num>
   ```

   `<IPv4>` is an address, and `<num>` defines number of the first address bits used as subnet address.

4. A reserved word "`not`" can precede to each condition and it means its inversion.
5. One comma '`,`', separating conditions, denotes logical operation `OR`.
6. Using the ampersand '`&`' instead of the comma '`,`' means logical `AND`.

At the next versions of TCPconal we suppose to expand a set of logical primitives by including for instance analysis of substrings in a host name, restrictions on TCP ports, and considering of clients and servers separately, to number a few. Besides, it is a good idea to use the built-in variables for the filters constructing. Now these variables are used only for the reports generation.

3.2.4 Report specification

This group of statements allows to set necessary format of the report.

The report may consist of three parts:

- Heading
- Main part
- Final part

The heading allows to specify the report identification. The main part of the report contains the dynamic results of traffic processing based on the type of analysis and output patterns defined by the user. The final part is intended to output summarized and averaged characteristics of the traffic.

The main tool to form the report format is patterns mechanism to output values of the built-in variables. The variables are used to access the results output in all three parts. The values depend on the part of the report, where the the corresponding variable pattern is located. More over, in some parts of the report values of some variables have not been defined.

At the first language prototype there exists a rich enough set of the variables, divided onto the following classes:

Identificators Values of these variables are names, addresses, and titles. For example, IP address of TCP connection client (`ClientAddress`) or application protocol name (`ProtocolName`).

Time and date. Values of the class variables are time or date of some traffic events. For example, a day of TCP connection establishment (`StartDay`) or final second of host activity (`FinishSecond`).

Ordinal variables. Their values define ordinal numbers of consequently processed events in the traffic data. For example, serial number of the current TCP connection (`ConnectionNumber`).

Characteristics. These variables have values of some traffic characteristics. For example, duration of the next TCP connection (`Duration`).

Frequency characteristics. Values of the variables are frequencies of traffic events. For example, fraction of packets transfered by the given application protocol in percents (`PacketsPercent`).

Sum characteristics. They are used to sum up traffic processing. For example, the total number of connections, activated by the given group of hosts (`GroupConnectionsNumber`).

The patterns, defining the report content, are combinations of built-in variables names with textual strings. Commas are used as separators. The strings are written inside quotations, for example `"This is a host name: "`. Each variable can have a preceding string to template the output, if the default format is not adequate for the user. This is an analogy with function `printf()` of C language:

```
"TCP port is %5u" < ClientPort
```

A symbol '<' is used instead a comma to indicate that the string is a template for the variable `ClientPort`.

Report heading format.

```
headings = { <rep_string> };
```

`<rep_string>` is a list of strings and variables for output, separated by commas.

Records, specified by headings patterns, are output in the following cases:

1. If the connection analysis is set on, then one time only before the real TCP connection processing.

2. If the hosts activity or application protocols usage analysis is set on, then before processing of each interval in the time window.

Report main part format.

```
report = { <rep_string> };
```

`<rep_string>` is a list of strings and variables for output, separated by commas.

The part is output in the following cases:

1. If the connection analysis is set on, then each time after processing of each connection.
2. If the hosts activity analysis is set on, then for each host of the given group after processing of each time interval.
3. If the application protocols usage analysis is set on, then for each protocol after processing of each time interval.

Report final part format.

```
bottom = { <rep_string> };
```

`<rep_string>` is a list of strings and variables for output, separated by commas.

Report final part is output in the following cases:

1. If the connection analysis is set on, then only one time after the whole traffic processing completion.
2. If the hosts activity or application protocols usage analysis is set on, then after each time interval processing, when the main part of the report has been output for this interval.

4 The prototype implementation

The prototype of TCPconal language was implemented as subsystem of the prototype of TCPconan system. This system is a TCP traffic analyzer and it is maintained by the authors.

TCPconan system consists of the following modules:

capture. It serves to capture raw TCP packets and to extract the corresponding data from the each captured packet.

machine. It emulates TCP machines of each participant of TCP connection with the aim to gather data of each TCP connection.

pconan. It performs initial processing of detected TCP connections.

condb. It supports a data base containing data of all detected connections.

conan. It performs TCP connection data processing specified by a researcher.

interface. It supports a user interface to specify TCP connections data processing.

TCPconal compiler is built in the **conan** module. This module takes a program in TCPconal created by the module **interface** as an input. Then **conan** compiles the program into a special inner table, which contains the following data:

- Filters definitions to use for substream extracting.
- List of necessary characteristics to evaluate.
- Description of the report format.

According to the table **conan** requests the module **condb** for necessary data of selected connections and performs specified processing and the results output.

TCPconal language is described by LALR(1) grammar. The compiler was written with well–known tools `flex` and `bison`. All modules of TCPconan system are written in the C programming language.

5 Results of Experiments

We have performed series of experiments to test the system TCPconan and the TCPconal language. One Ethernet segment was chosen as a testbed. This segment contains hosts belonging to mathematical and physical departments of University of Petrozavodsk. The whole TCP traffic in the

range of some hours was investigated. Results of some these experiments are presented below.

5.1 Connections analysis

A TCPconal program was designed to output the following characteristics of each detected connection:

- Serial number.
- Names of participants (active and passive side)
- Application protocol name.
- Duration in seconds.
- Status of connection establishment and termination.
- Volume of transfered octets and packets in both directions.
- Average throughput in octet/second and packet/second

The report contains records, in which some ASCII lines are assigned for each connection. A short extraction from the report is presented below:

