A Corpus of Group Dynamics Data from Internet Chatrooms

Galen Pickard[*], Roger Khazan, Ben Fuller, and Joe Cooley
MIT Lincoln Laboratory

[*] Corresponding author at: MIT Lincoln Laboratory, 244 Wood St. Lexington MA 02420.
E-mail: gpickard@ll.mit.edu

Motivation

What follows is a presentation of a methodology for extracting useful group dynamic data from the logs of group-chat sessions, and also the corpus of data that results from applying this methodology to a large, publicly-available collection of Internet chatlogs. This work is motivated by the lack of a large, consistent data-set in the public domain that might be used to apply quantitative methods to the study of group dynamics. While there has been a wealth of research regarding social networks, much of which is precise enough to allow for quantitative measurement, virtually none of it deals with the dynamics of groups in flux, and there does not exist a common data set, or even a widely accepted data format, that could be used to support such research.

Our research focuses on group oriented communications, and in particular on the problem of securing communication among dynamic groups. We present this data-set as an example of the kind of data which we have found useful in the evaluation and improvement of our algorithms. It is our hope that by making the data available to the public, we will encourage further research in this area – and that by making the methodology for producing such data available, we will encourage others to collect and share similar data.

Introduction

This data corpus consists of parsed, sanitized logs of internet text-chat in public chatrooms, spanning nearly 3 years (Apr '06 – Dec '08) and 42 separate chatrooms. The corpus contains information about changes to the group membership of the room (user joins, user departures, and user namechanges) and about the sizes of messages sent (in bytes), as well as the times at which all of these events occurred. Actual usernames have been anonymized through the use of a keyed HMAC, and no message content is stored. We believe that recovering any identifying information should be impossible without using the raw logs themselves. Though the raw logs from which these files were produced are still publicly available, any extraction of personal information though the combination of our logs and the original logs could be carried out just as easily from the raw logs alone.

The data are presented in a sqlite3 database[1]. This database contains one table per chatroom, each row of which corresponds to one chat event (join, leave, namechange, or message). Additionally, each of these rows contains information about the date, time, (sanitized) username, event type, and additional information such as number of bytes in a message, or the new name in a namechange. These data should be sufficient to perform analysis of a wide variety of group dynamics, and to test the performance of algorithms sensitive to group makeup, such as access control, content distribution, or group keying algorithms.

The data format is a simple table, a representative sample is included here:

```
chatlogs.jabber.ru unix 1225473175 2008 10 31 17 12 55 message "RaymondWilson64"    29
chatlogs.jabber.ru unix 1225473177 2008 10 31 17 12 57 message "PatriciaCampbell81" 47
chatlogs.jabber.ru unix 1225473200 2008 10 31 17 13 20 join    "PaulRichardson53"
chatlogs.jabber.ru unix 1225473223 2008 10 31 17 13 43 message "BarbaraBarnes29"    26
chatlogs.jabber.ru unix 1225473239 2008 10 31 17 13 59 leave   "RaymondWilson64"
chatlogs.jabber.ru unix 1225473245 2008 10 31 17 14 5  leave   "PaulRichardson53"
```

Additionally, various scripts and documentation are included with the database, to facilitate analysis of this dataset as well as addition of new raw logs to the corpus. These scripts include:

- **sanitize.pl –** takes raw chatlogs and presents them in a "canonical format"
- **sani2db.pl –** prepares sanitized chatlogs for insertion into a SQL database
- **groupparse.pl** – extracts group-dynamic information from sanitized chatlogs
- **messageparse.pl** – extracts size and recipient data from sanitized chatlogs
- **population.m** – plots group population over time
- **popimage.m** – presents group dynamic information in a pictoral format
- **condpop.m** – presents information about the overlap of pairs of users in a group

[1] Sqlite3 is freely available at www.sqlite.org

Scripts

sanitize.pl – takes as input a text/html file containing a chat log, parses and optionally anonymizes it. Its inputs are a raw chatlog, a secret key used for anonymization, and a list of names used to make anonymized names human-readable. Formatting of logs differs widely between chat protocols, logging programs, and even servers; the first lines of sanitize.pl consist of a series of regular expressions which allow a user to modify the script to target nearly any log syntax. When presented with a raw log named chatlog.html, sanitize.pl produces a sanitized version called chatlog.html.sani. The expected format of the "names" file is as follows: the first line contains a number which gives the number of components a name has; the second contains a number which gives the number of options each component can take; the rest of the file is interpreted as text, with each line giving one option of one component of a name. For example, the following is a valid names file:

```
3
4
Alice
Bob
Carol
David
Smith
Jones
Brown
Davis
1
2
3
4
```

Which will produce, for example, the names "AliceJones1" and "CarolSmith2" – the 3 components, notionally, are "first name," "last name," and "number," each of which has 4 possible values. There are, thus, 4^3 = 64 possible names. We recommend use of a names file allowing for a number of possible names greatly exceeding the number of unique usernames in a log, to avoid hash collisions. If the same secret key and names file are used in the parsing of two different logs, raw usernames in the two logs will correspond to the same anonymized names in the two sanitized outputs. We provide a sample names file with 1,000,000 possible names – censusnames – based on the top names from the most recent US census.

