

Internet Measurement and Data Analysis (3)

Kenjiro Cho

2013-10-09

review of previous class

Class 2 Data and variability (10/02)

- ▶ 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

- ▶ Network management tools
- ▶ Data format
- ▶ Log analysis methods
- ▶ exercise: log data and regular expression

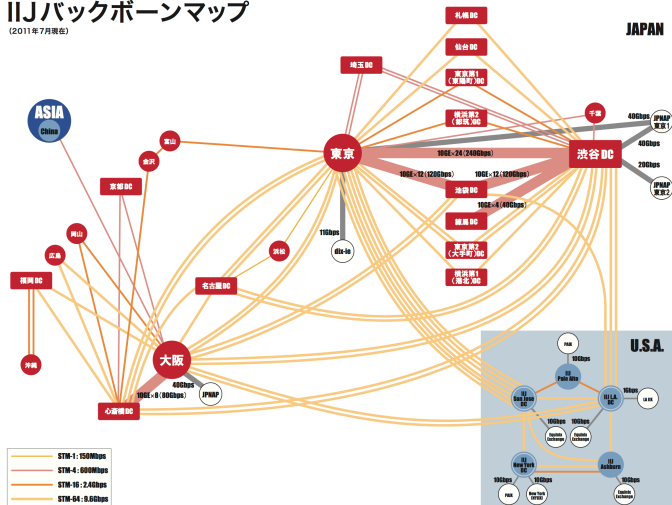
network management tools

example network structure from a Japanese ISP

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

IIJバックボーンマップ

(2011年7月現在)



routers

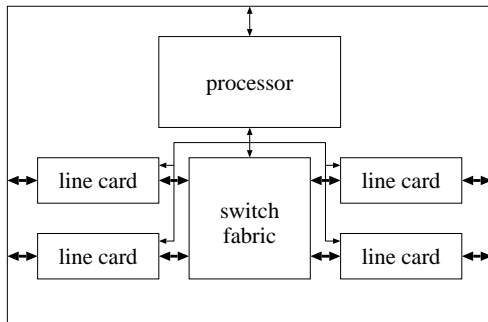
router: equipment to connect networks

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



router architecture

- ▶ fast path: hardware assisted processing
- ▶ slow path: software processing
 - ▶ ICMP packets are processed via slow path



commonly-used management tools

network management tools (originally not designed for measurement)

- ▶ ping
 - ▶ reachability, round-trip time
- ▶ traceroute
 - ▶ path detection
- ▶ tcpdump
 - ▶ packet capturing
- ▶ SNMP
 - ▶ usage monitoring, network equipment status monitoring

ping

- ▶ a popular and widely-available tool to check connectivity
- ▶ ICMP-echo request/reply
- ▶ limitations
 - ▶ ping responses do not mean network is working correctly
 - ▶ ICMP is not representative of host/network performance

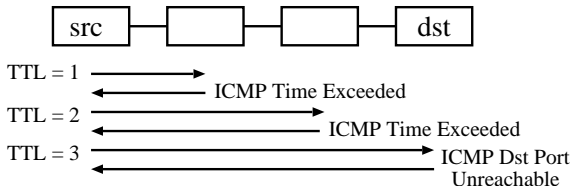
ping sample output

```
% ping -c 10 www.ait.ac.th
PING www.ait.ac.th (202.183.214.46): 56 data bytes
64 bytes from 202.183.214.46: icmp_seq=0 ttl=114 time=112.601 ms
64 bytes from 202.183.214.46: icmp_seq=1 ttl=114 time=106.730 ms
64 bytes from 202.183.214.46: icmp_seq=2 ttl=114 time=106.173 ms
64 bytes from 202.183.214.46: icmp_seq=3 ttl=114 time=111.704 ms
64 bytes from 202.183.214.46: icmp_seq=4 ttl=114 time=112.412 ms
64 bytes from 202.183.214.46: icmp_seq=5 ttl=114 time=114.603 ms
64 bytes from 202.183.214.46: icmp_seq=6 ttl=114 time=111.755 ms
64 bytes from 202.183.214.46: icmp_seq=7 ttl=114 time=115.273 ms
64 bytes from 202.183.214.46: icmp_seq=8 ttl=114 time=106.525 ms
64 bytes from 202.183.214.46: icmp_seq=9 ttl=114 time=111.562 ms

--- www.ait.ac.th ping statistics ---
10 packets transmitted, 10 packets received, 0% packet loss
round-trip min/avg/max/stddev = 106.173/110.934/115.273/3.142 ms
```