```
1) 17:43:35-17:43:35
delta.cs.karelia.ru -> proxy.karelia.ru
apps=81, duration=1sec, status={NOSYN,ABbyCLI}
octets->0 (0.00oct/sec); packets->1 (1.00pack/sec)
octets<-1176 (1176.00oct/sec); packets<-1 (1.00pack/sec)

2) 17:43:38-17:43:38
epsilon.cs.karelia.ru -> delta.cs.karelia.ru
apps=auth, duration=1sec, status={SYN,FINbySRV}
octets->9 (9.00oct/sec); packets->10 (10.00pack/sec)
octets<-36 (36.00oct/sec); packets<-9 (9.00pack/sec)

3) 17:43:44-17:43:44
epsilon.cs.karelia.ru -> delta.cs.karelia.ru
apps=1431, duration=1sec, status={SYN,FINbyCLI}
octets->2055 (2055.00oct/sec); packets->11 (11.00pack/sec)
octets<-0 (0.00oct/sec); packets<-6 (6.00pack/sec)
```

More then 1,000 TCP connections were monitored. A result of the experiments is that the most popular server in the investigated segment is proxy server (port 81) at the host `proxy.karelia.ru`. This server passes through itself the most part of WWW and ftp traffic.

5.2 Hosts Activity Analysis

A TCPconal program was designed to output the following characteristics of each active host of the segment:

- Name and address.
- Start and finish times of host activity.
- Number of connections, packets and octets which belong the host treated as a client, as a server and in both roles.

The text of the program is introduced in the appendix. Here is a short fragment of the report generated:

```
students.soros.karelia.ru (194.85.173.117)
1998.12.3 17:36:53:677866  -  1998.12.3 20:58:32:100164
CliCon=0 (0.00%), SrvCon=293 (9.52%), AllCon=293 (10.60%)
CliPack=0 (0.00%), SrvPack=13991 (10.24%), AllPack=2383 (4.44%)
CliOct=0 (0.00%), SrvOct=373216 (15.05%), AllOct=373216 (11.92%)

gamma.cs.karelia.ru (194.85.172.137)
1998.12.3 17:36:53:447884  -  1998.12.3 20:17:51:325746
CliCon=42 (9.05%), SrvCon=0 (0.00%), AllCon=42 (1.52%)
CliPack=1409 (4.43%), SrvPack=0 (0.00%), AllPack=1409 (2.78%)
CliOct=39471 (3.06%), SrvOct=0 (0.00%), AllOct=39471 (1.26%)
```

The measurements were performed continuously during five hours and the generated report showed, in particular, that 62% of all data octets have been exchanged through server `proxy.karelia.ru`. This confirm the result of the experiment mentioned above.

5.3 Application Protocols Usage Analysis

A TCPconal program was designed to output the following characteristics of each TCP application protocol (protocols, that have not used, are omitted in the report):

- Name and TCP port of an application protocol.
- Start and finish times of protocol activity detecting.
- Number and percent of connections established by the protocol as client, as server and both.
- Corresponding numbers and percents of packets and octets for each of three classes above.

A short extraction from the report is presented below:

```
auth(113)
 1998.12. 4|17:40:40: 27076 --- 1998.12. 4|21:23:40:427045
CliCon=0 (0.00%), SrvCon=104 (8.33%), AllCon=104 (4.17%)
CliPack=0 (0.00%), SrvPack=823 (0.84%), AllPack=823 (0.40%)
CliOct=0 (0.00%), SrvOct=3607 (0.01%), AllOct=3607 (0.01%)

ftp(21)
 1998.12. 4|17:40:40: 17077 --- 1998.12. 4|21:24:17:624192
CliCon=0 (0.00%), SrvCon=105 (8.64%), AllCon=105 (4.42%)
CliPack=0 (0.00%), SrvPack=3782 (3.89%), AllPack=3782 (1.86%)
CliOct=0 (0.00%), SrvOct=72580 (0.24%), AllOct=72580 (0.17%)
```

As a result we got that fractions of octets transfered by the most popular protocols are 40% for WWW and 15% for ftp-data.

5.4 Intruders monitoring

The visual analysis of TCPconan reports on connection analysis helped us to get not only standard expected results, but let us discover strange activity of one host. This activity can be treated as preparation to nonlegal access to a host. More exactly, this activity is called "TCP port scanning". The activity essence is that intruder's host attempts to establish TCP connections with different ports (services) of the host under attack. This allows to the intruder to discover available services. If the attacked host has not a service, it aborts the connection (ABbySRV status). Otherwise the connection is started normally and the intruder aborted it by himself (ABbyCLI status). After discovering active services the intruder can try to use them for the nonlegal access.

As an example, see the following fragment of the report:

