インターネット数理科学第5回

～ネットワークそのものを支える数理科学その2～

2006年11月2日

株式会社インターネット総合研究所代表取締役所長
東京大学大学院数理科学研究科客員教授

藤原　洋

目　　次

1．ネットワークそのものを支える数理科学とは？

現在のインターネット数理科学の課題

IRI
Ubilabs Innovations

③(ネットワークの)あちら側

⇒「グラフ理論」「金融工学理論」に基づくデータベース、検索
エンジン最適化、検索連動データベース、ネット金融サービス

①ネットワークそのもの

⇒「グラフ理論」による動的ルーティング、帯域制御、放送型ルーティング
「ディジタル信号処理理論」に基づく変復調技術

②(ネットワークの)こちら側

⇒「ディジタル信号処理理論」に基づくコンテンツ符号化技術

今後の「インターネット数理科学」の講義の進め方

IRI
Ubilabs Innovations

以下の3つの分野にわたって①②③⇒①②③⇒・・順に

③ネットワークのあちら側を支える数理科学

⇒「グラフ理論」「金融工学理論」に基づくデータベース、検索
エンジン最適化、検索連動データベース、ネット金融サービス

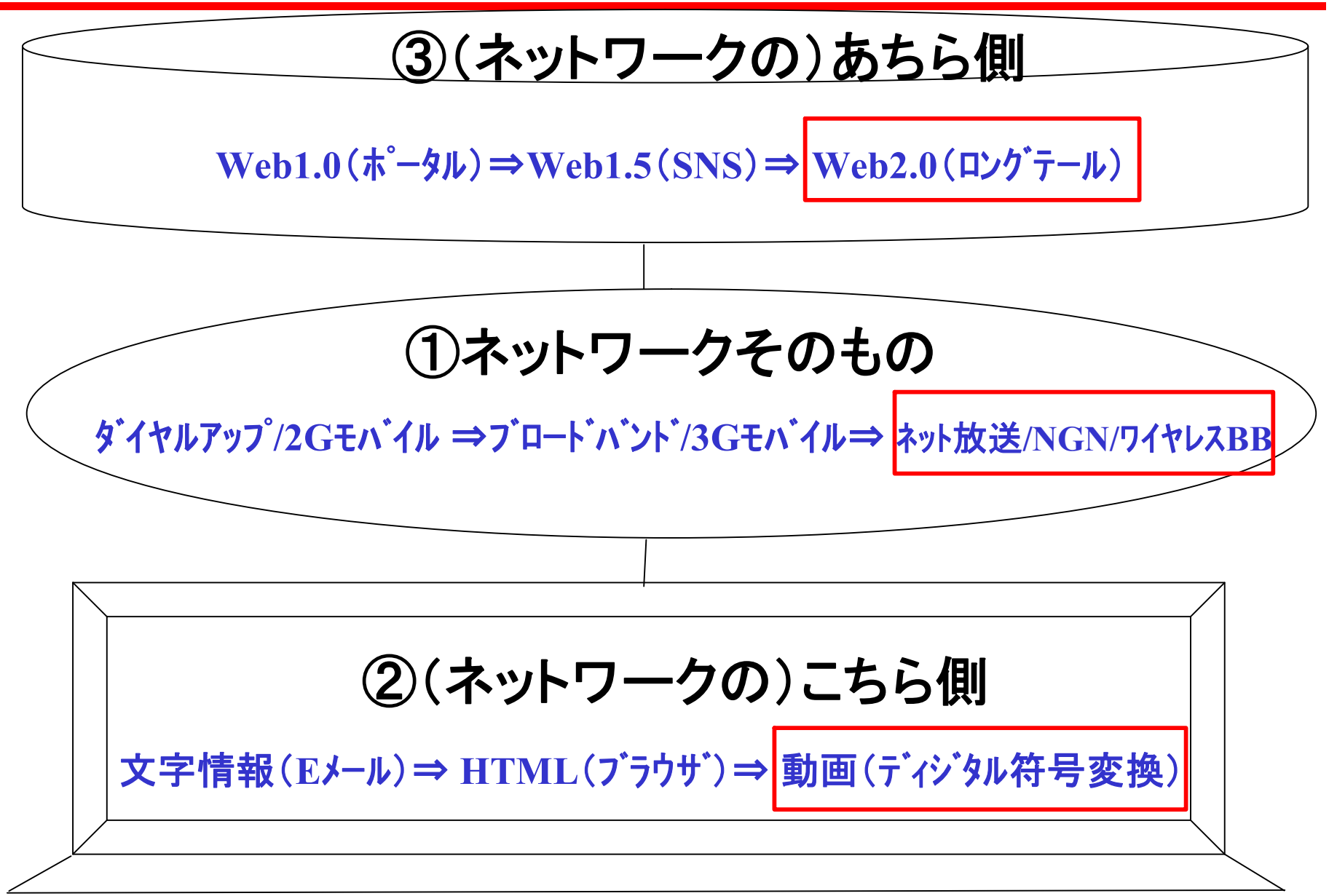
①ネットワークそのものを支える数理科学

⇒「グラフ理論」による動的ルーティング、帯域制御、放送型ルーティング
「ディジタル信号処理理論」に基づく変復調技術

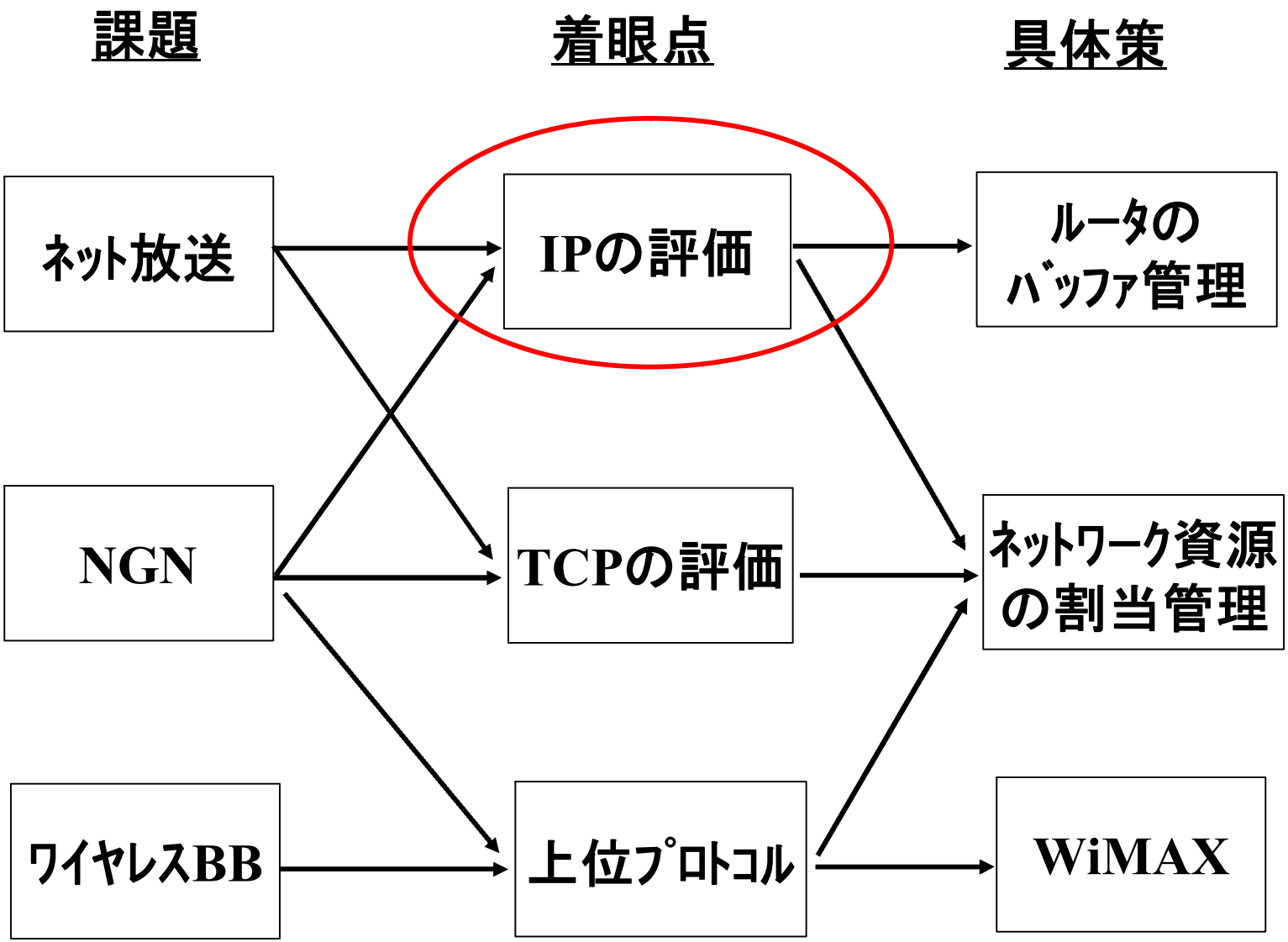
②ネットワークのこちら側を支える数理科学

⇒「ディジタル信号処理理論」に基づくコンテンツ符号化技術

インターネットにおける技術革新と最先端のテーマ



IP放送/NGN/ワイヤレスBB(ブロードバンド)などに潜在する数理科学とは？



インターネットへ「ネットワークそのもの」が変化した結果？

IRI Ubilabs Innovations

本講の第2回にIPネットワークの普及で「通信業界の構造変化」についてふれましたが、今回の話題では、「コンピュータ業界の構造変化」という話題を取り上げます。その昔、「1台の高価なコンピュータを多くの人々が利用するメインフレーム・コンピューティング」の時代でした。この頃から、私も社会人として日本法人に所属した、1つの巨大企業と数社の類似企業が、ハードと基本ソフトを独占し、各国で個別の適用業務用ソフトの開発受託型情報処理サービス企業群が階層構造を形成してきました。その後、国内電機メーカー子会社へ転じ、8年ほど高価なコンピュータを設計する仕事をした後、PC用Basic言語が中心の小さなソフト会社の国内独占代理店会社へ移り、同社が世界一のPCソフト会社へと成長していく過程を目の当たりにしました。即ち、マイクロプロセッサの登場によって、「1人が1台のコンピュータを利用するパーソナル・コンピューティング」へと進化し、オフィスでの「便利さ」を追求する時代となりました。同分野では、市場淘汰の結果、世界に標準CPUを作る1社と標準ソフトを作る1社の事実上の独占となりました。しかし、これはPCだけの話で、社会インフラと家庭生活を変えるまでの変化は起こりませんでした。

そこへ登場したのが、インターネットです。インターネットは、組織毎に個々のコンピュータを接続して組織内ネットワークを形成し、その組織内ネットワークがさらにISPを介して世界につながり、また個人が直接ISPに加入し、ISP同士が相互接続することで世界規模のネットワークを構成しています。インターネットでは、コンピュータ1台毎に固有のIPアドレスが割り当てられていて（実際はアドレス不足でIPv6への移行が進行中）、世界中のコンピュータを相互接続できる仕組みになっています。この仕組みが社会に普及し始めてから約10年ですが、コンピュータは、家庭と携帯電話の中にも入るようになりました。即ち、PCだけではできなかった、「つながる便利さ」と「楽しさ」を、追求する時代となりました。今、IPという技術革新が、通信業界に続いて、コンピュータ業界の構造変化を起こしつつあります。コンピュータ業界は、IPネットワークで相互接続された経営資源に対してGIS（地理情報システム）を活用して経営効率を向上させるアプローチのように、「系列取引型垂直分業構造」から、「テクノロジー要素型水平分業構造」への転換が進むものと思われます。

