Internet Measurement and Data Analysis (8)

Kenjiro Cho

2014-12-08

review of previous class

Class 7 Multivariate analysis (12/1)

- Data sensing and GeoLocation
- Linear regression
- Principal Component Analysis
- exercise: linear regression
- **assignment 2**

today's topics

Class 8 Time-series analysis

- Internet and time
- Network Time Protocol
- Time series analysis
- exercise: time-series analysis

time in measurement

- absolute time
  - UTC (Universal Coordinated Time)
    - the international standard time based on atomic clocks
- relative time
  - difference between events
- clock adjustment
  - clock could jump forward or backward!
  - ntp slews clock if difference is less than 128ms

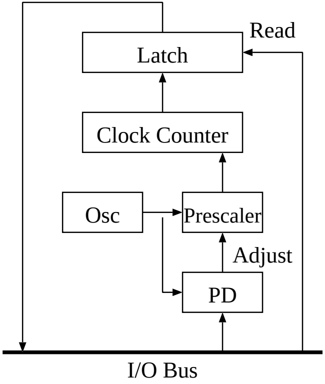
clock uncertainty

- clock uncertainty
  - synchronization
    - difference of 2 clocks
  - accuracy
    - a given clock agrees with UTC
  - resolution
    - precision of a given clock
  - skew
    - change of accuracy or of synchronization with time
- time precision
  - local clock skew/drift: 0.1-1sec/day
  - NTP: synchronizes clock within 10-100ms
  - tcpdump timestamp: 100usec-100msec (usually $< 1msec$)

PC clock

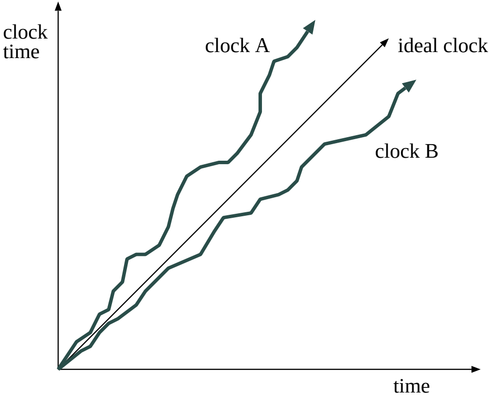
i8254 programmable interval timer

- free-running 16-bit down-counter
  - driven by 1,193,182 Hz oscillator
  - when counter becomes zero, generates interrupt, and reloads the counter register



clock drift

- oscillator drift
  - hardware error margin: 10^{-5}
    - 0.86 sec/day within the spec
  - drift heavily affected by temperature



alternative clocks

- Pentium TSC (Time Stamp Counter)
  - a 64bit free-running counter driven by CPU clock
  - issues with variable clock rate and multi-processors
- ACPI (Advanced Configuration and Power Interface)
  - a free-running counter provided by power management unit
- Local APIC (Advanced Programmable Interrupt Controller)
  - timer with interrupt function embedded on each processor
- HPET (High Precision Event Timer)
  - a new time specification of IA-PC
  - built in chipsets since around 2005
- external clock source
  - GPS, CDMA, shortwave radio
    - access overhead of the interfaces

OS time management

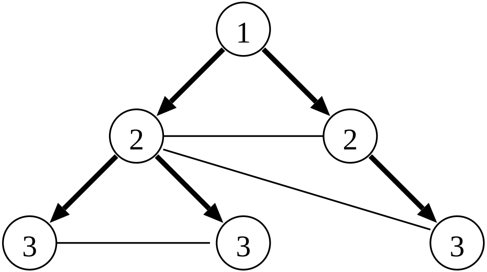
- OS manages software clock
  - initialized at boottime from time-of-day chip
  - updated by hardware clock interrupts
- standard UNIX sets the clock counter (and divider) to interrupt every 10ms (configurable)