```
109) 16:40:16-16:40:16
k143.karelia.ru (194.85.172.221) -> epsilon.cs.karelia.ru
apps=echo, duration=1sec, status={NOSSYN,ABbySRV}
octets->0 (0.00oct/sec); packets->1 (1.00pack/sec)
octets<-0 (0.00oct/sec); packets<-1 (1.00pack/sec)

113) 16:40:18-16:40:18
k143.karelia.ru (194.85.172.221) -> epsilon.cs.karelia.ru
apps=ftp, duration=1sec, status={SYN,ABbyCLI}
octets->0 (0.00oct/sec); packets->3 (3.00pack/sec)
octets<-111 (111.00oct/sec); packets<-2 (2.00pack/sec)

114) 16:40:18-16:40:18
k143.karelia.ru (194.85.172.221) -> epsilon.cs.karelia.ru
apps=smtp, duration=1sec, status={SYN,ABbyCLI}
octets->0 (0.00oct/sec); packets->4 (4.00pack/sec)
octets<-88 (88.00oct/sec); packets<-2 (2.00pack/sec)

115) 16:40:18-16:40:18
k143.karelia.ru (194.85.172.221) -> epsilon.cs.karelia.ru
apps=time, duration=1sec, status={NOSSYN,ABbySRV}
octets->0 (0.00oct/sec); packets->1 (1.00pack/sec)
octets<-0 (0.00oct/sec); packets<-1 (1.00pack/sec)

117) 16:40:18-16:40:19
k143.karelia.ru (194.85.172.221) -> epsilon.cs.karelia.ru
apps=telnet, duration=1sec, status={SYN,ABbyCLI}
octets->0 (0.00oct/sec); packets->4 (4.00pack/sec)
octets<-12 (12.00oct/sec); packets<-3 (3.00pack/sec)
```

The generated report shows that a host `k143.karelia.ru` tried to establish TCP connections with a host `epsilon.cs.karelia.ru`, using a various set of ports (apps = ftp, telnet, time, echo, domain, ...). Some of them part were aborted by the host `epsilon.cs.karelia.ru` (NOSYN, ABbySRV). Thus, the host `k143.karelia.ru` has detected that the host `epsilon.cs.karelia.ru` does not support services echo, time, www, nntp, pop3 (status=NOSYN), and does support services ftp, smtp, telnet (status=SYN).

A special type of analysis may be implemented in the TCPconal language for detection of this kind of clients behavior.

6 Conclusion

The prototype of a new language TCPconal was introduced in this paper. A purpose of the language is to specify various types of TCP traffic data processing. These problems often arise in distributed systems investigation. Today TCP traffic analysis becomes one of the most popular approach for network management. TCPconal provides flexibility and expandability for setting type of analysis and specifying a report format. These features are promised to be very useful for any kind of the research in this area.

The implemented prototype of the language is oriented to solve three types of problems: connections analysis, frequency hosts activity analysis, and frequency application protocols usage analysis. All of them are everyday tasks in network management and their goal is the estimation of general state of the network and its resources.

The language is generated by LALR(1) grammar. This let us use `flex` and `bison` tools to construct the compiler. The last fact means that the language is easy to modify and expand.

Today TCPconal is built in the TCPconan system, which is an analyzer of TCP traffic. The results of the experiments are described.

The main adventure of TCPconal is its flexibility and universality. It is oriented to solve various types of problems which arise or may arise in any network research.

References

[1] Network Management Basics. Cisco Systems, Inc. 1996.

[2] *Vadim A. Ponomarev, Iouri A. Bogoiavlenski, Timo Alanko* An Ethernet Segment Performance and Workload Characterization Using Set of Filters Based on Free Software Tools, in this Proceedings.

[3] `http://cio.cisco.com/warp.public/732/netflow/index.html`

[4] *Kojo M., Raatikainen K., Liljeberg M., Kiiskinen J., and Alanko T.* An Efficient Transport Service for Slow Wireless Telephone Links. *IEEE Journal on Selected Areas of Communication; Special Issue on*

Networking and Performance Issues of Personal Mobile Communications, 15(7), pages 1337–1348, September 1997.

[5] *Korzun D. G.* Design of a language for network traffic filtering and data processing. Bachelor of Science Thesis. University of Petrozavodsk, 1997.

[6] *Kay J.* CAIDA Measurement Tool Taxonomy. 1997.

[7] *Parker S., Schmechel C.* Some Testing Tools for TCP Implementors. Network Working Group, RFC 2398, August 1998.

[8] *Paxon V.* TCP-reduce documentation. `http://www.cs.berkeley.edu/~suchi/classes/networks/tools/tcpreduce.html`

[9] *Osterman S.* TCPtrace Home Page. `http://jaroc.cs.ohiou.edu/software/tcptrace/tcptrace.html`

[10] *Paxon V.* Automated Packet Trace Analysis of TCP Implementations. Proceedings of SIGCOMM'97, September 1997.

[11] *Paxon V.* Measurements and analysis of End-to-End Internet Dynamics. Ph.D. dissertation, University of California, Berkeley, April 1997.

[12] *Mathis M., Mahdavi J.* Diagnosing Internet Congestion with a Transport Layer Performance Tool. 1996.

[13] *Murayama Y., Yamaguchi S.* DBS: a powerfool tool for TCP performance evaluations. 1997.

[14] Netperf: A Network Performance Benchmark. Hewlett–Packard Company, 1995.

[15] *Danzig P. B, Jamin S.* tcplib: A Library of TCP Internetwork Traffic Characteristics. Report CS-SYS-91-01, Computer Science Department, University of Southern California, 1991.

[16] *Jacobson V., Leres C., McCanne S.* Tcpdump documentation. `http://www.cs.berkeley.edu/ suchi/classes/networks/tool/tcpdump.html`

[17] *H. Balakrishnan, V. Padmanabhan, S. Seshan, M. Stemm, R. Katz.* TCP Behavior of a Busy Internet Server: Analysis and Improvements. 1997.

Appendix

Here we introduce an example of TCPconal program aimed to specify standard host activity analysis. An example of the program report fragment is presented in section 5.2.

