情報技術応用特論 第01回

TCP/IP(1)

Copyright (C) Ken'ichi Fukamachi <fukachan@fml.org>, 2021-2025. CC BY-NC-SA 4.0

概要

- アプリケーションプロトコルに慣れましょう。可読性,非暗号化の簡単な例のみ
  - ウエブ: HTTP
  - メールの受信: POP3
  - メールの送信,転送: SMTP
- だいたい、なんでも「**サーバクライアントモデル**」(<- 復習, 分かっている想定)
- アプリケーション層 ... 俗にレイヤー7とも呼ぶが、正確にはOSIモデルのL7と同等ではない
- 実際には裏でDNSも動いていますが、簡単化のため最初はDNS抜きで説明し、後にDNSあり版

(脚注1) **学部の「コンピュータネットワーク」は履修済**の想定です。 ところどころ復習もありますが、 基本用語は知っているはずなので、 省略か（スライドにあっても）軽くふれるだけで飛ばしていきます

(脚注2) われわれプロにとってのインターネットは**データ(パケット)転送システム**のことです。 素人さんには「インターネットってブラウザでいろいろできることじゃないの?」というイメージがあると思いますが、 それは(データ転送後の)ある一つのアプリケーションの動作の話にすぎません。 ブラウザというアプリが圧倒的に利用頻度が高いので、 そういう偏見が生まれるのでしょうけどもね

(脚注3) 多くのものはアプケーション層とトランスポート層の間に暗号化レイヤーをはさむ形なので、 アプリケーションプロトコル単体では暗号云々という話は無関係です

(脚注4) [歴史] TCP/IPの設計は1970年代に始まっていますが、OSIモデルは1980年代です。つまりOSIが後

WWW
- アプリケーション層の例(1) -

WWW (ワールドワイドウエブ)の全体像

- 転送プロトコルは**HTTP**
  - 特にHTTP/1.0はシンプルな**ステートレス**プロトコルの良い例(後述)
- データ(転送内容=コンテンツ)は何でも良い
- ブラウザでは URI を指定する
  - かつては URL と呼んでいたため、歴史的に URL と呼ぶ人も多い

- 用語: URI が URL と URN の上位概念
  - 情報空間で一位に特定できるIDがURI
  - URL (Uniform Resource Locator) ... （サイバースペース上の）場所に相当するもの
    - 例: https://サーバ名/パス(OS上の場所)/
  - URN ... 場所という概念が無いもの
    - 例: 電話番号, 本のISBN番号

```
[URIの例]
https://portal.mc.chitose.ac.jp/portal2/
tel:0123-27-6097
isbn:978-0321336316
```

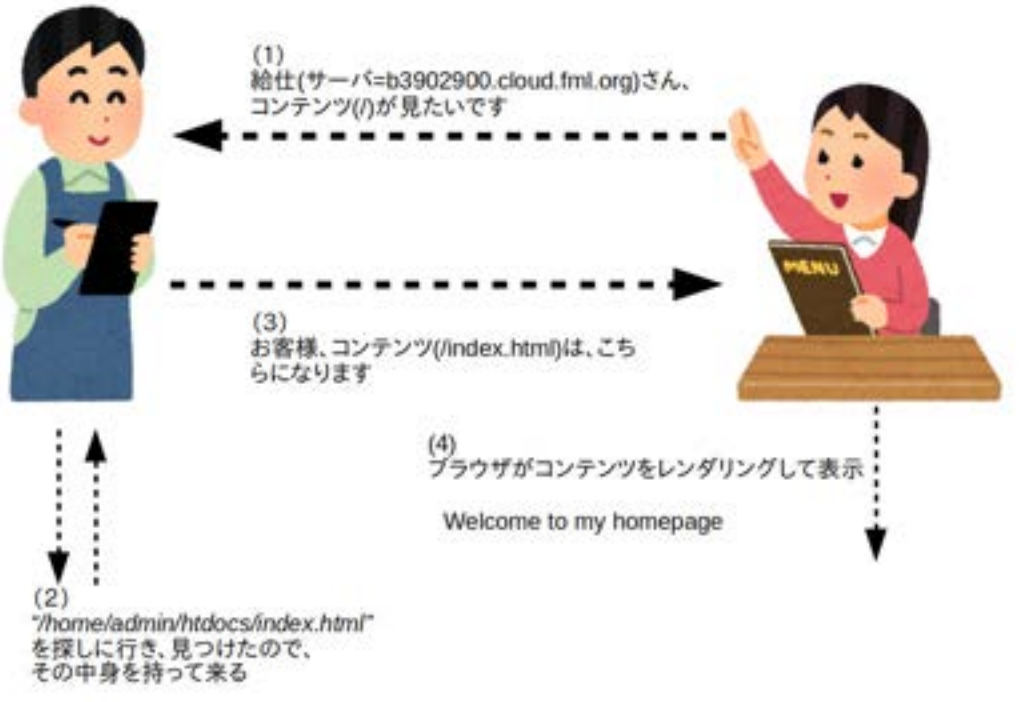
(脚注1) コンテンツの代表例はHTML形式のファイルですが、画像でも動画でも何でもOK。 技術的には、すべて**バイトストリーム**

(脚注2) 【重要】データの種類や表現などはTCP/IPと無関係です！ TCP/IPの仕事はブラウザにデータを確実に届けること

(脚注3) 組織と規格文書： W3C (World Wide Web Consorcium)からの勧告。 IETFでRFCになるものもある。 **だれでも参加可能**

サーバクライアントモデル(復習)

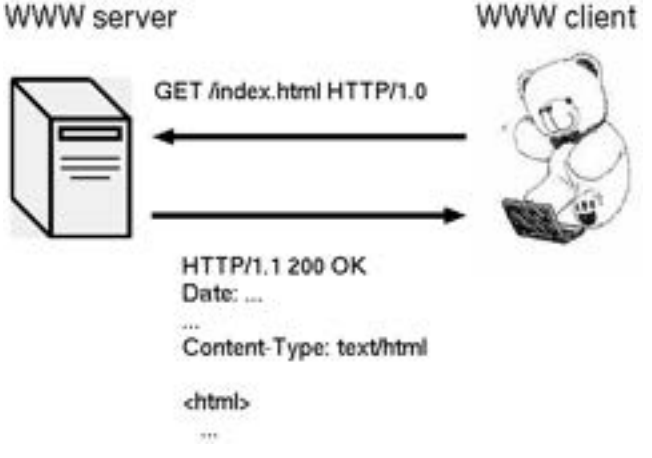
- （クライアントが）**依頼**し、（サーバが）**処理結果を返す。 世の中ほぼ何でもコレで説明可**
  - 例：レストラン、コンビニ、買い物、ポータル、メール、スマートフォンの動作全般
- 例：WWW（右図）
  - (1) URL(http://b2902900.cloud.fml.org/)をブラウザでクリックすると
  - (2) 【WWWサーバの内側の動作】 URL右端の/を/index.htmlと解釈しindex.htmlの中身を返す
  - (3) WWWサーバはブラウザにコンテンツ(index.html)を送り返します
  - (4) ブラウザは受け取ったコンテンツを解釈して表示します(レンダリング)



(脚注) サーバクライアントモデルが抽象化しているのは図の(1)(3)にあたります。 (2)(4)は技術の詳細/各論で、ここは様々です

WWW サーバの動作 (HTTP/1.0 の動作説明)

- WWW サーバはリクエストを待つ
  - ポート番号は 80, プロトコルはTCP
  - 専門用語では「80/tcp で listen(2)」
- クライアント(WWWブラウザ)〜サーバ間にTCPの通信回路を確立(詳細は後日)
- **クライアントからサーバに「このURIのデータをください」とリクエスト(ここからがHTTP)**
  - サーバクライアントモデル
  - サーバ = WWWサーバ
  - クライアント = WWWブラウザ
- サーバからクライアントに返答
  - 「HTTP ヘッダ + データ本体」を返す
  - **ヘッダ**には制御情報が書かれます
  - ヘッダとデータ本体の区切りとして空行を1行挿入します (これはメールと同じRFC822準拠フォーマット)

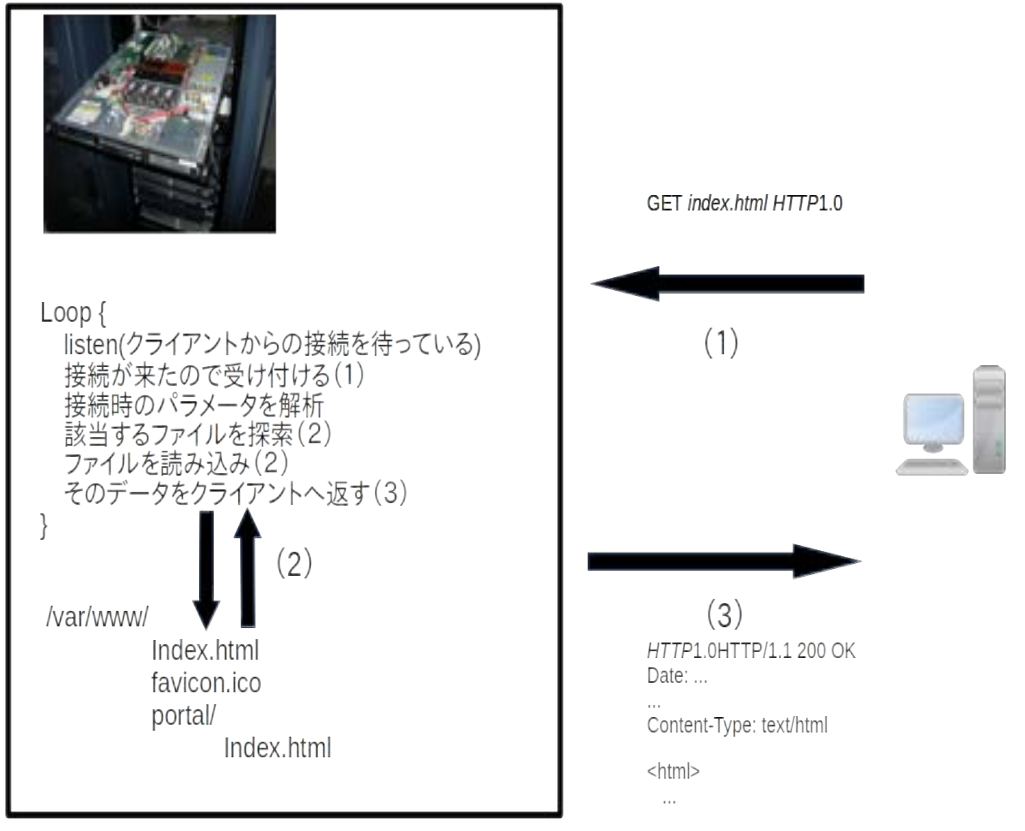


(脚注1) この演習「手動HTTP」は3年生のときにやりましたね？ このあと動画だけ見ます。 ？の人は各自で復習してください

(脚注2) listen(2)の2は引数ではなく、Unix マニュアルの第2章「システムコール」に書いてあるから読めという意味です

WWW サーバの動作 (サーバの内側をもうすこし詳細に)

- WWW サーバは要求を待っている
- (1) クライアントからサーバに接続し「このURIをください」と要求
  - `GET /index.html HTTP/1.0` は「/index.htmlをください」という要求で、 HTTP/1.0はプロトコル番号の指定
- (2) WWWサーバ内の動作
  - URIから対象のコンテンツが/index.htmlであることを割り出す
  - サーバのコンテンツ領域を検索しindex.htmlファイルを探しだす
  - ファイルからデータを読み込む
- (3) サーバからクライアントに返答
  - HTTP ヘッダ + データ本体(index.html)



【参考】 デモおよび演習で使うコマンドについて

```
telnet ホスト
telnet ホスト ポート番号
```

- telnetを使い手動でプロトコルを再現できます、デバッグの基本です
  - telnet というコマンドは本来リモートログインするためのコマンドですが、 オプションで任意のポート番号を引数に取れます
  - 任意のポート番号へアクセスし手動でプロトコルを人間が再現できます
  - ホスト名のlocalhostはホスト自身を意味しています(詳細は後日)

- ただし、このようなことが出来るのは、 歴史のあるTCPアプリケーションの一部に限られます
- 1970〜1980年代からある代表的なアプリケーション群で、 このような人間デバッグ操作が可能です
- 1ビットで十分なのに、 わざわざ可読性のあるテキストで命令をやりとりするのはデータ量の無駄です。 デバッグ要素が強いといえます

(脚注1) 「厳選Unixコマンド 第2版」のtelnet節を参照してください。 -> https://distribution.techbooks.fml.org/

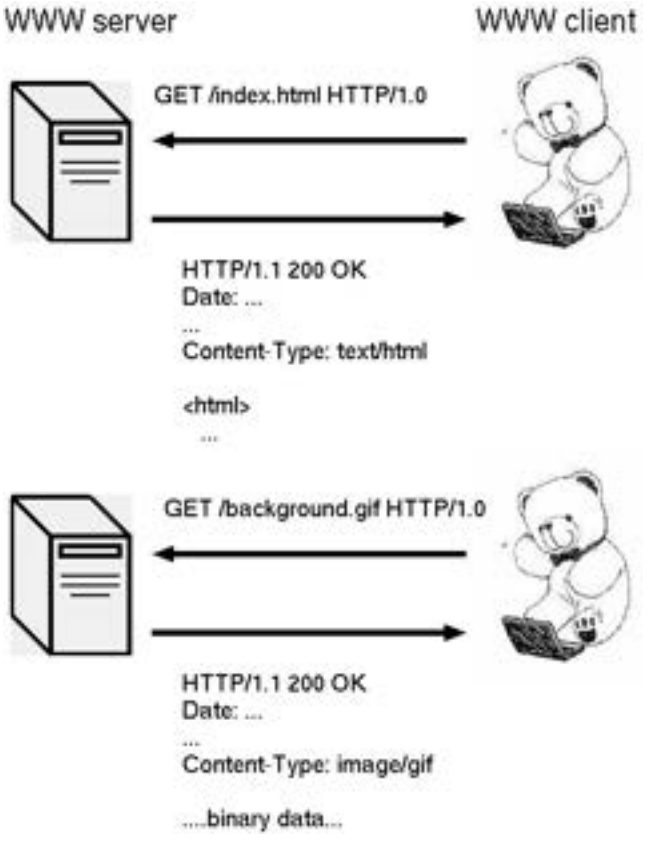
(脚注2) AWSでは Macのひとはtelnetのかわりにncコマンドで試してください、例: nc localhost 80

【復習】HTTP/1.0のデモ

埋め込みが動作しない方は[こちら]

ステートレスプロトコル, ステートフルプロトコル

- **stateless** protocol (例: HTTP 1.0)
  - サーバに要求、データを受信
  - 一回ずつ終了、つまり「状態」がない（ステートレス）
  - HTTP/1.0ではブラウザで表示する部品の数だけ取りにいく
    - 右図の例: index.html 背景画像 ...
    - (もちろん、あれば広告なども)
- **statefull** protocol (例: POP, SMTP)
  - 現在の状態を追跡、状態によって挙動も変わりうる。 メールの受信を例にとると、認証前後で挙動が異なる (次節以降を参照)



【参考資料】
WWWサーバのソースコードの概略

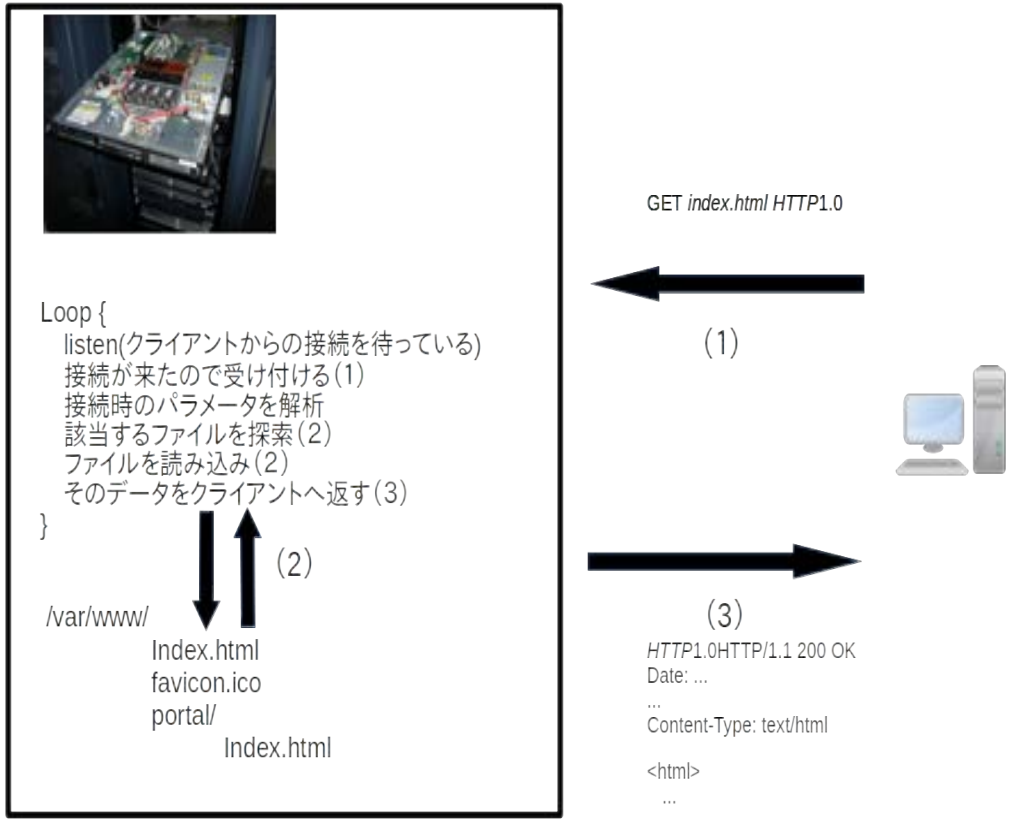
(脚注)【参考】パートは試験の範囲外です

【参考】 listenしてデータIO (C言語)

```c
struct sockaddr_in sa;  // サーバのIPアドレスとポート番号を格納する構造体
int listen_socket, fd;  // ソケット

// TCP/IP の socket (出入り口)を作成, 作成後オプション設定
listen_socket = socket(PF_INET, SOCK_STREAM, IPPROTO_IP);
setsockopt(listen_socket, SOL_SOCKET, SO_REUSEADDR, &sopt, sizeof(sopt));

// socket とアドレス情報 sa を結びつける(だから bind システムコール)
bind(listen_socket, (struct sockaddr *)&sa, sizeof(sa));

// 作成した socket で待ち受ける(listen)
listen(listen_socket, SOMAXCONN);

for(;;) { // 待受して無限ループ,ここでは省略したがデータの送受信はforkした子プロセスが行い,親は待つループを続行
    fd = accept(listen_socket, (struct sockaddr *)&src_addr, &len);// リクエストを受け取る、それまで待つ(停止)
    recv(fd, buf, buflen,....); // データを受け取る
    send(fd, buf, buflen,....); // データを送信する
}
```

【参考】 WWWサーバのソースコード(C言語)

- micro_httpd
  - 300行程度でWWWサーバの本質的なところが分かります！ ただlistenする部分は無いので、すべてが分かるわけではありません
  - 図中の(2)部分だけをC言語で書くとどうなるか?を理解するなら300行で十分ということ
  - 図中の(1)(3)部分はOS(Unix)の機能を利用しています (inetdから起動する前提です。 詳細はUnixオペレーティングシステムの話になるので，割愛します)



【参考】 WWWサーバのソースコード(C言語)

- mini_httpd
  - 4600行でWWWサーバとしての基本機能はすべて備えています
  - **コードリーディングの素材としては、これが適切**でしょう
- thttpd
  - シンプルで小さく移植性のあるセキュアなWWWサーバ。 **実用性のある最小**のもの
  - 11000行なので、なんとか読めるでしょう
  - **WWWサーバとして必要十分**な機能がありますが、 拡張性を求めるなら apache
- 実用的にはapacheとnginxですが大きすぎて勉強の素材としては不適切でしょう
- apache（発音：あぱっち）
  - NCSA httpd に起源をもつ最も古いサーバ
  - 2.x系はスレッド駆動型の設計
  - モジュールによる拡張機能が豊富
- nginx (発音：えんじんえっくす)
  - 非同期通信型の設計
  - リバースプロキシサーバ として有名

(脚注) NCSA = イリノイ大学 米国立スーパーコンピュータ応用研究所。 元々は、 スーパーコンピュータを提供する5施設の一つですが、 最初のモダンなWWWブラウザMosaicの開発で有名です。 ソースコードの系譜は、NCSA Mosaic -> Netscape Navigator -> Mozilla Firefox となります。 初期のWWWブラウザについては、 説著のqiita記事を参照 -> Ep.001「1993/04 世界は色づいた」

【参考】 listenしてhello worldを返す (Go言語)

```
package main

import (
  "fmt"
  "net/http"
)

func handler(w http.ResponseWriter, req *http.Request) { // "変数 型宣言"の順に書く流派
  fmt.Fprintf(w, "hello world")
}

func main() {
  http.HandleFunc("/", handler)
  http.ListenAndServe(":8080", nil)
}
```

- 【参考】 Go言語でWWWサーバをつくる (秋学期のAWS構築ガイドより)

【参考】 listenする (言語Python3)