UNIX gettimeofday

- older OS has only clock-interrupt resolution
- modern OS has much better resolution
  - interpolate software clock by reading the remaining counter value
    - resolution: 838ns (1 / 1193182)
  - inside kernel
    - access to the i8254 register: 1-10usec
    - conversion to struct timeval: 10-100usec
  - user space - kernel
    - system call overhead: 100-500usec
    - process might be scheduled: 1-100msec or more
- timer events (e.g., setitimer):
  - triggered only by timer tick (10msec by default)
  - effects of process scheduling

NTP (Network Time Protocol)

- multiple time servers across the Internet
  - primary servers: directly connected to UTC receivers
  - secondary servers: synchronize with primaries
  - tertiary servers: synchronize with secondary, etc
- scalability
  - 20-30 primaries, 2000 secondaries can synchronize to $< 30ms$
- many features
  - cope with server failures, authentication support, etc



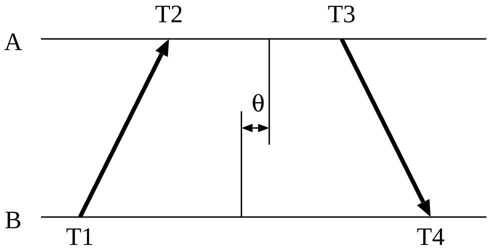
NTP synchronization modes

- multicast (for LAN)
  - one or more servers periodically multicast
- remote procedure call
  - client requests time to a set of servers
- symmetric protocol
  - pairwise synchronization with peers

NTP symmetric protocol

measuring offset and delay

- $a = T2 - T1 \qquad b = T3 - T4$
- clock offset: $\theta = (a + b)/2$, assuming symmetric round-trip
- roundtrip delay: $\delta = a - b$

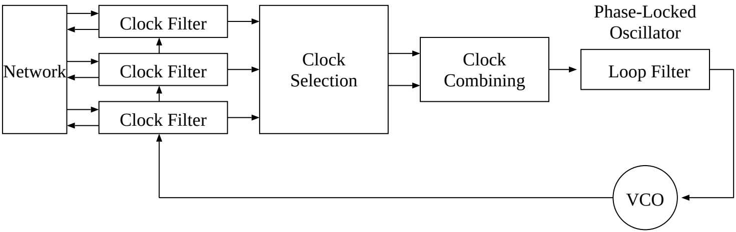


every message contains

- T3: send time (current time)
- T2: receive time
- T1: send time in received message

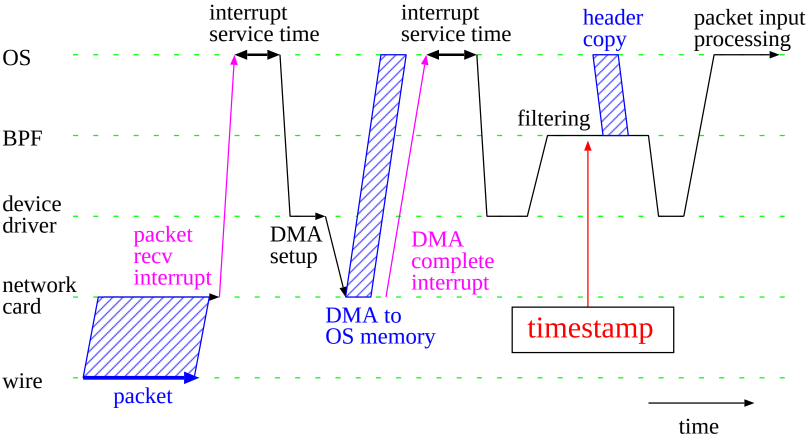
NTP system model

- clock filter
  - temporally smooth estimates from a given peer
- clock selection
  - select subset of mutually agreeing clocks
  - intersection algorithm: eliminate outliers
  - clustering: pick good estimates
- clock combining
  - combine into a single estimate



