Internet Measurement and Data Analysis (3)

Kenjiro Cho

2016-04-25

review of previous class

Data and variability (9/29)

- Summary statistics
- Sampling
- How to make good graphs
- exercise: computing summary statistics by Ruby
- exercise: graph plotting by Gnuplot

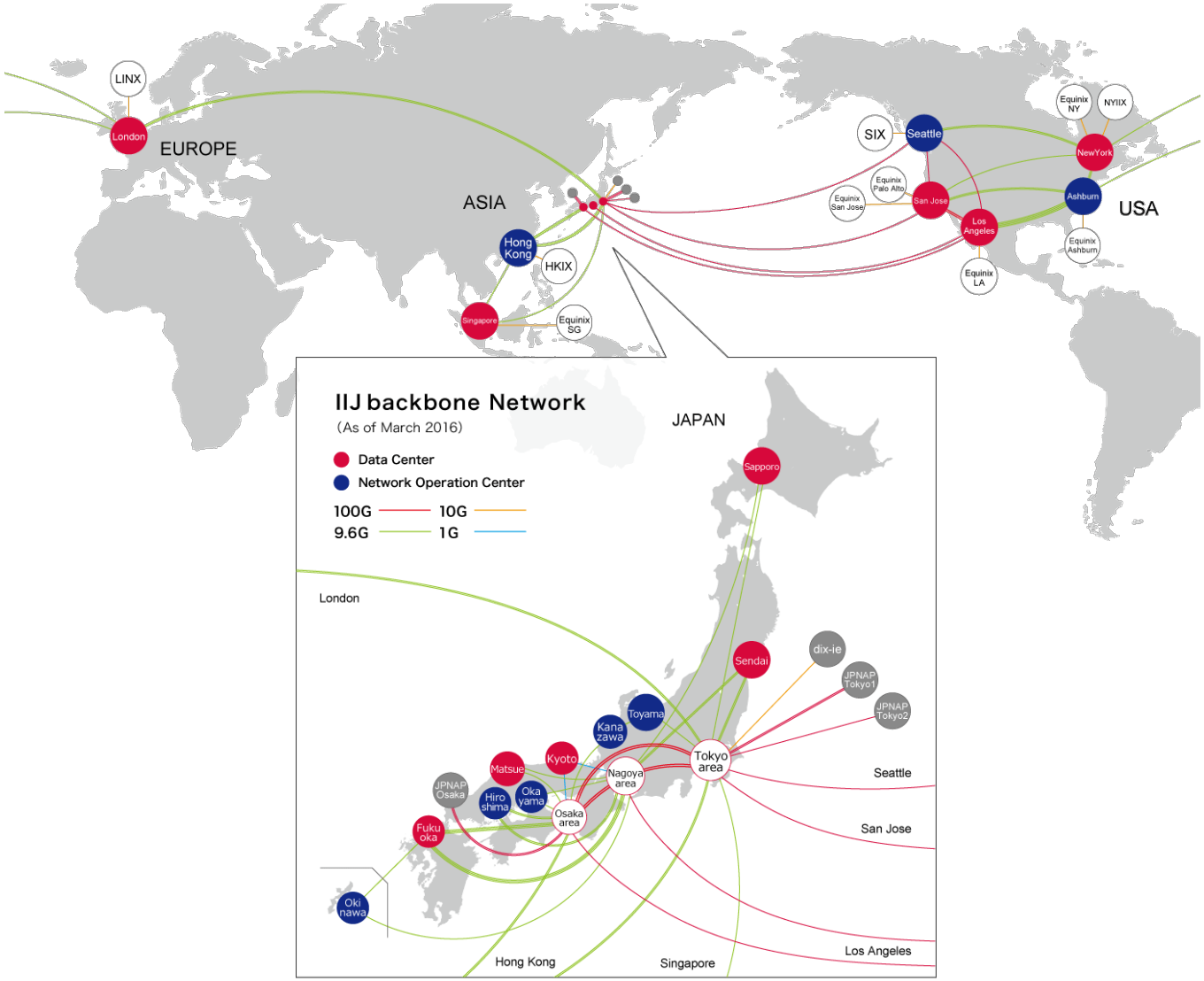
today's topics

Class 3 Data recording and log analysis

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

example network structure from a Japanese ISP

main facilities in Tokyo Osaka and Nagoya, connecting regional POPs with redundant configuration



