HOMEWORK FOR QUANTITATIVE SYSTEMS ANALYSIS EXAM

MASSIMO NOCENTINI
5422207, GROUP 1

July 12, 2014

Abstract

This article collects the work I did in order to support my Quantitative Systems Analysis exam. The goal is to study resources usage respect a collection of user profiles, each one of them producing different network traffic and CPU/memory consumption patterns in different time windows. I performed some experiments repeating them over time, analyzing results with a piece of Smalltalk software, initially implemented for the problem at hand, but it turns out to be far more general.

CONTENTS

In this very first section we approach the problem taking it apart in major components toward fulfilling our needs. We introduce the main concept of *user profile*, what we need to study and what quantities deserve our attention.

1.1 *User profiles*

A central concept is that of *user profile* which abstract the behavior of a person using a computer. In this homework I've considered three user profiles: a *landlady*, a *computer scientist* and a *Free Software Foundation supporter*. Informally, a landlady will use the computer for simple tasks such as checking emails, surfing the web and watching some YouTube videos; on the other hand a computer scientist will use the computer for writing TEXdocuments, programming (in Smalltalk especially), collaborating with colleagues via instant messaging and checking its email account. Finally a FSF supporter just makes available its bandwidth supplying Ubuntu like distros via Torrent protocol. Under the previous setting we think a fairly comprehensive patterns of network traffic and CPU/memory consumption can be analyzed, varying some operative condition as explained next.

1.2 *Different workloads in different time windows*

In order to study the user profiles I arranged that each one of them performs their activities repeating the same sequence of actions (with some minor variants) in three different time windows, namely in the morning, in the afternoon and in the evening. This allow us to characterize the same activity in different time windows with different workloads, in order to compare them in a quite real scenario.

For example the *landlady* user profile will do a quick check of emails in the morning, writing short messages to her friends in the afternoon while in the evening she replies to important mails with more care, preparing some drafts of her reply. On the other side, the *computer scientist* writes articles in the morning, programs in the middle of the day and reviews the morning work in the evening. Finally, the *FSF supporter* keeps its Bit Torrent client open for the entire day, switching the distros image supplied between morning, afternoon and evening.

1.3 *Quantities to assess*

We are interested on the resource consumption for each user profile during the three time windows as explained in the previous section. In particular we identifies the following quantities to be relevant for our analysis:

MEMORY the amount of free, buffered and cached memory;

CPU the time spent running non-kernel code, kernel code and idle cycles;

NETWORK TRAFFIC incoming and outgoing packets through a network interface, in particular we perform a complete packet sniffing.

The previous list is the tip of the iceberg, since we collect actually much more information, such as the number of blocked, running and waiting process, the number of context switches per second, the number of interrupt per second and a lot of details for each network packet. We focus our attention here on the listed ones, however section 4 reports summary tables with additional aggregated measures.

To collect data about quantities of interest we've instrumented a laptop (from now on we call it the "instrumented system" or just "system") used by user profiles. Since our study is a kind of monitoring, we do not need to setup neither hardware/software probes nor failure injectors.

The system have a Core i7 64bit Intel architecture, equipped with 8 GB of RAM and powered by Ubuntu Linux 14.04 LTS distribution. In next sections we describe two tools we've used to collect measures.

2.1 *vmstat*

To measure memory and CPU usage, we've used the *vmstat* [3] command line tool: it is very lightweight and releases summary lines containing a super set of requested quantities. In particular the command we've used is the following:

```
vmstat <sec> -n -S M
```

just a few explanations: every *sec* seconds it drops a summary line on the standard output, "-n" doesn't repeat header lines periodically, "-S M" use Megabytes as measure unit for memory quantities. Usually we redirect the printed output to a file in order to post process data.

While repeating experiments we change *sec* respect to duration of the experiment at hand, that is for "short" experiments we need more granularity, hence we use $sec = 1$ (this is the case for *experiment one* regarding *landlady* user profile). On the other hand, we increase it for "long" experiment (this is the case for experiments regarding *FSF supporter* user profile).

A "negative" aspect of this tool is that has a maximum rate of 1 second, hence for some experiment we couldn't use data produced since they arrived too far one from the other.

2.2 *SNORT*

To measure network traffic we've used the *SNORT* tool [1], as required by homework assignment. We used *SNORT* to save *all* packages, in their complete structure (ie, including headers and payloads) saving them in binary files. So, according the manual [2], we sniffed packages with the following command:

```
sudo snort -l <log-directory>/ -b
```

In order to analyze the collected data, we access information saved as binary file with the "read" feature provided by *SNORT* itself, redirecting the output into a plain text file on which we run a "cleaning" *sed*[4] script which transform it into another plain text file containing a row for each incoming/outgoing network package, hence no SNORT custom rule has been used. We value this approach because it allow us to have a *complete* history of what happened, implementing the processing phase in Smalltalk as described in section 4.

In this section we plan the experiments to gain insight about resource usage by different user profiles, describing experiments for each profile in a dedicated subsection and the motivations underlying each plan.

3.1 *Landlady experiments plan*

Experiments design

We created the following experiments in order to stress the activity of mail checking in particular, since this is the main task performed by our landlady user profile. Here we're not very interested on CPU and memory consumption, instead we remarkably vary the workloads to see changes in network traffic: especially, in *experiment two* we pair writing short messages with watching a YouTube video, while in *experiment three* we perform drafts preparation using Google web mail interface in order to study the impact of automatic saving policy.

Experiment two has an additional video watching action just to differentiate substantially from *experiment one*, since our goal here is to compare data relative to the *same* experiment collected in *different* days, instead of comparing data relative to *different* experiments collected in the *same* day, toward studying if experiments are repeatable or not.

Experiments descriptions

EXPERIMENT ONE In the morning, the landlady open Mozilla Firefox browser, log in into her *GMail* account, checks for new mail and log out after reading new messages, under the assumption that she reads at least two messages.

EXPERIMENT TWO In the afternoon, the landlady open Mozilla Firefox browser, log in into her *GMail* account, checks for new mail and writes three short greetings to her friends. Here by "short greetings" we mean a message with at most 50 words. While writing greetings, she watches her favorite YouTube video [1] for about two minutes at the 480p resolution.

EXPERIMENT THREE In the evening, the landlady open Mozilla Firefox browser, log in into her *GMail* account, checks for new mail and prepare two draft for an important reply (assuming that at least one message received during the day deserve careful response). After saving the two draft, it randomly select one of those and send it. Here by "important reply" we mean a message with at least 120 words.

[1] `https://www.youtube.com/watch?v=yaaoEyWQ6eI`

Experiments design

We created the following experiments in order to observe resources consumption of multiple applications running at the same time, in particular *experiments one* focus on email checking while typesetting a document; *experiment two* and *three* focus on programming and video conference (partly instant messaging).

In each experiment we couple an application that produce network traffic with one that consume CPU and memory, in order to have a profile quite opposite to that of *landlady*. In this setting would make sense compare *different* experiments given that they share the same activities.

Particular interest can be pointed to *experiment two* where we can study (small) network traffic during an instant messaging session between two peers while doing a computation intensive task. On the other hand in *experiment three* we can observe (huge) network traffic generated by a Google Hangout video call while running a process with low priority.

Experiments descriptions

EXPERIMENT ONE In the early morning, the computer scientist open the Mozilla Firefox browser, logs in into his GMail account and do a quick check for new messages; we assume that he reads at least one message. Then, keeping the browser running, he writes a draft of at least 400 words using the Emacs text editor, compiles it using TEXtwo times. Then he writes one message to his advisor with the compiled draft as attachment. Finally he logs out from Google web mail interface.

EXPERIMENT TWO At noon, the computer scientist do some hacking on his Pharo Smalltalk system, executing an intensive fluid dynamics analysis for a gas network. This analysis is repeated three times, each of one is 30 seconds longer. Waiting for analysis results, he talks with his colleague using Google Hangout directly from his Google web mail interface, sending and receiving at least 10 short phrases. Here by "short phrase" we mean a message with at most 10 words and we also assume that the log in had already succeeded before starting the registering tools.

EXPERIMENT THREE In the afternoon, the computer scientist discuss the fluid analysis results obtained previously in a joint work through a video call using Google Hangout, via web interface using the Mozilla Firefox browser. While the conversation takes place the computer scientist performs one more time the fluid dynamics analysis. Here we assume that no textual instantaneous messages are exchanged, that the conversation is about 1'30" longer and that the log in had already succeeded and the call is already running before starting the registering tools.

3.3 *FSF supporter experiment plan*

Experiments design

We created the following experiments in order to observe exactly one activity respect network traffic. Both three experiments focus on the Transmission client for Bit Torrent protocol supplying, without any limit on upstream bandwidth, three different Ubuntu-like distro images, one for each experiment. For all repetitions, we assume Transmission is already running before starting the registering tools.

This settings allow a comparison of both the *same* experiment among *different* repetitions, both *different* experiments respect the *same* repetition. As a side note, this allow us to make a comparison of the traffic generated by the three Ubuntu-like distros.

Experiments descriptions

EXPERIMENT ONE In the morning, the FSF supporter opens the Bit Torrent Transmission client application in order to supply bandwidth providing latest Kubuntu Linux ISO image distribution. He doesn't set any bandwidth upload limit and keeps Transmission running for about two minutes.

EXPERIMENT TWO In the afternoon, the FSF supporter opens the Bit Torrent Transmission client application in order to supply bandwidth providing latest Ubuntu Linux ISO image distribution. He doesn't set any bandwidth upload limit and keeps Transmission running for about two minutes.

EXPERIMENT THREE In the evening, the FSF supporter opens the Bit Torrent Transmission client application in order to supply bandwidth providing latest Xubuntu Linux ISO image distribution. He doesn't set any bandwidth upload limit and keeps Transmission running for about two minutes.

3.4 *Execution methodology*

Given an user profile, we do five repetitions for each experiment. We'll associate each repetition to a *numbered day*, in particular we've used *Day one*, *Day two*, *Day three*, *Day four* and *Day five* for the first, second, third, fourth, fifth repetition, respectively. We ensure that each repetition happen within time window as required by experiment description, even though instant timings can varies among repetitions of the same experiment. In order to make an experiment repeatable as much as possible, we strictly follows the experiment plan of each user profile, doing the fixed set of action required by each experiment, focusing to eliminate any change of deviation from the main track. In order to make this work quite real, we've chosen to not read the same mails, with the same contents, but if a new mail is actually present in the mail account, the user can choose to read it or not. This assumption introduce a sort of non-determinism, on the other hand increase the truthfulness of the entire work. All network traffic registration about experiment's repetitions has been performed in the same environment, in particular the instrumented system always registers from a domestic LAN network, using the same IP address 192.168.0.4. The internet connection used has 20 Mb in download and 1 Mb in upload.

In this section we'll describe the software architecture developed, dealing with it in the former part and we give our results interpretation in the latter one.

4.1 *Software architecture*

We developed a quite rich architecture to support our results interpretation. We would like to have an automatic tool that parses Snort log files, builds a comprehensive facts collection and supplies a fancy and flexible interface (programmatically, of course) to plot quantities of interests, exporting automatically in .eps files, and to aggregate measures into a summary, exporting automatically in TEXtabular files. In the following we provide a brief description of this piece of work.

4.1.1 *Pharo Smalltalk programming environment and TDD*

We implemented our code in Pharo Smalltalk, an image based programming environment. It allow a natural Test Driven Development style, so we strictly follow it. In what rest of this section we refer to classes we've implemented and that are available in the image, assuming the curious reader has set up the environment correctly (see section 5 for instructions).

The class *AQSLogFileTest* is the test suite we've created to drive our implementation: it contains test methods that assert on log file parsing, facts collecting, plotting and summary generation, both for "under the hood" implementation detail, both for producing plots and tables to be attached in this document.