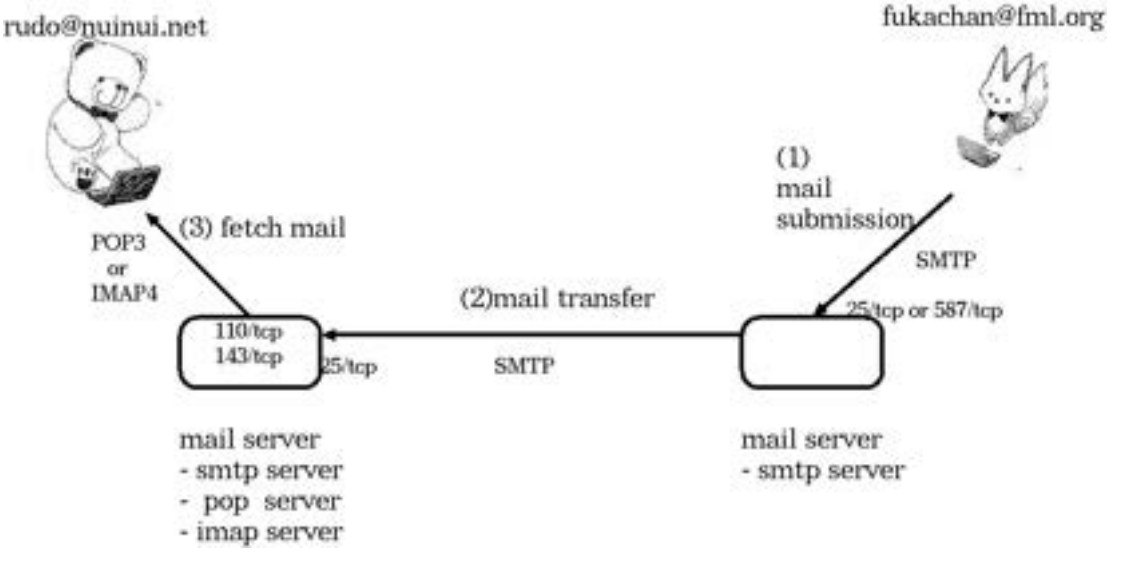
```
#!/usr/bin/env python3

import http.server
import socketserver

port    = 8080
handler = http.server.SimpleHTTPRequestHandler

with socketserver.TCPServer(("", port), handler) as httpd:
    httpd.serve_forever()
```

- 8080/tcp で listen している WWW サーバ
  - 接続を受け付けますが、何も返事を返してくれません;-)
- それにしてもブラックボックス過ぎて裏側が何だか分からないですね;_;
- 演習? hello worldとか返すようにhandlerを書いてみるのもよいものですね?
- 【参考】 PythonでWWWサーバを作る (秋学期のAWS構築ガイドより)

電子メール
- アプリケーション層の例(2) -

メール関連プロトコル

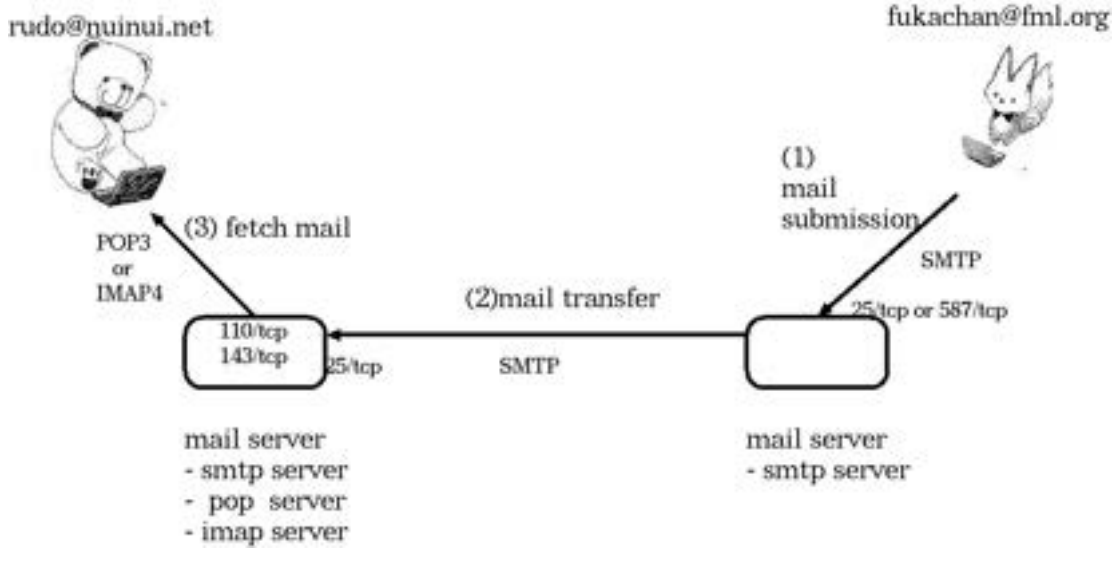
- HTTP/1.0より少し複雑なstatefullの例
- メールの送信(図の(1)),転送(図の(2))
  - SMTP = Simple Mail Transfer Protocol
  - 25/tcp が伝統のポート番号で、 サーバ間メール転送(図の(2))のデフォルト
  - 587/tcpはMail Submission(送信,図(1))
- メールをPCに取込む(図の(3))プロトコル
  - メールソフト(メールリーダ)の裏側
  - POP3
    - Post Office Protocol
    - 110/tcp
  - IMAP4
    - Internet Message Access Protocol
    - 143/tcp



(脚注) 今どき暗号モード(XXXP over TLS)の方を使いますが、 プロトコルの動作を説明することが目的なので、素の方をやります

POP3 全体構成

- 図の一番左の部分
  - 図の(2):メールサーバがメールを受信
    - 受信したメール群はメールサーバに保存されています。 メールの保存場所がメールスプール(spool)
  - メールサーバと同じホスト上で、POPサーバも動いています
  - ユーザはPOP3(もしくはIMAP4)プロトコルでサーバからメールを取り込みます(図の(3))



(脚注1)POP3のほうが単純なので、こちらから先にやります

(脚注2) ブラウザ(webmail)やスマホのアプリも裏側では IMAP4 でアクセスしています。 Webmailがメインでアプリは表示しているだけですという場合はRESTかもしれません。 このへんはアプリの設計思想次第でしょう

POP3 動作の詳細

- POP3
  - 最初は挨拶
  - ログイン(**認証**)
    - ユーザ名とパスワードを送信 -> 本人であることを証明
  - 本人であることを確認後
    - メールを取得
    - メールを削除
  - 本人であることを確認できない時は?
    - 何もできません
    - ログアウトできるだけ
  - ログアウト

```
  メールサーバ starting. ...略...
.. 省略 ...
USER ユーザ名                    // ログイン
+OK Password required for ユーザ名.
PASS パスワード                  // パスワード
+OK ユーザ名 has 1 visible... // 認証OK
RETR 1                           // メール1番を取得
   ... 略(メール本体が表示) ...
+OK 434 octets
DELE 1                           // メール1番を削除
+OK Message 1 has been deleted.
   ... 略 ...
QUIT                             // 終了
+OK Pop server at メールサーバ signing off.
```

(脚注1)+で始まる一連の行は、サーバからの応答

(脚注2)クライアントから「4文字のコマンド 引数」を送信

```
+OK Qpopper (version 4.0.5) at // 挨拶
```

(脚注3) // C言語風のコメントをつけておきます

POP3でメールを受信するデモ

```
+OK Dovecot ...     // サーバからの挨拶
USER ユーザ名               // ログイン
PASS パスワード           // パスワード
STAT                  // 統計情報を表示
LIST                  // メール一覧を表示
RETR 1                // メール1番を取得
DELE 1                // メール1番を削除
QUIT                        // 終了
```

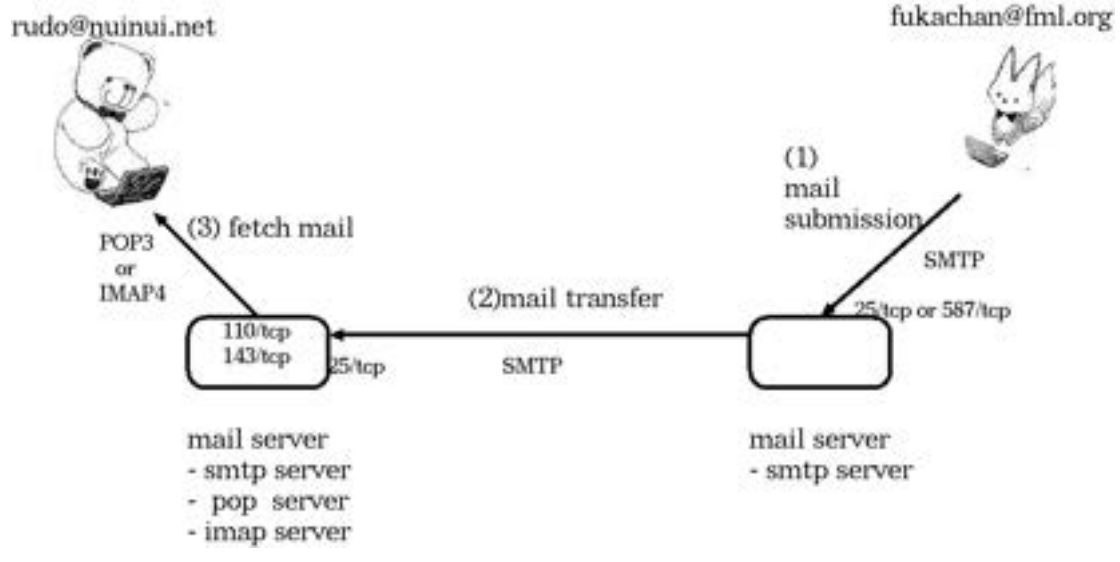
埋め込みが動作しない方はこちら

(脚注1)+で始まる一連の行は、サーバからの応答

(脚注2)クライアントから「4文字のコマンド 引数」を送信

SMTP 全体構成

- 図の(2)がメール転送(mail transfer)
  - 伝統の25/tcpを使います
- 図の(1)が(ユーザからの)メール送信
  - mail submission。 **submissionではユーザを認証して不正アクセスを防ぐ必要**があります
    - SPAM送信サーバとして悪用されないように、自組織のユーザだけが送信可
  - 使うポートは25/tcpもしくは587/tcp



(脚注) 587/tcpは人間用、25/tcpはシステム用といった使い分けができるように、あえて別ポートにした運用がのぞましい

システム用submissionの例: 客先に納品したサーバからログをメールで送信したい場合。 メールサーバで25/tcpのフィルタを書き、このサーバからのみ25/tcpを受け付け、 一般ユーザはは587/tcpを使う

SMTP 動作の詳細

- SMTP
  - 最初は挨拶
  - 送信者を指定
  - 送信先を指定
    - 必要なら複数指定可
    - 一度に指定できるメールアドレスは一つなので、 必要な回数だけ繰り返します
    - 複数指定も、ふつう上限があります(念のためのSPAM対策)
      - 上限は1000人くらいが普通ではないかと
  - メールを送信
  - ログアウト

```
EHLO client.nuinui.net
250-mta.fml.org
 ... 略 ...
250 8BITMIME
MAIL FROM:<fukachan@fml.org>  // 送信者
250 Ok
RCPT TO:<rudo@nuinui.onet>    // 送信先
250 Ok
DATA                          // データ転送開始
354 End data with <CR><LF>.<CR><LF>
メール本文をここで送信
.
250 Ok: queued as 9BB9F86640
QUIT                          // 終了
221 Bye
```

(脚注) // C言語風のコメントをつけておきます

```
220 ... fml.org ESMTP Postfix // サーバからの挨拶
```

SMTPでメールを送信するデモ(ビデオ)

```
220 〜.fml.org ESMTP Postfix  // サーバからの挨拶
MAIL FROM:<fukachan@fml.org>  // 送信者
RCPT TO:<rudo@nuinui.onet>    // 送信先
DATA                          // データ転送開始
QUIT                          // 終了
```

埋め込みが動作しない方は[こちら]

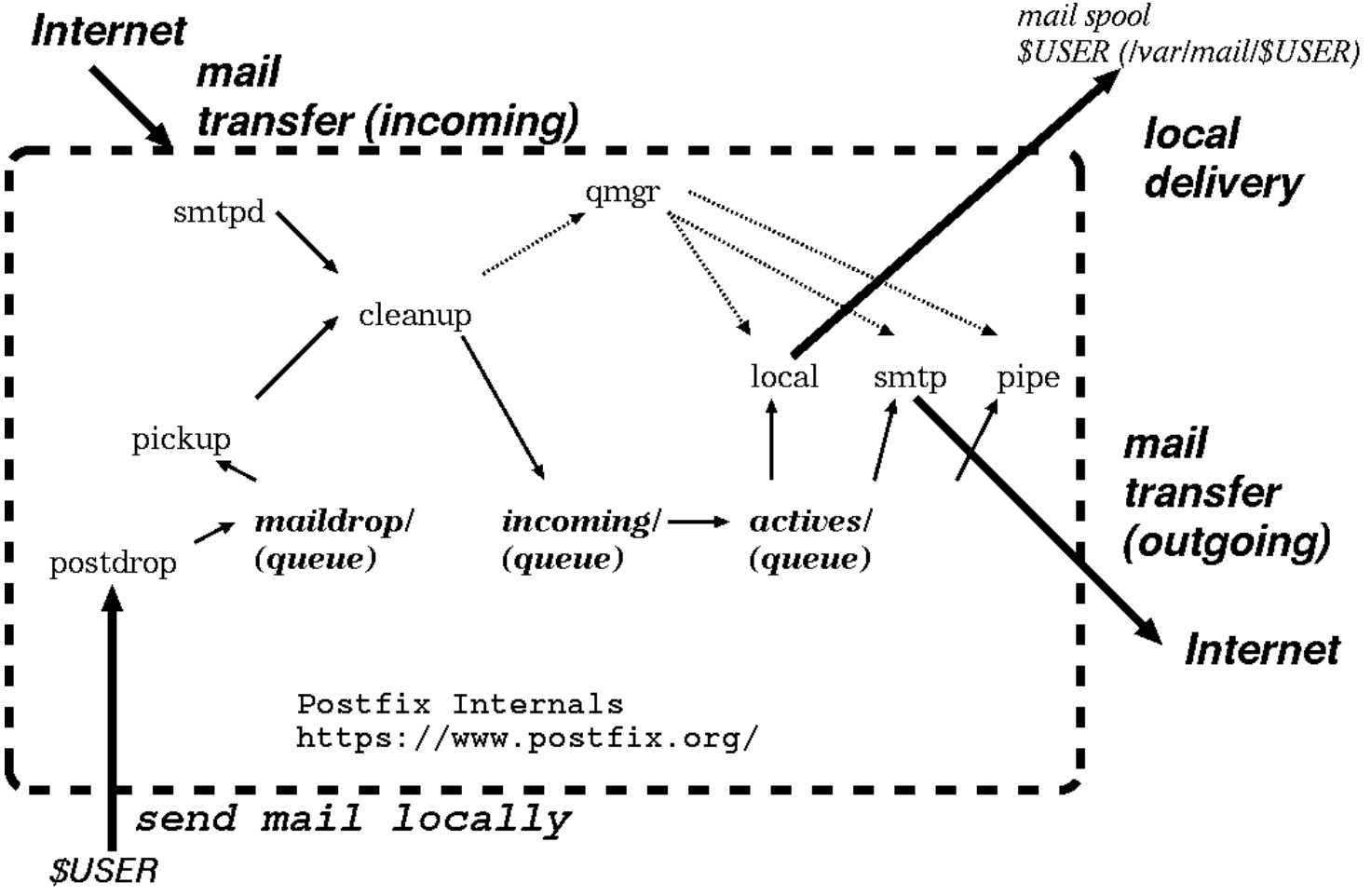
参考資料
メールサーバのソフトウエア紹介

(脚注)【参考】パートは試験の範囲外です

【参考】メールソフトウエア

- コードを読むならPostfixがおすすめです
- メールサーバのソフトウエア (登場順)
  - Sendmail
    - もっとも初期からあるメールサーバで、 互換の基準、いまは商用サポートあり
    - モノリシックアーキテクチャ
    - バグが多かったしな...
    - (90年代末にPostfixがリリースされた時にSendmailは捨てました)
  - qmail
    - 高速でセキュアなのは確かですが、 DJB(作者)の設計思想では運用しづらい。 DJB教に入信しないと...(数学屋とは馬があわない:-)
  - Exim
    - Debian/Linuxのデフォルトなので運用されているサーバ数は多いらしい
  - Postfix
    - 作者W.Venemaは Postfix以前にtcp_wrappersやSATAN、 またFIRSTの運営などセキュリティ分野で知られた人です (高エネルギー物理出身ですけどね:-)
    - モデュラーアーキテクチャの勉強にもおすすめです

【参考】 Postfixの(だいぶ簡略化した)内部構造

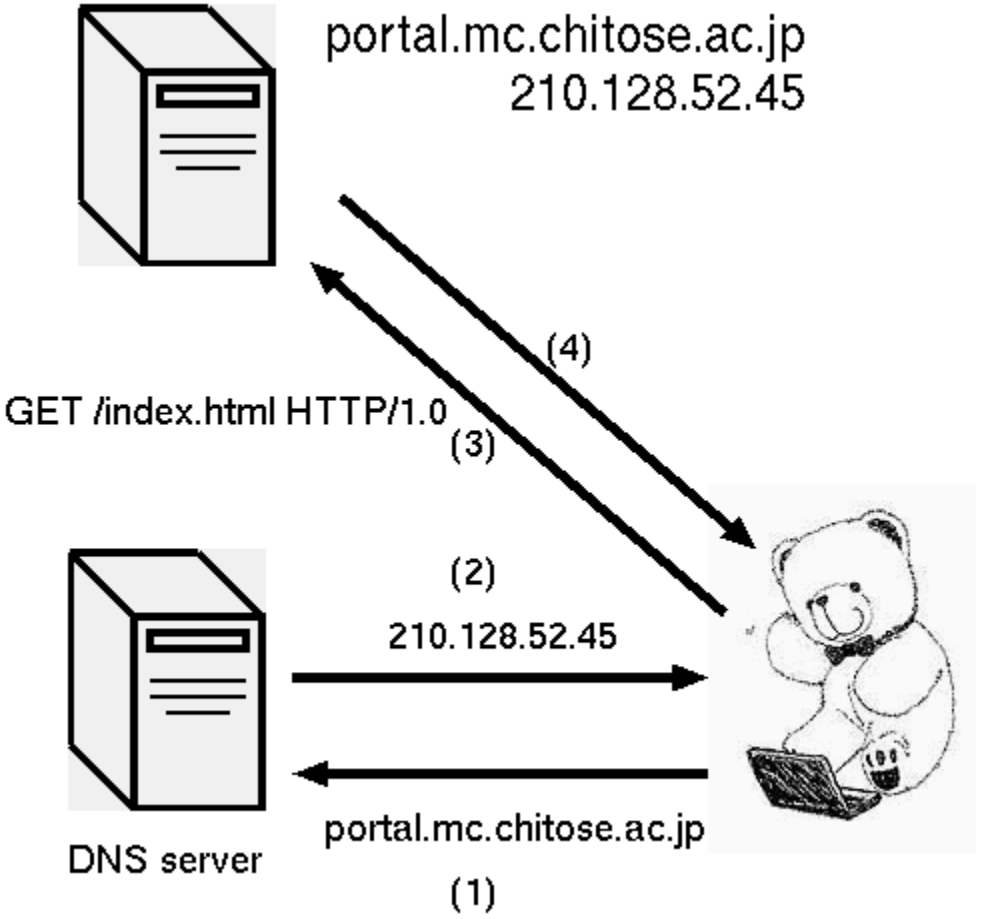


(脚注) だいぶ複雑なので説明は省略しますが、 概略は、ぼくの本の第37〜38章を参照 -> "fml バイブル" (大学図書館にあります)

DNS Part I 全体像と用語

WWWの動作(DNS込み)

- 今まではDNSを省略していました(図(3)(4)のみ)
- URI例: http://porta.mc.chitose.ac.jp/portal2/
- 右図がDNSを含めたHTTPの動作です。 ブラウザにURIを指示すると
  - (1) ブラウザはURIに含まれるサーバ名portal.mc.chitose.ac.jpの IPアドレスをDNSサーバに問い合わせます
  - (2) DNSサーバはブラウザにIPアドレス=210.128.52.45を回答
  - (3) ブラウザは210.128.52.45と通信を始めます(ここからは前節までに解説したとおり)
- このIPアドレスを検索する仕組みを**DNS**と呼んでいます。 **本節はDNS（図の(1)(2)）の解説です**



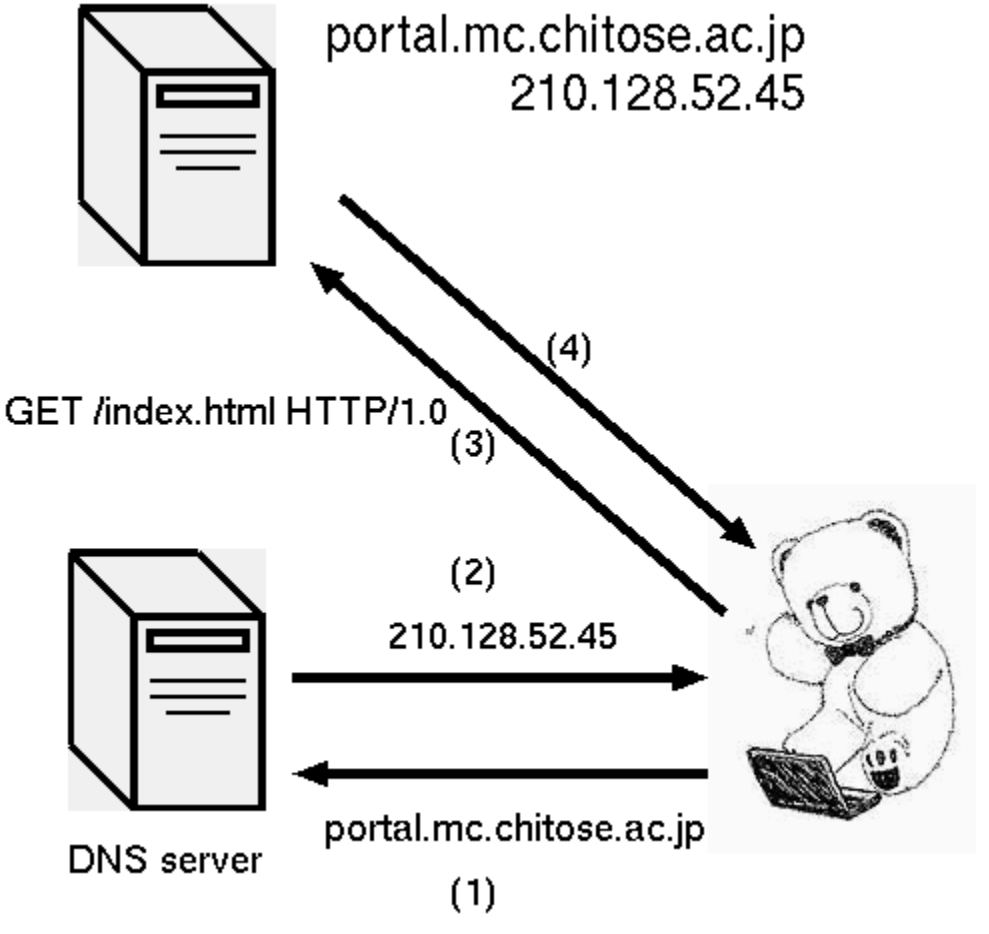
DNS = Domain Name System (Service)

- Domain Name System (Service)の略
- **ディレクトリサービス**の一種
- 文字列からIPアドレスを検索する仕組みが代表例です: DNSの検索例
  - portal.mc.chitose.ac.jpドメインのIPアドレスを知りたい
  - IPアドレス210.128.52.45に対応するサーバ(のドメイン)名が知りたい
  - user@photon.chitose.ac.jp宛メールの送信先(メールサーバ名)を知りたい
  - photon.chitose.ac.jpについて管理しているDNSサーバを知りたい

(脚注) ディレクトリサービス(DS)の典型例が電話帳(e.g. タウンページ)です。 認証システムもユーザとパスワードの一覧という電話帳似のデータを持つので DS です。 例: Active Directory, OpenLDAP(-> 後継? 389-ds?)