traceroute

- ▶ exploit TTL (time-to-live) of IP designed for loop prevention
 - ▶ TTL is decremented by each intermediate router
 - ▶ router returns ICMP TIME EXCEEDED to the sender when TTL becomes 0
- ▶ limitations
 - ▶ path may change over time
 - ▶ path may be asymmetric
 - ▶ can observe only out-going paths
 - ▶ report from one of the interfaces of the router
 - ▶ hard to identify interfaces belonging to same router



traceroute sample output

```
% traceroute www.ait.ac.th
traceroute to www.ait.ac.th (202.183.214.46), 64 hops max, 40 byte packets
 1 202.214.86.129 (202.214.86.129) 0.687 ms 0.668 ms 0.730 ms
 2 jc-gw0.IIJ.Net (202.232.0.237) 0.482 ms 0.390 ms 0.348 ms
 3 tky001ix07.IIJ.Net (210.130.143.233) 0.861 ms 0.872 ms 0.729 ms
 4 tky001bb00.IIJ.Net (210.130.130.76) 10.107 ms 1.026 ms 0.855 ms
 5 tky001ix04.IIJ.Net (210.130.143.53) 1.111 ms 1.012 ms 0.980 ms
 6 202.232.8.142 (202.232.8.142) 1.237 ms 1.214 ms 1.120 ms
 7 ge-1-1-0.tokenf-cr2.ix.singtel.com (203.208.172.209) 1.338 ms 1.501 ms
 1.480 ms
 8 p6-13.sngtp-cr2.ix.singtel.com (203.208.173.93) 93.195 ms 203.208.172.
229 (203.208.172.229) 88.617 ms 87.929 ms
 9 203.208.182.238 (203.208.182.238) 90.294 ms 88.232 ms 203.208.182.234
(203.208.182.234) 91.660 ms
10 203.208.147.134 (203.208.147.134) 103.933 ms 104.249 ms 103.986 ms
11 210.1.45.241 (210.1.45.241) 103.847 ms 110.924 ms 110.163 ms
12 st1-6-bkk.csloxinfo.net (203.146.14.54) 131.134 ms 129.452 ms 111.408
ms
13 st1-6-bkk.csloxinfo.net (203.146.14.54) 106.039 ms 105.078 ms 105.196
ms
14 202.183.160.121 (202.183.160.121) 111.240 ms 123.606 ms 112.153 ms
15 * * *
16 * * *
17 * * *
```

tcpdump

- ▶ packet capturing tool
 - ▶ capture the first N bytes of packets
- ▶ flexible filtering
 - ▶ e.g., capture only TCP SYN from host X
- ▶ enables detailed analysis
- ▶ limitations
 - ▶ huge volume
 - ▶ difficult to capture on high-speed links

tcpdump sample output

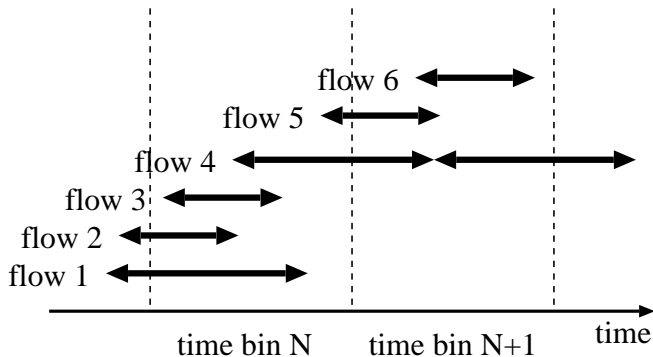
```
18:45:29.767497 IP 202.214.86.132.50052 > 202.210.220.18.80: \  
  S 3304970307:3304970307(0) win 65535 <mss 1460,nop,nop,sackOK,nop, \  
  wscale 1,nop,nop,timestamp 710778973 0>  
18:45:29.770038 IP 202.210.220.18.80 > 202.214.86.132.50052: \  
  S 3129218301:3129218301(0) ack 3304970308 win 65535 <mss 1460,nop, \  
  ywscale 1,nop,nop,timestamp 2523776361 710778973,nop,nop,sackOK>  
18:45:29.770090 IP 202.214.86.132.50052 > 202.210.220.18.80: \  
  . ack 1 win 33304 <nop,nop,timestamp 710778973 2523776361>  
18:45:29.787084 IP 202.214.86.132.50052 > 202.210.220.18.80: \  
  P 1:521(520) ack 1 win 33304 <nop,nop,timestamp 710778975 2523776361>  
18:45:29.791392 IP 202.210.220.18.80 > 202.214.86.132.50052: \  
  P 1:222(221) ack 521 win 33304 <nop,nop,timestamp 2523776363 710778975>  
18:45:29.887024 IP 202.214.86.132.50052 > 202.210.220.18.80: \  
  . ack 222 win 33304 <nop,nop,timestamp 710778985 2523776363>  
18:45:34.792726 IP 202.210.220.18.80 > 202.214.86.132.50052: \  
  F 222:222(0) ack 521 win 33304 <nop,nop,timestamp 2523776864 710778985>  
18:45:34.792763 IP 202.214.86.132.50052 > 202.210.220.18.80: \  
  . ack 223 win 33304 <nop,nop,timestamp 710779475 2523776864>  
18:45:42.528539 IP 202.214.86.132.50052 > 202.210.220.18.80: \  
  F 521:521(0) ack 223 win 33304 <nop,nop,timestamp 710780249 2523776864>  
18:45:42.531088 IP 202.210.220.18.80 > 202.214.86.132.50052: \  
  . ack 522 win 33303 <nop,nop,timestamp 2523777637 710780249>
```

