# Internet Measurement and Data Analysis (3)

Kenjiro Cho

2011-10-12

# review of previous class

Measuring the size of the Internet

- the number of users and hosts
- the number of web pages
- precision, errors, significant digits
- how to make good graphs
- exercise: graph plotting by gnuplot

# today's topics

items left off from previous class

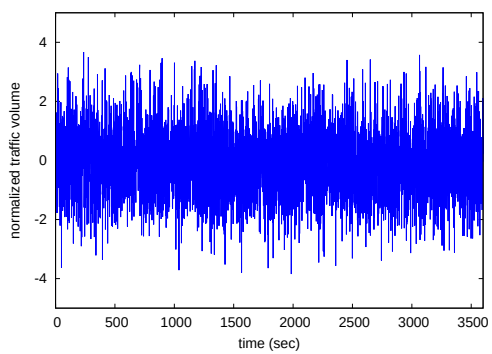
- how to make good graphs
- exercise: graph plotting by gnuplot

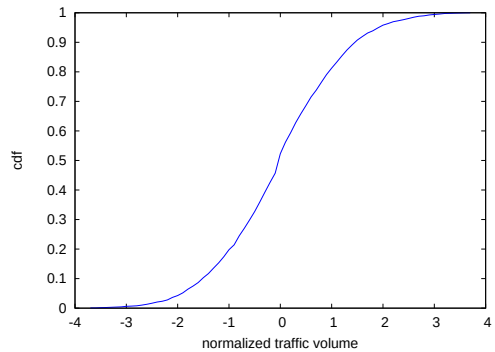
Data recording and log analysis

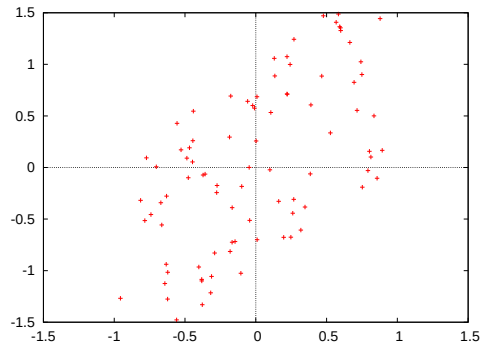
- data format
- log analysis methods
- exercise: log data and regular expression

# graph plotting

create a set of plots using statistical techniques to intuitively understand the data



# guidelines for plotting

require minimum effort from the reader

- label the axes clearly
- label the tics on the axes
- identify individual curves/bars
- select appropriate font size
- use commonly accepted practices
  - zero-origins, math symbols, acronyms
- show variation/distribution of variables
- select ranges properly
- do not present too many items in a single chart
- when comparing data sets, use appropriate normalization
- when comparing plots, use the same scale for the axes
- when using colors
  - make sure it is readable in black-and-white print
  - make sure readable on data projectors (e.g., do not use yellow)

# variables in data

- univariate analysis
  - explores a single variable in a data set, separately
- multivariate analysis
  - looks at more than one variables at a time

# plotting raw data

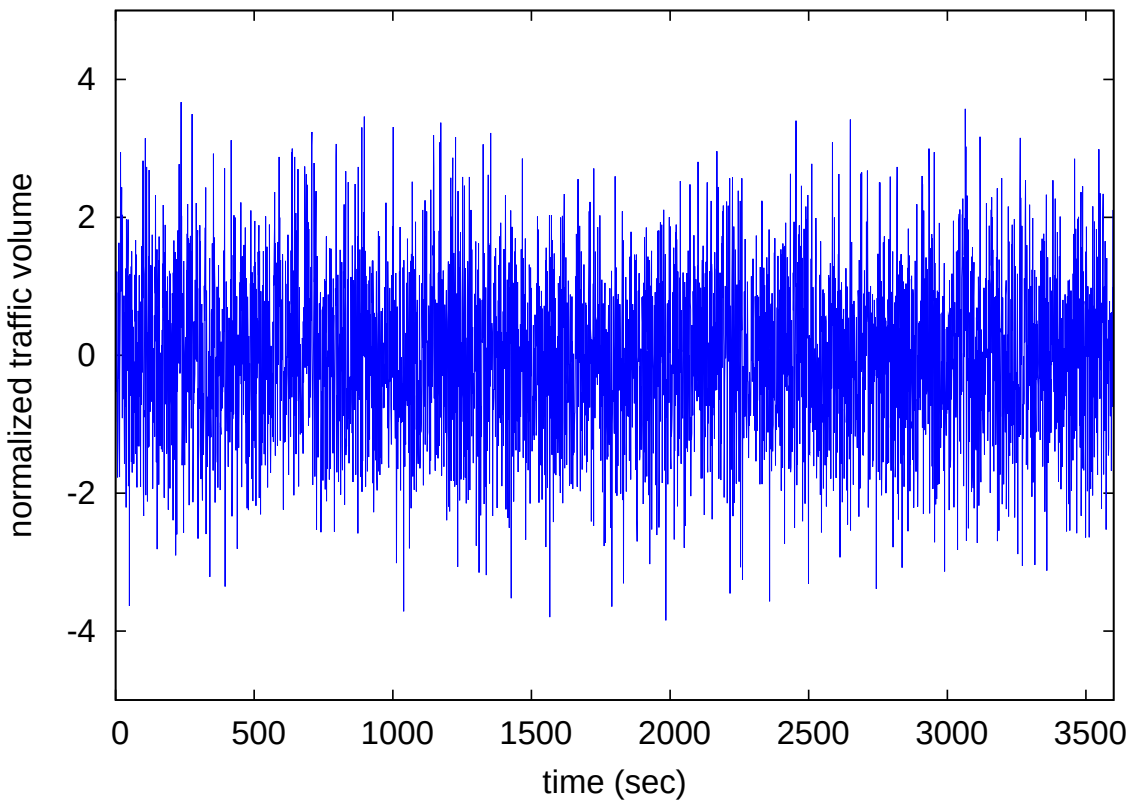
- time series plots
- histograms
- probability plots
- scatter plots