あらゆるサービスが裏でDNSを使っています

- プログラムにとっては、すべて**数字**ですが
  - **OSの内部**では、通信先の指定を**数字=IPアドレス**で行っています (例: 210.128.52.45)
- ユーザ利便性はドメインの方がよいはず
- ユーザがIPアドレスでインターネットにアクセスするのは現実的ではありません
  - 何十個も数字を覚えらません
  - 英語に近い表記を使いたい
    - 例: `https://portal.mc.chitose.ac.jp/`
    - 例: `username@photon.chitose.ac.jp`



右図: portal.mc.chitose.ac.jp にアクセスしている様子。

ブラウザは、裏で (1)(2)DNSを使ってIPアドレスを割り出し、 (3)(4)そのIP(210.128.52.45)へアクセスします

DNS関連の重要用語まとめ

- **DNS**
  - Domain Name System (Service)
  - インターネットの基盤技術
  - 地球規模の**分散システム**の成功例
  - **UDP**ベースの数少ない例
- **ドメイン名**
  - .(発音はdot)区切の文字列
    - chitose.ac.jp
  - **FQDN** (Fully Qualified Domain Name)
    - ホストの完全なドメイン名
    - 階層をすべて表記したもの
      - portal.mc.chitose.ac.jp
      - g201pc001.pc.chitose.ac.jp
- **木構造**をしたデータベース
  - 格納している情報を**レコード**と呼ぶ
  - 例:Aレコード(IPアドレス)
- **ネームサーバ** (歴史的にはコレ)
  - 最近はDNSサーバと言うことも多い
  - スライドは**DNSサーバ**に統一
- **名前解決**(name resolution)
  - ネームサーバから情報を取得することで、この動作をするソフトウエアが**リゾルバ** (resolver)
  - 例:ドメイン名からIPアドレスを取得

(脚注1)業界人はdotを発音しませんが、あうんの呼吸で理解してね (脚注2)Googleのchubbyも地球規模ですな...

ドメイン名

- ドット`.` 区切りの文字列
  - 使える文字は、英数字とハイフン(-)です (大文字小文字は区別しません)
  - 階層構造を表現しています(後述)
- 例: URL: http://portal.mc.chitose.ac.jp/
- 例: メールアドレス b2902900@photon.chitose.ac.jp
  - photon.chitose.ac.jp がドメイン名
  - b2902900 部分はユーザ名。 **大文字小文字は区別したほうが良い**でしょう。 区別するかしないかは微妙な話題で、 メールサーバとOSの実装依存ですが、 厳しい方(Unix)に合わせるのが安全です

(脚注) 用法上の注意： ドメインとドメイン名の使い分けはRFCも曖昧で断言しづらい...;_; 領土のような意味合いに力点がある場合にドメインを使っているように読めますが、 同じ意味で使っているようにみえる場合もあり...

ドメイン名をドメインと呼ぶ例: TLD(後述), サブドメイン

たとえばportal.mc.chitose.ac.jpの場合、(一番左の) ホストの役割を説明するportal部分をホスト名と呼ぶ風習がありますが、 **コンピュータを識別するための名前(ホスト名)にはFQDNを使うべき**です

ドメインの階層構造(全体とTLD)

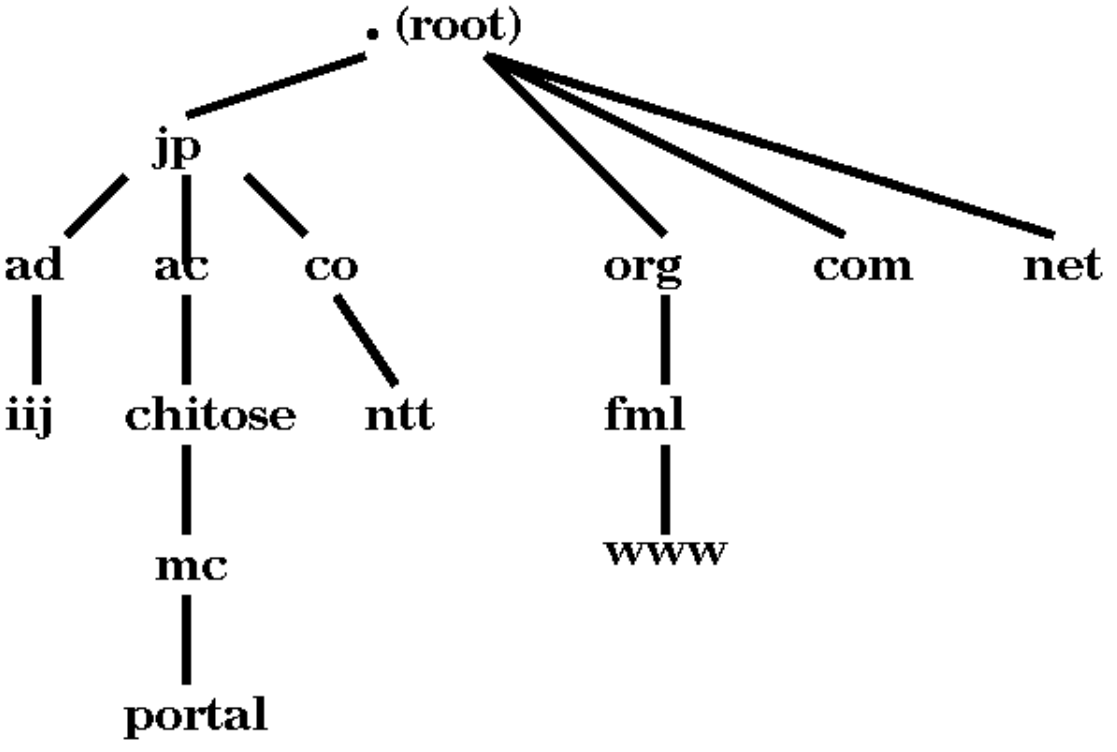
- ドット`.`区切りの文字列
- 階層構造は「**ホスト名.組織.組織種別.国名**」のように並ぶ。より大きな単位が右側
  - 例: chitose.ac.jp
    - chitose (千歳、この場合は大学名)
    - ac (高等教育機関,academic)
    - jp (日本,TLD)
- TLD (top level domain) ... 一番大きな単位(ドメイン名の一番右側)
  - 各TLDごとに登録を管理する組織(レジストラ,民間企業)があります
  - 基本は二文字の国名: jp (日), uk (英), us (米国,USA)ですが、アメリカ特例枠あり
  - アメリカ特例枠 ... (インターネット初期から存在する)us をつけないアメリカのドメイン
    - .com .net .org が有名
      - 1990年代後半に一般開放したため、ドメイン取得者がアメリカ人とは限りません
      - 慣習的に**「ソフトウエア名(OSS).org」** 例: fml.org (このサーバ:-)
    - .edu .gov .mil はアメリカでのみ利用しているドメイン
  - その他も(気づくと)新TLDが登場して収拾がついていない感じ...、例: 目的別? info tel、地域? asia

【参考】ドメインの例

TLD	2LD	属性	例
net		ネットワーク	nuinui.net
com		営利企業	amazon.com zoom.com(zoom.usへ転送)
org		任意団体(ソフト名)	netbsd.org debian.org postfix.org fml.org
edu		アメリカの高等学術機関	mit.edu
gov		アメリカ政府	www.whitehouse.gov
mil		アメリカ軍	af.ml (US空軍)
us			zoom.us
jp	ac	日本国内の高等教育機関	chitose.ac.jp
	co	民間の営利企業	toyota.co.jp amazon.co.jp
	go	政府関係	mext.go.jp
	ed	教育機関	sapporonishi.hokkaido-c.ed.jp
	lg	地方自治体	pref.hokkaido.lg.jp
	名前	なし	toyota.jp amazon.jp(co.jpへ転送)

ドメインの階層は木構造

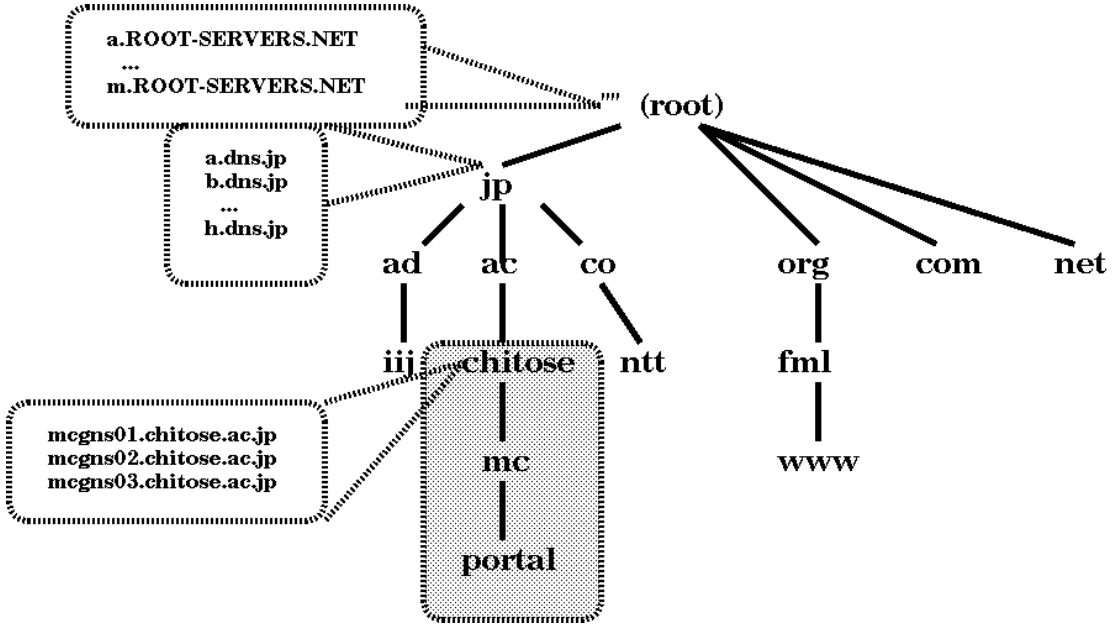
- 各TLDごとに管理
  - 各TLDごとにレジストラ(+ レジストラへの登録代行業者)がいます
  - jpドメインはJPRS(2002/04/01-)です。 なお2002/03/31まではJPNICでした
- DNSは図のように木構造と考えられます
  - 数学のグラフ理論由来で、接点はnode、端はleaf(葉)、線はedge(枝)と呼ばれます。 一番上には「.」で表現される**root(根)ノード**があるという想定です
  - DNSの検索(query)とは、一番上の**rootノードから枝を下っていくこと**です



(脚注) 正しいドメイン名は一番右に.がつきますが、 面倒(?)なので、たいてい一番右の.を省略して書くことが多いです (例: portal.mc.chitose.ac.jp.)

ドメインの木構造とゾーンとドメイン

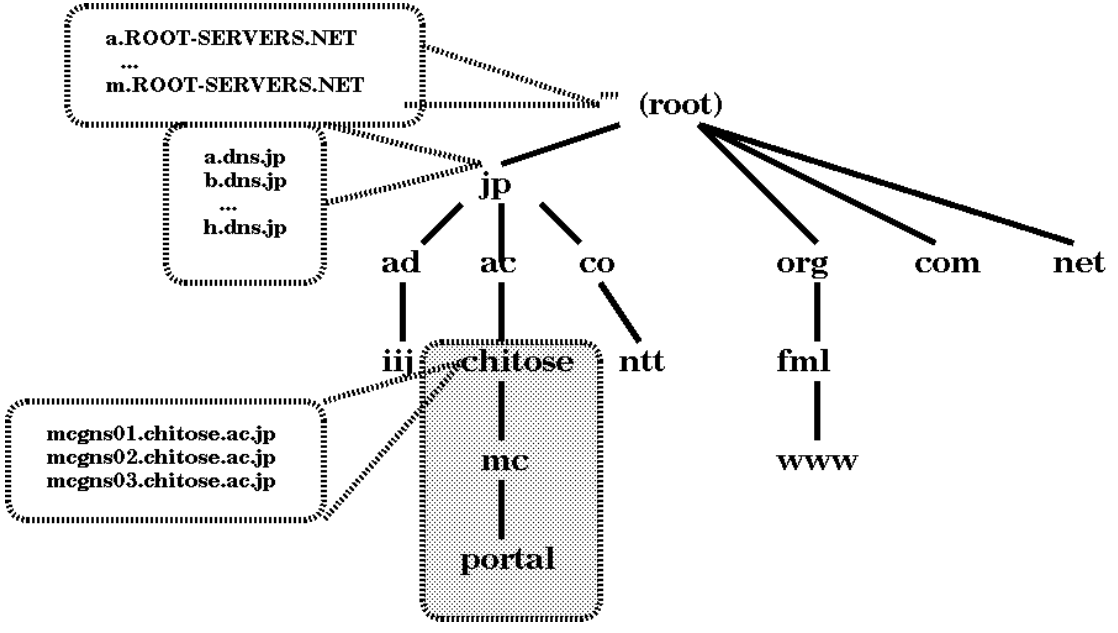
- DNSは無数のゾーンに分割され管理されています。**委任された管理単位をゾーン**と呼びます
- JPRSは(ルートの管理組織から)jpゾーン全体の管理を委任されています
  - さらに小さな単位の ad.jp ac.jp co.jp なども管理しています
- **大学はJPRSからchitose.ac.jpゾーンの管理を委任されています(図の網掛け部分)**
  - ゾーンの名称には一番上のノード名chitose.ac.jpを使います
  - ゾーン内の、 より小さな管理単位であるphoton.chitose.ac.jpやmc.chitose.ac.jpも管理しています。 なお、photonやmcなどは**サブドメイン**と呼ばれます



(脚注) このあたりがDNSサーバの実務（ゾーンファイルには何を書くのか？）に一番関係する部分でしょう。よくわかってないと間違った設定をして、あなたのドメインがインターネットから消えたりしますね:-) とはいえ、 たいていDNSは自爆系で他人には迷惑かからないので、 ぜひドメインとって自分で（bindとかnsdを生）設定してみるといいよね

ドメインの木構造とゾーンとドメインとDNSサーバ

- 各ゾーンにはDNSサーバ群があります
  - 障害に備え複数(最低2)台のサーバ (メインのサーバをprimary,予備機をsecondaryと呼ぶ**ならわし**)
  - 定期的にサーバ間でデータを同期しています。 同期の際のデータ転送は**TCP**です

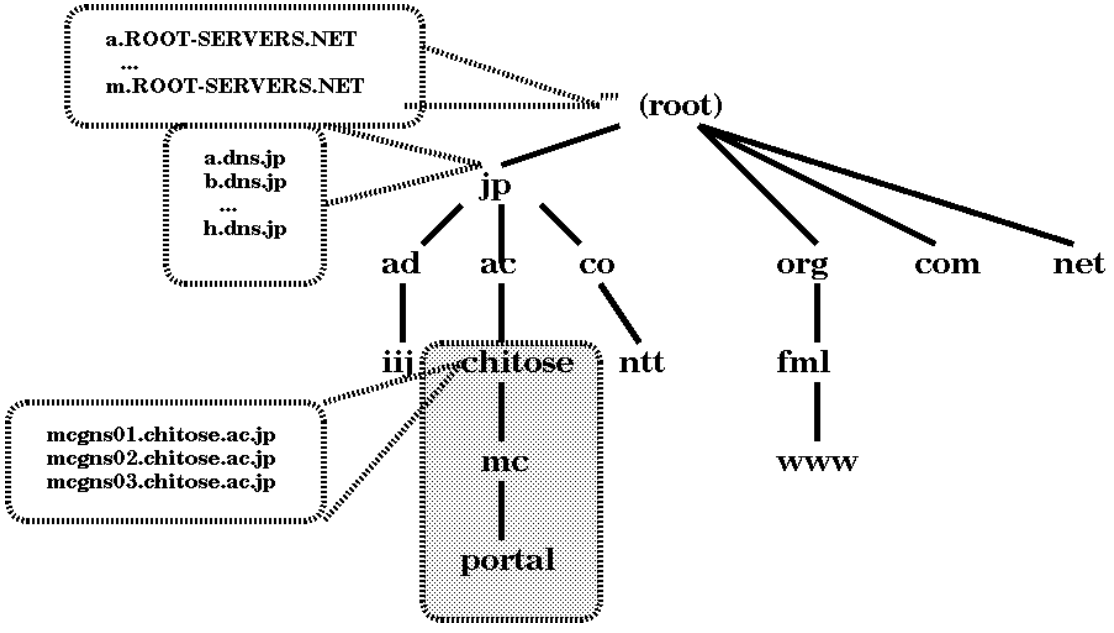


(脚注1) 各ノードにドメイン名が対応するのだから、 各ノードに各ドメイン担当のDNSサーバ群を配置するのが素直に感じられますが、サーバの台数を増やすのも管理が面倒なので、 ゾーンごとに2〜3台が普通

(脚注2) (a)JPRSのDNSサーバはjpとac.jp他 (b)大学のDNSサーバはchitose.ac.jpおよびmc.chitose.ac.jp を担当

ドメインの階層とクエリ

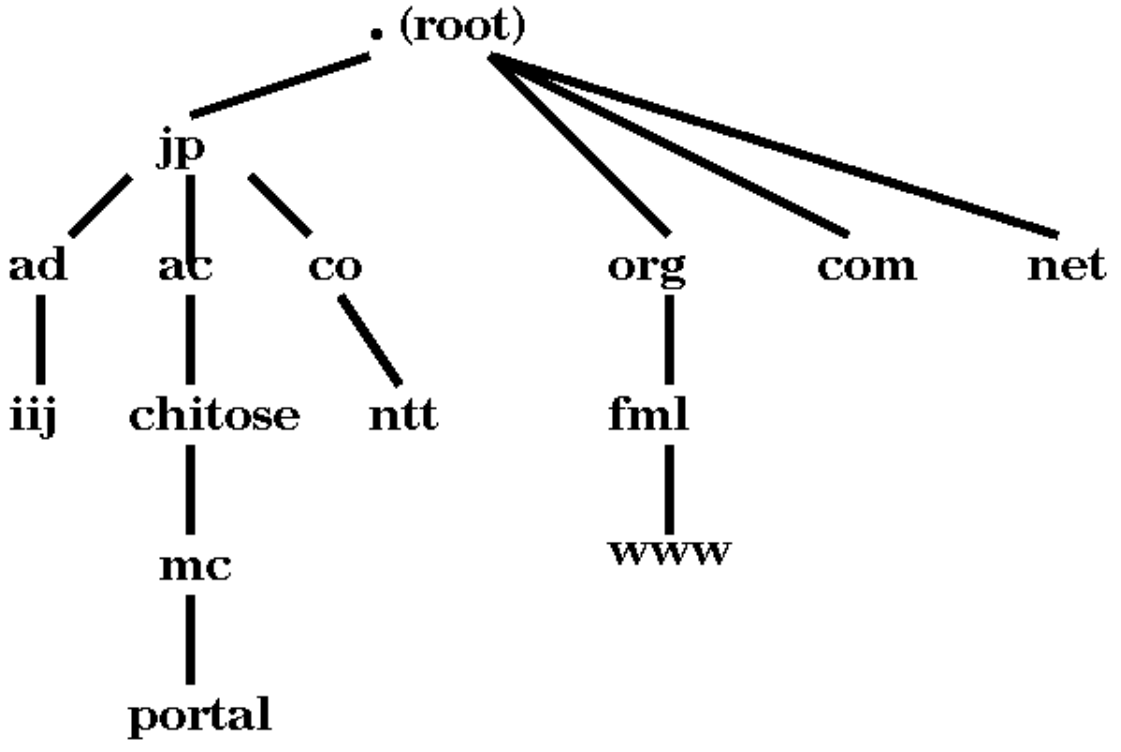
- DNSの問い合わせ動作が**クエリ(query)**
  - **正引き**: ドメイン名->IPアドレス
  - **逆引き**: IPアドレス->ドメイン名
- この問い合わせ(のデータ転送)は**UDP**
  - パケットが何度も往復するので**動作が軽いことは重要**です (詳細は Part II)
  - 一方、前頁で述べたように、 DNSサーバ(primaryとsecondary)間の設定の同期では**信頼性**が重要なので、 データ転送に**TCP**を使っています
  - この**TCPとUDPの使い分けに注目**
- クエリの際は木構造を一番上のルートから下にたどっていきます (詳細は Part II)



