# Internet Measurement and Data Analysis (9)

Kenjiro Cho

2014-12-15

# review of previous class

Class 8 Time-series analysis (12/8)

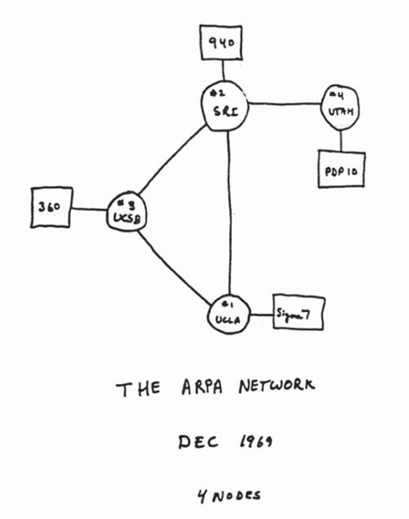
- Internet and time
- Network Time Protocol
- Time series analysis
- exercise: time-series analysis

# today's topics

Class 9 Topology and graph

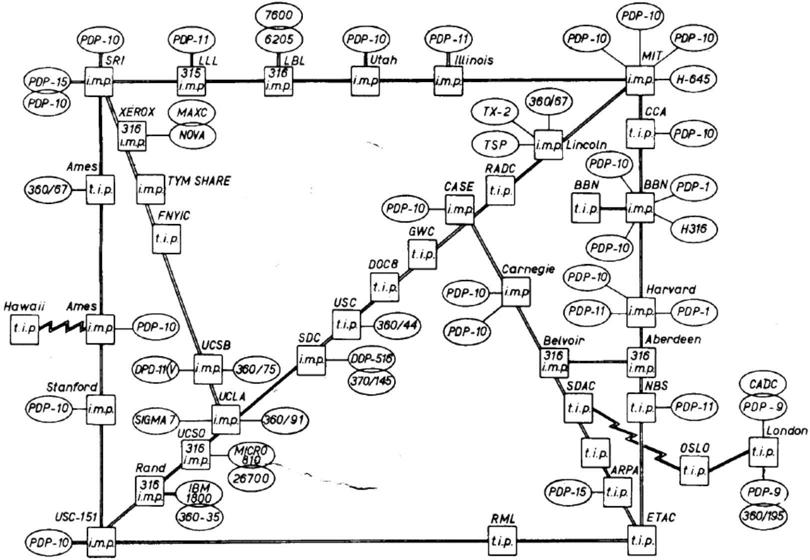
- Routing protocols
- Graph theory
- exercise: shortest-path algorithm

# the first packet switching network



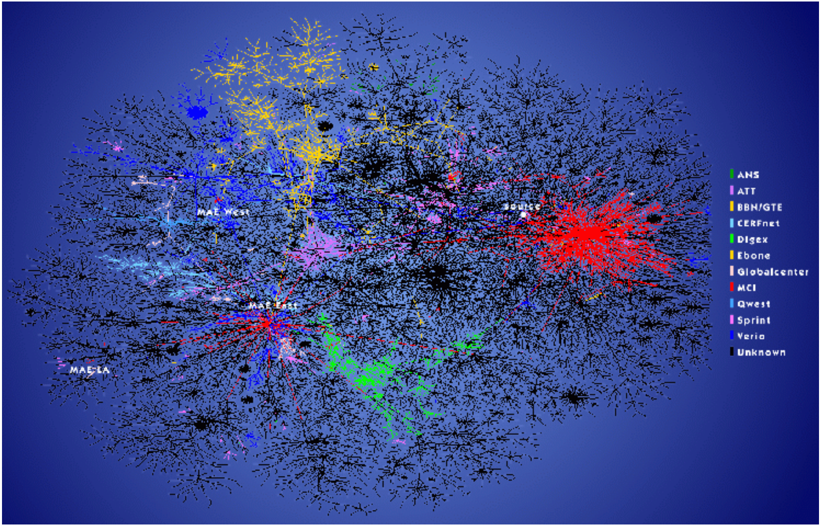

ARPANET in 1969

# ARPANET, 4 years after




ARPANET in 1973

# the Internet

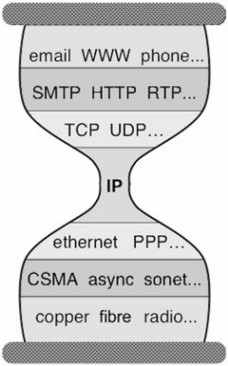



lumeta internet mapping http://www.lumeta.com

http://www.cheswick.com/ches/map/

# the Internet architecture

- IP as a common layer for packet delivery
  - the narrow waist supports diverse lower and upper layers
- the end-to-end model
  - simple network and intelligent end nodes

