Internet Measurement and Data Analysis (5)

Kenjiro Cho

2011-10-26

review of previous class

Class 4 Measuring the speed of the Internet

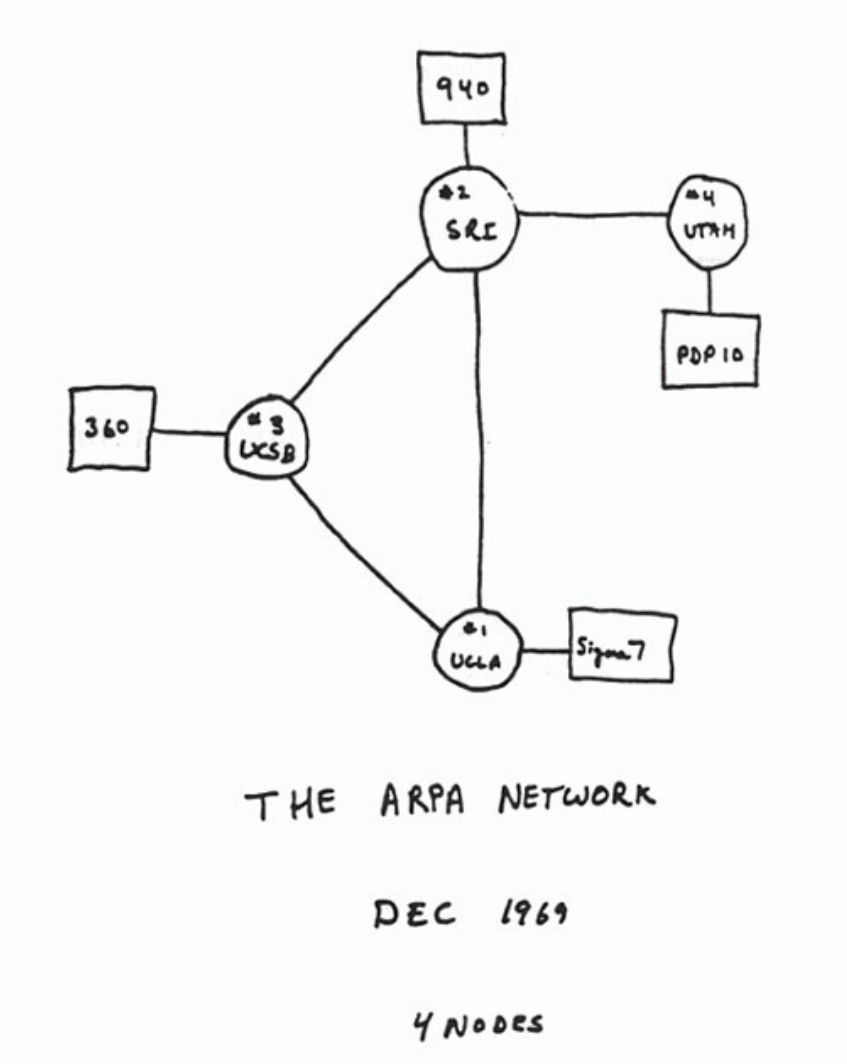
- bandwidth measurement
- inferring available bandwidth
- mean, standard deviation
- linear regression
- exercise: mean, standard deviation, linear regression
- **assignment 1**

today's topics

Class 5 Measuring the structure of the Internet

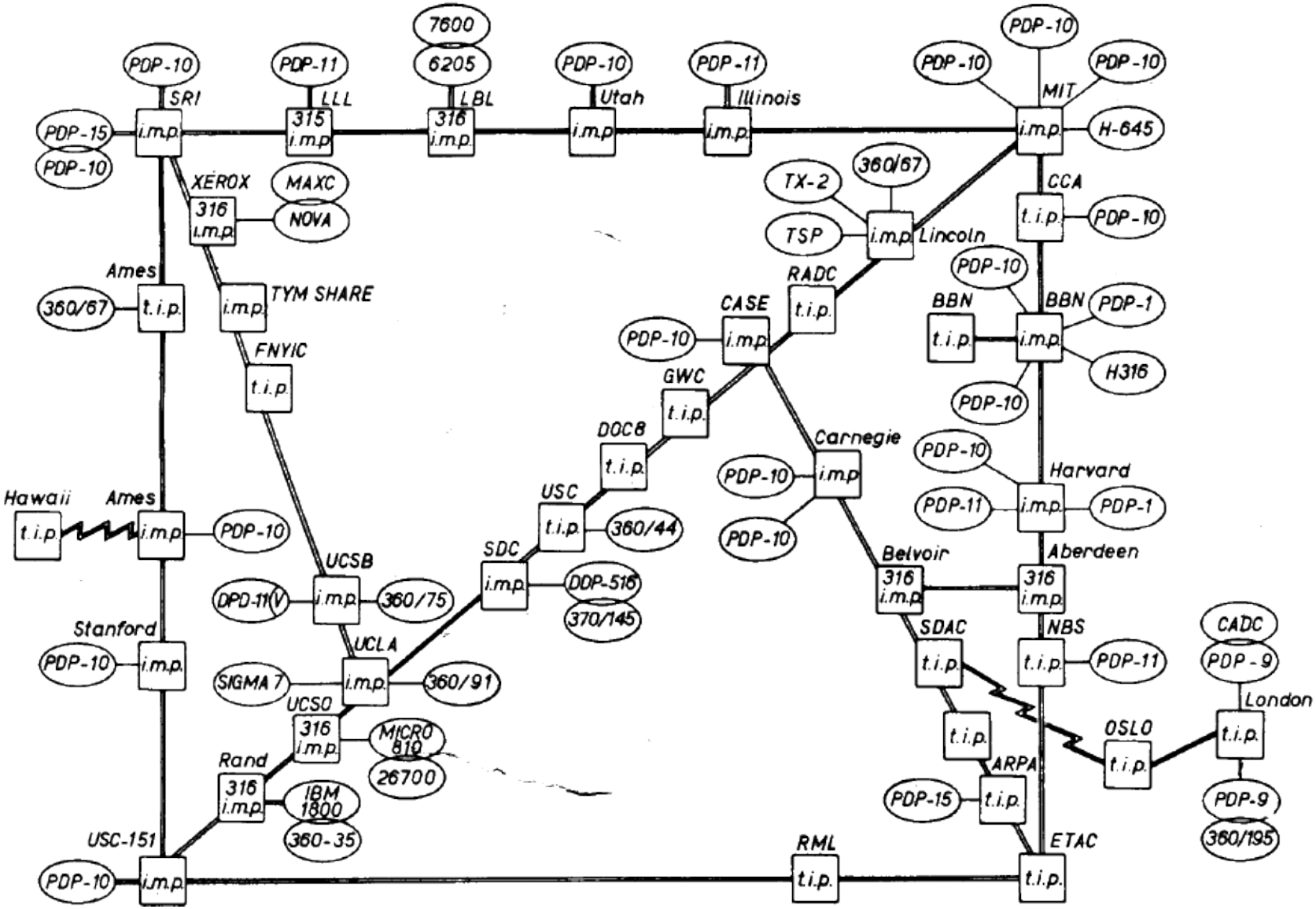
- Internet architecture
- network layers
- topologies
- graph theory
- exercise: topology analysis

