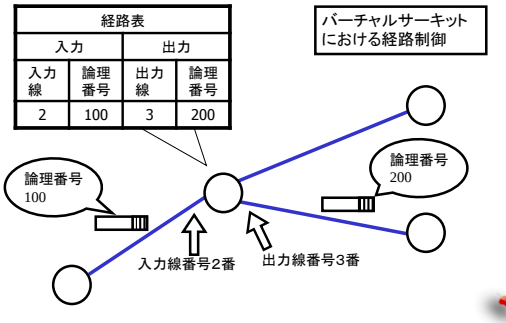
4. インターネットの技術

九州工業大学　情報工学部
電子情報工学科
尾家　祐二、川原憲治
oie@cse.kyutech.ac.jp

4.1 経路制御(routing)



4.1 経路制御(routing)

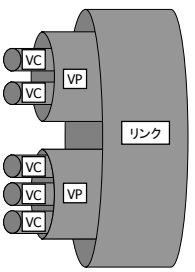
- パケットがあて先ホストに正しく到着するために…
  - 中継ルータは正しい経路を選択する必要がある
- 経路制御方式
  - バーチャルサーキット方式（仮想回線方式）
    - ATM, X.25で用いられている
    - あて先までの経路を確認してからパケットを配送する
  - **データグラム方式**
    - IPで用いられている
    - あて先までの経路を確認せずにパケットを配送する
    - ベストエフォートサービス

4.1 経路制御(routing)

- **データグラム方式**
  - ソースルーティング(source routing)
    - 送信側が経路を決定
    - 各中継ノードの負荷は軽い
    - 送信ホストで全経路を決定することが難しい
  - **ホップバイホップルーティング**(hop-by-hop routing)
    - 各中継ルータがパケットに記載された宛先ホストアドレスと経路表を参照して経路を決定
      - **静的経路制御** (static)
        手動で経路表を設定
      - **動的経路制御** (dynamic)
        ネットワークの状態に応じて自動的に経路表を更新

4.1 経路制御(routing)

- バーチャルサーキット方式
  - 高い信頼性を備えている
  - 経路を確立する際に，各中継ノードにおいて，論理回線番号(VPI+VCI)の登録が必要
  - VPI(Virtual Path Identifier)
    - 物理リンク内に作成される論理的な通信路
  - VCI(Virtual Channel Identifier)
    - VP内に作成される論理的な通信路
  - 1つ1つの通信に対する処理負荷は高い

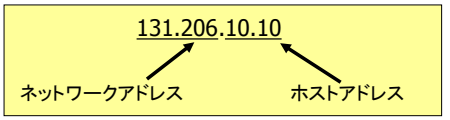


4.1 経路制御(routing)

- (a) IPおよびIPアドレス
  - 現在，主に使用されているのはIPv4
  - 次世代のIP(IPng, IP Next Generation)はIPv6
  - IPv4のアドレス
    - 11000000-10101000-00000000-00000010(ビット列表記)
      - 192.168.0.2(DDN)
      - 2^{32}≒ 43億
  - IPアドレスの目的
    - ホストを識別する
    - 効率のよい経路制御を行うための構造をもつ必要がある
      - （例）電話番号：092は福岡市

4.1 経路制御(routing)

- IPアドレスの構造
  - ネットワークアドレス部
    - ホストが接続されているネットワークを識別
    - 九州工業大学のネットワークアドレス = 131.206
  - ホストアドレス部
    - ネットワーク内でのホストを識別
    - 下の例ではホストアドレス = 10.10



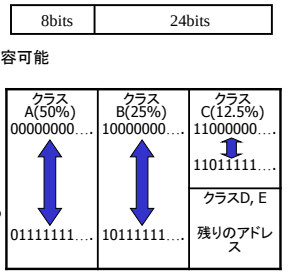
4.1 経路制御(routing)

- クラス化
  - 利点：経路制御を容易にした
  - 問題点：
    - クラスAがアドレス空間の50%を占めている
      - 無駄が多い
    - クラスBのネットワーク識別子
      - 2^{14}= 16384（先頭の2ビットが固定） → 需要が大きい
    - クラスCのネットワーク識別子
      - 2^{21}≒ 210万（先頭の3ビットが固定）

使用可能なIPアドレス空間が足りない

4.1 経路制御(routing)