The output format of sanitize.pl is what we call the "canonical form" of a chatlog:

```
22:51:05 message "MariaPerry13" 18
22:52:17 leave "MargaretLewis79"
22:52:30 join "MelissaButler100"
22:52:33 leave "MelissaButler100"
23:00:15 join "JasonButler74"
```

Each line gives the time of the event, the event type, the relevant username, and if needed (as in the case of a message or namechange), the additional data. sanitize.pl can accept timestamps in either HH:MM:SS or UTC format.

sani2db.pl – takes as input the output of sanitize.pl, and produces sqlite3 commands which add its contents to a database. The output is presented to stdout, and can be piped directly into sqlite3. sani2db.pl infers date and room information from the file name; it should be modified to suit.

groupparse.pl – takes as input through stdin a chatlog in canonical format, and produces a bit vector which represents the chat room's population over time. Each line of the output corresponds to a join, leave, or namechange. The first number is the UTC of the event, and the rest is a bit vector representing the room's population. Each column corresponds to a single user, and the columns map onto an alphabetical list of all users seen in the log. If the input is in HH:MM:SS format, it will be converted to a UTC as though the date were Jan 1, 1970.

For example, the following input:

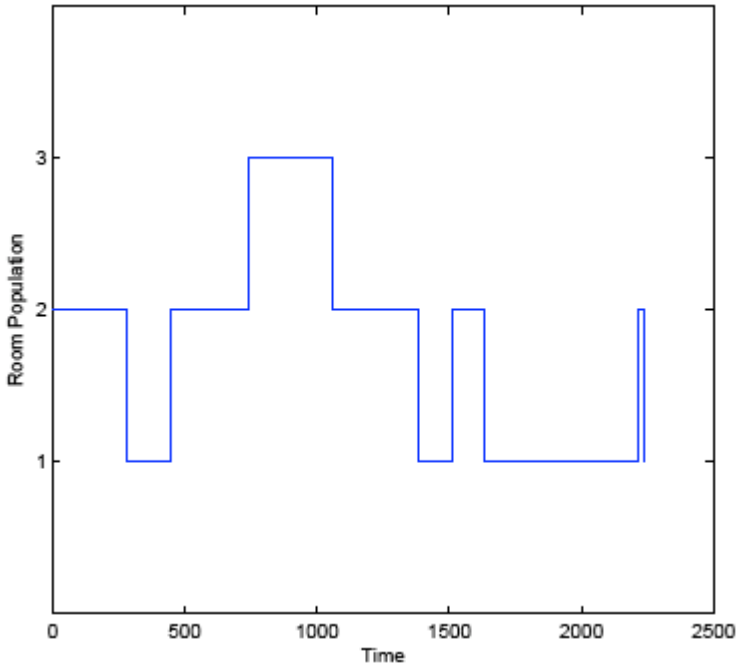
```
00:00:01 join "DeborahPeterson68"
00:04:41 leave "DeborahPeterson68"
00:07:27 join "DeborahPeterson68"
00:12:23 join "ChristopherGarcia84"
00:17:35 leave "ChristopherGarcia84"
00:23:02 leave "MichaelLopez78"
00:25:15 join "TimothyTorres94"
00:27:10 leave "TimothyTorres94"
00:36:58 join "ChristopherJackson39"
00:37:14 leave "ChristopherJackson39"
```

Produces the following output:

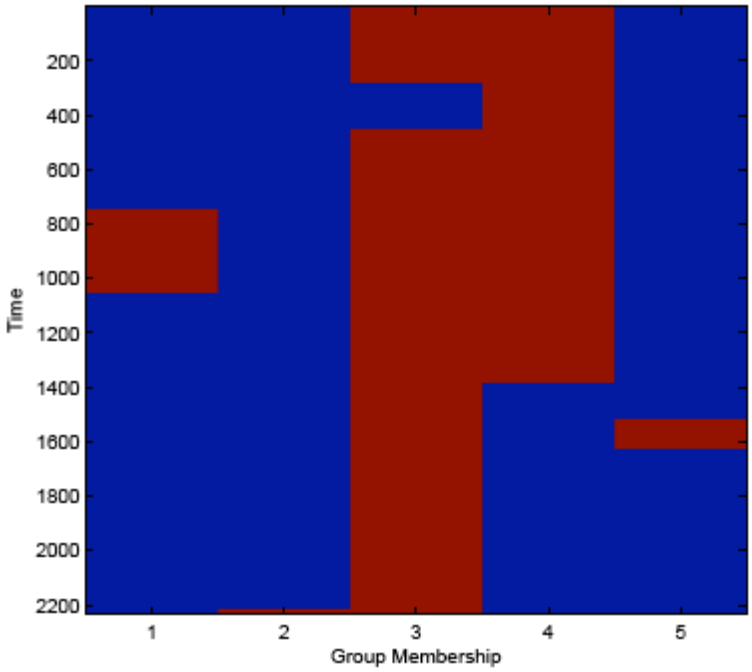
```
1          0 0 1 1 0
281        0 0 0 1 0
447        0 0 1 1 0
743        1 0 1 1 0
1055       0 0 1 1 0
1382       0 0 1 0 0
1515       0 0 1 0 1
1630       0 0 1 0 0
2218       0 1 1 0 0
2234       0 0 1 0 0
```

messageparse.pl – similar to groupparse.pl, except that each output line corresponds to a message, and specifies the set of users who were recipients of that message.

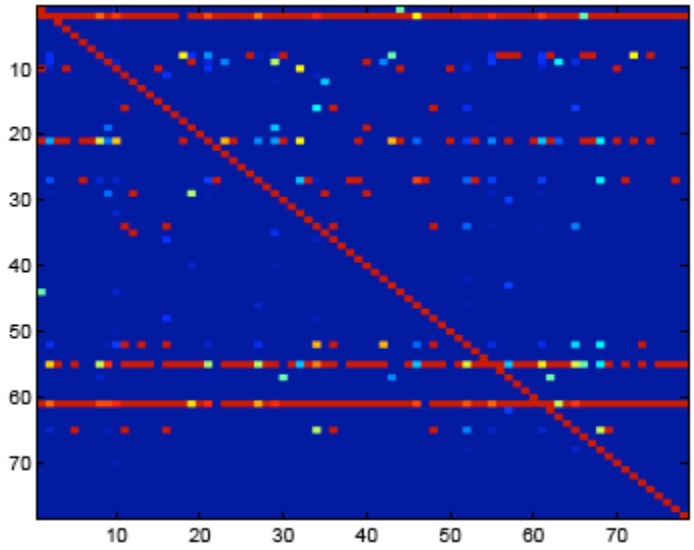
population.m – a MATLAB script which takes as input the output of groupparse.pl and produces a plot of room population over time. The above, for example, creates:



popimage.m – a MATLAB script which takes as input the output of groupparse.pl and produces an image showing which users were in the room at any given time.



condpop.m – a MATLAB script which takes as input the output of groupparse.pl and produces an image showing how correlated the activity patterns of two users were. Specifically, it shows the probability that user X was present given that user Y was present over the course of the dataset. Below is an example showing the relation of users over the course of a month in one of the less populated chatrooms.



SQL Primer

Additionally, users with limited SQL experience might find the following useful as a primer for the schema used to store the data:

```
sqlite3 -separator " " [database] "select utc, type,
member, params from '[room]' where [conditions]"
```

returns data in the form:

```
1230704780 message "MariaPerry13" 18
1230705082 leave "MargaretLewis79"
1230705465 join "MelissaButler100"
1230708158 leave "MelissaButler100"
1230731060 join "JasonButler74"
```

Some sample queries:

Print all activity from room 'unix@chatlogs.jabber.ru'

```
sqlite3 -separator " " [database] "select utc, type,
member, params from 'unix@chatlogs.jabber.ru'"
```

… on Dec 10, 2008

```
sqlite3 -separator " " [database] "select utc, type,
member, params from 'unix@chatlogs.jabber.ru' where mon=12
and day=10 and year=2008"
```

… between April and June of 2007

```
sqlite3 -separator " " [database] "select utc, type,
member, params from 'unix@chatlogs.jabber.ru' where mon
between 4 and 6 and year=2007"
```

… between Jan 5 and Feb 10 of 2007

```
sqlite3 -separator " " [database] "select utc, type,
member, params from 'unix@chatlogs.jabber.ru' where (mon=1
and day>=5) or (mon=2 and day<=10) and year=2007"
```

