Internet Measurement and Data Analysis (9)

Kenjiro Cho

2012-11-28

review of previous class

Class 8 Time-series analysis (11/20)

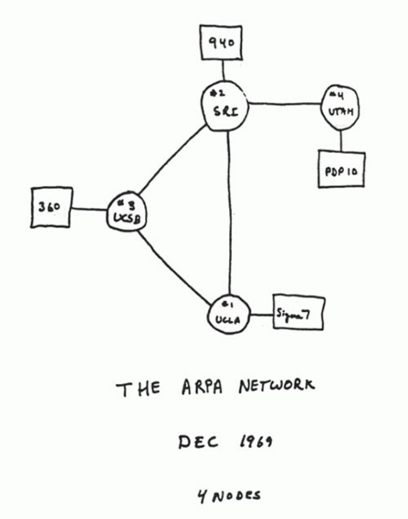
- Internet and time
- Network Time Protocol
- Time series analysis
- exercise: time-series analysis
- **assignment 2**

today's topics

Class 9 Topology and graph

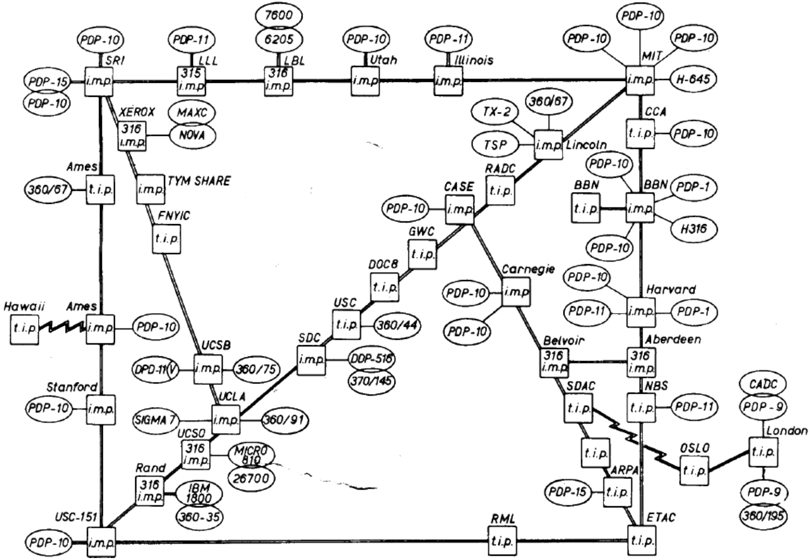
- Routing protocols
- Graph theory
- exercise: shortest-path algorithm

