インターネット計測とデータ解析 第5回

長 健二朗

2011年6月8日

前回のおさらい

インターネットの速度を計る

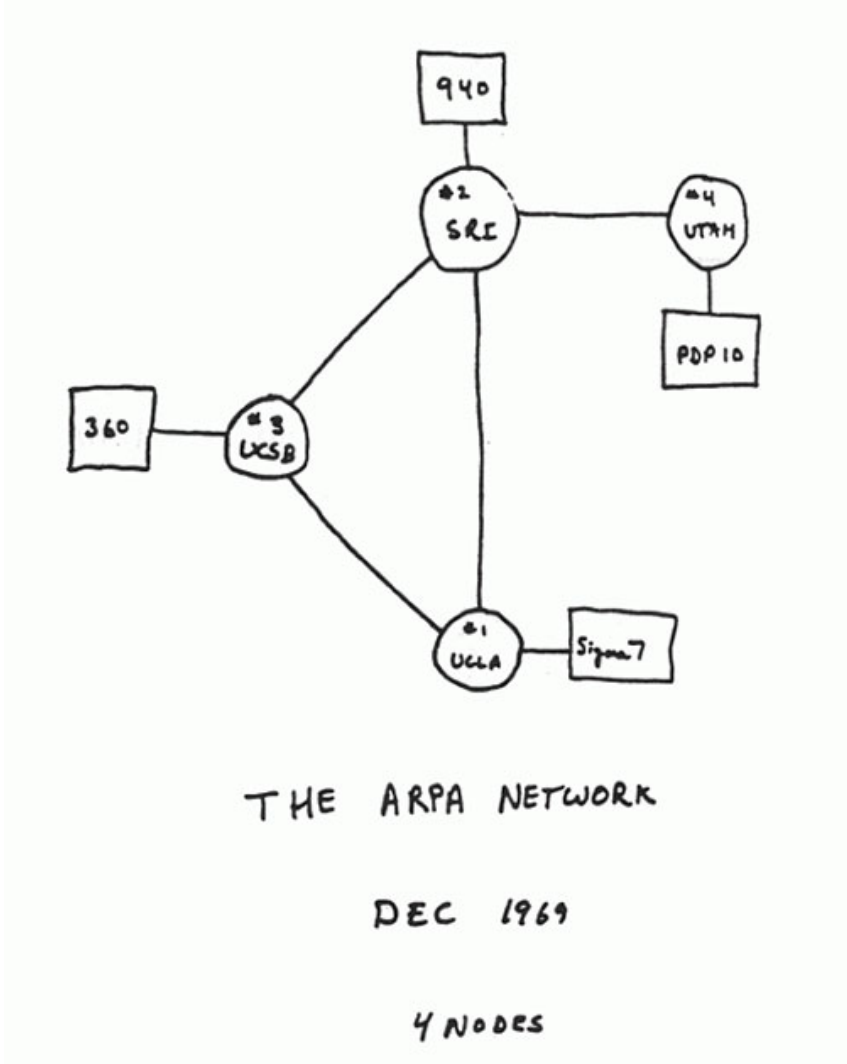
- 速度計測
- 利用可能帯域の推測
- 平均 標準偏差
- 線形回帰
- 演習:平均、標準偏差、線形回帰
- **課題 1**

今日のテーマ

インターネットの構造を計る

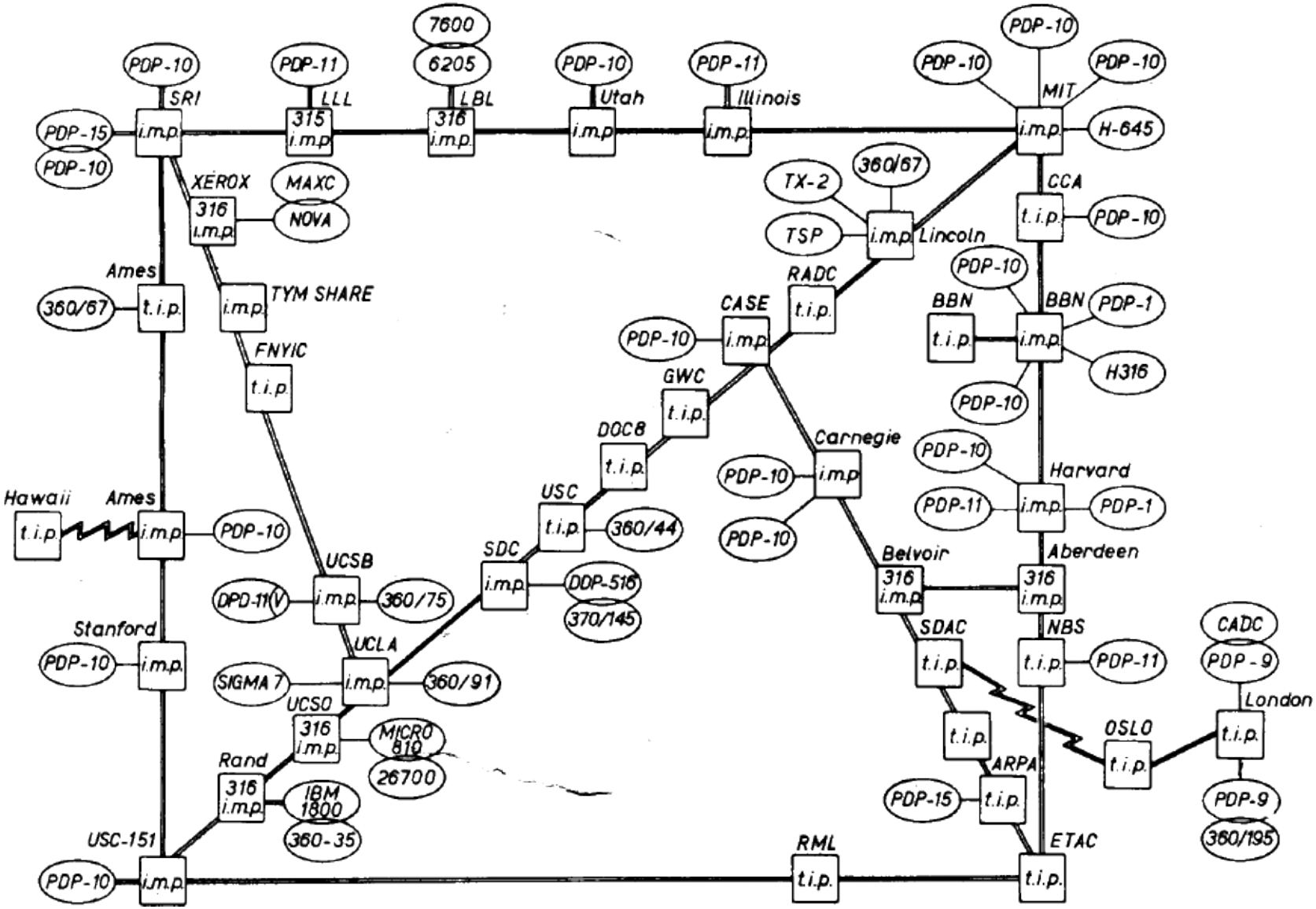
- インターネットアーキテクチャ
- ネットワーク階層
- トポロジー
- グラフ理論
- 演習:トポロジ解析