SNMP (Simple Network Management Protocol)

- ▶ SNMP allows a remote user to
 - ▶ query information, store information, set traps
 - ▶ by UDP (unreliable)
- ▶ standardized set of traffic statistics
 - ▶ supported by most of routers, switches, host OS
 - ▶ many management/monitoring products
- ▶ MIB (Management Information Base)
 - ▶ tree structured database of SNMP objects
 - ▶ e.g., interfaces.ifTable.ifEntry.ifOutOctets
 - ▶ standard MIBs and private MIBs
 - ▶ get, set, get-next to access MIB
- ▶ limitations
 - ▶ supported statistics are limited
 - ▶ most counter statistics are hard-coded, e.g., interface counters
 - ▶ accessing to MIB objects is expensive

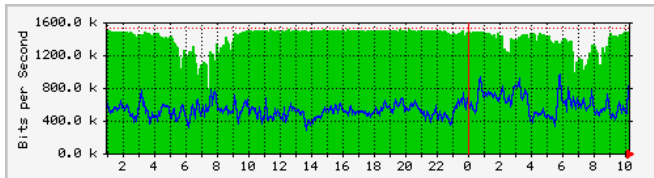
flow-based measurement

- ▶ SNMP: limited to counters (e.g., byte count)
 - ▶ only total amount
- ▶ flow-based measurement: router exports flow statistics by udp
 - ▶ 5 tuples (protocol, srcaddr, dstaddr, srcport, dstport), AS, etc
 - ▶ protocols: NetFlow, sFlow, IPFIX, etc.
- ▶ allows sampling to reduce exported data size



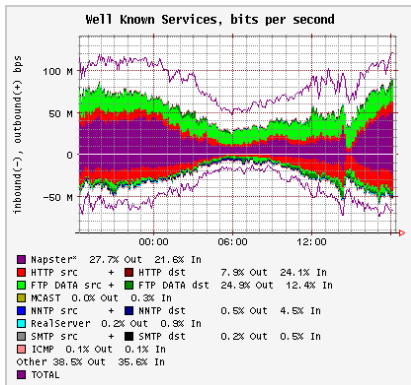
MRTG

- ▶ popular tool to show SNMP data
- ▶ time series data aggregated over time
 - ▶ daily, weekly, monthly to bound the storage size
- ▶ inbound/outbound traffic
 - ▶ can be used for other types of time series data



RRDtool

- ▶ RRDtool: successor of MRTG
 - ▶ flexible configuration, graphing
 - ▶ can be used for any time-series data
- ▶ flowscan: visualizes netflow data by rrdtool



from caida web site

summary of network management tools

- ▶ not originally designed for measurement
- ▶ still often used for measurement
- ▶ when using for measurement, need to understand the mechanisms and limitations

data format

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

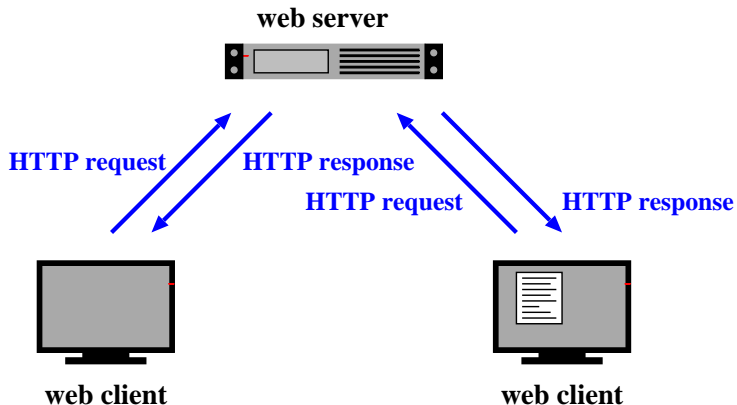
- ▶ log collection
 - ▶ programming (e.g., use of the syslog API)
 - ▶ building a data collection system
- ▶ log rotation
 - ▶ remove old data after a certain period
 - ▶ according to log size, time order, ages of data
 - ▶ should not lose data at log rotation
- ▶ RRD (Round Robin Database)
 - ▶ keep 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

access to a web server

- ▶ HTTP protocol
- ▶ request/response



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 header), "-" 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