the first packet switching network



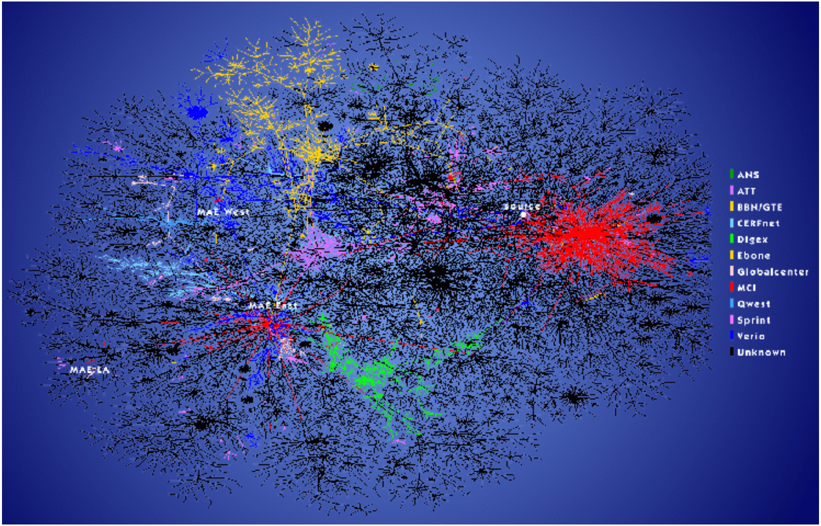
ARPANET in 1969

ARPANET, 4 years after



ARPANET in 1973

the Internet

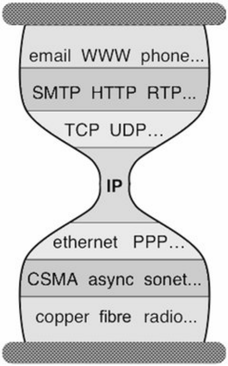


lumeta internet mapping http://www.lumeta.com

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

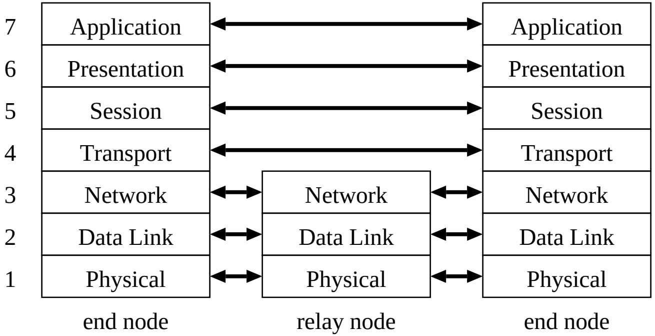


the hour glass model of the Internet architecture

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

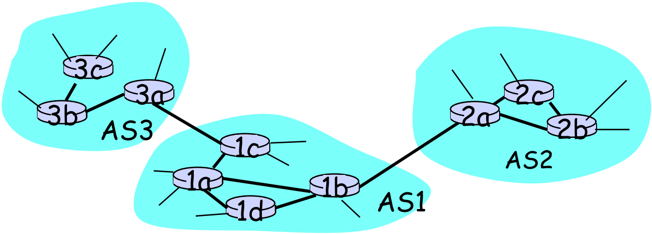


OSI 7 layer model

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)

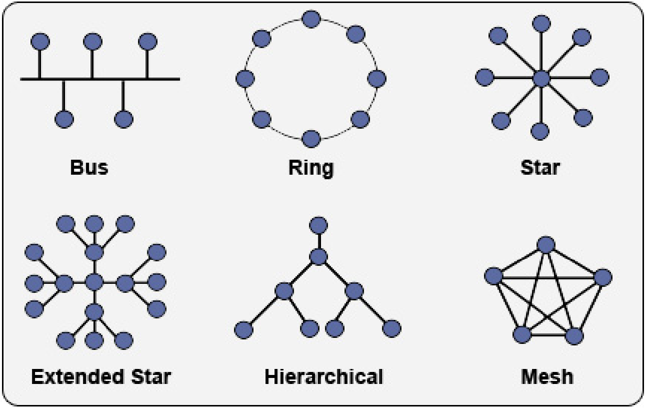
EGP (Exterior Gateway Protocol): inter-AS

- 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



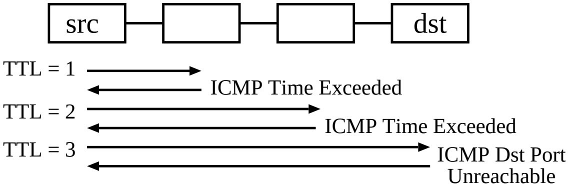
topology of the Internet

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:

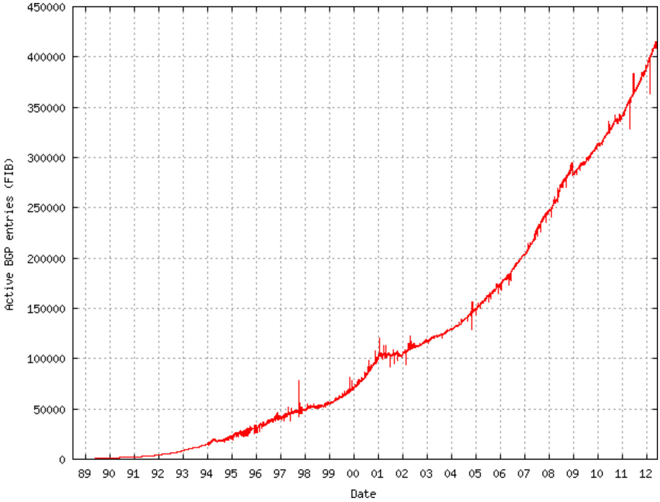
```
BGP table version is 33157262, local router ID is 198.32.162.100
Status codes: s suppressed, d damped, h history, * valid, > best, i -
internal, S Stale
Origin codes: i - IGP, e - EGP, ? - incomplete

    Network          Next Hop      Metric LocPrf Weight Path

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

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

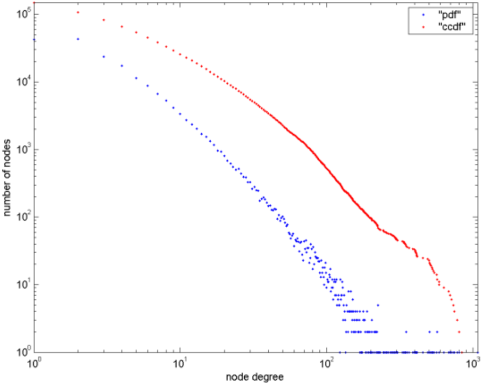
- active BGP entries (FIB): 416k on 2012/6/7



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

CAIDA's skitter/ark projects

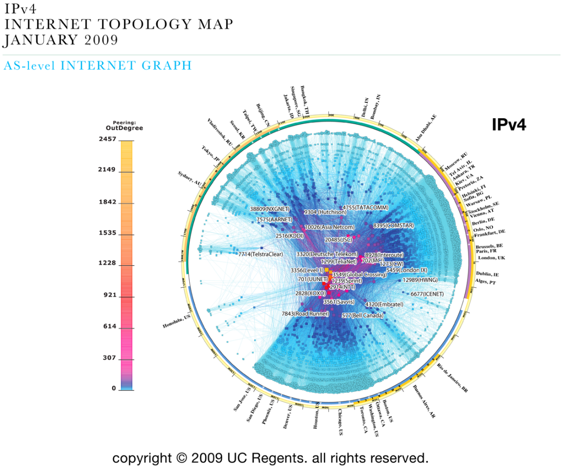
- a topology measurement project by CAIDA
  - skitter/ark: parallel execution of traceroute
  - exhaustive path search by about 20 monitors



router-level degree distribution

CAIDA AS CORE MAP 2009/03

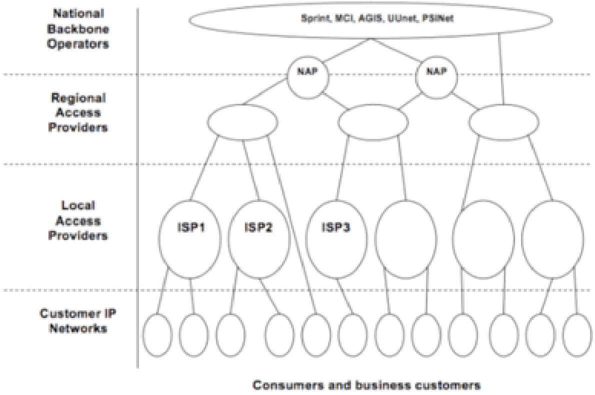
- visualization of AS topology using skitter/ark data
- longitude of AS (registered location), out-degree of AS



http://www.caida.org/research/topology/as_core_network/

Internet AS hierarchy

Textbook Internet (1995 – 2007)

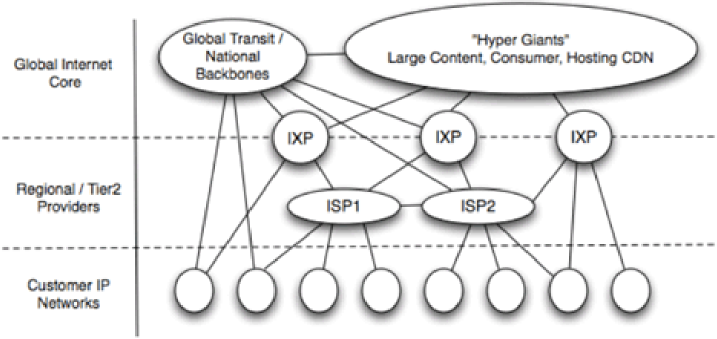


- Tier1 global core (modulo a few name changes over the years)
- Still taught today

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