最初のパケットスイッチングネットワーク



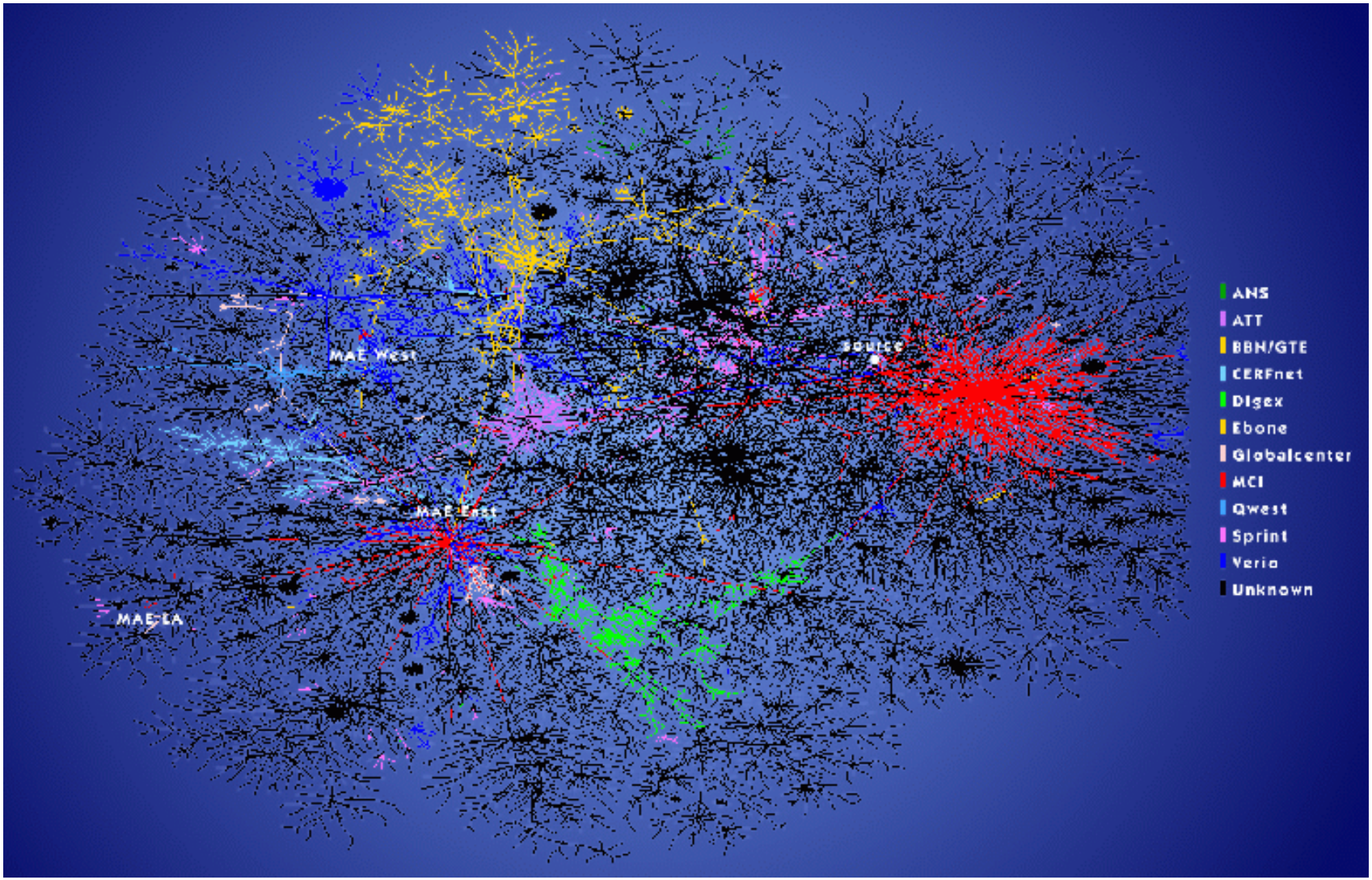
ARPANET in 1969

4 年後の ARPANET



ARPANET in 1973

インターネット

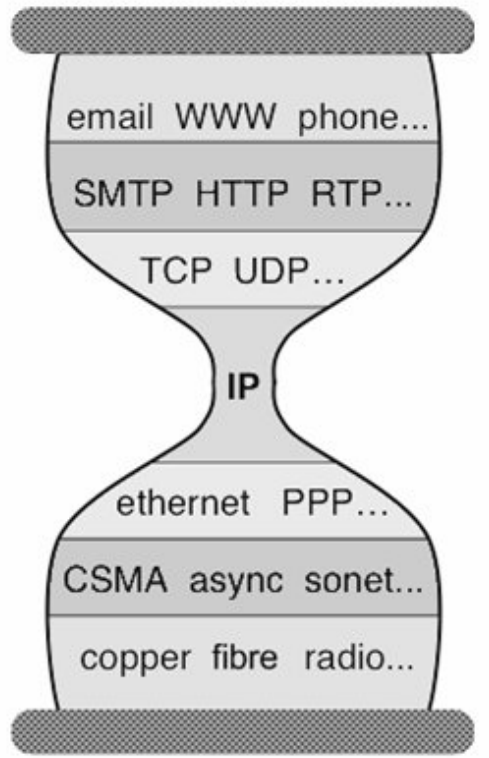


lumeta internet mapping http://www.lumeta.com

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

インターネットアーキテクチャ

- パケット配送を IP で共通化
  - 多様な上位層と下位層をサポート
- エンド・ツー・エンド モデル
  - シンプルなネットワーク、機能はエンドノードで実現