2. 経路制御（ルーティング）とルータ

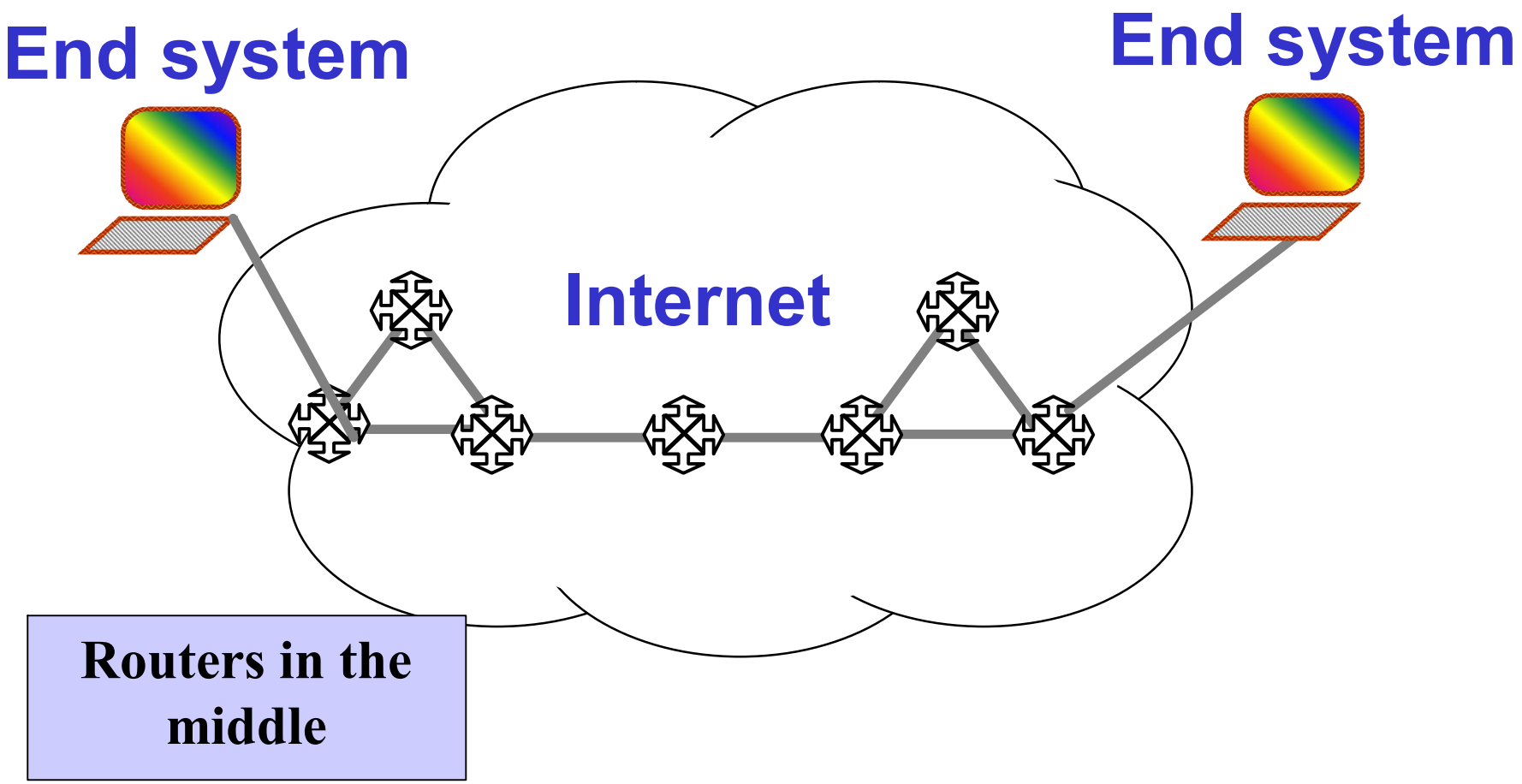
インターネットの父Robert Kahn博士の話

- インターネットは、ロジカルなアーキテクチャ。
  実装形態としての TCP/IP と ネットワーク機器
  - 複数のメディアを利用可能にする“環境”(アーキテクチャ)の提供がいろいろな意味で “鍵” となる。
- 人々は、新しい利用法やアプリケーションを発明開発する“可能性(Possibility)”のために、透明性(Transparency)が必要なのである。
- “Commons” としてのインフラの提供が、インターネットアーキテクチャである。

Robert Kahn：**Vint.Cerf博士と共にTCP/ IPの開発者**

出典：江崎浩（東京大学大学院情報理工学系研究科教授）　Interop2006 Executive Summit

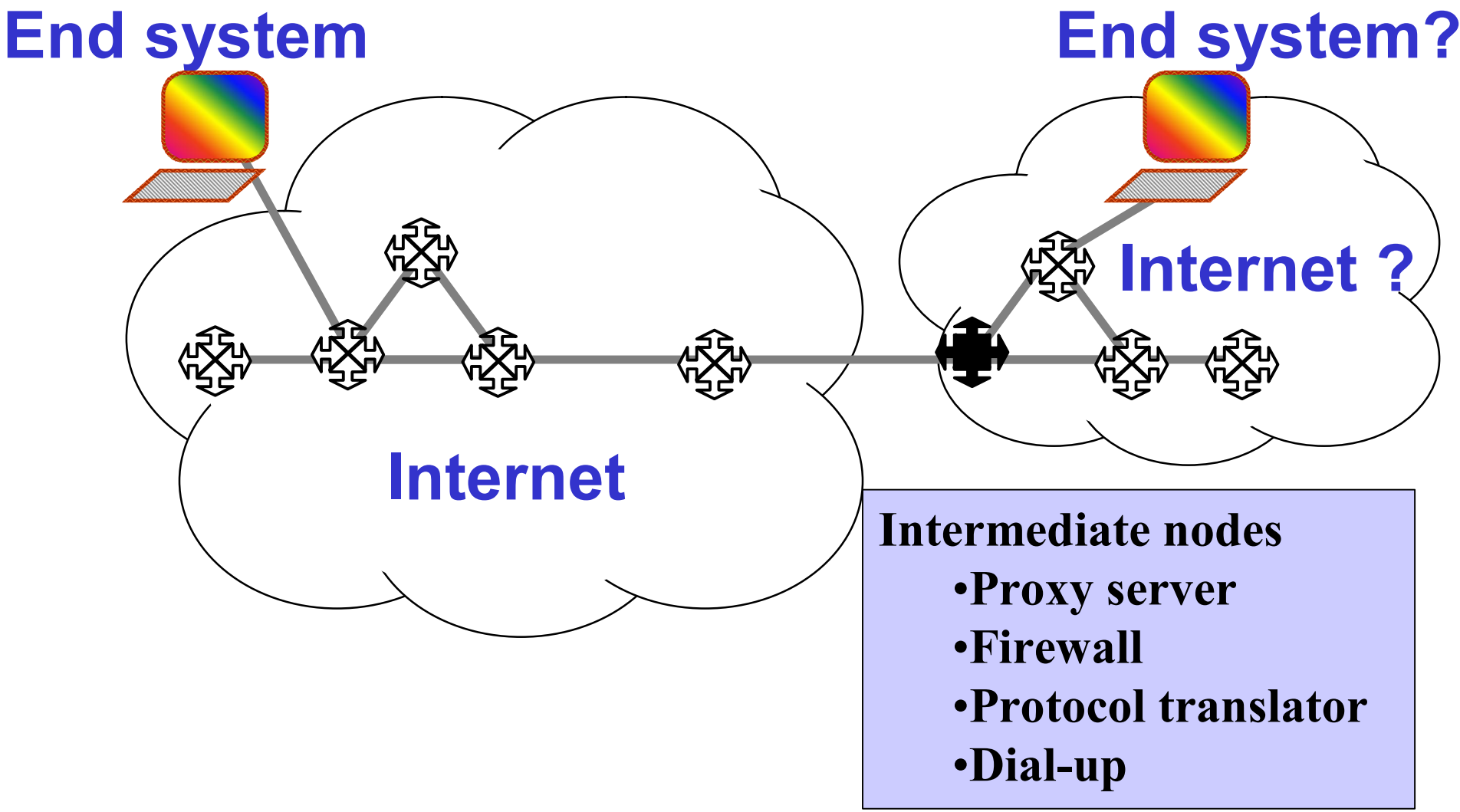
インターネットの基本は“End-to-End Model”である



出典：江崎浩（東京大学大学院情報理工学系研究科教授）　Interop2006 Executive Summit

実際のインターネットではネットワークの分割が起こっている

IRI Ubilabs Innovations

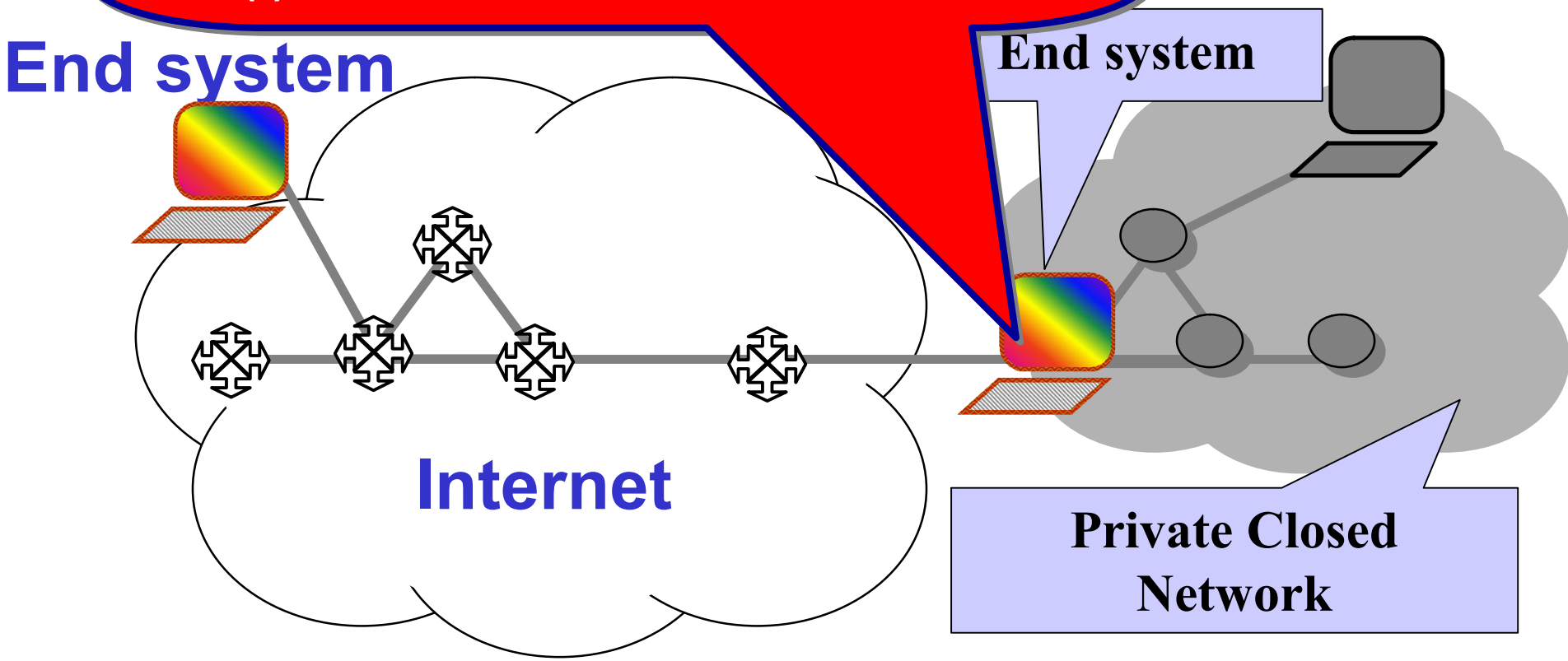


出典：江崎浩（東京大学大学院情報理工学系研究科教授）　Interop2006 Executive Summit

ビジネスの現場ではインターネットと隔離された世界もある

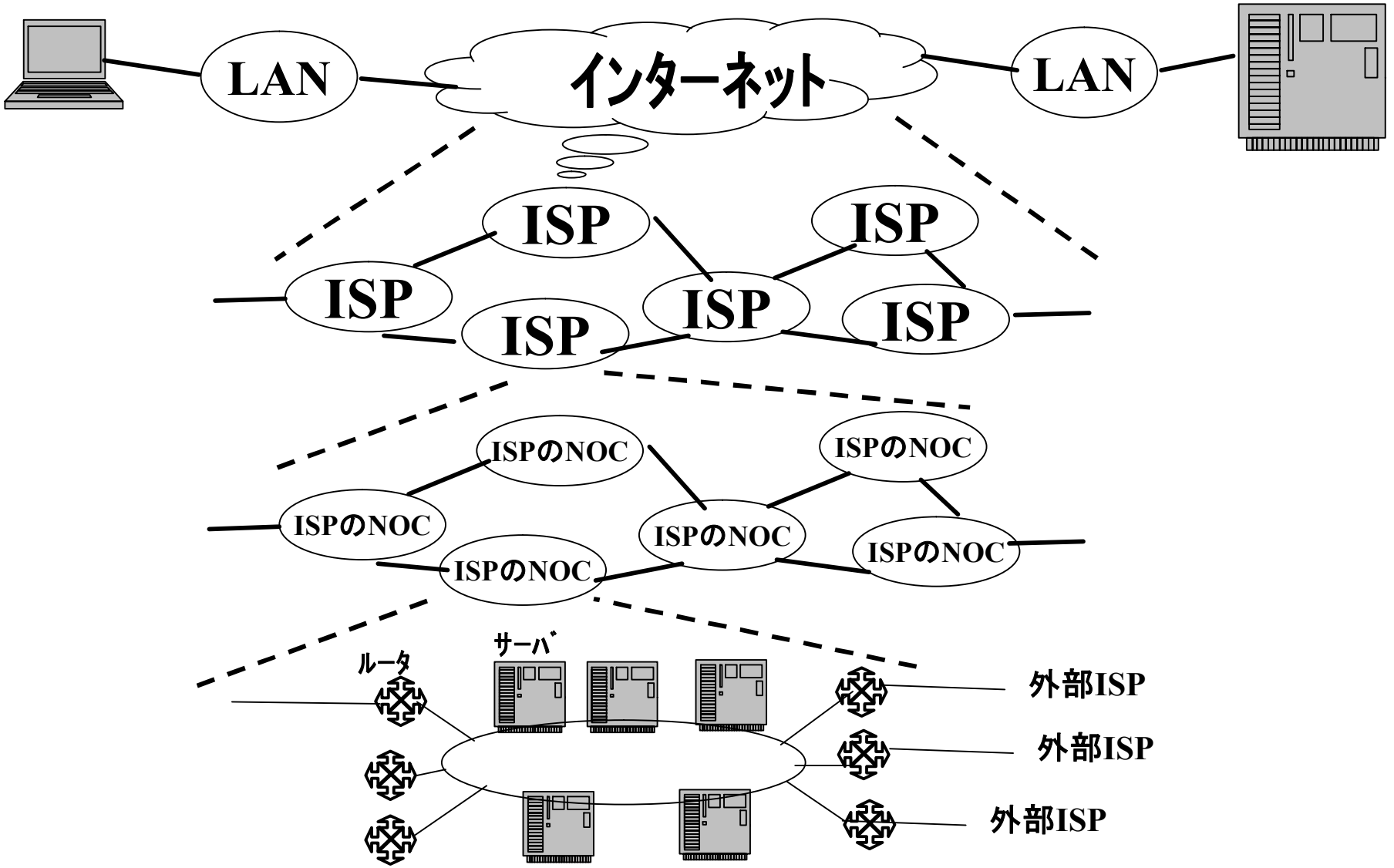
IRI
Ubilabs Innovations

ゲートウェイ装置の存在
- 技術面： 良い面は見当たらない
  要求＝高機能・高性能・高信頼性
- ビジネス面： 排他性・管理性
  (*) つながないのが一番かも?