- IPアドレスのクラス
  - クラスA(先頭ビットが0)
    - ネットワーク識別子は8ビット
    - 最も大規模なネットワークを収容可能
  - クラスB(先頭ビットが10)
    - ネットワーク識別子は16ビット
  - クラスC(先頭ビットが110)
    - ネットワーク識別子は24ビット
  - クラスD(先頭ビットが1110)
    - マルチキャスト用に用いられる
  - クラスE(先頭ビットが1111)
    - 実験用に予約

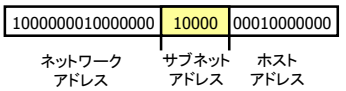


4.1 経路制御(routing)

- IPアドレス枯渇問題への対策
  - IPv6：アドレス空間の拡大
    - 32ビットから128ビットへ
    - 例： 1A2B:3C4D:5E6F:7081:92A3:B4C5:D6E7:F809
  - プライベートIPアドレスの導入：
    - セキュリティ向上
    - End-to-End間の通信に制限
    - グローバルな一意性を緩和
      - NAT(Network Address Translator)でアドレス変換

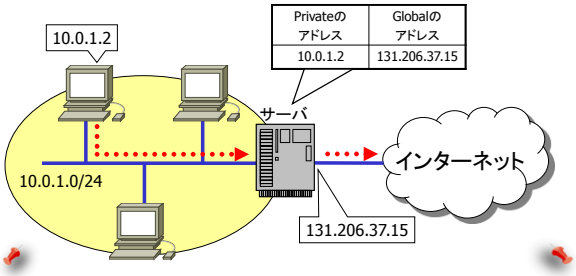
4.1 経路制御(routing)

- 特殊なホストアドレス
  - すべてのビットが0 ＝ ネットワーク自体を表す
    - 例：131.206.37.0
  - すべてのビットが1 ＝ ブロードキャスト
    - 例：131.206.37.255
- サブネットワーク化
  - ホスト識別子の空間が広い場合，その空間をさらに分割する(サブネット，subnet)



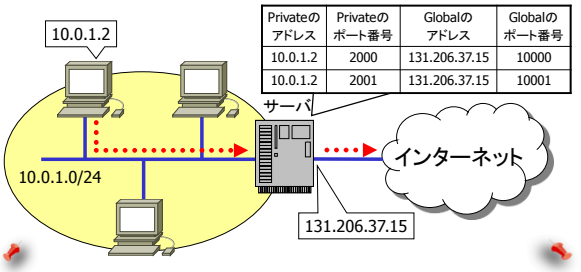
4.1 経路制御(routing)

- NAT
  - プライベートIPアドレスをグローバルIPアドレスに変換



参考

- NAPT(IPマスカレード)
  - ポート番号も変換
  - 複数のプライベートIPに対応できる



Privateの アドレス	Privateの ポート番号	Globalの アドレス	Globalの ポート番号
10.0.1.2	2000	131.206.37.15	10000
10.0.1.2	2001	131.206.37.15	10001

4.1 経路制御(routing)

- 経路情報の種類(3種類)
  - 距離ベクトル(distance vector)型
    - 距離と方向による経路制御
    - RIP(Routing Information Protocol)
  - リンク状態(link state)型
    - ネットワーク全体の接続状態を把握して経路制御
    - OSPF(Open Shortest Path First)
  - パスベクトル(path vector)型
    - 自分自身から目的地へ到達するまでに辿る経路を使用する
    - BGP(Border Gateway Protocol)

4.1 経路制御(routing)

- クラスレス化
  - IPアドレス空間を有効利用
  - 効率的な経路制御
  - CIDR(Classless InterDomain Routing)
  - ネットワーク識別子を可変長にした

192.64.0.0 / 16

ネットワークアドレス　　プレフィックス(prefix)

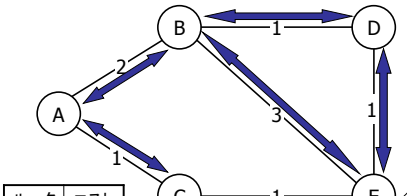
4.1 経路制御(routing)

- RIP(Routing Information Protocol)
  - ネットワークのIPアドレスとネットワークまでのホップ数(0~16)を交換
  - 直接接続されたルータ間で，UDP(ポート番号は520)を用いて情報を交換
  - 30秒ごとに情報を交換
  - 180秒間隣接ルータから情報が無い場合は，到達不可能とする
  - Bellman-Fordアルゴリズムで最短経路を計算