there are many other plotting techniques

# time series plots

time-series plots (or other sequence plots) provides a feel for the data

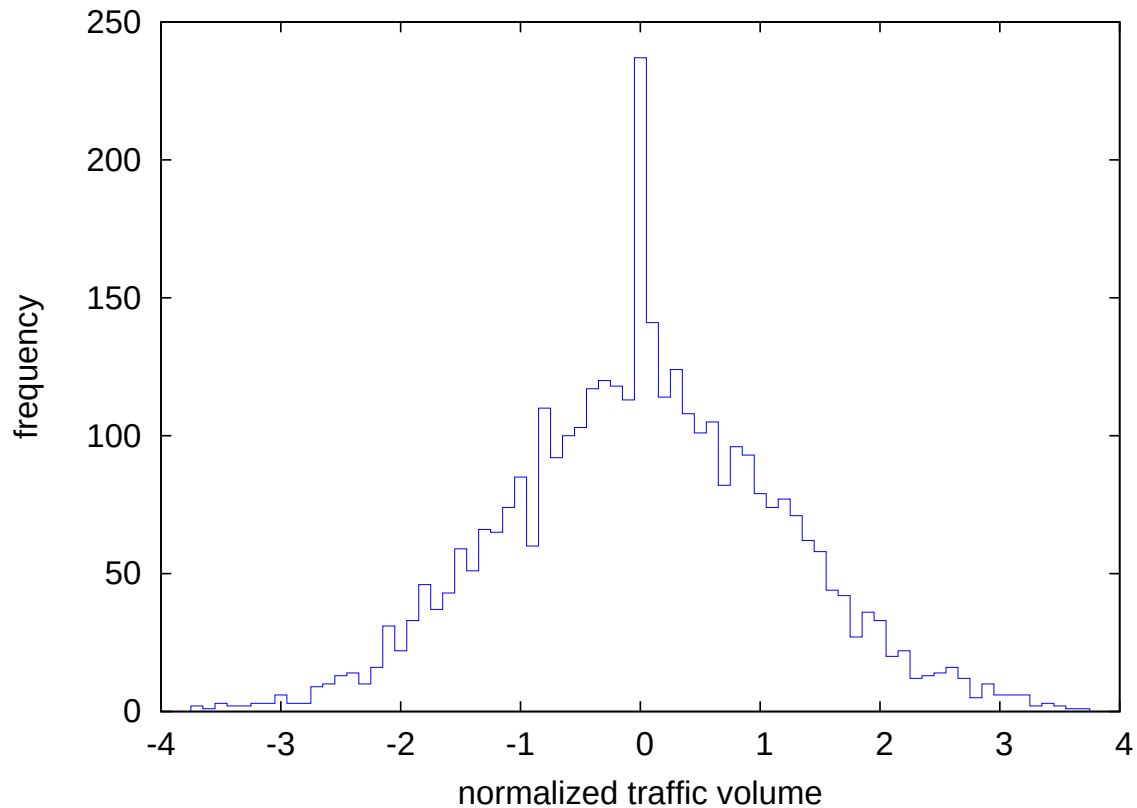
- you can identify
  - shifts in locations
  - shifts in variation
  - outliers



# histograms

to see distribution of the data set

- split the data into equal-sized bins by value
- count the frequency of each bin
- plot
  - X axis: variable
  - Y axis: frequency



# histograms (cont)

with histograms

- you can identify
  - center (i.e., the location) of the data
  - spread (i.e., the scale) of the data
  - skewness of the data
  - presence of outliers
  - presence of multiple modes in the data

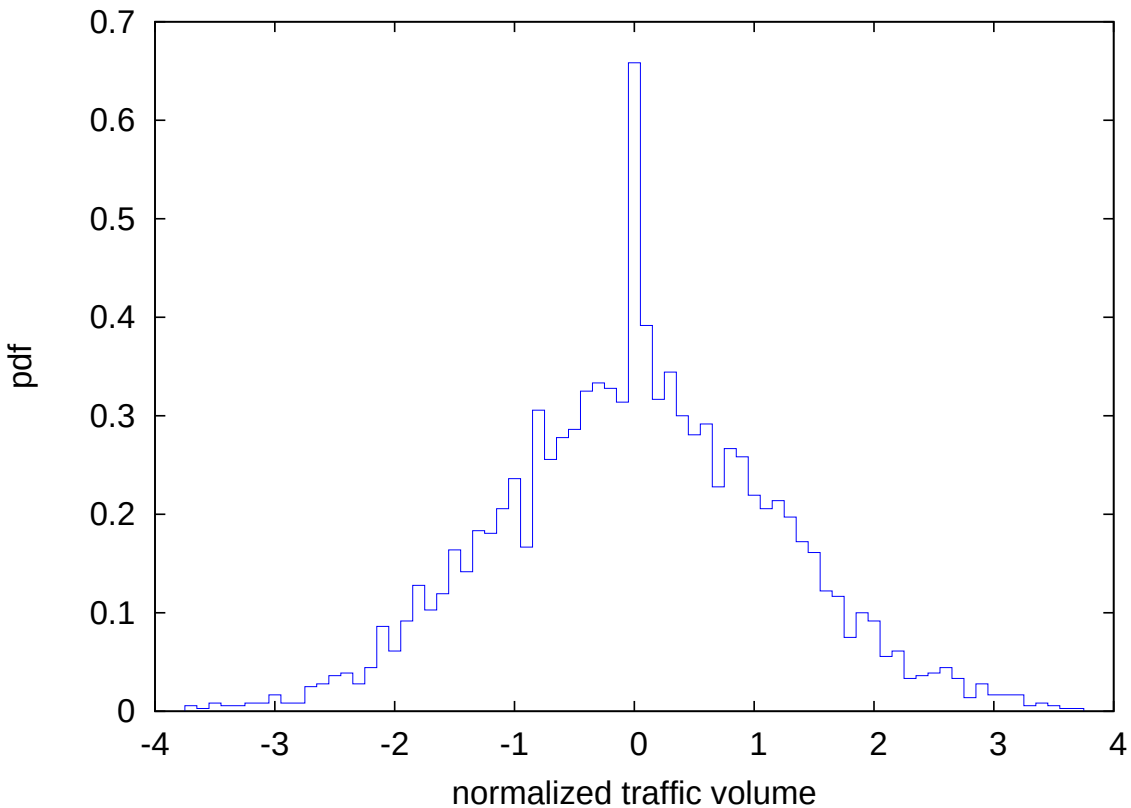
limitations of histograms

- needs appropriate bin size
  - too small: each bin doesn't have enough samples (e.g., empty bins)
  - too large: only few regions available
  - difficult for highly skewed distribution
- enough samples needed