BPF timestamp on BSD Unix

- timestamp usually placed after 2 interrupts: recv packet, DMA complete
  - recv packet, DMA complete



time-series analysis of network traffic

analysis of dynamic behaviors which change over time

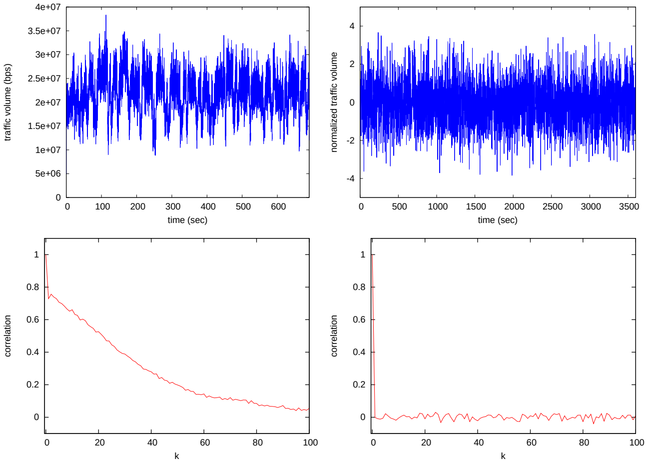
- difficult for mathematical modeling
- only limited tools are available

topics

- autocorrelation
- stationary process
- long-range dependence
- self-similar traffic

autocorrelation of network traffic

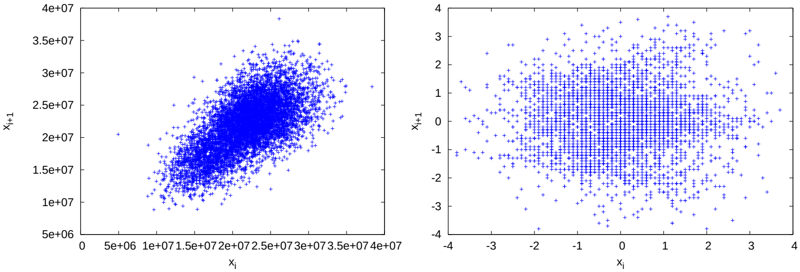
- trends (influence from the past) and periodicity (day, week, season)
- autocorrelation: correlation between two values of the same variable at different times



real traffic (left) and randomly generated traffic (right) timeseries (top) and autocorrelation (bottom)

autocorrelation and lag plot

- lag plot: scatter plot of x_i and x_{i+k}
  - simple way to observe whether autocorrelation exists
  - larger k can find longer cycles of repeating patterns



sample lag plot: real traffic (left) and randomly generated traffic (right)

autocorrelation

- stochastic process

$$\{x(t), t \in T\}$$

- autocorrelation: correlation between two values of the same variable at times t_1 and t_2
- autocorrelation function

$$R(t_1, t_2) = E[x(t_1)x(t_2)]$$

- autocovariance

$$Cov(t_1, t_2) = E((x(t_1) - \mu_{t_1})(x(t_2) - \mu_{t_2})] = E[x(t_1)x(t_2)] - \mu_{t_1}\mu_{t_2}$$

stationary process

- time-series X_t is stationary if
  - mean does not change with time: $E(X_t) = \mu$
  - and autocovariance depends only on k

$$\gamma_k = Cov(X_t, X_{t+k}) = E((X_t - \mu)(X_{t+k} - \mu))$$

$$\gamma_0 = Var(X_t) = E((X_t - \mu)^2)$$

- autocorrelation coefficient
  - autocovariance normalized by variance
  - shows influence of the past

$$\rho_k = \frac{\gamma_k}{\gamma_0}$$

white noise

white noise: stationary process whose autocorrelation coefficient is zero

$$\rho_k = 0 \; (k \neq 0)$$

IID process (independent identically distributed process)