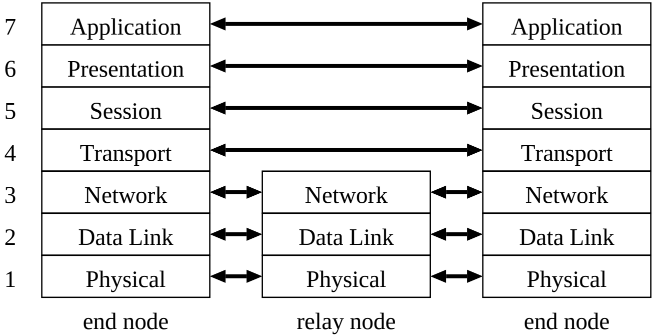



the hour glass model of the Internet architecture

# network layers

abstraction layers to characterize and standerdize the functions of a complex communication system

- the network layer (L3)
  - packet delivery: sending, receiving, and forwarding
  - routing: a mechanism to select the next hop to forward a packet, according to the destination of the packet

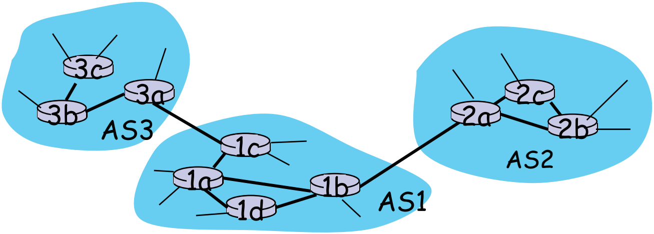



OSI 7 layer model

# routing architecture

hierarchical routing

- Autonomous System (AS): a policy unit for routing (an organization)
  - Keio University: AS38635
  - WIDE Project: AS2500
  - SINET: AS2907
- 2 layers of the Internet routing: intra-AS and inter-AS
  - for scalability
  - inter-AS routing connects networks with different policies
    - hide internal information, and realize operational policies



# routing protocols

exchange routing information with neighbor routers, and update its own routing information

IGP (Interior Gateway Protocol): intra-AS

- RIP (Routing Information Protocol)
  - distance vector routing protocol (Bellman-Ford algorithm)
- OSPF (Open Shortest Path First)
  - link state routing protocol (Dijkstra's algorithm)

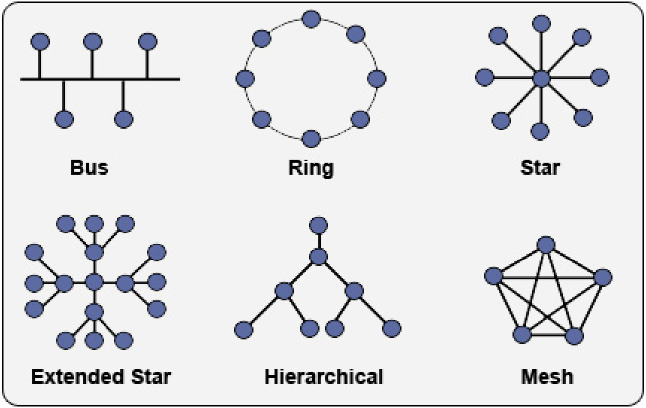
EGP (Exterior Gateway Protocol): inter-AS

- BGP (Boader Gateway Protocol)
  - path vector routing protocol

# topology

topologies (network structure)

- simple topologies
  - bus, ring, star, tree, mesh
- topologies at different layers
  - physical cabling, layer-2, IP-level, overlay
  - hyper-link, social network



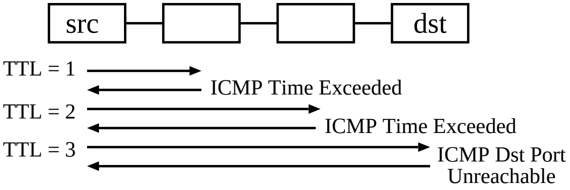
# topology of the Internet

Internet-scale topology information

- router-level topology
  - traceroute
  - data plane information
  - public data:
    - skitter/ark (CAIDA): observations from about 20 monitors
    - iPlane (U. Washington): observations from PlanetLab machines
    - DIMES (Tel Aviv U.) observations from end-users
- AS-level topology
  - BGP routing table
  - control plane information
  - public data: RouteViews (U. Oregon), RIPE RIS