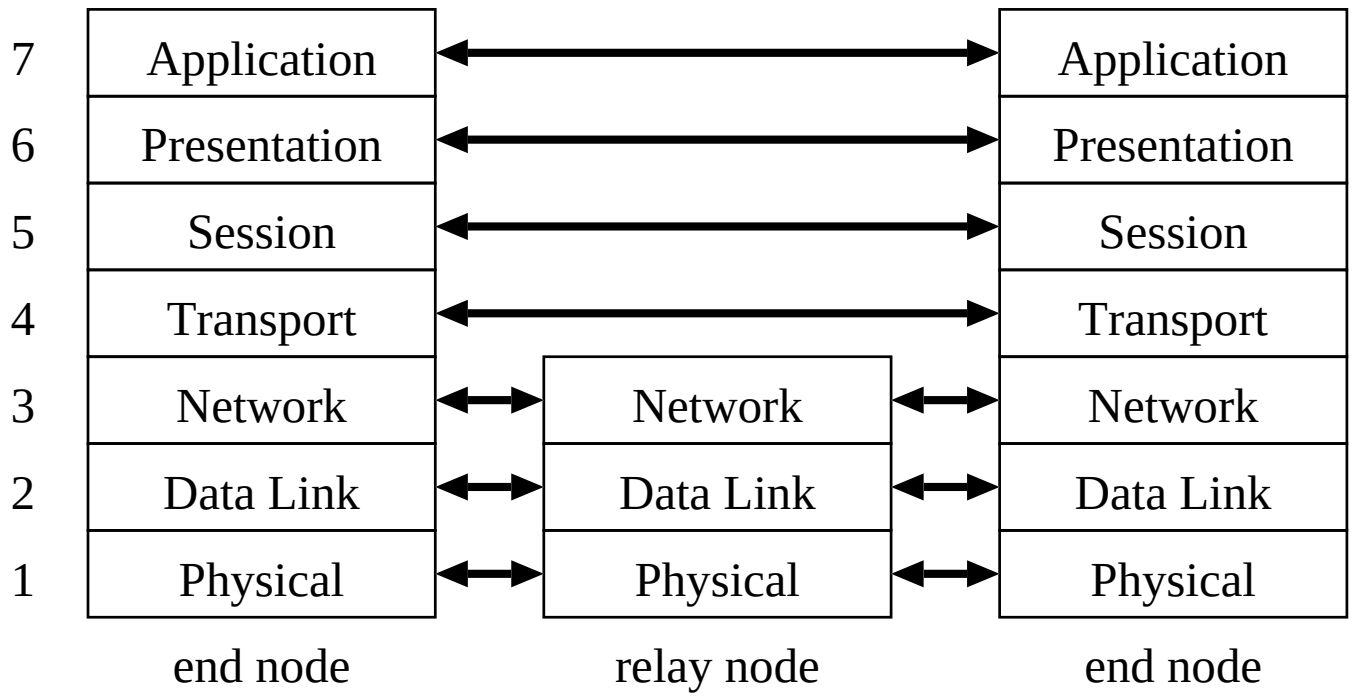


インターネットアーキテクチャの砂時計モデル

ネットワーク階層モデル

複雑なシステムを機能別レイヤ (階層) に分けて抽象化

- ネットワーク層 (L3)
  - パケット配送: パケットを受け取り、転送
  - 経路制御: 宛先に応じて、転送先を決定する仕組み

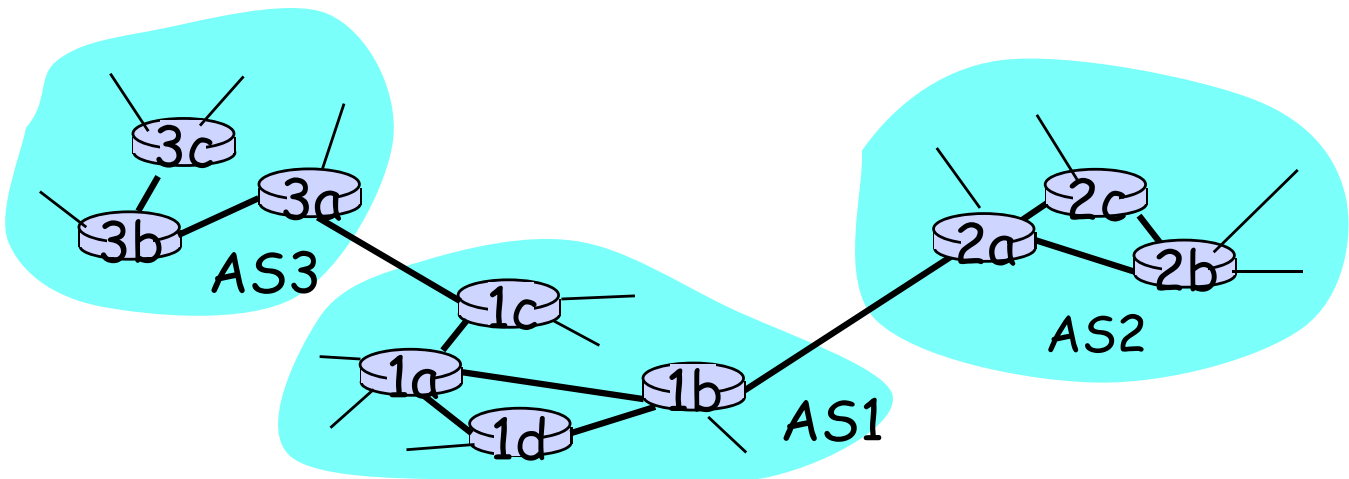


OSI 7 階層モデル

経路制御アーキテクチャ

階層的経路制御 (hierarchical routing)