the first packet switching network



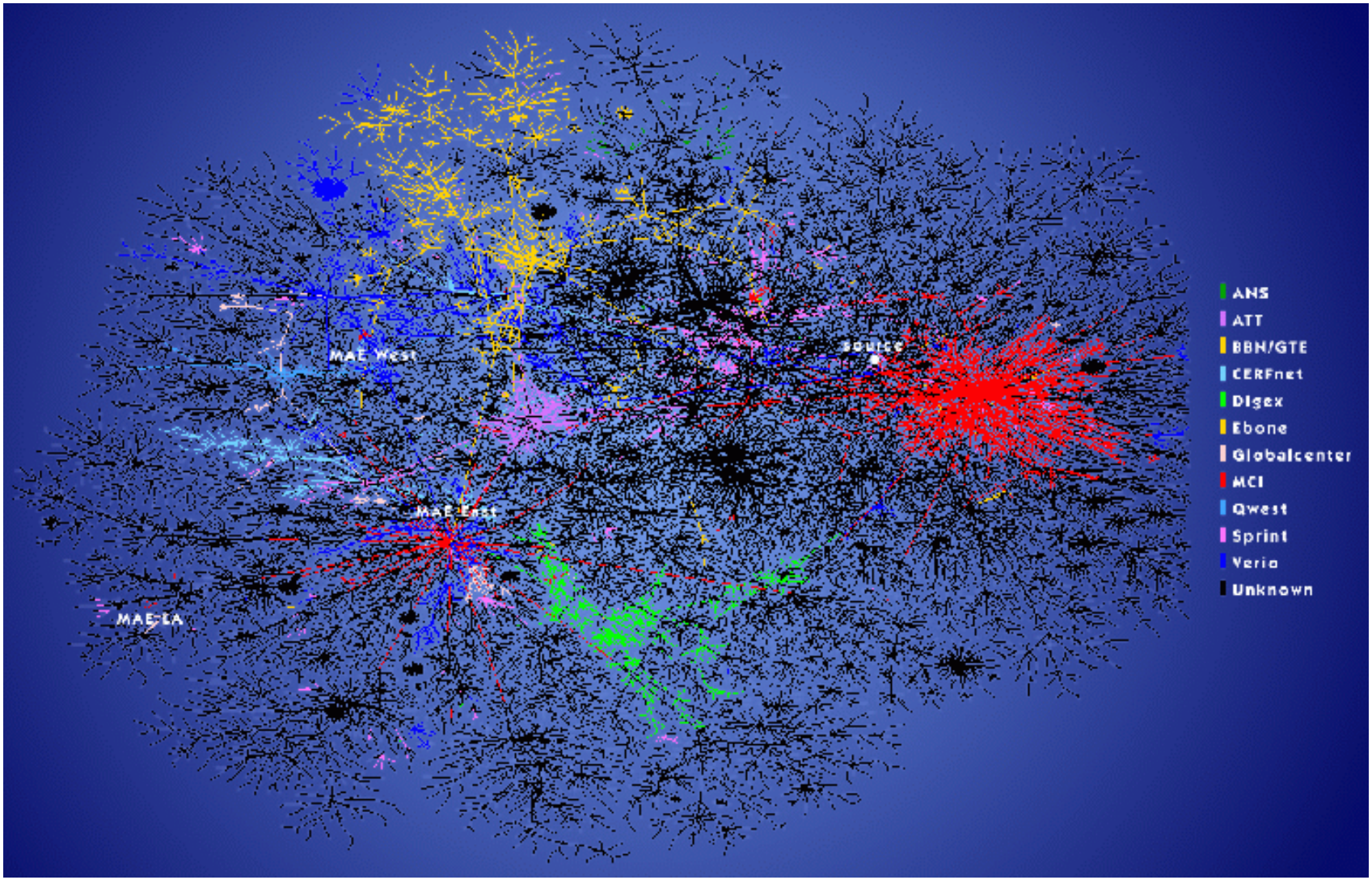
ARPANET in 1969

ARPANET, 4 years after



ARPANET in 1973

the Internet

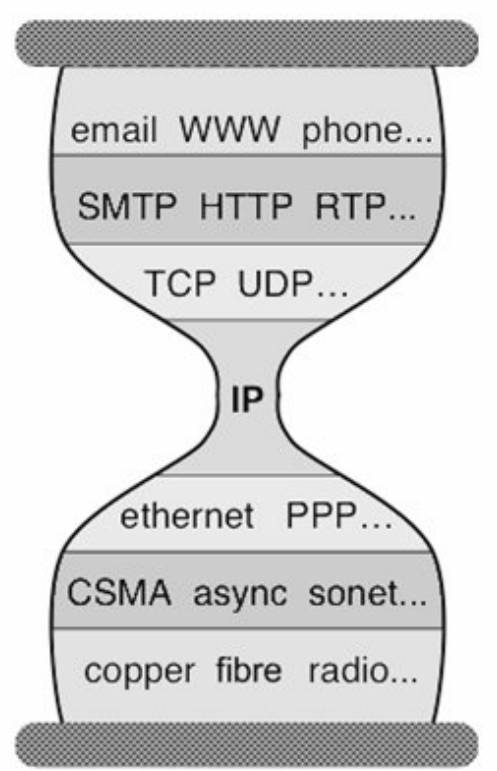


lumeta internet mapping http://www.lumeta.com

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

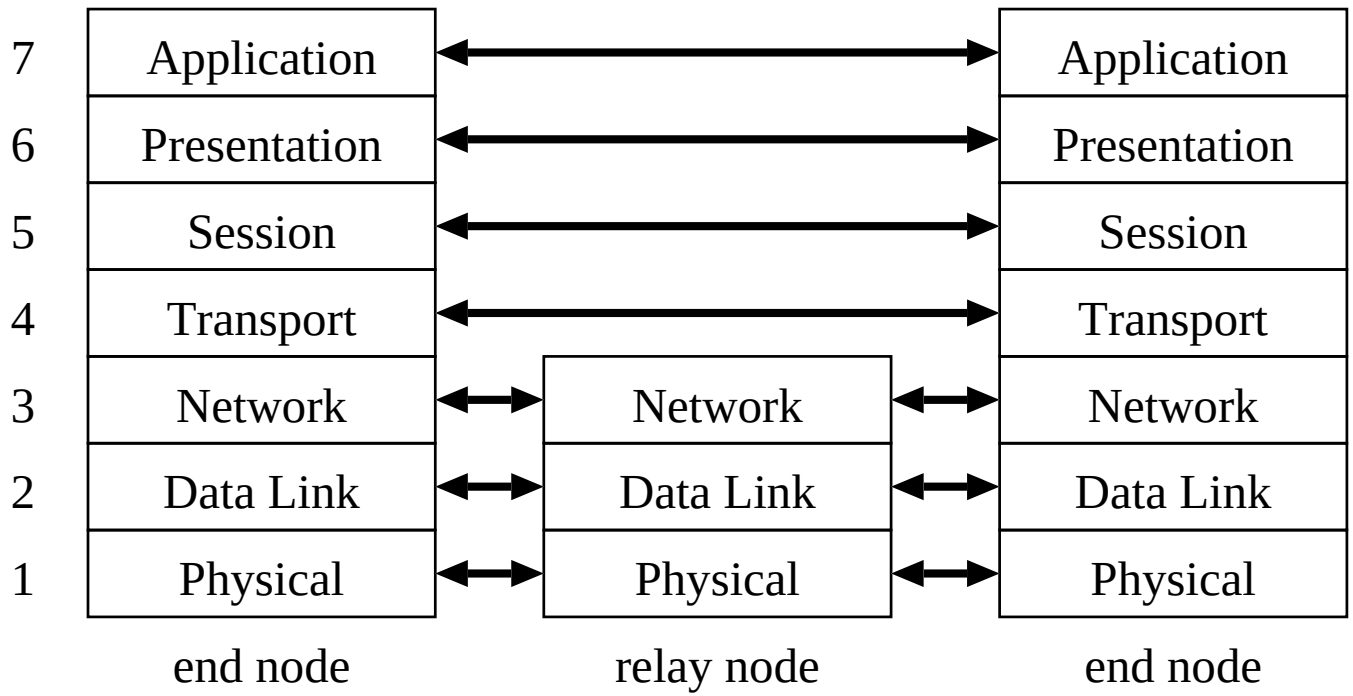