# 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.toknf-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  * * *
```

# BGP information

- each AS announces paths to neighbor ASes following its policies
  - prepending its AS to the AS path
  - policy: how to announce a path to which AS
- BGP data: routing table dump, updates
- sample BGP data:

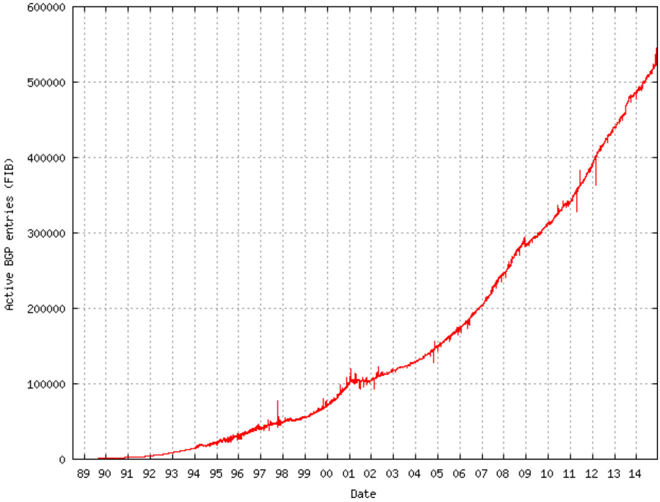
```
BGP table version is 33157262, local router ID is 198.32.162.100
Status codes: s suppressed, d damped, h history, * valid, > best, i -
internal, S Stale
Origin codes: i - IGP, e - EGP, ? - incomplete

    Network              Next Hop    Metric LocPrf Weight Path

*  202.48.48.0/20    196.7.106.245     0         0 2905 701 2500 i
*>                   203.181.248.233           0 7660 2500 i
```

# RouteViews project

- a project to collect and publish BGP data by University of Oregon
  - http://www.routeviews.org/
- about 10 collectors: data provided by major ASes
- publicly available data from 1997

# historical routing table size

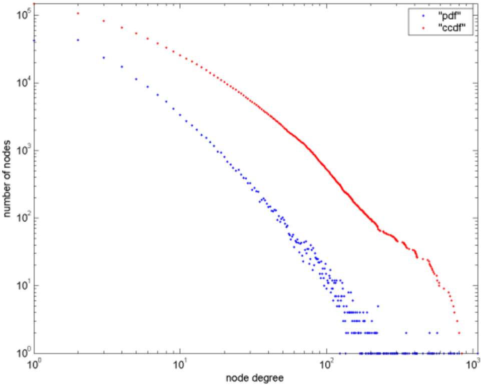
- active BGP entries (FIB): 526k on 2014/12/09




http://www.cidr-report.org/

# CAIDA's skitter/ark projects

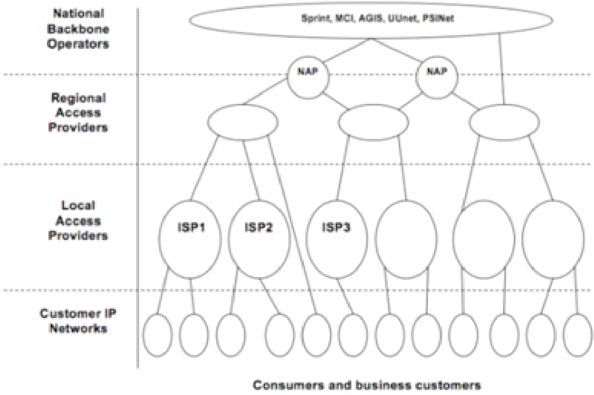
- a topology measurement project by CAIDA
  - skitter/ark: parallel execution of traceroute
  - exhaustive path search by about 20 monitors




router-level degree distribution

# Internet AS hierarchy

## Textbook Internet (1995 – 2007)

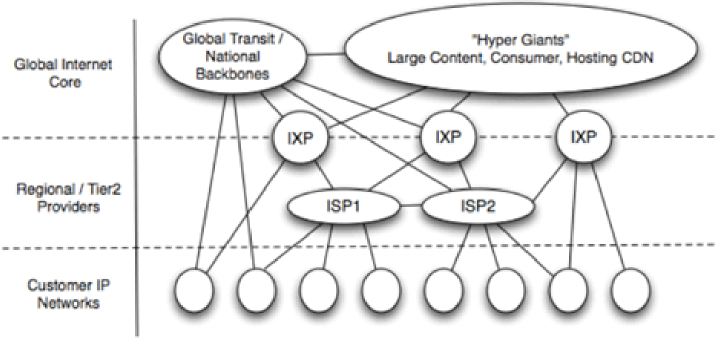



- Tier1 global core (modulo a few name changes over the years)
- Still taught today

source: 2009 Internet Observatory Report (NANOG47)

# recent change in Internet AS hierarchy

## The New Internet




- New core of interconnected content and consumer networks
- New commercial models between content, consumer and transit
- Dramatic improvements in capacity and performance

source: 2009 Internet Observatory Report (NANOG47)

# graph theory

topology can be described by graph theory

- a graph is a collection of nodes (or vertices) and edges
- an undirected graph and a directed graph: whether edges are directional
- a weighted graph: an edge has a weight (cost)
- a path: a series of edges between 2 nodes
- a subgraph: a subset of a graph
- degree: the number of edges connected to a node

applications for network algorithms

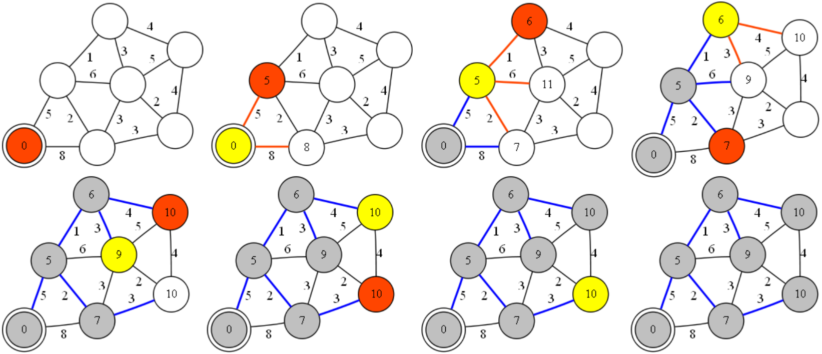
- spanning tree algorithm (loop prevention)
- shortest path algorithm (routing)
  - Bellman-Ford algorithm
  - Dijkstra algorithm

analysis of network characteristics

- clustering
- average shortest path (small world)
- degree distribution analysis (scale-free: degree distribution follows power-law)

# Dijkstra algorithm