4.1 経路制御(routing)

- (b) 経路情報およびその交換のためのプロトコル
  - IGP(Interior Gateway Protocol)
    - AS内の経路制御プロトコル
  - EGP(Exterior Gateway Protocol)
    - AS間の経路制御プロトコル
  - 各ルータは経路情報を交換している
    - ある宛先までの距離（ホップ数）
    - この情報をもとに宛先までの経路表を作成

4.1 経路制御(routing)



ルータ	コスト
A	1
B	3
D	2
E	1

ルータ	コスト
A	2
B	2
C	1
D	1

4.1 経路制御(routing)

- OSPF(Open Shortest Path First)
  - ネットワークの地図に相当する情報を交換
    - 各リンクが直接接続しているルータ
    - リンクのコスト
  - ダイクストラアルゴリズムで最短経路を計算
- BGP(Border Gateway Protocol)
  - 利用可能な経路の中継AS番号と，到達可能なネットワークのリストを交換

4.1 経路制御(routing)

- Bellman-Fordアルゴリズム

宛先ノード： D
m回情報交換して求められたノードiからノードDまでのコスト： b_j^m

初期条件： $b_i^0 = \infty, i \neq D$

ステップ1： b_i^mを次式で求める

$$b_i^m = \min_{j \in V_i}[b_j^{m-1} + c_{ij}]$$

ステップ2： 次式が成立すると終了し，成立しない場合はステップ1に戻る

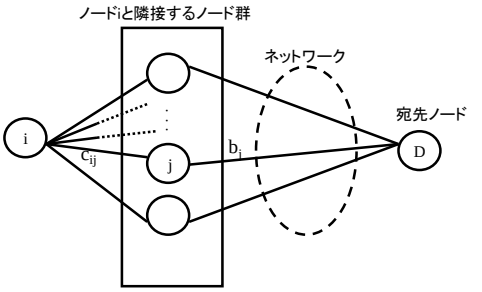
$$b_j^k = b_i^{k-1}$$

ノードiと隣接するノードの集合： V_i
ノードiとノードj間のリンクのコスト： c_{ij}

4.1 経路制御(routing)

リンクの始点	リンクの終点	コスト
A	B	2
A	C	1
B	A	2
B	D	1
B	E	3
C	A	1
C	E	1
D	B	1
D	E	1
E	B	3
E	C	1
E	D	1

4.1 経路制御(routing)



4.1 経路制御(routing)

- (c)経路探索のアルゴリズム
  - Bellman-Fordアルゴリズム
    - ルータ自身を始点として経路を計算
    - ネットワークの全体像はわからない
    - ネットワークに変化が生じると，隣接ルータに通知
    - 隣接ルータは経路を再計算し，もとの経路表との違いがあれば，経路表を更新し隣接ルータに通知
    - ネットワークの変化が全体に伝わるまでに時間がかかる
    - ループができる場合がある
    - 計算量は $O(N^3)$

4.1 経路制御(routing)

- ダイクストラアルゴリズム
  - 各ノードがネットワーク全体の「地図」を知っていることを仮定
  - ネットワークトポロジとノード間を接続するリンクのコストに関する情報を交換
  - ある始点から他の全てのノードまでの経路を計算可能
  - 計算量は $O(N^2)$

4.1 経路制御(routing)

- ダイクストラアルゴリズム

初期条件: $d_s = 0, d_j = \infty, j \in V - \{S\}, P = \phi$

ステップ1: $d_i = \min_{j \in V-P} \{d_j\}$ を満たすノード i を選ぶ

ステップ2: そのノード i に対して最小コストを確定し，集合 P に追加．ノード i と直接接続されている全てのノード k に対する距離を更新

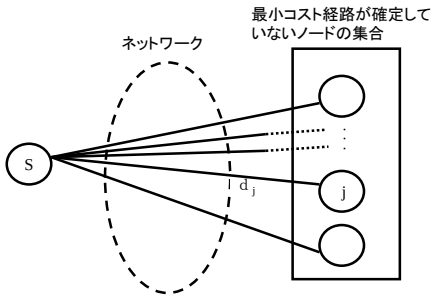
ステップ3: $P = V$ であればアルゴリズムを終わり，そうでなければ，ステップ1に戻る