the hour glass model of the Internet architecture

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

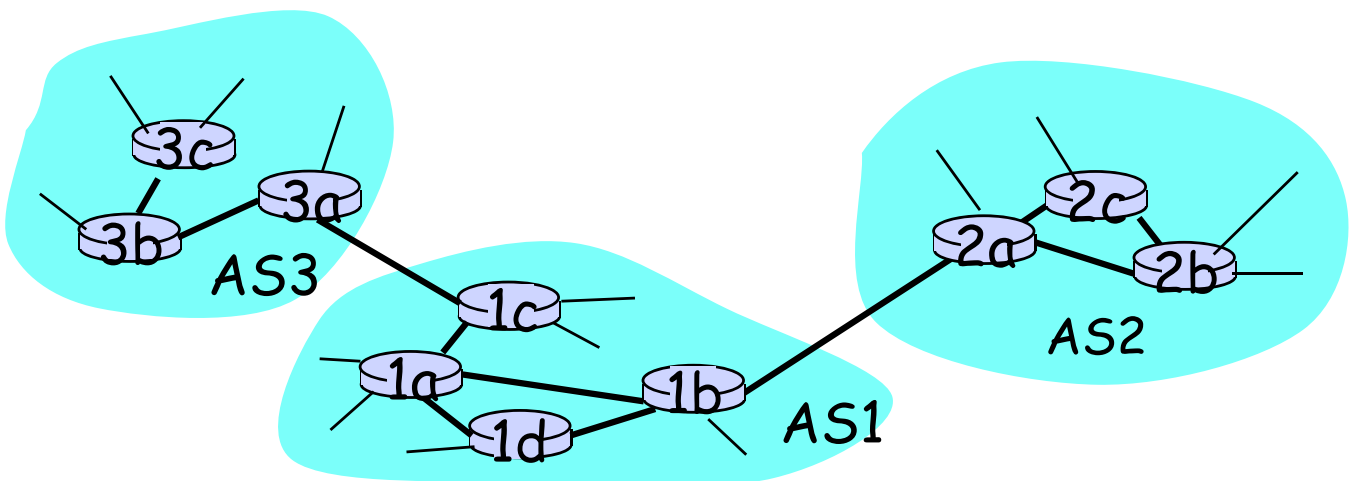


OSI 7 layer model

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)

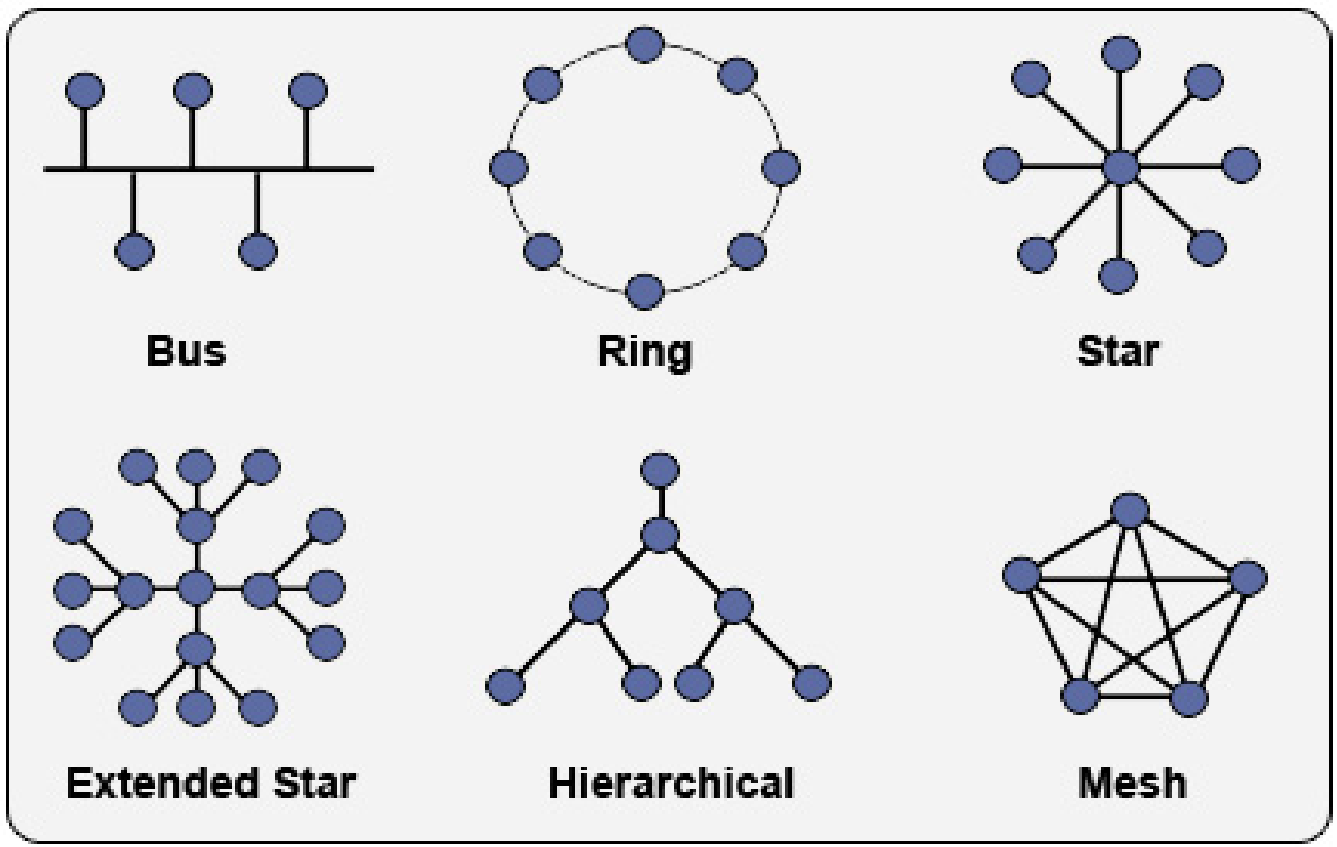
EGP (Exterior Gateway Protocol): inter-AS

- 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:

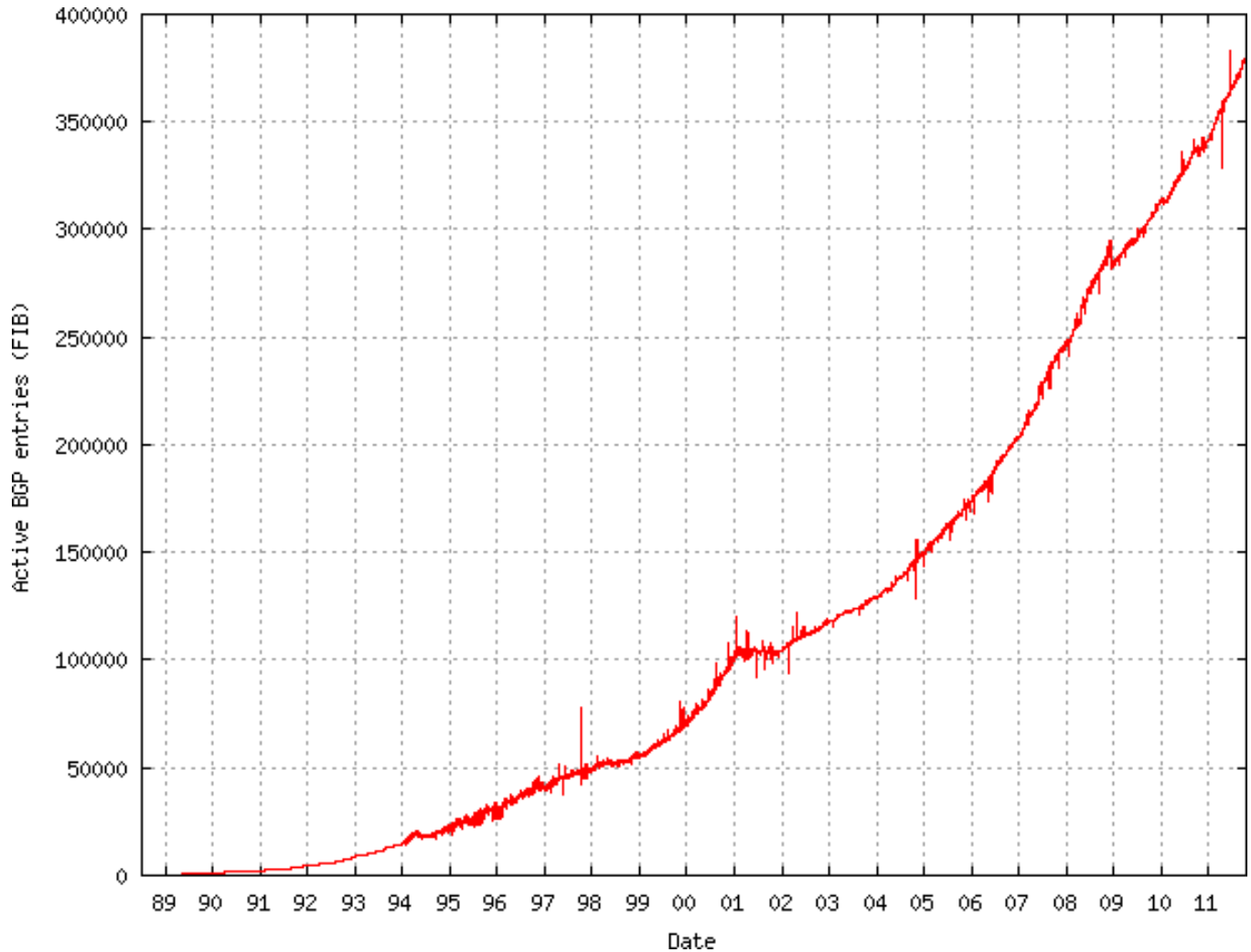
```
BGP table version is 33157262, local router ID is 198.32.162.100
Status codes: s suppressed, d damped, h history, * valid, > best, i -
internal, S Stale
Origin codes: i - IGP, e - EGP, ? - incomplete

    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

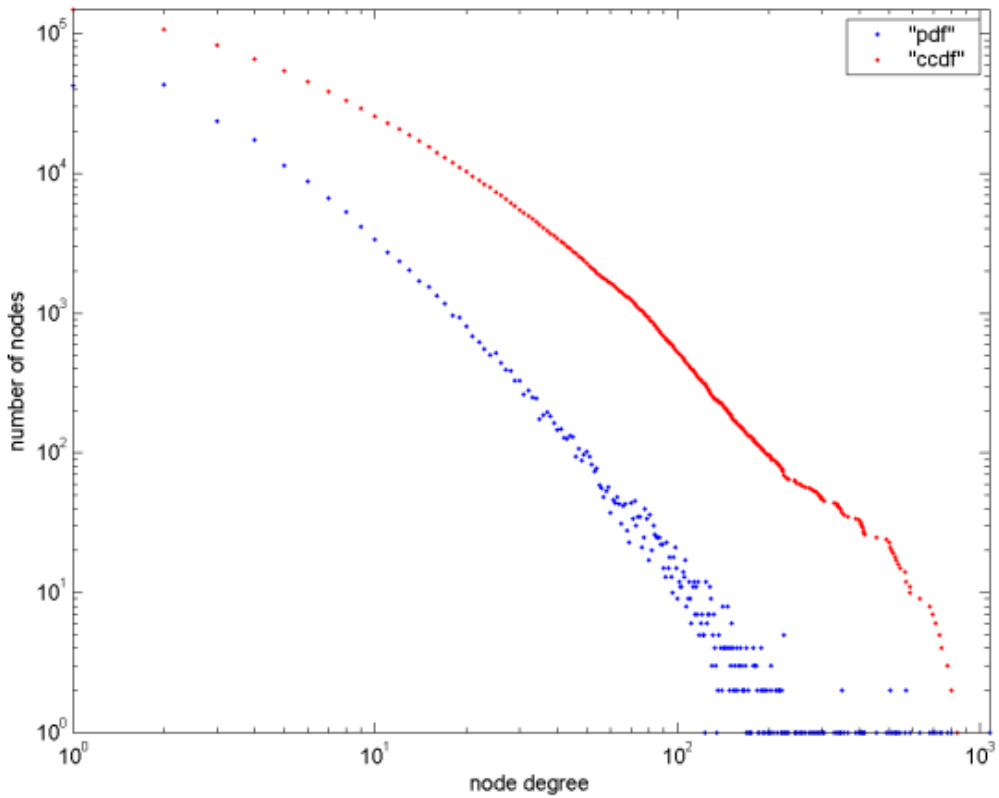
- active BGP entries (FIB): 384k on 2011/10/21



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

CAIDA's skitter/ark projects

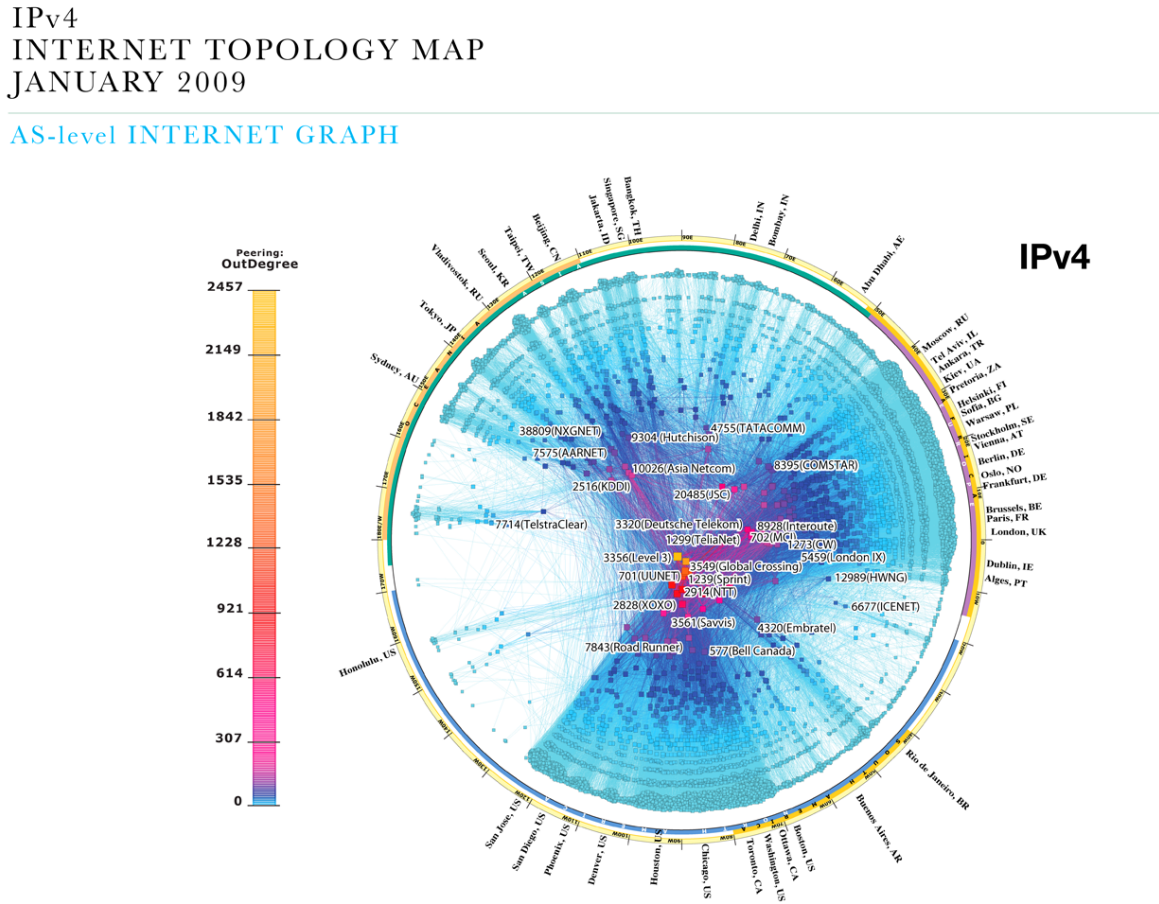
- a topology measurement project by CAIDA