```
1. cost initialization: start_node = 0, other_nodes = infinity
2. loop:
     (1) find the node with the lowest cost among the unfinished nodes,
         and fix its cost
     (2) update the cost of its neighbors
```

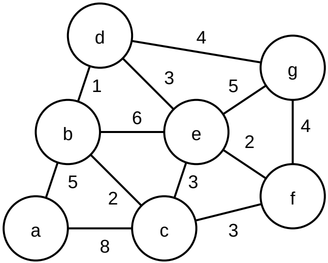


dijkstra algorithm

# today's exercise: Dijkstra algorithm

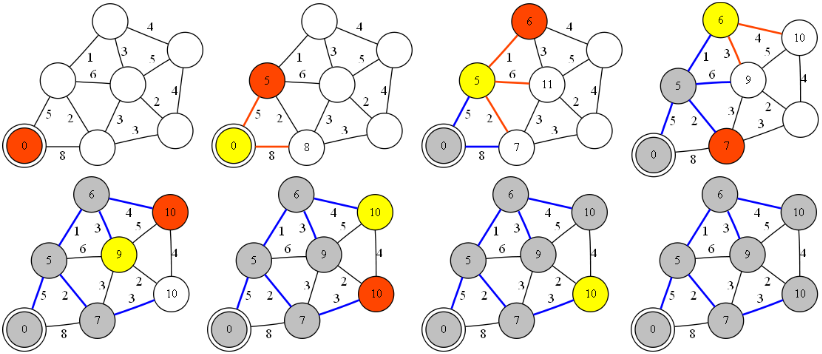
- read a topology file, and compute shortest paths

```
$ cat topology.txt
a - b 5
a - c 8
b - c 2
b - d 1
b - e 6
c - e 3
d - e 3
c - f 3
e - f 2
d - g 4
e - g 5
f - g 4
$ ruby dijkstra.rb -s a topology.txt
a: (0) a
b: (5) a b
c: (7) a b c
d: (6) a b d
e: (9) a b d e
f: (10) a b c f
g: (10) a b d g
$
```



# Dijkstra algorithm