ネットワーク内の全ノードの集合: V　　始点ノード: S
他の全ノード j までの最小コスト: d_j

4.2 誤り制御

- 誤り制御(error control)
  - ビット誤り，パケット廃棄などのパケット誤り(packet error)を検出・回復する
  - 誤り検出
    - トランスポート，インターネット，データリンクの各プロトコルに備わっている
  - 誤り回復
    - FEC(Forward Error Correction)方式
      - 誤り回復のために冗長な情報を付加
    - ARQ(Automatic Repeat reQuest)方式
      - パケットの再送により誤り回復
      - TCPで用いられる
      - パケット誤りを陽に知らせる方法と，送信者が推測する方法がある

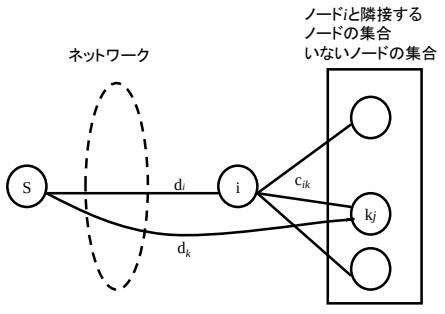
4.1 経路制御(routing)



4.2 誤り制御

- 選択的確認応答(selective acknowledgement)
  - パケット誤りをはっきりと知らせる方法
  - TCPにオプションとして搭載し始められている
- 累積確認応答(cumulative acknowledgement)
  - 現在のTCPで広く用いられている
  - パケット誤りの判断の方法
    - タイムアウト(timeout)
    - 重複確認応答(duplicate acknowledgement: 重複ACK)

4.1 経路制御(routing)

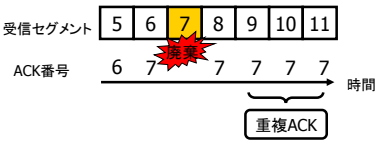


4.2 誤り制御

- (a)　受信側におけるビット誤り検出
  - 雑音により，送信したビット列の信号が正しく受信機に伝わらない場合がある
  - ビット誤りを検出するには冗長(redundant)な情報を真のデータに付加する必要がある
  - IP,UDP,TCPではチェックサムフィールドにパリティチェック(parity check)方式が使われている
  - データリンク層ではCRC(Cyclic Redundancy check)符号が使われている

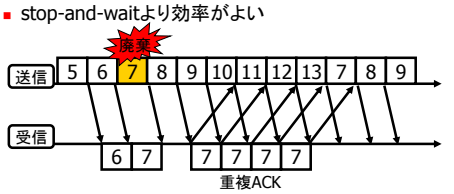
4.2 誤り制御

- (b) 送信側におけるパケット誤り検出
  - 送信側では7番の確認応答を複数受信する(重複ACK)
  - 3つの重複ACKを受信するとセグメント廃棄と判断し，再送(高速再送(fast retransmit))
  - 重複ACKも返ってこないような場合，重度の輻輳と判断
    - 再送タイムアウト時間(Retransmission Time Out, RTO)の設定
  - 選択的確認応答はSACK(Selective ACK)オプションつきTCP(TCP with SACK option)において採用されている



4.2 誤り制御

- go-back-n ARQ
  - 受信側であるパケットに誤りが発生すると，それに対するACKは返さず，それに続いて正しく受信したパケットに対してACKを返す
  - 送信側で重複ACKまたはタイムアウトによってパケット誤りを検出すると，それ以後のパケットを再送する
  - stop-and-waitより効率がよい

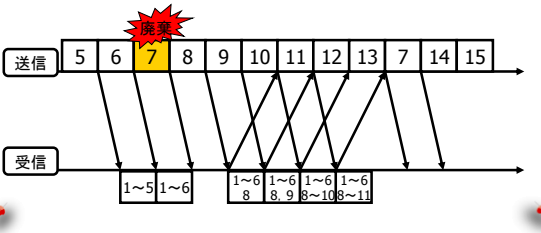


4.2 誤り制御

- (c) 誤り回復のための再送
- ARQには3種類
  - stop-and-wait (ストップアンドウェイト)ARQ方式
  - Go-back-n ARQ 方式
  - selective ACK (選択的 ACK) ARQ方式
  - 現在用いられているのは Go-back-n と selective ACK
    - TCP TahoeとRenoはGo-back-n
    - SACKオプション付きTCPはselective ACK

