Budapest University of Technology and Economics
Faculty of Electrical Engineering and Informatics
Department of Telecommunications and Media Informatics

Péter Megyesi

Matching Algorithm for Network Traffic Descriptive Strings

(Hálózati Forgalmat Leíró Karakterláncok Illesztését Végző Algoritmus Tesztelése)

Supervisors:

Dr. Sándor Molnár

Dr. Géza Szabó (Ericsson Hungary Ltd.)

Budapest

2011

Contents

Összefoglaló

Az internet egyre nagyobb térhódításával egyre gyorsabb és gyorsabb hálózati eszközöket kell tervezni. Az ilyen nagysebességű eszközök - mint routerek, tűzfalak, mobil átjárók, stb. - tesztelése nehéz feladat. Az internet szolgáltatók ritkán vállalják fel a valós idejű tesztek kockázatát és a rögzített forgalmi adatok terjesztése - elsősorban a felhasználói anonimitás miatt - szintén korlátolt.

Ennek a problémának a megoldására készítettünk egy keretrendszert az Ericsson Hungary Kft. és a BME TMIT Nagysebességű Hálózatok Laboratóriuma együttműködésével [1, 2]. A rendszer tipikus felhasználói viselkedések emulálásával képes hálózati forgalmat rögzíteni, majd ezen rögzített adatok alapján nagysebességű forgalmat előállítani. A tipikus felhasználói viselkedések azonosítása speciális karakterláncok segítségével történik, mely utal az adott felhasználó által használt alkalmazások típusára. A keretrendszer akkori állapotában a tipikus viselkedések keresését és illesztését a teljes forgalomra egy teljes egyezést kereső algoritmus végezte. A keretrendszer ezen részét tovább fejlesztettük és lecseréltük egy heurisztikus pontozási rendszerrel működő algoritmusra, mely képes megtalálni azt a tipikus felhasználói viselkedést, mely egy tetszőleges forgalommintához a legjobban hasonlít. Ezen algoritmus adja a dolgozat fő témáját. A dolgozatban bemutatásra kerül az általunk épített forgalom emuláló keretrendszer általános felépítése.

Ezen felül részletesen áttekintésre kerül a továbbfejlesztett rendszerben használt forgalomleíró karakterláncok előállítása, illetve az illesztő algoritmus alapelvei. A dolgozatban bemutatja az algoritmus pontozási rendszerének tesztelését egy mesterségesen előállított adatbázison keresztül. Bemutatjuk a legoptimálisabb teszteset alkalmazását azon a valós mérési eredményekből kapott adatbázison, mely az emuláló keretrendszer alapját képzi.

Abstract

As the Internet spreads more and more engineers have to design faster network devices. Testing such high speed devices - like routers, firewall, mobile gateway, etc. - is an unsolved problem. Internet Service Providers are not willing to take the risks of online testing and sharing recorded traffic data is also limited due to user privacy.

As a solution for this problem a framework has been made by the co-operation of Ericsson Hungary Ltd. and High Speed Networks Laboratory at the Budapest University of Technology and Economics [1, 2]. The tool is capable to emulate typical user Internet activities, record their network traffic and create a high speed network stream using the previously recorded data. The definition of typical user activities uses special strings which refers the applications run by the given user. The framework at that time was using a full match algorithm for covering the entire traffic stream. Ever since this part of the framework has been replaced by an approximately string match algorithm which uses a heuristic scoring scheme. The new algorithm is capable of finding the best matching typical user activity for an arbitrary traffic pattern. This algorithm is the main topic of this paper.

This paper presents the general architecture of the traffic emulation tool. I will present the generation of the traffic description strings and the principles of the approximately string match algorithm. The scoring scheme of the algorithm is tested via an artificially generated database. The optimal test case is also tested in a database created by a real traffic measurement which is the base of the traffic emulation tool.

1. Introduction

Today the Internet has become an important part of our lives. While a few years ago we could only access the Internet at home or at our workplace, now it is ordinary to find a public Wi-Fi area in every corner in a metropolis. Moreover, by the introduction of new generation mobile networks and smart phones people can browse the Internet almost anywhere. More and more of our common devices can access to the Internet for providing further services: televisions can run web browsers, refrigerant can order food, and heating systems can be controlled from distant locations.

As a result these tends grow the Internet traffic every year. Thus Internet Service Providers have to upgrade their network continuously with new devices which can handle the increased traffic. Testing such high speed devices - like routers, firewall, mobile gateway, etc. - is an unsolved problem. The structure of thousands of users' aggregated Internet traffic is very complex. However a normalized device test would require as realistic data as possible. Testing online in an operator's network would provide realistic results but it could cause unexpected failures since the devices were previously untested. This is a risk that network operators are not willing to take.

Another solution is to record multiple users' aggregated Internet traffic and replay it every time a device is tested. Although this technique is the most frequent way of testing high speed devices accessing this form of testing data is complicated. The main reason for that is user privacy: network operators can not share the private data of their users with a third party. To solve this every recorded data must go through an anonymity process after which every trace of the source of the traffic is erased. Even after this process the organization who owns these forms of testing data can not hand it over to a third party.

For a better solution we created a framework in the co-operation of Ericsson Hungary Ltd. and High Speed Networks Laboratory at the Department of Telecommunications and Media Informatics. The Traffic Emulation Tool (TET) is capable to emulate typical individual user Internet activities and record the generated network traffic. With these recorded user data the system can assemble a high speed network flow which contains the traffic of multiple users at the same time. During this process the framework uses real traffic measurement data to extract typical user behaviors. These behaviors are defined by special strings called Traffic Descriptive Strings (TDS). The main purpose of this paper is to present the functions of the Traffic Descriptor Strings in the Traffic Emulation Tool from the definition to the assembly of the high speed aggregated traffic.

In the next section I will present the general architecture of the Traffic Emulation Tool. The second part of this paper deals with the creation of the Traffic Descriptive Strings using real traffic measurements. I will also introduce the algorithm which TET uses to find to most similar typical user behavior for any given TDS. This algorithm is also tested via an artificially created database which helps us adjust its scoring system. Finally, I will present the result the algorithm gives to the real measurement result provided by Ericsson Hungary Ltd.

2. The architecture of the Traffic Emulation Tool

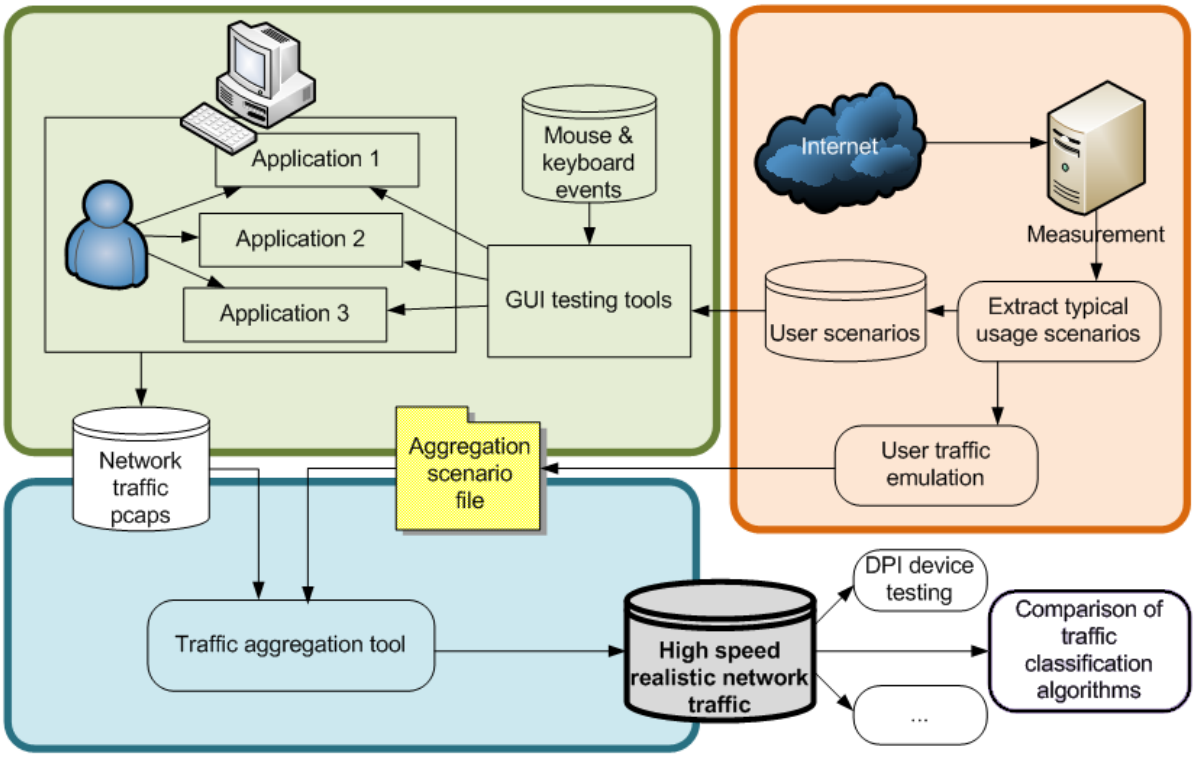

Figure 1. The architecture if the Traffic Emulation Tool

The architecture of TET can be seen in Figure 1. As the figure shows TET can be divided to three separate parts. The first part – called the Input Processor - is responsible for processing the real traffic measurements. This part of the framework will be detailed later on, for now I only introduce the format of the Traffic Descriptive Strings. A simple TDS looks like the following:

AZAZABZABZACZAZAZ

In a TDS the "Z" character has a special meaning; it separates the minutes in the user's activity. Thus this example presents a seven minute long user scenario. The other characters refer for the application types the user was running. This means that this user was running an "A" type application for all seven minutes long, used a "B" type application in the third and the fourth, and a "C" type in the fifth minute.