```
1. cost initialization: start_node = 0, other_nodes = infinity
2. loop:
     (1) find the node with the lowest cost among the unfinished nodes,
         and fix its cost
     (2) update the cost of its neighbors
```



dijkstra algorithm

# sample code (1/4)

```
# dijkstra's algorithm based on the pseudo code in the wikipedia
# http://en.wikipedia.org/wiki/Dijkstra%27s_algorithm
#
require 'optparse'

source = nil # source of spanning-tree

OptionParser.new {|opt|
  opt.on('-s VAL') {|v| source = v}
  opt.parse!(ARGV)
}

INFINITY = 0x7fffffff # constant to represent a large number
```

# sample code (2/4)

```
# read topology file and initialize nodes and edges
# each line of topology file should be "node1 (-|->) node2 weight_val"
nodes = Array.new # all nodes in graph
edges = Hash.new # all edges in graph
ARGF.each_line do |line|
  s, op, t, w = line.split
  next if line[0] == ?# || w == nil
  unless op == "-" || op == "->"
    raise ArgumentError, "edge_type should be either '-' or '->'"
  end
  weight = w.to_i
  nodes << s unless nodes.include?(s) # add s to nodes
  nodes << t unless nodes.include?(t) # add t to nodes
  # add this to edges
  if (edges.has_key?(s))
    edges[s][t] = weight
  else
    edges[s] = {t=>weight}
  end
  if (op == "-")  # if this edge is undirected, add the reverse directed edge
    if (edges.has_key?(t))
      edges[t][s] = weight
    else
      edges[t] = {s=>weight}
    end
  end
end
# sanity check
if source == nil
  raise ArgumentError, "specify source_node by '-s source'"
end
unless nodes.include?(source)
  raise ArgumentError, "source_node(#{source}) is not in the graph"
end
```

# sample code (3/4)

```
# create and initialize 2 hashes: distance and previous
dist = Hash.new # distance for destination
prev = Hash.new # previous node in the best path
nodes.each do |i|
  dist[i] = INFINITY # Unknown distance function from source to v
  prev[i] = -1 # Previous node in best path from source
end

# run the dijkstra algorithm
dist[source] = 0 # Distance from source to source
while (nodes.length > 0)
  # u := vertex in Q with smallest dist[]
  u = nil
  nodes.each do |v|
    if (!u) || (dist[v] < dist[u])
      u = v
    end
  end
  if (dist[u] == INFINITY)
    break # all remaining vertices are inaccessible from source
  end
  nodes = nodes - [u] # remove u from Q
  # update dist[] of u's neighbors
  edges[u].keys.each do |v|
    alt = dist[u] + edges[u][v]
    if (alt < dist[v])
      dist[v] = alt
      prev[v]  = u
    end
  end
end
```

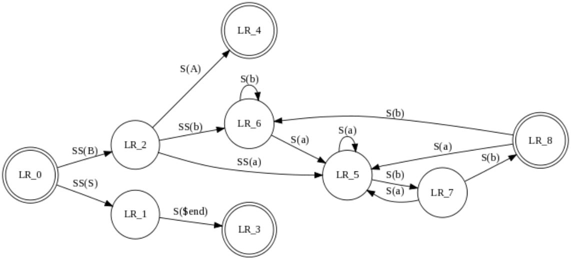
# sample code (4/4)

```
# print the shortest-path spanning-tree
dist.sort.each do |v, d|
  print "#{v}: " # destination node
  if d != INFINITY
    print "(#{d}) " # distance
    # construct path from dest to source
    i = v
    path = "#{i}"
    while prev[i] != -1 do
      path.insert(0, "#{prev[i]} ") # prepend previous node
      i = prev[i]
    end
    puts "#{path}" # print path from source to dest
  else
    puts "unreachable"
  end
end
```

# graph drawing tools based on graph theory

- reads definitions of nodes and edges, and lays out a graph
- example: graphviz (http://www.graphviz.org/)