```
analysis = {hosts};     # to analyze hosts
# time window
time = {
  1998.01.01.00:00:00,  # start monitor
  1999.11.01.16:03:35   # finish monitor
};
# there are 25 subintervals
divide into 25 connections;
# output report headings
headings = {
 "Hosts analysis:", RET,
 "-----------------------------------------", RET
};
# main part of output report
report = {
# Name and address of a host
 HostName, "(",HostAddress,")", RET,
# time of discovery host activity
 " ", StartYear, ".", StartMonth,".", StartDay, " ",
 StartHour,":", StartMinute,":", StartSecond,
 "  -  ",
# time of finishing of host activity
 FinishYear,".", FinishMonth,".", FinishDay, " "
 FinishHour,":", FinishMinute,":", FinishSecond,
 RET,

# Number of connections for a host
 "CliCon=", ClientConnectionsNumber,
       " (", ClientConnectionsPercent, "%), ",
 "SrvCon=", ServerConnectionsNumber,
       " (", ServerConnectionsPercent, "%), ",
 "AllCon=", ConnectionsNumber,
```

```
       " (", ConnectionsPercent, "%)",
 RET,

# Number of packets for a host
 "CliPack=", ClientPacketsNumber,
       " (", ClientPacketsPercent, "%), ",
 "SrvPack=", ServerPacketsNumber,
       " (", ServerPacketsPercent, "%), ",
 "AllPack=", PacketsNumber,
       " (", PacketsPercent, "%)",
 RET,

# Number of octets for a host
 "CliOct=", ClientOctetsNumber,
       " (", ClientOctetsPercent, "%), ",
 "SrvOct=", ServerOctetsNumber,
       " (", ServerOctetsPercent, "%), ",
 "AllOct=", OctetsNumber,
       " (", OctetsPercent, "%)",
 RET
};

# format of final part of output report
bottom = {
 "-----------------------------------------", RET,
# number of octets from clients
 "ClientOctets=", ClientGroupOctetsNumber, ", ",
# number of octets from servers
 "ServerOctets=", ServerGroupOctetsNumber, ", ",
# total number of octets in a group
 "AllOctets=", GroupOctetsNumber, ". ",
 RET,

# number of packet from clients
 "ClientPackets=", ClientGroupPacketsNumber, ", ",
# number of packets from servers
 "ServerPackets=", ServerGroupPacketsNumber, ", ",
```

```
# total number of packets in a group
 "AllPackets=", GroupPacketsNumber, ". ",
 RET,

# number of connections from clients
 "ClientConnections=", ClientGroupConnectionsNumber, ", ",
# number of connections from servers
 "ServerConnections=", ServerGroupConnectionsNumber, ", ",
# total number of connections in a group
 "AllConnections=", GroupConnectionsNumber, ". ",
 RET,
 "-------------------------------------------", RET,
RET
};
```