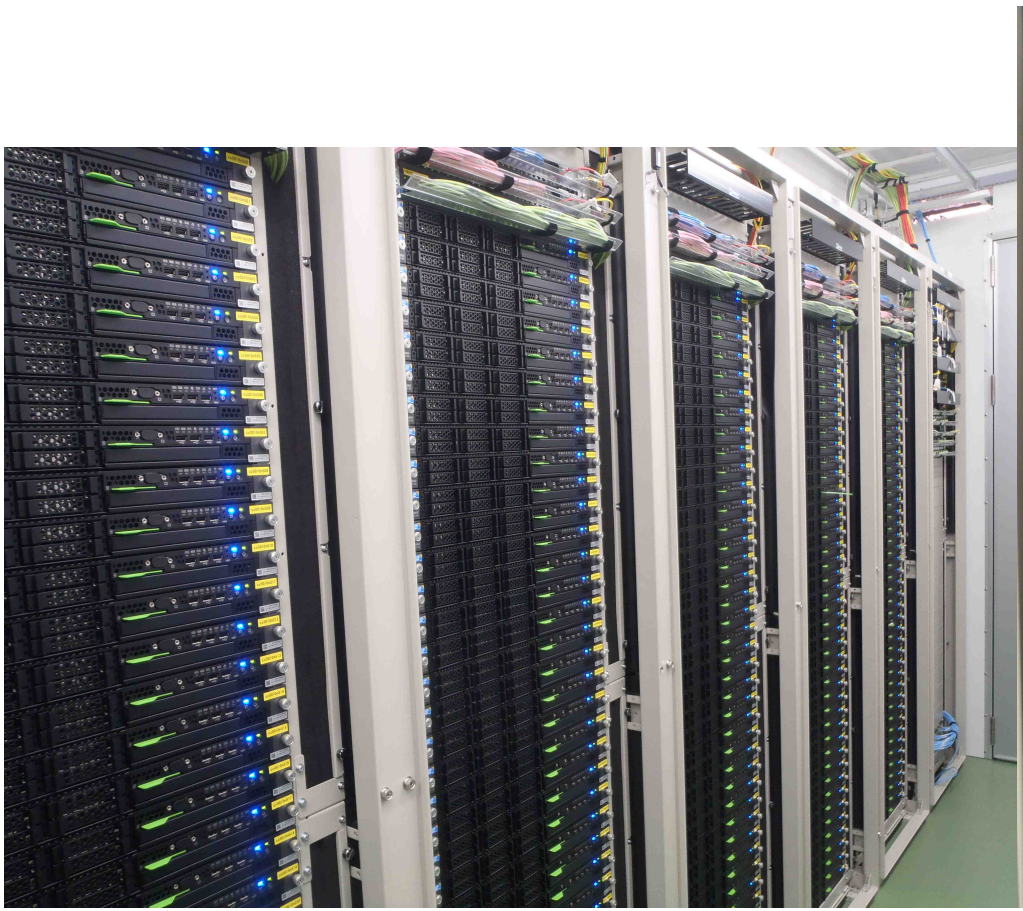
routers

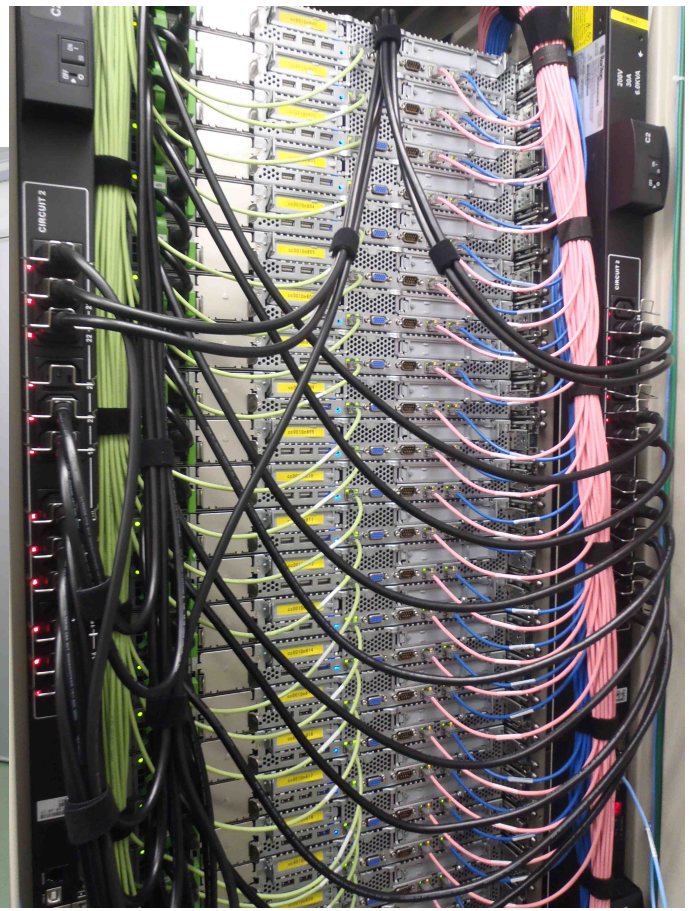
router: equipment to connect networks

- functions
  - routing, packet-forwarding, management
- classes of routers
  - core-routers, edge-routers, broadband routers, etc.

data centers

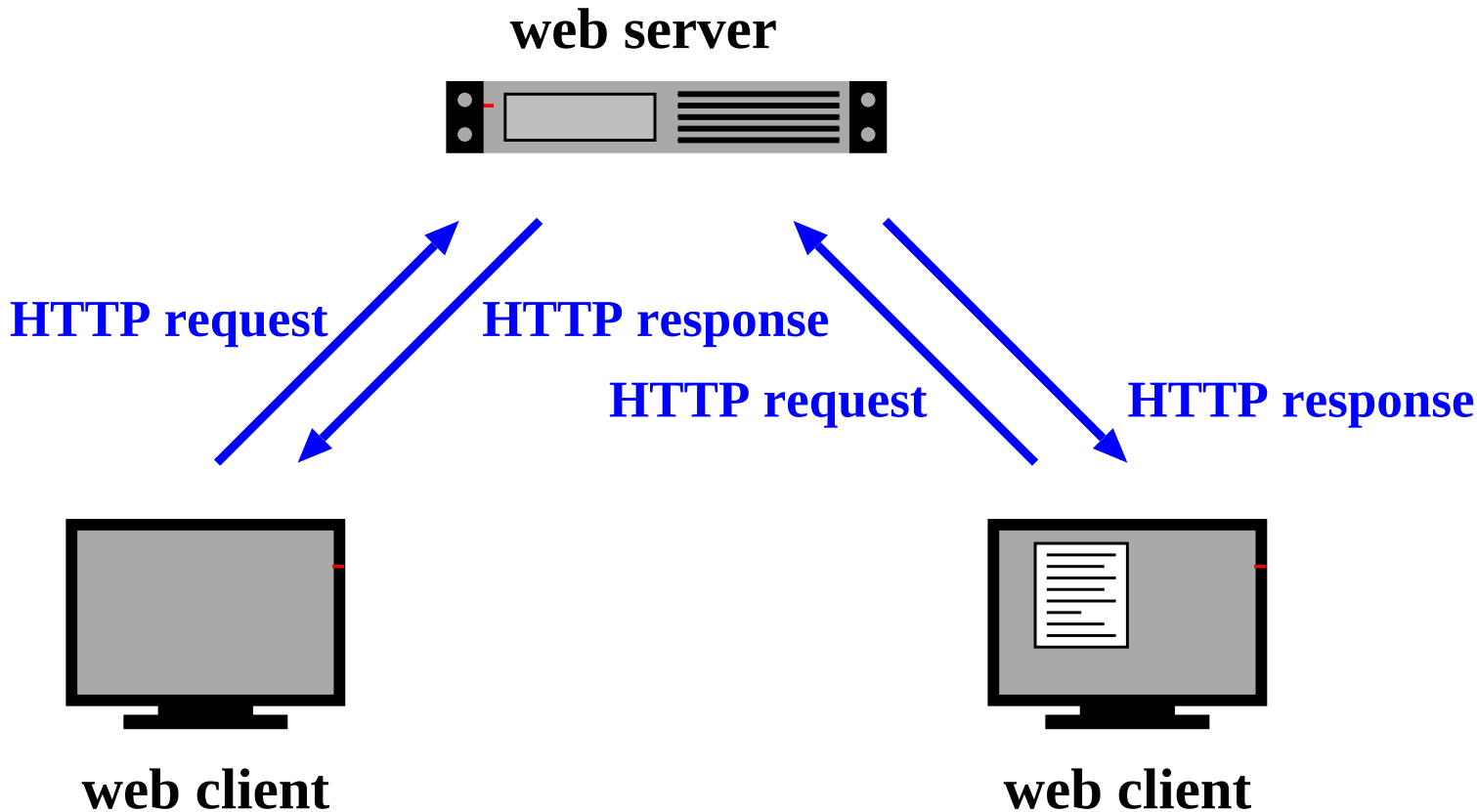
- facilities accommodating servers and communication equipment
- power supply, air conditioning, free-access floors, earthquake or other disaster resistant structures