(脚注) (中間試験の範囲ではないので)具体的な動作を説明しないため、モヤモヤするとおもいますが、 クエリの動作の実際は Part II で実演するので、気になる人は、そちらも見ててください。

世界規模の分散システムとしてのDNS

- 木構造を階層的に多数のゾーンに分け、 各ゾーンが自分(および自分以下の階層)を管理することで、 インターネットの名前空間を一極集中ではなく**小さな管理単位**の集合体として再編成しています
- DNSは、世界中に分散した無数のDNSサーバ群からなる**分散システム**で、 世界規模で実に約40年動きつづけています



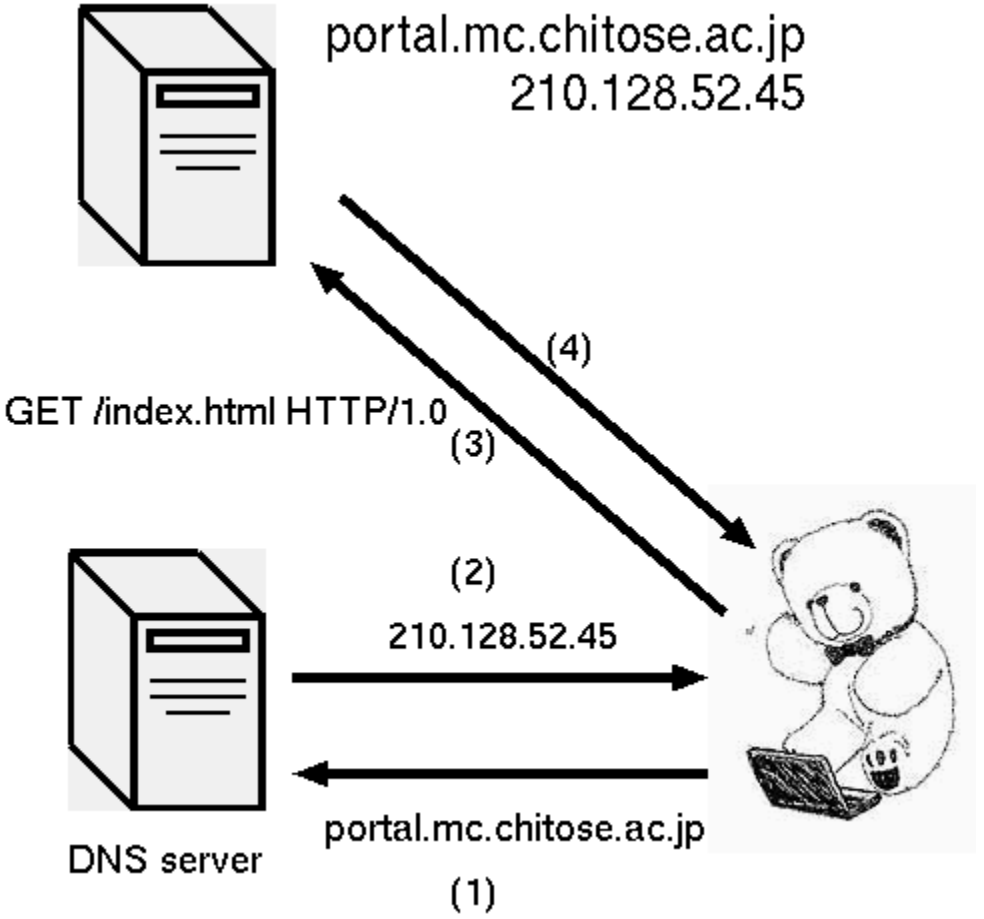
(脚注1) (インターネットのご先祖の時代の)初期にはファイル方式でしたが、破綻は見えていたので1980年代前半にDNSへ移行しました (脚注2) 非常に多くのDNSサーバの自律的な分散と協調を期待したシステムです。 DNS管理者たちのスキルと善意に依存とも言う;-) 自信がない人はDNSサービス(SaaS)を買うのが推奨です (脚注3) でも、プロを目指す人は試験に出ない範囲も勉強してほしい

DNS Part II 【参考】リゾルバの実際

余力がありそうな、やりますが、なんとなく無理そうな気がします。 この場合、試験範囲からは削除。
ちなみに、リゾルバのデモ動画を見ながら、おなじようにコマンドを叩くのが理解の早道だと思います。

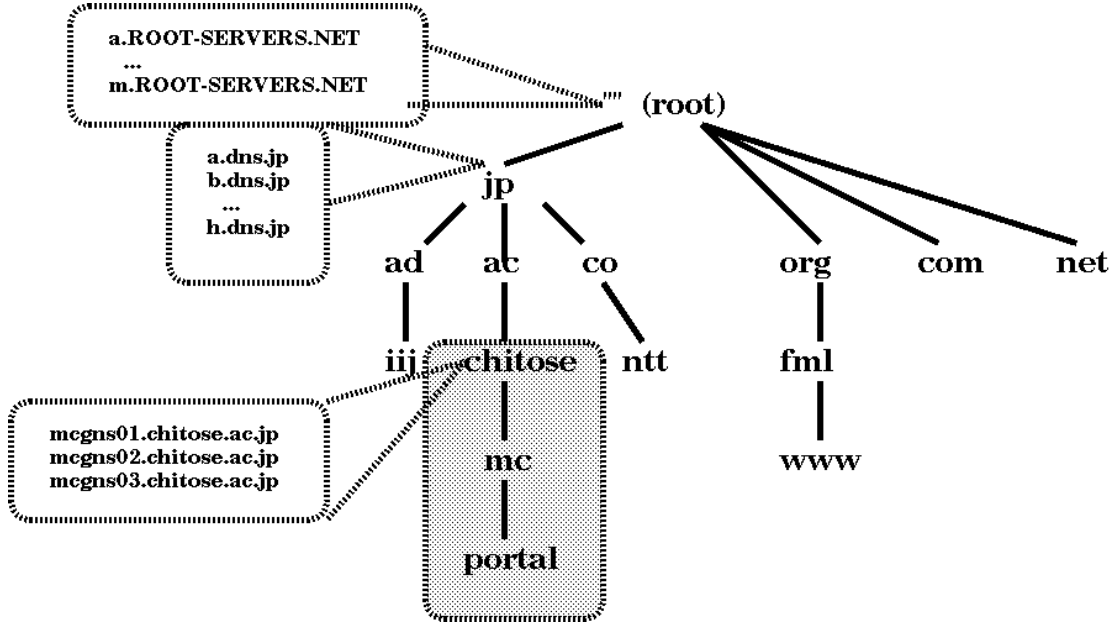
WWWの動作(DNS込み)（再掲）

- 前述の図をもう一度
- 右図の(1)(2)
  - (1) ブラウザはURIに含まれるサーバ名 portal.mc.chitose.ac.jpの IPアドレスをDNSサーバに問い合わせます
  - (2) DNSサーバはブラウザにIPアドレス =210.128.52.45を回答
  - (3)(4)はブラウザがHTTPでコンテンツを取りにいく様子
- この(1)(2)の裏側を解説します
  - 正確にはOSの「DNSクエリをおこなうライブラリ関数」 + (右図の)「DNSサーバとインターネット上のDNSサーバ群との間のやりとり」からなる



前図(2)の裏側, このあとのデモの概略

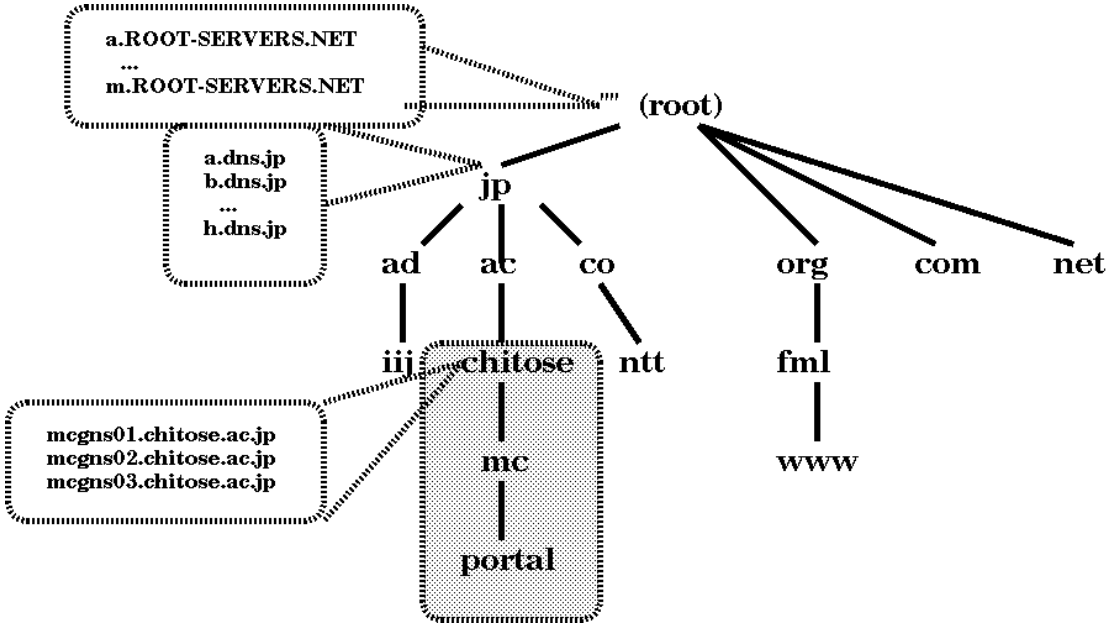
- 各ゾーンは直下のゾーンを管理するDNSサーバの情報を持っています
- portal.mc.chitose.ac.jp検索時(デモ参照)
  - ルートサーバへの問い合わせ
  - ルートサーバはjpゾーンのDNSサーバに問い合わせよと回答
  - jpのDNSサーバはchitose.ac.jpゾーンのDNSサーバについて知っており、 そちらに問い合わせよと回答
  - chitose.ac.jpはportalのIPアドレスを知っているので、アドレスを回答



(脚注) ルートサーバ群のどれに問い合わせるか?はDNSソフトウエアに依存するでしょう。 ただし、常識的に考えれば、次のどちらかのロジックではないかと思います (a)サーバ群に均等に問い合わせる(roundrobin) (b)とりあえず一番近いサーバ(日本ならm)を使うが、応答が遅い場合は他のサーバへ切り替え (random? roundrobin?)

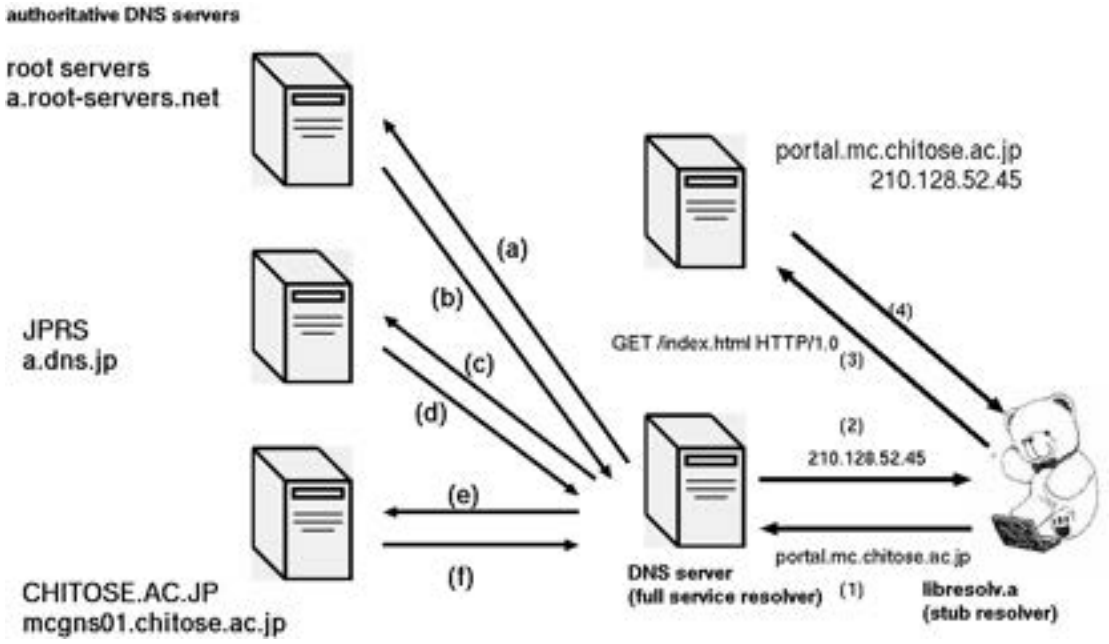
DNSの問い合わせと木構造(再掲)

- 前述のように、DNSの検索とは木構造を.(root)から下っていくことです
- 右図にはデモで登場する各ゾーンのDNSサーバ群が書き込まれています
  - 実際には、各ゾーンに複数のサーバがあり、全サーバが同じデータをもっているので、どのサーバに問い合わせても結果は同じです
  - (以下の説明およびデモでは名前順の先頭のサーバを使っています)



DNSの問い合わせ(1): 正引きの例

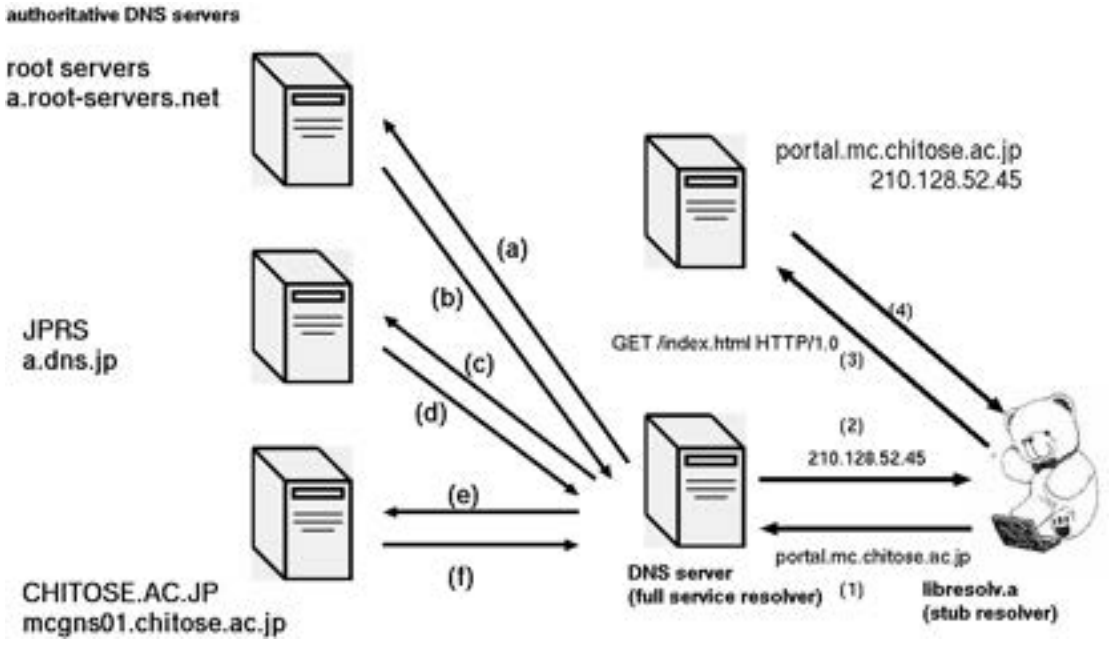
- 図(1)の裏側(a)〜(f)を解説します
  - 例: ドメインportal.mc.chitose.ac.jpのAレコードを問い合わせる
- クライアントのソフトウエア(例:WWWブラウザ)はリゾルバ関数を呼び出します (歴史的にはライブラリlibresolv.a)。これは通常stub resolverという役割をしていて、 一番身近な full service resolver へ(1)問い合わせを送り(2)返事を待ちます
- 実際に検索の仕事をするのは full service resolver です(右図中央下のサーバ)



(脚注1) (いまどきDHCPばかりで、OS設定経験者のほうが少数でしょうが) OSの「DNSサーバの設定」という項目で指定するべき情報がfull service resolverのIPアドレスです (脚注2) Googleの8.8.8.8などのPublic DNSもfull service resolverですが、 分散システムであることが重要なインターネットの理念に逆らっていると批判あり

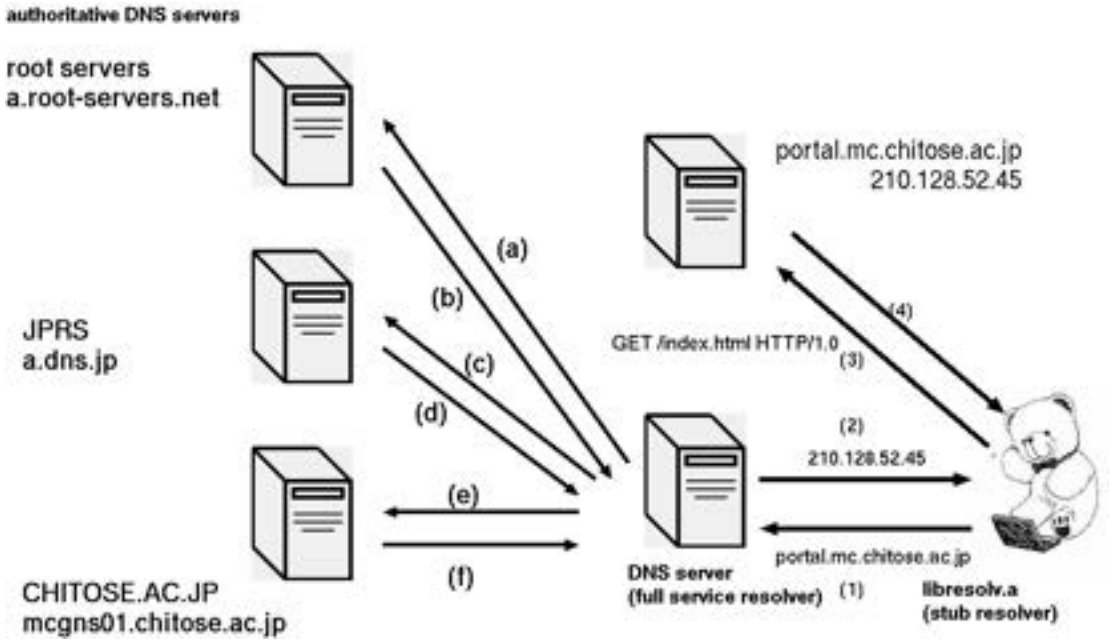
DNSの問い合わせ(a)(b): 正引きの例

- 出発点は木構造の一番上.(root)ゾーンへの問い合わせです
  - .(root)の情報をもつDNSサーバ群をルートサーバと呼んでいます。 これは世界各国に分散して設置している13台のサーバです
  - この情報は、出荷設定でOSに埋め込まれています (この情報が無いとDNSの検索が始められない)
- ここではルートサーバとしてA.ROOT-SERVERS.NET.を使うとします
- (a)ルートサーバに問い合わせを送ります
- (b)ルートサーバから「自分は知らない。 JPゾーンについてはJPRSのサーバ(a.dns.jp)に聞きなさい、IPアドレスは〜」と返事をもらいます