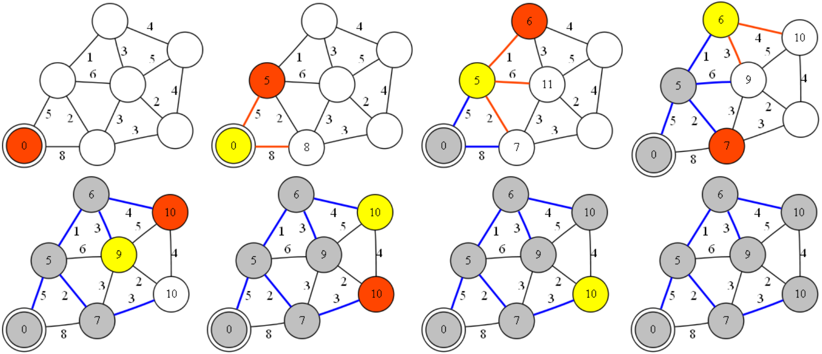
- spanning tree algorithm (loop prevention)
- shortest path algorithm (routing)
  - Bellman-Ford algorithm
  - Dijkstra algorithm

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

previous exercise 1: 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 $k = 0$, $R(k) = R(0)$

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

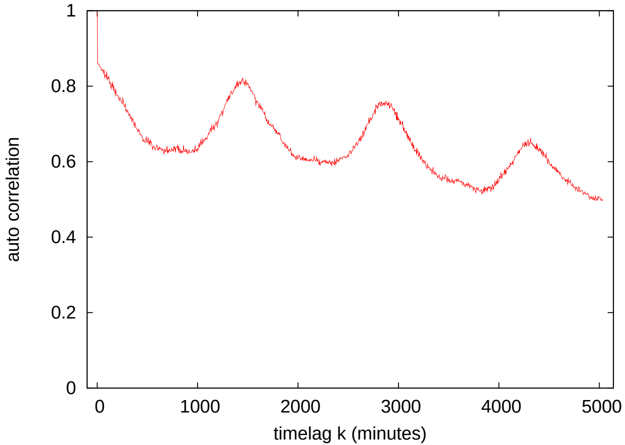
need 2n data to compute $k = n$

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

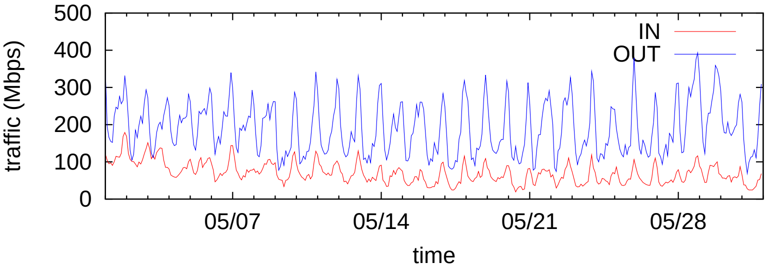
```
set xlabel "timelag k (minutes)"
set ylabel "auto correlation"
set xrange [-100:5140]
set yrange [0:1]
plot "autocorr.txt" using ($1*5):2 notitle with lines
```



previous exercise 2: traffic analysis

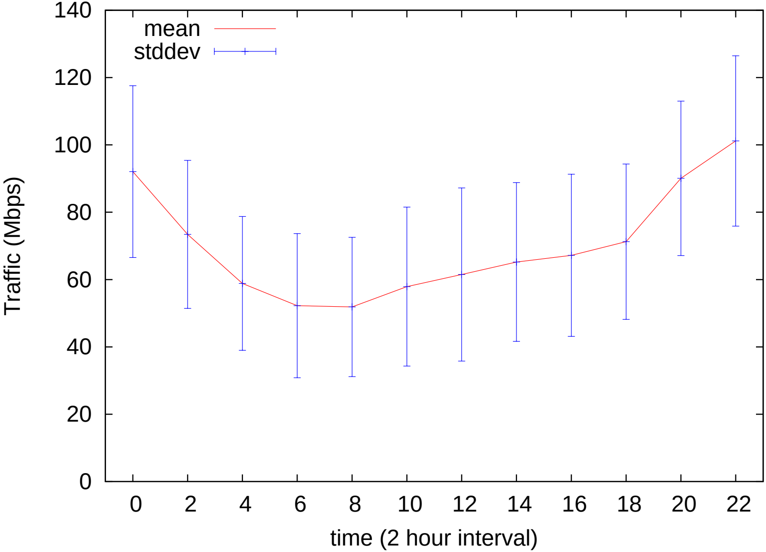
exercise data: ifbps-2011.txt

- interface counter values from a router providing services to broadband users
- one month data from May 2011, 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 "IN" traffic for exercise



plotting time-of-day traffic

- plot mean and standard deviation for each time of day



script to extract time-of-day traffic

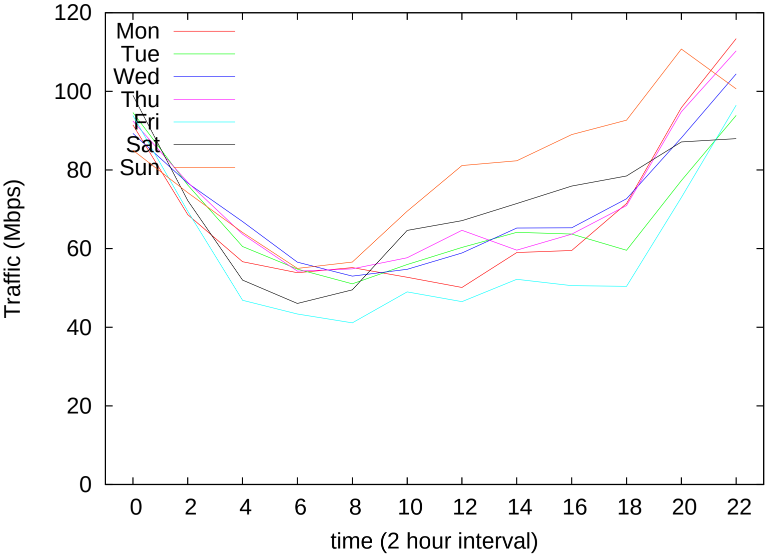
```
#   time  in_bps  out_bps
re = /^\d{4}-\d{2}-(\d{2})T(\d{2}):\d{2}:\d{2}\s+(\d+\.\d+)\s+\d+\.\d+/