# probability density function (pdf)

- normalize the frequency (count)
  - sum of the area under the histogram to be 1
  - divide the count by the total number of observations times the bin width
- probability density function: probability of observing $x$

$$f(x) = P[X = x]$$



# cumulative distribution function (cdf)

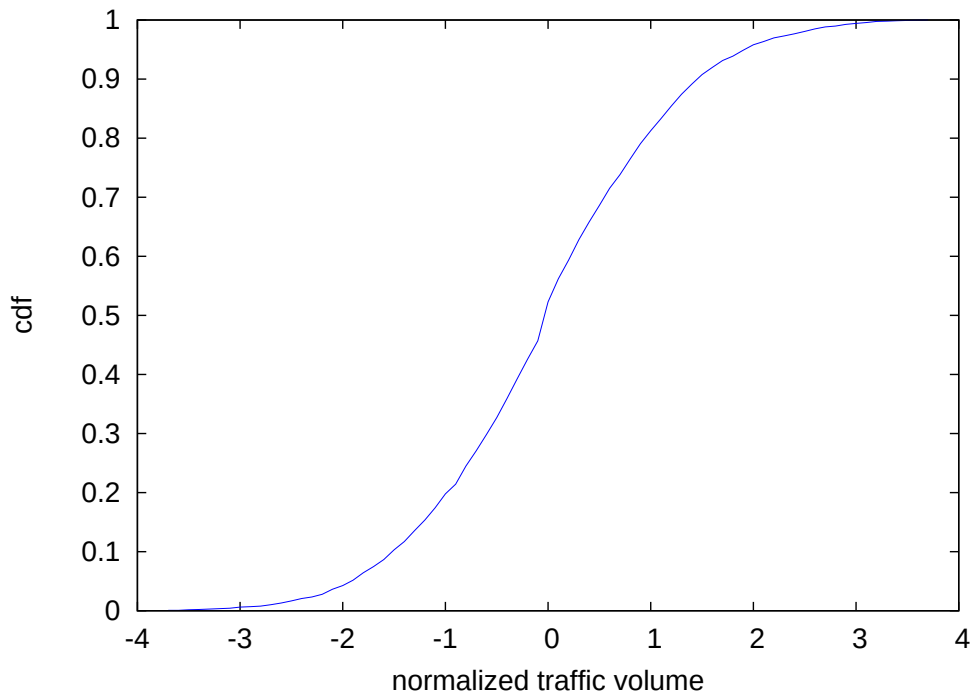
- density function: probability of observing $x$

$$f(x) = P[X = x]$$

- cumulative distribution function: probability of observing $x$ or less

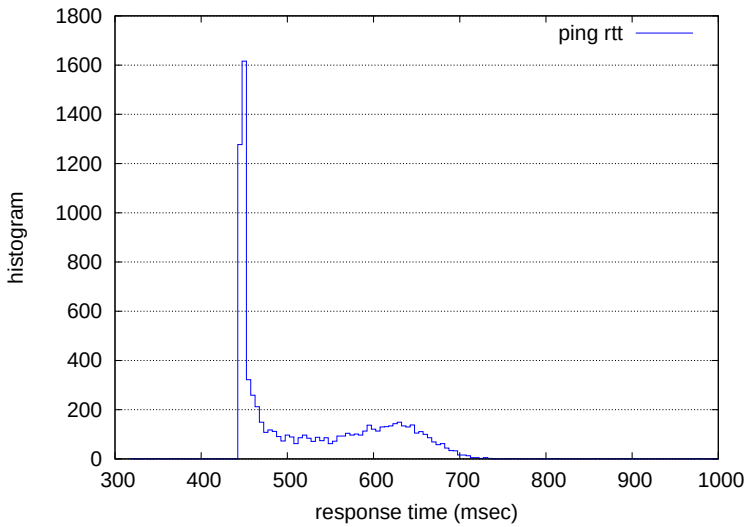
$$F(x) = P[X <= x]$$

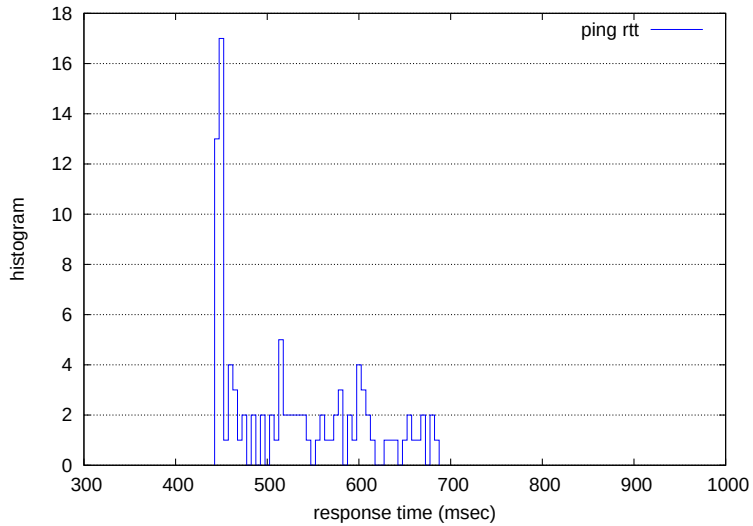
- better than histogram when distribution is highly skewed, sample count is not enough, or outliers are not negligible