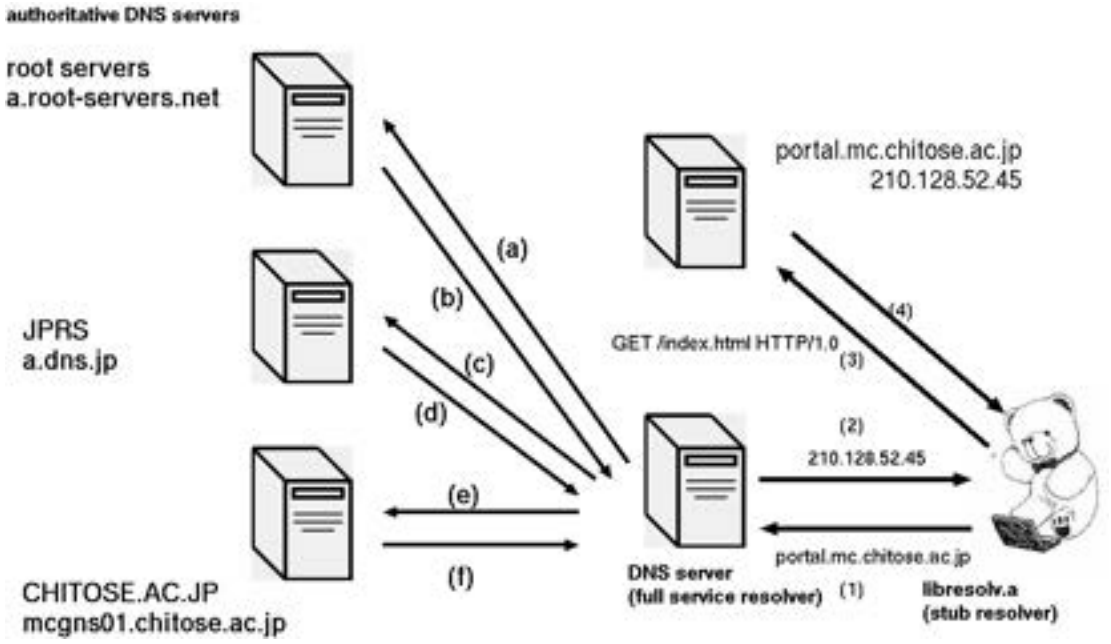
DNSの問い合わせ(c)(d): 正引きの例

- (c)再度a.dns.jpに問い合わせを行います
- (d)a.dns.jpからは「自分は知らない。CHITOSE.AC.JPゾーンについてはmcgn01.chitose.ac.jpに聞きなさい、IPアドレスは〜」と返事をもらいます
  - a.dns.jpはJPドメインとAC.JPドメイン両方の情報を管理しているので、(ノードの階層順に2回問い合わせそうですが、1回分は省略されて) いっきにCHITOSE.AC.JPゾーンのDNSサーバ情報を教えてもらえます



DNSの問い合わせ(e)(f): 正引きの例

- (e)再度mcgns01.chitose.ac.jpに問い合わせを行います
- (f)このサーバはmc.chitose.ac.jpゾーンも管理しているので、 いっきに答えまでたどりつき「portal.mc.chitose.ac.jpのIPアドレスは210.128.52.45です」 と返事をもらいます(前頁と同様の理屈)
- (2)full service resolverは、 問い合わせてきたクライアントに210.128.52.45という返事を返します
- (3)(4)このあとHTTPの処理がはじまります



デモ: ルートサーバ

デモで使うコマンドの説明

- DNSサーバに問い合わせをするdigコマンド

```
dig @DNSサーバ ドメイン名 [RR]

例: RR省略時は正引き
dig @210.128.52.11 portal.mc.chitose.ac.jp

例: メールサーバの問い合わせ(RRがMX)
dig @210.128.52.11 photon.chitose.ac.jp mx
```

- 類似のDNSを引くコマンドに**nslookup**と**host**があります
- プロは**dig**を好んで使っている雰囲気がありますね
- いずれのコマンドもBINDという最初期からあるDNSサーバソフトウエアの一部です
- **nslookup**は多くのOSでサポートされていて、 UnixでもWindowsでも、たいてい使えるはずです

デモ: 正引き

リソースレコード

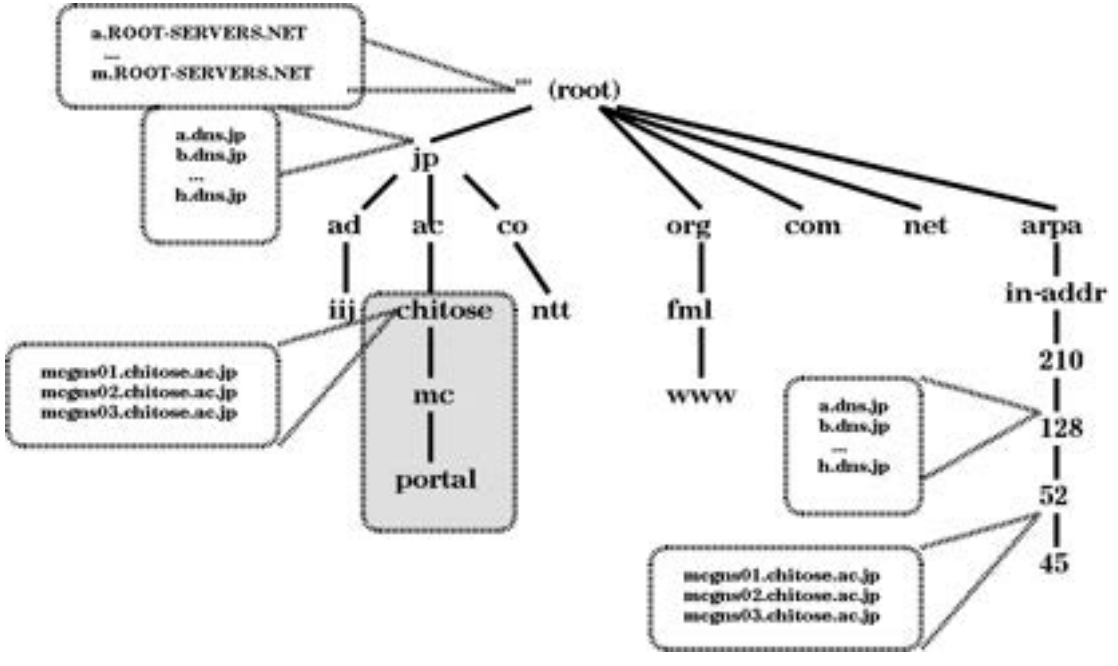
- ドメインに関連付けられた情報を**リソースレコード(RR)**と呼びます
  (略して**レコード**と呼ぶことも多いです)
- **正引き**とは**ドメイン名から該当するIPアドレスを取得する**ことで、 これは該当する**Aレコード**の検索になります。 サーバが答えを知らない場合たとえば「jpドメインはa.dns.jpに聞け」 と回答をしますが、 ここで答える情報が(jpドメインの)**NSレコード**
- **逆引き**とは**IPアドレスからドメイン名を探す**ことで**PTRレコード**の検索 (正引きの逆)

リソースレコード	値	備考
A	IPアドレス	Aレコードの検索が正引き
PTR	ドメイン名	PTRレコードの検索が逆引き
MX	メールサーバ	メールの送信先サーバの検索
NS	DNSサーバ	ゾーンを管理しているDNSサーバの情報
TXT	テキスト(任意のテキスト)	例: SPF

表:代表的なDNSレコード

逆引きの木構造とクエリ

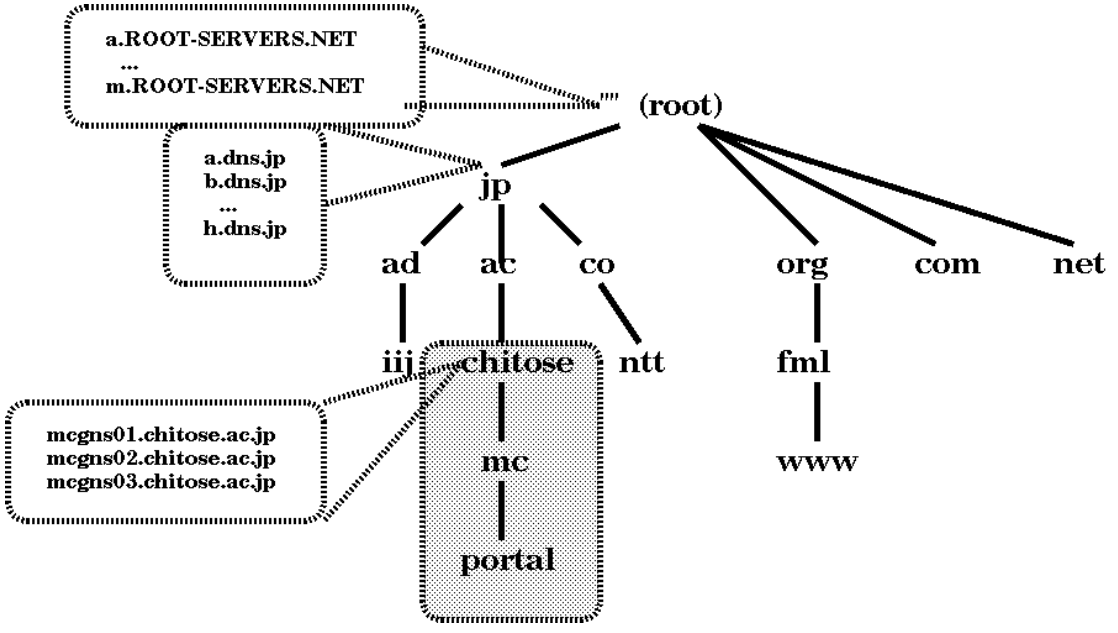
- arpaという仮想TLDを用意し木構造を作ります(図中一番右の縦1列)
- 正引きと同様に.(root)から下る検索
- メリットは、正引きと同じしくみがそのまま使えることです
  - 関数、木構造、設定ファイルなど
- デメリットは、DNS管理者が設定を書くときに頭を使うことだけです
  - 大学側で管理しているドメインは**52.128.210.in-addr.arpa.**です
  - 問い合わせるドメイン名は**45.52.128.210.in-addr.arpa.**になります
  - 一番右側の列を下から上に読むと、この文字列になりますよね?



デモ: 逆引き

MXレコードの検索

- 手順は正引きと同じで、問い合わせるものがMXレコードになるだけです
  - 例: photon.chitose.ac.jpのMXレコードを知りたい
- 問い合わせ先のドメインにMXレコードという情報があれば、それを返します
- photon.chitose.ac.jpの場合 photon-chitose-ac-jp.mail.eo.outlook.com. というサーバ名(Office365のサーバ名)が返されます
- メールを送信する側は、このサーバにメールをSMTPで送ります



デモ: MXレコード

UDP の意義, TCP の使いどころ

- 本節で見たように、 一回の問い合わせだけで、パケットが4往復(1)(2)(a)〜(f)します。 場合によっては、もっとたくさんの往復がありえます
- HTTP(TCPアプリケーションの代表例)とは、だいぶ挙動が違います
  - HTTPはWWWサーバと対向で様々なサイズのデータを返しうるのに対し
  - DNSでは相手が複数、短い問い合わせ(小サイズ)を何度も行います
  - DNSは処理速度も要求されます
  - DNSは静的で冗長性が高いシステムです。同じ情報を持ったDNSサーバ群が運用されており、どのサーバに問い合わせても大丈夫です
- DNSの問い合わせはUDPという転送方式をデフォルトで使います。 ただし不具合や大量の答えが返る際にはTCPへ移行することがあります
- 問い合わせに不具合がある際は、 再び問い合わせるなり、 タイムアウト後に別のDNSサーバへ問い合わせるなりは、 resolverが制御します
- こういった (TCPアプリとは挙動が異なる)処理を、 **プログラム作成者(user)がすべてコントロールするためUDPで作られている**と言えます
- (再掲)各ゾーンには複数のDNSサーバがあり、同じ**設定情報**を持ちますが、 確実な転送を行うため、 そのサーバ間の**コピーはTCP**です

まとめ

- Domain Name System (Service)
  - アプリにドメイン名名前空間の情報を取得する仕組みを提供します
    - 名前解決, リゾルバ
    - 正引き,逆引き,メールの送信先
  - 特徴
    - インターネットの基盤技術
    - 分散システムの大成功例
    - UDPベースの数少ない例
- ドメイン
  - .区切の文字列
  - FQDN (Fully Qualified Domain Name)
    - portal.mc.chitose.ac.jp
- 木構造をしたデータベース
  - 各自の管理(を委任された)部分がゾーンで、各ゾーンは複数のドメイン名を持ちえます
  - 各ノード(=ドメイン名)には複数のレコードが対応しえます。例:
    - IPアドレス(Aレコード), ドメイン名(PTR), メール転送先(MX), ドメイン管理情報(NS)
- 検索は、木構造を.(root)ノードからリーフへ下っていくことにあたります
- 転送方式
  - クエリはUDP、場合によってはTCP
  - サーバ間のコピーはTCP

DNS Part III 【参考】 ロードバランス,SPF

(脚注)【参考】パートは試験の範囲外です

DNSサーバの冗長化構成

- DNSサーバの冗長化は元々想定されているので、 単に複数台のDNSサーバを用意します
  - 1台がメイン(primary)です
  - それ以外のサーバ(secondary)は定期的にメインのサーバから情報をコピーします
  - DNSの情報更新は稀なので、一時間に一回確認するといったペースで十分です
- 問い合わせはクライアント側のfull service resolverの実装依存ですが
  - たいていは均等になるようNSレコードの順に聞いてくるはずです (roundrobin)
  - 例えば、a.dns.jpに尋ねたら、次の問い合わせはb.dns.jpといった具合に...

(脚注) メインのサーバを primary DNS server (name server)、 二台目以降のコピーをもつサーバを secondary DNS server (name server)と呼びます。 ちなみに何台あってもsecondaryです

DNSサーバのロードバランス

- クライアントはDNSサーバに均等に問い合わせるはずという性質を利用します
- 各DNSサーバに違う内容のレコードを用意することで、サービスを冗長化することが出来ます
- たとえば**portalサーバが二台(.45 と .46)ある場合**、 右表のようにDNSサーバを設定しておけば、 多数のユーザの問い合わせが、なんとなくDNSサーバ1と2に分散され、 portalサーバへのアクセスも分散されます (長い時間でみれば統計的に均等なアクセス、つまり**ロードバランス**が実現されます)

DNSサーバ	portalのAレコード(返事)
DNSサーバ1	portalのアドレスは210.128.52.45
DNSサーバ2	portalのアドレスは210.128.52.46

- ロードバランスの原始的なテクニック
- 設定ファイルをDNSサーバごとに個別管理しないといけないのが面倒ですが、
- それ以外にも、明らかな欠陥がありますけどね... (さて、それは何でしょう?:-)

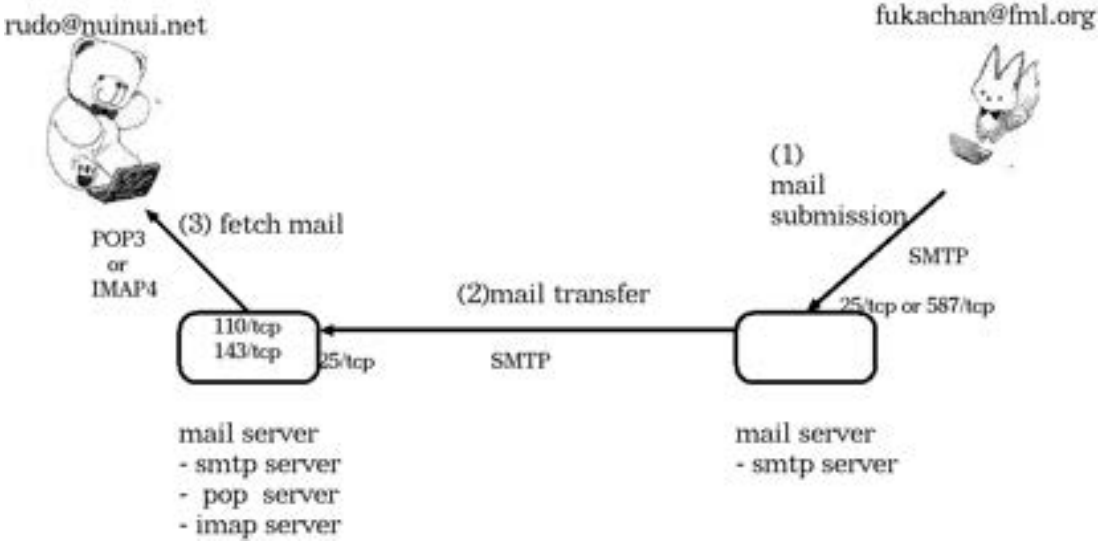
(脚注1) 特定のクライアントAがアクセスするサーバは、どちらか一つです。 サービス提供側からみた時に「多数のクライアントからのアクセスが分散して嬉しい」という意味です (脚注2) 秋学期OSの「高信頼化」を参照

メールの認証: SPF (TXTレコードの応用)

- メール送信サーバの正当性を検証する
  - メールで利用するドメインのTXTリソースレコードにSPF情報を書きます
  - SPFは**正しい送信サーバのIPアドレス**の情報です。 このサーバ以外から送られてきたらSPAMメールのはず
- 受信側メールサーバがSPFを調べ受け取りを判断します(図中(2)の受信部分)
  - SPAMの場合、 メールのドメイン名から調べたSPFのサーバと送信元サーバ情報が異なるので、 受信を拒否

[図右側のサーバがあるドメインに設定するSPFの例]

```
v=spf1 ip4:210.128.52.0/24 mx ptr ~all
```



(脚注) DKIMおよびDMARCは省略しますが、最低限SPFは設定したい。 DKIMは非対称鍵暗号ベースで送信者の正当性を確認する技術、DMARCはSPFとDKIMをさらに補強する技術

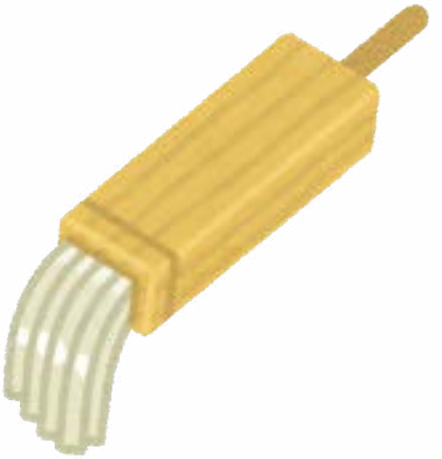
情報技術応用特論 第02回
TCP/IP(2) トランスポート層

Copyright (C) Ken'ichi Fukamachi <fukachan@fml.org>, 2021-2025. CC BY-NC-SA 4.0

TCP
(Transmission Control Protocol)

TCP: 信頼性のあるデータストリーム通信を提供します

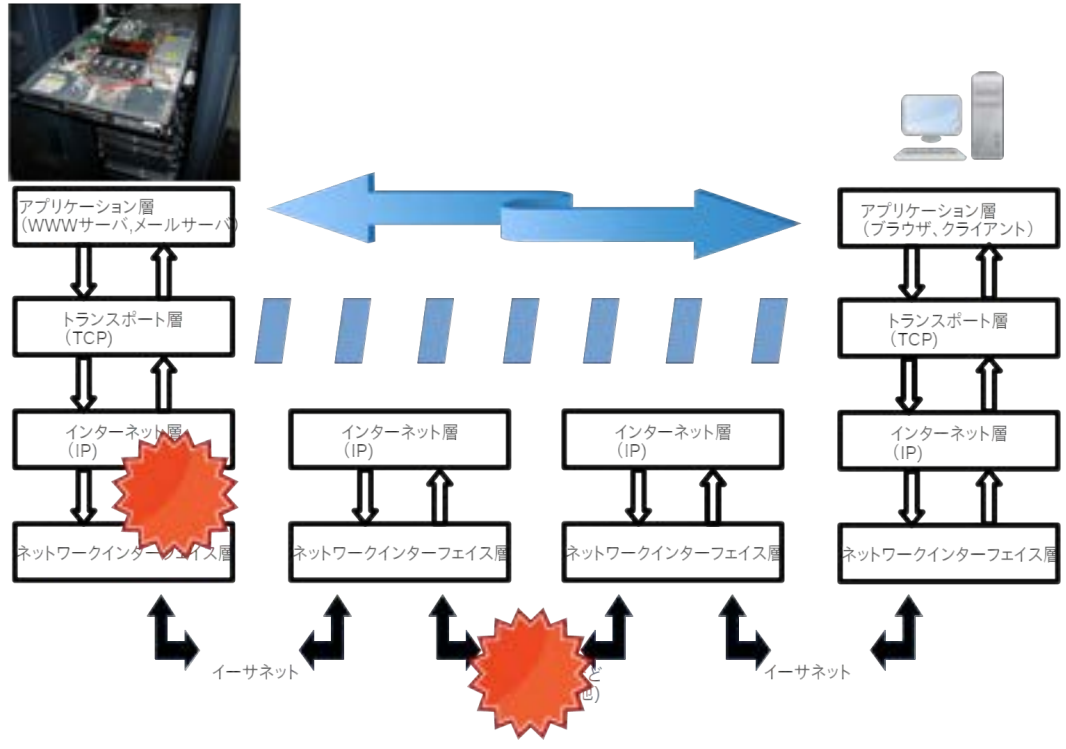
- **END-TO-END**で**信頼性のあるデータ転送の仕組み**を提供することがTCPの役割
- 入力された**データストリーム**はTCPによりパケット群に分割され、転送、 到着後に再構成される
- データストリームとは? ...
  特定の構造を仮定しないデータ(バイト列)
  ところてんもしくは**流しそうめん**のように、 入れたものがバイト単位で反対側から次々と出てくるイメージです (注意: TCPでは同じ通信路で返事も返ってくるため、 一方通行に聞こえる**流しそうめん**は少しよくない例えですが、 ストリーム(流れ)のイメージとしては良いでしょう)



(脚注1) 地球の反対側まで20,000kmあろうとも(速度はともかく)通信できます (脚注2) char buf[NBUF] (C言語)、 []byte (Go言語)