- white noise with constant mean and variance
  - IID process often appears in the literature
- X_t is IID
  - independent: X_t is independent (no autocorrelation)
  - identically distributed: X_t follows the same distribution

non-stationary process

- non-stationary
  - mean changes with time
  - or, autocovariance changes with time
- hard to tackle mathematically
  - generally, take differential time-series to make it stationary
- stationarity test
  - by power spectral density
    - if power-law exponent > 1.0, non-stationary
- network data: sometimes, non-stationary behaviors are observed
  - caused by congestion, attack, etc

power spectral density

- power spectral density of a stationary random process is the fourier transform of the autocorrelation function
  - from time-domain to frequency-domain

$$S(f) = \int_{-\infty}^{\infty} R(\tau) e^{-2\pi i f \tau} d\tau$$

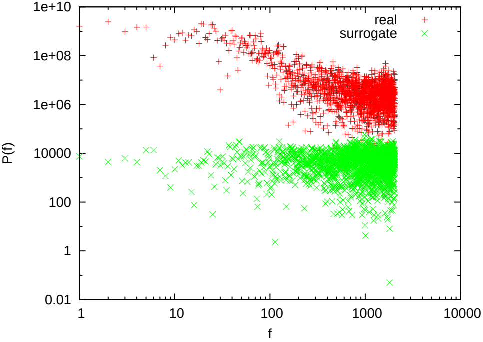
  - power spectral density

$$P(f) \equiv |S(f)|^2 + |S(-f)|^2, \; 0 \leq f < \infty$$

- power spectral density gives relative power contributed by each frequency component

characteristics of power spectral density

- white noise: $P(f) \sim const$
- self-similar (long-range dependence): $P(f) \sim f^{-\alpha}, 0 < \alpha \leq 1.0$
- 1/f fluctuation: $\alpha = 1.0$
- non-stationary: $\alpha > 1.0$



example: real traffic (red) and randomly generated traffic (green)

short-range dependence and long-range dependence

autocovariance shows the influence of each time difference k

sum of autocovariance of all time differences k gives a total view

- short-range dependence
  - $\sum_k \rho(k)$ is finite

$$\sum_{k=0}^{\infty} |\rho(k)| < \infty$$

  - $\rho(k)$ decays at least as fast as exponentially
  - characteristics
    - fluctuates around mean
    - not affected by long past
- long-range dependence
  - $\sum_k \rho(k)$ is infinite

$$\sum_{k=0}^{\infty} |\rho(k)| = \infty$$

  - autocorrelation coefficient decays hyperbolically
  - characteristics
    - values far from mean can be observed

self-similar traffic

network traffic is not exactly self-similar but often better modeled than other models

- scale-invariant
- long-range dependence
- autocovariance decays exponentially

$$\rho(k) \sim k^{-\alpha} \quad (k \to \infty) \quad 0 < \alpha < 1$$

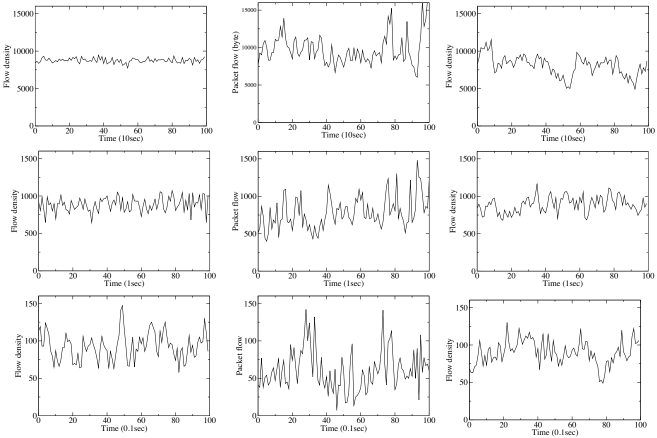
- similarly, power spectral density decays exponentially
  - larger contributions by low frequency components

$$P(f) \sim |f|^{-\alpha} \quad (f \to 0)$$

- infinite variance

self-similarity in network traffic

- exponential model (left), real traffic (middle), self-similar model (right)
- time scale: 10sec (top), 1 sec (middle), 0.1 sec (bottom)