- Autonomous System (AS): 経路制御上のポリシの単位 (組織)
  - Keio University: AS38635
  - WIDE Project: AS2500
  - SINET: AS2907
- インターネットの経路制御は AS 内部と AS 間の 2 階層
  - スケーラビリティ
  - AS 間は管理ポリシーの異なるネットワークを繋ぐ
    - 内部情報の隠蔽や運用ポリシーの反映が必要



経路制御プロトコル

隣接ルータと経路情報を交換、自身の持つ経路情報を更新する

IGP (Interior Gateway Protocol): AS 内部で使用

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

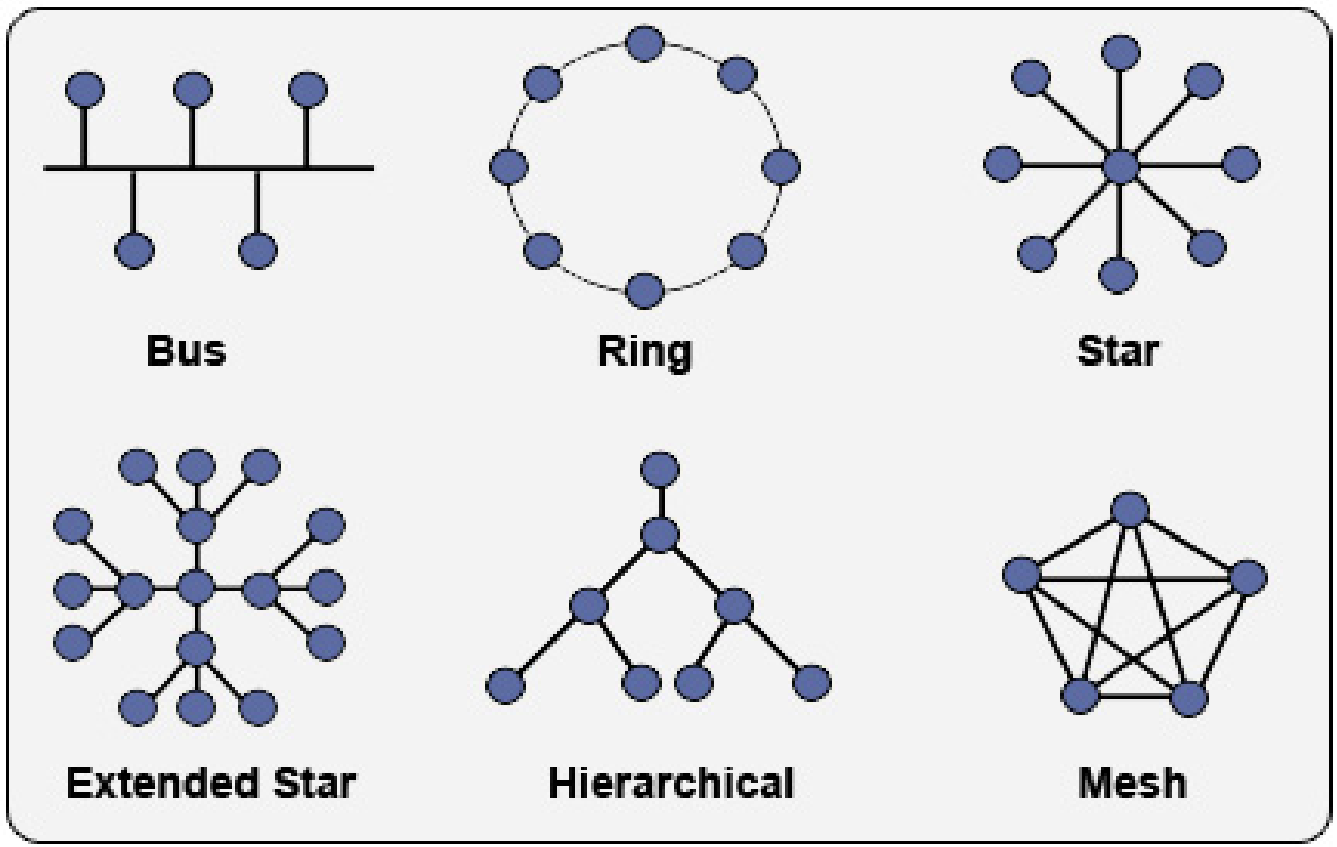
EGP (Exterior Gateway Protocol): AS 間で使用

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

トポロジ (topology)

トポロジー (ネットワーク構造)

- 簡単なトポロジ
  - バス、リング、スター、ツリー、メッシュ
- さまざまなレベルでのトポロジ
  - 物理配線、レイヤ 2、IP レベル、オーバレイ
  - ハイパーリンク、ソーシャルネットワーク



インターネットのトポロジ

インターネットスケールのトポロジ情報

- ルータレベル
  - traceroute
  - データプレーン情報
  - データの収集公開:
    - skitter/ark (CAIDA): 20 程のモニターから定点観測
    - iPlane (U. Washington): PlanetLab の利用
    - DIMES (Tel Aviv U.) エンドユーザによる計測
- AS レベル
  - BGP 経路表
  - コントロールプレーン情報
  - データの収集公開: RouteViews (U. Oregon), RIPE RIS

traceroute

- IP パケットのループ検出のための TTL (time-to-live) を利用
  - ルータはパケット転送時に TTL を 1 減らす
  - TTL が 0 になると ICMP TIME EXCEEDED を送信者に返す
- 制約
  - 経路は時間とともに変化する可能性
  - 非対称な経路も存在する
    - 行きのパスしか分からない
  - 通常ルータはインターフェイス毎に IP アドレスを持つ
    - IP アドレスだけでは同一ルータか判定できない

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 情報

- 各 AS はポリシーに従って隣接 AS に経路を広告
  - AS パスに自 AS をプリペンド
  - ポリシー: どの AS にどの経路をどのように広告するか
- BGP データ: 経路表のダンプ、アップデート情報
- BGP データのサンプル:

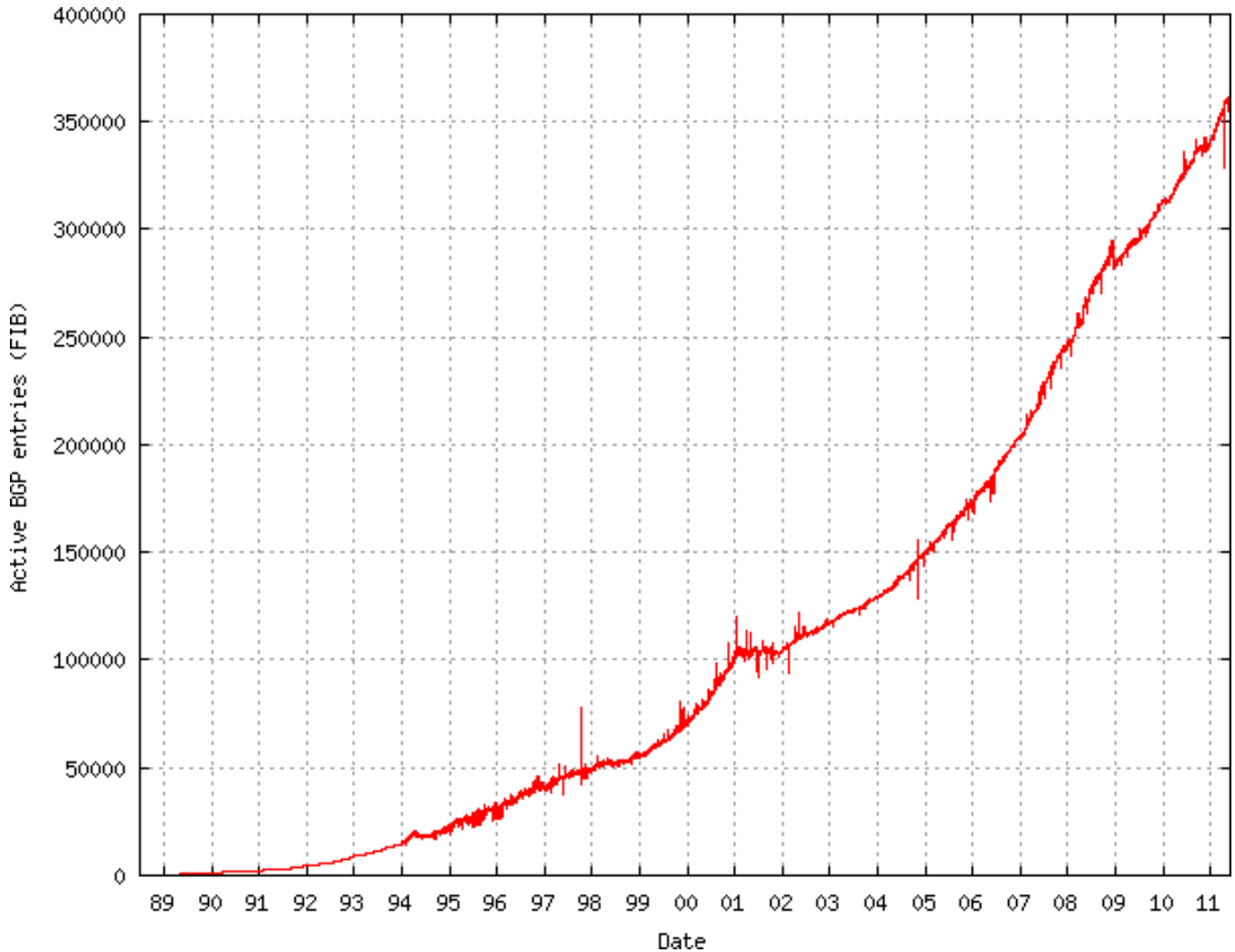
```
BGP table version is 33157262, local router ID is 198.32.162.100
Status codes: s suppressed, d damped, h history, * valid, > best, i -
internal, S Stale
Origin codes: i - IGP, e - EGP, ? - incomplete

    Network              Next Hop    Metric LocPrf Weight Path

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

RouteViews プロジェクト

- University of Oregon によるデータ収集公開プロジェクト
  - http://www.routeviews.org/
- 10 のコレクタ: メジャーな AS からのデータ提供
- 1997 年からのデータを蓄積、公開

経路表サイズの推移

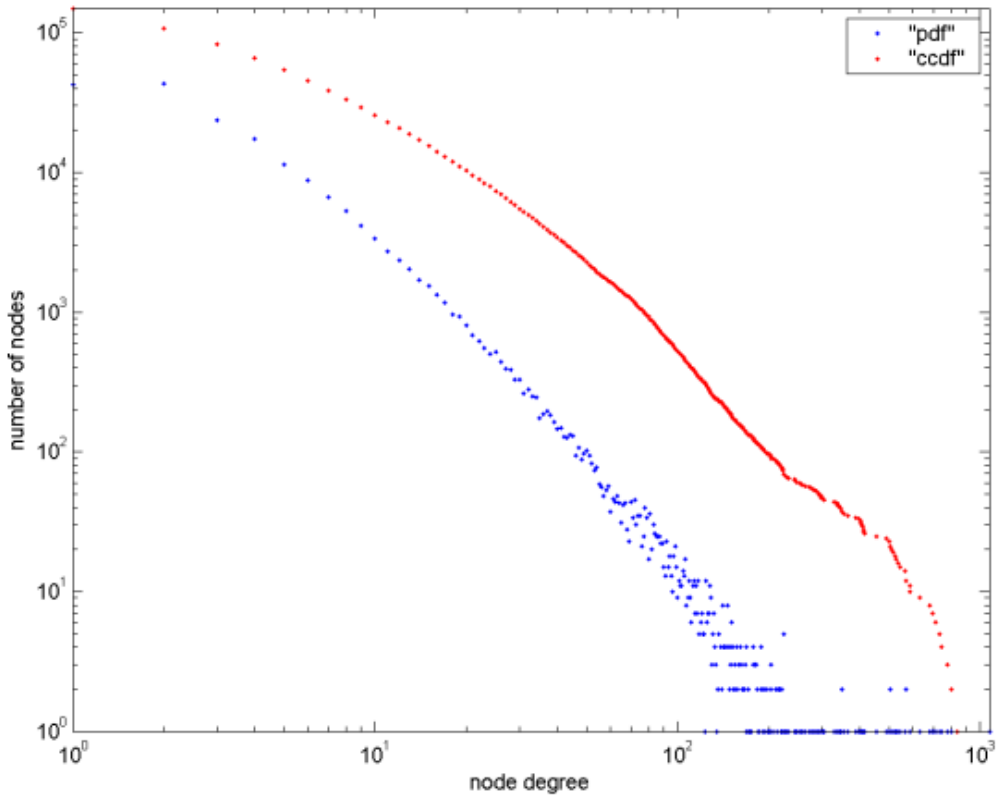
- active BGP entries (FIB): 362k on 2011/5/31