# 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_in.txt" using 1:($3/1000000) title 'mean' with lines, \
"hourly_in.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



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+/

# 2011-05-01 is Sunday, add wdoffset to make wday start with Monday
wdoffset = 5

# 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_in.txt" using 1:($2/1000000) title 'Mon' with lines, \
"week_in.txt" using 1:($3/1000000) title 'Tue' with lines, \
"week_in.txt" using 1:($4/1000000) title 'Wed' with lines, \
"week_in.txt" using 1:($5/1000000) title 'Thu' with lines, \
"week_in.txt" using 1:($6/1000000) title 'Fri' with lines, \
"week_in.txt" using 1:($7/1000000) title 'Sat' with lines, \
"week_in.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

	Mon	Tue	Wed	Thu	Fri	Sat	Sun
Mon	1.000	0.888	0.970	0.974	0.919	0.785	0.736
Tue	0.888	1.000	0.935	0.927	0.989	0.840	0.624
Wed	0.970	0.935	1.000	0.980	0.938	0.811	0.745
Thu	0.974	0.927	0.980	1.000	0.941	0.813	0.756
Fri	0.919	0.989	0.938	0.941	1.000	0.829	0.610
Sat	0.785	0.840	0.811	0.813	0.829	1.000	0.853
Sun	0.736	0.624	0.745	0.756	0.610	0.853	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: traffic analysis