  - skitter/ark: parallel execution of traceroute
  - exhaustive path search by about 20 monitors



router-level degree distribution

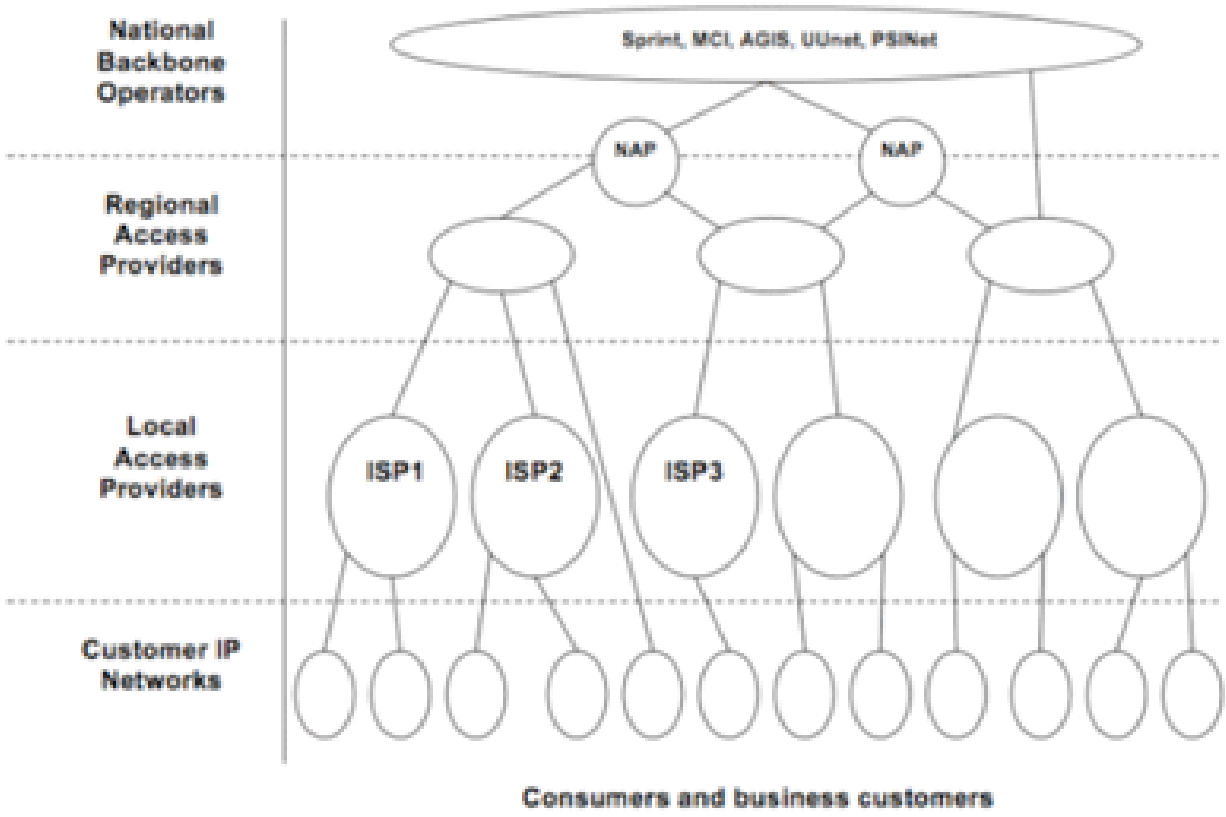
CAIDA AS CORE MAP 2009/03

- 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

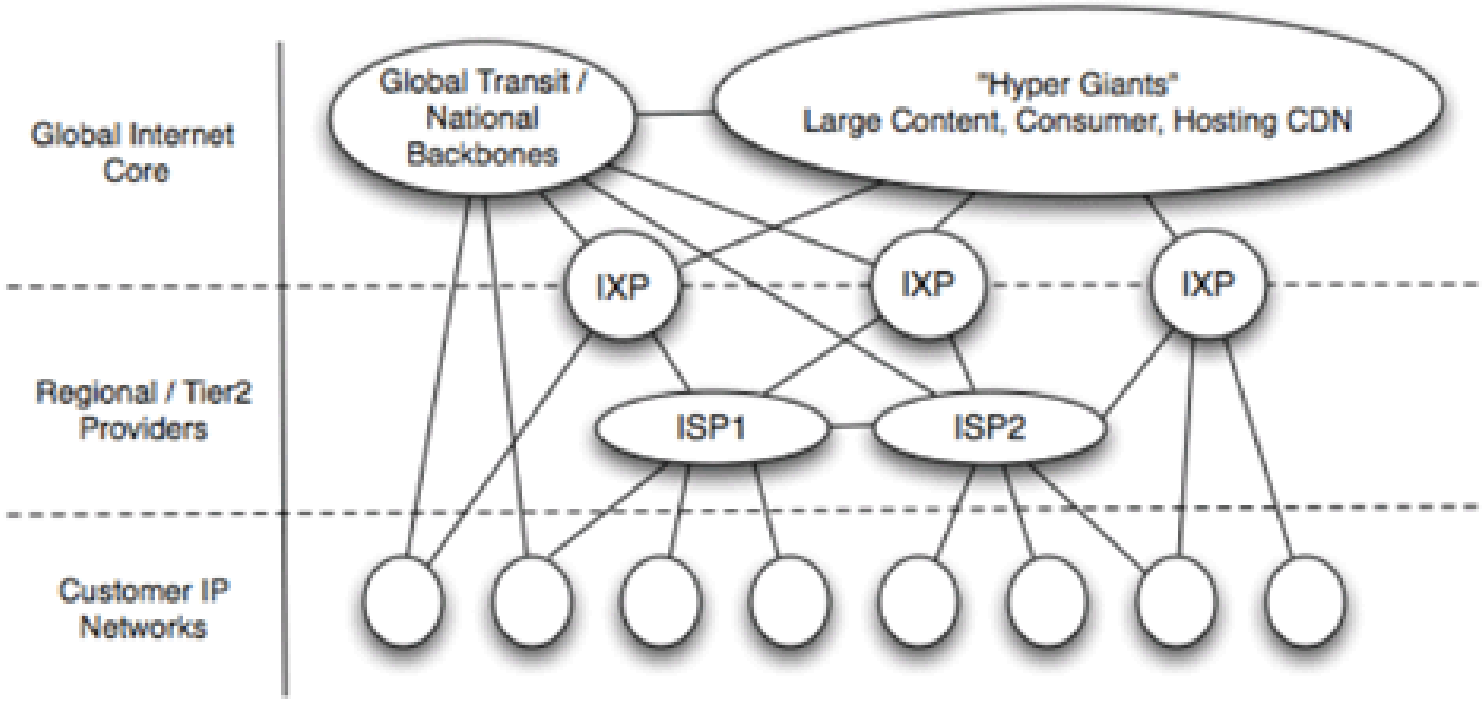


- Tier1 global core (modulo a few name changes over the years)
- Still taught today

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

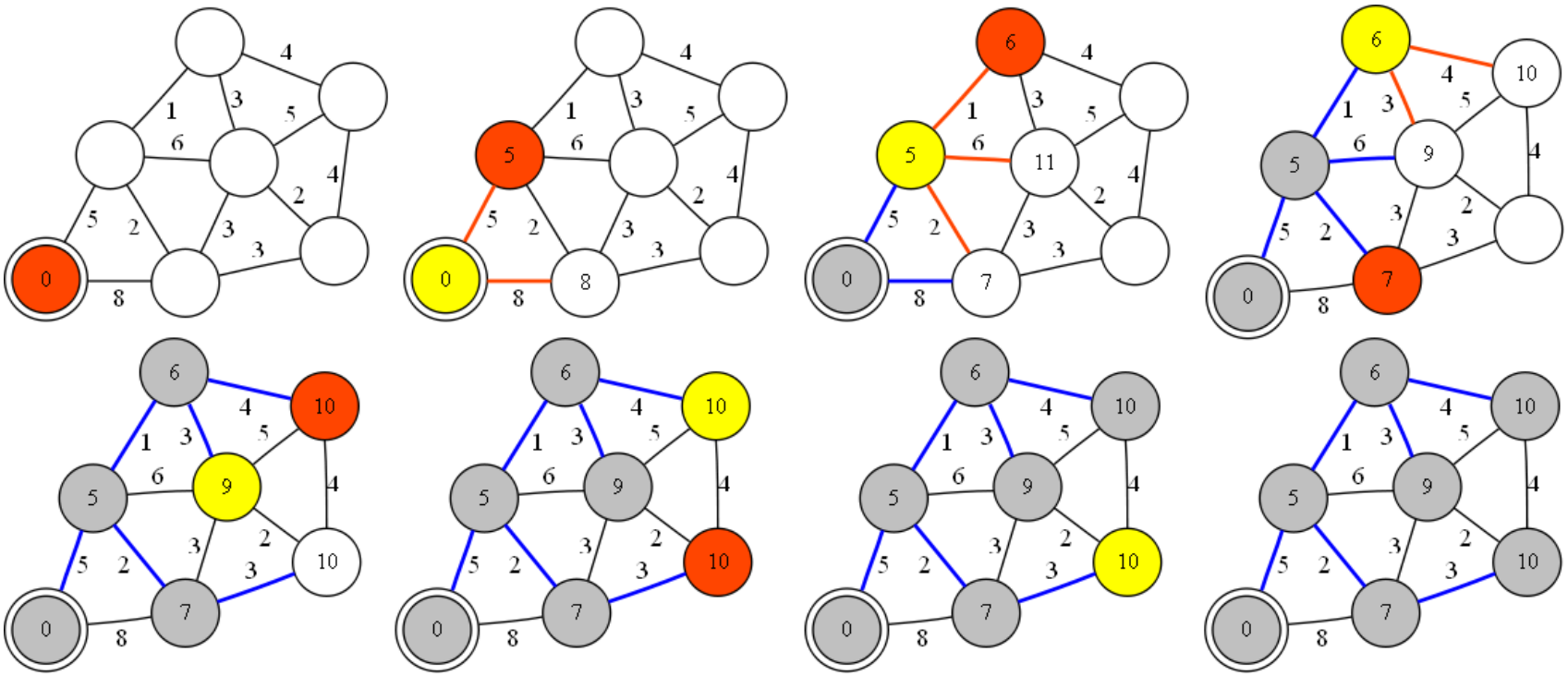
- spanning tree algorithm (loop prevention)