access to a web server

- World Wide Web
  - URI: identifiers to specify resources on the Internet
  - HTML: mark up language for Web documents
  - HTTP: protocol to send and receive Web contents



Uniform Resource Identifier (URI)

- an identifier to specify a resource on the Internet
  - a reference to a resource, name, or other types of object
  - URL (Uniform Resource Locator): a reference to resource location, part of URI
- design philosophy of WWW: enables to specify any information

```
Example URIs:
      http://www.ietf.org/rfc/rfc2396.txt
      ftp://ftp.is.co.za/rfc/rfc1808.txt
      ldap://[2001:db8::7]/c=GB?objectClass?one
      mailto:John.Doe@example.com
      tel:+1-816-555-1212
      urn:oasis:names:specification:docbook:dtd:xml:4.1.2

Syntax:
        foo://example.com:8042/over/there?name=ferret#nose
        \_/   \______________/\_________/ \_________/ \__/
         |           |            |            |        |
      scheme     authority       path        query   fragment
         |   _____________________|__
        / \ /                        \
        urn:example:animal:ferret:nose
```

HyperText Markup Language (HTML)

- mark up language for Web documents
  - adds meta-data to elements in plain text
- HTML tags: markup elements enclosed by "<" and ">"

```
<!DOCTYPE html>
<html>
  <head>
    <title>sample title</title>
  </head>
  <body>
    <h1>Heading level 1</h1>
    <h2>Heading level 2</h2>

    <p>This is a paragraph.</p>

    <p>Another paragraph with
      <a href="http://www.keio.ac.jp/">a link to Keio</a>.
    </p>
    <img src="logo.gif" alt="keio logo" width="50" height="50">
  </body>
</html>
```

HyperText Transfer Protocol (HTTP)

- protocol to send and receive Web contents
  - a text-based protocol on top of TCP

Client request:

```
GET /index.html HTTP/1.1
Host: www.example.com
Referer: http://www.example.co.jp/somepage.html
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.9; rv:28.0) Gecko/20100101 Firefox/28.0
```

Server Response:

```
HTTP/1.1 200 OK
Date: Mon, 23 May 2005 22:38:34 GMT
Server: Apache/1.3.3.7 (Unix) (Red-Hat/Linux)
Last-Modified: Wed, 08 Jan 2003 23:11:55 GMT
ETag: "3f80f-1b6-3e1cb03b"
Content-Type: text/html; charset=UTF-8
Content-Length: 131
Accept-Ranges: bytes
Connection: close

<html>
<head>
  <title>An Example Page</title>
</head>
<body>
  Hello World, this is a very simple HTML document.
</body>
</html>
```

data log

- records automatically generated by computers
- network logs
  - routers/switches, assigned IP addresses, packets, usage, etc.
- Internet service logs
  - Web access, mail delivery, firewalls, etc.
- broader logs
  - online user behaviors, location information, automobile records, etc.

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
  - find 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

- to provied services customized to a specific user

problems in log analysis

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

- privacy issues

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

web crawlers

data collection by crawlers

- crawler: programs to automatically collect data from many places
- web crawlers: automatically visit web pages and collect data
  - to create database and indices for search engines
  - move to next page by following links in the visiting page
- many existing tools
  - e.g., Ruby's Mechanize
  - note: rapid crawling is often considered as attacks

scraper

- extracts necessary information by analyzing HTML documents
- many existing tools
  - e.g., Ruby's Nokogiri

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

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>"
```

parentheses for grouping and capturing

```
expressions inside "(" and ")" are grouped and captured
group: e.g. "(Alice|Bob)"  # "Alice" or "Bob"
capture: matched string is captured, and subsequently referred by "$N" N=1..
non-capturing groups: "(?:regexp)", e.g., "(\d+(?:\.\d+)?)" for decimals
        can be used to simplify capture numbering in nested groups
```

previous exercise: computing summary statistics

- mean
- standard deviation
- median

- finish-time data of a city marathon: from P. K. Janert "Gnuplot in Action"

`http://web.sfc.keio.ac.jp/~kjc/classes/sfc2016s-measurement/marathon.txt`