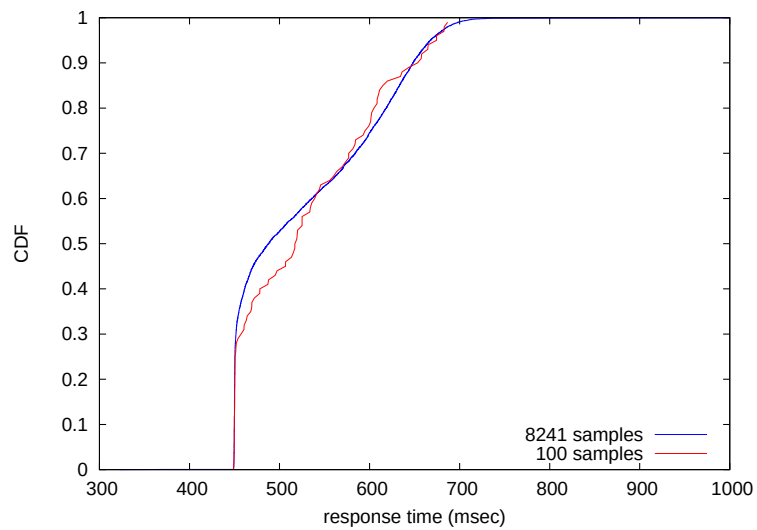


# histogram vs cdf

- no need to worry about bin size or sample count for cdf



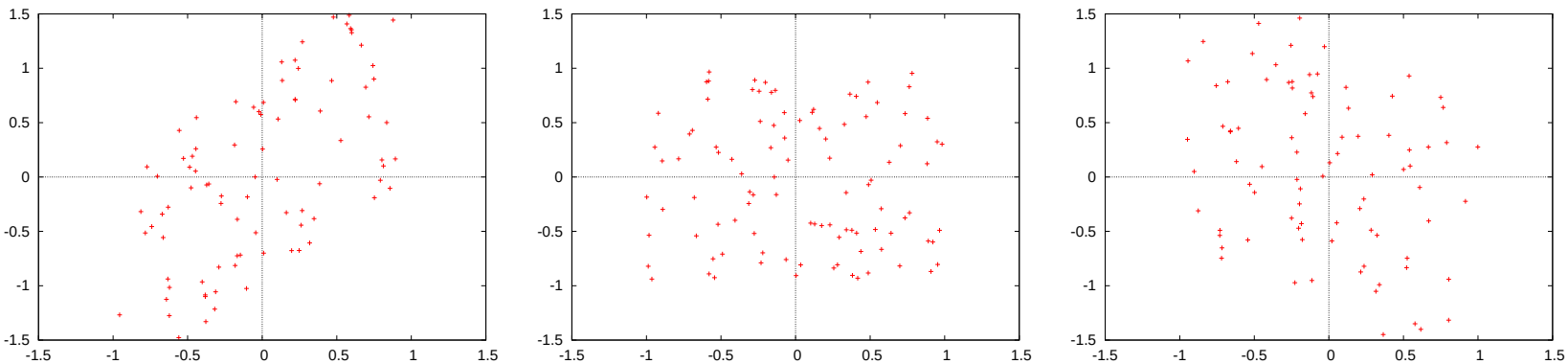




original data (left), 100 samples (right), cdfs (bottom)

# scatter plots

- explores relationships between 2 variables
  - X-axis: variable X
  - Y-axis: corresponding value of variable Y
- you can identify
  - whether variables X and Y related
    - no relation, positive correlation, negative correlation
  - whether the variation in Y changes depending on X
  - outliers
- examples: positive correlation 0.7 (left), no correlation 0.0 (middle), negative correlation -0.5 (right)




examples: positive correlation 0.7 (left), no correlation 0.0 (middle), negative correlation -0.5 (right)

# plotting tools

- gnuplot
  - command-line tool suitable for automated plotting
  - http://gnuplot.info/
- grace
  - comes with graphical user interface
  - powerful for fine-tuning the output
  - http://plasma-gate.weizmann.ac.il/Grace/

# exercise: gnuplot

- plotting a simple graph using gnuplot
- sample data from a book: P. K. Janert "Gnuplot in Action"
  - `http://web.sfc.keio.ac.jp/~kjc/classes/sfc2011f-measurement/marathon.txt`
  - `http://web.sfc.keio.ac.jp/~kjc/classes/sfc2011f-measurement/prices.txt`

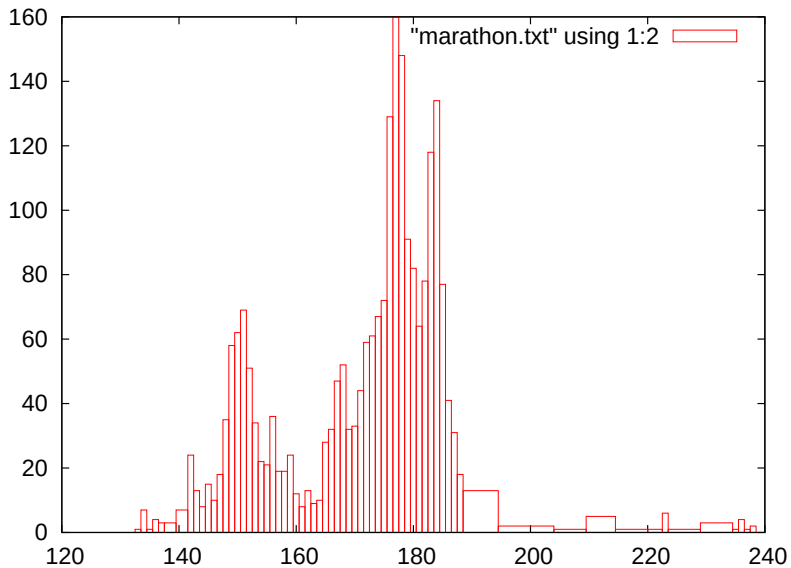
# histogram

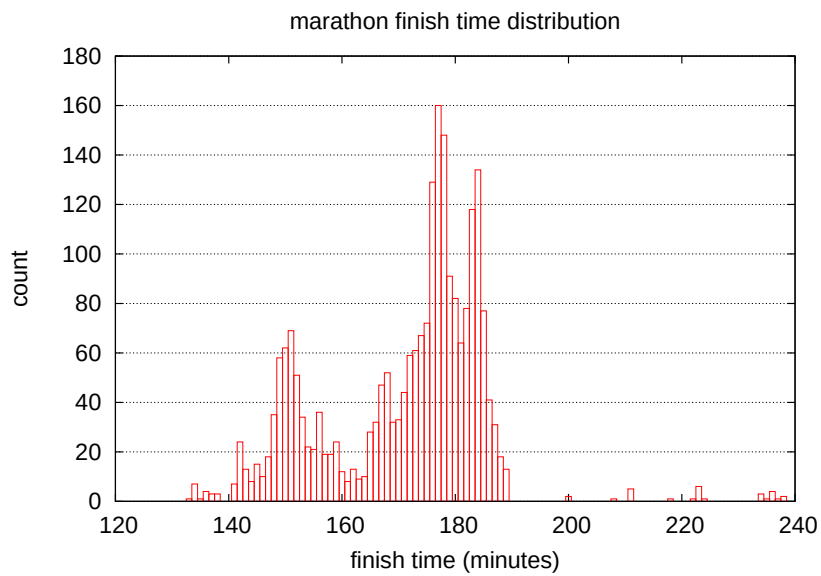
- distribution of finish time of a city marathon

```
plot "marathon.txt" using 1:2 with boxes
```

make the plot look better (right)

```
set title "marathon finish time distribution"
set boxwidth 1
set xlabel "finish time (minutes)"
set ylabel "count"
set yrange [0:180]
set grid y
plot "marathon.txt" using 1:2 with boxes notitle
```