The Input Processor has three tasks. Firstly, it has to translate the input measurement to Traffic Descriptive Strings which will be detailed in Section 3. Then the system extracts the typical user scenarios from the TDSs which will be the input for the emulation part. During this procedure we tend to find shorter time of activities which occur frequently in different users' traffic. Then the algorithm tries to replace the user's whole traffic with a series of typical scenarios. This procedure creates the Aggregation Scenario File which will be the input of the Traffic Aggregation Tool. Both of these algorithms use the scoring mechanism introduced in Section 4.

The second part of the framework is called the User Emulator. Although this part can operate individually it was designed to co-operate with the Input Processor. The User Emulator's main task is to control a remote computer by launching previously written AutoIt script. AutoIt is a freeware BASIC-like scripting language designed for automating the Windows GUI and general scripting [3]. This paper will not present any specification from the AutoIt scripting language; I will mention only the main purpose behind using it. The full documentation of AutoIt can be found in [3], while a short summary and the specific script integrated to TET are presented in [1].

With AutoIt we wrote several scripts which can automate the running of popular internet application – such as opening a web page with a browser or downloading a torrent file. With a series of these scripts we can emulate one type of user sitting behind a computer. For easier control we created a website for the User Emulator. The main page of this site is shown in Figure 2.

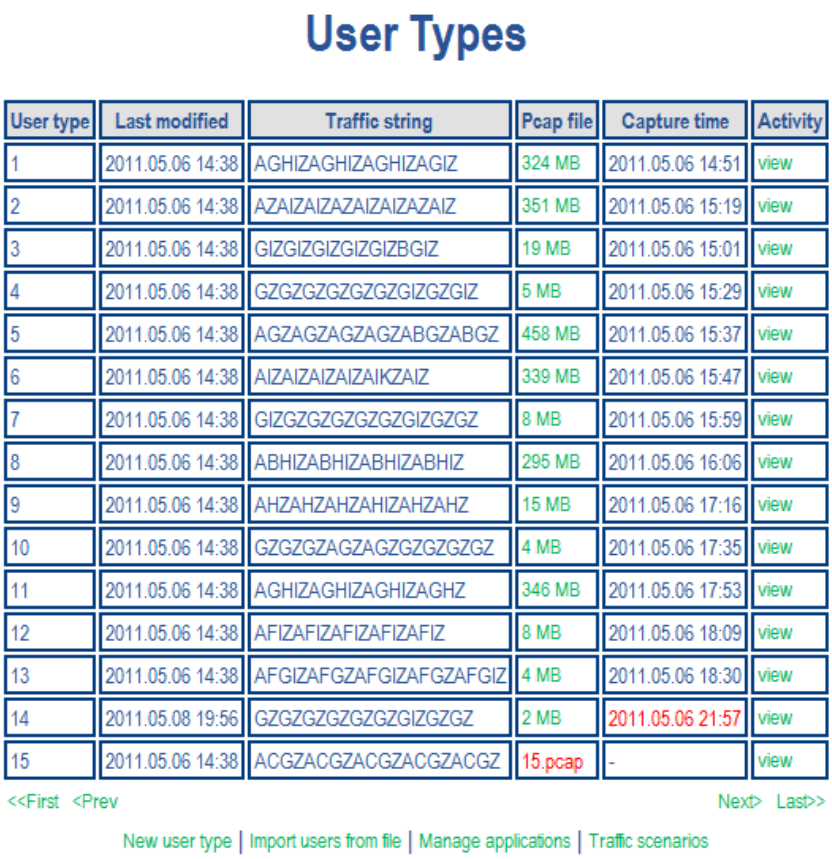

Figure 2. The main page of TET's website

This table shows the information about the typical user scenarios integrated into TET. As it can be seen we define a user type by a TDS string. With the links found at the bottom of th page we can integrate the typical user behaviors extracted by the Input Processor or we can create a new user scenario manually. The manual create navigates to a page similar to the modification page which came up clicking the *view* link at the table's last column. This page can be seen if Figure 3.

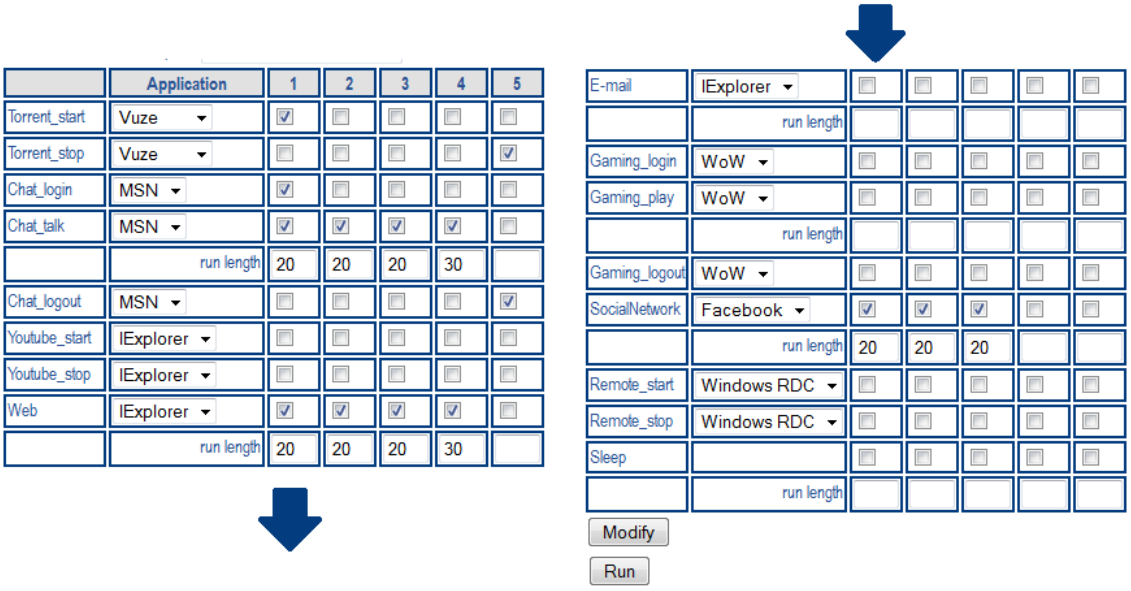

Figure 3. Example for a user scenario.

In this page the series of events can be seen which will be executed in the remote computer. The events executed from top to bottom and from left to right. Thus this example scenario does the following: it launches the torrent application Vuze and logs into the MSN server. After that it uses the MSN for chatting 20 second, browses the Internet for 20 second and then it navigates to Facebook for another 20 second. These three activities add up to one minute total. This minute of activity is repeated for additional two times while in the fourth minute the user chats for 30 second and browses the Inter for another 30 second which is an another minute total. For the last step it closes the opened running applications finishing its activity.

Our assembled test environment is shown in Figure 4. The web site is running on a Linux server which is connected to Internet via a symmetric leased line. The Windows test computer which runs the applications can reach the Internet via a bridge interface through the Linux server. When a user scenario is played the server connects to the client using a Telnet connection and executes the given AutoIt scripts. This procedure is controlled by the Perl's Expect library [4].