4.1.2 *Snort log files parsing*

To set the stage we need to handle binary Snort log file in a more comfortable way. We saved original Snort binary log files in the file system, organizing them hierarchically, by experiment number, user profile and day number, in the given order. Hence we process every log file by performing a depth-first visit, applying to each one of them a bash script which does some clean work, especially removing empty and separating lines, joining the rest toward the creation of new log file which has exactly one line per packet; here we report a chunk of the latter log (intentionally breaking page boundaries):

```
07/01-07:41:07.220098 92.104.129.98:6881 -> 192.168.0.4:51413 UDP TTL:45 TOS:0x0 ID:0 IpLen:20 DgmLen:129 DF
07/01-07:41:07.220190 192.168.0.4 -> 92.104.129.98 ICMP TTL:64 TOS:0xC0 ID:48370 IpLen:20 DgmLen:157 Type:3  Co
07/01-07:41:07.277850 98.232.94.206:59805 -> 192.168.0.4:51413 TCP TTL:112 TOS:0x0 ID:14283 IpLen:20 DgmLen:48
07/01-07:41:07.277898 192.168.0.4:51413 -> 98.232.94.206:59805 TCP TTL:64 TOS:0x0 ID:41997 IpLen:20 DgmLen:40 D
07/01-07:41:07.528356 91.65.105.145:51413 -> 192.168.0.4:51413 UDP TTL:47 TOS:0x0 ID:32138 IpLen:20 DgmLen:58 D
```

Performing this cleaning allow to handle files much shorter than the original ones produced by the "read" Snort feature: for example, one of our files shrinks from about 70000 to 14000 lines.

4.1.3 *Building facts collection*

Following [6], we introduced the concept of *fact*, which collect all relevant information about events we care about. We do not use a database back end, preferring live objects to talk with. This allow an interactive session where the user can query the facts collection using all the pretty stuff supplied by Pharo Smalltalk to inspect and explore them, for example it is possible to filter facts using a predicate and use that selection programmatically. This methodology has the drawback to be slower respect a relational database engine, while gaining in flexibility.

4.1.4 *Selecting facts toward actions*

We use the facts collection to define a selection over it, reifying this concept in its own dedicate class *FactsSelection*. This allow us to use the selection as a "trampoline" object [7] toward actions, from our point of view, plotting and summarizing. Moreover, this make the system extensible, since if a developer would like to use facts for a different goal, let say drawing histograms, it is required only to implement an action object able to receive a selection of facts. The two action classes relative to plotting and summarizing are *FactsPlotter* and *FactsSummary*, respectively.

It is possible to create a selection of fact providing a user profile, an experiment number and a day, additionally to an arbitrary predicate.

4.1.5 *Plotting a facts selection*

We supply one implementation for plotting a facts selection, with two main features: the one allow to scatter a quantity of interest against time instant, the other allow to partition a quantity by another fact property, associating a different formatting for each group. We've used the former to scatter the entire network traffic, while the latter to partition packet length by network protocol. It is interesting to observe that both quantity selection both partitioning is implemented via an high order system, that is the user can select *any* quantity using a declarative style, nothing is hard coded for the problem at hand. From another point of view, if another Snort log file is under study, with different properties than ours, it is possible to use this plotting mechanisms just plugging in a block that selects the quantity it is required to scatter (of course, the data series format is customizable too). As plotting back end we lie on *GnuPlot* [5].

4.1.6 *Summarizing a facts selection*

We supply one implementation for summarizing a facts selection, with a main feature to aggregate facts along some dimensions, an example can be see in tables reported in the following results section. Here the main issue is, given an user profile, build a matrix which has days on rows and experiments on columns. For each fact we assign it to a matrix cell, in order to post process each cell depending on its data content. If some data are available, the job is done by class *DatafulCellStatus*, which interpret facts measures to supply the required aggregated values.

As the case for plotting, if a user would like to aggregate respect different dimensions, it is required to specialize *DatafulCellStatus* to have an output formatted as TEXtabular environment.

4.1.7 *Putting it all together*

To make a long business short, we've implemented a set of classes whose responsibility is to interpret a log file, providing extension points to make a facts selection and using it for plotting against quantities of interest, which can be seen as a whole or as a partition; or for summarizing measures producing a TEXfile containing a tabular environment region, ready to be used as input file in an bigger document (as we're doing while writing this article). All the implementation abstracts from the problem at hand, being flexible to handle log files produced by different log producers. A final remark, all the data necessary to draw plots has been created using our implementation, as well as the automatic generation of each scatter, histogram and summary table.

4.2 *Results commentary*

In this section we'll report and explain obtained results. In order to do this, for each user profile we'll comment each experiment, reporting some scatters of interest (focusing on network traffic in particular) and histograms to show and interpreting a relation between what really happened with what it was supposed to be.

To not overwhelming the presentation, for each experiment we report first a scatter with the complete network traffic, immediately followed by a refined scatter filtering out packets without payload, while rest plots always refers to the latter situation. Moreover, we report only one plot taken from the five available for each experiment, leaving room in a dedicated subsection to observe some differences among experiment's repetitions if there exists any [2].

Summary tables reported at the end of each user profile section, do contain *average* values each corresponding to the reported quantity on the left side of each cell.

4.2.1 *On Landlady user profile*

Experiment one

In Figure 1 we report the scatter of the complete traffic, attaching comprehensive packet length on ordinates, against capturing time stamps on abscissa. We observe some background noise during all the registering time window, and we map the first spike to GMail web page request, the following traffic to performing log in, the successive two spikes to reading two mails and the final one to log out, respectively. We report in Figure 2 a "cleaned" version of the same quantity, filtering out packets such that do not contain payload. In Figure 3 we aggregate traffic information in a histogram, counting exchanged packets per second, where it is possible to recognize the same pattern described for the scatter.

Traffic is ruled by a connection oriented protocol, due to the strict majority of TCP packets as reported in Figure 4, and is quite balanced respect packet direction as reported in Figure 5.

We do not report plots about CPU and memory usage since they are quite flat and do not catch significant events outside experiment main track actions.

Experiment two

In Figure 6 and Figure 7 we report the scatter of network traffic produced in experiment two, the former covers all packets instead the latter filter out packets without a payload. We can observe the presence of background noise due to sync and ack messages, but we're not able to deduce which spike correspond to which user profile action. We suppose the initial traffic is due to YouTube video streaming, which produce many packets in the first part of the registering time window, while isn't possible to see the sending of three greetings from the scatter. On the other hand, histogram reported in Figure 8, allow to see three spikes but it isn't possible to associate them to packets for greetings or to video streaming since they appears to be too regular. Another interpretation is to associate greeting packets to three little spikes, each preceding a bigger one.

This traffic is still connection oriented as shown in Figure 9 and quite balanced from route direction point of view as shown in Figure 10. In particular is more connection oriented than traffic produced by *experiment one*, have a look at TCP percentages reported in the second column of Table 1; moreover, it is surprising to observe a similar number of packets per second (landlady checks mail very quickly).

[2] However, in the Appendix we report a Dropbox link to an archive which contains all plots.

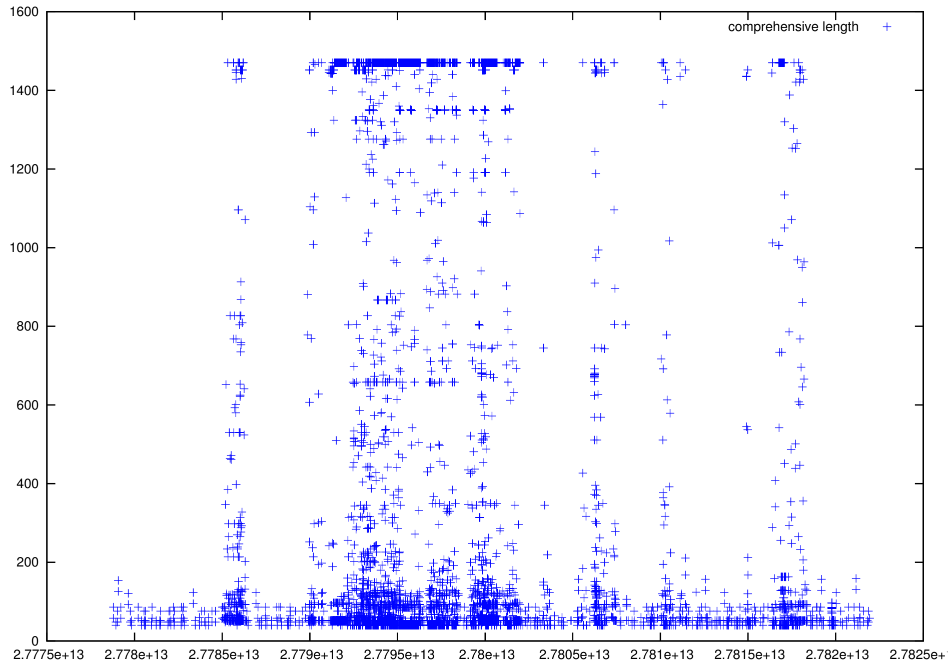

Figure 1: *Landlady* user profile, experiment one: scatter of comprehensive length against captured time stamp in nanoseconds

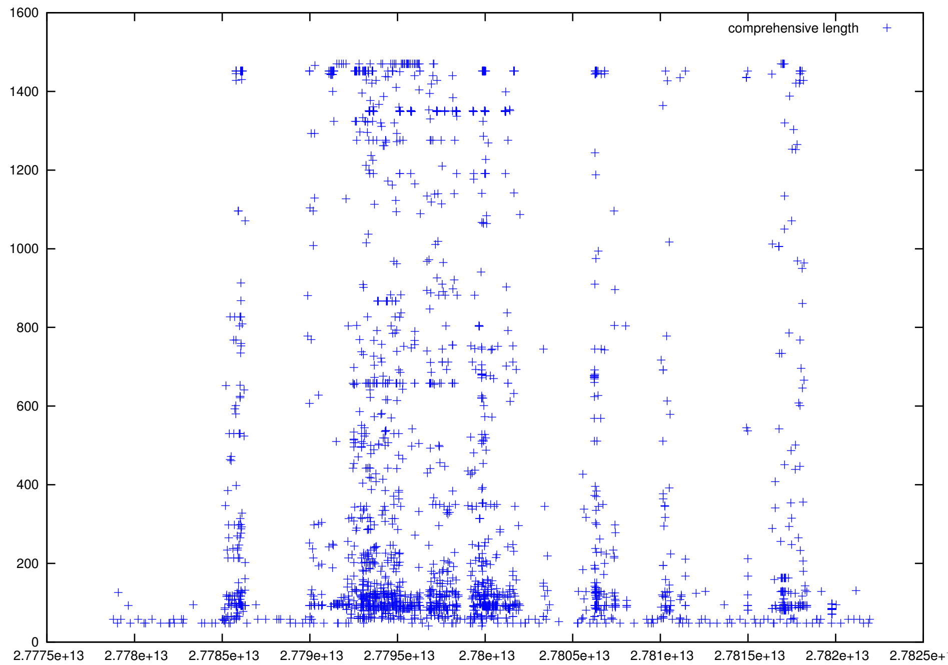

Figure 2: *Landlady* user profile, experiment one: scatter of comprehensive length, filtering out packets without payload

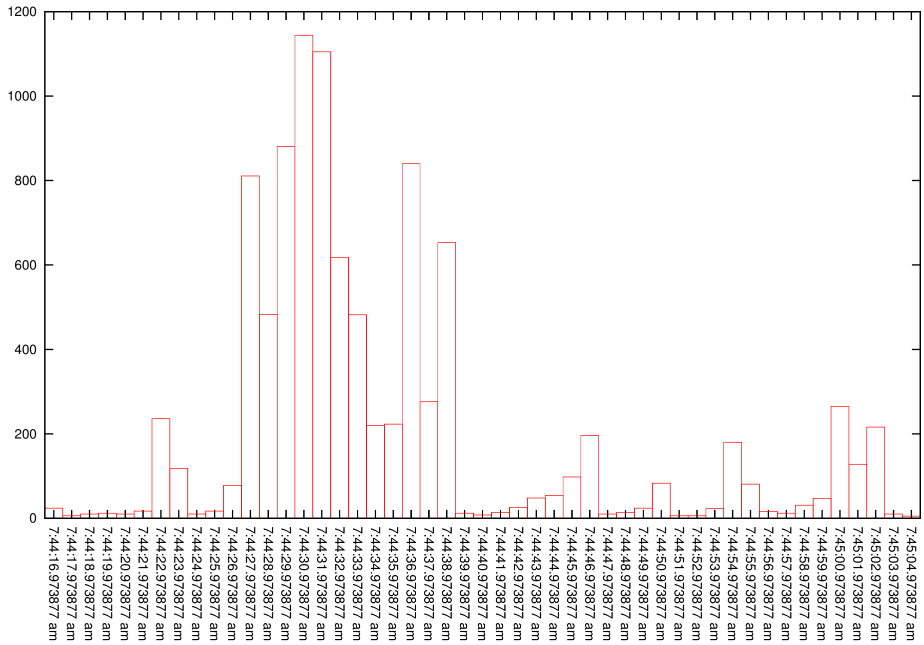

Figure 3: *Landlady* user profile, experiment one: histogram reporting number of packets in bins with length equals to 1 second.