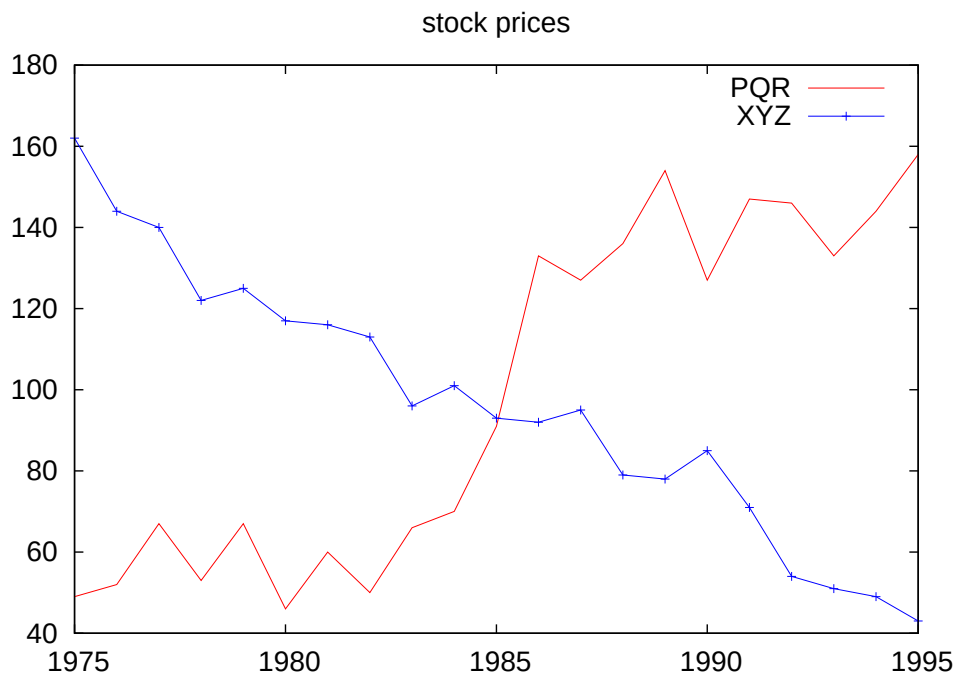


# time-series plot

- stock prices over time: PQR and XYZ

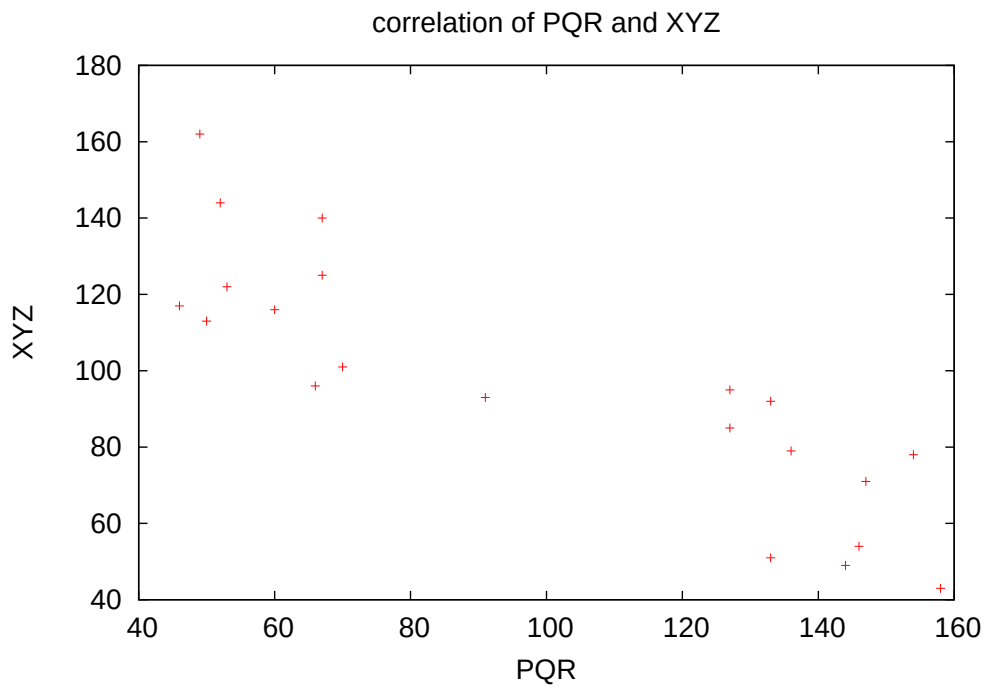
```
set title "stock prices"
plot "prices.txt" using 1:2 title "PQR" with lines, \
"prices.txt" using 1:3 title "XYZ" with linespoints linetype 3
```



# scatter plot

- correlation of stock prices: PQR and XYZ

```
set title "correlation of PQR and XYZ"
set xlabel "PQR"
set ylabel "XYZ"
plot "prices.txt" using 2:3 notitle with points
```



# log data

- web server accesslog
- mail log
- syslog
- firewall log
- IDS log
- other forms of event records

# why do we analyze logs?

- understand current situations
  - new findings: technical advances, changes in usage
  - then, predict the future
- identify security problems and equipment failures, and their symptoms
- improve techniques for analysis
  - automation
- report outages, and responses to problems
- record events
  - for legal and other reasons

if not analyzed, logs have no value
(do not be satisfied only with collecting logs)