SYSLLOG 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
 - ▶ note: rapid crawling is often considered as attacks

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>"
```

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/sfc2012f-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
```

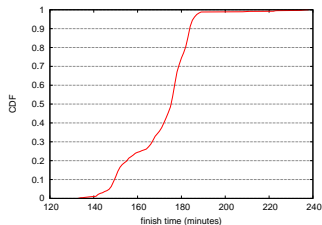
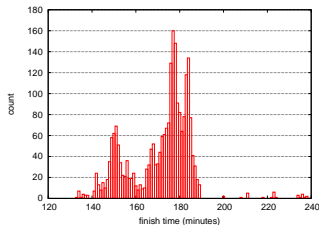
```
n = data.length # number of array elements
r = n / 2 # when n is odd, n/2 is rounded down
if n % 2 != 0
  median = data[r]
else
  median = (data[r - 1] + data[r])/2
end
```

```
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



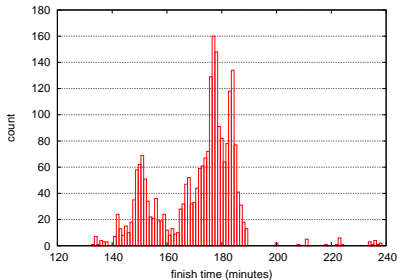
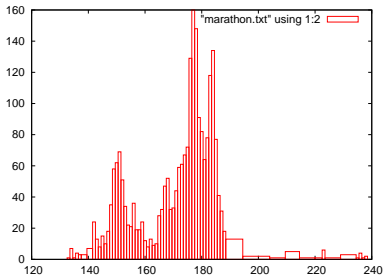
previous exercise: histogram

- ▶ 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)

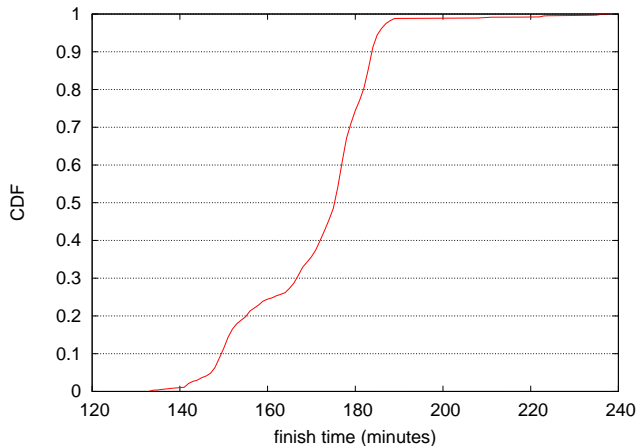
ruby code:

```
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
```

previous exercise: CDF plot of finish-time of city marathon



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 “ip6loganon -anonymize-careful”

access log for 24 hours:

http://www.iijlab.net/~kjc/classes/sfc2013f-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'
```

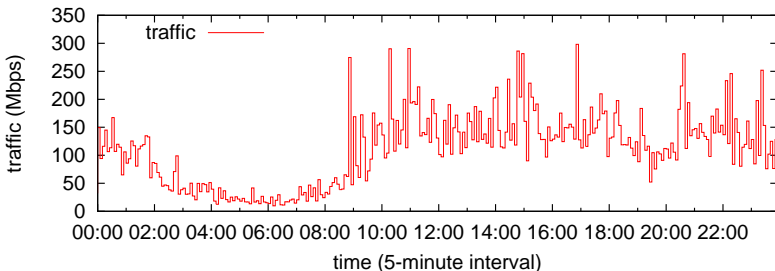
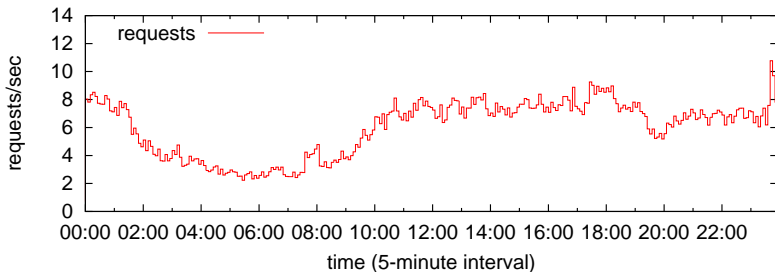
extract request counts and transferred bytes with 5 minutes bins

```
#!/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

- ▶ Network management tools
- ▶ Data format
- ▶ Log analysis methods
- ▶ exercise: log data and regular expression

next class

Class 4 Distribution and confidence intervals (10/16)

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