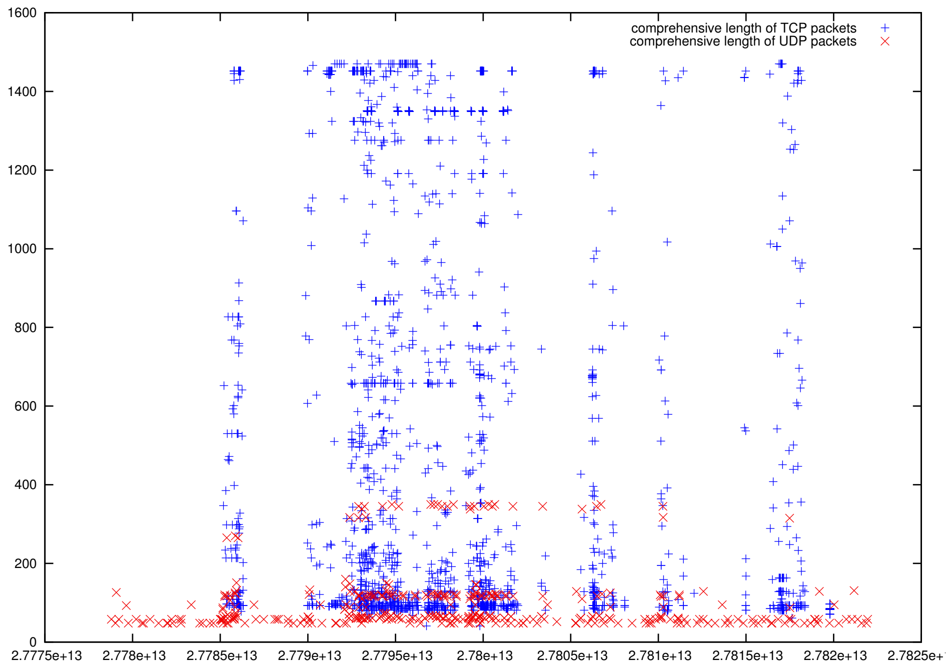

Figure 4: *Landlady* user profile, experiment one: scatter of comprehensive length, filtering out packets without payload and partitioning by protocol

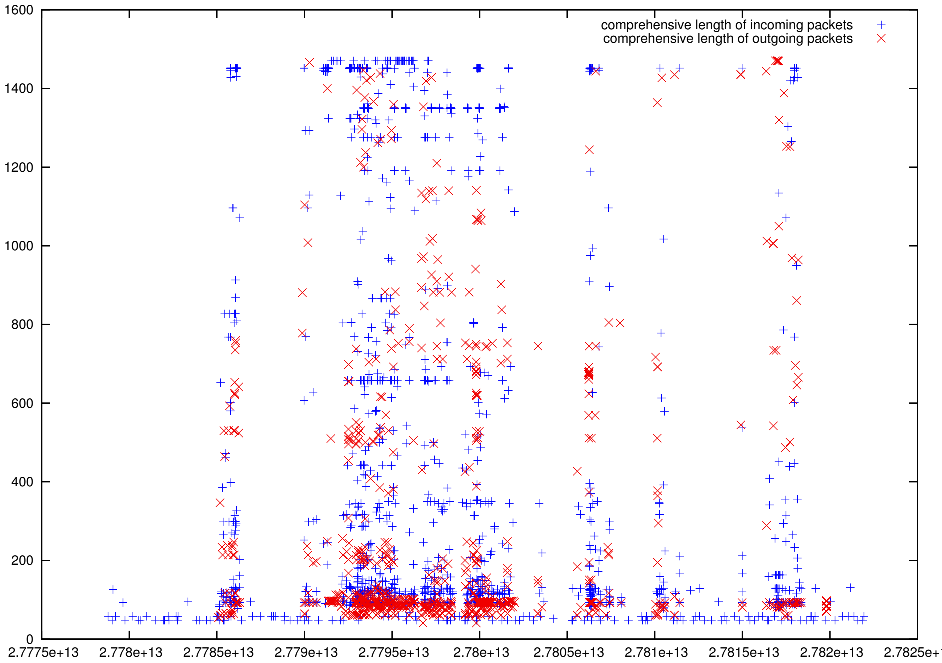

Figure 5: *Landlady* user profile, experiment one: scatter of comprehensive length, filtering out packets without payload and partitioning by traffic direction

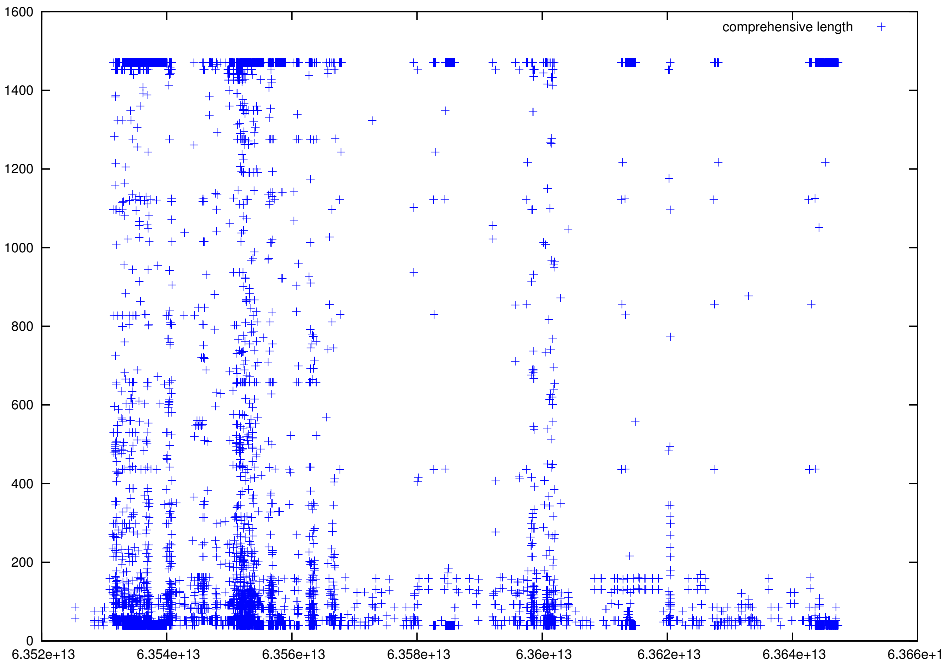

Figure 6: *Landlady* user profile, experiment two: scatter of comprehensive length against captured time stamp in nanoseconds

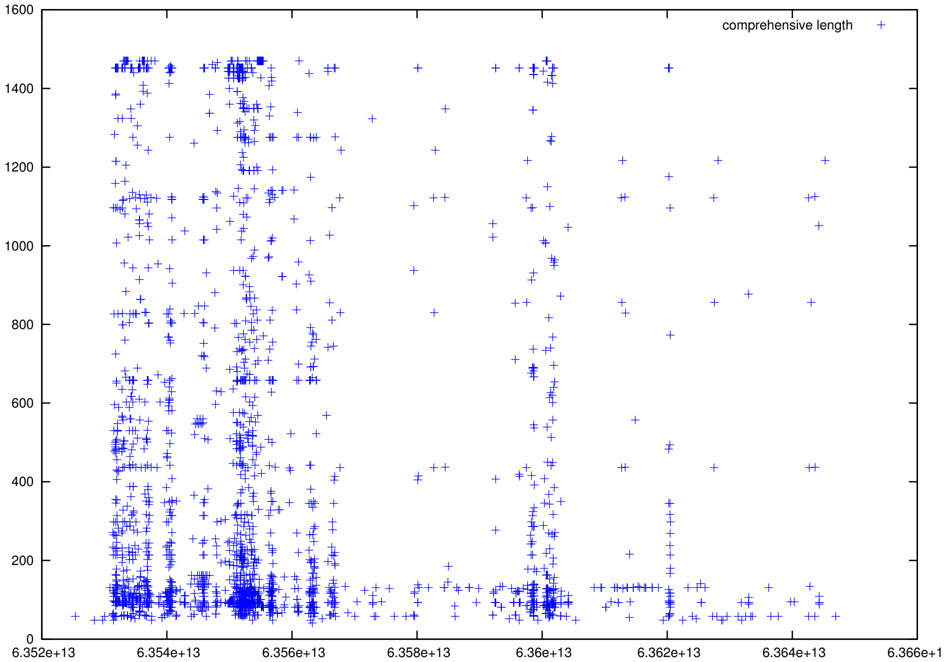

Figure 7: *Landlady* user profile, experiment two: scatter of comprehensive length against captured time stamp in nanoseconds, filtering out packets without payload

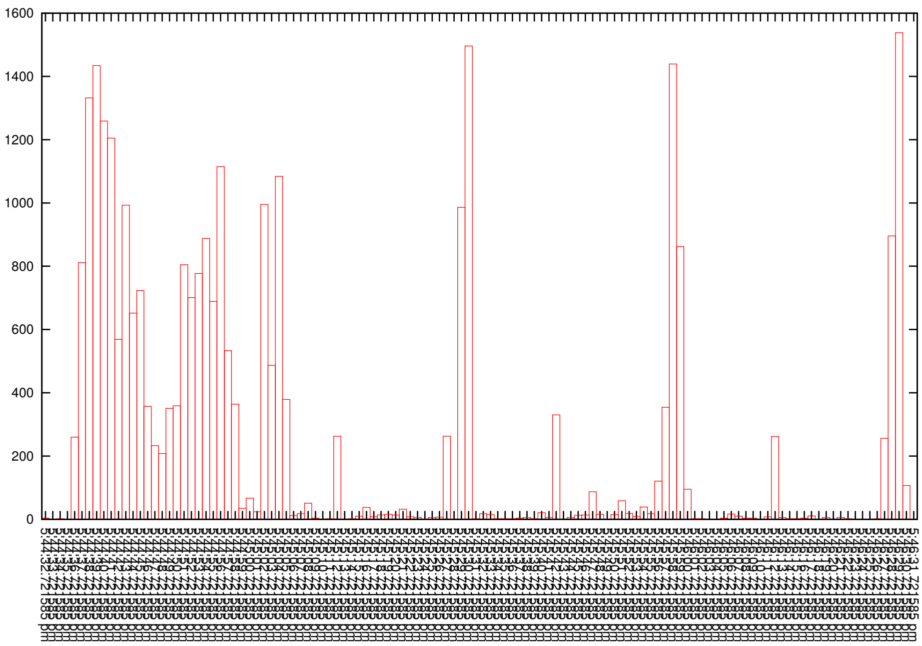

Figure 8: *Landlady* user profile, experiment two: histogram reporting number of packets in bins with length equals to 1 second.

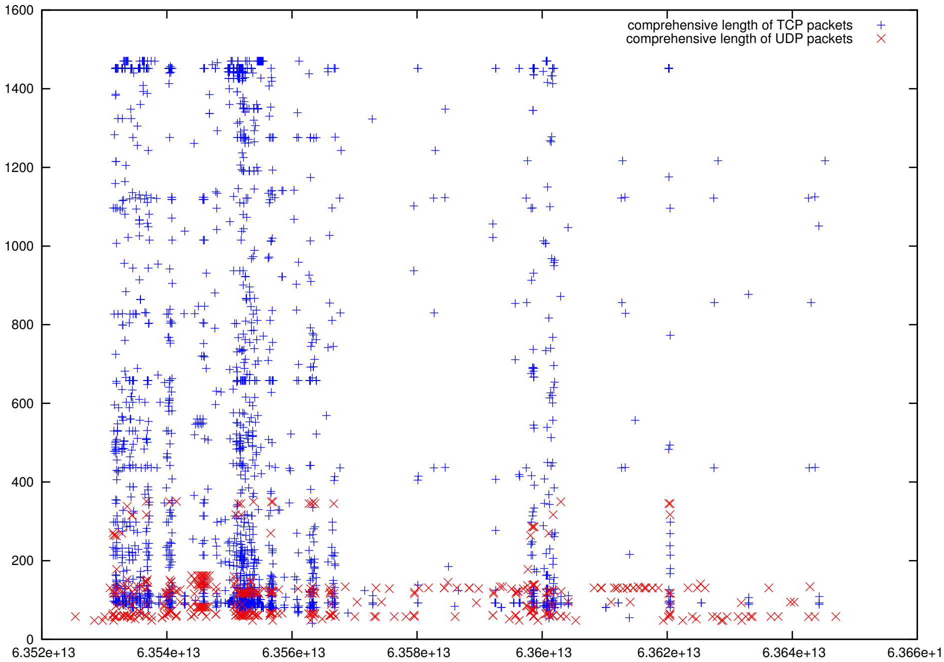

Figure 9: *Landlady* user profile, experiment two: scatter of comprehensive length, filtering out packets without payload and partitioning by protocol