# problems in log analysis

- huge data volume
- lack of necessary information and precision, credibility of timestamps and content
- missing records (due to failures of data collection systems)
- many different formats
- data analysis requires time and efforts
- many people think data analysis is difficult

# log management

- log collection
  - programming (e.g., use of the syslog API)
  - building a data collection system
- log rotation
  - remove old data after a certain period
  - according to log size, time order, ages of data
  - should not lose data at log rotation
- RRD (Round Robin Database)
  - keep the data size by aggregating old logs
  - examples: 5 min data for 1 week, 2 hour data for a month, 1 day data for a year
- visualization
  - make it easier to grasp situation

# log formats

- web server access log
- mail log
- DHCP server log
- syslog

# web server access log

- Apache Common Log Format
  - client_IP client_ID user_ID time request status_code size
- Apache Combined Log Format
  - Common Log Format plus "referer" and "User-agent"
  - client_IP client_ID user_ID time request status_code size referer user-agent
- other customizations are possible

```
client_IP: IP address of the client
client_ID: identity of the client (when the client is authenticated)
user_ID: authenticated user name
time: the time that the request was received
request: the first line of the request
status_code: HTTP response status
size: the size of the object returned (not including the deader), "-" means
referer: the site that the client referred from (source of the link)
user-agent: client's browser type
```

Example Combined Log Format:

```
127.0.0.1 - frank [10/Oct/2000:13:55:36 -0700] \
  "GET /apache_pb.gif HTTP/1.0" 200 2326 \
  "http://www.example.com/start.html" \
  "Mozilla/4.08 [en] (Win98; I ;Nav)"
```

# mail log

logging when email is processed (receiving, sending, etc)
example:

```
Oct 27 13:32:54 server3 sm-mta[24510]: m9R4WsBe024510:\
 from=<client@example.com>, size=2403, class=0, nrcpts=1 \
 msgid=<201012121547.oBCFlPX6032787@example.com>, \
 proto=ESMTP, daemon=MTA, relay=mail.example.co.jp [192.0.2.1] \
Oct 27 14:43:04 server3 sm-mta[24511]: m9R4WsBe024510: \
 to=<user@example.co.jp>, delay=01:10:10 xdelay=00:00:00, \
 mailer=local, pri=32599, dsn=2.0.0, stat=Sent
```

- time
- host name
- process owner [process id]
- Queue ID: internal id for the email
- ...
- nrcpts: number of recipients
- relay: next mail server to send the message
- dsn: Delivery Status Notification, RFC3463
  - 2.X.X:Success, 4.X.X:Persistent Transient Failure, 5.X.X:Permanent Failure
- stat: Message Status
  - Sent, Deferred, Bounced, etc

# DHCP server log

```
SYSLOG messages:

Oct 28 15:04:32 server33 dhcpd: DHCPDISCOVER from 00:23:df:ff:a8:a7 via eth0
Oct 28 15:04:32 server33 dhcpd: DHCPOFFER on 192.168.2.101 \
  to 00:23:df:ff:a8:a7 via eth0
Oct 28 15:04:32 server33 dhcpd: DHCPREQUEST for 192.168.2.101 \
  from 00:23:df:ff:a8:a7 via eth0
Oct 28 15:04:32 server33 dhcpd: DHCPACK on 192.168.2.101 \
  to 00:23:df:ff:a8:a7 via eth0
Oct 28 15:09:32 server33 dhcpd: DHCPREQUEST for 192.168.2.101 \
  from 00:23:df:ff:a8:a7 via eth0
Oct 28 15:09:32 server33 dhcpd: DHCPACK on 192.168.2.101 \
  to 00:23:df:ff:a8:a7 via eth0

dhcpd.leases: records of status of each assigned IP

lease 192.168.100.161 {
  starts 4 2010/12/09 23:13:39;
  ends 5 2010/12/10 00:13:39;
  tstp 5 2010/12/10 00:13:39;
  binding state free;
  hardware ethernet 5c:26:0a:17:06:00;
}
```

# syslog

- a framework to send and store arbitrary messages on UNIX-like systems
  - originally designed for mail server logs
  - widely used for other purposes
  - supports sending messages to other servers
  - log rotation support
- Windows Event Log

# log analysis techniques

- try out ideas by plotting graphs
  - new ideas often come up when working on data
- scripts and command line tools (grep, sort, uniq, sed, awk, etc)
- consider how to process huge data sets efficiently
- automate processes which you will repeat
  - do not rely too much on automated processes

# how to handle huge data sets

- naive algorithms often consume too much memory
  - it helps to study data structures and algorithms
- how to handle huge data sets
  - remove unnecessary information
  - aggregate data temporally and spatially
  - divide and conquer
  - distributed and/or parallel processing
- convert to an intermediate file
- estimate required memory
  - use of efficient data structures
  - limit the size and/or dimensions to process at a time
- estimate processing time
  - a test run with a smaller data set
  - use scalable algorithms
- trade-off between memory size and processing time

# Web access log sample data

- apache log (combined log format)
- from a JAIST server, access log for 24 hours
  `http://www.iijlab.net/~kjc/classes/sfc2011f-measurement/sample_access_log.bz2`
- about 14MB (bzip2 compressed), about 280MB after bunzip2
- 1/10 sampling
- client IP addresses are anonymized (1-to-1 mapping) for privacy
- test data (first 100 lines)
  `http://www.iijlab.net/~kjc/classes/sfc2011f-measurement/test-100lines`