出典：江崎浩（東京大学大学院情報理工学系研究科教授）　Interop2006 Executive Summit

ルータによるインターネットの基本構成



NOC: Network Operation Center

経路制御（ルーティング）とルータとは？

IRI Ubilabs Innovations

1．目的地までの接続経路＝ルート

2．どの接続経路かを決定する装置＝ルータ

3．接続経路を決定すること＝経路制御

4．経路制御の実際＝ルータXに接続される

Webサーバをアクセスする

→DNSからWebサーバのIPアドレスを見つける

→最寄のルータにIPアドレスを渡すだけで

自動的にアクセスする

3. ルーティング・アルゴリズムの基本となるグラフ理論

グラフの定義

有向グラフ

V をノードの集合、E をエッジの集合とする。エッジに二つのノードの対を対応させる関数 f を

$$f : E \to V \times V$$

とすると、有向グラフ G は

G := (f, V, E)

と定義される。

無向グラフ

P(V) を V のベキ集合とする。エッジにいくつかのノードを対応させる関数 g を

$$g : E \to P(V)$$

とし、任意のエッジ e に対して g(e) = {v_1, v_2} のように値は二つのノードからなっているとする。
この時、無向グラフ G は

G := (g, V, E)

と定義される。g が三つ以上のノードに対応するとき、ハイパーグラフという。
E を最初からある集合の部分集合と考えれば、上の定義から関数を除くこともできる。有向グラフでは、E を V×V の部分集合、無向グラフでは、E を P(V) の部分集合とすればよい。

グラフ理論の用語

グラフの定義によっては、辺に重み（コスト）が付いていることがある。このようなグラフは、重み付きグラフと呼ばれる。

グラフ G の頂点集合は V(G)、枝集合は E(G)で表すことが多い。
辺 e の両端の点を端点といい、端点は e に接合しているという。また、辺と辺がある頂点を共有しているとき、その辺同士は隣接しているという。ある辺の両端点が等しいとき、ループ（自己ループ）という。また、2 頂点間に複数の辺があるとき、多重辺という。ループも多重辺も含まないグラフのことを、単純グラフという。

二つのグラフ G と G' について、G' の頂点集合と辺集合が共に G の頂点集合と辺集合の部分集合になっているとき、G' は G の部分グラフであるという。逆に、G は G' の拡大グラフであるという。特に、頂点集合が等しい部分グラフのことを、全域部分グラフ（生成部分グラフ・因子）という。また、G の頂点集合 V の部分集合 S を取り出して、両端点が S に属する全ての辺を辺集合とする G の部分グラフを、誘導部分グラフという。それから、グラフ G からある辺を取り除き、その辺の両端点を一つの頂点に縮約したとき、縮約グラフ（商グラフ）という。

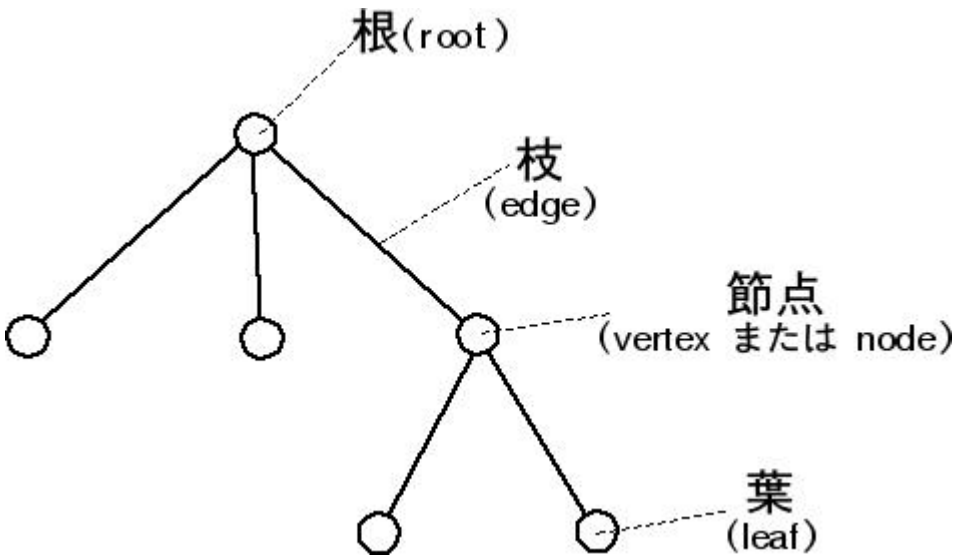
有向グラフにおいて、ある頂点 v に入ってくる枝の数のことを入次数、出て行く枝の数のことを出次数という。そして、頂点 v に接続する枝の数を次数といい、d(v) で表す。すべての v について、d(v) = k が成り立つとき、k-正則という。ある k について k-正則なグラフのことを正則グラフという。グラフ G 中の最小次数の頂点の次数を δ(G)、最大次数の頂点の次数を Δ(G) で表すことが多い。また、次数 0 の頂点のことを孤立点という。

隣接している頂点同士をたどった v1, e1, v2, e2, ..., en-1, vn の系列を歩道（鎖・ウォーク）という。辺の重複を許さない場合、路（小径・トレイル）といい、頂点の重複も許さない場合、道（パス（開いた歩道をパスという場合は単純パス））という。また、始点と終点が同じ路のことを閉路（回路・サイクル）という。

任意の 2 頂点間に枝があるグラフのことを完全グラフ（完備グラフ）という。n 頂点の完全グラフは、Kn で表す。また、完全グラフになる誘導部分グラフのことをクリークという。サイズ n のクリークを含むグラフは「n-クリークである」と言う。辺を持つグラフは必ず 2 頂点の完全グラフを含むので 2-クリークである。また n-クリークであって、直径が n 未満となるグラフを n-クランと言う。

木

木構造(tree structure。単に木(tree)とも)は、グラフ理論の用語で、単連結で閉路を持たない無向グラフの事。閉路を持たない有向グラフをダグという。
あるノードを選んで、それを固定して考えることが多い。このとき、そのノードを根（ルート）という。根を持つ木を単なる木と区別して根付き木という。



木 T には、次の4つの主な性質がある。

(T の辺の個数)=(T の節点の個数) - 1
　　記号で表すと、||T|| = |T| - 1
任意の2点 x,y に対して、x,y を結ぶ道が丁度一つある。
T の2点を結ぶ T に含まれない辺 e に対して、T+e には e を通るただ一つの閉路があり、この閉路上の任意の辺 f に対して T+e-f は木となる。
少なくとも2個の端末点がある。また、端末点とは次数1の点である。
上の定理から、木には必ず端末点があり、その端末点を除去すると位数の一つ小さい木が得られる。逆に言えば、位数 n の木は、位数 n-1 の木に一つの新しい点と、これに接続する一本の新しい辺を加えて得られる。

根つき木

根つき木を家系図と見たてた用語が使われる。点v_1とv_2が辺で結ばれており、しかもv_1の方がv_2よりも根に近いとき、v_1はv_2の親であるといい、v_2はv_1の子であるという。点v_2とv_3が共通の親を持つとき、v_2とv_3は兄弟という。
子孫や先祖も同様に定義する。(厳密に言うと、根つき木上の２点v_1,v_2に対し、v_2と根を結ぶ線上にv_1があるとき、v_1はv_2の先祖であるといい、v_2はv_1の子孫であるという)。
各辺の長さを1とするとき、点と根との距離をその点の高さという。木の上の点の高さの最大値を、その木の高さという。
nを自然数とする。葉ではない各点に対しその点の子の数が常にnであるような木をn分木(nぶんぎ)という。

森

閉路を持たず連結でない可能性がある無向グラフを森という。

グラフ抽象化（The Graph Abstraction）

グラフは、多くの種類の問題を解くのに有効な数学的抽象化である。基本的には、グラフは頂点と辺から構成され、辺は二つの頂点を結ぶ。もっと正確には、グラフ(graph)とは組(V,E)で表され、Vは有限集合で、EはVの2項関係である。Vは 頂点集合(vertex set) と呼ばれ、その要素を 頂点(vertex) と呼ぶ。Eは辺の集合で、辺(edge) とは(u,v)の組でu,vはVの要素である。有向グラフ(directed graph)においては、辺は順序付けられた組で、始点(source)を 終点(target)へと接続する。無向グラフ(undirected graph)においては、辺は順序付けされていない組で、2つの頂点を両方向につなぐ。つまり、無向グラフでは (u,v)と(v,u)は同じ辺の2通りの書き方である。

グラフのこの定義は、いくつかの点であいまいである。辺や頂点が何を表現するかが述べられていない。グラフの例としては、連絡道路やハイパーリンク付きのウェブページなどを挙げることができる。これらの詳細がグラフの定義からは除外されているのは、大きな理由がある。それらの詳細はグラフの抽象化の中では必要な部分ではない。詳細を定義から除外することで再利用可能な理論を構築でき、そのことは多くの異なった種類の問題を解く際に役に立つのである。

定義にもどろう。グラフは頂点と辺の集合である。実際の様子を見せるため、頂点に文字のラベルがついたグラフを考え、辺を単純に文字の組としよう。ここで、有向グラフの例を次のように書くことができる。

V = {v, b, x, z, a, y }
E = { (b,y), (b,y), (y,v), (z,a), (x,x), (b,x), (x,v), (a,z) }
G = (V, E)

このグラフを図示すると 図1のようになる。辺 (x,x) は、輪(self-loop)と呼ばれる。(b,y)と (b,y)は平行辺(parallel edges)であり、これはマルチグラフ(multigraph) でのみ許される(ただし、通常は有向グラフでも無向グラフでも許されない)。

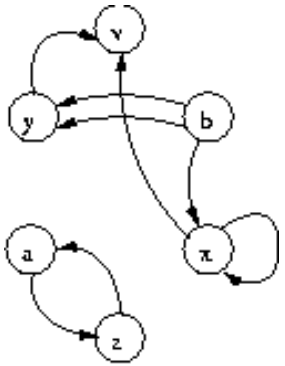
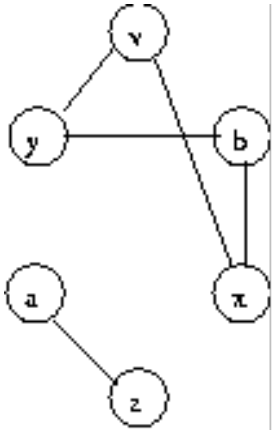
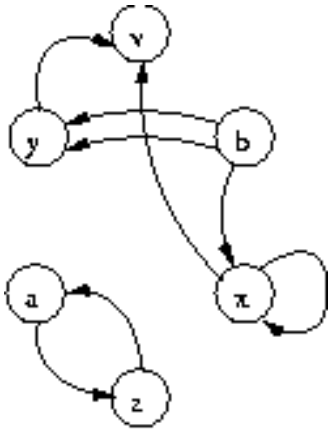


図1. 有向グラフの例

次に似たようなグラフを示すが、今度は無向グラフである。
これは右に図示する。無向グラフでは輪は許されない。
上記のグラフ(から平行辺(b,y)を除いたもの)の
無向版(undirected version)である。それはつまり、
同じ頂点をもち、同じ辺から方向を除いたものを持つことを
意味し、(a,z)と(z,a)いう2つの辺は一つの辺に退化する。
また、逆を考えることもできる。無向グラフの
有向版(directed version)は、すべての辺をそれぞれの
方向を向く2つの辺で置き換えることで得られる。



無向グラフの例
（図2）



有向グラフの例
（図1）

V = {v, b, x, z, a, y }
E = { (b,y), (y,v), (z,a), (b,x), (x,v) }
G = (V, E)

ここでさらにグラフの用語を定義する。辺(u,v)がグラフに含まれるとき、頂点vは頂点uについて隣接している(adjacent)と言う。有向グラフでは、辺(u,v)は頂点uの出辺(out-edge)であり、頂点vの入辺(in-edge)である。無向グラフでは、辺(u,v)は頂点uとvを 接合している(incident on)という。

図1で、頂点yは頂点bに対して隣接している (ただしbはyに対して隣接していない)。辺(b,y)はbの出辺であり、yの入辺である。後図で、yはbに隣接していて、また逆も同様である。辺(y,b)は頂点yとbを接合している。

有向グラフにおいて、ある頂点の出辺の数は 出次数(out-degree)と呼ばれ、入辺の数は 入次数(in-degree)と呼ばれる。無向グラフにおいて、ある頂点に対して接合している辺の数は 次数(degree)と呼ばれる。図1で、頂点bの出次数は3であり、入次数は0である。
図2では単純に頂点bの次数は2である。

グラフの路(path)とは辺の列で、それぞれの辺の終点が次の辺の始点であるものである。頂点uから始まり頂点vで終わる路があれば、頂点vはuから到達可能(reachable)であるという。