```
% head marathon.txt
# Minutes Count
133 1
134 7
135 1
136 4
137 3
138 3
141 7
142 24
143 13
```

previous exercise: computing mean

- read finish-time(in minutes) and the number of finishers from each line, sum up the product, and finally divide it by the total number of finishers

```
# regular expression to read minutes and count
re = /^(\d+)\s+(\d+)/

sum = 0          # sum of data
n = 0            # the number of data
ARGF.each_line do |line|
    if re.match(line)
      min = $1.to_i
      cnt = $2.to_i
      sum += min * cnt
      n += cnt
    end
end

mean = Float(sum) / n

printf "n:%d mean:%.1f\n", n, mean
```

```
% ruby mean.rb marathon.txt
n:2355 mean:171.3
```

previous exercise: computing standard deviation

- algorithm: $\sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2$

```
# regular expression to read minutes and count
re = /^(\d+)\s+(\d+)/

data = Array.new
sum = 0          # sum of data
n = 0            # the number of data
ARGF.each_line do |line|
    if re.match(line)
      min = $1.to_i
      cnt = $2.to_i
      sum += min * cnt
      n += cnt
      for i in 1 .. cnt
        data.push min
      end
    end
end
mean = Float(sum) / n
sqsum = 0.0
data.each do |i|
  sqsum += (i - mean)**2
end
var = sqsum / n
stddev = Math.sqrt(var)
printf "n:%d mean:%.1f variance:%.1f stddev:%.1f\n", n, mean, var, stddev
```

```
% ruby stddev.rb marathon.txt
n:2355 mean:171.3 variance:199.9 stddev:14.1
```

previous exercise: computing standard deviation in one-pass

- one-pass algorithm: $\sigma^2 = \frac{1}{n}\sum_{i=1}^{n} x_i^2 - \bar{x}^2$

```
# regular expression to read minutes and count
re = /^(\d+)\s+(\d+)/

sum = 0          # sum of data
n = 0            # the number of data
sqsum = 0        # su of squares
ARGF.each_line do |line|
    if re.match(line)
      min = $1.to_i
      cnt = $2.to_i
      sum += min * cnt
      n += cnt
      sqsum += min**2 * cnt
    end
end

mean = Float(sum) / n
var = Float(sqsum) / n - mean**2
stddev = Math.sqrt(var)

printf "n:%d mean:%.1f variance:%.1f stddev:%.1f\n", n, mean, var, stddev
```

```
% ruby stddev2.rb marathon.txt
n:2355 mean:171.3 variance:199.9 stddev:14.1
```

previous exercise: computing median

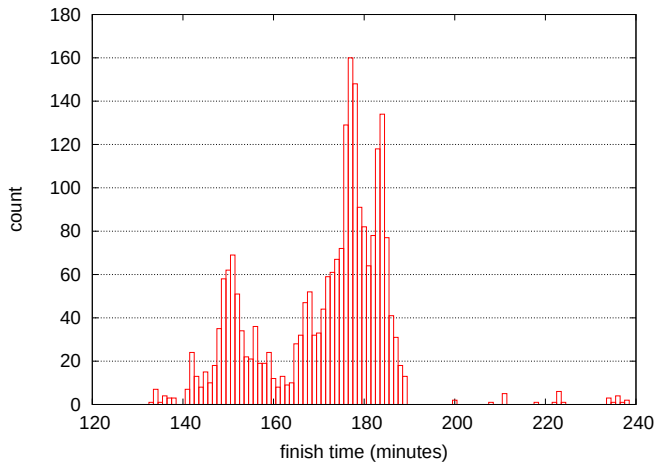
- create an array of each finish time, sort the array by value, and extract the central value

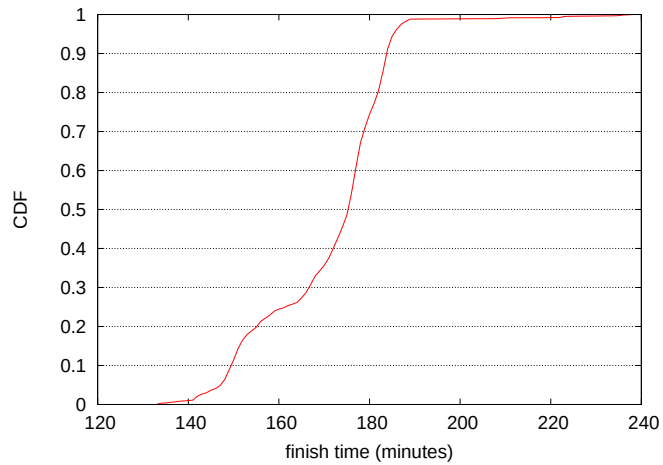
```
# regular expression to read minutes and count
re = /^(\d+)\s+(\d+)/

data = Array.new

ARGF.each_line do |line|
    if re.match(line)
      min = $1.to_i
      cnt = $2.to_i
      for i in 1 .. cnt
        data.push min
      end
    end
end

data.sort!              # just in case data is not sorted

n = data.length         # number of array elements
r = n / 2               # when n is odd, n/2 is rounded down
if n % 2 != 0
  median = data[r]
else
  median = (data[r - 1] + data[r])/2
end