# sample data

```
143.207.214.239 - - [18/Jul/2010:23:59:53 +0900] "GET /pub/mozilla.org/firefox/releases/3.6.6/\
  update/mac/de/firefox-3.6.6.complete.mar HTTP/1.1" 206 300371 "-" "Mozilla/5.0 (Macintosh; U;\
  Intel Mac OS X 10.6; de; rv:1.9.2.3) Gecko/20100401 Firefox/3.6.3" ftp.jaist.ac.jp
161.42.4.49 - - [18/Jul/2010:23:59:20 +0900] "GET /pub/PC-BSD/8.0/i386/PCBSD8.0-x86-DVD.iso\
  HTTP/1.1" 206 58970 "http://ftp.jaist.ac.jp/pub/PC-BSD/8.0/i386" "Mozilla/4.0 (compatible;\
  MSIE 6.0; Windows NT 5.1; SV1; .NET CLR 1.1.4322; .NET CLR 2.0.50727)" ftp.jaist.ac.jp
150.107.216.201 - - [18/Jul/2010:23:59:56 +0900] "GET /pub/mozilla.org/firefox/releases/3.6.6/\
  update/win32/en-GB/firefox-3.6.6.complete.mar HTTP/1.1" 206 300368 "-" "Mozilla/5.0 (Windows;\
  U; Windows NT 6.0; en-GB; rv:1.9.2.3) Gecko/20100401 Firefox/3.6.3 (.NET CLR 3.5.30729)"\
  ftp.jaist.ac.jp
22.32.128.50 - - [19/Jul/2010:00:00:00 +0900] "HEAD /project/clamav/clamav/win32/ClamAV-0.96.1\
  -64bit-beta.zip HTTP/1.0" 200 302 "http://jaist.dl.sourceforge.net/project/clamav/clamav/\
  win32/" "Wget/1.10.2 (Red Hat modified)" jaist.dl.sourceforge.net
137.29.144.83 - - [19/Jul/2010:00:00:00 +0900] "GET /pub/mozilla.org/thunderbird/releases/\
  2.0.0.24/update/win32/en-US/thunderbird-2.0.0.24.complete.mar HTTP/1.1" 200 65845 "-"\
  "Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.1.22) Gecko/20090605 Thunderbird/\
  2.0.0.22" ftp.jaist.ac.jp
22.32.128.50 - - [19/Jul/2010:00:00:00 +0900] "HEAD /project/clamav/clamav/win32/Clamunrar-\
  0.96.zip HTTP/1.0" 200 298 "http://jaist.dl.sourceforge.net/project/clamav/clamav/win32/"\
  "Wget/1.10.2 (Red Hat modified)" jaist.dl.sourceforge.net
209.235.74.175 - - [18/Jul/2010:23:59:52 +0900] "GET /pub/mozilla.org/firefox/releases/3.6.6/\
  update/win32/en-US/firefox-3.6.6.complete.mar HTTP/1.1" 206 300368 "-" "Mozilla/5.0 (Windows;\
  U; Windows NT 6.1; en-US; rv:1.9.2) Gecko/20100115 Firefox/3.6" ftp.jaist.ac.jp
153.42.115.45 - - [18/Jul/2010:23:59:56 +0900] "GET /pub/mozilla.org/firefox/releases/3.5.10/\
  update/win32/pl/firefox-3.5.10.complete.mar HTTP/1.1" 206 300368 "-" "Mozilla/5.0 (Windows;\
  U; Windows NT 6.0; pl; rv:1.9.1.5) Gecko/20091102 Firefox/3.5.5 (.NET CLR 3.5.30729)"\
  ftp.jaist.ac.jp
...
```

# regular expressions

regular expressions

- expressions of patterns of characters, used for search and replace of strings
- originally designed to specify formal language in formal language theory
- later widely used for text pattern matching
  - grep, expr, awk, vi, lex, perl, ruby, ...

Ruby's regular expression

```
Regexp class
regular expression literal: /regexp/opt
=~ operator: subject =~ /regexp/
match() method:  /regexp/.match(subject)
string class: string.match(/regexp/)
```

# Ruby regular expressions: quick reference

```
[abc]  A single character: a, b or c
[^abc]  Any single character but a, b, or c
[a-z]  Any single character in the range a-z
[a-zA-Z]  Any single character in the range a-z or A-Z
^  Start of line
$  End of line
\A  Start of string
\z  End of string
.  Any single character
\s  Any whitespace character
\S  Any non-whitespace character
\d  Any digit
\D  Any non-digit
\w  Any word character (letter, number, underscore)
\W  Any non-word character
\b  Any word boundary character
(...)  Capture everything enclosed
(a|b)  a or b
a?  Zero or one of a
a*  Zero or more of a
a+  One or more of a
a{3}  Exactly 3 of a
a{3,}  3 or more of a
a{3,6}  Between 3 and 6 of a
```

# Ruby regular expressions: quick reference (cont'd)

```
options:
i case insensitive
m make dot match newlines
x ignore whitespace in regex
o perform #{...} substitutions only once
```

## longest match and shortest match (shortest match is faster)

```
"*" and "+" are longest match, "*?" and "+?" are shortest match
/<.*>/.match("<a><b><c>")  # => "<a><b><c>"
/<.*?>/.match("<a><b><c>") # => "<a>"
```

# exercise: plotting request counts over time

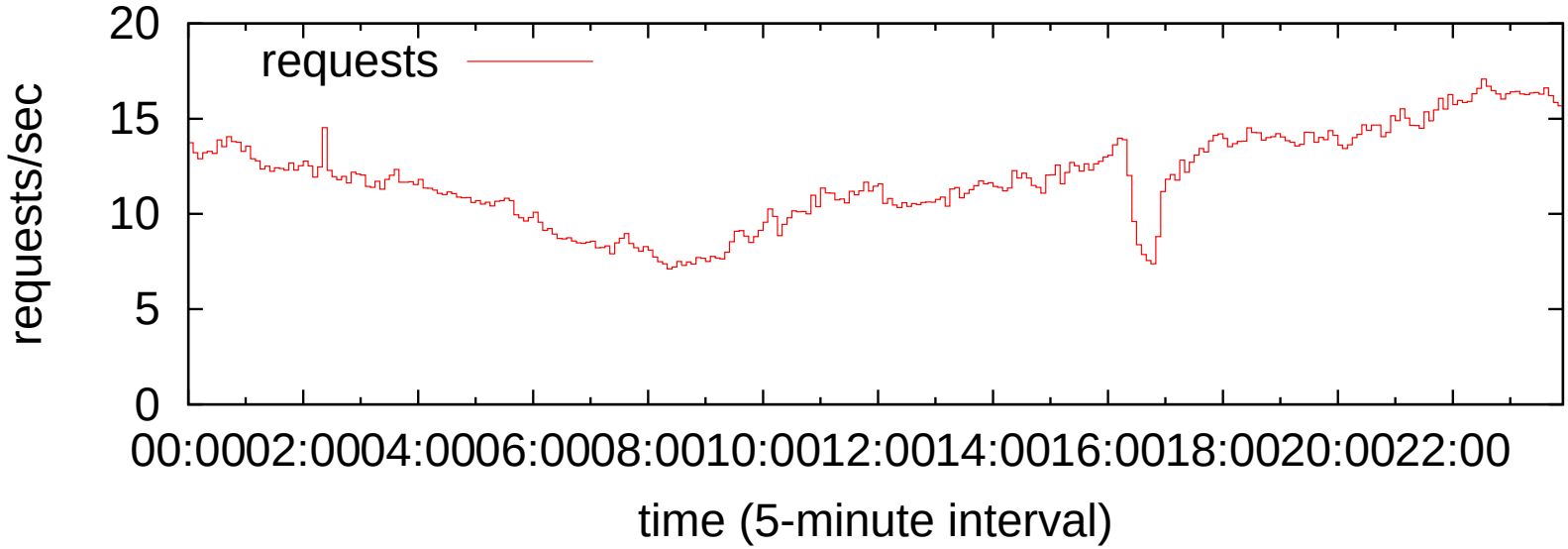
- use the sample data
- extract request counts and transferred bytes with 5 minutes bins
- plot the results