- purposes: analyzing real time-series data
- data: ifbps-2012.txt (the same interface counter for the exercise 2 but for 2012)
  - 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)
- items to submit
  1. IN/OUT traffic plot for the entire month with 2 hour resolution
  2. time-of-day traffic of OUT
     - plot mean and standard deviation for each time of day
  3. time-of-day traffic plot of OUT for each day of the week
  4. correlation coefficient matrix of OUT among days of the week
  5. option
     - other analysis (e.g., IN vs. OUT, 2011 vs. 2012)
  6. discussion
     - describe your observations about the data and plots
- submission format: a single PDF file including item 1-6
- submission method: upload the PDF file through SFC-SFS
- submission due: 2012-12-07

assignment 1 answer: the finish time distribution of a marathon

- purpose: investigate the distribution of a real-world data set
- data: the finish time records from honolulu marathon 2010
  - http://results.sportstats.ca/res2010/honolulu.htm
  - the number of finishers: 20,181
- items to submit
  1. mean, standard deviation and median of the total finishers, male finishers, and female finishers
  2. the distributions of finish time for each group (total, men, and women)
     - plot 3 histograms for 3 groups
     - use 10 minutes for the bin size
     - use the same scale for the axes to compare the 3 plots
  3. CDF plot of the finish time distributions of the 3 groups
     - plot 3 groups in a single graph
  4. optional
     - other analysis of your choice (e.g., CDF plots of age groups or countries)
  5. discussion
     - describe your observations about the data and plots
- submission format: a single PDF file including item 1-5
- submission method: upload the PDF file through SFC-SFS
- submission due: 2012-11-09

honolulu marathon data set

data format

```
      Chip     Pace                                                        Gender   Category        @10km  @21.1  @3
Place Time     /mi       #    Name                 City         ST CNT Plce/Tot Plc/Tot Category Split1 Split2
---- -------- ---- ----- -------------------- ------------ -- --- -------- ------ ------ ----- ------- ---
   1 02:15:18 5:10     4 Chelimo, Nicholas    Ngong Hills     KEN 1/10586  1/9    MElite 32:57 1:07:41 1:
   2 02:17:18 5:15     3 Limo, Richard        Eldoret         KEN 2/10586  2/9    MElite 32:56 1:07:40 1:
   3 02:19:54 5:21     5 Bushendich, Solomon  Eldoret         KEN 3/10586  3/9    MElite 32:57 1:07:40 1:
   4 02:20:58 5:23     8 Kirwa, Gibert        Iten            KEN 4/10586  4/9    MElite 32:56 1:07:40 1:
   5 02:22:34 5:27     1 Muindi, Jimmy        Kangundo        KEN 5/10586  5/9    MElite 32:56 1:08:11 1:
   6 02:22:36 5:27     2 Hussein, Mbarak      Albuquerque  NM USA 6/10586  6/9    MElite 32:57 1:09:57 1:
   7 02:27:25 5:38    11 Stanko, Nicholas     Haslette     MI USA 7/10586  7/9    MElite 32:57 1:10:22 1:
   8 02:30:20 5:45 29712 Ogura, Makoto        Hiroshima    Hi JPN 8/10586  1/1229 M35-39 34:11 1:13:14 1:
   9 02:32:13 5:49  9670 Gebre, Belainesh     Flagstaff    AZ USA 1/9830   1/12   WElite 34:33 1:14:46 1:
  10 02:33:00 5:51    F1 Zakharova, Svetlana  Cheboksary      RUS 2/9830   2/12   WElite 36:15 1:15:52 1:
...
```

- Chip Time: finish time
- Category: MElite, WElite, M15-19, M20-24, ..., W15-29, W20-24, ...
  - note some runners have "No Age" for Category
- Country: 3-letter country code: e.g., JPN, USA
  - note some runners have "UK" for country-code
- check the number of the total finishers when you extract the finishers

item 1: computing mean, standard deviation and median

- round off to minute (slightly different from using seconds)
- exclude "No Age" for the male and female groups