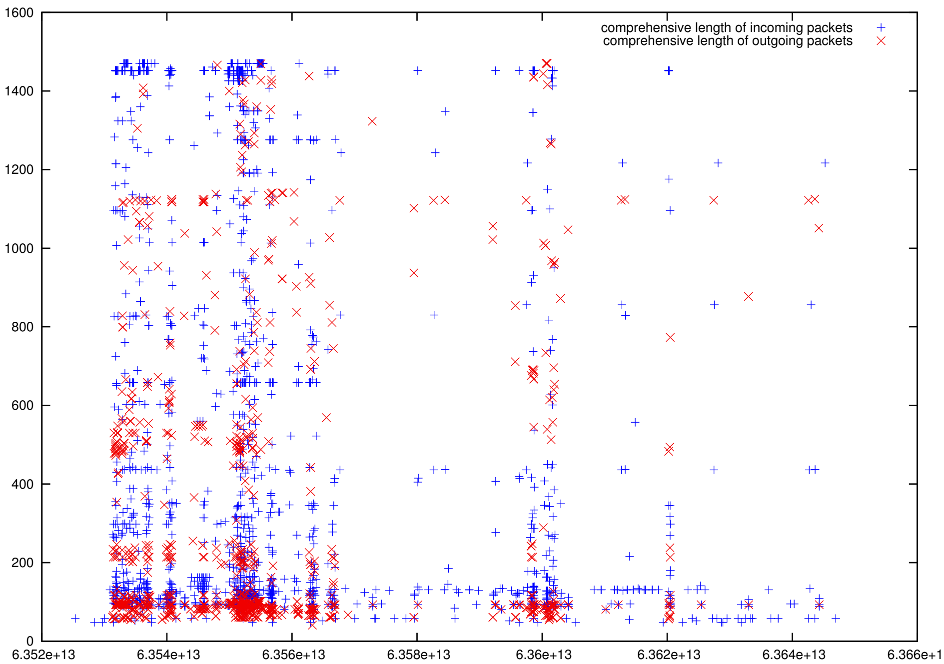

Figure 10: *Landlady* user profile, experiment two: scatter of comprehensive length, filtering out packets without payload and partitioning by traffic direction

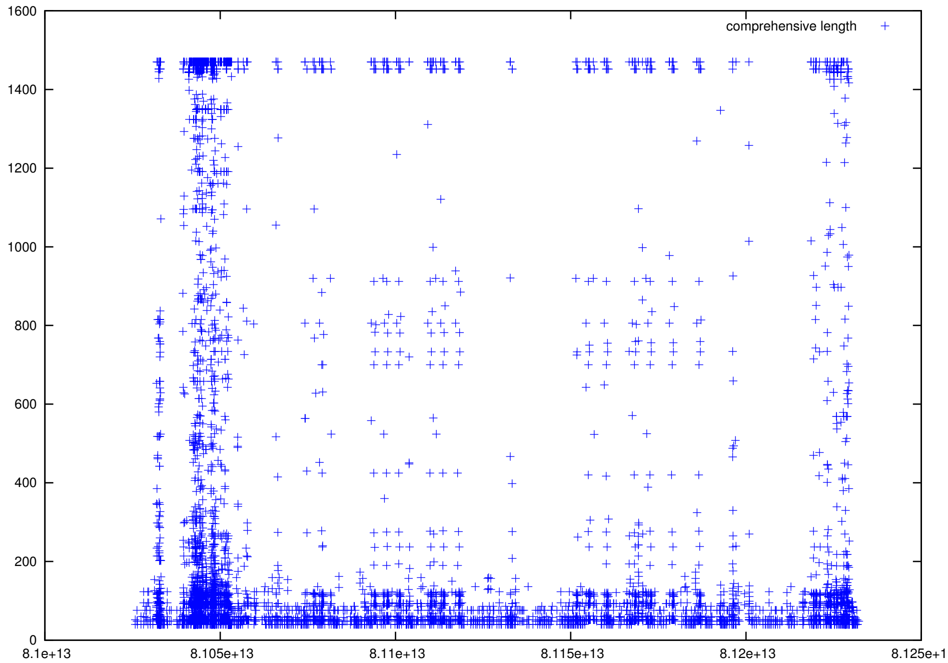

Figure 11: *Landlady* user profile, experiment three: scatter of comprehensive length against captured time stamp in nanoseconds

Experiment three

In Figure 11 and Figure 12 we report the scatter of network traffic produced in experiment three, the former covers all packets instead the latter filter out packets without a payload. Its quite clear the association with user actions: the first intense spike is for GMail log in, then there are two "blocks" with sparse traffic, each corresponding to one draft preparation and the final spike is one mail sending. It is quite surprising that packet distribution is quite uniform after the initial log in, as histogram reports in Figure 13 using bins of length 5 seconds each.

This traffic is again connection oriented as shown in Figure 14 and length per packet in bytes is lower than those registered for *experiment one* and *two* and the number of packets per second is remarkably smaller due to a different sequence of actions, as reported in the third column of Table 1.

Repeating experiments issues

We observe that experiments repeats with the same patterns, as proved by scatters, both in traffic distribution and by protocol partitioning. Looking at Table 1 we observe concordance among averaged quantities without any remarkable outsider. An odd case for *experiment three* is reported in Figure 15, where we observe a quantity of exchanged packets not present in other experiment's repetitions.

Measures' quality

Network traffic is quite accurate, as in scatters we've to resort to nanoseconds in order to see events. On the other hand, CPU and memory consumption didn't have the same accuracy, since *vmstat* drops lines with a maximum rate of 1 second: for *experiment one* it is useless since the entire time window duration is lower than a minute.

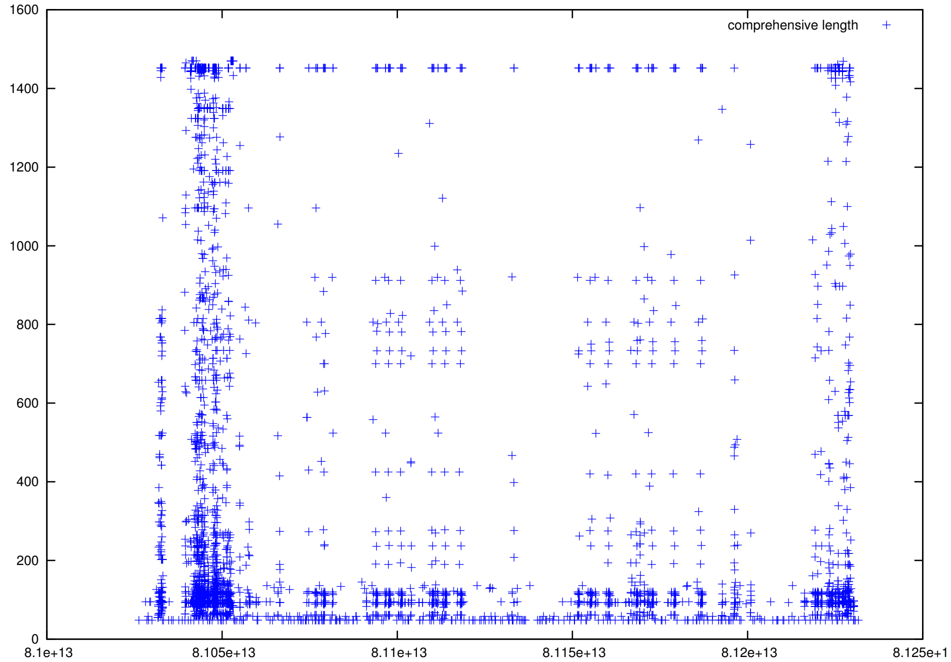

Figure 12: *Landlady* user profile, experiment three: scatter of comprehensive length against captured time stamp in nanoseconds, filtering out packets without payload

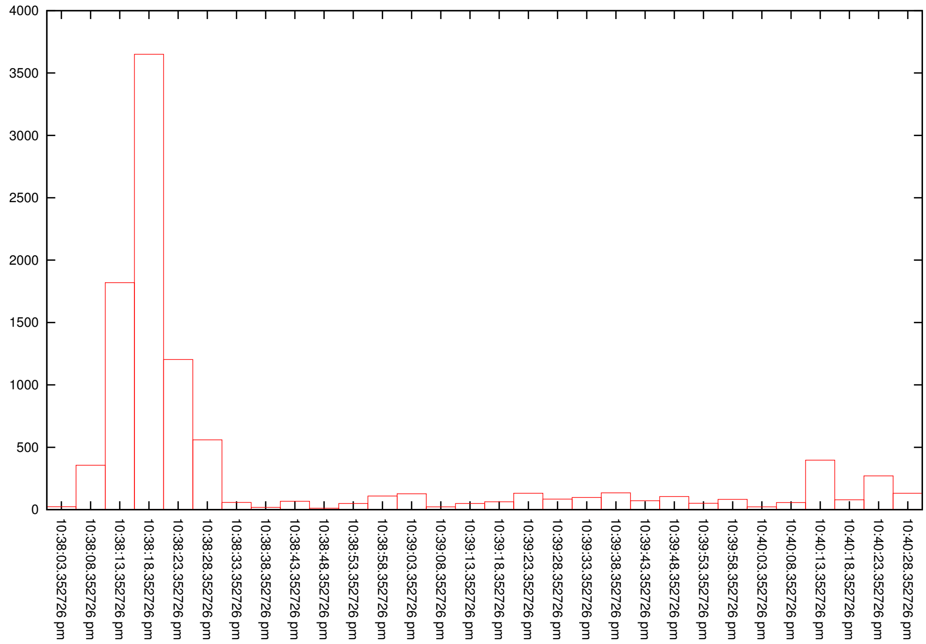

Figure 13: *Landlady* user profile, experiment three: histogram reporting number of packets in bins with length equals to 5 seconds each.

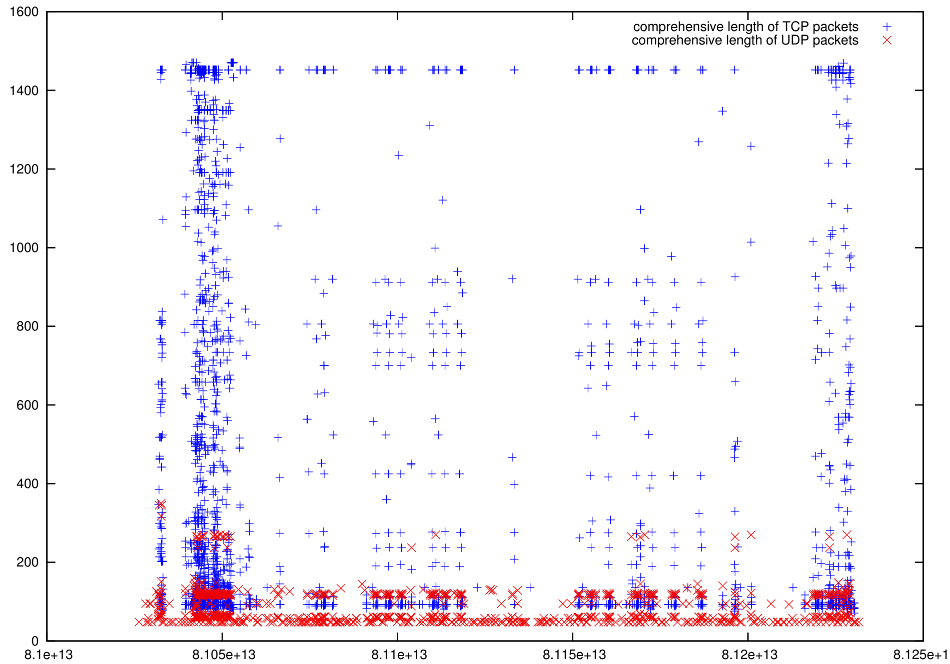

Figure 14: *Landlady* user profile, experiment three: scatter of comprehensive length, filtering out packets without payload and partitioning by protocol

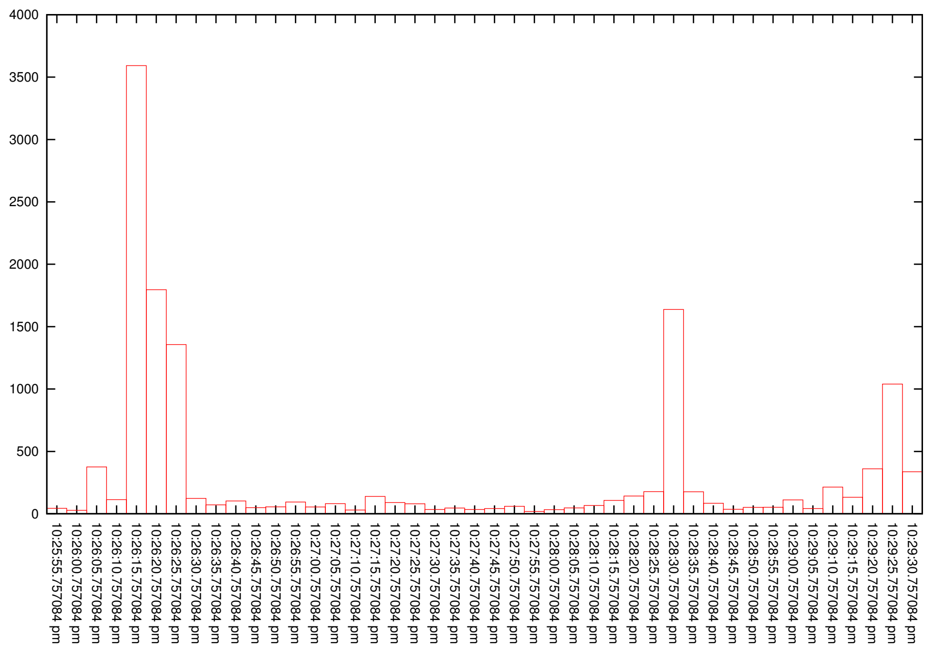

Figure 15: *Landlady* user profile: histogram reporting a bad case, with a spike not present in all the remaining histograms about experiment three.