printf "r:%d median:%d\n", r, median
```

```
% ruby median.rb marathon.txt
r:1177 median:176
```

previous exercise: gnuplot

- plotting simple graphs using gnuplot
  - to intuitively understand the data





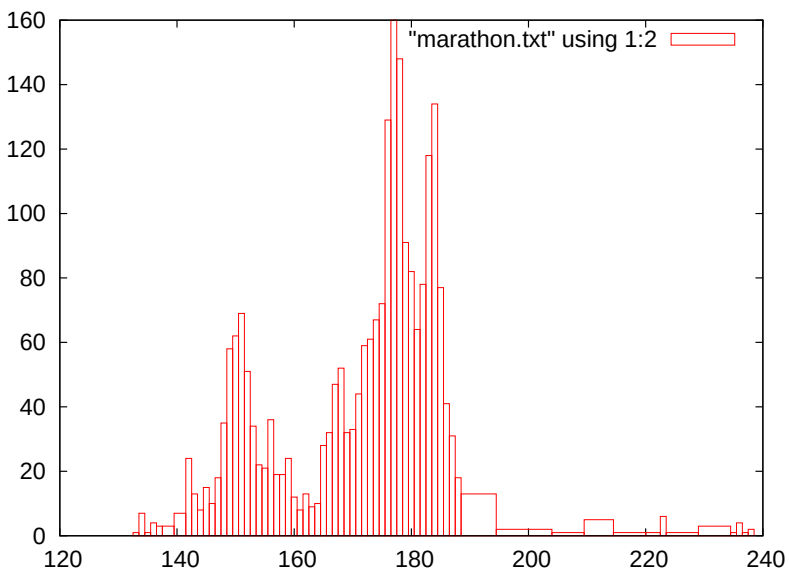
histogram

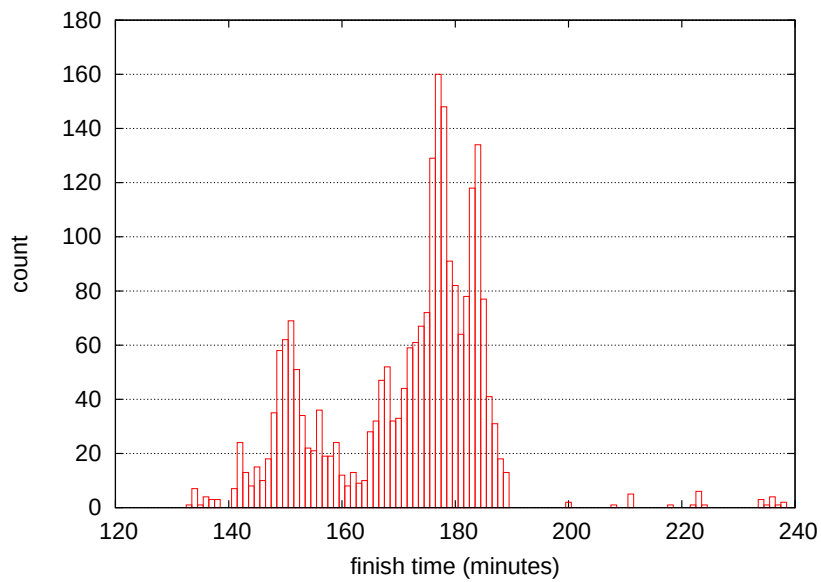
- distribution of finish time of a city marathon

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

make the plot look better (right)

```
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
```





previous exercise: plotting CDF of finish-time

original data:

```
# Minutes Count
133 1
134 7
135 1
136 4
137 3
138 3
141 7
142 24
...
```

add cumulative count:

```
# Minutes Count CumulativeCount
133 1 1
134 7 8
135 1 9
136 4 13
137 3 16
138 3 19
141 7 26
142 24 50
...
```

previous exercise: CDF (2)

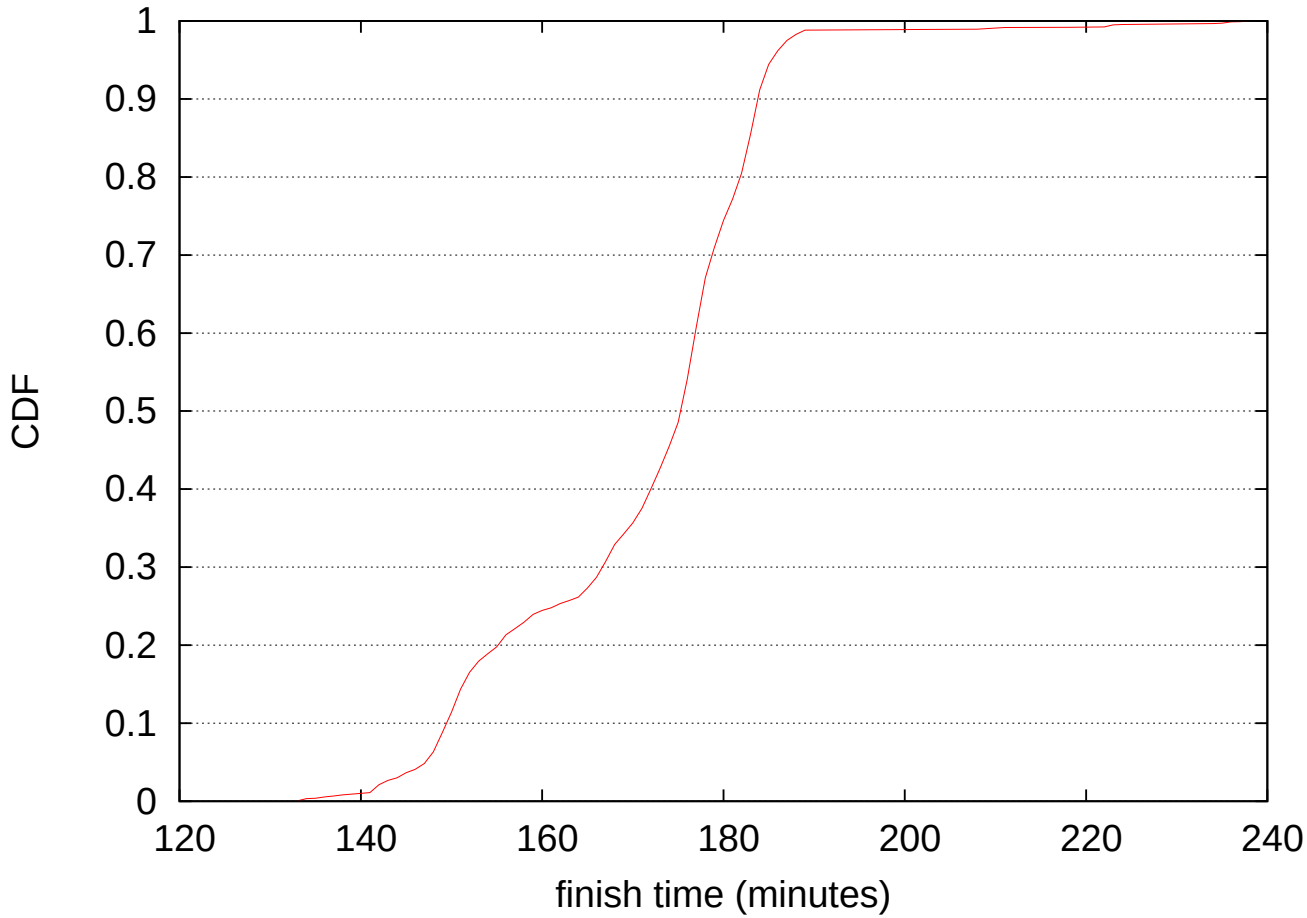
ruby code:

```
re = /^(\d+)\s+(\d+)/
cum = 0
ARGF.each_line do |line|
  begin
    if re.match(line)
      # matched
      time, cnt = $~.captures
      cum += cnt.to_i
      puts "#{time}\t#{cnt}\t#{cum}"
    end
  end
end
```

gnuplot command:

```
set xlabel "finish time (minutes)"
set ylabel "CDF"
set grid y
plot "marathon-cdf.txt" using 1:($3 / 2355) with lines notitle
```

CDF plot of finish-time of city marathon



previous exercise: saving a plot to an image file

to specify an image format and save to a file:

```
gnuplot> set terminal png
gnuplot> set output "plotfile.png"
gnuplot> replot
```

to run a script:

```
gnuplot> load "scriptfile"
```

to exit:

```
gnuplot> exit
```

today's exercise: web access log sample data

- apache log (combined log format)
- from a JAIST server, access log for 24 hours
- about 20MB (zip compressed), about 162MB after unzip
- 1/10 sampling
- client IP addresses are anonymized for privacy
  - using "ipv6loganon –anonymize-careful"

```
access log for 24 hours:
 http://www.iijlab.net/~kjc/classes/sfc2016s-measurement/sample_access_log.zip
```

sample data

```
117.136.16.0 - - [01/Oct/2013:23:59:58 +0900] "GET /project/morefont/liangqiushengshufaziti.apk \
  HTTP/1.1" 200 524600 "-" "-" jaist.dl.sourceforge.net
218.234.160.0 - - [01/Oct/2013:23:59:59 +0900] "GET /pub/Linux/linuxmint/packages/dists/olivia/\
  upstream/i18n/Translation-ko.xz HTTP/1.1" 404 564 "-" "Debian APT-HTTP/1.3 (0.9.7.7ubuntu4)" \
  ftp.jaist.ac.jp
119.80.32.0 - - [01/Oct/2013:23:59:59 +0900] "GET /project/morefont/xiongtuti.apk HTTP/1.1" 304 \
  132 "-" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; Foxy/1; InfoPath.1)" \
  jaist.dl.sourceforge.net
218.234.160.0 - - [02/Oct/2013:00:00:00 +0900] "GET /pub/Linux/linuxmint/packages/dists/olivia/\
  import/i18n/Translation-en.gz HTTP/1.1" 404 562 "-" "Debian APT-HTTP/1.3 (0.9.7.7ubuntu4)" \
  ftp.jaist.ac.jp
117.136.0.0 - - [02/Oct/2013:00:00:00 +0900] "GET /project/morefont/xiaoqingwaziti.apk HTTP/1.1"\
  200 590136 "-" "-" jaist.dl.sourceforge.net
123.224.224.0 - - [02/Oct/2013:00:00:00 +0900] "GET /pub/Linux/ubuntu/dists/raring/main/i18n/\