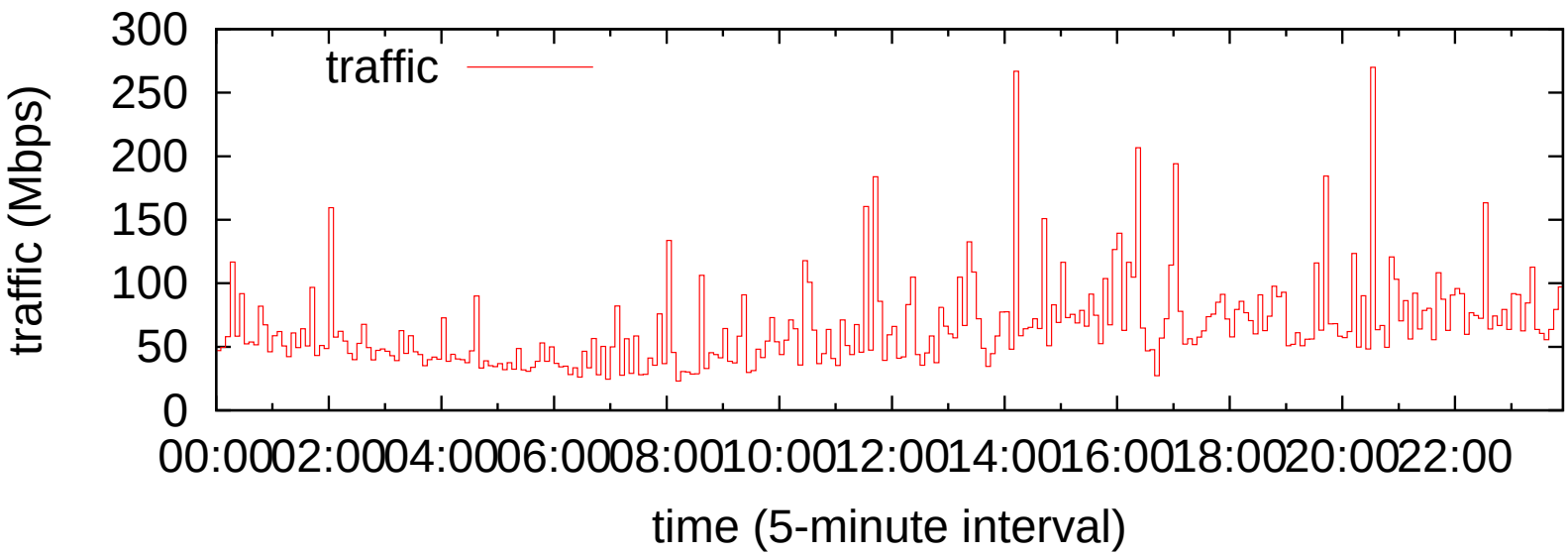
# extract request counts and transferred bytes with 5 minutes bins

```
#!/usr/bin/env ruby
require 'date'

# regular expression for apache common log format
#   host ident user time request status bytes
re = /^(\S+) (\S+) (\S+) \[(.*?)\] "(.*?)" (\d+) (\d+|-)/
timebins = Hash.new([0, 0])
count = parsed = 0
ARGF.each_line do |line|
  count += 1
  if re.match(line)
    host, ident, user, time, request, status, bytes = $~.captures
    # ignore if the status is not success (2xx)
    next unless /2\d{2}/.match(status)
    parsed += 1
    # parse timestamp
    ts = DateTime.strptime(time, '%d/%b/%Y:%H:%M:%S %z')
    # create the corresponding key for 5-minutes timebins
    rounded = sprintf("%02d", ts.min.to_i / 5 * 5)
    key = ts.strftime("%Y-%m-%dT%H:#{rounded}")
    # count by request and byte
    timebins[key] = [timebins[key][0] + 1, timebins[key][1] + bytes.to_i]
  else
    # match failed
    $stderr.puts("match failed at line #{count}: #{line.dump}")
  end
end
timebins.sort.each do |key, value|
  puts "#{key} #{value[0]} #{value[1]}"
end
$stderr.puts "parsed:#{parsed} ignored:#{count - parsed}"
```

# plot graphs of request counts and transferred bytes



# gnuplot script

- put 2 graphs together using multiplot

```
set xlabel "time (5-minute interval)"
set xdata time
set format x "%H:%M"
set timefmt "%Y-%m-%dT%H:%M"
set xrange ['2010-07-19T00:00':'2010-07-19T23:55']
set key left top

set multiplot layout 2,1

set yrange [0:20]
set ylabel "requests/sec"
plot "access-5min.txt" using 1:($2/300) title 'requests' with steps

set yrange [0:300]
set ylabel "traffic (Mbps)"
plot "access-5min.txt" using 1:($3*8/300/1000000) title 'traffic' with steps

unset multiplot
```

# summary

items left off from previous class

- how to make good graphs
- exercise: graph plotting by gnuplot

Data recording and log analysis

- data format
- log analysis methods
- exercise: log data and regular expression

# next class

Class 4 Measuring the speed of the Internet (10/19)

- bandwidth measurement
- inferring available bandwidth
- mean, standard deviation
- linear regression
- exercise: mean, standard deviation, linear regression
- **assignment 1**