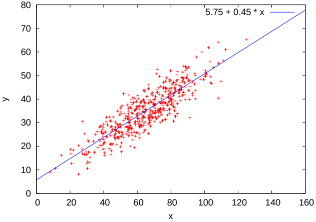
previous exercise: linear regression

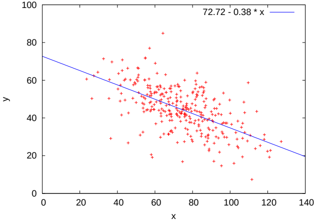
- linear regression by the least square method
- use the data for the previous exercise
  - correlation-data-1.txt, correlation-data-2.txt

$$f(x) = b_0 + b_1 x$$

$$b_1 = \frac{\sum xy - n\bar{x}\bar{y}}{\sum x^2 - n(\bar{x})^2}$$

$$b_0 = \bar{y} - b_1 \bar{x}$$





data-1:r=0.87 (left), data-2:r=-0.60 (right)

script for linear regression

```
#!/usr/bin/env ruby

# regular expression for matching 2 floating numbers
re = /([-+]?\d+(?:\.\d+)?)\s+([-+]?\d+(?:\.\d+)?)/

sum_x = sum_y = sum_xx = sum_xy = 0.0
n = 0
ARGF.each_line do |line|
    if re.match(line)
      x = $1.to_f
      y = $2.to_f

      sum_x += x
      sum_y += y
      sum_xx += x**2
      sum_xy += x * y
      n += 1
    end
end

mean_x = Float(sum_x) / n
mean_y = Float(sum_y) / n
b1 = (sum_xy - n * mean_x * mean_y) / (sum_xx - n * mean_x**2)
b0 = mean_y - b1 * mean_x

printf "b0:%.3f b1:%.3f\n", b0, b1
```

adding the least squares line to scatter plot

```
set xrange [0:160]
set yrange [0:80]

set xlabel "x"
set ylabel "y"

plot "correlation-data-1.txt" notitle with points, \
5.75 + 0.45 * x lt 3
```

today's exercise: autocorrelation

- compute autocorrelation using traffic data for 1 week

```
$ ruby autocorr.rb autocorr_5min_data.txt > autocorr.txt
$ head -10 autocorr_5min_data.txt
2011-02-28T00:00 247 6954152
2011-02-28T00:05 420 49037677
2011-02-28T00:10 231 4741972
2011-02-28T00:15 159 1879326
2011-02-28T00:20 290 39202691
2011-02-28T00:25 249 39809905
2011-02-28T00:30 188 37954270
2011-02-28T00:35 192 7613788
2011-02-28T00:40 102 2182421
2011-02-28T00:45 172 1511718
$ head -10 autocorr.txt
0 1.000
1 0.860
2 0.860
3 0.857
4 0.857
5 0.854
6 0.851
7 0.849
8 0.846
9 0.841
```