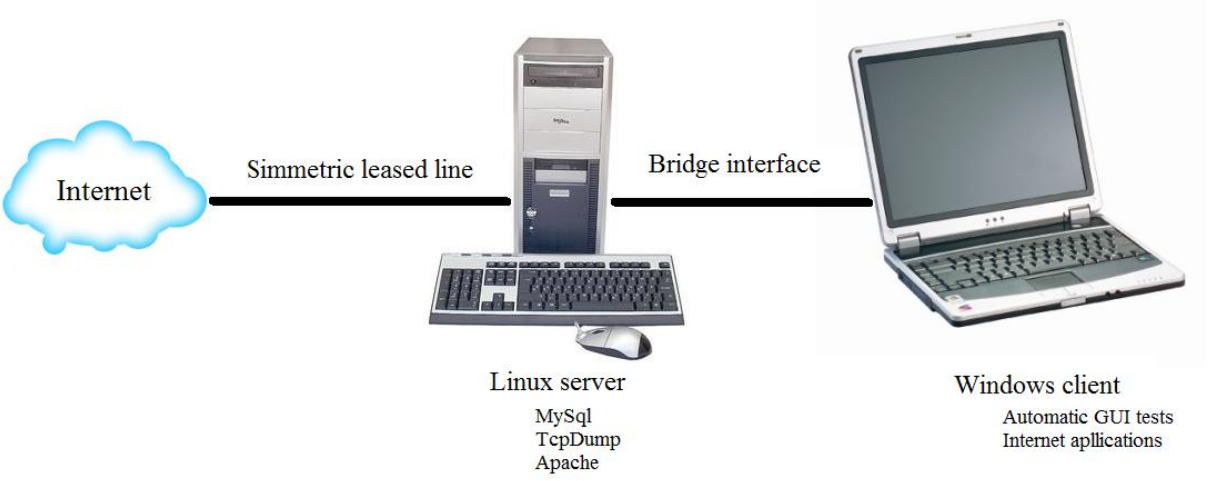

Figure 4. The assembled test environment.

When a user scenario is running, the server captures the network traffic using *tcpdump* and stores the recorded data in *libpcap* format. With these recorded *.pcap* files and the Aggregation Scenario File made by the Input Processor, the third part of the framework can assemble a high speed aggregated traffic stream.

The Traffic Aggregation Tool was developed by Bálint Csatári in C language [2]. Its task is to open the given *libpcap* files, sort the packet they contain to the right time order, and make the required packet manipulations. For the bigger picture Figure 5 shows the format of the Aggregation Scenario File.

```
1223378304.17461|/opt/TET/pcaps/506.pcap|213.16.101.154|192.168.10.185|
1223378304.41010|/opt/TET/pcaps/469.pcap|213.16.101.154|192.168.10.211|
1223378304.48218|/opt/TET/pcaps/439.pcap|213.16.101.154|192.168.10.58|
1223378304.57481|/opt/TET/pcaps/416.pcap|213.16.101.154|192.168.10.104|
1223378304.65761|/opt/TET/pcaps/439.pcap|213.16.101.154|192.168.10.85|
1223378304.69096|/opt/TET/pcaps/439.pcap|213.16.101.154|192.168.10.80|
```

Figure 5. The format of the Aggregation Scenario File.

Every line contains four pieces of information. The first one is timestamp in microsecond accuracy. This has to be unique and increasing line-by-line through the whole file. This timestamp defines where the first packet of the recorded trace has to be shifted. The other packets will be shifted with the same time interval so the interarrival times in the individual users' traffic remain the same after the modification. The scenario, which is actually a trace stored as a capture file is given by the second parameter using the absolute path of the file. The last two parameters are used for distinguishing the individual user's traffic. As we use the same computer to record the trace files, every packet contains the IP address of our Windows clients. The Traffic Aggregation Tool changes this IP address in the IP header to the one given as the fourth parameter.

The tool is also capable to search for this IP address in the packet's payload in both hexadecimal and string formats. However, this feature is optional as the extra operations increase the application's runtime significantly. After the packet modifications the program also recalculates the checksums found in the IP and the TCP/UDP headers. An example for the packet manipulations can be seen in Figure 6.

During the aggregation process the tool can operate in two different modes. In the first mode – called the offline mode - the output trace is stored in a *libpcap* file in the server's hard drive. The other mode is the online transmission mode where the application sends the packet out to a given network interface. Further information about the timing method of the online mode and performance test of the Traffic Aggregation Tool can be found in [2].

This finishes the overview about the Traffic Emulation Tool. The next sections will detail the functions of the Input Processor part and the role of the Traffic Descriptive String in TET's operations.

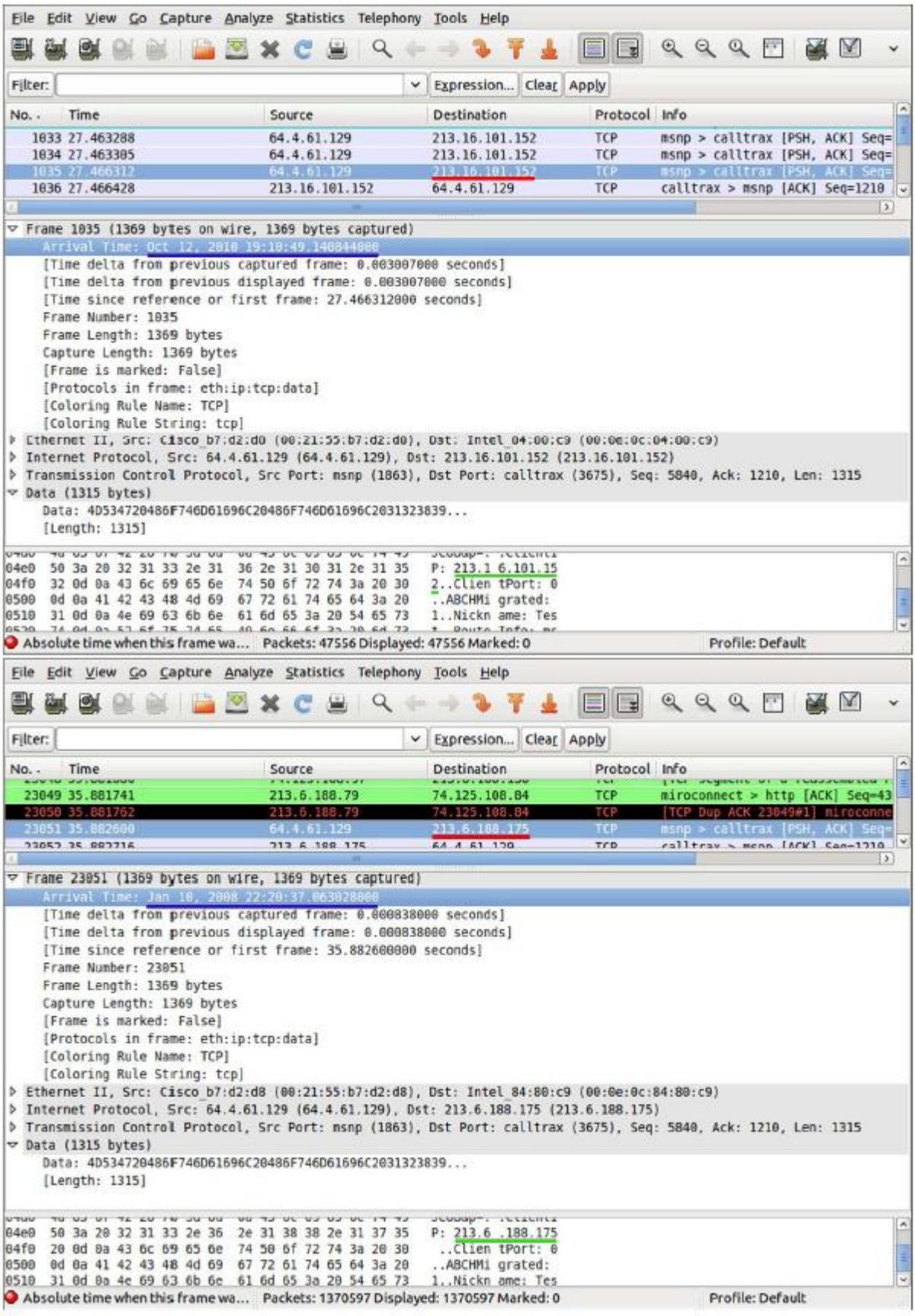

Figure 6. Packet manipulation is practice [2].