	Experiment 1		Experiment 2		Experiment 3	
Day 1	cpu idle	79.20 %	cpu idle	75.19 %	cpu idle	89.59 %
	user code	16.36 %	user code	18.84 %	user code	7.98 %
	kernel code	3.79 %	kernel code	6.03 %	kernel code	2.24 %
	free memory	4.218 GB	free memory	0.189 GB	free memory	0.728 GB
	num packets	10276	num packets	28412	num packets	10812
	in packets	59.93 %	in packets	59.25 %	in packets	57.39 %
	payloaded	30.78 %	payloaded	12.34 %	payloaded	31.49 %
	TCP	93.99 %	TCP	97.54 %	TCP	88.15 %
	tot weight	6.64 MB	tot weight	21.82 MB	tot weight	5.92 MB
	weight/packet	677.13 B	weight/packet	805.28 B	weight/packet	574.46 B
	tot time	40.65″	tot time	2.08′	tot time	3.85′
	packets/sec	250.63	packets/sec	227.30	packets/sec	46.81
Day 2	cpu idle	80.60 %	cpu idle	77.91 %	cpu idle	83.31 %
	user code	15.77 %	user code	16.96 %	user code	13.53 %
	kernel code	3.58 %	kernel code	5.18 %	kernel code	3.19 %
	free memory	4.198 GB	free memory	0.257 GB	free memory	0.602 GB
	num packets	9290	num packets	30618	num packets	13358
	in packets	59.30 %	in packets	59.06 %	in packets	59.13 %
	payloaded	28.98 %	payloaded	11.91 %	payloaded	30.89 %
	TCP	93.74 %	TCP	97.93 %	TCP	90.58 %
	tot weight	5.93 MB	tot weight	23.66 MB	tot weight	7.96 MB
	weight/packet	669.64 B	weight/packet	810.26 B	weight/packet	624.47 B
	tot time	43.15″	tot time	2.13′	tot time	3.66′
	packets/sec	211.14	packets/sec	239.20	packets/sec	60.72
Day 3	cpu idle	79.76 %	cpu idle	77.09 %	cpu idle	82.87 %
	user code	16.77 %	user code	17.61 %	user code	13.93 %
	kernel code	3.55 %	kernel code	5.35 %	kernel code	3.23 %
	free memory	4.176 GB	free memory	0.247 GB	free memory	0.625 GB
	num packets	9887	num packets	28384	num packets	12030
	in packets	59.62 %	in packets	61.15 %	in packets	58.40 %
	payloaded	32.12 %	payloaded	11.01 %	payloaded	34.06 %
	TCP	94.50 %	TCP	97.91 %	TCP	89.73 %
	tot weight	6.28 MB	tot weight	22.83 MB	tot weight	6.57 MB
	weight/packet	666.30 B	weight/packet	843.38 B	weight/packet	572.36 B
	tot time	48.64″	tot time	2.03′	tot time	3.44′
	packets/sec	201.78	packets/sec	232.66	packets/sec	58.12
Day 4	cpu idle	80.13 %	cpu idle	75.85 %	cpu idle	86.89 %
	user code	16.33 %	user code	18.17 %	user code	10.68 %
	kernel code	3.63 %	kernel code	5.64 %	kernel code	2.76 %
	free memory	4.153 GB	free memory	0.232 GB	free memory	0.642 GB
	num packets	10768	num packets	29421	num packets	10206
	in packets	59.39 %	in packets	59.75 %	in packets	59.15 %
	payloaded	30.51 %	payloaded	10.25 %	payloaded	34.09 %
	TCP	95.31 %	TCP	98.28 %	TCP	91.99 %
	tot weight	7.01 MB	tot weight	23.26 MB	tot weight	5.99 MB
	weight/packet	682.67 B	weight/packet	828.85 B	weight/packet	615.33 B
	tot time	43.56″	tot time	2.05′	tot time	3.02′
	packets/sec	244.73	packets/sec	239.20	packets/sec	56.08
Day 5	cpu idle	80.10 %	cpu idle	79.60 %	cpu idle	82.13 %
	user code	16.45 %	user code	15.78 %	user code	14.69 %
	kernel code	3.60 %	kernel code	4.50 %	kernel code	3.14 %
	free memory	4.152 GB	free memory	0.287 GB	free memory	0.599 GB
	num packets	9807	num packets	29682	num packets	9903
	in packets	59.30 %	in packets	59.88 %	in packets	60.21 %
	payloaded	28.12 %	payloaded	11.19 %	payloaded	35.70 %
	TCP	94.08 %	TCP	98.28 %	TCP	93.45 %
	tot weight	6.25 MB	tot weight	23.44 MB	tot weight	6.18 MB
	weight/packet	668.32 B	weight/packet	827.96 B	weight/packet	654.65 B
	tot time	46.84″	tot time	1.99′	tot time	2.47′
	packets/sec	208.66	packets/sec	247.35	packets/sec	66.46

Table 1: Summary table for *landlady* user profile

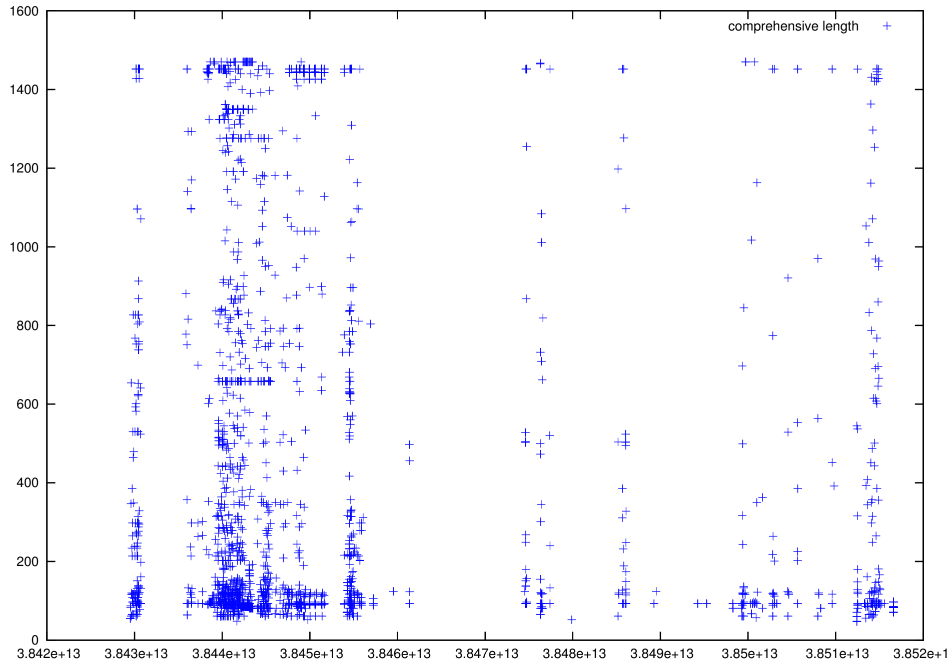

Figure 16: *ComputerScientist* user profile, experiment one: scatter of comprehensive length, filtering out packets without payload

4.2.2 *On Computer scientist user profile*

For this profile we do not report many plots about network traffic, instead we move focus on CPU consumption in order to take into account the programming activity and Google Hangout video call. The complete set of plots is however available in the distributed package.

Experiment one

In Figure 16 we report the scatter of the complete traffic after removing background noise filtering protocol control messages: we map the first spike to GMail web page request, the following traffic to performing log in (compare a similar behavior with Figure 2), two thin spikes to reading two mails and the final one to attaching the compiled draft and sending mail, respectively. In Figure 17 we aggregate traffic information in a histogram, counting exchanged packets per second, where it is possible to recognize the same pattern described for the scatter: poor network activity.

Traffic is ruled by a connection oriented protocol, due to the strict majority of TCP packets as reported in Figure 18, this confirm the fact obtained from *experiment one* for *landlady* profile that Google web mail is totally connection based, as we would expect.

Figure 19 shows the two TEXcompilations, reporting two gaps of CPU *idle* percentage in the first half of histogram.

Experiment two

For this experiment we leave the plot about comprehensive network traffic, paying attention to some details. In Figure 20 we report a scatter of the traffic of pay-loaded packets, partitioned by route direction: it is interesting that, since this traffic is generated only by textual instant messages, the incoming packets have a smaller length than outgoing packets, always respecting the assumption that both peers sent at most 10 words in each message.

Another interesting comparison arise by Figure 21 and Figure 22: those two histograms seems to mirror each other, in the sense that when there is network activity the fluid dynamic simulation is running and, on the contrary, when the simulation is suspended, no network traffic is present.

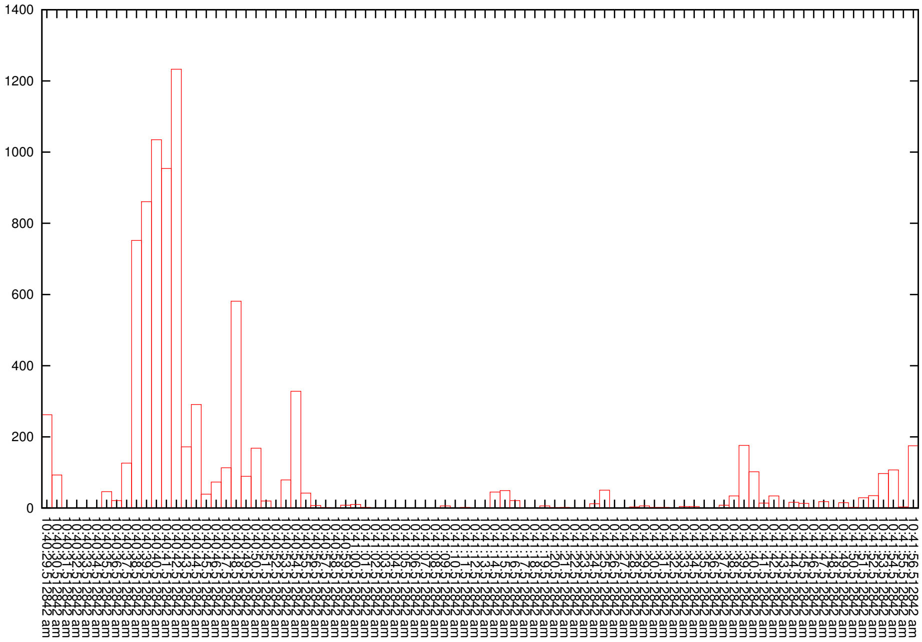

Figure 17: *ComputerScientist* user profile, experiment one: histogram reporting number of packets in bins with length equals to 1 second.