```
digraph finite_state_machine {
    rankdir=LR;
    size="8,5"
    node [shape = doublecircle]; LR_0 LR_3 LR_4 LR_8;
    node [shape = circle];
    LR_0 -> LR_2 [ label = "SS(B)" ];
    LR_0 -> LR_1 [ label = "SS(S)" ];
    ...
    LR_8 -> LR_6 [ label = "S(b)" ];
    LR_8 -> LR_5 [ label = "S(a)" ];
}
```



# previous 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 $k = 0$, $R(k) = R(0)$

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

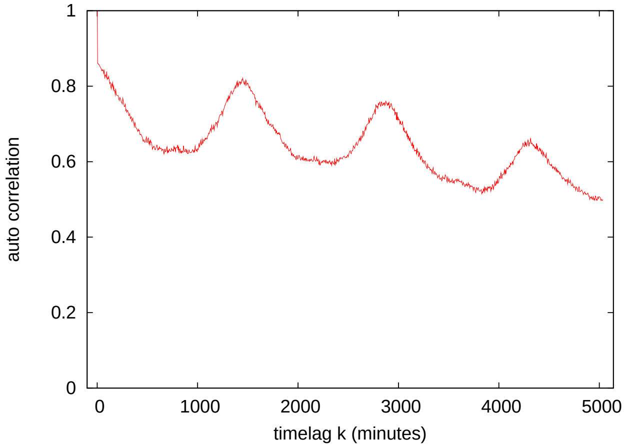
need 2n data to compute $k = n$

# 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

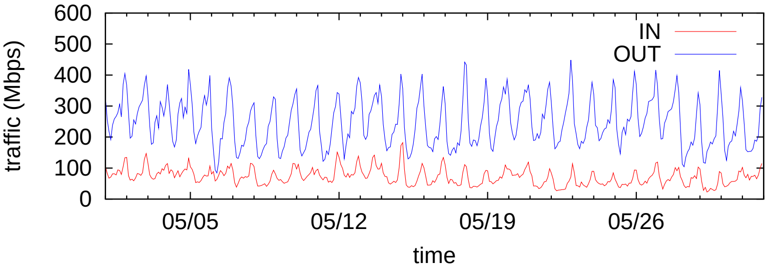
```
set xlabel "timelag k (minutes)"
set ylabel "auto correlation"
set xrange [-100:5140]
set yrange [0:1]
plot "autocorr.txt" using ($1*5):2 notitle with lines
```



# previous exercise 2: traffic analysis

exercise data: ifbps-201205.txt

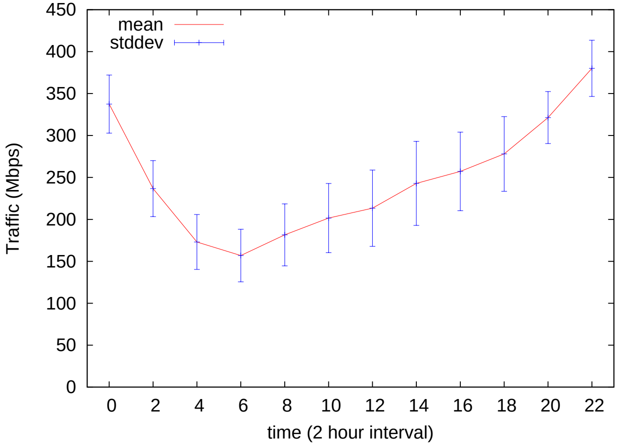
- interface counter values from a router providing services to broadband users
- one month data from May 2012, with 2-hour resolution
- format: time IN(bits/sec) OUT(bits/sec)
- converted from the original format
  - original format: unix_time IN(bytes/sec) OUT(bytes/sec)
- use "OUT" traffic for exercise



# 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
```

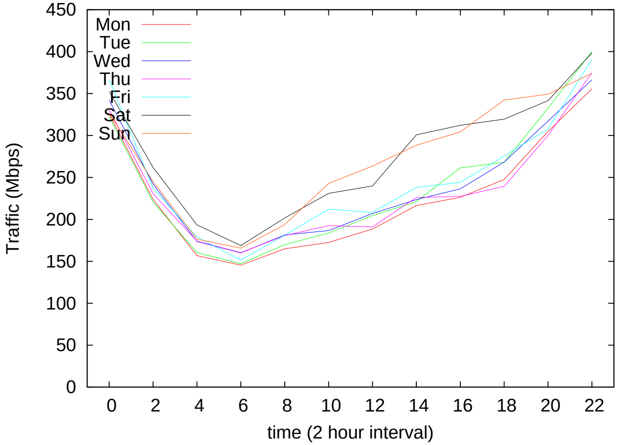
# plot script for time-of-day traffic

```
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 9 Topology and graph

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

# next class

Class 10 Anomaly detection and machine learning (12/22)

- Anomaly detection
- Machine Learning
- SPAM filtering and Bayes theorem
- exercise: naive Bayesian filter
- **the final report**