3. About the Traffic Descriptive Strings

As I have mentioned in the previous section, the Traffic Emulation Tool was designed to work based on real traffic measurement. The measurement result which is integrated into TET was made available by Ericsson Hungary Ltd. This data was recorded in 2009 in the network of a Swedish operator and been processed by Ericsson's internal Deep Packet Inspection tool named *Captool*. The source and the binary of Captool is not public due to the fact that it is a very detailed, comprehensive, and reliable traffic classification application. The DPI tool has concise output of the flows found in the input *pcap* trace file. The output format of Captool – which is the input of the Traffic Emulation Tool - can be seen in Table 1.

start	end	ip	up	down	protocol	functionality	crypt	sp	app
1223503020	1223503080	192.168.160.141	0	262	BitTorrent	file-sharing	\N	P2P	\N
1223503020	1223503080	192.168.163.108	44837	2715325	RTSP	media-playback	\N	\N	\N
1223503020	1223503080	192.168.209.74	41747704	702635	BitTorrent	file-sharing	\N	P2P	\N
1223503020	1223503080	192.168.208.2	92	92	HTTP	web-browsing	\N	Google	Firefox
1223503020	1223503080	192.168.163.168	3082	1148	\N	\N	SSL	\N	\N
1223503020	1223503080	192.168.228.16	138	0	SIP	\N	\N	\N	\N
1223503020	1223503080	192.168.229.83	771	0	ICMP	system	\N	\N	\N
1223503020	1223503080	192.168.164.210	799	10850	HTTP	software-update	\N	Microsoft	Windows
1223503020	1223503080	192.168.208.193	37058	467701	HTTP	\N	\N	\N	\N
1223503020	1223503080	192.168.162.48	278	134	\N	\N	\N	\N	\N
1223503020	1223503080	192.168.163.131	15260	229116	HTTP	\N	\N	\N	\N
1223503020	1223503080	192.168.208.116	69688	4828996	HTTP	media-playback	\N	YouTube	\N
1223503020	1223503080	192.168.209.110	91297	1699695	HTTP	\N	\N	\N	\N
1223503020	1223503080	192.168.162.36	0	438	ICMP	system	\N	\N	\N
1223503020	1223503080	192.168.163.217	19476	993380	HTTP	file-download	\N	iTunes	iTunes
1223503020	1223503080	192.168.228.136	1218	2425	\N	\N	SSL	\N	\N
1223503020	1223503080	192.168.163.199	248	11849	BitTorrent	file-sharing	\N	P2P	\N
1223503020	1223503080	192.168.209.74	448579	26880542	RTSP	media-playback	\N	Qbrick	\N
1223503020	1223503080	192.168.229.63	7347	9598	BitTorrent	file-sharing	\N	P2P	\N
1223503020	1223503080	192.168.164.231	6639	39912	HTTP	email	\N	Yahoo	Firefox
1223503020	1223503080	192.168.160.247	563	278	\N	instant-messaging	\N	MSN	\N
1223503020	1223503080	192.168.228.184	92	92	HTTP	\N	\N	Google	\N
1223503020	1223503080	192.168.162.11	4239894	74067	DirectConnect	file-sharing	\N	P2P	\N

Table 1. The output format of Captoo

Every line contains the following information about one flow in the traffic stream:

- the start time of the flow in POSIX format
- the end time of the flow in POSIX format
- the user's IP address (due to privacy reasons the public addresses had been switched to randomly chosen local addresses)

- the number of bytes transmitted in upstream direction
- the number of bytes transmitted in downstream direction
- the type of the protocol
- the functionality of the flow
- the encryption method
- the service provider the connection is made to
- the application that generated to flow

As the previous example showed the same protocol can be used for different activities – like HTTP for web browsing, e-mailing, downloading, etc – and the same functionality can use different protocols – for example both RTSP and HTTP can be used for media-playback. Thus during the generation of Traffic Descriptive String we use the information under the functionality column. There are thirteen different types of functionalities that the Captool is able to distinguish. However, we only integrated twelve to the Traffic Emulation Tool as the flows of *system* functionality – such as DNS and ICMP messages – are usually generated by other process. These integrated functionalities and the characters assigned to them are to following:

- A: file-sharing
- B: media-playback
- C: remote access
- D: software update
- E: voice over IP
- F: gaming
- G: instant-messaging
- H: social-networking
- I: web browsing
- J: file-download
- K: e-mail
- L: photo-sharing

As the Table 1 shows these report files come in minute resolution so during the generation of the Traffic Descriptive Strings we kept that time partitioning. That is the reason why the "Z" characters separate one minute of traffic in a TDS. During the conversation we collect for every individual user which of these types of applications was used. In practice it means that if the transmitted bytes of a functionality is grater then 10000, we add its character to the user's TDS. The measurements results we integrated into TET were containing information about 1747 users in one day of time period. Thus a TDS of a heavy Internet user contains thousands of characters (as the number of "Z" characters equals the time interval in minutes the user was generating traffic). Later on we will use the timestamp when the user has begun his activity so we added this information to the output. Thus the format of the User TDS File can be seen in Figure 7.

```
1223378280 ABGIZABGIZABGIZABGIZABGZABGZABGZABGIZABGIZABGIZABGIZ
1223378280 ABZABIZABIZABZABZABDZAZAZAZAZAZAZAZAZAZAZAZAZAZAZAZAZ
1223378280 AGIZAGZAGIZAGZAGIZAGZAGZAGZAGZAGZAGZABGIZABGZABGIZABGZABGZ
1223378280 AGZAGIZABGIZABGIZABGZABGZABGIZABGIZABGZABGZAGZAGZAGZAGZAGZAGZ
1223378280 AGZAGIZABGIZABGZABGIZABGZAGZABGIZABGIZABGZABGIZAGIZAGZAGZAGZ
1223378280 GZAGZAGIZAGIZAGZAGZAGZAGZAGIZAGIZABGZABGZBGZBGZABGZBGZGZGZAGIZ
```

Figure 7. The format of the User TDS File.

All of the mentioned twelve kinds of Internet activities can be emulated by at least one AutoIt script. For better handling of the automatic conversation of Traffic Descriptive Strings to user scenarios we categorized these activities into three different classes:

Class 1 - active activities:

These activities require active action by the user so only one type of these activities can run at the same time. Also the launch of the application doesn't generate any source of characteristic traffic. These activities run in web browsers so the following traffic types belong to this class: web browsing, social-networking, e-mailing and photo-sharing.

Class 2 - passive activities:

These kinds of applications generate traffic in the background so they don't require any action by the user. Therefore multiple applications can run at the same time from this class. During the automatic control we use two scripts to control these kinds of applications: the first lunches the application with the proper attributes and the second one simply close it. The following traffic types belong to this class: file-sharing, media-playback, software update and file download.

Class 3 - login followed by an active activity

This class is similar to the first one. The difference between them is here we use applications which need some sort of login process. This login process usually generates a unique type of traffic and only appears when the application is started. When the application is logged in it also generates some sort of background traffic synchronizing with a server. During the automatic control we use three scripts to control these kinds of applications: the first one performs the login. The second is the active part of the class so the same rule applies than in the first class: only one type of active action can run at a time. The third script performs the logout which can also cause unique type of traffic and also closes the application. The following traffic types belong to this class: VoIP, gaming, instant messaging and remote access.

There is a script in TET which can automatically convert a TDS to a user scenario. The script splits the Traffic Descriptive Strings to minutes along the "Z" characters and does the following procedure for every minute:

1. closes every Class 2 type application which was in the previous minute but not in the actual
2. performs the logout and closing procedure for every Class 3 type application which was in the previous minute but not in the actual

3. starts every Class 2 type application which was not in the previous minute but already in the actual
4. performs the login process for every Class 3 type application which was not in the previous minute but already in the actual
5. counts the number of active activities which are in the minute and runs each of them one after the another for equal time period

After the last minute the script also closes every running application so the next user scenario will start with no Internet application running. This procedure allows us to integrate user scenarios by only giving Traffic Descriptive Strings. Our intentions were integrating only those TDSs which describe shorter term of typical behaviors using the Internet. In the next subsection I will present the algorithm which finds these in the User TDS File.

3.1. Finding typical user scenarios

As I have previously mentioned we tend to emulate shorter term user scenarios which are frequently occur in the traffic stream. This means that in the Traffic Emulation Tool the lengths of the integrated user scenarios are between four and ten minutes. The reason for the lower limit is that too short scenarios wouldn't generate stabile traffic. For example if a torrent client runs for only one minute the number of the peer-to-peer connections wouldn't reach its maximum. On the other hand, the longer a user scenario is the lesser it would occur in the traffic stream. Besides for example emulating a half our long web browsing activity with three ten minute long is an acceptable option. Thus we have maximized the time length of the typical user scenarios in ten minutes.

The first step in the finding algorithm is to get the most frequent TDS substring for the input to a given time length. The function written in Perl which handles this task can be seen in Figure 8. The key point in handling fixed time length sub TDSs is to split the users TDS along the "Z" characters. That way we make sure that the individual minutes' activities won't be corrupted.

Before dealing with the substrings the function firstly split the user's TDS along three consecutive "Z" characters. Consecutive "Z" characters mean that the user was not transmitting any data for minutes. Naturally in the emulation process we want to avoid long idle periods. Thus by this splitting the typical scenarios won't contain more than one consecutive idle minute.

After counting the occurrences of sub TDSs we keep those which can be found in the User TDS File at least a hundred times. In the list of the remaining strings we have found that there are several TSD groups which describe similar activities. An example for these groups can be seen in Table 2. This example shows five five-minute-long user activities. All of them describe a scenario where the user was using an instant-messaging application in the entire time and performed a web browsing action in one of the five minutes. The only difference between them is the minute where the web browsing happened. Thus we consider these kinds of scenarios similar so we only integrate the one with the most occurrences from these types of groups.