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

CAIDA's skitter/ark projects

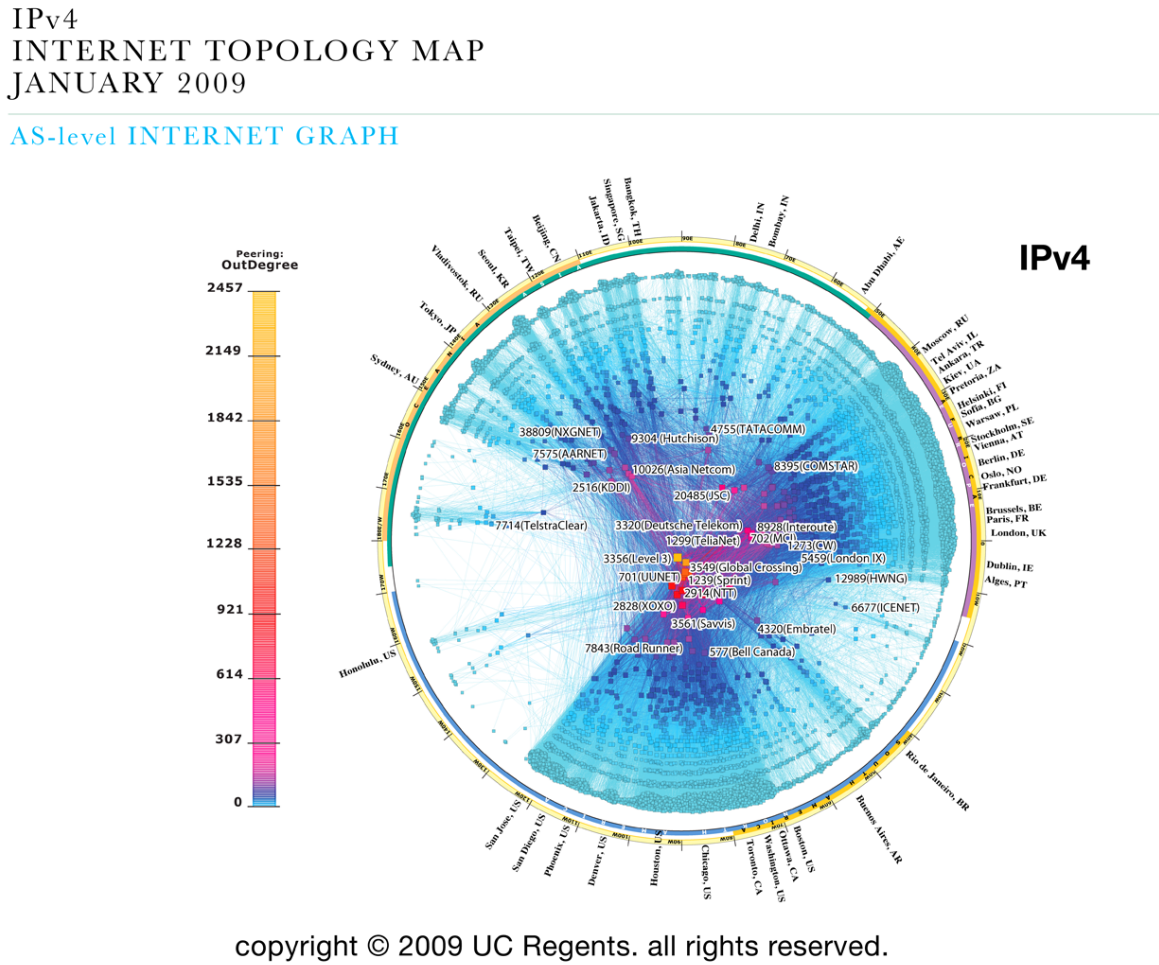
- CAIDA によるトポロジー計測
  - skitter/ark: traceroute を並列実行
  - 約 20 のモニターが全域へのパスを調査



ルータレベルの次数分布

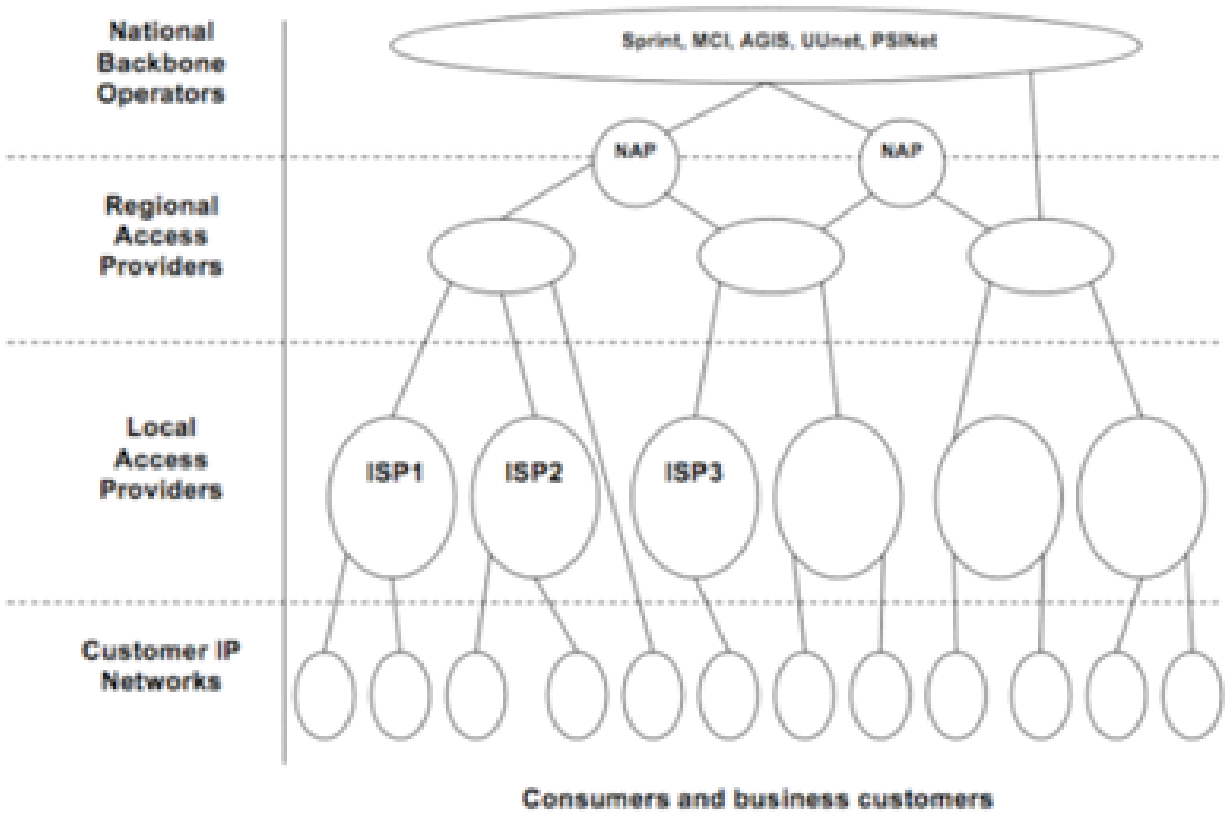
CAIDA AS CORE MAP 2009/03

- skitter/ark データによる AS 接続の可視化
- AS の登録都市の経度、AS の out-degree



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

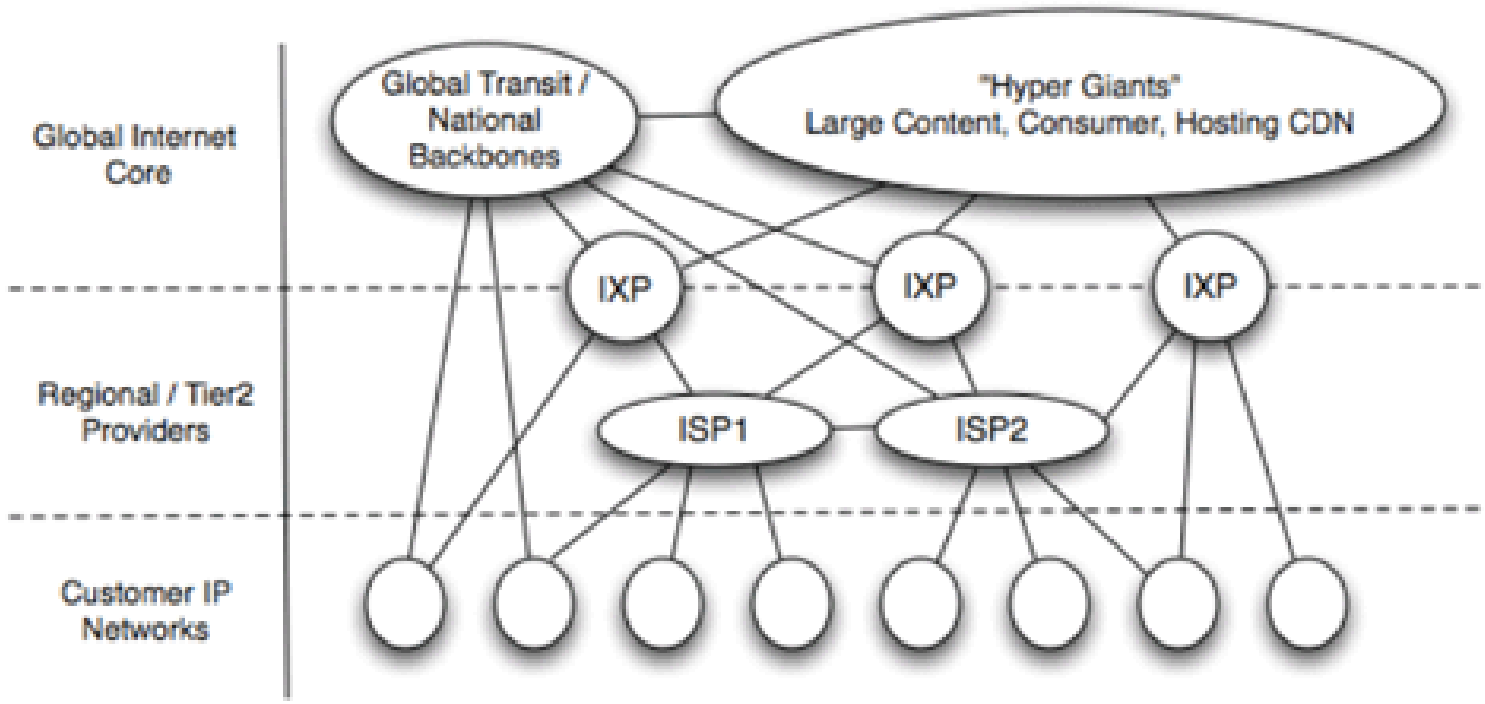
インターネット AS 階層



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

source: 2009 Internet Observatory Report (NANOG47)

インターネット AS階層 最近の変化



- **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)