- shortest path algorithm (routing)
  - Bellman-Ford algorithm
  - Dijkstra algorithm

analysis of network characteristics

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

Dijkstra algorithm

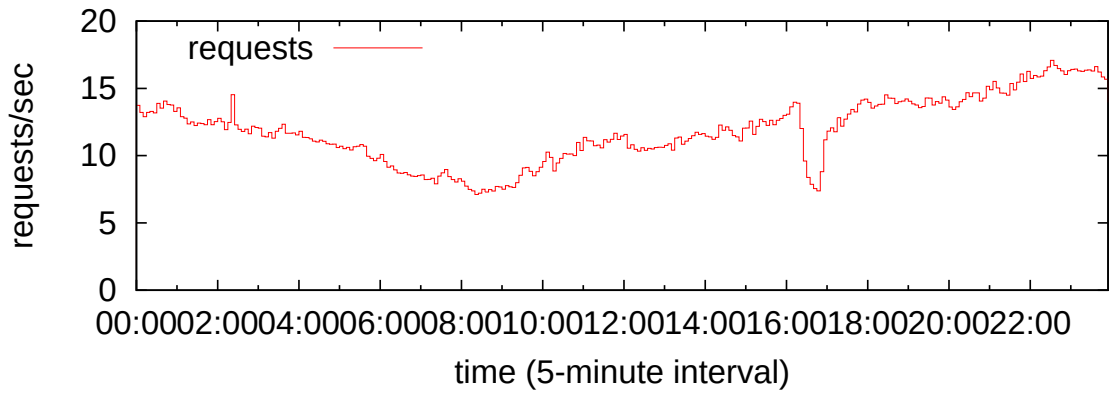
```
1. cost initialization: start_node = 0, other_nodes = infinity
2. loop:
      (1) find the node with the lowest cost among the unfinished nodes,
          and fix its cost
      (2) update the cost of its neighbors
```

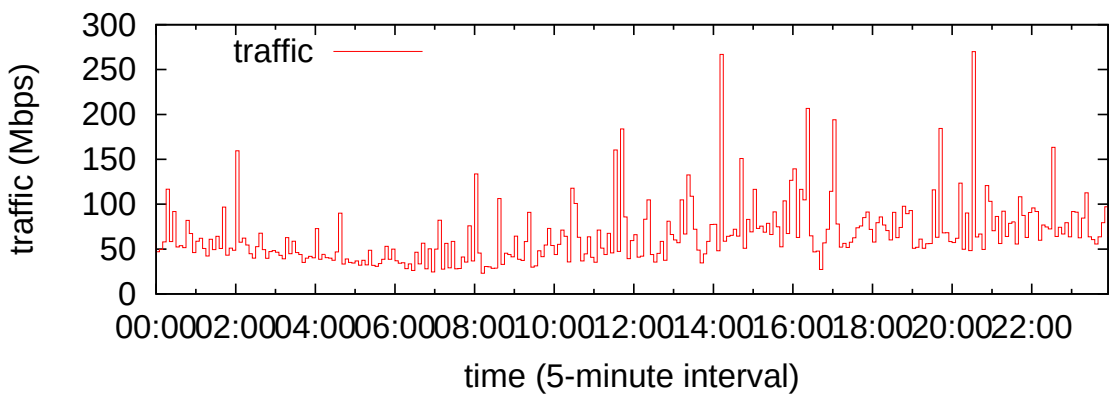


dijkstra algorithm