  Translation-en.bz2 HTTP/1.1" 304 187 "-" "Debian APT-HTTP/1.3 (0.9.7.7ubuntu4)" ftp.jaist.ac.jp
123.224.224.0 - - [02/Oct/2013:00:00:00 +0900] "GET /pub/Linux/ubuntu/dists/raring/multiverse/\
  i18n/Translation-en.bz2 HTTP/1.1" 304 186 "-" "Debian APT-HTTP/1.3 (0.9.7.7ubuntu4)" \
  ftp.jaist.ac.jp
124.41.64.0 - - [01/Oct/2013:23:59:58 +0900] "GET /ubuntu/pool/universe/s/shorewall6/\
  shorewall6_4.4.26.1-1_all.deb HTTP/1.1" 200 435975 "-" "Wget/1.14 (linux-gnu)" ftp.jaist.ac.jp
...
240b:10:c140:a909:a949:4291:c02d:5d13 - - [02/Oct/2013:00:00:01 +0900] "GET /ubuntu/pool/main/m/\
  manpages/manpages_3.52-1ubuntu1_all.deb HTTP/1.1" 200 626951 "-" \
  "Debian APT-HTTP/1.3 (0.9.7.7ubuntu4)" ftp.jaist.ac.jp
...
```

exercise: plotting request counts over time

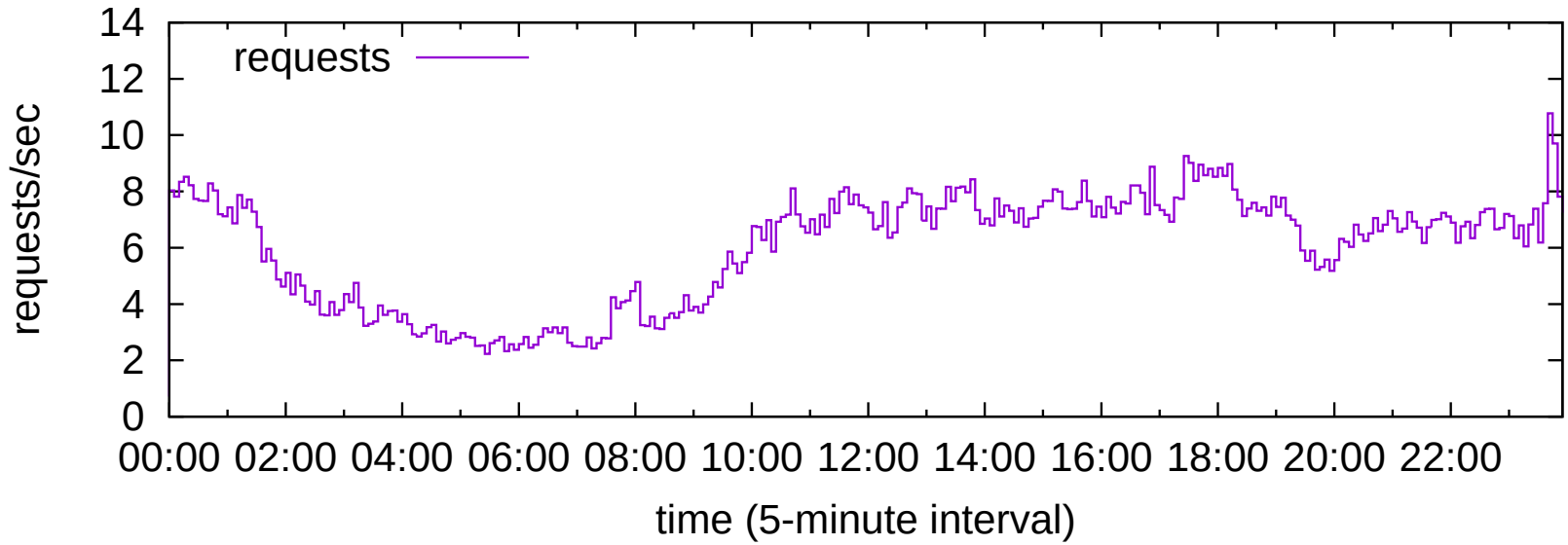
- use the sample data
- extract request counts and transferred bytes with 5 minutes bins
- plot the results

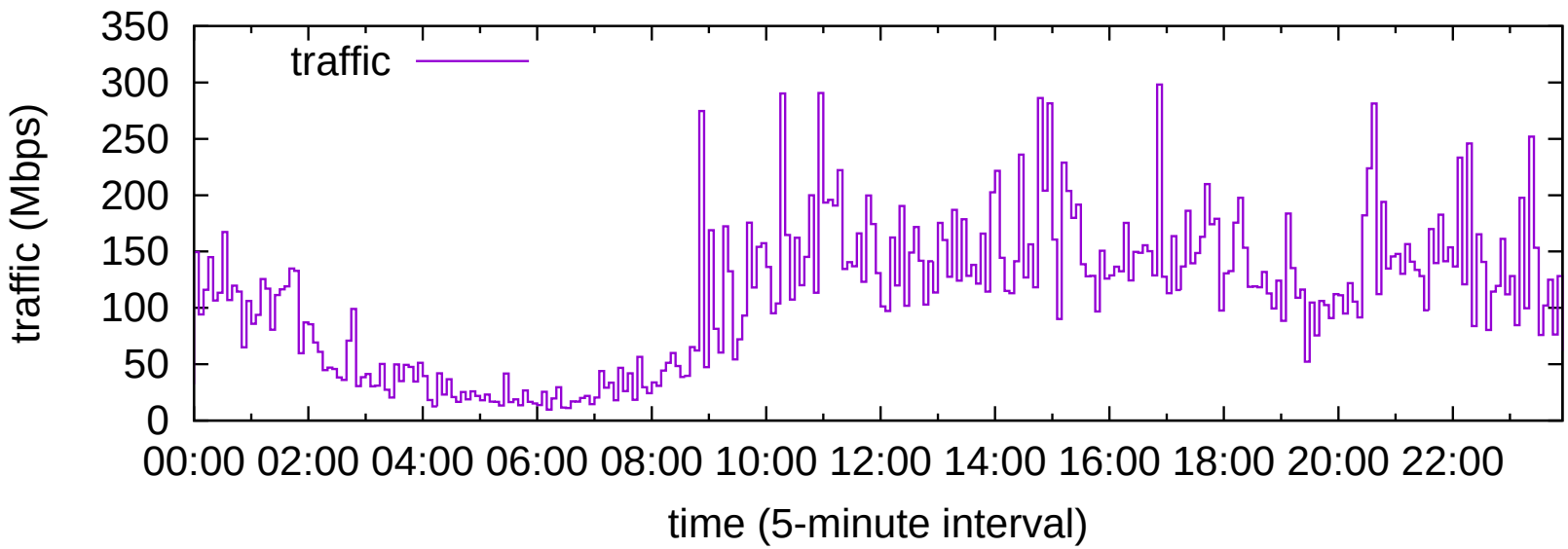
```
% ruby parse_accesslog.rb sample_access_log > access-5min.txt
% more access-5min.txt
2013-10-01T20:00 1 1444348221
...
2013-10-01T23:55 215 1204698404
2013-10-02T00:00 2410 5607857319
2013-10-02T00:05 2344 3528532804
2013-10-02T00:10 2502 4354264670
2013-10-02T00:15 2555 5441105487
...
% gnuplot
gnuplot> load 'access.plt'
```

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

    next unless request.match(/GET\s.*/) # ignore if the request is not "GET"
    next unless status.match(/2\d{2}/) # ignore if the status is not success (2xx)
    parsed += 1
    # parse timestamp
    ts = DateTime.strptime(time, '%d/%b/%Y:%H:%M:%S')
    # 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 ['2013-10-02T00:00':'2013-10-02T23:55']
set key left top

set multiplot layout 2,1

set yrange [0:14]
set ylabel "requests/sec"
plot    "access-5min.txt" using 1:($2/300) title 'requests' with steps

set yrange [0:350]
set ylabel "traffic (Mbps)"
plot    "access-5min.txt" using 1:($3*8/300/1000000) title 'traffic' with steps

unset multiplot
```

summary

Class 3 Data recording and log analysis

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

next class

Class 4 Distribution and confidence intervals (5/2)