To get these groups we convert the Traffic Descriptive Strings into Action Strings. In practice it means that we generate a string which contains the number of the different characters in a TDS in alphabetic order. For example, the five TDSs found in Table 2 have the same Action

String which is “5G1I5Z”. The Perl function which converts a TDS into an Action String can be seen in Figure 9.

```
$minlength = 5;
open(FILE, 'usertds.txt');
while (<FILE>) {
	chomp($_);
	$_ =~ s/^\d* //;
	my @active = split('ZZZ', $_);
	foreach $act (@active) {
		my @minutes = split('Z', $act);
		my $tds = '';

		for ($i=0;(($i<$minlength)and($i<scalar(@minutes))); $i++) {
			$tds .= $minutes[$i].'Z';
		}

		for ($i = $minlength; $i < scalar(@minutes); $i++) {
			if (defined($motifcount{$tds})) {
				$motifcount{$tds}++;
			}
			else {
				$motifcount{$tds} = 1;
			}

			$tds =~ s/^[^Z]*Z//;
			$tds .= $minutes[$i].'Z';
		}
	}
}
close(FILE);
```

Figure 8. Perl function for counting the occurrence of five minute long TDSs.

# occurances	TDS
4578	GIZGZGZGZGZ
4428	GZGZGZGZGIZ
2060	GZGIZGZGZGZ
2034	GZGZGZGIZGZ
1971	GZGZGIZGZGZ

Table 2. Example for similar TDSs.

```
sub actionstring(@) {
	my $str = shift;
	my $chrs = 'ABCDEFGHIJKL';
	my $as = '';
	foreach (split('',$chrs)) {
		my $c = &charcount($str, $_);
		$as .= "$c$_" if ($c);
	}
	return $as;
}
```

Figure 9. Perl function for converting a TDS into Action String.

Table 3 shows the ten most frequent five-minute-long user scenarios after these considerations. We can determine by these typical scenario result that which type of applications users usually use simultaneously. The table shows that in the most frequent cases users use only one type of application or use an instant-messaging application with another type.

# occurrences	TDS	Action String	Description
109599	AZAZAZAZAZ	5A5Z	file-sharing
101981	GZGZGZGZGZ	5G5Z	instant-messaging
40775	AGZAGZAGZAGZAGZ	5A5G5Z	file-sharing and instant-messaging
14037	BZBZBZBZBZ	5B5Z	media-playback
13503	IZIZIZIZIZ	5I5Z	web browsing
9680	FZFZFZFZFZ	5F5Z	gaming
8349	GIZGIZGIZGIZGIZ	5G5I5Z	web browsing and instant-messaging
5698	KZKZKZKZKZ	5K5Z	e-mail
5016	FGZFGZFGZFGZFGZ	5F5G5Z	gaming and instant-messaging
4764	BGZBGZBGZBGZBGZ	5B5G5Z	media-playback and instant-messaging

Table 3. The ten most frequent five-minute-long user scenarios.

Another interesting result is the occurrences of user scenarios which contain social-networking activity. In social-networking we mean browsing today's popular sites such as Facebook, MySpace or Twitter. Captool is able to distinguish this kind of traffic from browsing a general website and treat it as separate functionality. In Table 4 we can see that it is more common to use another application while social-networking that using it alone.

# occurances	TDS	Action String
992	GHIZGHIZGHIZGHIZGHIZ	5G5H5I
645	BGHIZBGHIZBGHIZBGHIZBGHIZ	5B5G5H5I
603	AGHIZAGHIZAGHIZAGHIZAGHIZ	5A5G5H5I
535	HZHZHZHZHZ	5H
286	GHZGHZGHZGHZGHZ	5G5H
270	AHZAHZAHZAHZAHZ	5A5H
178	HIZHIZHIZHIZHIZ	5H5I
176	AGHZAGHZAGHZAGHZAGHZ	5A5G5H
115	AHIZAHIZAHIZAHIZAHIZ	5A5H5I

Table 4. The most frequent five-minute-long user scenarios containing social-networking.

Finally, in Table 5 I present the number of typical scenarios we have extracted from the User TDS File. The table shows what we have previously expected: the longer a scenario is the lesser it occurs in the users' traffic. As it can be seen in Table 5 in total we have integrated 748 different user scenarios into the Traffic Emulation Tool.

Length in minutes	Number of typical scenarios
4	145
5	129
6	110
7	103
8	93
9	88
10	80
sum	748

Table 5. The number of extracted typical scenarios.

3.2. Assembling Aggregation Scenario File

As I have mentioned in the Section 2 we need an input file for the Traffic Aggregation Tool. The format of the Aggregation Scenario File was given in Figure 5. In order to create this file we need both the User TDS File and the extracted typical user scenarios. During this procedure we deal with the users separately thus we process the User TDS File line by line. Since that way the output won't be sorted by time as it would be required, we need to use the LINUX inbuilt *sort* application as well to get the correct file format.

As a starting point we know two things about a user: the transmitted traffic in TDS format and the UNIX timestamp of its beginning. As the reports from which the TDSs were generated had a minute resolution, multiple users can have the same start time. During the aggregation this would occur that the first packet of these users had the exact same timestamp. To avoid this, the assembling script chooses a random number between 0 and 60 with six decimal points and adds it to the user's start timestamp. That way we grant the required millisecond timestamp accuracy as well.

Under the emulation process we mean that we try to substitute a user's long term traffic stream with consecutive previously recorded short term traces. Thus in practice we need an algorithm which is capable to cover a user's entire Traffic Descriptive String with the extracted typical user scenarios' TDSs.

The first and easiest part of this algorithm is the search for full-matching typical scenarios in the user's TDS. During this process two rules should be kept. The firstly we have to start the searching with the longer scenarios. As I have mentioned before we prefer the longer scenarios because we consider their traffic more stabile. With this action we make sure that we use the longer scenarios as much as possible. The second rule is that if we find a full-matching user scenario somewhere in the user's TDS we have to switch that substring to only "Z" characters. That way we guarantee that the minutes in the user's traffic will be covered by only one trace file. However we have to leave the minute delimiter characters in the TDS in order to properly calculate to timing information.

The Perl source code fragment which handles the full-match part can be seen in Figure 10. In the script with the help of the variable *printmode* we can operate the algorithm in different debug modes. If the variable is set 0 only the information for the Aggregation Scenario File will be printed. However setting it to 2 will print the information about the full-matching results, while setting it to 1 will print details about the approximately-match part.

```
open(FILE, $userTDSfile);
while (<FILE>) {
      my $newip = &getnewuserip();
      my ($timestamp, $usertds) = split(' ', $_);
      #shifting the user's start time in the minute with a random value
      $timestamp += rand(60);
      #full match first, starting with longer actions
      foreach $id (sort {&zcount($scenarios{$b}) <=>
                         &zcount($scenarios{$a}) } keys %scenarios) {
            my $tds = $scenarios{$id};
            while ($usertds =~ /^(.*?Z)$tds/) {
                  my $before = $1;
                  my $mins = &zcount($before);
                  my $starttime = $timestamp + 60*$mins;
                  print "$id = $tds found $mins offset\n"
                        if ($printmode == 2);
                  my $subZs = &zstring(&zcount($tds));
                  #substitution of the found tds for only Zs
                  $usertds =~ s/$tds/$subZs/;
                  print "$starttime|$dir/$id.pcap|$switchip|$newip|\n"
                        unless ($printmode);
            }
      }
      #continue with approximately string match
```

Figure 10. Perl source for handling full match.

After we found and switched the full-matching typical user scenarios the remaining TDS contains many consecutive "Z" characters. Since this "Z" runs means an inactive period or that it has been previously covered by a full-matching scenario, we split the TDS along three or more consecutive "Z" characters.

At this point we have no knowledge about how long is the user scenario which would be ideal to cover the remaining TDS. Thus the created algorithm tries every possibility. In practice it means that we find the most similar typical user scenario for the first four minutes than the first five minutes than so on to the first ten minutes. The scoring algorithm which can determine the similarities between two TDSs will be presented in Section 4.

The Perl source code fragment which realizes these functions can be found in the Appendix. After getting the relative score value from the scoring function the algorithm does not compare them instantly, it multiplies with a variable. This variable helps us favoring the longer term scenarios against the shorter ones. Section 4 also describes some test cases which helped us adjusting these *length score* values.

4. TDS scoring algorithm

When we were designing the TDS scoring algorithm the following rules were laid down:

- If we compare an A sting to a B sting, the returned score must be less than or equal to score the algorithm gives back comparing the A string with itself.
- The equality must only stand if the Action Strings of A and B are the same.
- The algorithm must inspect the time length of the TDSs and give lesser score if it differs.
- We must have a way of setting unique values for which traffic types are suitable substitutions for each other and which are completely excluded.