… between UTC 1202250000 and Mar 5 of 2008

```
sqlite3 -separator " " [database] "select utc, type,
member, params from 'unix@chatlogs.jabber.ru' where utc >=
1202250000 and (year<2008 or (year=2008 and (mon<3 or
(mon=3 and day<=5))))"
```

… involving member "NancyHall47"

```
sqlite3 -separator " " [database] "select utc, type,
member, params from 'unix@chatlogs.jabber.ru' where
member='\"NancyHall47\"'"
```

Print all activity from rooms 'unix@chatlogs.jabber.ru' and 'vim@chatlogs.jabber.ru' on Dec 10, 2008

```
sqlite3 -separator " " [database] "select utc, type,
member, params from (select * from
'unix@chatlogs.jabber.ru' union select * from
'vim@chatlogs.jabber.ru') where mon=12 and day=10 and
year=2008"
```

… with room names appended to output

```
sqlite3 -separator " " [database] "select room utc, type,
member, params from (select * from
'unix@chatlogs.jabber.ru' union select * from
'vim@chatlogs.jabber.ru') where mon=12 and day=10 and
year=2008"
```

Print all users of room 'unix@chatlogs.jabber.ru' sorted alphabetically

```
sqlite3 -separator " " [database] "select distinct member
from 'unix@chatlogs.jabber.ru' order by member"
```

… sorted time of first appearance

```
sqlite3 -separator " " [database] "select distinct member
from 'unix@chatlogs.jabber.ru' order by utc”
```

… who have ever sent a message longer than 100 bytes

```
sqlite3 -separator " " [database] "select distinct member
from 'unix@chatlogs.jabber.ru' where type='message' and
cast(params as int)>100"
```

Print the number of join events in room 'unix@chatlogs.jabber.ru'

```
sqlite3 -separator " " [database] "select count(*) from
'unix@chatlogs.jabber.ru' where type='join'"
```

Print total bytes sent by each member of room 'unix@chatlogs.jabber.ru'

```
sqlite3 -separator " " [database] "select member,
sum(cast(params as int)) from 'unix@chatlogs.jabber.ru'
where type='message' group by member"
```

Print message size frequency for room 'unix@chatlogs.jabber.ru'

```
sqlite3 -separator " " [database] "select cast(params as
int), count(cast(params as int)) from
'unix@chatlogs.jabber.ru' where type='message' group by
cast(params as int)"
```

Database Generation

With the scripts provided, it should be quite easy to obtain the publicly available chatlogs we used, and then recreate the database from scratch to ensure that it has not been manipulated. To do so, issue the following commands:

```
wget –mirror http://chatlogs.jabber.ru

for i in $(ls
chatlogs.jabber.ru/*\@conference.jabber.ru/*/*/*.html); do
echo $i; ./sanitize.pl $i SECRETKEY censusnames; done

find ./chatlogs.jabber.ru/*\@conference.jabber.ru –iname
“*sani*” –print –exec sh –c ‘./sani2db.pl $1 | sqlite3
chatlogs.db’ {} {} \;
```

This will take some number of hours, depending on your internet connection and processor speed. To verify success, compare some prefix of a table against the one provided. For example, the table ‘unix@chatlogs.jabber.ru’ should start like:

```
1169789998 message "AnnRobinson27" 41
1169790019 message "JohnPatterson56" 36
1169790024 message "AnnRobinson27" 164
1169790076 message "JohnPatterson56" 15
1169790088 message "JohnPatterson56" 132
1169790095 message "AnnRobinson27" 31
1169790106 message "AnnRobinson27" 18
1169790132 message "JohnPatterson56" 13
1169790620 leave "AnnRobinson27"
1169790641 join "AnnRobinson27"
1169790769 leave "AnnRobinson27"
1169791166 message "JohnPatterson56" 123
1169791208 message "JohnPatterson56" 124
1169794240 leave "AmyWalker61"
1169819313 leave "DouglasCox92"
1169819364 join "DouglasCox92"
1169823341 message "JohnPatterson56" 396
1169823848 leave "JohnPatterson56"
1169824526 join "JohnPatterson56"
1169824554 message "JohnPatterson56" 93
```

Conclusion

In this document, we have outlined our motivation for releasing a corpus of data and scripts which are useful for the analysis of group dynamics and behavior. We have provided documentation of the functionality of these scripts and the schemas of the database and output. We have provided examples of post-processing computations that could be carried out using this data, and have given examples which should make it easy for end-users to extract useful data from the corpus. We have provided the means to generate the corpus using the provided scripts and publicly-available data, demonstrating that the attached database does not contain any information whose release should be restricted. All together, we believe that we have provided the motivation and justification necessary to support public release of these data and scripts.