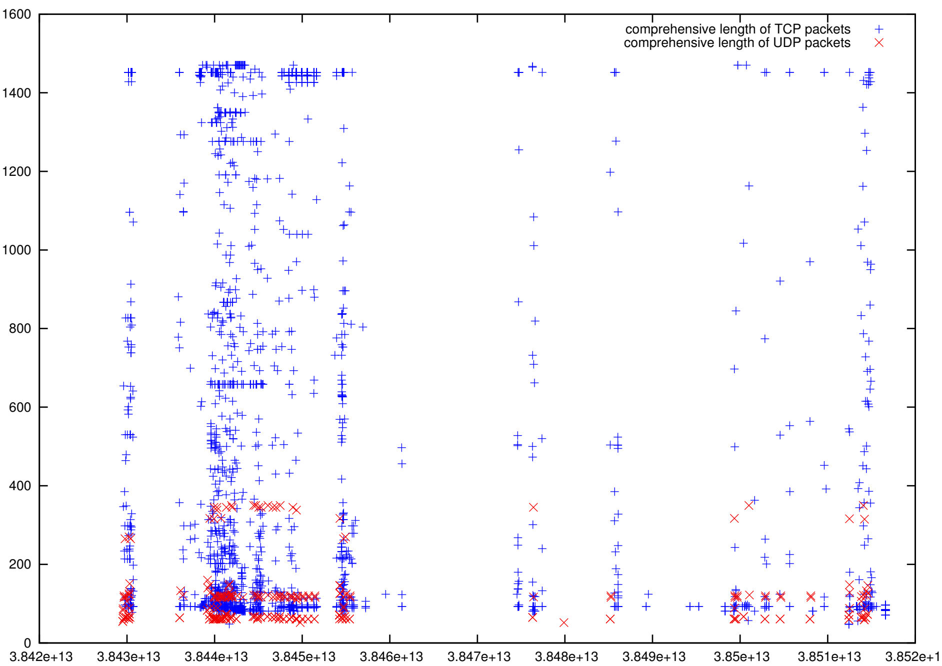

Figure 18: *ComputerScientist* user profile, experiment one: scatter of comprehensive length, filtering out packets without payload and partitioning by protocol

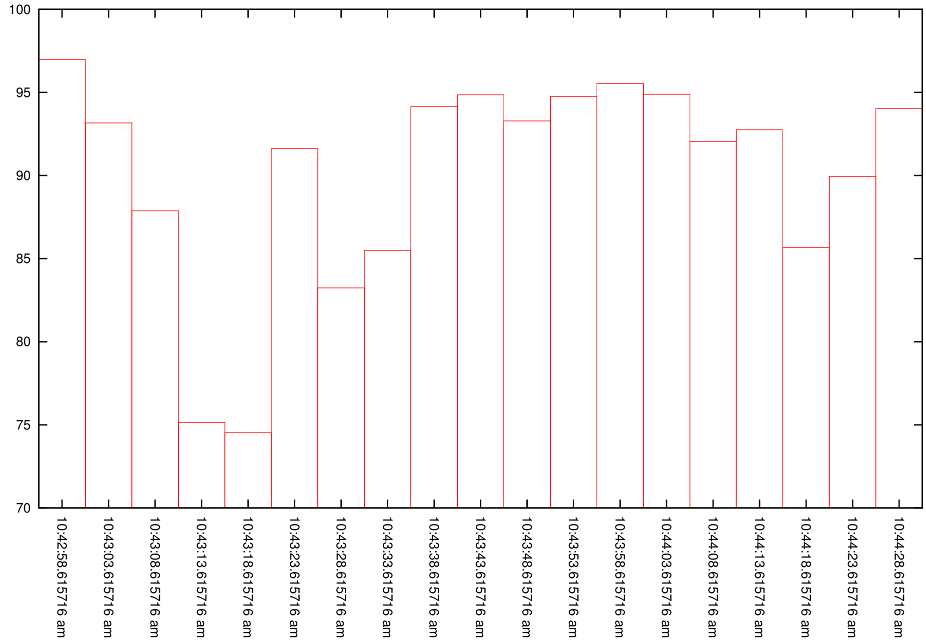

Figure 19: *ComputerScientist* user profile, experiment one: histogram about CPU consumption against bins of length 5 seconds each.

We do not report the scatter partitioned by protocol since, using Google web mail interface, the majority of it belongs again to TCP.

Looking at Table 2, we prove that both peer played according laws, since the percentage of incoming packets equals that of outgoing ones. Due to textual instant message exchanging only, the number of packets per seconds is very small, about 10 in average.

Experiment three

In Figure 23 we report the scatter of network traffic, filtering pay loaded packets: we see two "partitions", a lower one possibly related to control packets, and a higher one possibly related to video frames. As we would expect, Table 2 confirm that here the traffic is datagram oriented, with a small percentage of TCP packets. In Figure 24 we report the histograms of packets per time unit and we cannot extract a pattern from it: it is possible only to associate a period where the traffic is quite stable to the running of fluid dynamics simulation, as shown in Figure 25 too.

Repeating experiments issues

We observe that *experiments one* and *two* repeats with the same patterns both in traffic distribution and by protocol partitioning. Also CPU consumption seems to repeat, although with more variance. Looking at Table 2 we observe concordance among averaged quantities without any remarkable outsider. It is useful to observe that for *experiment three*, even though quantities repeat, the patterns of traffic network relative to different days behaves like independent random variables, it cannot be possible to extract common aspects from those because of the non repeatable nature of movements by peers in the video call, assuming that both camera never catch a "fixed" image.

Measures' quality

CPU consumption, always subject to vmstat maximum rate of 1 second, has an acceptable quality since the registering time window is quite large: this allow to analyze it, contrary of what happened in *experiment one* for *landlady* user profile, which has a smaller time window.

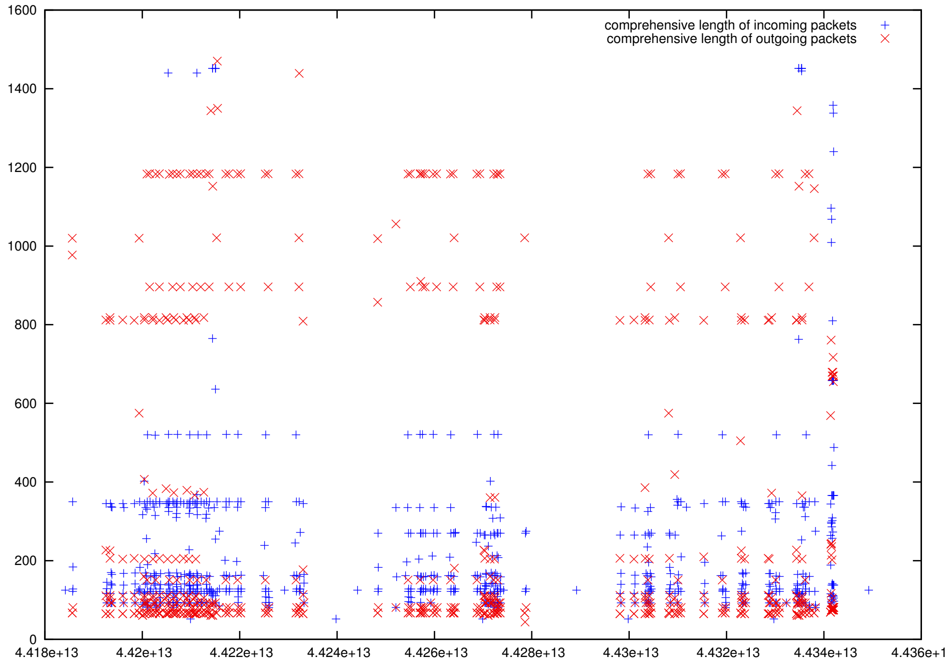

Figure 20: *ComputerScientist* user profile, experiment two: scatter of comprehensive length, filtering out packets without payload and partitioning by route direction

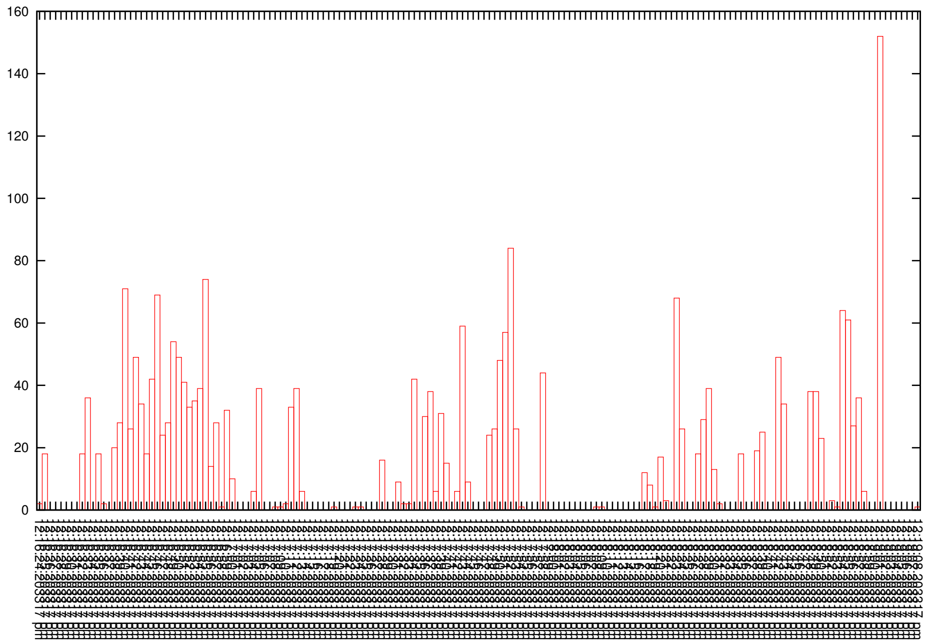

Figure 21: *ComputerScientist* user profile, experiment two: histogram reporting number of packets in bins with length equals to 1 second.

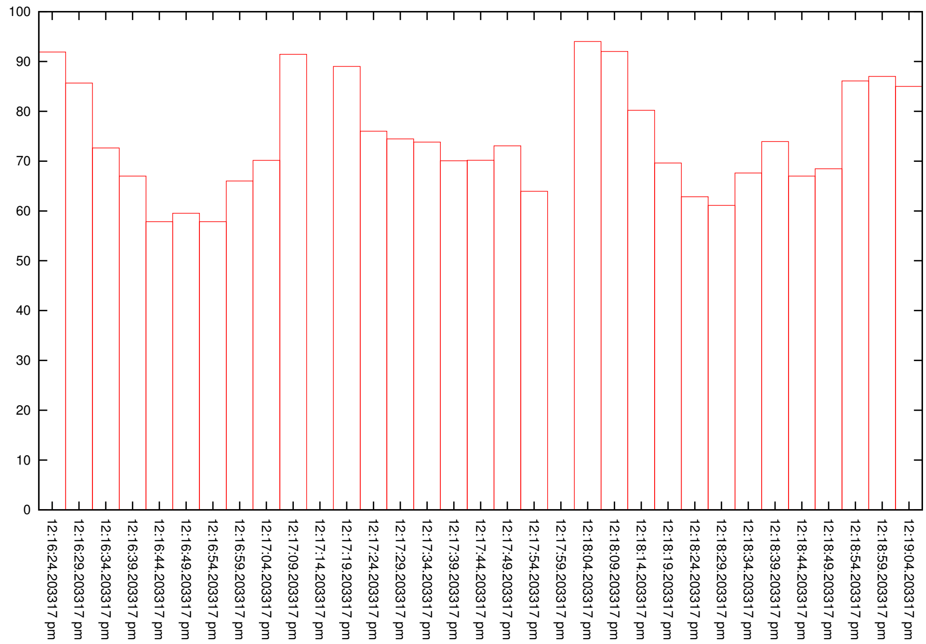

Figure 22: *ComputerScientist* user profile, experiment two: histogram about CPU consumption against bins of length 5 seconds each.