グラフ理論

トポロジはグラフ理論で表現可能

- グラフはノード (node or vertex) とエッジ (edge) から構成
- 無向グラフと有向グラフ: エッジが方向を持つかどうか
- 重み付きグラフ: エッジに重み (コスト) が付く
- パス (path): 2 つのノード間の経路
- 部分グラフ (subgraph):
- 次数 (degree): ノードに接続するエッジ数

ネットワークアルゴリズムへの応用

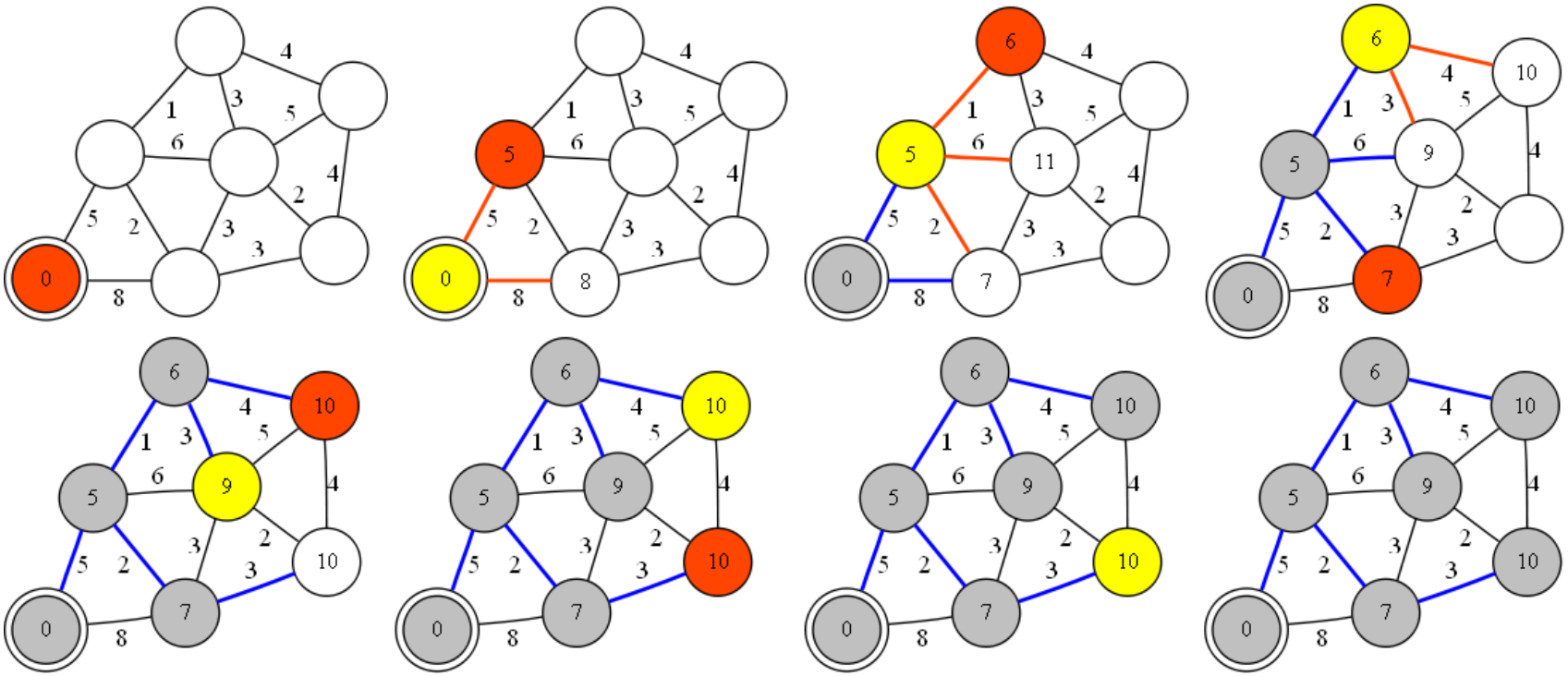
- スパニングツリーの作成 (ループ回避)
- 最短経路探索 (経路制御)
  - Bellman-Ford アルゴリズム
  - Dijkstra アルゴリズム

ネットワークの特徴解析

- クラスタリング
- 平均最短距離 (スモールワールド)
- 次数分布解析 (スケールフリー:次数分布がべき乗)

Dijkstra アルゴリズム

1．初期化：スタートノード値 = 0、他のノード値 = 未定義
2．ループ：
　　（1）未確定ノード中、最小値のノードを確定
　　（2）確定したノードの隣接ノードのコスト更新



dijkstra algorithm

演習: Dijkstra アルゴリズム

- トポロジファイルを読んで、最短経路木を計算する