Taking these considerations into account, we have defined a scoring matrix labeling its rows and columns with the twelve functionality characters. We also add the "X" character which will refer to no action. Firstly, the algorithm concatenate "X" characters to the shorter string until both of them contain the same amount of characters. After this the following procedure is repeated until both strings are empty: the algorithm searches for the highest value in the scoring matrix which row character is in the first string and the column character is in the second. This value is added to the point counter, and the characters are being removed from the strings.

The Perl function which calculates the score for the two input TDS can be found in the Appendix. In the Perl script we use hash variables for handling the scoring matrix. The keys of this hash are the characters labeling the rows and their values are pointers for another hash which keys are the characters labeling the columns. The actual values of the scoring matrix can found in these hash variables. Before returning with the score the script divides the calculated points with a variable. With this variable we can decrease the calculated score depending on the differences between time lengths of the two TDSs.

In order to get the most ideal values of the scoring matrix and the length modifier we created a test database of Traffic Descriptive Strings. In contrast with the typical user scenarios this artificial database contains only TDSs in which the actions are the same in every minute. We integrated every variation of minimum four maximum ten minute long scenarios which contains maximum 4 type of traffic simultaneously. Since this database is symmetric we can calculate the number of TDS integrated into it with the following mathematical formula:

$$7 * \left[\binom{12}{1} + \binom{12}{2} + \binom{12}{3} + \binom{12}{4}\right] = 5551$$

4.1. Adjusting scoring matrix

The first test case contains values what we have thought "logical" before running any test. The idea was to categorize the traffic types to three groups by their assumed bandwidth. We have classed instant-messaging, social-networking, web-browsing, e-mailing and photo-sharing as low-bandwidth, media-playback, remote access, software update, VoIP and gaming as middle-bandwidth, and file-sharing and file download as high-bandwidth types. Thus the values in one group correspond to this consideration. The scoring matrix of the first test case can be seen in Table 6.

	X	A	B	C	D	E	F	G	H	I	J	K	L
X	0	3	2	2	2	2	2	1	1	1	3	1	1
A	3	5	2	2	2	2	1	1	1	1	4	1	1
B	2	1	5	3	3	3	3	2	2	2	1	2	2
C	2	1	3	5	3	3	3	2	2	2	1	2	2
D	2	1	3	3	5	3	3	2	2	2	1	2	2
E	2	1	3	3	3	5	3	2	2	2	1	2	2
F	2	1	3	3	3	3	5	2	2	2	1	2	2
G	1	0	2	2	2	2	2	5	4	4	0	4	4
H	1	0	2	2	2	2	2	4	5	4	0	4	4
I	1	0	2	2	2	2	2	4	4	5	0	4	4
J	3	4	2	2	2	2	1	1	1	1	5	1	1
K	1	0	2	2	2	2	2	4	4	4	0	5	4
L	1	0	2	2	2	2	2	4	4	4	0	4	5

Table 6. Scoring matrix, testcase 1.

After running the algorithm for the TDS: *IZIZIZIZIZIZAIZ* the ten strings which got the most score can be seen in Table 7. During the first few tests we will only test those TDSs which have the same time length than the input; we will deal with time differences later on. In these tables 0^{th} row will always show the scoring results with input itself.

#	score	ratio	time length	TDS
0	40	1	7	AZAZAZAZAZAZAIZ
1	86	2.15	7	ABEJZABEJZABEJZABEJZABEJZABEJZABEJZABEJZ
2	86	2.15	7	ABFJZABFJZABFJZABFJZABFJZABFJZABFJZABFJZ
3	86	2.15	7	ADEJZADEJZADEJZADEJZADEJZADEJZADEJZADEJZ
4	86	2.15	7	ADFJZADFJZADFJZADFJZADFJZADFJZADFJZADFJZ
5	86	2.15	7	AEFJZAEFJZAEFJZAEFJZAEFJZAEFJZAEFJZAEFJZ
6	86	2.15	7	ACEJZACEJZACEJZACEJZACEJZACEJZACEJZACEJZ
7	86	2.15	7	ACFJZACFJZACFJZACFJZACFJZACFJZACFJZACFJZ
8	85	2.12	7	ACDJZACDJZACDJZACDJZACDJZACDJZACDJZACDJZ
9	85	2.12	7	ABDJZABDJZABDJZABDJZABDJZABDJZABDJZABDJZ
10	85	2.12	7	ABCJZABCJZABCJZABCJZABCJZABCJZABCJZABCJZ

Table 7. Scoring algorithm test results, testcase 1.

The first thing that stands out from these results that something is wrong with the operation of the algorithm. Strings which describe way different user traffic got more score than the actual

input string with itself. The reason for this behavior is that the more length difference is between two strings the more "X" characters are added to the shorter one. Thus extra characters earn extra points. Obviously we need to invert this behavior making lesser length differences preferable. Or in another way we have to find a way to punish these differences. Writing negative values in the row and column of the "X" character will solve this problem since it will reduce the score if a length difference occurs. Table 8 shows the second test case in which the numbers are the same as in the first case, only the values are negative in the first row and column.

	X	A	B	C	D	E	F	G	H	I	J	K	L
X	0	-3	-2	-2	-2	-2	-2	-1	-1	-1	-3	-1	-1
A	-3	5	2	2	2	2	1	1	1	1	4	1	1
B	-2	1	5	3	3	3	3	2	2	2	1	2	2
C	-2	1	3	5	3	3	3	2	2	2	1	2	2
D	-2	1	3	3	5	3	3	2	2	2	1	2	2
E	-2	1	3	3	3	5	3	2	2	2	1	2	2
F	-2	1	3	3	3	3	5	2	2	2	1	2	2
G	-1	0	2	2	2	2	2	5	4	4	0	4	4
H	-1	0	2	2	2	2	2	4	5	4	0	4	4
I	-1	0	2	2	2	2	2	4	4	5	0	4	4
J	-3	4	2	2	2	2	1	1	1	1	5	1	1
K	-1	0	2	2	2	2	2	4	4	4	0	5	4
L	-1	0	2	2	2	2	2	4	4	4	0	4	5

Table 8. Scoring matrix, test case 2.

#	score	ratio	time length	TDS
0	40	1	7	AZAZAZAZAZAZAIZ
1	34	0.85	7	AZAZAZAZAZAZAZ
2	34	0.85	7	AIZAIZAIZAIZAIZAIZAIZ
3	33	0.82	7	AHZAHZAHZAHZAHZAHZAHZ
4	33	0.82	7	AGZAGZAGZAGZAGZAGZAGZ
5	33	0.82	7	ALZALZALZALZALZALZALZ
6	33	0.82	7	AKZAKZAKZAKZAKZAKZAKZ
7	27	0.67	7	AILZAILZAILZAILZAILZAILZAILZ
8	27	0.67	7	AGIZAGIZAGIZAGIZAGIZAGIZAGIZ
9	27	0.67	7	IJZIJZIJZIJZIJZIJZIJZ
10	27	0.67	7	JZJZJZJZJZJZJZ

Table 9. Scoring algorithm test results, test case 2, input 1.

Table 9 collects the best ten results for the same input TDS which we have used in the previous test case. These results are getting closer to our goal as that the winning two scenarios are the ones we have expected. However, the fact that those two scenarios got the same amount of score suggests that scoring matrix needs further refinement. Table 10 shows

the results for the same scoring matrix but used the TDS "AZAZAZAZAZAIZAIZ" as an input.

#	score	ratio	time length	TDS
0	45	1	7	AZAZAZAZAZAIZAIZ
1	40	0.88	7	AIZAIZAIZAIZAIZAIZAIZ
2	38	0.84	7	AHZAHZAHZAHZAHZAHZAHZ
3	38	0.84	7	AGZAGZAGZAGZAGZAGZAGZ
4	38	0.84	7	ALZALZALZALZALZALZALZ
5	38	0.84	7	AKZAKZAKZAKZAKZAKZAKZ
6	33	0.73	7	AZAZAZAZAZAZAZ
7	33	0.73	7	AILZAILZAILZAILZAILZAILZAILZ
8	33	0.73	7	AGIZAGIZAGIZAGIZAGIZAGIZAGIZ
9	33	0.73	7	IJZIJZIJZIJZIJZIJZIJZ
10	33	0.73	7	AHIZAHIZAHIZAHIZAHIZAHIZAHIZ

Table 10. Scoring algorithm test results, test case 2, input 2.

Based on these results we have found out that the algorithm punishes the absence of a character with an excessive rate. If one character is missing from the compared TDS not only it doesn't get the plus five point for it, but it gets a negative value from the first column. Thus we increased the negative values in the first row and lowered the values to four in the main diagonal. At the same time we also decreased the values which were previously four to three. The modified scoring matrix can be found in Table 11.