路が単純(simple)であるとは、辺の列の中でどの頂点も繰り返し現れないことである。路<(b,x), (x,v)>は単純であるが、路<(a,z), (z,a)>は単純ではない。また、路<(a,z), (z,a)>は最初の頂点と最後の頂点が一致するので、サイクル(cycle)と呼ばれる。サイクルのないグラフは アサイクリック(acyclic)と呼ばれる。

planar graph とは、すべての辺が交差しないように平面上に描けるグラフのことである。そのように描かれたものはplane graphと呼ばれる。plane graphの面(face)とは、辺に囲まれた連結成分のことである。

planar graphの重要な特性は、面、辺、頂点の数が
オイラーの定理：|F| - |E| + |V| = 2によって関係付けられることである。

このことは、planar graphは最大でもO(|V|)個の辺しか持たないことを意味する。

グラフデータ構造（Graph Data Structures）

データ構造を考えるときに最初に考えるべきグラフの属性は、まばらさ(sparsity)である。まばらさとは頂点に対する相対的な辺の数である。EがV2に近いグラフは密(dense)であると呼ばれ、E = alpha VでalphaがVより十分に小さい場合はまばらな(sparse)グラフと呼ばれる。密なグラフについては、通常隣接行列表現(adjacency-matrix representation)が最良の選択であり、一方まばらなグラフについては隣接リスト表現(adjacency-list representation)が最良である。また、まばらなグラフについては辺リスト表現(edge-list representation)も適切な状況下では記憶効率面でよい選択である。

Adjacency Matrix Representation

グラフの隣接行列表現はV x Vの2次元配列である。行列auvの要素は、辺(u,v)がグラフに含まれるかどうかを示すブーリアン値である。図3に図1(から(b,y)を引いたもの)の隣接行列表現を表す。保存に必要な領域はO(V2)である。任意の辺について、アクセス、追加、除去にかかる時間はO(1)である。頂点の追加や除去は、再割り当てとすべてのグラフのコピーが必要になり、手順数はO(V2)になる。adjacency_matrixクラスは、隣接行列表現によってBGLグラフインターフェースを実装する。

	v	b	x	z	a	y
v	0	0	0	0	0	0
b	0	0	1	0	0	1
x	1	0	1	0	0	0
z	0	0	0	0	1	0
a	0	0	0	1	0	0
y	1	0	0	0	0	0

図3: 隣接行列によるグラフの表現

隣接リスト表現（Adjacency List Representation）

グラフの隣接リスト表現では、すべての頂点に対して出辺の列を保存する。まばらなグラフでは、こうすることでメモリ領域を節約でき、必要な領域はO(V + E)だけになる。さらに、すべての頂点の出辺にはより効果的にアクセスできる。辺の挿入のコストはO(1)で、任意の辺へのアクセスはO(alpha)である。ここで、alphaは行列のまばらさ(グラフ中のすべての頂点についての出辺の数の最大値)である。図4は図1のグラフの隣接リスト表現である。adjacency_listは隣接リスト表現の実装である。

辺リスト表現（Edge List Representation）

グラフの辺リスト表現は、単純に辺の列であり、辺は頂点のIDの組で表される。必要なメモリはO(E)だけである。辺挿入のコストはO(1)であり、特定の辺のアクセスするのはO(E)(あまり効果的でない)である。図5は図1のグラフの辺リスト表現である。edge_listアダプタクラスは、辺リスト表現の実装を作るのに使うことができる。

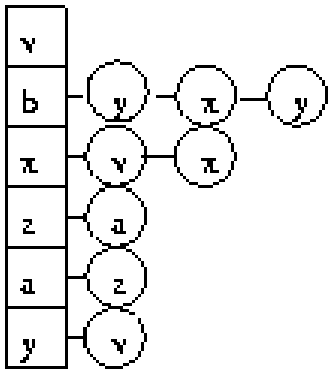


図4: 隣接リストによるグラフ表現

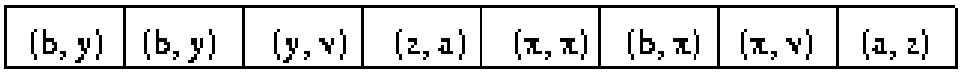


図5: 辺リストによるグラフの表現

グラフ探査アルゴリズム（Graph Search Algorithms）

木辺(tree edge)とは、グラフ探索アルゴリズムをグラフに適用することによって作られた探索木(またはフォレスト)の辺ことである。
辺(u,v)は木辺であるのは、辺(u,v)の探索(ビジタのexplore()メソッドにあたる)をしているときにvが最初に見つかるときである。
後退辺(back edge)とは、探索木上で頂点を先祖につなぐ辺である。したがって、辺(u,v)ではvはuの先祖である。輪は後退辺とみなされる。

先行辺(forward edge)は、木辺ではない辺(u,v)で、探索木上uを子孫vへとつなぐ。交差辺(cross edge)とは、以上の　3つのカテゴリに含まれない辺のことである。

幅優先探査（Breadth-First Search）

幅優先探索(Breadth-First Search, BFS)とは、グラフに対して横断的であり、特定の原点から到達可能な頂点をすべて探索する。また横断する順番については、頂点のすべての近傍を探索してから近傍の近傍の探索へと進む。幅優先探索について考えるには、例えば水溜りに石を落としたときに波が放射状に広がるように拡散すると思えばよい。同じ「波」の中の頂点は原点から同じ距離にある。頂点は最初にアルゴリズムによって遭遇するときに発見される(discovered)と言う。頂点は、その近傍がすべて探索されたときに完了した(finished)と言われる。これらをわかりやすくする例がある。グラフを図6に示し、そのBFSにおける発見と完了の順番をその下に示す。

発見の順番: s r w v t x u y
完了の順番: s r w v t x u y

sから開始して、最初はrとw(sの近傍)にたどり着く。
sの両方の希望に到達してから、rの近傍(頂点v)に到達し、
wの近傍tとxに到達する (rとwの順序は意味を持たない)。
最後にtとxの近傍、uとyに到達する。
今グラフ上のどこにいるか、次にどこの頂点に行くかを
アルゴリズムが把握するために、BFSは頂点に色を塗る。
塗る色を置く場所は、グラフの中でもよいし、
アルゴリズムに引数として渡すこともできる。

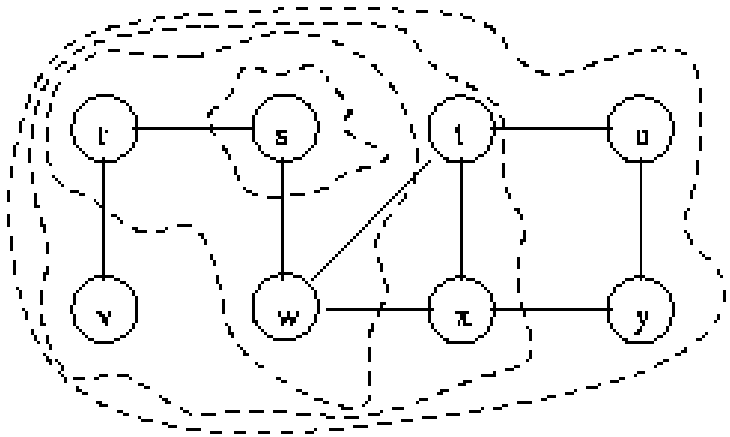


図6: 幅優先探索がグラフに広がる様子

深度優先探査（Depth-First Search）

深度優先探査(Depth-First Search, DFS) はグラフ中の全頂点を探査する。このアルゴリズムでは、常にグラフ中の”深い”部分を次に探査すべき辺として選択していく。これは、到達した頂点が未訪問の隣接頂点を持たなくなるまで次の未訪問な隣接頂点を選択していき、端に到達すれば前の頂点へと戻り、その頂点から任意の未探査な辺へと探査を継続していくことである。深度優先探査は出発する頂点から到達可能な全ての頂点を訪問した後に、残りの未訪問な頂点のうちから1頂点を選択して探査を継続していく。このプロセスは、深度優先の森からともに深度優先の木というセットを形成する。

深度優先探索は、グラフ中の辺を3つのカテゴリーに分類する：木辺、後退辺、先行辺か交差辺(どちらにも明確に分類しません)。与えられたグラフから多くの有効な深度優先の森が典型的に存在し、それゆえ辺を分類するには様々な(かつ等しく有効な)方法である。

深度優先探査の興味深い特性は、各頂点の発見時と完了時の間において、括弧（入れ子）構造を形成するということである。頂点が発見される場合、私たちが開いた括弧を使用すれば、頂点が探査終了される場合には、閉じた括弧が使用され、その結果、括弧により適切に入れ子にされたセットが出来上がる。図7は、探査された順番にラベル付けされた辺による無向グラフに適応された DFS（深度優先探査）である。図の下に、探査を開始した順序と探査を終了した順序を示し、それらから導かれる括弧構造を示す。

DFS（深度優先探査）は、2つが接続されたコンポーネント・アルゴリズム、トポロジカル・ソート、等を含む他のグラフ・アルゴリズムによって使用される核となるアルゴリズムである。これは循環を検知するために利用できる。

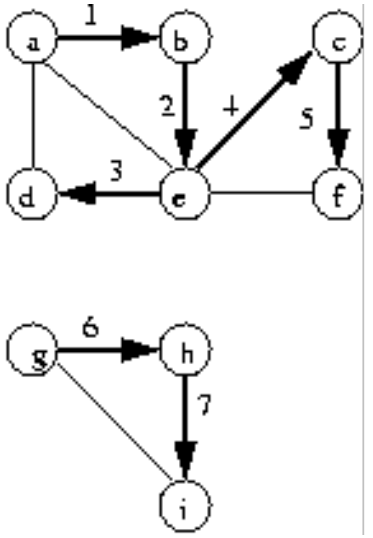


図 7: 無向グラフにおける深度優先探査
発見の順序: a b e d c f g h i
完了の順序: d f c e b a
括弧構造: (a (b (e (d d) (c (f f) c) e) b) a) (g (h (i i) h) g)

最小全域木問題(Minimum Spanning Tree Problem)

IRI Ubilabs Innovations

最小全域木問題は以下のように定義される:グラフ E 中の全頂点を接続する循環のサブセット T を接続の全コストが最小となるように選択することである。全コストは下記の式により与えられる。

w(T) = T における辺 (u,v) におけるコスト w(u,v) の合計、
w(u,v) は辺 (u,v) のコスト

T は全域木(spanning tree) と呼ばれる.

最短経路アルゴリズム（Shortest-Paths Algorithms）

Ubilabs Innovations

グラフ理論における基本的な問題のひとつは、グラフ中の2頂点間を結ぶ最短経路を見つけることである。形式的に経路はグラフ G = (V, E) 中の頂点のシーケンス <v0,v1,...,vk> で表される（辺 (vi,vi+1) for i=0,1,...,k-1 は 辺の集合 E ）。
シーケンスにおいて各頂点は次の頂点へ接続される。最短経路問題において、各辺は重みを数値として与えられている。それゆえ、経路の重み(weight of a path) について記す w(p) = i=1..k of w(vi-1,vi) の合計

頂点 u から v に至る最短経路の重み (shortest path weight) は
delta (u,v) = min { w(p) : u --> v } もし 頂点 u から v に至る経路が存在すれば
delta (u,v) = 無限(infinity) そうでなければ（ u から v に至る経路が無ければ）

最短経路は重みの合計が最小となる経路といえる。
最短経路問題には、いくつかの変形された問題がある。ここでは単一ペアの問題を定義した、しかし、さらに単一出所問題(グラフ中の1つの頂点から各頂点ごとまでの最短のパス)があり、等価な単一目的地問題、全ペア問題、等である。単一出所の問題を解決するアルゴリズムより漸近的に速い、単一ペアの問題を解決するアルゴリズムは存在しない。
最短経路木(shortest-paths tree) は、グラフ G=(V,E) 中のある頂点を原点とした有向サブグラフである。V‘ を V のサブセット、E’ を E のサブセットとし、V‘ はG’ から到達可能な頂点のセット、G‘ は原点から連なる経路木を成すものとすれば、V’ 中の全ての頂点 v は G‘ 中の頂点v から唯一の経路を持つ。再帰的に、単一頂点アルゴリズムによる結果は最短経路木である。

ネットワークの流れは送信(source)頂点 s から受信(sink)頂点t へと向かう有向グラフ G=(V,E) である。各辺は数値による、容量（capacity）関数 c 、及び、流れ（flow ）関数 f を持つ。流れ関数は次の3条件を満たしていなければならない：

f(u,v) <= c(u,v) for all (u,v) in V x V (容量制限)
f(u,v) = - f(v,u) for all (u,v) in V x V (流れ対称性)
sumv in V f(u,v) = 0 for all u in V - {s,t} (流れ保存則)

ネットワークにおける流れ（flow)は、受信頂点 t に流れ込むネットの流れである（それは、送信頂点 s から流れ出るネットの流れに等価である）。

|f| = sumu in V f(u,t) = sumv in V f(s,v)

辺における余剰容量（residual capacity）を r(u,v) = c(u,v) – f(u,v) とする。r(u,v) > 0 を満たす辺は余剰辺 Ef であり、それは余剰グラフ Gf = (V, Ef) を成す。

r(u,v) = 0 を満たす辺は飽和（saturated ）している。

最大流問題（maximum flow problem ）は、最大に可能な流量値 |f| を決定することであり、そのときのグラフ中における各辺に対する流量値を決定することである。

ネットワークの流れを 図 8に示す。A は送信頂点で、H は受信頂点。

最大流ネットワーク問題を解決するには長い歴史があり、最初のアルゴリズムは Ford and Fulkersonによる。現在に至る最良のアルゴリズムである push-relabel アルゴリズムは Goldbergによるもので、これは Karzanov による preflow introduced という概念を元に成り立っている。