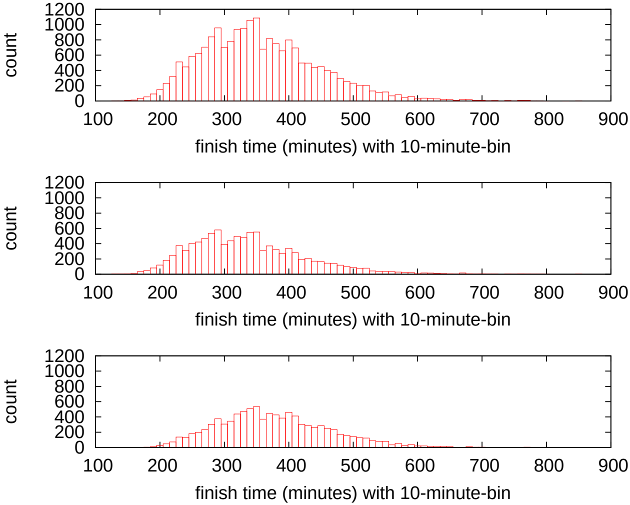
	n	mean	stddev	median
all	20,181	361.2	91.3	350
men	10,463	342.4	90.7	331
women	9,717	381.4	87.5	373

example script to extract data

```
# regular expression to read chiptime and category from honolulu.htm
re = /\s*\d+\s+(\d{2}:\d{2}:\d{2})\s+.*((?:[MW](?:Elite|\d{2}\-\d{2})|No Age))/

filename = ARGV[0]

open(filename, 'r') do |io|
  io.each_line do |line|
    if re.match(line)
      puts "#{$1}\t#{$2}"
    end
  end
end
```

item 2: histograms for 3 groups

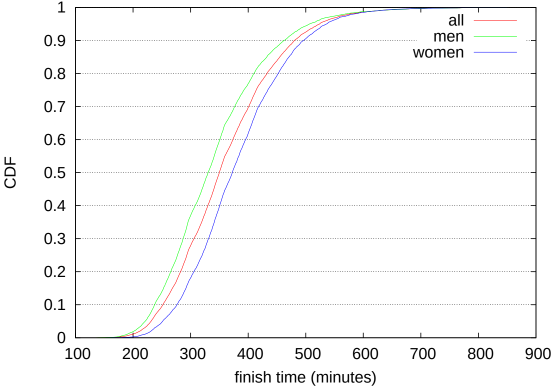
- plot 3 histograms for 3 groups
- use 10 minutes for the bin size
- use the same scale for the axes to compare the 3 plots



finish time histograms total(top) men(middle) women(bottom)

item 3: CDF plot of the finish time distributions of the 3 group

- plot 3 groups in a single graph



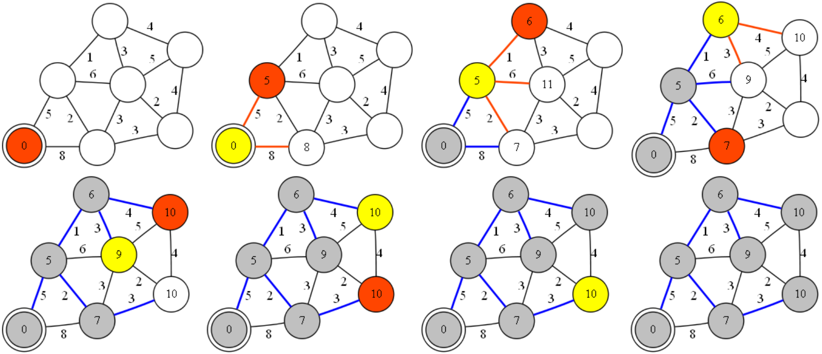
today's exercise: Dijkstra algorithm

- 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
```

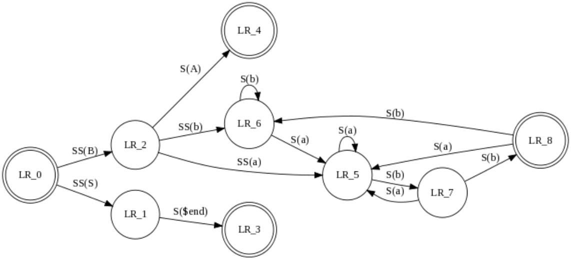
sample code (4/4)

```
# 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)" ];
}
```



summary

Class 9 Topology and graph

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

next class

Class 10 Anomaly detection and machine learning (12/5)

- Anomaly detection
- Machine Learning
- SPAM filtering and Bayes theorem
- exercise: naive Bayesian filter