	X	A	B	C	D	E	F	G	H	I	J	K	L
X	0	-6	-5	-5	-5	-5	-5	-4	-4	-4	-6	-4	-4
A	-3	4	2	2	2	2	1	1	1	1	3	1	1
B	-2	1	4	3	3	3	3	2	2	2	1	2	2
C	-2	1	3	4	3	3	3	2	2	2	1	2	2
D	-2	1	3	3	4	3	3	2	2	2	1	2	2
E	-2	1	3	3	3	4	3	2	2	2	1	2	2
F	-2	1	3	3	3	3	4	2	2	2	1	2	2
G	-1	0	2	2	2	2	2	4	3	3	0	3	3
H	-1	0	2	2	2	2	2	3	4	3	0	3	3
I	-1	0	2	2	2	2	2	3	3	4	0	3	3
J	-3	3	2	2	2	2	1	1	1	1	4	1	1
K	-1	0	2	2	2	2	2	3	3	3	0	4	3
L	-1	0	2	2	2	2	2	3	3	3	0	3	4

Table 11. Scoring matrix, test case 3.

The results for the previous seven minute user scenario with the new matrix can be found in Table 12. As table shows, we have achieved our main goal: the algorithm prefers the scenario with only one activity over the one with both. However, another issue has been revealed: the scoring scheme replaces too soon an entire traffic type with another in the same bandwidth class. In order to solve that we have significantly decreased the values which are neither in the row or column of the "X" character nor in the main diagonal. Thus the scoring matrix for the fourth test case can be seen in Table 13.

#	score	ratio	time length	TDS
0	36	1	7	AZAZAZAZAZAIZAIZ
1	26	0.72	7	AZAZAZAZAZAZAZ
2	19	0.52	7	JZJZJZJZJZJZ
3	16	0.44	7	AIZAIZAIZAIZAIZAIZAIZ
4	14	0.38	7	AHZAHZAHZAHZAHZAHZAHZ
5	14	0.38	7	AGZAGZAGZAGZAGZAGZAGZ
6	14	0.38	7	ALZALZALZALZALZALZALZ
7	14	0.38	7	AKZAKZAKZAKZAKZAKZAKZ
8	10	0.27	7	CZCZCZCZCZCZCZ
9	10	0.27	7	BZBZBZBZBZBZBZ
10	10	0.27	7	DZDZDZDZDZDZDZ

Table 12. Scoring algorithm test results, test case 3.

	X	A	B	C	D	E	F	G	H	I	J	K	L
X	0	-5	-4	-4	-4	-4	-4	-3	-3	-3	-5	-3	-3
A	-3	4	0.1	0.1	0.1	0.1	0.1	0	0	0	0.2	0	0
B	-2	0	4	0.2	0.2	0.2	0.2	0.1	0.1	0.1	0	0.1	0.1
C	-2	0	0.2	4	0.2	0.2	0.2	0.1	0.1	0.1	0	0.1	0.1
D	-2	0	0.2	0.2	4	0.2	0.2	0.1	0.1	0.1	0	0.1	0.1
E	-2	0	0.2	0.2	0.2	4	0.2	0.1	0.1	0.1	0	0.1	0.1
F	-2	0	0.2	0.2	0.2	0.2	4	0.1	0.1	0.1	0	0.1	0.1
G	-1	0	0.1	0.1	0.1	0.1	0.1	4	0.2	0.2	0	0.2	0.2
H	-1	0	0.1	0.1	0.1	0.1	0.1	0.2	4	0.2	0	0.2	0.2
I	-1	0	0.1	0.1	0.1	0.1	0.1	0.2	0.2	4	0	0.2	0.2
J	-3	0.2	0.1	0.1	0.1	0.1	0.1	0	0	0	4	0	0
K	-1	0	0.1	0.1	0.1	0.1	0.1	0.2	0.2	0.2	0	4	0.2
L	-1	0	0.1	0.1	0.1	0.1	0.1	0.2	0.2	0.2	0	0.2	4

Table 13. Scoring matrix, test case 4.

Table 14 shows the results for the seven-minute file-sharing with two-minute web browsing, while in Table 15 the input TDS contained one more "I" character. As the tables demonstrate all the previous issues have been solved. The results also show that if the input TDS contains at least three minutes of a traffic type than the most similar scenario will contain it all seven minute long. Of course, if we would want that change happen one more character later we could continue the adjustment of the scoring matrix. However, we consider this behavior acceptable and strict to the scoring matrix given in Table 13.

#	score	ratio	time length	TDS
0	36	1	7	AZAZAZAZAZAIZAIZ
1	26	0.72	7	AZAZAZAZAZAZAZ
2	21	0.58	7	AIZAIZAIZAIZAIZAIZAIZ
3	13.4	0.37	7	AHZAHZAHZAHZAHZAHZAHZ
4	13.4	0.37	7	AGZAGZAGZAGZAGZAGZAGZ
5	13.4	0.37	7	ALZALZALZALZALZALZALZ
6	13.4	0.37	7	AKZAKZAKZAKZAKZAKZAKZ
7	8.2	0.22	7	ADZADZADZADZADZADZADZ
8	8.2	0.22	7	ACZACZACZACZACZACZACZ
9	8.2	0.22	7	AEZAEZAEZAEZAEZAEZAEZ
10	8.2	0.22	7	AFZAFZAFZAFZAFZAFZAFZ

Table 14. Scoring algorithm test results, test case 4, input 1.

#	score	ratio	time length	TDS
0	40	1	7	AZAZAZAZAIZAIZAIZ
1	28	0.7	7	AIZAIZAIZAIZAIZAIZAIZ
2	25	0.62	7	AZAZAZAZAZAZAZ
3	16.6	0.41	7	AHZAHZAHZAHZAHZAHZAHZ
4	16.6	0.41	7	AGZAGZAGZAGZAGZAGZAGZ
5	16.6	0.41	7	ALZALZALZALZALZALZALZ
6	16.6	0.41	7	AKZAKZAKZAKZAKZAKZAKZ
7	12.3	0.30	7	ADZADZADZADZADZADZADZ
8	12.3	0.30	7	ACZACZACZACZACZACZACZ
9	12.3	0.30	7	AEZAEZAEZAEZAEZAEZAEZ
10	12.3	0.30	7	AFZAFZAFZAFZAFZAFZAFZ

Table 15. Scoring algorithm test results, test case 4, input 2.

4.2. Adjusting length difference dividers

As I have mentioned at the beginning of this section, we have the possibility to modify the calculated score if the time lengths of the Traffic Descriptive Strings differ. Since the test in the previous subsection were investigating only TDSs which have the same time length, Table 16 shows the result for a test case where we let the algorithm score all the TDSs which time length difference from the input is within two. The input TDS was a simply seven-minute-long file-sharing activity.

#	score	ratio	time length	TDS
0	28	1	7	AZAZAZAZAZAZAZ
1	28	1	7	AZAZAZAZAZAZAZ
2	23	0.82	8	AZAZAZAZAZAZAZAZ
3	21	0.75	6	AZAZAZAZAZAZ
4	18	0.64	9	AZAZAZAZAZAZAZAZAZ
5	14	0.5	5	AZAZAZAZAZ
6	11	0.39	5	AIZAIZAIZAIZAIZ
7	11	0.39	5	AGZAGZAGZAGZAGZ
8	11	0.39	5	AHZAHZAHZAHZAHZ
9	11	0.39	5	AKZAKZAKZAKZAKZ
10	11	0.39	5	ALZALZALZALZALZ

Table 16. Scoring algorithm test with different time lengths.

After analyzing these results, we draw the conclusion that even two minutes of length differences make the Traffic Descriptive Strings too distant. Thus we only need to choose a value for minute time difference which the final score will divide if the two TDSs differ in time. After investigating many test results, we defined the length difference divider value for 1.3. Tables 17 and 18 present the how the divider works in practice. In the first test case the input TDS contains only one minute of web browsing activity so we consider the only-file-sharing scenarios more similar even if their time length differs. However, the second test case shows that we consider two minutes of the web browsing activity enough to make a better substitute than the only-file-sharing for different time length.

#	score	ratio	time length	TDS
0	32	1	7	AZAZAZAZAZAZAIZ
1	27	0.84	7	AZAZAZAZAZAZAZ
2	21.53	0.67	8	AZAZAZAZAZAZAZAZ
3	15.38	0.48	6	AZAZAZAZAZAZ
4	14	0.43	7	AIZAIZAIZAIZAIZAIZAIZ
5	12.30	0.38	6	AIZAIZAIZAIZAIZAIZ
6	10.2	0.31	7	AHZAHZAHZAHZAHZAHZAHZ
7	10.2	0.31	7	AGZAGZAGZAGZAGZAGZAGZ
8	10.2	0.31	7	ALZALZALZALZALZALZALZ
9	10.2	0.31	7	AKZAKZAKZAKZAKZAKZAKZ
10	9.38	0.29	6	AKZAKZAKZAKZAKZAKZ

Table 17. Scoring algorithm test with different time lengths.