computing autocorrelation functions

autocorrelation function for time lag k

$$R(k) = \frac{1}{n}\sum_{i=1}^{n} x_i x_{i+k}$$

normalize by $R(k)/R(0)$, as when $k = 0$, $R(k) = R(0)$

$$R(0) = \frac{1}{n}\sum_{i=1}^{n} x_i^2$$

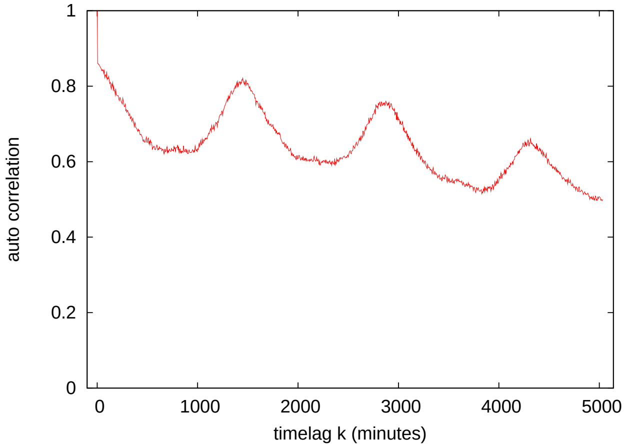
need 2n data to compute $k = n$

autocorrelation computation code

```
# regular expression for matching 5-min timeseries
re = /^(\d{4}-\d{2}-\d{2})T(\d{2}:\d{2})\s+(\d+)\s+(\d+)/

v = Array.new() # array for timeseries
ARGF.each_line do |line|
  if re.match(line)
    v.push $3.to_f
  end
end

n = v.length # n: number of samples
h = n / 2 - 1 # (half of n) - 1

r = Array.new(n/2) # array for auto correlation
for k in 0 .. h # for different timelag
  s = 0
  for i in 0 .. h
    s += v[i] * v[i + k]
  end
  r[k] = Float(s)
end

# normalize by dividing by r0
if r[0] != 0.0
  r0 = r[0]
  for k in 0 .. h
    r[k] = r[k] / r0
    printf "%d %.3f\n", k, r[k]
  end
end
```

autocorrelation plot

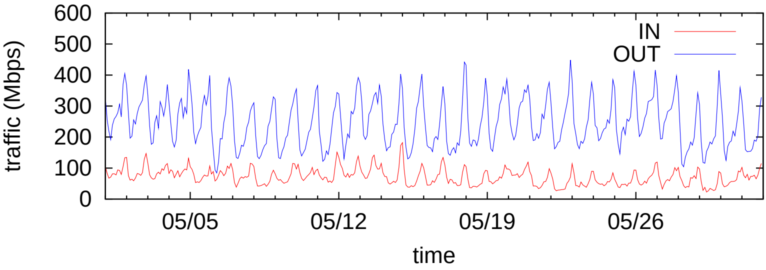
```
set xlabel "timelag k (minutes)"
set ylabel "auto correlation"
set xrange [-100:5140]
set yrange [0:1]
plot "autocorr.txt" using ($1*5):2 notitle with lines
```



today's exercise 2: traffic analysis

exercise data: ifbps-201205.txt

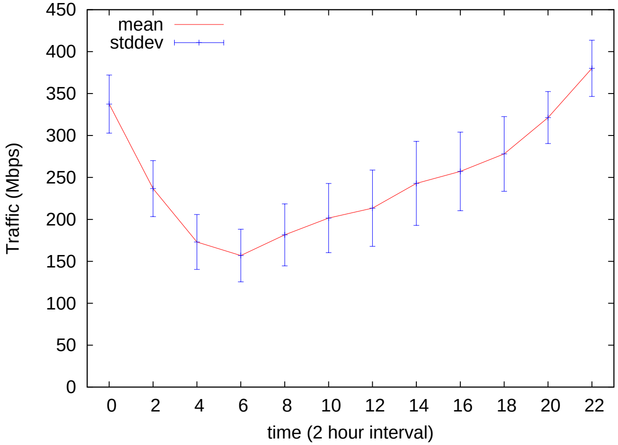
- interface counter values from a router providing services to broadband users
- one month data from May 2012, with 2-hour resolution
- format: time IN(bits/sec) OUT(bits/sec)
- converted from the original format
  - original format: unix_time IN(bytes/sec) OUT(bytes/sec)
- use "OUT" traffic for exercise



plotting time-of-day traffic

- plot mean and standard deviation for each time of day

```
$ ruby hourly_out.rb ifbps-201205.txt > hourly_out.txt
```



script to extract time-of-day traffic

```
#   time  in_bps  out_bps
re = /^\d{4}-\d{2}-(\d{2})T(\d{2}):\d{2}:\d{2}\s+\d+\.\d+\s+(\d+\.\d+)/

# arrays to hold values for every 2 hours
sum = Array.new(12, 0.0)
sqsum = Array.new(12, 0.0)
num = Array.new(12, 0)

ARGF.each_line do |line|
  if re.match(line)
    # matched
    hour = $2.to_i / 2
    bps = $3.to_f

    sum[hour] += bps
    sqsum[hour] += bps**2
    num[hour] += 1
  end
end
printf "#hour\tn\tmean\t\tstddev\n"
for hour in 0 .. 11
  mean = sum[hour] / num[hour]
  var = sqsum[hour] / num[hour] - mean**2
  stddev = Math.sqrt(var)

  printf "%02d\t%d\t%.1f\t%.1f\n", hour * 2, num[hour], mean, stddev
end
```