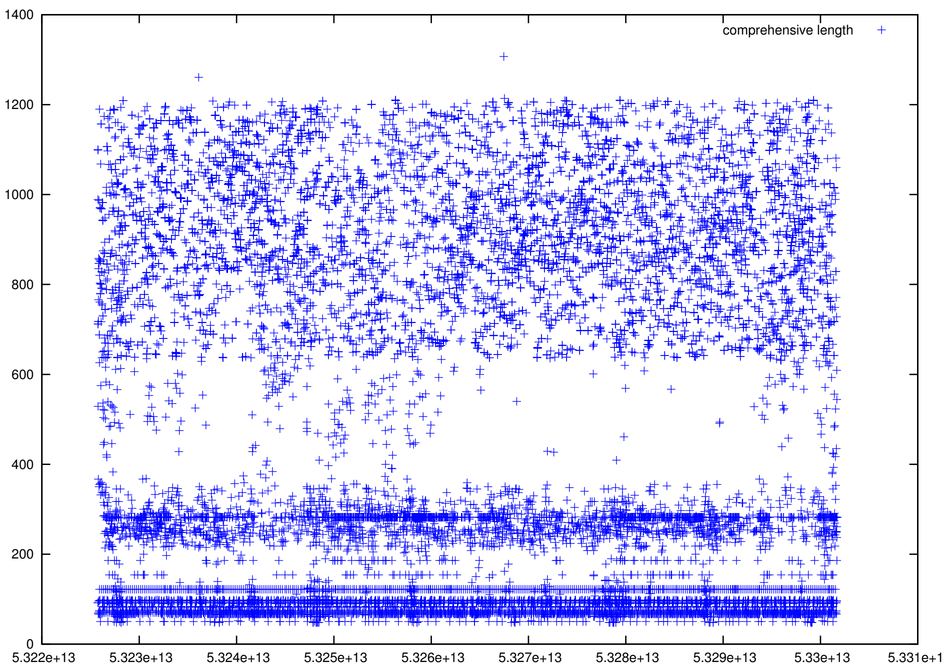

Figure 23: *ComputerScientist* user profile, experiment three: scatter of comprehensive length, filtering out packets without payload

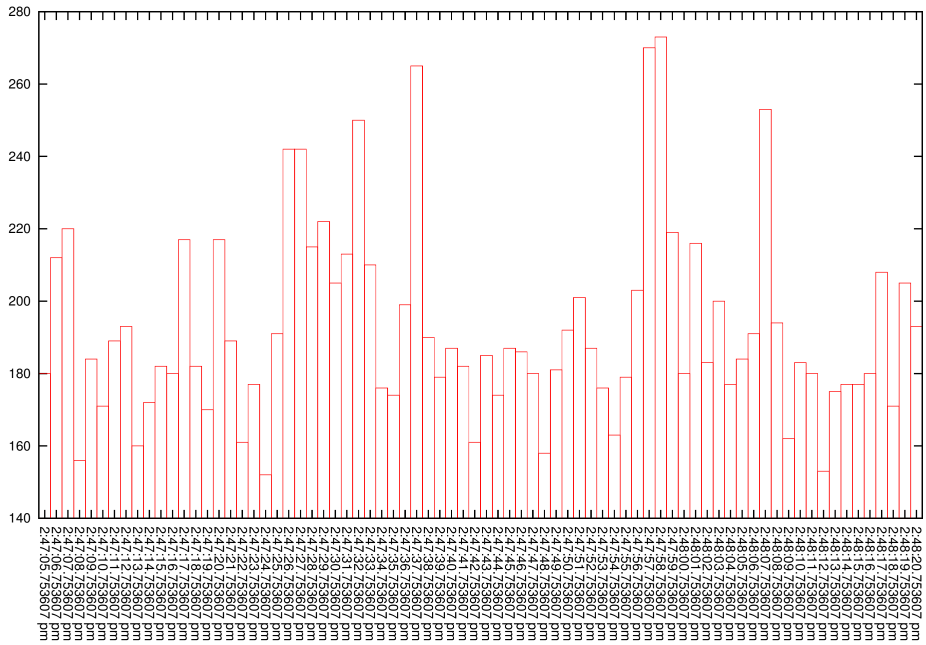

Figure 24: *ComputerScientist* user profile, experiment three: histogram reporting number of packets in bins with length equals to 1 second.

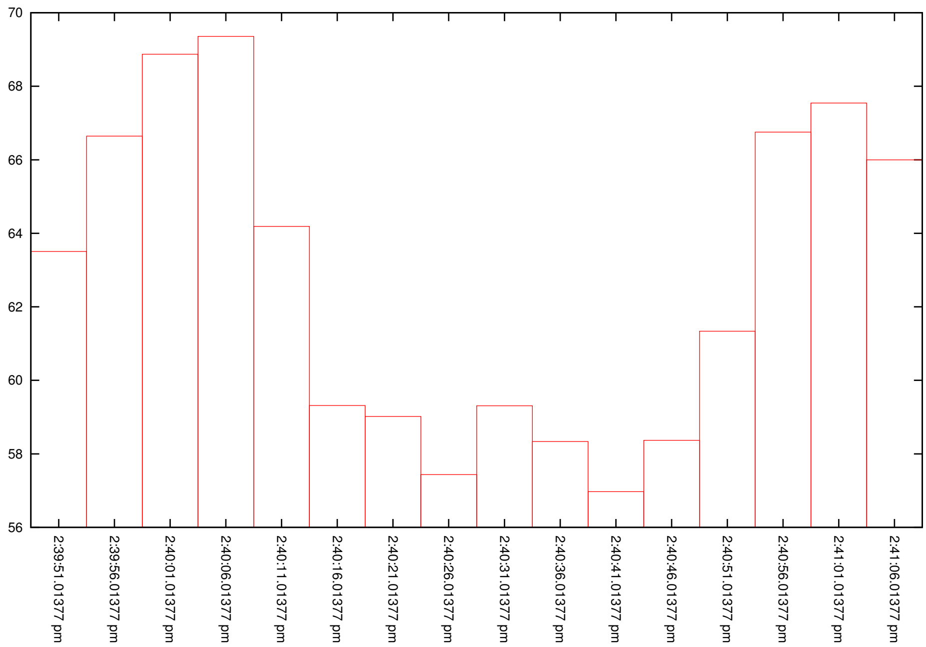

Figure 25: *ComputerScientist* user profile, experiment three: histogram about CPU consumption against bins of length 5 seconds each.

	Experiment 1		Experiment 2		Experiment 3	
	cpu idle	87.87 %	cpu idle	78.46 %	cpu idle	60.00 %
	user code	9.90 %	user code	18.26 %	user code	33.03 %
	kernel code	2.35 %	kernel code	3.17 %	kernel code	7.04 %
	free memory	4.211 GB	free memory	3.472 GB	free memory	3.200 GB
	num packets	9628	num packets	1688	num packets	14666
Day 1	in packets	60.27 %	in packets	51.72 %	in packets	39.70 %
	payloaded	31.56 %	payloaded	51.72 %	payloaded	97.16 %
	TCP	96.80 %	TCP	82.94 %	TCP	5.43 %
	tot weight	6.62 MB	tot weight	0.47 MB	tot weight	6.94 MB
	weight/packet	720.89 B	weight/packet	290.49 B	weight/packet	496.46 B
	tot time	1.59′	tot time	3.00′	tot time	1.27′
	packets/sec	100.29	packets/sec	9.33	packets/sec	192.97
	cpu idle	90.52 %	cpu idle	75.33 %	cpu idle	62.71 %
	user code	7.35 %	user code	21.67 %	user code	30.14 %
	kernel code	2.26 %	kernel code	3.08 %	kernel code	7.09 %
	free memory	4.227 GB	free memory	3.444 GB	free memory	3.266 GB
	num packets	8499	num packets	2065	num packets	14545
Day 2	in packets	58.87 %	in packets	51.91 %	in packets	37.41 %
	payloaded	30.60 %	payloaded	52.98 %	payloaded	97.01 %
	TCP	96.56 %	TCP	81.69 %	TCP	5.71 %
	tot weight	5.62 MB	tot weight	0.60 MB	tot weight	7.53 MB
	weight/packet	693.95 B	weight/packet	302.90 B	weight/packet	542.84 B
	tot time	1.46′	tot time	2.68′	tot time	1.30′
	packets/sec	96.58	packets/sec	12.75	packets/sec	186.47
	cpu idle	87.25 %	cpu idle	75.76 %	cpu idle	62.77 %
	user code	10.13 %	user code	21.30 %	user code	30.40 %
	kernel code	2.70 %	kernel code	2.87 %	kernel code	6.85 %
	free memory	4.205 GB	free memory	3.445 GB	free memory	3.250 GB
	num packets	10282	num packets	1875	num packets	14639
Day 3	in packets	59.62 %	in packets	52.43 %	in packets	43.92 %
	payloaded	32.20 %	payloaded	52.48 %	payloaded	96.65 %
	TCP	96.72 %	TCP	81.97 %	TCP	6.37 %
	tot weight	7.06 MB	tot weight	0.54 MB	tot weight	7.33 MB
	weight/packet	720.23 B	weight/packet	303.94 B	weight/packet	524.77 B
	tot time	1.55′	tot time	2.88′	tot time	1.26′
	packets/sec	110.56	packets/sec	10.78	packets/sec	192.62
	cpu idle	86.96 %	cpu idle	75.23 %	cpu idle	59.88 %
	user code	10.36 %	user code	21.73 %	user code	33.18 %
	kernel code	2.96 %	kernel code	3.07 %	kernel code	7.01 %
	free memory	4.189 GB	free memory	3.433 GB	free memory	3.229 GB
	num packets	9535	num packets	1394	num packets	15180
Day 4	in packets	59.73 %	in packets	52.37 %	in packets	46.49 %
	payloaded	29.87 %	payloaded	52.87 %	payloaded	97.67 %
	TCP	96.57 %	TCP	81.99 %	TCP	4.51 %
	tot weight	6.60 MB	tot weight	0.40 MB	tot weight	7.20 MB
	weight/packet	726.16 B	weight/packet	300.38 B	weight/packet	497.15 B
	tot time	1.44′	tot time	2.26′	tot time	1.27′
	packets/sec	109.60	packets/sec	10.25	packets/sec	197.14
	cpu idle	89.59 %	cpu idle	70.90 %	cpu idle	59.82 %
	user code	8.15 %	user code	26.25 %	user code	32.98 %
	kernel code	2.35 %	kernel code	2.90 %	kernel code	7.15 %
	free memory	4.174 GB	free memory	3.433 GB	free memory	3.204 GB
	num packets	9125	num packets	2451	num packets	14623
Day 5	in packets	59.04 %	in packets	52.22 %	in packets	46.29 %
	payloaded	31.56 %	payloaded	52.47 %	payloaded	97.26 %
	TCP	96.49 %	TCP	82.50 %	TCP	5.28 %
	tot weight	6.04 MB	tot weight	0.72 MB	tot weight	7.05 MB
	weight/packet	693.85 B	weight/packet	305.90 B	weight/packet	505.76 B
	tot time	1.62′	tot time	2.75′	tot time	1.27′
	packets/sec	93.11	packets/sec	14.85	packets/sec	192.41

Table 2: Summary table for *computer scientist* user profile

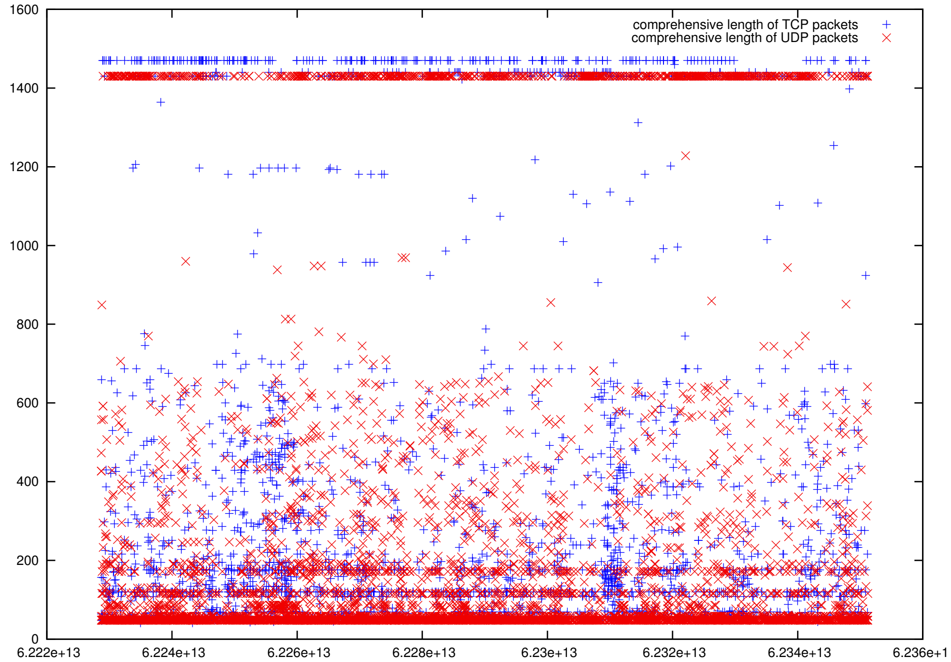

Figure 26: *FSF Supporter* user profile, experiment two: scatter of packet comprehensive length, filtering pay loaded packets and partitioning by network protocol

4.2.3 *On FSF supporter user profile*

Repeating experiments issues

Even though experiments had been repeated in the same conditions, for this user profile hadn't been possible to have repeatable experiments. We justify it by the connection less nature of experiments and we couldn't do anything more to have some recurring pattern, all dependends by world wide users' requests. Hence we make a few observation without proceeding by division as did until now.