TCP: 信頼性

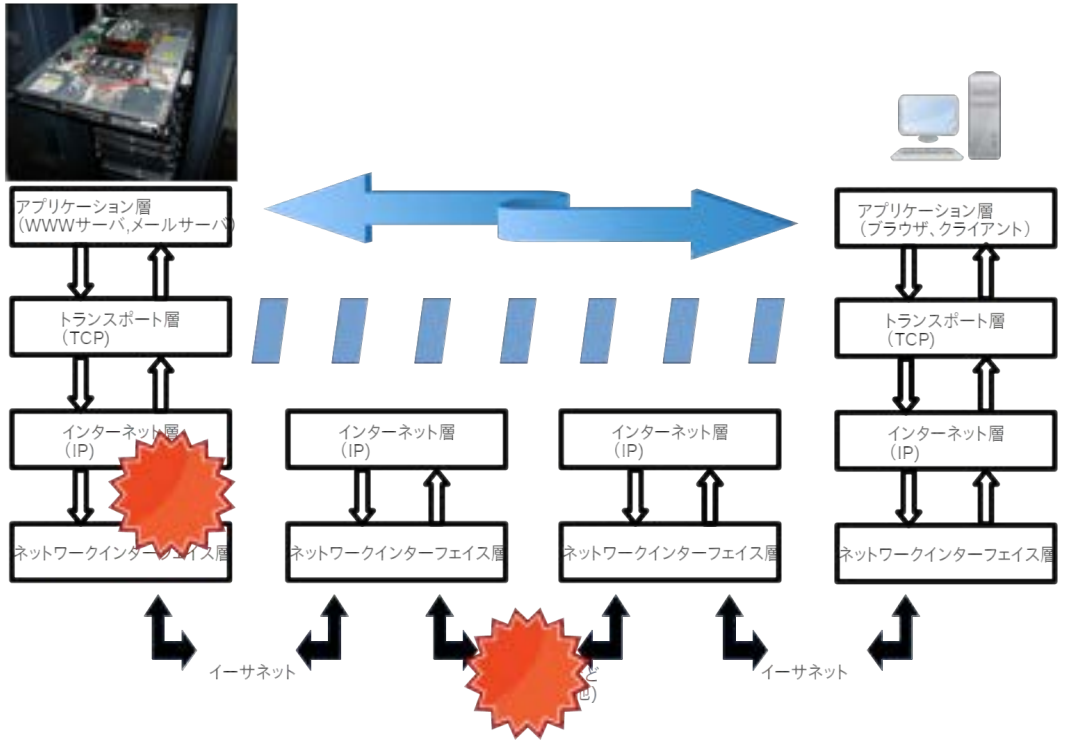
- **コネクション指向**の通信
  - 論理的な(仮想)通信路(**コネクション**)を構築し、その上でパケットを転送します。 よって、**コネクション指向**とも呼ばれています
  - 逆に後述のUDPは**コネクションレス**
- 通信状態をモニタしエラー訂正などはTCPが担当
  - TCPは**ステートフル**なプロトコルの例です 状態があり、それをモニタしています
  - **パケットの損傷、紛失、重複、順序の乱れを修復**し、 必要ならパケットを**再送**します
  - アプリケーションはエラーに気づきません



(脚注) 本当にいろいろな品質のネットワークがあるので、 複雑なリカバリができないと障害時や僻地との通信は難しい。 どんな品質でも接続し続けられることが本来の主目標と考えられます （ただ、これは現実とずれてきているような... -> HTTP/3節を参照）

TCP: フローコントロール(流量調節)

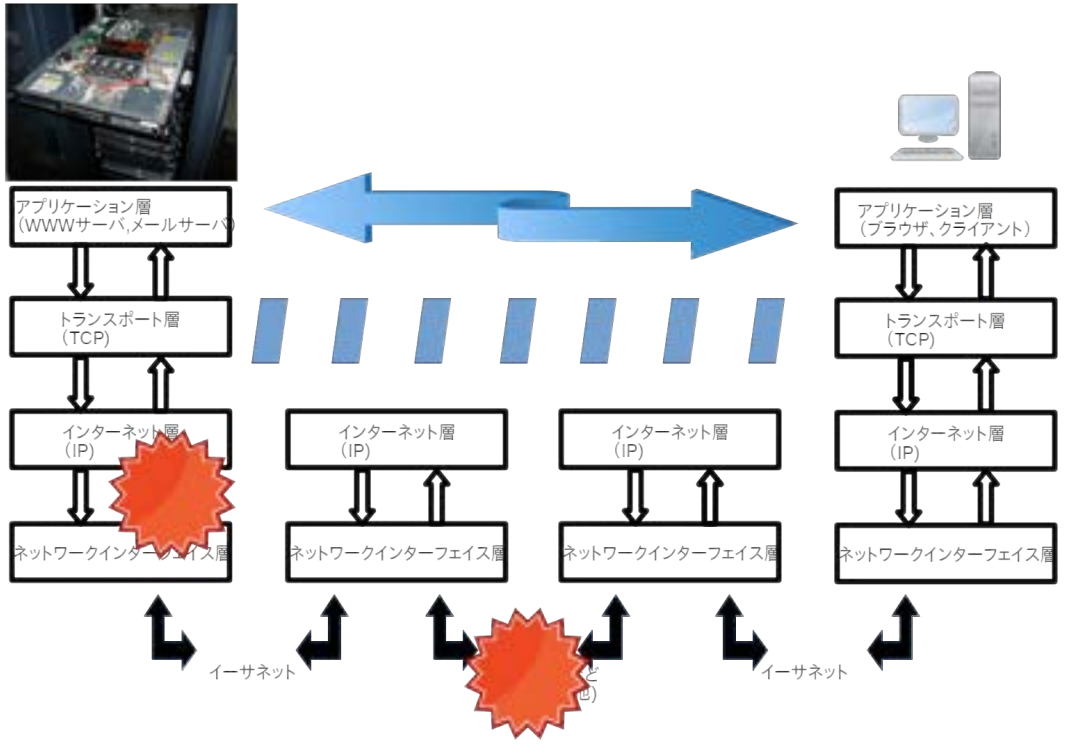
- 相手ホストの忙しさ(仕事量)の具合や回線品質によって送信頻度を自動的に調節します



(脚注) 送れそうなら、どんどん送ります(転送速度をあげます)

TCP: TCPとIPはコンビです

- TCPはIPとコンビで動作します
  - だからTCP/IPというセットの名称なのです
- **誰(アプリ)宛**に渡すのか?は**TCP(トランスポート層)が面倒を見ます**
- **IPは住所まで(だけ)**の面倒を見ます
  - IPは転送途中で何かあっても気にしません
  - きちんと**データが届いているか？はTCPが気にかけます。** そういう**役割分担**（層=役割）

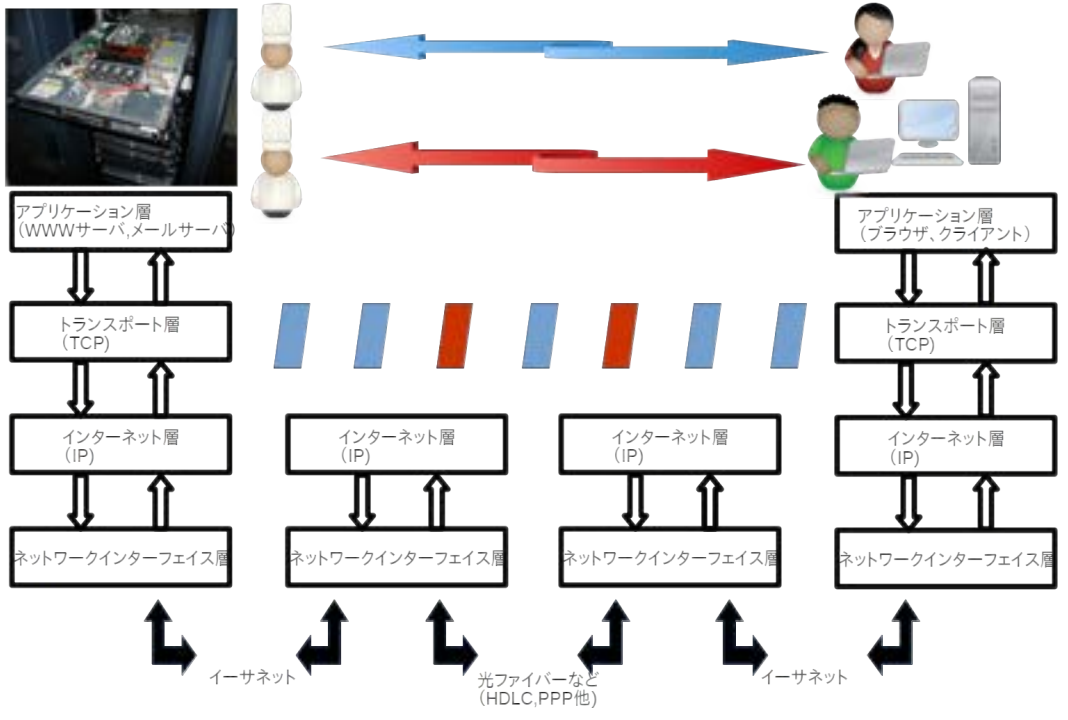


(脚注1)「パケットを落とす」という表現があります (脚注2) **IPは普通郵便**(その家の郵便受けには入れますが相手に届いているかは未確認)、 **TCPは書留**(相手に届いたことまで確認する)というイメージはどうでしょうね?

TCP: 多重化(multiplexity)

- ポート番号とIPアドレスの対で通信路を区別するので、 ホスト間で同時に複数のTCP通信路を利用可能

- **サーバのIPアドレス,サーバのポート(80/tcp),クライアントのIPアドレス,クライアントのポート(ランダム)**の4つの情報の組で識別可能



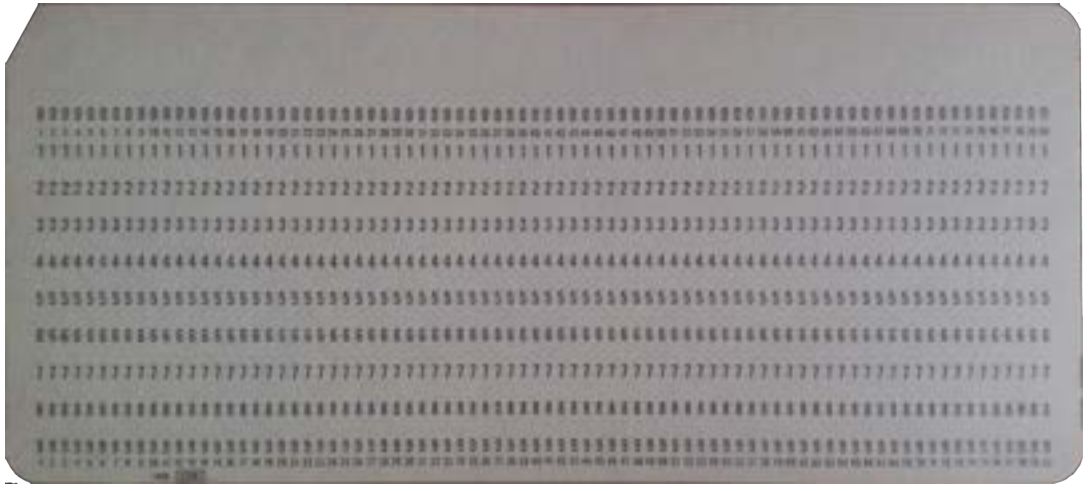
(脚注1) 一つのWWWサーバに、 あるPC上から複数のアプリケーション(クライアント)が接続している場合、 クライアントのポート番号以外は、すべて同じになります i.e. 図の青線と赤線の通信では、 (サーバのIPアドレス,サーバのポート,クライアントのIPアドレス)の3つ組は同じですが、 クライアントのポートだけ異なります

(脚注2) 一つの通信路の上に多重化することもmultiplexityと言いますが、これは違います。 「ホスト間通信の多重化」=「必要なだけ複数のコネクションを作成」です。 HTTP/2やHTTP/3はアプリケーションプロトコルがmultiplexityの機能を持ちます (-> スライド末)

データストリーム

- 入力したデータが反対側から**1オクテット**(8ビット=1バイト)ずつ出てきます
  - 改行などは特別なコードを決めてありエディタやシェルなどは理解できます (ファイル表現の取り決め)
  - 例: Unixファイルの改行(LF: line feed)
    012(8進数) **0A**(16進数)

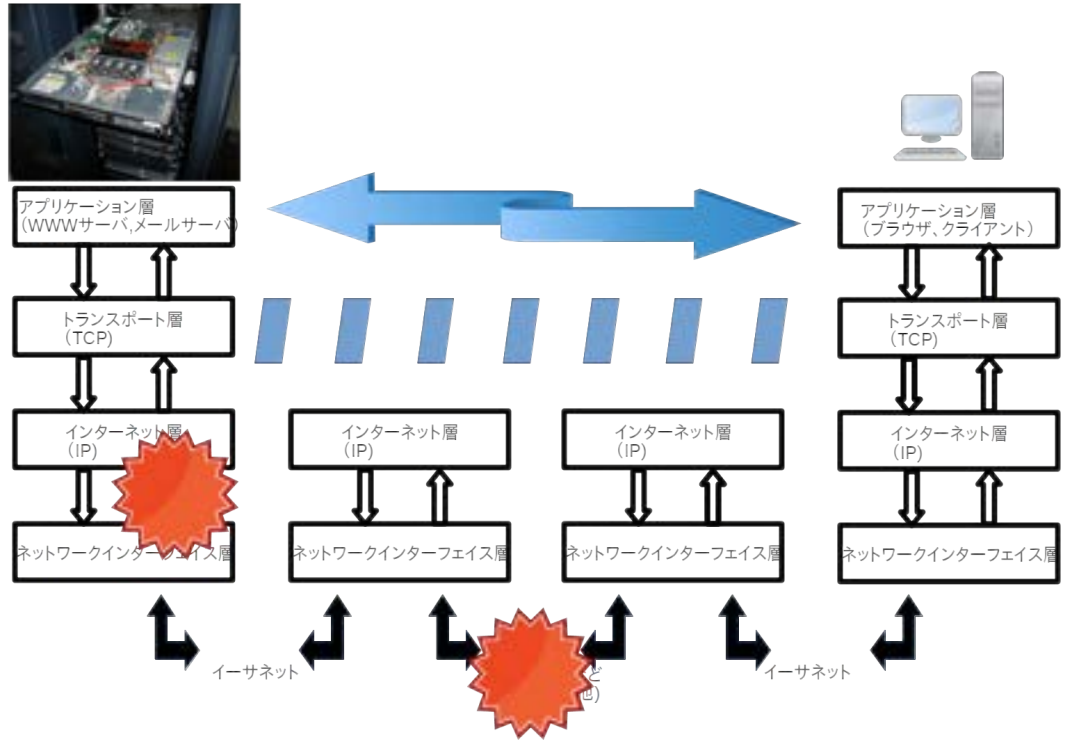
- 反対の例:通帳など、1行80文字固定(右図の下)



(脚注1) 1オクテットという表現があるのは、昔のコンピュータでは1バイトが8ビットとは限らないため (脚注2) 銀行通帳 ... 印刷を考えると便利なのかもしれませんが、データが無いところは空白?...非効率？ 大昔のpunchcardでprogrammingというやつですね。

パケットへの分割

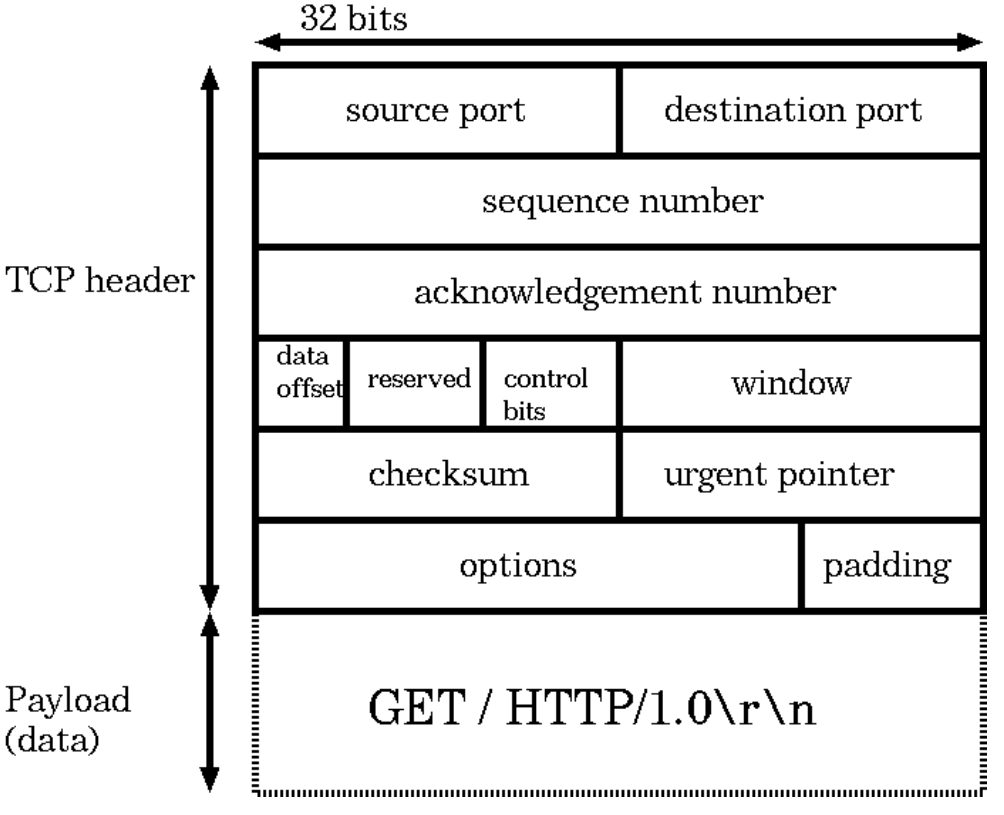
- アプリケーションは任意の長さのデータをTCPで送信/受信できます(**socket**への入出力)
  - 入力されたデータストリームは、TCPがパケット群に分割、転送、（配送先で）再構成します (右図の箱の列を参照)
  - TCPがデータをある長さ(**セグメント**サイズ)に分割して送信し、(受信側で)再構成します
- セグメントサイズはTCPより下の層次第、可変 (だいたい1400バイト前半が多いか?)



(脚注1) ソケットプログラミング ... ネットワークのプログラミングでは、 いつもの標準入出力(e.g. STDIN,STDOUT)と異なるsocketという入出力機構を使います。 BSD UnixがTCP/IPを実装したときに、この仕組みを作りました (脚注2) セグメントはTCPヘッダとIPヘッダ分を差引いたサイズになります。 例: 1500 - 20 - 20。 この1500というマジックナンバーは イーサネットのペイロードサイズ=1500バイト(IP,TCPヘッダ含)から来ています。 なお、一般家庭などのB-FletsはPPPoEなので1500より小さくなります(詳細は略)

TCPヘッダ

図中の表現	内容	長さ
source port	送信元のポート番号	16
destination port	送信先のポート番号	16
sequence number	シークエンス番号	32
acknowledgement number	アクノリッジ番号	32
data offset	ヘッダの長さ	4
reserved	予約(未使用)	6
control bits	制御命令/状態	6
window	流量制御	16
checksum	チェックサム	16
urgent pointer	緊急ポインタ	16



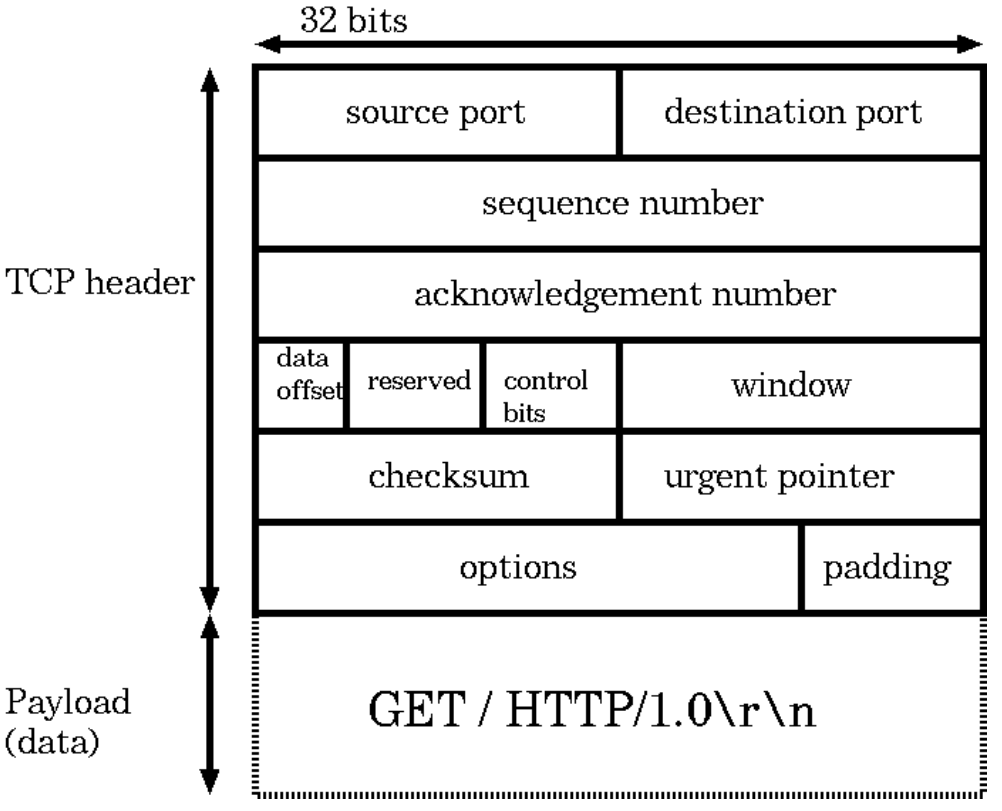
(脚注1)ヘッダ構造の詳細は覚えなくてOK

(脚注2)予約の3ビット分はECNへ;RFC3168(2001)

TCPヘッダ

- 信頼性、フローコントロール、多重化のための情報がヘッダにあります
- **ポート番号はありますがIPアドレスは無い**
  - TCPの担当は**アプリケーションの区別(誰?)**
    - ポート番号とTCPを制御する情報のみ
    - インターネット層がIPアドレス(住所)を扱うので、 IPヘッダの方に、IPアドレスを書きます

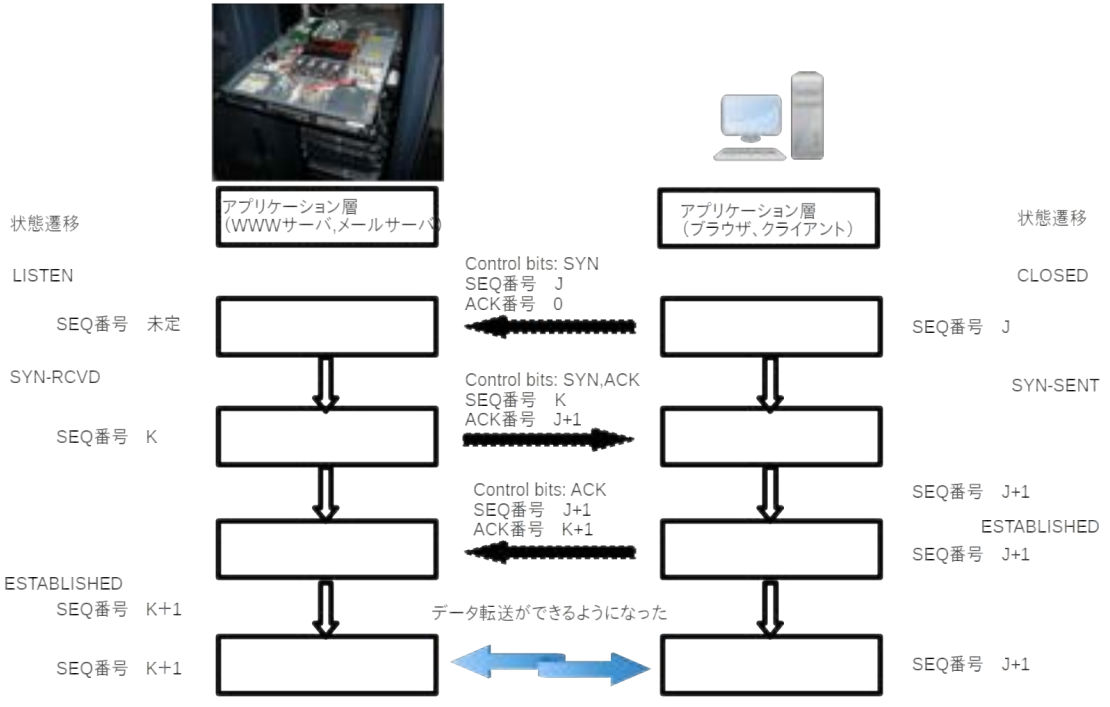
      TCPとIPはコンビで動作することを忘れずに!