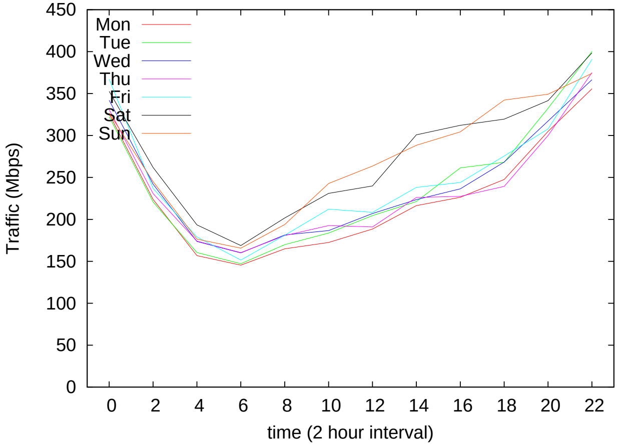
plot script for time-of-day traffic

```
set xlabel "time (2 hour interval)"
set xtic 2
set xrange [-1:23]
set yrange [0:]
set key top left
set ylabel "Traffic (Mbps)"

plot "hourly_out.txt" using 1:($3/1000000) title 'mean' with lines, \
"hourly_out.txt" using 1:($3/1000000):($4/1000000) title "stddev" with yerrorbars lt 3
```

plotting time-of-day traffic for each day of the week

- plotting traffic for each day of the week

```
$ ruby weekview_out.rb ifbps-201205.txt > week_out.txt
```



script to extract time-of-day traffic for each day of the week

```
#   time  in_bps  out_bps
re = /^\d{4}-\d{2}-(\d{2})T(\d{2}):\d{2}:\d{2}\s+\d+\.\d+\s+(\d+\.\d+)/

# 2012-05-01 is Tuesday, add wdoffset to make wday start with Monday
wdoffset = 0

# traffic[wday][hour]
traffic = Array.new(7){ Array.new(12, 0.0) }
num = Array.new(7){ Array.new(12, 0) }

ARGF.each_line do |line|
  if re.match(line)
    # matched
    wday = ($1.to_i + wdoffset) % 7
    hour = $2.to_i / 2
    bps = $3.to_f

    traffic[wday][hour] += bps
    num[wday][hour] += 1
  end
end
printf "#hour\tMon\tTue\tWed\tThu\tFri\tSat\tSun\n"
for hour in 0 .. 11
  printf "%02d", hour * 2
  for wday in 0 .. 6
    printf " %.1f", traffic[wday][hour] / num[wday][hour]
  end
  printf "\n"
end
```

plot script for each day of the week

```
set xlabel "time (2 hour interval)"
set xtic 2
set xrange [-1:23]
set yrange [0:]
set key top left
set ylabel "Traffic (Mbps)"

plot "week_out.txt" using 1:($2/1000000) title 'Mon' with lines, \
"week_out.txt" using 1:($3/1000000) title 'Tue' with lines, \
"week_out.txt" using 1:($4/1000000) title 'Wed' with lines, \
"week_out.txt" using 1:($5/1000000) title 'Thu' with lines, \
"week_out.txt" using 1:($6/1000000) title 'Fri' with lines, \
"week_out.txt" using 1:($7/1000000) title 'Sat' with lines, \
"week_out.txt" using 1:($8/1000000) title 'Sun' with lines
```

correlation coefficient matrix among days of the week

- compute correlation coefficients between days of the week
  - use mean of time-of-day traffic

```
$ ruby correlation_out.rb ifbps-201205.txt
```