previous exercise: mean, standard deviation, linear regression

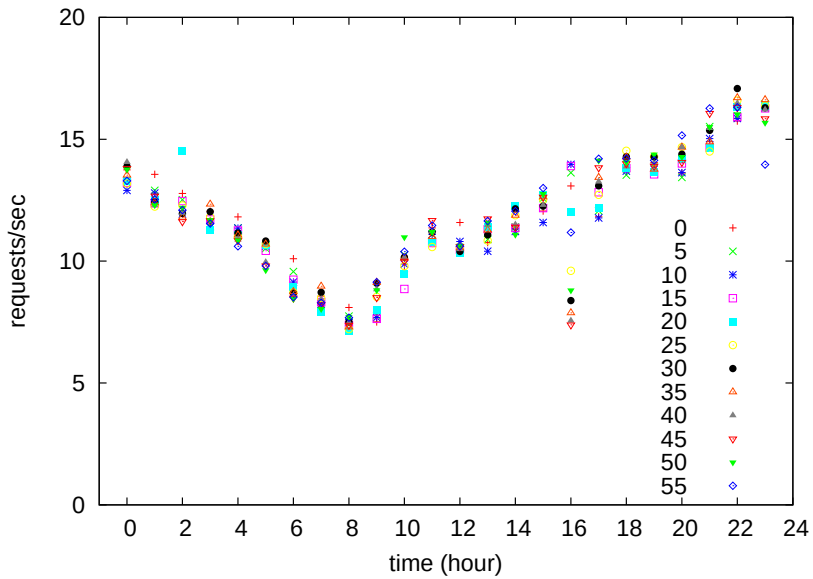
- use the 5-minute bin output from the previous class
  - remove data not for 7/19
- focus on the request counts (leave transferred bytes for the assignment)
- use 12 5-minute bins for an hour as 12 samples per hour

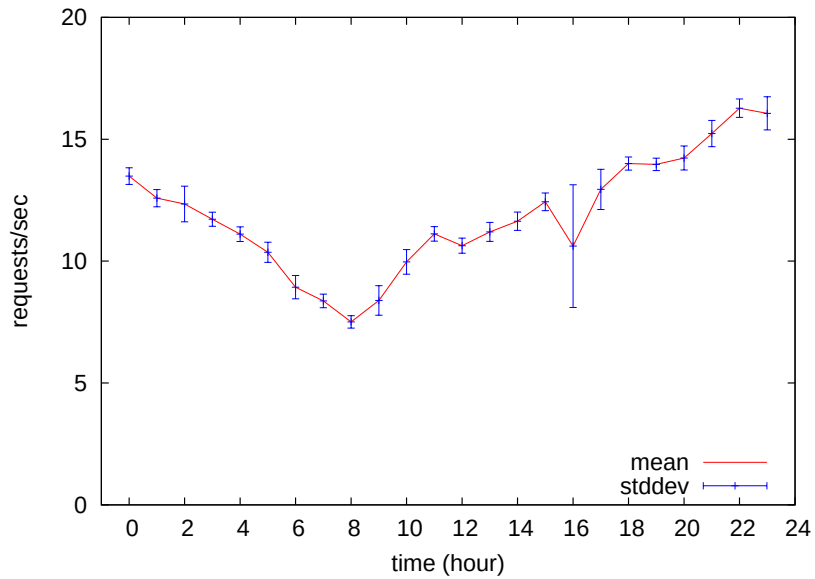




previous exercise: graphs for request counts

- graph 1: 12 samples per hour, for 24 hours
- graph 2: mean and standard deviation for 24 hours





previous exercise: compute mean and standard deviation