```
% 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
```

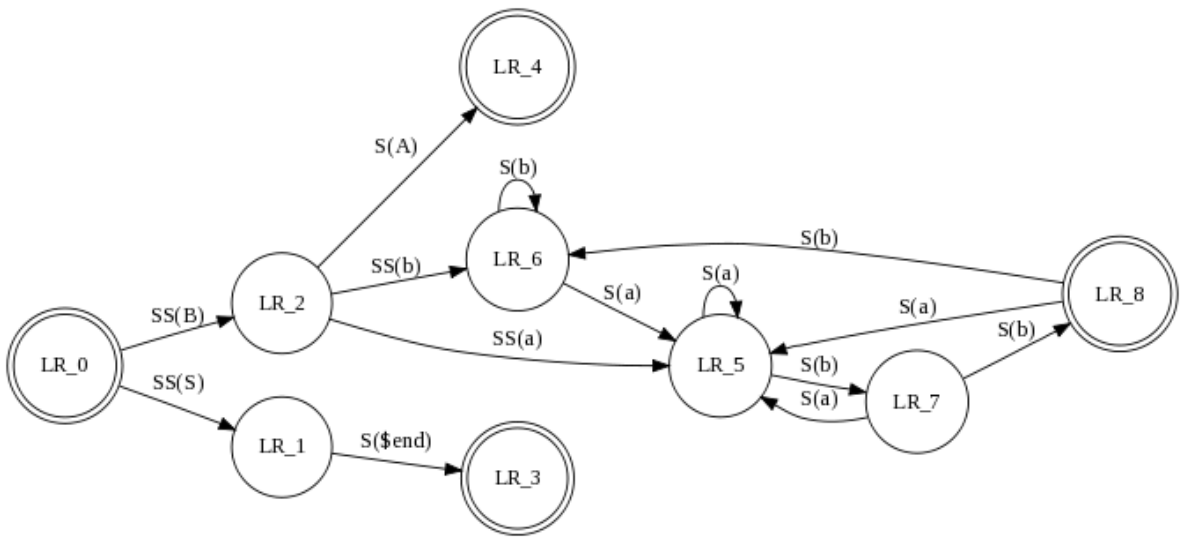
sample code (4/4)

```
# 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
```

グラフ理論的なグラフ描画ツール

- ノードとエッジの関係を定義すればレイアウト
- graphviz (http://www.graphviz.org/) の例

```
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)" ];
}
```



まとめ

インターネットの構造を計る

- インターネットアーキテクチャ
- ネットワーク階層
- トポロジー
- グラフ理論
- 演習:トポロジ解析

次回予定

第 6 回 インターネットの特徴量を計る (6/15)

- 遅延、パケットロス、ジッタ
- フロー計測
- 相関と多変量解析
- 主成分分析
- 演習:多変量と相関