(脚注) ヘッダ構造は覚えなくてOK; ヘッダにポート番号がありますよね?そこだけ覚えておいて欲しいですね

TCPヘッダとTCPの動作（概略）

- TCPはコネクションを確立するところから開始
  - 仮想通信路の生成 or コネクションの初期化とも言う
- この初期化プロセスが**3-way ハンドシェイク**
  - TCPヘッダの Control bits で命令(**SYN** = シンクロナイズ命令)を送り、 sequence number と acknowledge number をモニタして、 サーバとクライアントが互いに確認 (**ACK**, acknowledge が確認という意味)を取りながら行います (右図、詳細はこちら以降を参照)



(脚注) 詳細はスライド末に参考資料として収録してあります。 このあたりのTCP詳細は応用情報処理試験の範囲すら越えていますが、プロのエンジニアを目指す人にとってTCPの基本動作は必須要件なので各自で勉強してください (TCPの動作がわからないと基本的な障害対応すらできません)。 IPAの資格試験群は、ソフトウエア偏重なので、 TCPなんて分からなくてOKなどと勘違いしないように

(アーカイブ動画で見ている人も)一時停止して休憩

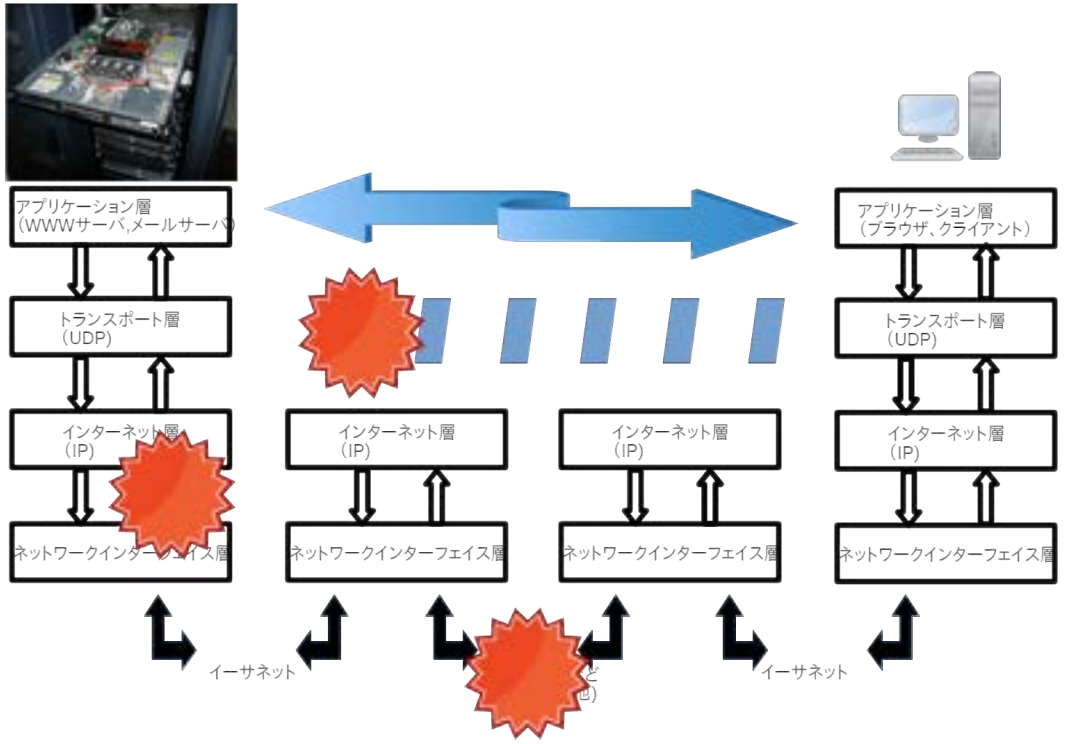
30分ごとに雑談（休憩）をしろというのが指導教官の教え

正確には「アメリカでは30分ごとにジョークを言わないといけない」だけれど、そんな洒落乙なこと無理:-)

UDP (User Datagram Protocol)

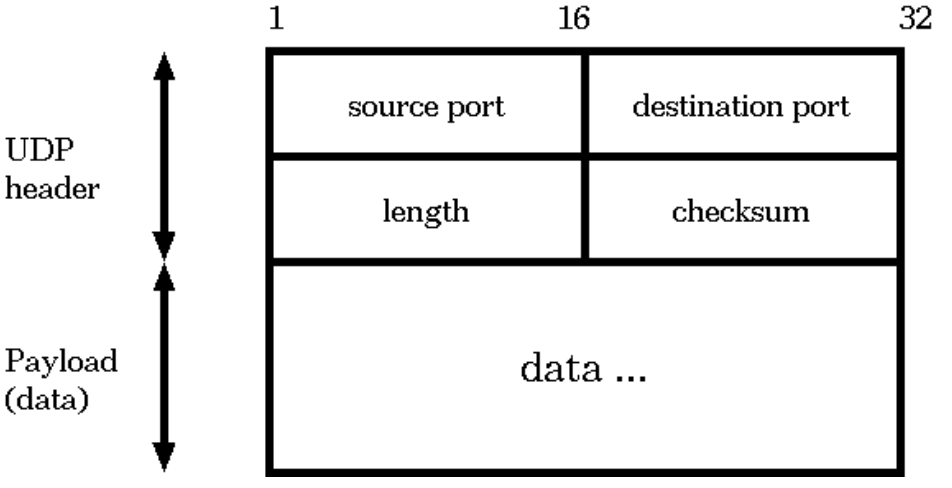
UDPの動作の概略

- TCPと異なり、最小限なので**処理が軽いです**
- 最低限の通信はできますが、 パケットの損傷、紛失、重複、順序の乱れなどについて何も配慮しません (-> アプリが実装する必要性)
- 右図は、 UDPやIP、イーサネットのレベルのあちこちでパケットが損失しているイメージ図を表現しています。 TCPであればTCPが回復させますが、**UDPでは単にパケットが届かない**だけです (たいていは、**エラーの連絡もありません**)



UDPヘッダ

- 最小限の情報のみで**処理が軽い**ため、 最低限の通信が出来るだけです
- （必要であれば）障害への対処は、すべて各ソフトウエアが自力で実装する必要があります
- そのかわり、 **プログラマつまりユーザがプロトコルをすべてコントロールできるのがメリット**



図中の表現	内容	長さ
source port	送信元のポート番号	16
destination port	送信先のポート番号	16
length	データの長さ	16
checksum	チェックサム	16

(脚注) HTTP/3節も参照のこと

UDPの例 (長距離)

- DNS (詳細は前回を参照)
  - 小さなデータを、多くのサーバと、多くのやりとりが発生するプロトコル
- IP電話や動画配信などの**音声,動画**系
  - **少々のパケット損失が問題にならない**メディア

(脚注1) これをTCPでやると、とても重そうというか初期化のオーバーヘッドだけで、 ずいぶん時間を取られてしまいます。 順調なら TCP 3-way handshake の時間で DNS は答えをもらっていそうです。 だからDNSでは軽い動作も求められていると思うんですよね〜
(脚注2) DNSクエリのデフォルトはUDPでも、サーバ間の設定ファイルの同期はTCPです。
(脚注3) 音声・動画系のアプリでは、最初の認証をTCP、 実際のデータ転送はUDPといったハイブリッドな作りにするのでは？ 商用が多いから、詳細はよくわかりませんけど...

UDPの例 (短距離)

- 視認範囲、1部屋などの規模感での話
  - この近距離なら、UDPでも障害おこらないだろうという淡い期待が持てる:)と言いますか...
- DHCP (Dynamic Host Configuration Protocol)
  - OSのネットワークまわりの自動設定
- TFTP (Trivial File Transfer Protocol)
  - ネットワーク機器の設定やバックアップ

(脚注) 組み込み機器などではsoftware(firmware)の格納領域が小さいため、 firmwareを大きくしないために、 TCPではなくUDPで転送するTFTPを使いたいという理由もあります

TCP vs UDP: 意義、比較

- TCP
  - TCPは仮想通信路を確立して転送を行いますが、 その確立する処理自体が重たい
  - TCPは何でもやってくれて便利ですけど…
  - TCPはOSの機能なので(プログラマが何かしたくとも)**細かな変更は不可能**です
- UDP
  - 軽い動作
  - **プログラマが変更可能なプロトコル**
    - (トランスポート層の上で)TCP似の機能を自由に取捨選択して設計可能という意味
    - 前回の**DNS**、このあと紹介する**QUIC(HTTP/3)**が好例

(脚注) TCPの場合OSがエラー処理など何でも面倒をみてくれて便利です。 アプリ屋さんはデータ転信の詳細など一切気にせずにアプリを作れます。 だから多くのアプリがTCPベースなわけですよ

HTTPそれから(HTTPS,HTTP/1.1)

HTTP/1.0 のメリット・デメリット

- メリット ... シンプル、分かりやすい、だからデバッグしやすい(のは良いのですが、やはり設計が古い...)
- デメリット
  - **平文**かつ冗長なテキスト
    - 盗聴すれば丸見えなので、個人情報やクレジットカードは流せない
    - インターネットショッピングは無理
  - **ステートレスプロトコル**
    - 昔ならindex.htmlを1つダウンロードで終了でしたが、 いまどき1ページで部品が何十もあるので、部品ごとにHTTP/1.0を実行します。 ただでさえTCP自体が重たいのに、それを何十回も...
    - 50個の部品を50ヶ所のサーバからダウンロードする場合、改良は無理です。でも、 そのページを構成する .html .css .js などを同じサーバからダウンロードするケースでは改良の余地あり
      -> HTTP/1.1、HTTP/2 を参照

(脚注1) 次頁以降、 HTTPSによるショッピング対策、 HTTP/1.1以降のstatelessへの対処について解説。 -> HTTP/2,/3 (参考資料)

(脚注2) GET ...などで分かるように（デバッグしやすい代わりに）冗長なので、このオーバヘッドが無駄なのですが、 WWWの出始めの時代は、 シンプルなページばかりだったので、このあたりは、まだ問題にされなかったのです（私見）

HTTPSと暗号化と認証(1994年末〜)

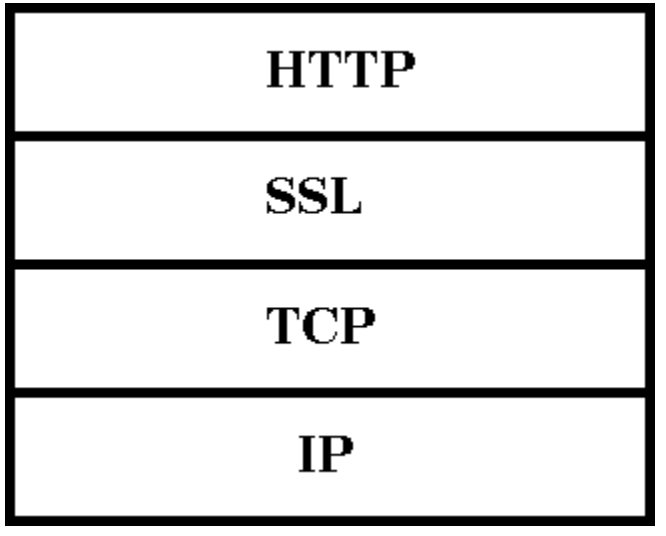
- インターネットショッピングに次の2項目は必須
  - **暗号** … 通信路上で個人情報やクレジットカード番号が盗聴されない対策
  - **認証** … このサイトは、まともな会社が運営していますという保証
- 例: 中間者攻撃(Man in the middle attack)
  - Web Spoofing (図)は1990年代後半すでに問題でした。図は「フェイクのサイトwww.amaz0n.co.jp経由でamazon.co.jpへアクセスさせる」様子。 真ん中のサーバ(amaz0n)でcrackerが盗聴し放題

(脚注1) かつては電子証明書ひとつ取得するのに10万円とかしていました。 これは「まともな会社」であることを確認するために、 所在地や実在性、登記簿の確認などをきちんとしていたからです。 これが「認証」です。 いまどきほとんどのサイトがLet's Encryptを利用して「暗号化」をしていますが、 これでは相手が「まとも」かどうかは確認できません

(脚注2) 経路上で盗聴できなくても、 サーバやクライアントをアタックすれば個人情報を漏洩させることができます。 近年は、これ(特にWWWブラウザ=クライアントを狙う方向性のアタック)が主流。 特にランサムウエアですよね〜

HTTPSと暗号化と認証(1994年末〜)

- HTTPSはHTTPとTCPの間にSSL層を追加したプロトコル(**階層モデル**なので簡単に出来ました)
  - 現在はSSLではなく**TLS**を挟みます
  - 今は亡きNetscape社が提案し Netscape navigator(Mozilla firefoxの御先祖)に実装しました
- HTTPSは暗号化と認証機能を提供
  - **暗号**...個人情報やクレジットカードの盗聴・漏洩阻止
  - **認証**...このサイトは、まともな会社が運営しています という保証 (別途、まともな**電子証明書**の手配が必要)



(脚注1) SSLはSecure Socket Layerの頭文字。 TLS(Transport Layer Security)はSSLをベースにIETFが規格化したものです。 基本的にSSLと一緒です。その後はIETFが規格を進化させてきたので今はTLSしか使いません。 現行バージョンはTLS 1.3

(脚注2) (前ページのくりかえしですが) 近年、Googleが暗号化しろと五月蝿いので、みんなHTTPS対応していますが、 たいてい暗号化をしているだけなので注意が必要です (マトモな認証なし=そのサイト運営側がマトモか?は未確認です) 参考: Let's Encrypt

HTTP/1.1 (RFC2068,1997/01)

- 毎回TCPを切断するのが重たい処理なので、 **TCPを使いまわせる**ようにしました
  - 一度サーバとTCPの通信路を確立した後、 その通信路の上でやりとりを続けられます
  - それでも間に合わないので、 たいていのブラウザは同時に6個のTCP通信路を作り並列処理をしています
- バーチャルドメインサポート
  - HTTPヘッダのHost:フィールドでサーバ名を渡せるようにしました。 **URL(ドメイン)ごとに異なるIPアドレスのサーバを用意するのは IPアドレスの無駄**だから
  - たとえば、このWWWサーバ(182.48.54.220)では複数のドメインを運用しています
    - https://www.fml.org/
    - https://technotes.fml.org/
    - https://lectures.fml.org/

(脚注1) HTTP/1.1がスタンダードでHTTP/2への移行も進んできています。 個人のクライアント環境ではHTTP/3がデフォルトかな

(脚注2) 表側は暗号化しても裏側は平文（HTTP/1.1）で運用してもいいよね？ （AWS用語で説明すると） ELBまでhttpsですが、ELBの裏つまりVPCの中はHTTP/1.1で運用する設計。 （VPC内には侵入されない大前提がありますが） このほうがオーバヘッドもかからないしね。 ゼロトラストセキュリティの時代なので末端まで全部httpsにしろという勢力もありますけど、 証明書の運用が面倒だし ...

(アーカイブ動画で見ている人も)一時停止して休憩

30分ごとに雑談（休憩）をしろというのが指導教官の教え

正確には「アメリカでは30分ごとにジョークを言わないといけない」だけれど、そんな洒落乙なこと無理:-)

参考資料

試験には出ません; TCPなんて**テオクレ**だよとAmazonとGoogleに言われました(;_;)

時代はTCPからUDPベースへ(HTTP/2, HTTP/3)

- 元々のTCPでは(1)END-TO-ENDでの確実なデータ転送が主目標で (2)転送速度の問題は次点
  - それでも、最適化をくりかえして、Gbpsくらいは出せますけれど
  - いま、データセンターの機材は25Gbps,40Gbps,100Gbpsの世界です
  - なにしろ半世紀前の設計なので、いろいろ古いといえば古いのです
- 信頼できる自社ネットワーク内であれば TCPはオーバースペック(or 設計が古いので性能が出せないの)ではないのか? と思われている昨今

- AWS (Amazon Web Service)の裏側はAmazon独自プロトコルだそうです
- GoogleはChromeブラウザの改良を続行中
  - ブラウザでGoogleを使ってもらうところが収入源だから当然ですよね?
  - Chromeを独自改良して、Chrome〜Googleサーバ間の高速化を探求・評価、 その実装をIETFに提案
    - **HTTP/2**はGoogleの**SPDY**が大元
    - **HTTP/3**はGoogleの**QUIC**が大元
      ついに**QUICはUDPベース**へ

(脚注1) TCP/IPは1970年代半ばに設計開始 (脚注2) GAFAM等がIETFに貢献する一方、 草の根の力が弱くなってきた傾向が懸念される昨今

HTTP/2(RFC7540,2015/05) SPDYベース

- Googleが考えたSPDY(2010年代前半〜)というプロトコルが大元です
- **latency(待ち時間)の短縮**が主目標
- テキストではなくバイナリ形式です
  - 命令は1ビットでも伝えられます
  - テキスト(e.g. GET ...)は冗長
- HTTPヘッダの圧縮
  - HTTPヘッダのほとんどは毎回同じなので、重複排除や圧縮により通信量を削減可
- サーバプッシュ
  - サーバからクライアントへデータを送り込むことができます
    例: レンダリングを早く開始できるようにcssを優先的に送りこむ
- フローコントロール ... HTTP側にもTCPのような機能が搭載されています
- **TCP上でHTTPという前提は継続**
- HTTP/1.1の上位互換でURIはそのまま
- 事実上、HTTP/2では暗号化(TLS)が前提
  - ちなみにFirefoxとChromeはhttps(HTTP over TLS)のみをサポートすると明言しました
- 参考文献
  - HTTP/2 explained
    - cURLという有名なデータ転送ソフトウエアの作者 Daniel StenbergによるHTTP/2の解説です

(脚注)【歴史的展望】 ちなみにiphone 3Gの発売が2008夏です。 プロトコルの設計・開発は10年くらいかかって普通でしょうから、SPDYは、こんなにスマートフォンが当たり前になった世界を想定していなかったのだと思います （私見）

HTTP/3(近々リリース) QUICベース

- **できれば新しいトランスポートプロトコルを作成して差し替えたいのが本音**ですが、 **運用的にはTCPかUDPしか選択肢がないという現実**
  - 昔とちがい、ずいぶんネットワークの品質は向上！
  - TCPを前提にしなくても良い？
  - END-TO-ENDで通信可なプロトコルはTCPとUDPだけ
  - なぜなら、 **危ない**ので、 **ルータやファイアウォールで謎プロトコルは通さないことが多いから。** これらは長年インターネットを運用してきた結果/しがらみ