#	score	ratio	time length	TDS
0	36	1	7	AZAZAZAZAZAIZAIZ
1	26	0.72	7	AZAZAZAZAZAZAZ
2	21	0.58	7	AIZAIZAIZAIZAIZAIZAIZ
3	20.76	0.57	8	AZAZAZAZAZAZAZAZ
4	17.69	0.49	6	AIZAIZAIZAIZAIZAIZ
5	14.61	0.40	6	AZAZAZAZAZAZ
6	13.4	0.37	7	AHZAHZAHZAHZAHZAHZAHZ
7	13.4	0.37	7	AGZAGZAGZAGZAGZAGZAGZ
8	13.4	0.37	7	ALZALZALZALZALZALZALZ
9	13.4	0.37	7	AKZAKZAKZAKZAKZAKZAKZ
10	11.84	0.32	6	AKZAKZAKZAKZAKZAKZ

Table 18. Scoring algorithm test with different time lengths.

4.3. Adjusting length bonuses

As I have mentioned in Section 3.2 during the assembly of the Aggregation Scenario File it is possible to prefer longer term scenarios over short ones. To realize this function we declare an array in which the value under an index will be the multiplier for that time length. During these test we set the *print mode* to 1 variable in the script presented in Section 3.2. This debug mode displays detailed information about the method as the algorithm chooses the ideal value to cover the beginning of a longer TDS. The output format of this mode can be seen in Figure 11.

This example shows how close the results can get to each other in the procedure. After examining many results we have found that the deviation between the score ratio of

consecutive minutes are within 1-2%. Thus we declared the length bonus variables to ensure a 3% bonus against the TDSs which time length lesser by one minute. Table 19 contains the exact values that TET uses to calculate the length modified scores.

time length	length bonus multiplier
4	1
5	1.03
6	1.06
7	1.09
8	1.12
9	1.15
10	1.18

Table 19. Length bonus multipliers.

```
This is what left:
      ABGZABGZABGIZABGIZABGIZABGZABGZABGZABGIZABGIZABGZABGZABGZABGIZABGZ
Testing the first 4 min:
0 56 1    4 ABGZABGZABGIZABGIZ  4A4B4G2I4Z
1 50 0.89 4 ABGIZABGIZABGIZABGIZ 4A4B4G4I4Z
 - +2I
Testing the first 5 min:
0 72 1    5 ABGZABGZABGIZABGIZABGIZ  5A5B5G3I5Z
1 66 0.91 5 ABGIZABGIZABGIZABGIZABGIZ 5A5B5G5I5Z
 - +2I
Testing the first 6 min:
0 84 1    6 ABGZABGZABGIZABGIZABGIZABGZ  6A6B6G3I6Z
1 75 0.89 6 ABGIZABGIZABGIZABGIZABGIZABGIZ 6A6B6G6I6Z
 - +3I
Testing the first 7 min:
0 96 1    7 ABGZABGZABGIZABGIZABGIZABGZABGZ  7A7B7G3I7Z
1 84 0.87 7 ABGIZABGIZABGIZABGIZABGIZABGIZABGIZ 7A7B7G7I7Z
 - +4I
Testing the first 8 min:
0 108 1   8 ABGZABGZABGIZABGIZABGIZABGZABGZABGZ 8A8B8G3I8Z
1 93 0.86 8 ABGZABGZABGZABGZABGZABGZABGZABGZ 8A8B8G8Z
 -3I +
Testing the first 9 min:
0 124 1    9 ABGZABGZABGIZABGIZABGIZABGZABGZABGZABGIZ  9A9B9G4I9Z
1 109 0.87 9 ABGIZABGIZABGIZABGIZABGIZABGIZABGIZABGIZABGIZ 9A9B9G9I9Z
 - +5I
Testing the first 10 min:
0 140 1    10 ABGZABGZABGIZABGIZABGIZABGZABGZABGZABGIZABGIZ
                                          10A10B10G5I10Z
1 125 0.89 10 ABGIZABGIZABGIZABGIZABGIZABGIZABGIZABGIZABGIZABGIZ
                                          10A10B10G10I10Z
 - +5I
The winner is the first 5 mins with length modified score: 0.91
```

Figure 11. Debug mode of the approximately string match algorithm.

5. Conclusion

In this paper I have presented the Traffic Emulation Tool which is capable of define typical user scenarios from real traffic measurements, capture and store their traffic in *libpcap* files and create a high speed aggregated traffic stream. In TET's operations Traffic Descriptive Strings have a key role: they are the base of both the extraction of typical user scenarios and the compilation of the input file for the Traffic Aggregation Tool.

The Traffic Descriptive Strings are generated by real traffic measurement results provided by Ericsson's internal deep packet inspection tool. With a suitable algorithm we can analyze these strings and find out the general way of how users using the Internet. I have presented an algorithm which is capable to examine the similarities between two TDS, and with the help of it we are able determine which one of integrated typical scenarios is the best replacement for any given TDS.

The scoring algorithm can be adjusted by many aspects. I have examined many test cases in an artificially created database in order to find the most suitable scoring values. The final numbers which are currently integrated into the Traffic Emulation Tool are also given.

List of figures

List of tables

References

[1]. Tamás Szőke, Péter Megyesi: Design and development of a traffic emulation tool (Forgalom emuláló keretredszer tervezése és fejlesztése), Scientific Students' Associations Conference, Budapest University of Technology and Economics, 2010

[2]. Bálint Csatári: Framework for Comparison of Traffic Classification Algorithms, Master's Thesis, Budapest University of Technology and Economics, 2011

[3]. http://www.autoitscript.com/site/autoit

[4]. http://search.cpan.org/~rgiersig/Expect-1.15/Expect.pod

Appendix

3.2. Perl source for handling the approximately match.

```
      #now split what has left
      foreach $left (split(/Z{3,}/,$usertds)) {
            #we skip the short ones (less the 3 min)
            while ($left){
                  last if (&zcount($left) < 3);
                  print "\nThis is what left:$left\n" if ($printmode==1);
                  my $comp = '';
                  my $switchid = 0;
                  if (&zcount($left) < 11) {
                        print "This is a length we have, find similar:\n"
                              if ($printmode == 1);
                        ($switchid, $comp) = &findsimilar($left,1);
                        $comp = $left;
                        $left = '';
                        print "The difference between: $comp and
                        $users{$switchid} is: \n" if ($printmode == 1);
                  } else {
                        my $bestscore = 0;
                        foreach $min (4..10) {
                              print "Testing the first $l min:\n" if
                                    ($printmode == 1);
                              my ($tds, $las) = &splitbymin($left, $min);
                              my ($simid, $relscore) =
                                    &findsimilar($tds,0);
                              $relscore *= $lengthbonus[$min];
                              if ($relscore > $bestscore) {
                                    $bestscore = $relscore;
                                    $switchid = $simid;
                                    $comp = $tds;
                              }
                        }
                        $left =~ s/$comp//;
                        print "----->\t$comp\t$bestscore\t$left\n" if
                              ($printmode == 1);
                        print "The winner is the first ",&zcount($comp),"
                              mins: $comp with length modified score:
                              $bestscore\n" if ($printmode == 1);
                        print "The difference between: $comp and
                        $users{$switchid} is: \n" if ($printmode == 1);
                  }
                  if ($usertds =~ /^(.*?)$comp/) {
                        my $before = $1;
                        my $mins = &zcount($before);
                        my $starttime = $timestamp + 60*$mins;
                        my $subZs = &zstring(&zcount($comp));
                        #substitution of the $comp for only Zs
                        $usertds =~ s/$comp/$subZs/;
                        print "$starttime|$dir/$switchid.pcap|
                        $switchip|$newip|\n" unless ($printmode);
                  }
            }
      }
}
close(FILE);
```

4. The TDS scoring algorithm in Perl language.

```
sub scoretds(@) {
      my ($stringA, $stringB) = @_;
      my $chrs = 'ABCDEFGHIJKL';
      my @lengthdiff = (1, 1.1);
      my $lengthmod = $lengthdiff[abs(&zcount($stringA) -
                                    &zcount($stringB))];
      my $point = 0;
      $stringA =~ s/Z//g; $stringB =~ s/Z//g;
      if (length($stringA) > length($stringB)) {$stringB .=
                  &xstring(length($stringA) - length($stringB));}
      if (length($stringA) < length($stringB)) {$stringA .=
                  &xstring(length($stringB) - length($stringA));}
      while (($stringA ne '') or ($stringB ne '')) {
            my $switcha = my $switchb = '';
            my $max = -20;
            foreach $chra (split('',$stringA)) {
                  foreach $chrb (split('',$stringB)) {
                        if ($scoretable{$chra}->{$chrb} > $max) {
                              $switcha = $chra;
                              $switchb = $chrb;
                              $max = $scoretable{$chra}->{$chrb};
                        }
                  }
            }
            $point += $max;
            $stringA =~ s/$switcha//;
            $stringB =~ s/$switchb//;
      }
      $point = $point / $lentghmod;
      return $point;
}
```