- Normal distribution
- Confidence intervals and statistical tests
- Distribution generation
- exercise: confidence intervals
- **assignment 1**

appendix: useful UNIX commands

- convenient UNIX commands for handling text files
  - sort, head, tail, cat, cut
  - diff, tee, grep, uniq, wc
  - join, find, sed, awk, screen

- for Windows, you need to install Gow (Gnu on Windows) or other tools

sort

sort command: sort lines of text files

```
$ sort [options] [FILE ...]
```

- options (useful for exercises)
  - -n : evaluate fields as a numerical value
  - -r : reverse the results
  - -k POS1[,POS2] : start a key at POS1, end it at POS 2 (origin 1)
  - -t SEP : use SEP as a separator
  - -m : merge already sorted files; do not sort
  - -T DIR : use DIR for a temporary directory

example: sort "file" in the reverse order by the numerical value of the 3rd field, use "/usr/tmp" as a temporary directory

```
$ sort -nr -k3,3 -T/usr/tmp file
```

head

head command: display first lines of a file

- shows the first 10 lines by default

```
head [-n lines | -c bytes] [file ...]
```

example:

```
$ sort -nr -k3,3 file | head -n 10
```

tail

tail command: display last lines of a file

- shows the last 10 lines by default

```
tail [-F | -f | -r] [-q] [-b number | -c number | -n number] [file ...]
```

- useful options
  - -f : watch the file and show lines appended to the file

example:

```
monitor a log file:
$ tail -f /var/log/httpd-access.log
```

cat

cat command: concatenate and print files

```
cat [-benstuv] [file ...]
```

example:

```
$ cat file1 file2 > file3
```

cut

cut command: cut out selected portions of each line of a file

```
cut -b list [-n] [file ...],
cut -c list [file ...],
cut -f list [-s] [-d delim] [file ...]
```

- useful optins
  - -b BYTE-LIST : specifies byte positions
  - -c CHAR-LIST : specifies character positions
  - -f FIELD-LIST : specifies field positions
  - -d DELIM : use DELIM as the field delimiter character

example:

```
extract users' login names and shells from the system passwd file:
$ cut -d : -f 1,7 /etc/passwd
show the names and login times of the currently logged in users:
$ who | cut -c 1-16,26-38
```

diff

diff command: compare files line by line

```
diff [OPTION]... FILES
```

- useful options
  - -u : use the unified diff format

example:

```
$ diff -u file1 file2
```

tee

tee command: duplicate standard input

```
tee [-ai] [file ...]
```

example:

```
$ ls | tee output.txt
```

grep

grep command: print lines matching a pattern

```
grep [options] PATTERN [FILE...]
grep [options] [-e PATTERN | -f FILE] [FILE...]
```

example:

```
search lines including 'abc':
$ grep 'abc' file
count the number of lines starting with 'abc':
$ grep -c '^abc' file
```

uniq

uniq command: filter out repeated lines in a file

```
uniq [-c | -d | -u] [-i] [-f num] [-s chars] [input_file [output_file]]
```

- useful options
  - -d : only output lines that are repeated in the input

example:

```
$ cat file1 file2 | sort | uniq > file3
```

```
$ sort file | uniq -d
```

wc

wc command: show word, line, and character counts of a file

```
wc [-Lclmw] [file ...]
```

join

join command: join lines of specified files which are already sorted by a common field

```
join [-a file_number | -v file_number] [-e string] [-o list] [-t char]
    [-1 field] [-2 field] file1 file2
```

examples:

```
$ cat file1
1001    orange
1002    apple
1003    grape
$ cat file2
1001    400
1002    250
1004    500
$ join file1 file2
1001 orange 400
1002 apple 250
$ join -a1 -a2 -e NULL -o '0,1.2,2.2' file1 file2
1001 orange 400
1002 apple 250
1003 grape NULL
1004 NULL 500
```

find

find command: walk a file hierarchy

```
find [-H | -L | -P] [-EXdsx] [-f pathname] pathname ... expression
find [-H | -L | -P] [-EXdsx] -f pathname [pathname ...] expression
```

example:

```
print files with ".rej" suffix:
$ find . -name "*.rej" -print

print ".o" files older than 1 year
$ find . -name "*.o" -mtime +365 -print

remove empty files:
$ find . -empty -exec rm {} \;
```

sed (streaming editor)

sed command:

```
sed [-Ealn] command [file ...]
sed [-Ealn] [-e command] [-f command_file] [-I extension]
    [-i extension] [file ...]
```

- useful options
  - -e command : append the command
  - -f command_file : append the command found in the file

example:

```
replace "old" by "new":
$ echo "old songs in old books" | sed 's/old/new/g'

print line 3-5:
$ sed -n '3,5p' file
```

awk

awk command:

- pattern-directed scanning and processing language
- useful for writing a one-line program

```
awk [ -F fs ] [ -v var=value ] [ 'prog' | -f progfile ] [ file ...  ]
```

example:

```
swap column1 and colimn2 and add sum to column3:
$ echo "12 56" | awk '{print $2,$1,$1+$2}'

extract the capacity in percent from the df command:
$ df | awk 'match($0, /[0-9]+%/) {print substr($0, RSTART, RLENGTH - 1)}'
```

screen

screen command: screen manager (this isn't a built-in command)

- you can use multile virtual terminals in a single terminal
- with a feature to detach a virtual terminal
  - you can detach a virtual terminal running a job to run it in background, and later, re-attach the detached virtual terminal
    - screen : invoke screen
    - "ctrl-a d" : detach the current virtual terminal
    - screen -r : re-attach the detached virtual terminal