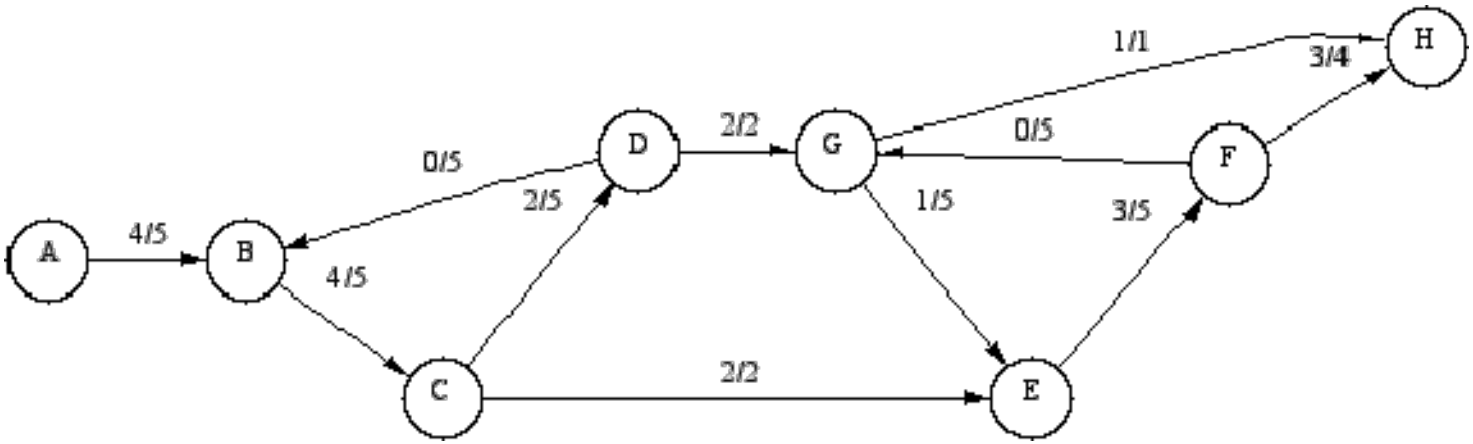


図 8: 最大流ネットワーク
各辺は（流れ/容量）のラベルで示している

4. IP上のルーティング・アルゴリズムとは？

ルーティング（インターネットにおける経路決定方法）

ルーティングあるいは経路制御（けいろせいぎょ）は、情報を送信するため、コンピュータ・ネットワーク上での経路を見つけだす手法である。

経路が判明すれば、その経路に沿って、発信元から最終的な宛先へ、結節点またはノード（グラフ理論において、リンクによって結ばれる頂点。ここではルータが相当する）を経由しながら転送を繰り返して情報が送られる。　情報はパケット（小包の意。データをある程度の量ずつに小分けして送信する、その一単位）として送られ、各パケットには論理的なアドレスが付加してある。

各ルータはルーティングテーブルという表を保持しており、この表に従ってパケットの転送先を決定する。

ルーティングテーブルにはネットワーク上の様々な宛先に対するアルゴリズム上最も良い経路が記録されている。

ルーティングにおいて最も重要な目的は、このルーティングテーブルを構築し維持管理することである。ルーティングでは、似たアドレスはネットワークの　近傍に存在するようにアドレスが構造化されていることを想定しており、ネットワーク的に近傍にある複数のアドレスをルーティングテーブル内の一つの項目にまとめることができる。

ルーティングを実行する場合は、*『トポロジー（ネットワーク構成）』*について理解しておくことが必要である。

小規模なネットワークでは、ルーティングテーブルは、手動構成可能

ネットワークが大規模になると複雑なトポロジーを持ち、しかも間断なく変更される。 そのためルーティングテーブルの構築は大きな問題となりがちである。
それでもなお、公衆交換電話網は、ルーティングテーブルを予め計算し用意した上で、最も短い経路が使えなくなった場合に備えて予備回線も用意する方法をとっている。

＊ネットワーク構成(Network topology)とは、ネットワークの構成方法などのこと。網構成、ネットワーク・トポロジー、あるいは単にトポロジーとも言う。
外見上のネットワーク構造「物理構造」（物理トポロジ）と、それぞれのネットワークの規格によって決まっている「論理構造」（論理トポロジ）がある。

＊公衆交換電話網（こうしゅうこうかんでんわもう、PSTN:Public Switched Telephone Network）は、固定電話回線の電話網である。

トポロジーの種別

星型（スター形）【star topology】
　交換設備(ハブ)から放射状に接続されるもので、主にLANの端末機器の接続点、公衆通信網の加入者収容部分などで用いられる。スポーク車輪の車軸のように見えるため、ネットワークの中心部はハブ(Hub)と呼ばれる。バス形に比べ、ハブ接続されているそれぞれの線については独立して扱われるため障害耐性が高いが、ハブが故障した場合、全ての通信が途絶するため、ネットワークの障害耐性はハブに依存する。主にXBASE-Tなどイーサネットの物理構造であり、バス形や環形（リング型）でも、ひとつの「ハブ」に収めることにより、この物理構造をとる物もある。

バス形【bus topology】
　送信した信号が全ての端末で受信されるものである。無線通信、10BASE-5、10BASE-2等の例がある。有線のバスでは、断線が起こった場合、それより遠位のノードとの通信が途絶するため障害耐性は低い。10BASE-Tなどのイーサーネットの論理構造はバス型である。バス制御の代表例としては、CSMA/CD(Carrier Sence Multiple Access with Collision Detection)がある。

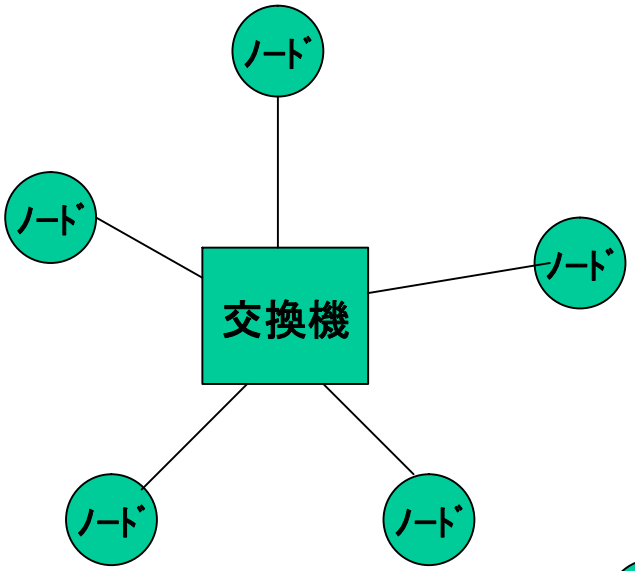
環形（リング型）【ring topology】
　環状に交換設備を接続したもので、一つの区間の障害時には逆向きの接続で伝送できるものもある。ただし、二箇所で断絶した場合、それ以上に通信が不可能になるため、ネットワーク構成としてこれ単体でそれほど障害耐性が高いわけではない。信号の伝送を1方向に限定し、多重化したものもある。代表例としては、Token Ringがある。

複合型（ハイブリッド型）【hybrid topology】
上記の形態のネットワークを組み合わせたもの。

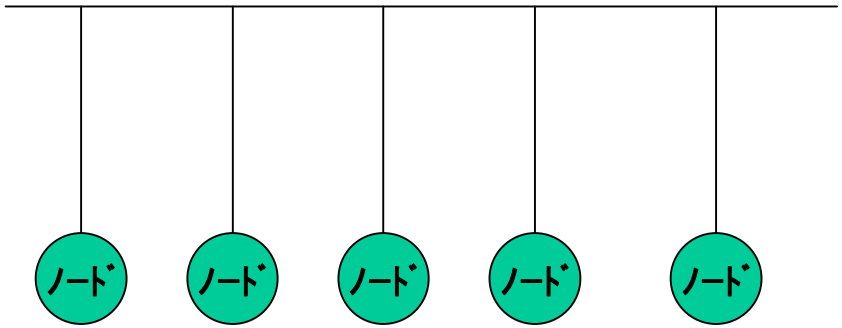
階層形【hierarchial topology】
スター形・リング形を階層に分けて接続したものであり、大規模な公衆通信網に用いられる。

論理トポロジーは、バス形が多い！

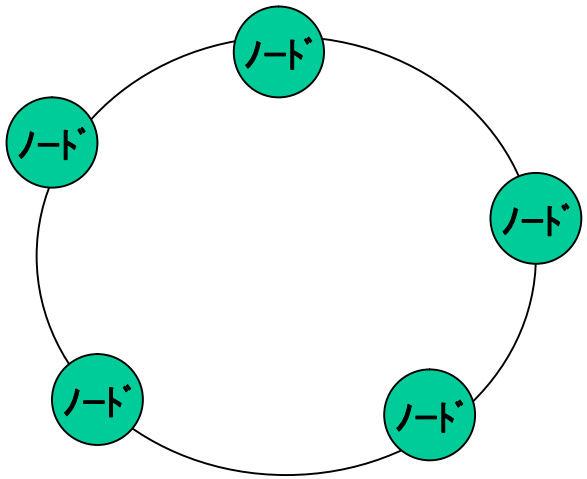
アクセス制御には数理科学的手法が内在（主として70年代後半から80年代半ばの論争）