In Figure 26 we report the scatter about Ubuntu iso image, against the one in Figure 27 relative to Xubuntu iso image: there's a huge traffic difference, the former use both connected and datagram protocol packets, while the latter prefers UDP packets. In Figure 28 we report an histogram that aggregate packet lengths in bins of 5 seconds each for *experiment one*. According Table 3, Ubuntu image has been the most requested one.

Measures' quality

Quality of traffic analysis is again accurate, producing Snort binary log files about 20 MB each for 2' of observation. An interesting aspect not considered in this analysis should be the number of disk accesses for fetching data. We discard measures about CPU consumption since not relevant for the experiments at hand, nothing stress the CPU by an interesting computation effort.

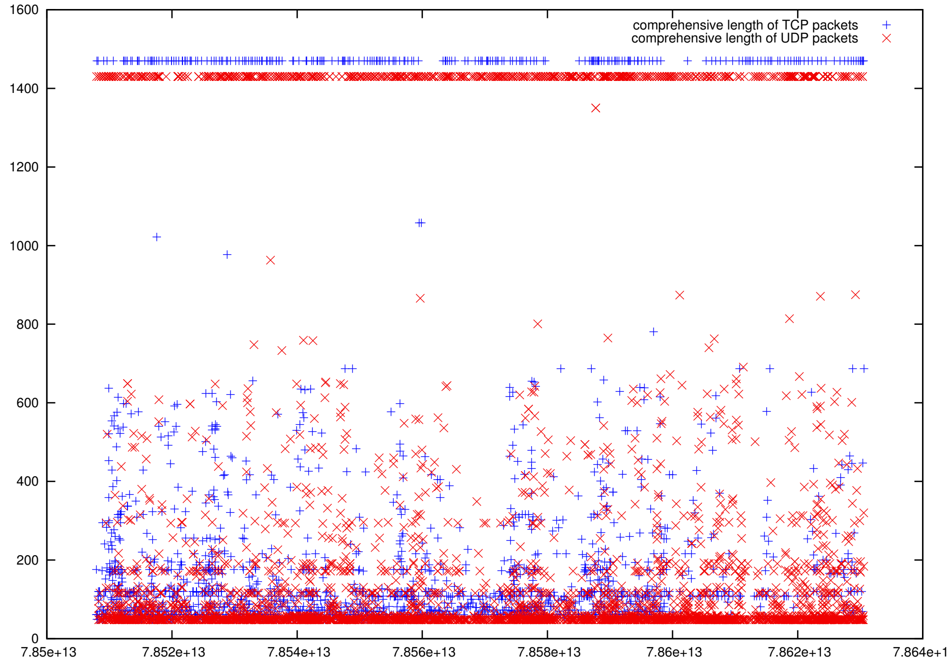

Figure 27: *FSF Supporter* user profile, experiment three: scatter of packet comprehensive length, filtering pay loaded packets and partitioning by network protocol

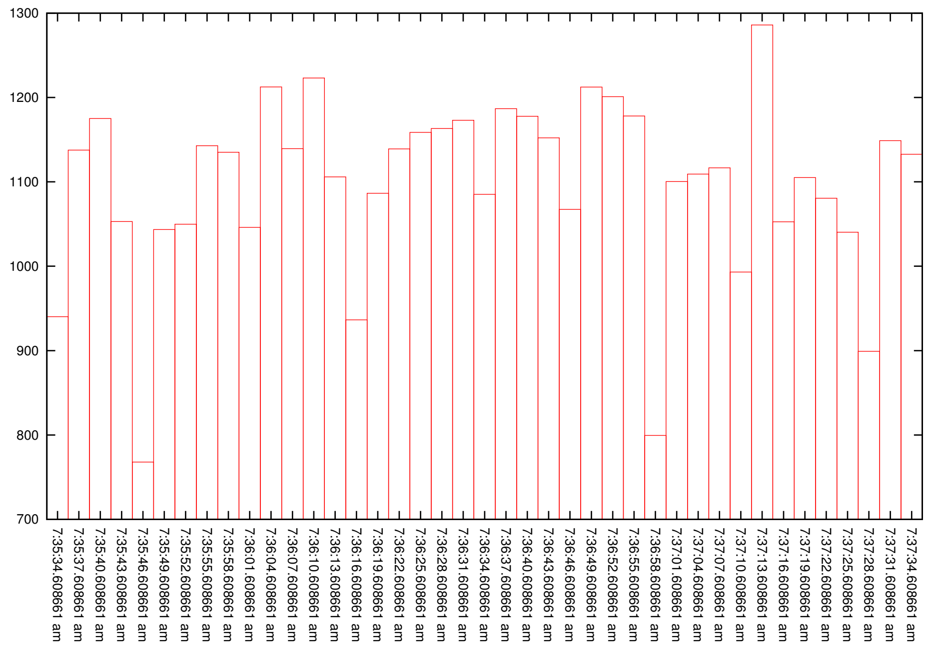

Figure 28: *ComputerScientist* user profile, experiment one: histogram about packet comprehensive length against bins of length 5 seconds each.

	Experiment 1		Experiment 2		Experiment 3	
Day 1	cpu idle	97.96 %	cpu idle	97.13 %	cpu idle	98.57 %
	user code	0.84 %	user code	1.02 %	user code	0.47 %
	kernel code	1.23 %	kernel code	1.86 %	kernel code	0.82 %
	free memory	5.138 GB	free memory	0.922 GB	free memory	0.897 GB
	num packets	14811	num packets	29737	num packets	4799
	in packets	46.27 %	in packets	43.75 %	in packets	44.41 %
	payloaded	72.98 %	payloaded	46.96 %	payloaded	57.05 %
	TCP	32.86 %	TCP	64.86 %	TCP	50.47 %
	tot weight	6.12 MB	tot weight	13.09 MB	tot weight	1.27 MB
	weight/packet	433.20 B	weight/packet	461.69 B	weight/packet	278.40 B
	tot time	2.05′	tot time	2.05′	tot time	2.12′
	packets/sec	120.41	packets/sec	239.81	packets/sec	37.79
Day 2	cpu idle	97.81 %	cpu idle	97.35 %	cpu idle	98.43 %
	user code	0.95 %	user code	1.14 %	user code	0.77 %
	kernel code	1.38 %	kernel code	1.73 %	kernel code	1.09 %
	free memory	5.196 GB	free memory	0.835 GB	free memory	0.893 GB
	num packets	17359	num packets	24974	num packets	8691
	in packets	45.03 %	in packets	42.47 %	in packets	43.60 %
	payloaded	74.74 %	payloaded	37.16 %	payloaded	57.38 %
	TCP	29.02 %	TCP	75.05 %	TCP	52.65 %
	tot weight	6.66 MB	tot weight	11.13 MB	tot weight	2.22 MB
	weight/packet	402.30 B	weight/packet	467.45 B	weight/packet	267.33 B
	tot time	2.08′	tot time	2.07′	tot time	2.07′
	packets/sec	137.77	packets/sec	199.79	packets/sec	70.09
Day 3	cpu idle	98.01 %	cpu idle	97.73 %	cpu idle	98.44 %
	user code	0.69 %	user code	0.98 %	user code	0.55 %
	kernel code	1.21 %	kernel code	1.58 %	kernel code	1.05 %
	free memory	5.076 GB	free memory	0.789 GB	free memory	0.886 GB
	num packets	14342	num packets	22040	num packets	8987
	in packets	45.27 %	in packets	41.33 %	in packets	44.45 %
	payloaded	77.63 %	payloaded	41.93 %	payloaded	59.73 %
	TCP	26.98 %	TCP	68.92 %	TCP	49.73 %
	tot weight	6.76 MB	tot weight	10.17 MB	tot weight	3.18 MB
	weight/packet	494.21 B	weight/packet	483.66 B	weight/packet	371.26 B
	tot time	2.03′	tot time	2.04′	tot time	2.05′
	packets/sec	117.56	packets/sec	179.19	packets/sec	73.07
Day 4	cpu idle	97.97 %	cpu idle	98.02 %	cpu idle	97.79 %
	user code	0.67 %	user code	0.64 %	user code	0.94 %
	kernel code	1.21 %	kernel code	1.14 %	kernel code	1.29 %
	free memory	5.024 GB	free memory	0.741 GB	free memory	0.873 GB
	num packets	14201	num packets	19140	num packets	16288
	in packets	44.71 %	in packets	42.75 %	in packets	44.59 %
	payloaded	78.98 %	payloaded	42.38 %	payloaded	39.92 %
	TCP	25.05 %	TCP	66.45 %	TCP	71.43 %
	tot weight	7.01 MB	tot weight	10.93 MB	tot weight	7.24 MB
	weight/packet	517.46 B	weight/packet	599.00 B	weight/packet	465.89 B
	tot time	2.07′	tot time	2.06′	tot time	2.05′
	packets/sec	114.52	packets/sec	154.35	packets/sec	132.42
Day 5	cpu idle	97.98 %	cpu idle	97.50 %	cpu idle	98.05 %
	user code	0.79 %	user code	1.02 %	user code	0.75 %
	kernel code	1.09 %	kernel code	1.66 %	kernel code	1.16 %
	free memory	4.958 GB	free memory	0.694 GB	free memory	0.860 GB
	num packets	13600	num packets	22454	num packets	15042
	in packets	44.46 %	in packets	42.19 %	in packets	44.78 %
	payloaded	80.54 %	payloaded	51.30 %	payloaded	37.90 %
	TCP	23.26 %	TCP	56.14 %	TCP	71.49 %
	tot weight	6.99 MB	tot weight	9.02 MB	tot weight	6.55 MB
	weight/packet	538.92 B	weight/packet	421.40 B	weight/packet	456.78 B
	tot time	2.05′	tot time	2.05′	tot time	2.05′
	packets/sec	110.57	packets/sec	181.08	packets/sec	122.29

Table 3: Summary table for *FSF supporter* user profile

5.1 *License*

5.2 *Project hosting*

All the content of this project has been versioned in a *Git* repository:
`https://github.com/massimo-nocentini/quantitative-systems-analysis-exam`.

Actually, the complete folder with Snort log cleaned files and the folder containing eps files aren't under version control since too large. To make it easier to download them we provide a *Makefile* in *pharo-smalltalk* directory under the project root, with the following rules:

- **download-pharo**, will download a complete Pharo environment, about 20 MB;
- **run-pharo**, will run the downloaded Pharo image;
- **download-eps**, will download all eps files necessary for compiling this TEXdocument, positioning the folder in the correct position, about 10 MB;
- **download-snort-logs**, will download the entire snort logs tree with the comprehensive set of eps not reported in this document, about 400 MB;
- **clean**, will remove all TEXcompilation output file and all files generated by running the Smalltalk test suite for our implementation, ie all plots and data files;
- **clean-all**, will remove eps and snort logs folders recursively, be sure to request this rule.

All code developed in *Pharo Smalltalk* is freely available at `http://smalltalkhub.com/#!/~MassimoNocentini/QuantitativeSystemsAnalysisExam` and can be loaded directly via *Monticello* smalltalk browser with the following message:

```
MCHttpRepository
location: 'http://smalltalkhub.com/mc/MassimoNocentini/QuantitativeSystemsAnalysisExam/main'
user: ''
password: ''
```

In order to use our implementation the package *CommandShell* is required: to load it into the image, open the world menu (click on an empty point on the image) then navigate to *Tools* and then *Configuration Browser*, after that select the *CommandShell* in the list and then click *Install Stable Version*.

REFERENCES

[1] SNORT Team, `https://www.snort.org/`.

[2] SNORT Team, `http://manual.snort.org/`.

[3] Free Software Foundation, `http://www.freebsd.org/cgi/man.cgi?query=vmstat`.

[4] Free Software Foundation, `http://www.freebsd.org/cgi/man.cgi?query=sed`

[5] Free Software Foundation,`http://www.gnuplot.info/`

[6] Andrea Bondavalli, Analisi quantitativa dei sistemi critici, March 2011, Progetto Leonardo, Esculapio Editore

[7] Marcel Weiher and Stephane Ducasse, Higher Order Messaging, Dynamic Languages Symposium (DLS) '05, October 18, 2005, San Diego, CA, USA