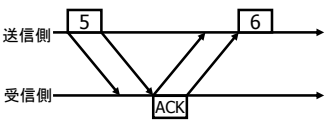
4.2 誤り制御

- selective ACK ARQ
  - ACKにおいて正しく受信したパケットを知らせる
  - 誤りが発生したパケットのみ再送する



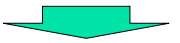
4.2 誤り制御

- stop-and-wait ARQ
  - 送信側は直前のパケットに対するACKが返ってくるまで次のパケットを送信しない
  - 500Byteのパケットを1Mb/sの伝送路を用いて送信
    - 伝送時間は4 msec(=4000bit/1,000,000bit)
    - RTT=40 msecのとき，伝送効率は10%(=4/40)
      - 1Mb/sの伝送媒体を100kb/sの能力としてしか使えていない
  - 広域ネットワークに不向き



4.3 フロー制御と輻輳制御

- ネットワークを介した通信を考えてみる…
  1. 相手ホストの処理能力を考えず送信すると
  2. 処理しきれずパケットが廃棄される
  3. 再送が行われる
  4. 再送パケットの増加により輻輳が発生する



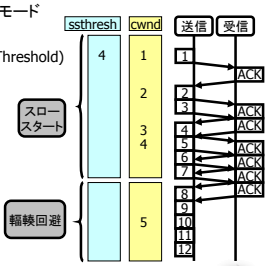
- 通信相手の処理能力を考慮して送信することが必要
  - フロー制御 (flow control)
- ネットワーク内の輻輳(congestion)解消のためのフロー制御も必要
  - 輻輳制御 (congestion control)

4.3 フロー制御と輻輳制御

- (a) 輻輳の検出
  - ネットワークからの明示的な輻輳通知(explicit congestion notation)
    - フレームリレー，ATMなどで採用されている
  - 送信側による輻輳発生の予測
    - パケット廃棄によって輻輳と判断
    - TCPで行われている

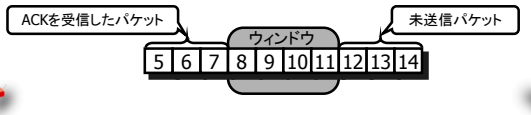
4.3 フロー制御と輻輳制御

- TCP Tahoe と Reno のフロー制御の違い
  - (1)増加
    - ウィンドウサイズの増加に2つのモード
    - スロースタートモード
      - cwnd ≦ ssthresh(Slow Start Threshold)
      - 1つのACK毎にcwnd=cwnd+1
      - 1RTTでcwndが2倍
    - 輻輳回避モード
      - cwnd ＞ ssthresh
      - 1RTT毎にcwnd=cwnd+1



4.3 フロー制御と輻輳制御

- (b) フロー制御
  - レイト制御(rate control)
    - パケットの送信間隔をあけて送信
    - 細かい時間単位(制度の高いタイマ)での制御が必要
  - ウィンドウフロー制御
    - TCPで行われている
    - 受信ホストが送信ホストにたいして使用可能なウィンドウサイズ(告知ウィンドウサイズ)を知らせる
    - 送信側は告知ウィンドウサイズの中で輻輳に応じてウィンドウサイズ(輻輳ウィンドウ(congestion window: cwnd))を変化させる

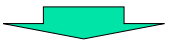


4.3 フロー制御と輻輳制御

- (2)減少
  - タイムアウトまたは重複ACKによってセグメントの廃棄を検出するとssthreth = cwnd/2
  - Tahoe
    - cwnd = 1
  - Reno
    - タイムアウト：cwnd = 1
    - 重複ACK：　cwnd = cwnd/2(廃棄セグメントの再送が成功するまで)
      - 高速リカバリー(fast recovery)

4.3 フロー制御と輻輳制御

- (c) TCPにおける動的なウィンドウサイズ制御
  - 1RTT毎にウィンドウサイズで規定された量のパケットを連続して送信
  - パケット廃棄がなく，全てのパケットのACKが返ってくると，輻輳ウィンドウサイズを増加
  - 輻輳を検出すると，それを回避するためにウィンドウサイズを減少



  - ウィンドウサイズの増減をどのように行うか？

4.3 フロー制御と輻輳制御