【star topology】

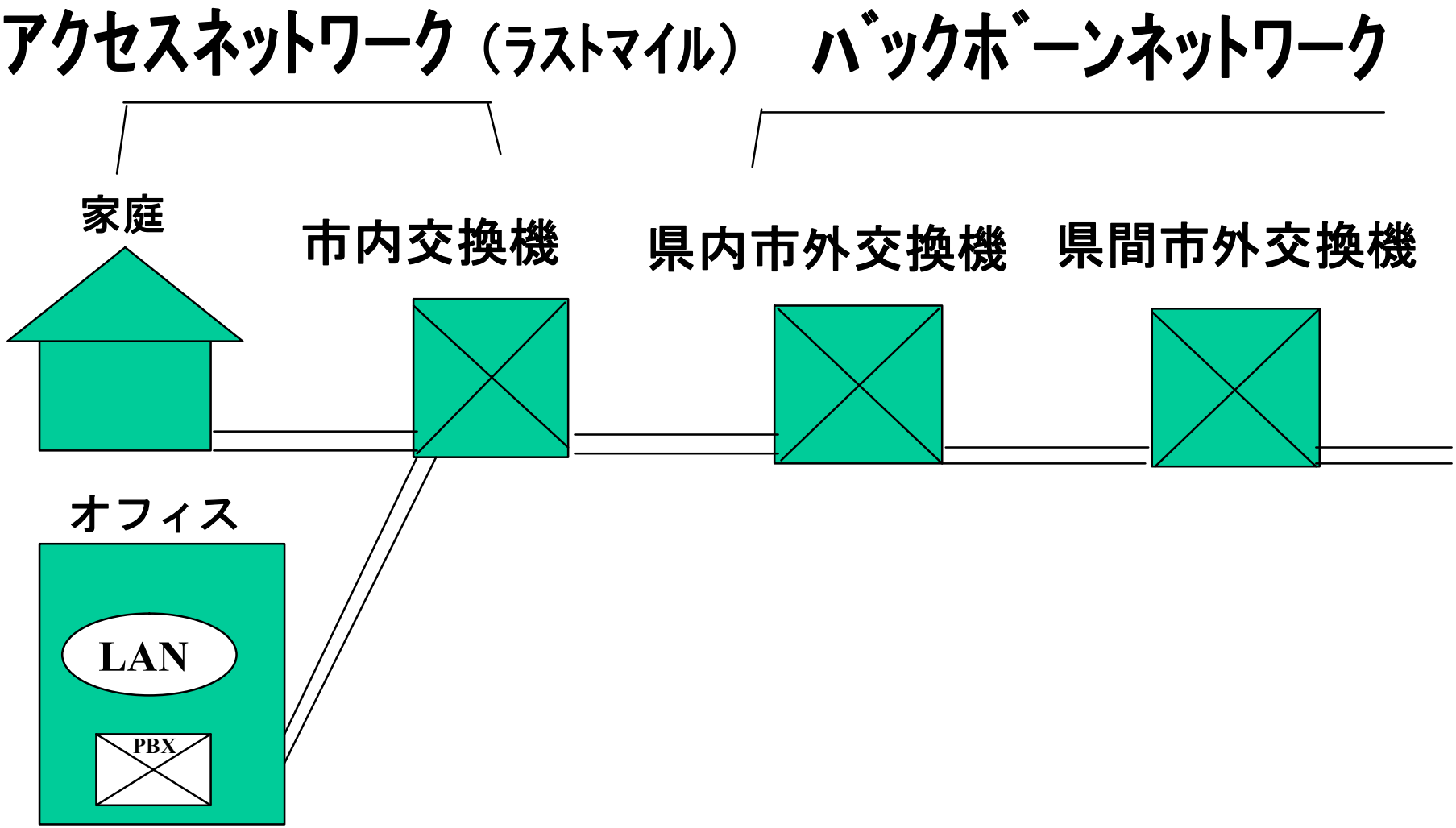


【bus topology】

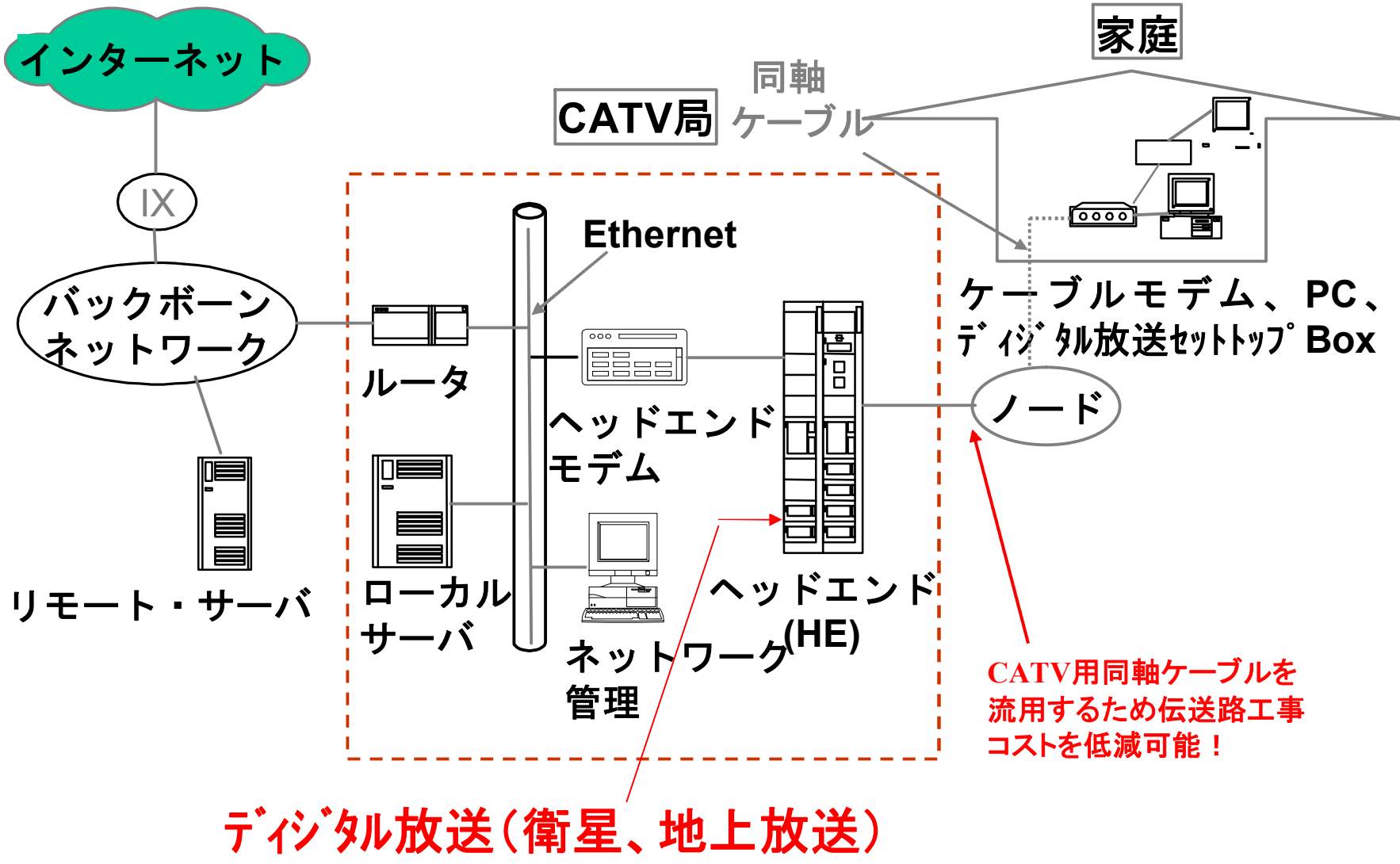


【ring topology】

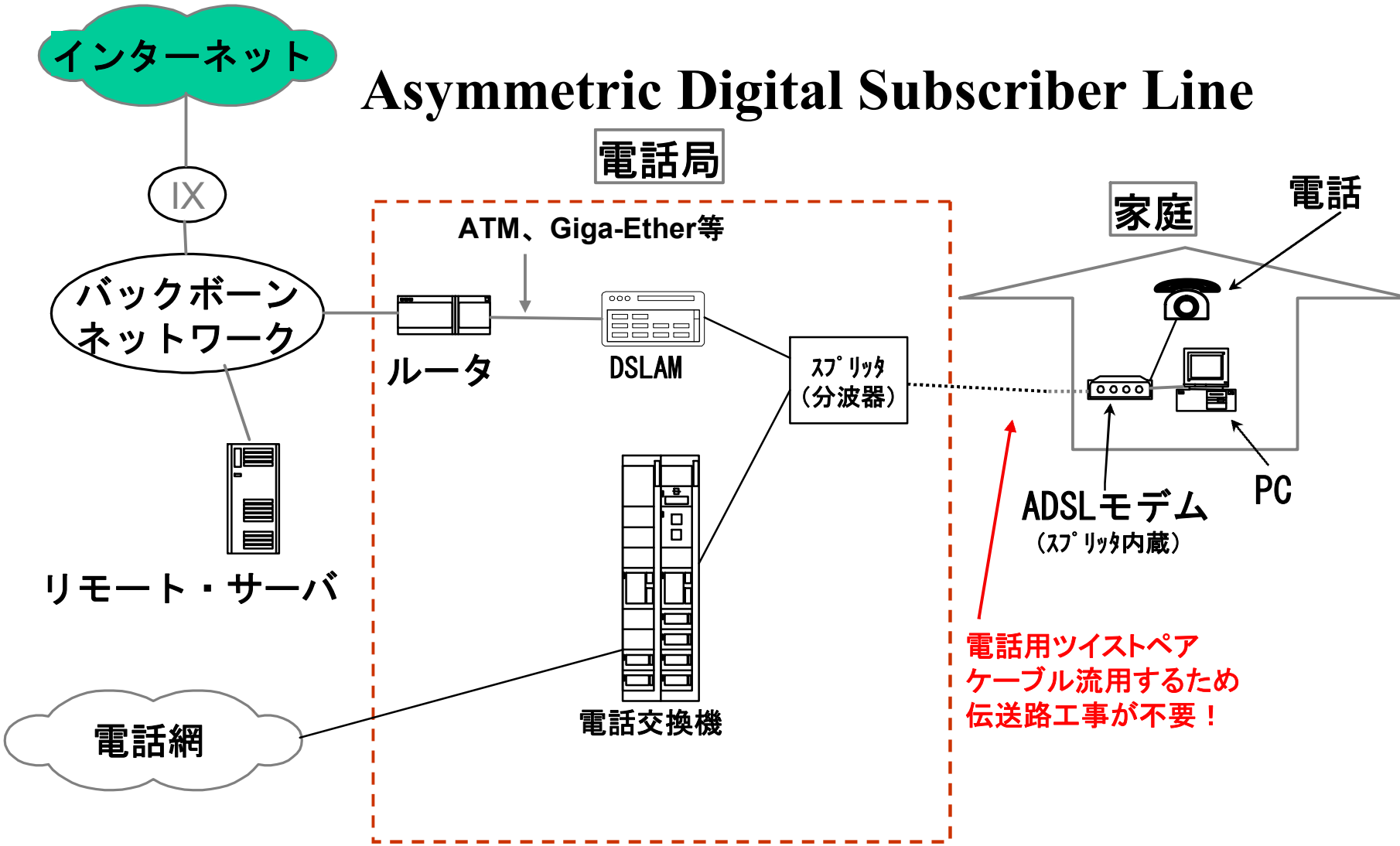
実際のアクセスネットワークとバックボーンネットワーク



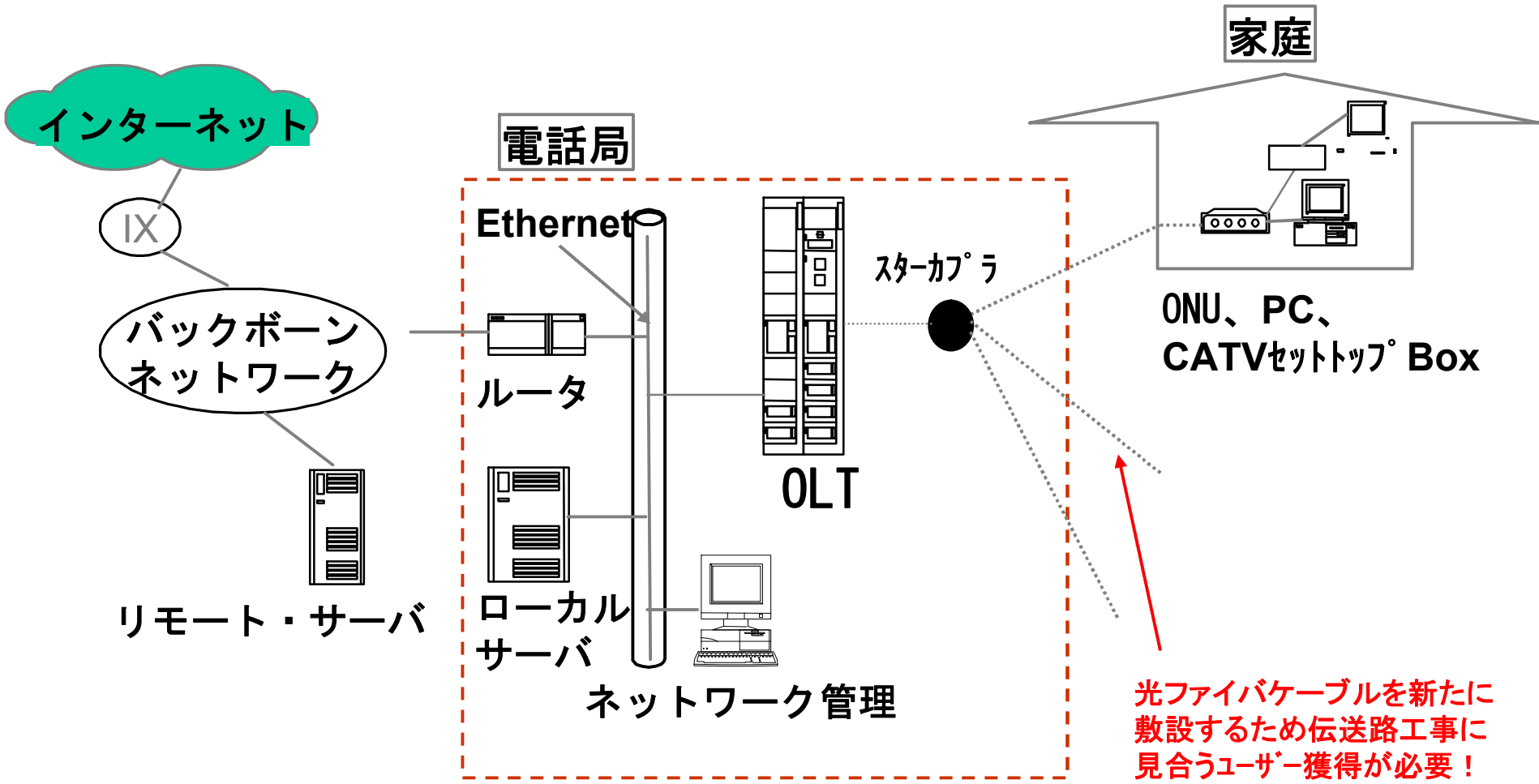
実際のﾃﾞｨｼﾞﾀﾙCATVネットワーク



実際のADSLのネットワーク構成



FTTHによるネットワーク構成



OLT:Optical Line Terminal、
ONU:Optical Network Unit

ルーティングの実際

IRI Ubilabs Innovations

1. ルータがルーティング・プロトコル・プログラムを実行し、ネットワーク内の他のルータとルート情報交換を実施
2. ルータが、同上情報を使用し、特定のルーティング・プロトコルに関連したルーティング・テーブルを作成
3. ルータが、複数のルーティング・プロトコルが動作している場合、各種ルーティング・テーブルを参照し、宛先への最適パスを選択
4. ルータが、該当する宛先に、ネクスト・ホップ・デバイスに付属するデータリンク・レイヤ・アドレスと、ローカル出力インタフェースとを関連づけ。通常は、ネクストホップデバイスは、他のルータ。
5. ルータのフォワーディング・テーブルにネクストホップ・デバイスの転送情報（データリンク・レイヤ・アドレスとローカル出力インタフェース）を書き込む。
6. ルータが、パケット受信し、ヘッダーから宛先アドレスを検出。
7. ルータが、フォワーディング・テーブルにネクストホップ・デバイスの転送情報（データリンク・レイヤ・アドレスとローカル出力インタフェース）を取得。
8. ルータが、付加機能*を実行し、パケットを適切なデバイスへ転送。
   *IP-TTL（パケットの有効期間を表す値）の減少、IP-TOS（Type of Service）操作等。
9. 以上の手順を宛先ホストに届くまで繰り返し実行。

動的ルーティング

IRI
Ubilabs Innovations

自動的にルーティングテーブルを構築する方法によって、経路制御を実行しようというものである。ルーティングテーブルの構築は、ルーティングプロトコルによって伝えられる情報に基づいておこなわれる。これによって、通信の断絶が起きないように、自律的なルーティング機能が実行されるが、現実には、意外と高度な技術が要求され、*現時点ではルーティングが完全に自動的に行われていない。*

インターネットのようなパケット交換方式ではデータはパケットに分解される。パケットには個々に宛先アドレスが付加され、独立にルーティングされる。
対照的な方法である公衆交換電話網のような回線交換方式でも、電話の発呼のように、回線への経路を探すためにルーティングが行われる。しかし一旦接続が成立すれば、完全な宛先をラベルとして貼らなくても連続的に大量のデータを送ることができる。

ルーティングを行う装置としては、スイッチングハブ、レイヤ3スイッチ、ルーターなどがあるが、一般的にルーティングと言う場合にはレイヤ3以上のアドレス（ここではIPアドレス）に関する経路制御を指す。

動的ルーティングの基本的考え方

IRI Ubilabs Innovations

指示された経路が有効でなくなっている場合、現存するノードを使った別の経路を決めなければならない。
これは通常ルーティングプロトコルと経路決定アルゴリズムによってなされる。経路決定アルゴリズムには二種類ある。