```
# extract a variable from each line
re = /^\S+\s+(\d+)\s+\d+/

# create an array for data
data = Array.new
ARGF.each_line do |line|
  if re.match(line)
    data.push $1.to_i
  end
end

# compute mean
sum = 0
data.each {|v| sum += v}
mean = Float(sum) / data.length

# compute standard deviation
sqsum = 0
data.each {|v| sqsum += (v - mean)**2}
var = Float(sqsum) /data.length
stddev = Math.sqrt(var)

puts "mean=#{mean} stddev=#{stddev}"
```

previous exercise: creating data table for request counts

create an hourly data table for plotting

- row: hourly data (0 .. 23)
- column: hour samples(00 05 10 ... 55) mean stddev

```
#hour  00     05     10    ...     55    mean     stddev
 0    4123   3963   3871    ...   3987   4046.8    102.3
 1    4068   3871   3838    ...   3760   3774.9    106.2
 2    3833   3755   3580    ...   3628   3703.6    219.0
 3    3614   3433   3418    ...   3462   3515.5     86.2
...
22    4724   4790   4757    ...   4893   4882.2    113.4
23    4922   4932   4889    ...   4188   4818.9    203.8
```

previous exercise: table creation code

```
re = /^(\d{4}-\d{2}-\d{2})T(\d{2}):(\d{2})\s+(\d+)\s+(\d+)/

hourly = Array.new(24){ Array.new(12) } # hourly[hour][min]
ARGF.each_line do |line|
  if re.match(line)
    day, hour, min, requests, bytes = $~.captures
    hourly[hour.to_i][min.to_i / 5] = requests.to_i
  end
end
hourly.each_index do |h|
  printf "%2d ", h
  sum = n = 0
  hourly[h].each_index do |m|
    printf "%6d ", hourly[h][m]
    sum += hourly[h][m]
    n += 1
  end
  mean = Float(sum) / n
  printf "%8.1f ", mean
  var = 0
  hourly[h].each_index do |m|
    var += (hourly[h][m] - mean) ** 2
  end
  var = var / n
  printf "%8.1f\n", Math.sqrt(var)
end
```

previous exercise: plot commands

For time-slots data

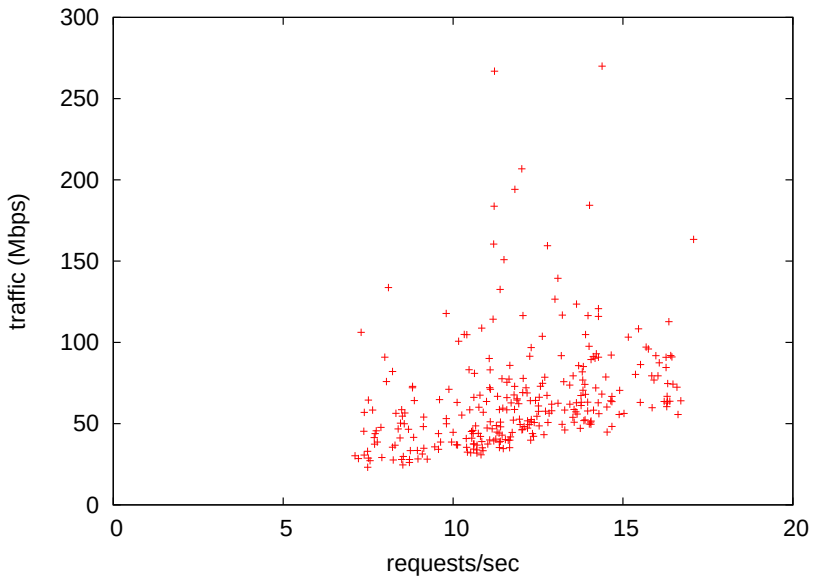
```
set xlabel "time (hour)"
set xrange [-1:24]
set yrange [0:20]
set xtic 2
set key bottom right
set ylabel "requests/sec"
plot "request-table.txt" using 1:($2/300) title '0' with points, \
"request-table.txt" using 1:($3/300) title '5' with points, \
"request-table.txt" using 1:($4/300) title '10' with points, \
"request-table.txt" using 1:($5/300) title '15' with points, \
"request-table.txt" using 1:($6/300) title '20' with points, \
"request-table.txt" using 1:($7/300) title '25' with points, \
"request-table.txt" using 1:($8/300) title '30' with points, \
"request-table.txt" using 1:($9/300) title '35' with points, \
"request-table.txt" using 1:($10/300) title '40' with points, \
"request-table.txt" using 1:($11/300) title '45' with points, \
"request-table.txt" using 1:($12/300) title '50' with points, \
"request-table.txt" using 1:($13/300) title '55' with points
```

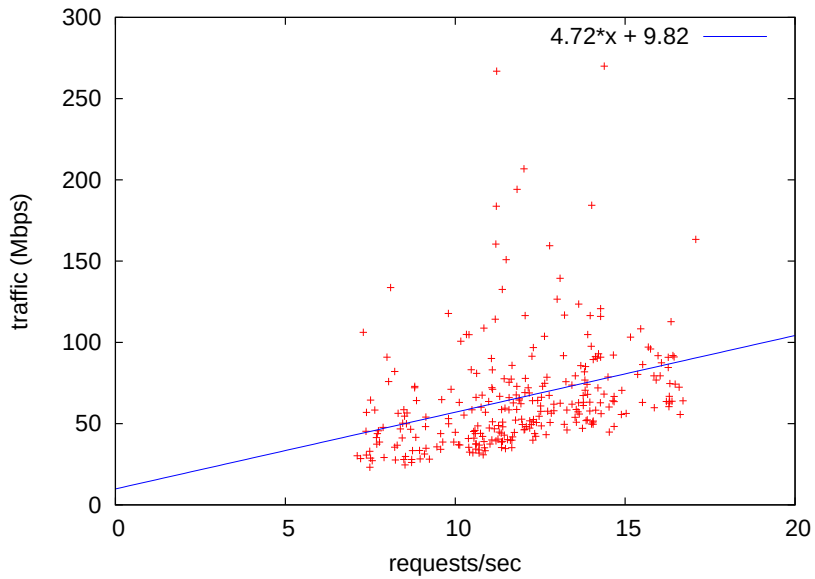
For mean and stddev:

```
set xlabel "time (hour)"
set xrange [-1:24]
set yrange [0:20]
set xtic 2
set key bottom right
set ylabel "requests/sec"
plot "request-table.txt" using 1:($14/300) title 'mean' with lines, \
"request-table.txt" using 1:($14/300):($15/300) title 'stddev' with errorbars lt 3
```

previous exercise: linear regression

- relationship of request count and traffic of 5-min bins
- linear regression by the least square method





previous exercise: linear regression code

```
# extract 2 variables from each line
re = /^\S+\s+(\d+)\s+(\d+)/

# compute (y = b0 + b1*x) by the least square method
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 / 300  # req/5-min to req/sec
      y = $2.to_f * 8 / 300 / 1000000 # bytes/5min to Mbps

      sum_x += x
      sum_y += y
      sum_xx += x * x
      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 * mean_x)
b0 = mean_y - b1 * mean_x

puts "b0=#{b0} b1=#{b1}"
```

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
%
```

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
```

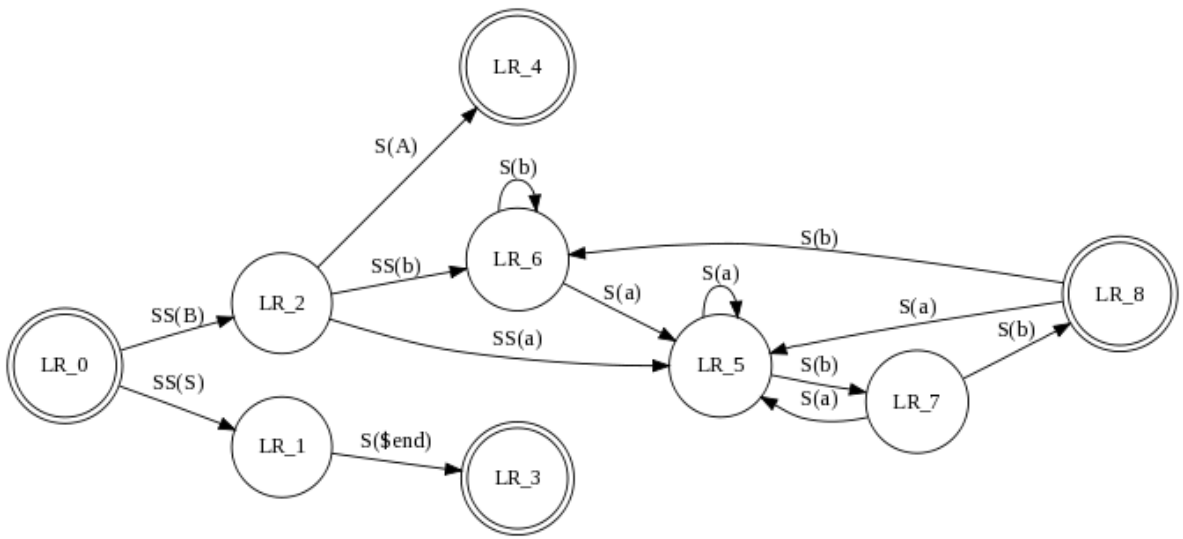
sample code (4/4)

```
# 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 5 Measuring the structure of the Internet

- Internet architecture
- network layers
- topologies
- graph theory
- exercise: topology analysis

next class

Class 6 Measuring the characteristics of the Internet (11/2)

- delay, packet loss, jitter
- correlation and multivariate analysis
- principal component analysis
- exercise: correlation analysis