	Mon	Tue	Wed	Thu	Fri	Sat	Sun
Mon	1.000	0.985	0.998	0.991	0.988	0.955	0.901
Tue	0.985	1.000	0.981	0.975	0.969	0.964	0.927
Wed	0.998	0.981	1.000	0.987	0.987	0.946	0.897
Thu	0.991	0.975	0.987	1.000	0.988	0.933	0.859
Fri	0.988	0.969	0.987	0.988	1.000	0.951	0.896
Sat	0.955	0.964	0.946	0.933	0.951	1.000	0.971
Sun	0.901	0.927	0.897	0.859	0.896	0.971	1.000

script to compute correlation coefficient matrix

- use the array created for the days of the week

```
n = 12
for wday in 0 .. 6
  for wday2 in 0 .. 6
    sum_x = sum_y = sum_xx = sum_yy = sum_xy = 0.0
    for hour in 0 .. 11
      x = traffic[wday][hour] / num[wday][hour]
      y = traffic[wday2][hour] / num[wday2][hour]

      sum_x += x
      sum_y += y
      sum_xx += x**2
      sum_yy += y**2
      sum_xy += x * y
    end
    r = (sum_xy - sum_x * sum_y / n) /
      Math.sqrt((sum_xx - sum_x**2 / n) * (sum_yy - sum_y**2 / n))
    printf "%.3f\t", r
  end
  printf "\n"
end
```

assignment 2: twitter data analysis

- purpose: processing realworld big data
- data sets:
  - twitter data for about 40M users by Kwak et al. in July 2009
    - http://an.kaist.ac.kr/traces/WWW2010.html
  - twitter_degrees.zip (164MB, 550MB uncompressed)
    - user_id, followings, followers
  - numeric2screen.zip (365MB, 756MB uncompressed)
    - user_id, screen_name
- items to submit
  1. CCDF plot of the distributions of twitter users' followings/followers
     - log-log plot, the number of followings/followers on X-axis
  2. list of the top 30 users by the number of followers
     - rank, user_id, screen_name, followings, followers
  3. optional
     - other analysis of your choice
  4. discussion
     - describe what you observe from the data
- submission: upload your report in the PDF format via SFC-SFS
- submission due: 2014-12-17 (Wed)

twitter data sets

twitter_degrees.zip (164MB, 550MB uncompressed)

```
# id followings followers

12      586     1001061
13      243     1031830
14      106     8808
15      275     14342
16      273     218
17      192     6948
18      87      6532
20      912     1213787
21      495     9027
22      272     3791
...
```

numeric2screen.zip (365MB, 756MB uncompressed)

```
# id screenname

12 jack
13 biz
14 noah
15 crystal
16 jeremy
17 tonystubblebine
18 Adam
20 ev
21 dom
22 rabble
...
```

items to submit

CCDF plot

- log-log plot, the number of followings/followers on X-axis
- plot the 2 distributions in a single graph

list of the top 30 users by the number of followers

- rank, user_id, screen_name, followings, followers
- you need to sort and merge 2 files

```
# rank   id            screenname    followings followers

1      19058681        aplusk           183      2997469
2      15846407        TheEllenShow     26       2679639
3      16409683        britneyspears    406238   2674874
4      428333          cnnbrk           18       2450749
5      19397785        Oprah            15       1994926
6      783214          twitter          55       1959708
...
```

sort command

sort command: sorts lines in a text file

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

- options (relevant to the assignment)
  - -n : compare according to string numerical value
  - -r : reverse the result of comparisons
  - -k POS1[,POS2] : start a key at POS1, end it at POS 2 (origin 1)
  - -t SEP : use SEP instead of non-blank as the field-separator
  - -m : merge already sorted files
  - -T DIR : use DIR for temporary files

example: sort "file" using the 3rd field as numeric value in the reverse order , use "/usr/tmp" for temporary files

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

summary

Class 8 Time-series analysis

- Internet and time
- Network Time Protocol
- Time series analysis
- exercise: time-series analysis

next class

Class 9 Topology and graph (12/15)

- Routing protocols
- Graph theory
- exercise: shortest-path algorithm