①距離ベクトルアルゴリズム (distance vector algorithm, DVA)

RIP (Routing information Protocol)

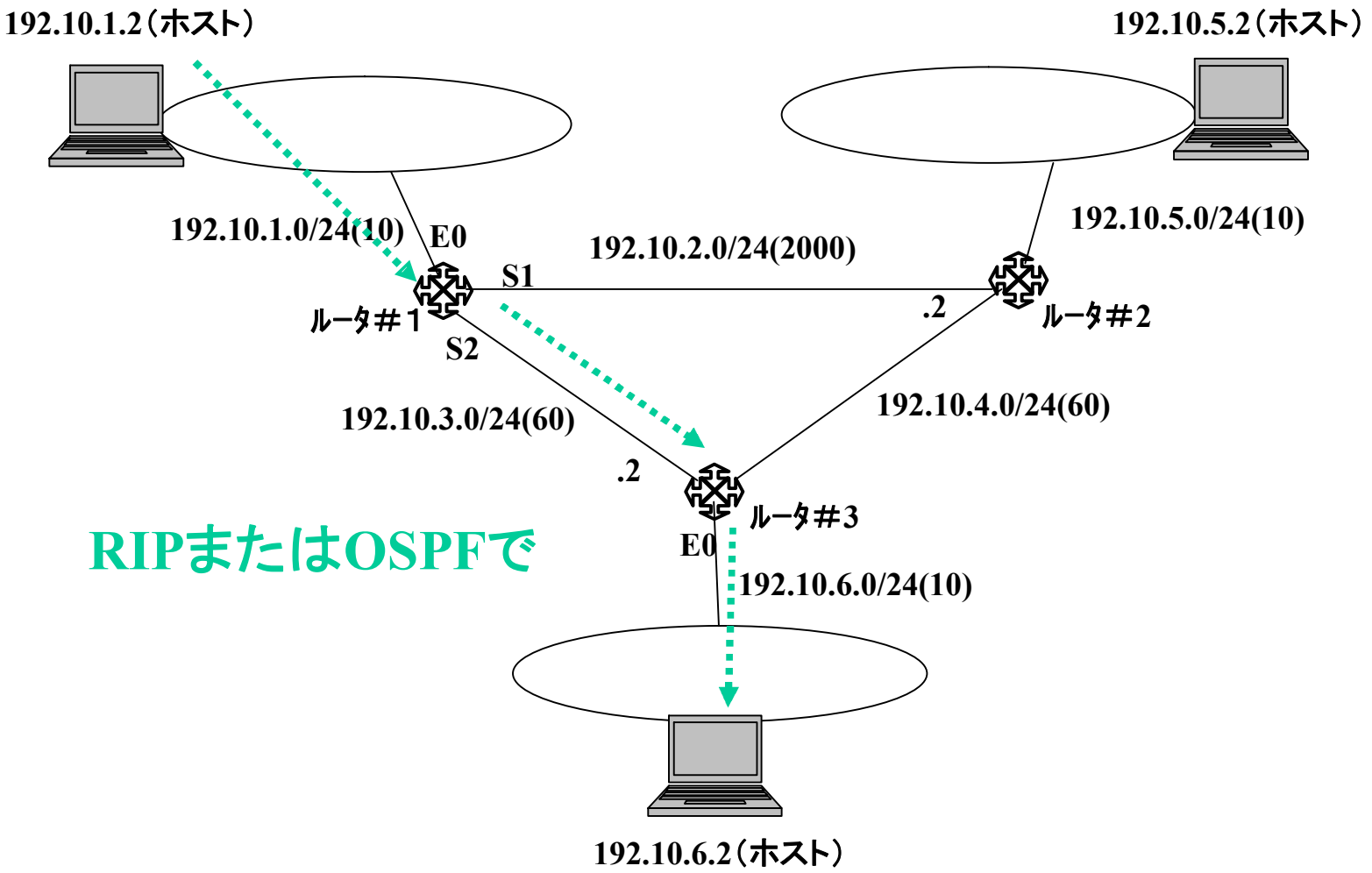
②リンク状態アルゴリズム (link state algorithm, LSA)

OSPF (Open Shortest Path Fast)

IS-IS (Intermediate System-to- Intermediate System)

この内どちらか一方が用いられる。
インターネット上の経路決定問題は、これら2つに尽きる。
以下用いられる「コスト」ないし「距離」は経由するルータの数（「ホップ数」）や回線速度を数値化したもので、「メトリック metric」と呼ばれる。メトリックの決定法はプロトコルによって異なる。

ルーティングの基本動作



①距離状態ｱﾙｺﾞﾘｽﾞﾑ：DVA (distance vector algorithm)

DVAは Bellman-Ford アルゴリズムを用いている。この方法では、各ノード間に「コスト」と呼ばれる数値が割り振られる。二点間を結ぶ経路のコストは、その間に経由するノード間のコストの総和であり、その情報はノードから得られる。

アルゴリズムは極めて単純である。最初の段階では、各ノードは直近のノードがどれかという情報と、それらの間とのコストだけを知っている（このような、「行き先リスト」とそれぞれの総コスト、やりとりするべき「次の相手(next hop)」を集めたものがルーティングテーブルないし、ディスタンステーブルである）。

代表例としてRIP (Routing information Protocol)がある！

定期的にノード間でやりとりがなされ、互いにルーティングテーブルのデータを交換する。もし隣から渡されたデータに、自分のルーティングテーブルより優れたもの（同じ行き先に到達するのに、コストが少ない）があれば、それを用いてテーブルを更新する。自分のテーブルにない相手への情報が入っていた場合も同様である。時間をかけると、全てのノードがあらゆる宛先についての最良の「次の相手」と最良の「コスト」を見つけだす。

あるノードが脱落した場合は、そこを「次の相手」としていたノード全てにおいて、ルーティングテーブルの破棄と再構築が行われる。この情報は隣のノードに次々 伝えられて行き、最終的には到達可能な全てのノードについて最良の経路が発見されることになる。

Bellman-Ford アルゴリズム は、正と負の両方の辺の重みを持つグラフの単一始点の最短経路問題を解く。最短経路問題の定義のために、Shortest-Paths Algorithms がある。

もし正の辺の重みを持つ最短経路問題を解く必要があるだけなら、Dijkstra のアルゴリズムがより効率的な代替手段を提供する。もし全ての辺の重みが 1 に等しいなら幅優先探索がより一層効率的な代替手段を提供する。

bellman_ford_shortest_paths() 関数を呼ぶ前に、ならない。

Bellman-Ford アルゴリズムはグラフ中の全ての辺を通してループし、各辺にリラックス操作 (減らす操作) を適用する事によって進められる。下記の擬似コード中で、v は u の隣接頂点で、w は辺にそれらの重みをマップし、d は今の所見られる各辺への最短経路の長さを記録する距離マップである。p は各辺の親を記録する先行点マップで、それは結局最短経路木中で親となるであろう。

ユーザは始点に 0 の距離を割り当て、他の全ての頂点に無限大の距離を割り当てなければ

```
RELAX(u, v, w, d, p)
 if (w(u,v) + d[u] < d[v])
d[v] := w(u,v) + d[u]   : 辺をリラックスする (減らす) (u,v)
 p[v] := u
 else ... 辺 (u,v) は リラックスされていない (減らされていない)
```

アルゴリズムはグラフ中に負の閉路が存在しないならば、各辺への距離が可能な限り最小に減らされた事が保証された後にこのループを |V| 回繰り返す。もし負の閉路が存在するならば、グラフ中に適当に最小化されない辺が存在する事になるだろう。つまり、w(u,v) + d[u] < d[v] であるような辺 (u,v) が存在することになるだろう。アルゴリズムは全ての辺が最小化されたかどうか最後に 一回調べるためにグラフ中の辺をループし、もしよければ true を返し、そうでなければ false を返す。

```
BELLMAN-FORD(G)
  for each vertex u in V     :  頂点 u の初期化
       d[u] := infinity
      p[u] := u
  end for
  for i := 1 to |V|-1
    for each edge (u,v) in E    :  辺 (u,v) の調査
        RELAX(u, v, w, d, p)
    end for
  end for for each edge (u,v) in E
     if (w(u,v) + d[u] < d[v])
         return (false, , )          :  辺 (u,v) は最小化されていない
    else
        ・・・                       :  辺 (u,v) は最小化されている
   end for
    return (true, p, d)
```

bellman_ford_shortest_paths() 関数から出力を得るための主な二つの選択が存在する。ユーザが distance_map() パラメータを通して距離プロパティ・マップを提供するならばグラフ中の始点から他の全ての頂点への最短距離は距離マップに記録されるだろう (もし関数が true を返すなら)。二番目の選択は最短経路木を predecessor_map() に記録する事である。V 中の各頂点 u にとって、最短経路木中では p[u] が u の先行点になるだろう (但し p[u] = u でここに u が始点であるかまたは始点からは到達不能な頂点である場合を除く)。これらの二つの選択に加え、ユーザはアルゴリズムのイベント・ポイントのどれかの間アクションを取れる独自のビジタをそこに提供する事ができる。

Parameters

IN: EdgeListGraph& g

型が Edge List Graph のモデルの有向グラフまたは無向グラフでなければならない。

IN: Size N

グラフ中の頂点の数。型 Size は汎整数型でなければならない。

Named Parameters

IN: weight_map(WeightMap w)

グラフ中の各辺の重み (そして“長さ”もしくは“コスト”として知られる)。WeightMap の型は Readable Property Map のモデルでなければならない。このプロパティ・マップのキー型はグラフの辺記述子でなければならない。重みマップの値型は距離マップの値型を伴った Addable でなければならない。

デフォルト: get(edge_weight, g)

OUT: predecessor_map(PredecessorMap p_map)
先行点マップ (predecessor map) は最小全域木中に辺を記録する。アルゴリズムの完了時に、V 中の全ての u のための辺 (p[u],u) は最小全域木中にある。もし p[u] = u なら u は始点かまたは始点から到達不能な頂点である。PredecessorMap の型はキーと頂点の型がグラフの頂点記述子型と同じ Read/Write Property Map でなければならない。
デフォルト: dummy_property_map

IN/OUT: distance_map(DistanceMap d)
グラフ g 中の始点から各頂点への最短経路の重みは、このプロパティ・マップ中に記録される。DistanceMap の型は Read/Write Property Map のモデルでなければならない。プロパティ・マップのキー型はグラフの頂点記述子型でなければならず、距離マップの値型は Less Than Comparableでなければならない。
デフォルト: get(vertex_distance, g)

IN: visitor(BellmanFordVisitor v)
ビジタ・オブジェクトで、その型は Bellman-Ford Visitor のモデルでなければならない。ビジタ・オブジェクトは値渡しされる。
デフォルト: bellman_visitor<null_visitor>

IN: distance_combine(BinaryFunction combine)
この関数オブジェクトはリラックス (減少) 段階中で、加算の役割を置き換える。第一引数の型は距離マップの値型に一致していなければならず、第二引数の型は重みマップの値型に一致していなければならない。結果型は距離マップの値型と同じでなければならない。
デフォルト:std::plus<D> ここに D=typename property_traits<DistanceMap>::value_type.

IN: distance_compare(BinaryPredicate compare)
　この関数オブジェクトはリラックス (減少) 段階中で、距離を比較する less-than (<) 演算子の役割を置き換える。引数の型は距離マップの値型に一致していなければならない。
デフォルト: std::less<D> ここに D=typename property_traits<DistanceMap>::value_type.

Complexity
時間計算量は O(V E) である。
Visitor Event Points
vis.examine_edge(e, g) はグラフ中の各辺において |V| 回呼び出される。
vis.edge_relaxed(e, g) は終点のための距離ラベルが減じられた時に呼び出される。頂点 v のための最近のリラックス (減少) にあずかった辺 (u,v) は最短経路木の中にある辺である。
vis.edge_not_relaxed(e, g) はもし終点のための距離ラベルが減じられなかった時に呼び出される。
vis.edge_minimized(e, g) はアルゴリズムの第二段階の間、各辺が最小化されたかどうかの検査の間に呼び出される。もし辺が最小化されていればこの関数が呼び出される。
vis.edge_not_minimized(e, g) もまた、アルゴリズムの第二段階の間、各辺が最小化されたかどうかの検査の間に呼び出される。もし辺が最小化されていなければ、この関数が呼び出される。これはグラフ中に負の閉路が存在する時に起こる。
Example
Bellman-Ford のアルゴリズムを用いた例が examples/bellman-example.cpp中にある。
Notes
ビジタのパラメータは値渡しされるので、もしビジタが状態を持っているなら、アルゴリズムの間のいかなる状態の変更も、送ったビジタ・オブジェクトには行われずビジタ・オブジェクトのコピーに対して行われる。故にポインタまたはリファレンスによってこの状態をビジタに保持させる事を望むかもしれない。

②リンク状態アルゴリズム：LSA（ link state algorithm ）

LSAでは、各ノードが用いるのはネットワークのマップであり、それはグラフの形で格納されている。このマップをつくるために、全てのノードがネットワーク全体に「自分が接続しているノード」をブロードキャストする。各ノードはそのデータをもとに、個々独立してマップを計算し生成する。自分で生成したマップをもとに、各ノードは他のノードへの最短経路を決定する。
最短経路の計算にはダイクストラのアルゴリズムが用いられる

このアルゴリズムはネットワーク全体を木構造で表現する。木の根（最初の要素）は各ノードそれ自体である。次いで、ノードの集合から未登録のノードを一つずつ木に加えていく。

加えるノードは既に木に存在するノードのどれかから到達できるノードのうち、最も少ないコストで到達できるものである。ネットワーク上の全てのノードを登録するまでこれを繰り返す。

木構造ができあがったら、それを用いて、ルーティングテーブルをつくる。最良の「次の相手」等がそこに登録される。

代表例としてOSPF (Open Shortest Path Fast)がある！

ダイクストラ法（構造化ﾌﾟﾛｸﾞﾗﾐﾝｸﾞの提唱者ｴﾄﾞｶﾞｰ･ﾀﾞｲｸｽﾄﾗ が考案）

Ubilabs Innovations

グラフ理論における最短経路問題(与えられた重み付きグラフの2頂点間を結ぶ路の中で最小の重みを持つ路を求める問題)を解くためのアルゴリズム。重み付きグラフにおいて始点sから他の点への最短経路を求める。まず、

S: = {s}

$T := V \setminus \{s\}$

p(s): = 0

とする。さらに、各 $i \in T$ に対して、もし辺siが存在すれば

p(i): = wsi

q(i): = s

とする。ただしここでwijは辺ijのコストとし、辺ijが存在しない場合は +∞とする。Vは頂点集合である。辺siが存在しない場合は、$p(i) := +\infty$ とする。

次に、以下の操作を、Tが空集合となるまで繰り返す。

$p(k) = \min\{ p(j) \mid j \in T \}$

と $k \in T$ を一つ取り、

$S := S \cup \{k\}$

$T := T \setminus \{k\}$

とする。

$j \in T$ に対して、もしp(k) + wkj < p(j)ならば

p(j): = p(k) + wkj

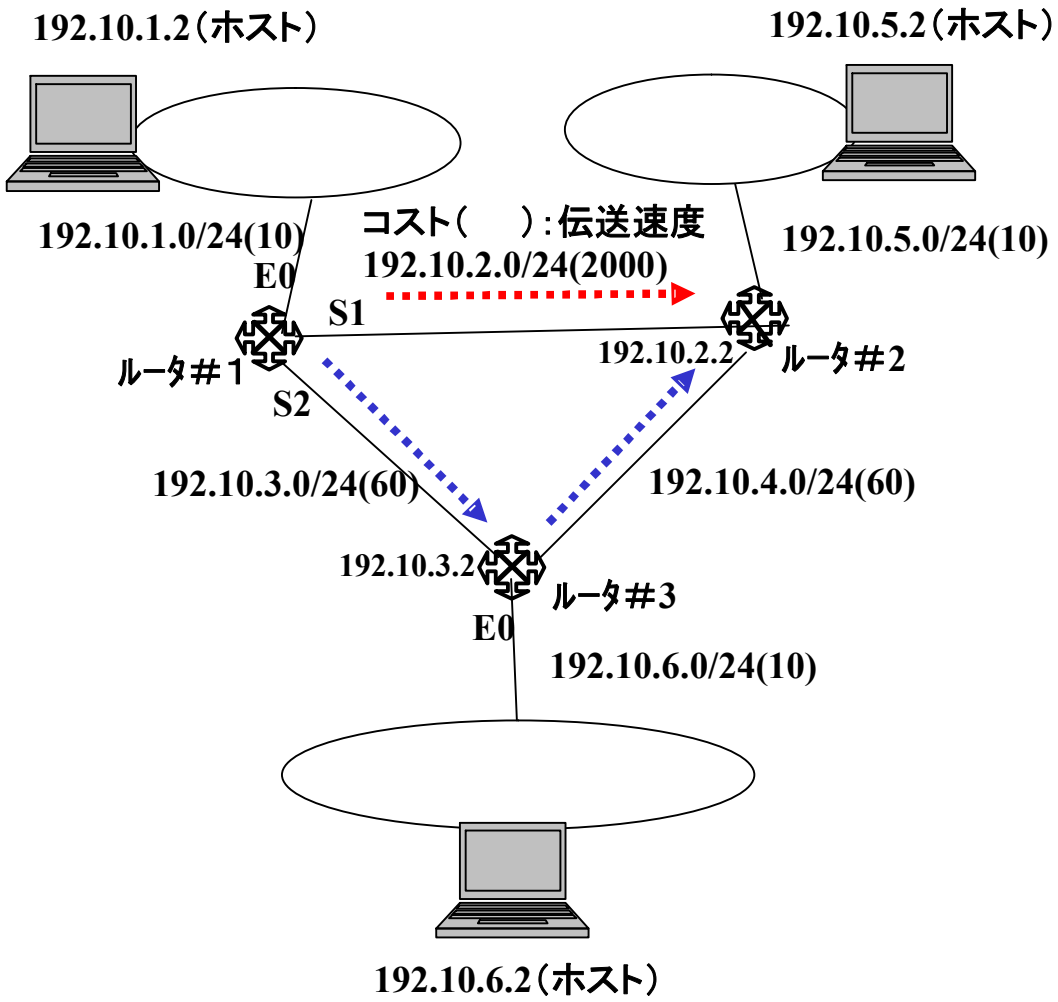
q(j): = k

とする。

これが終了したとき、pjはsからjへの最短距離となっている。また、点列 $\{j, q(j), q(q(j)), \ldots\}$ は、その最短経路での経由点を逆順にしたものになっている。

DVAかLSAどちらが有利か？

RIPまたはOSPFで



ルータ＃1のルーティングテーブル（RIP）

宛先	ネクストホップ	ホップ数
192.10.1.0	接続（E0）	—
192.10.2.0	接続（S1）	—
192.10.3.0	接続（S2）	—
192.10.4.0	192.10.2.2 (S1) 192.10.3.2 (S2)	1 1
192.10.5.0	192.10.2.2 (S1)	1
192.10.6.0	192.10.3.2 (S2)	1

ルータ＃1のルーティングテーブル（OSPF）

宛先	ネクストホップ	コスト
192.10.1.0	接続（E0）	—
192.10.2.0	接続（S1）	—
192.10.3.0	接続（S2）	—
192.10.4.0	192.10.3.2 (S2)	120
192.10.5.0	192.10.3.2 (S2)	130
192.10.6.0	192.10.3.2 (S2)	70

DVAの課題と対策

IRI
Ubilabs Innovations

1．RIP-1では、ホップ数だけを指標とするため、ルータ#1からルータ#2経由でネットワーク192.105.0へは直接パスを選択するが、実際は、ルータ#3経由の方が有利。

2．ホップ数の上限値（通常は15）があり、超えると到達不能となる。

3．ルート情報交換に定期的ﾃﾞｨｽﾀﾝｽ･ﾍﾞｸﾀｰ･ﾌﾞﾛｰﾄﾞ　キャストを用いるため、タイマー起動間隔でしか更新さrないため、障害発生からの収束時間が遅くなる。

→RIP-2での対策：ﾄﾘｶﾞｰ･ｱｯﾌﾟﾃﾞｰﾄ（障害発生時）

4．クラスフルでVLSMやCIDRをサポートしていない。

→RIP-2とEIGRPは、VLSM、CIDRをｻﾎﾟｰﾄ！

→IGRP（Interior Gateway Routing Protocol）とEIGRP（Enhanced Interior Gateway Routing Protocol）は、ﾊﾟｽに沿ったﾘﾝｸ特性をﾒﾄﾘｯｸに組合せた計算機能あり！

VLSM（Variable Length Subnet Mask：可変長サブネットマスク）

クラスフルアドレッシングの「クラスネットワークでは同一のサブネットマスクを使用する」という制限をなくす技術。
１つのネットワークで異なるサブネットマスクを使用することができるようになり，サブネットをさらにサブネット化したり，複数のサブネットをまとめて1つの大きなサブネットにしたりすることが可能。

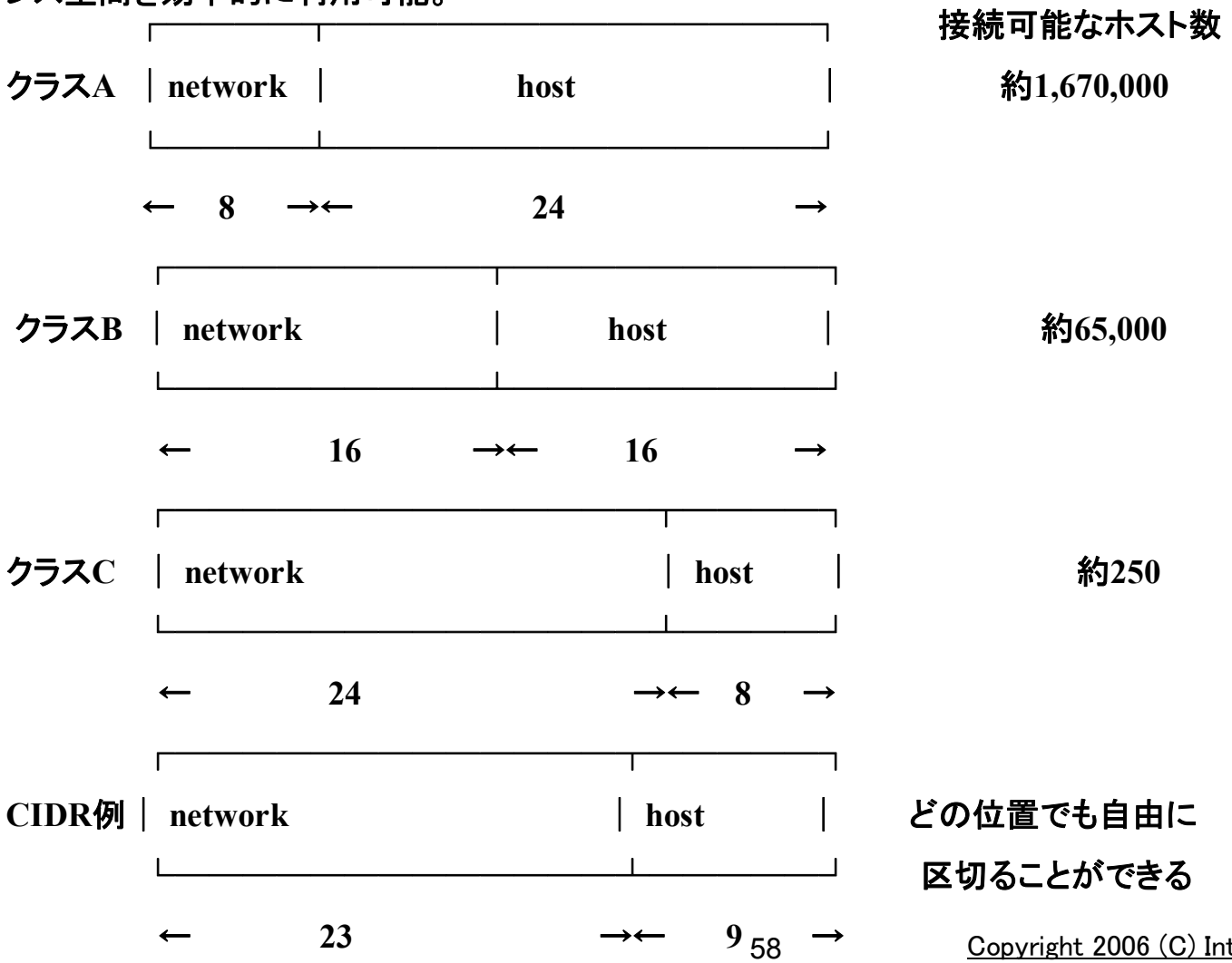
このVLSMの利用の効果

①IPアドレス割当に生じる無駄を省く。
②ネットワークで使用できるIPアドレス実数の増加
②ルート情報を集約

CIDR（Classless Inter-Domain Routing）

IRI Ubilabs Innovations

サイダーと読む。CIDRは、クラスを使わないIPアドレスの割り当てと、経路情報の集成を行う技術。クラスとは、IPアドレスのネットワーク部とホスト部を決められたブロック単位 で区切る方法で、簡単だがアドレス空間の利用に無駄が生じる。これに対しクラスを使わないCIDRでは、任意のブロック単位で区切ることができ、IPアドレス空間を効率的に利用可能。



LSAの特徴とメリット

IRI Ubilabs Innovations

1. 反復分散ﾃﾞｰﾀﾍﾞｰｽﾓﾃﾞﾙ使用したより複雑なﾌﾟﾛﾄｺﾙ

2. ﾘﾝｸｽﾃｰﾄ（情報要素：ﾄﾞﾒｲﾝ内ﾘﾝｸやﾉｰﾄﾞの情報）をﾙｰﾀが交換

3. ﾙｰﾃｨﾝｸﾞ・ﾃｰﾌﾞﾙは、交換せず、隣接ﾙｰﾀ、接続に関するﾒﾄﾘｯｸ情報を保有。

4. ｼﾞｸﾞｿｰ・ﾊﾟｽﾞﾙ型ｱﾙｺﾞﾘｽﾞﾑで、ﾄﾞﾒｲﾝ内の全ﾉｰﾄﾞが、ﾊﾟｽﾞﾙﾋﾟｰｽ情報のｺﾋﾟｰを受信。

5. ﾈｯﾄﾜｰｸ内の各ﾙｰﾀは、個々にﾊﾟｽﾞﾙを組み立てる。

→ホップ数に制約されない
→リンクの帯域と遅延を計算可能
→収束の速さ
→VLSMとCIDRをサポート
→階層化に優れる

しかし、LSAは、ﾄﾞﾒｲﾝ間ﾙｰﾃｨﾝｸﾞへの適用不可！

そこで、DVA型の*BGP*をﾄﾞﾒｲﾝ間ﾙｰﾃｨﾝｸﾞへ適用！

ご清聴ありがとうございました