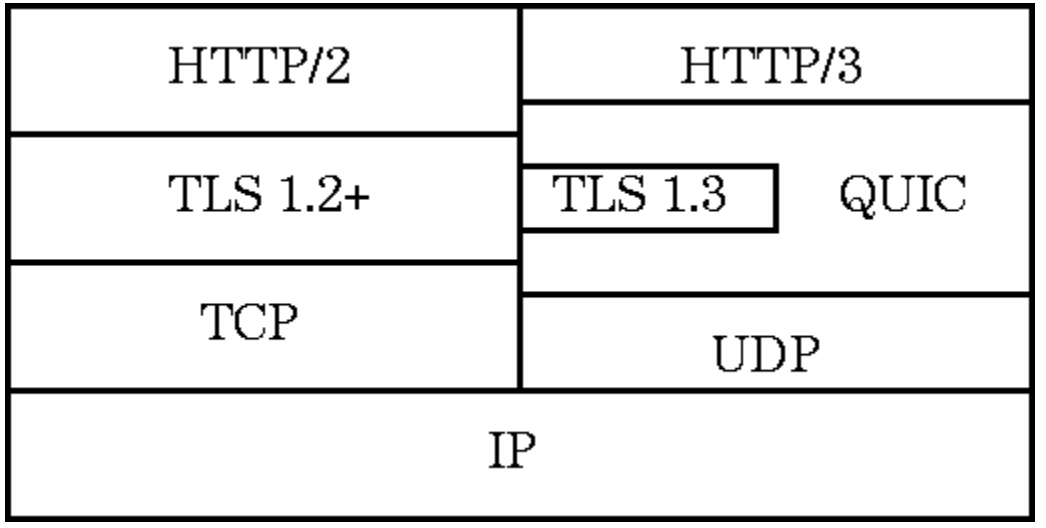
- Googleは**UDP上に(TCP似の機能をもった)独自のプロトコル(QUIC)を作り**、 QUICの上でHTTPを提供することにしました
  - GoogleはHTTP over QUICをChromeに実装し、Googleのサーバで評価後、IETFへ提案、2021年5月、RFC 9000 として標準化

- 参考文献
  - HTTP/3 explained
    - cURLという有名なデータ転送ソフトウエアの作者 Daniel StenbergによるHTTP/3の解説です

(脚注)【歴史的展望】 スマートフォンが猛烈に普及して問題を感じ始めたからQUICの開発が始まったと考えられます（私見）

HTTP/3 (QUICベース)の階層構造

- HTTP over QUIC
  - **UDPの上に独自の(TCPのような)QUICプロトコルを実装**しています
  - UDPの転送さえ何とかなれば、 その上で独自プロトコルによるHTTPの機能強化が実現できるところが重要です
- **暗号化(TLS 1.3)は必須**です
  - 2010年代半ばあたりからGoogleはHTTPS必須だと言っていますね...
- 諸問題や計測値は、この卒論を見てください
  - HTTP/3 がシステム運用に与える影響の評価



(脚注1) ふつうfirewallでUDPを通していないのでfirewallの設定変更をしないとHTTP/3が使えません

(脚注2) 大学ではHTTP/3が通りませんが、 通常、 自宅のルータは制限をかけていないので、 自宅ではHTTP/3が使われているはずです

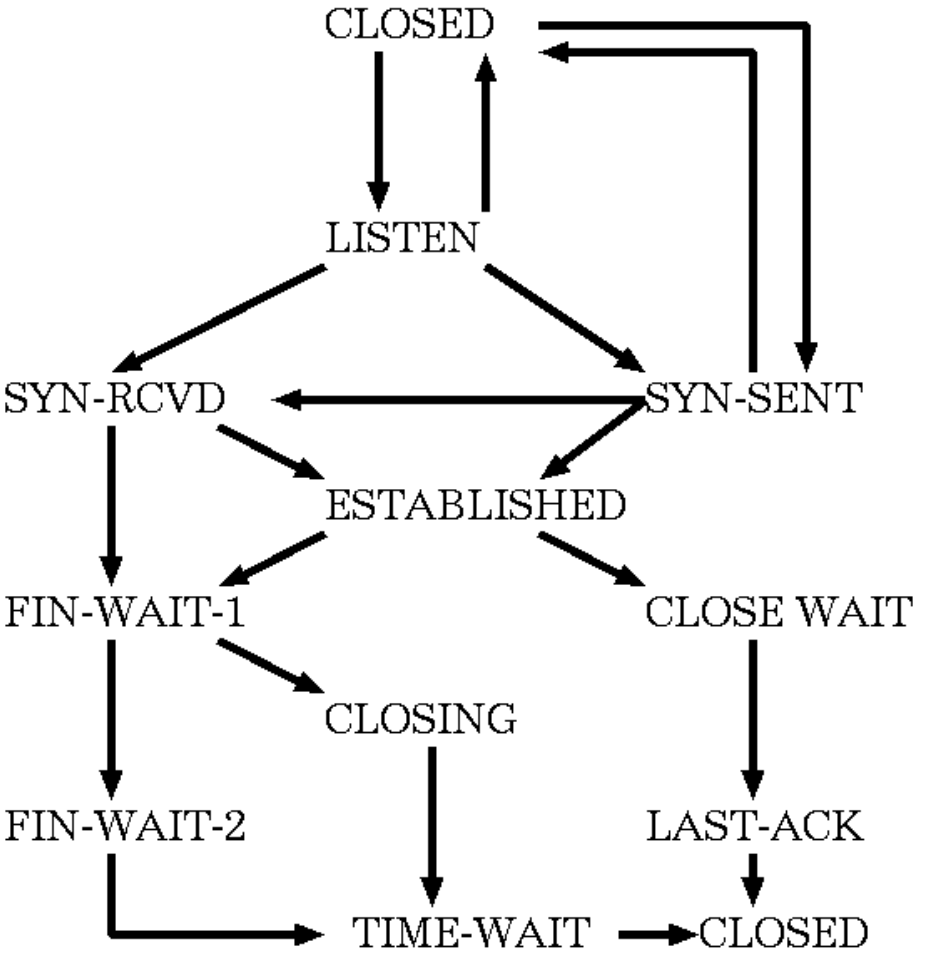
(自宅ではGoogleサーバとの通信が大学のときより一割くらい速く感じませんか?)

TCP (Transmission Control Protocol)
より細かな動作

応用情報処理試験でも出題されないレベルですが、 プロになるなら分かっておいてほしい

TCP: 状態遷移図

- 状態遷移(state transition)
  - TCPの状態が移り変わっていく様子
- CLOSEDから始まり、いくつかの状態を経て再びCLOSEDになります
  - 一番上のCLOSEDと右下のCLOSEDは同じものです (図は、右下と上がつながっていると読んでほしいのです)
- ESTABLISHED状態が実際のデータ転送を行っている一番大事な状態にあたります
  - 長くて言いにくいので業界人は、この状態をESTABなどと省略します
  - 「ACKビットが1」の状態がESTABです(「ACKが立っている」とも言います,後述)



(脚注) 日本の業界人はESTAB を「えすたぶ」と発音しています。外人に通じるのかは不明ですけど

TCP: コントロールビット

- TCPヘッダのコントロールビット部(control bits)
  - 命令および現在の状態を伝えるもの
  - 1bitずつ違う意味になっています
  - 本来このように1bitで十分なので、(debugしやすい)アプリケーションプロトコル群が冗長なのですね

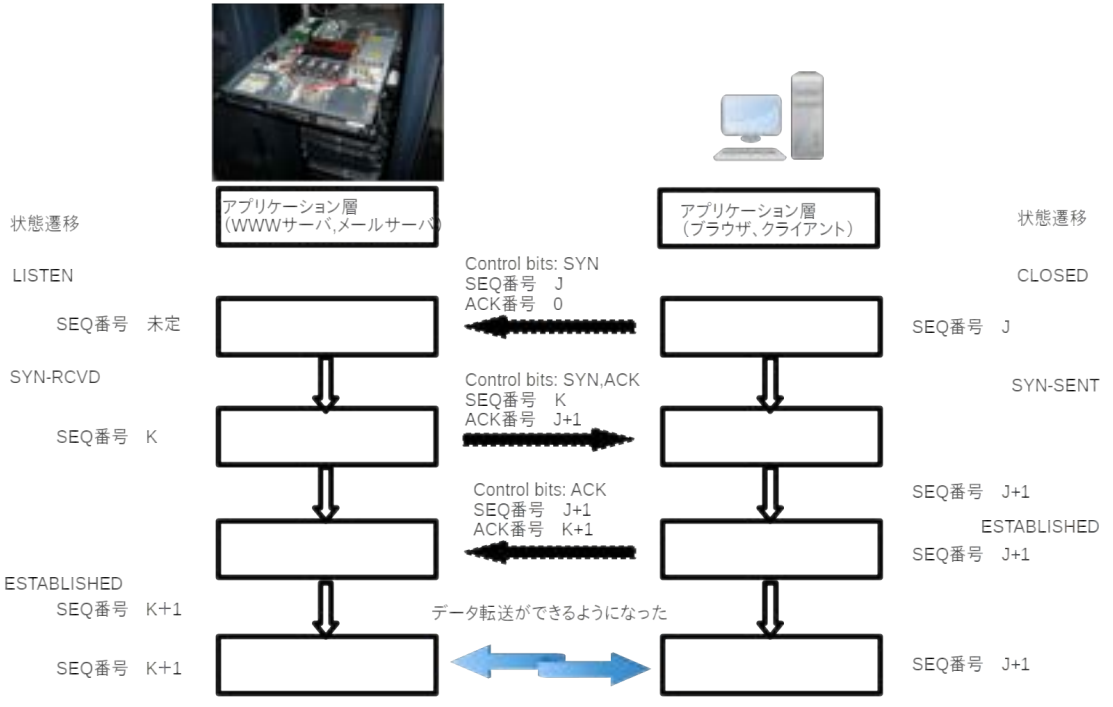
0	1	2	3	4	5
U	A	P	R	S	F
URG	ACK	PSH	RST	SYN	FIN

(脚注)ECN(予約の3ビット分;RFC3168(2001))は含めていないoriginal

コントロールビット

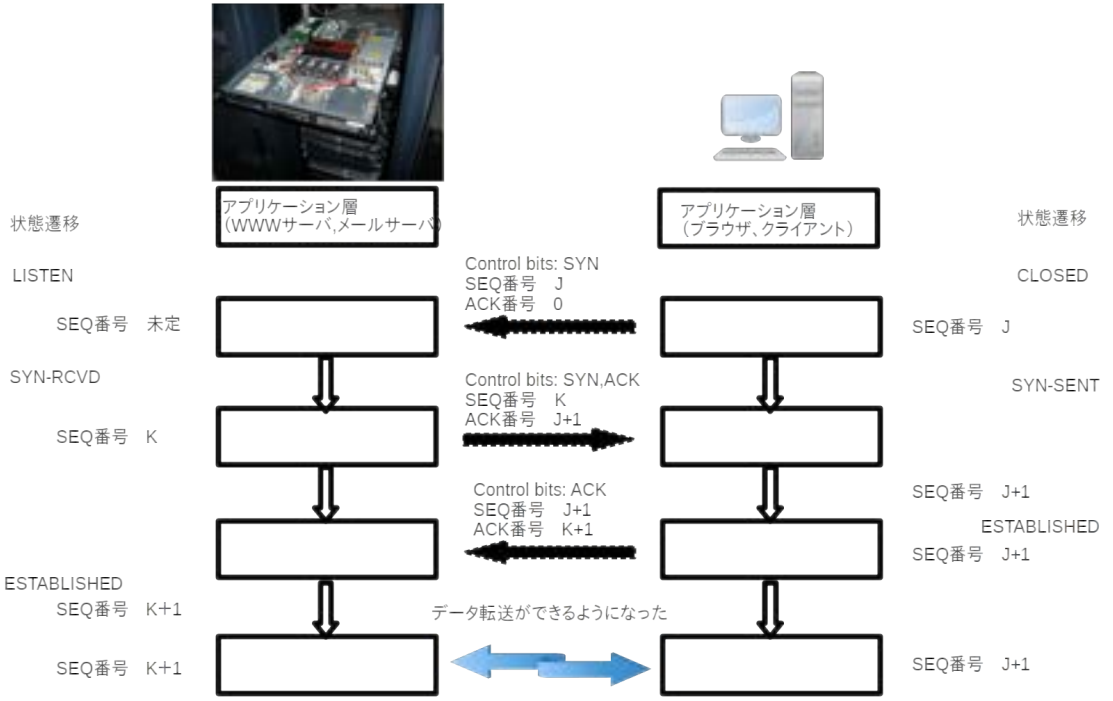
TCP: 通信路の確立 1

- 3-way handshake
  - 有名な3-way handshakeです
- (論理的な)**仮想通信路**を確立します
  - この通信路を**コネクション**と呼んでいます。
    TCPは**コネクション指向**と言われています。
- コントロールビット**SYN**が確立のお願いで、 **ACK**が了解したという意味です
- これを双方向に行い、 双方がACKを返せば**ESTABLISHED**状態になり、 データ転送が行えるようになります
- サーバとクライアント双方で、 それぞれのシーケンス番号(図中ではSEQ番号)を管理しています。 これは送ったデータのバイト数分増加する数字です



TCP: 通信路の確立 2

1. クライアント(C)からサーバ(S)へ
   - コントロールビットの**SYN**部分を1
   - **SEQ番号J**でサーバへ送信します。ACK番号は未定なので通常0
2. サーバが返事を返します
   - Cのお願いに了解の意味で**ACK**を1
   - **ACK番号**にJ+1を設定
   - SからCへのお願いの意味で**SYN**を1
   - **SEQ番号**にSの番号Kを設定
3. クライアントが返事を返します
   - SからCへのSYNに了解で**ACK**を1
   - **SEQ番号**にCの番号J+1を設定
   - **ACK番号**にK+1を設定

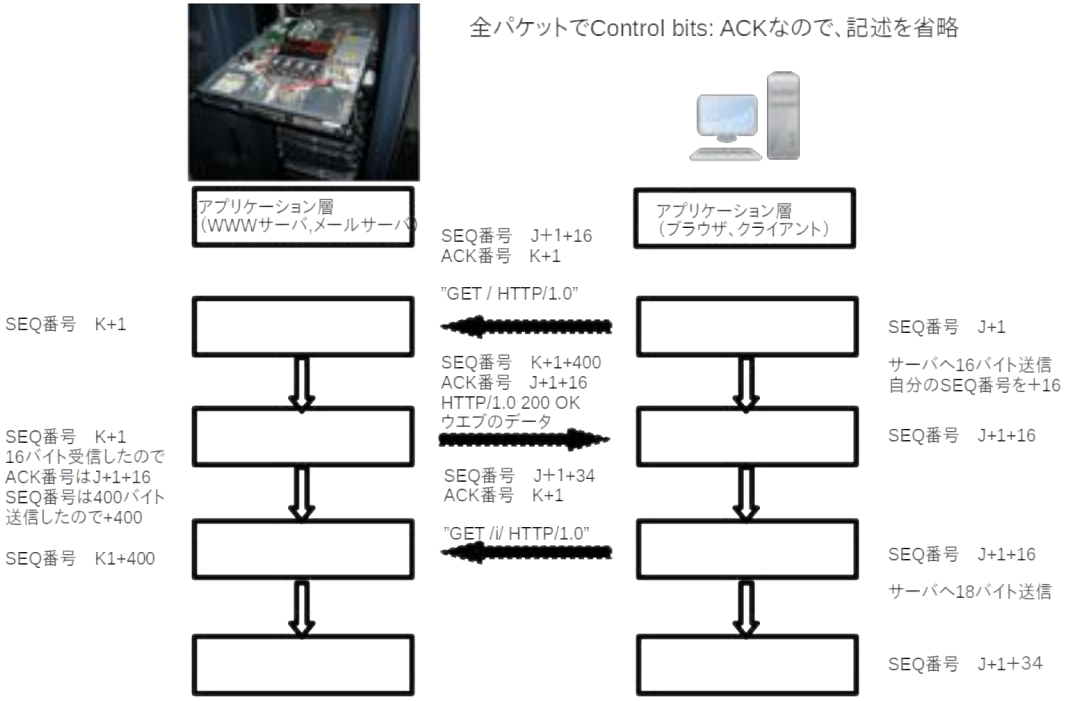


(脚注1) 1に設定することを「ビットを立てる」と表現します

(脚注2) 3-way handshakeではデータ転送をしませんが、ACKを返すときにSEQ/ACK番号を+1します

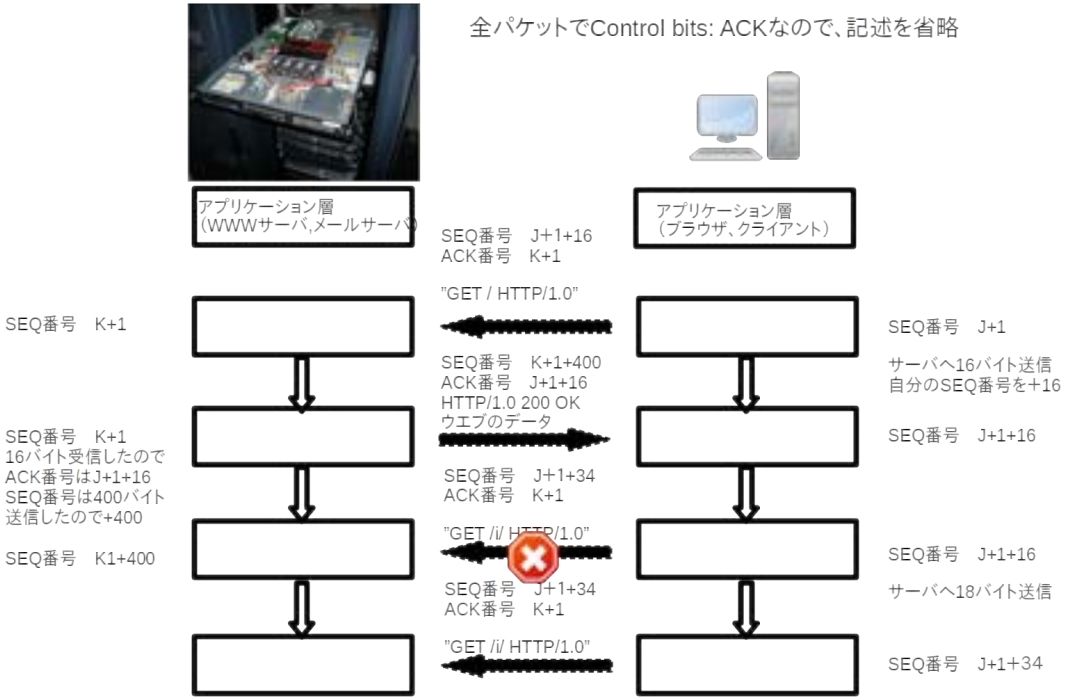
TCP: 転送

- 右図はHTTPを例にしています
- クライアント(C)からサーバ(S)へ`GET / HTTP/1.0`を送信します。 ここではペイロード(データ)サイズが**16バイト**と想定
- C、Sそれぞれで送ったデータ量分SEQ番号は増えます。 変化後の値は**相手から受領確認で返ってくるACK番号と一致**するはずです
- C側の**SEQ番号**は**(J+1)に+16**されます (Sが無事に受け取ったら**J+1+16**の**ACK**を返してくるはず)
- SはCからの16バイトを受信したので**ACK番号**に**J+1+16**を設定し、 SEQ番号にはSからCに送る400バイトを加算した**K+1+400**を設定



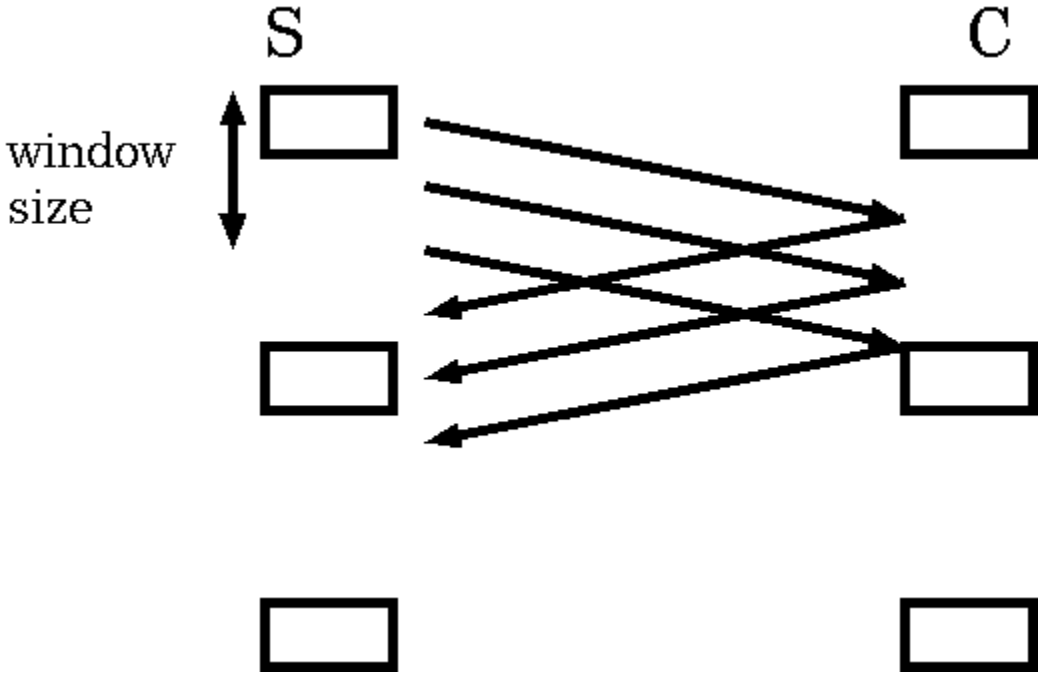
TCP: 転送(損傷、 紛失、 重複、 順序の乱れ)

- SEQ番号から自分がどこまで送信したか?は分かり、 返信パケットのACK番号を見れば、相手がどこまで受け取っているのか?が分かります
- SEQ/ACK番号の抜けや重複、順序の乱れを見ることで、それらの訂正が出来ます
- 届いていないパケットがあれば、 一定時間後にTCPが決断して再送もしくは相手に再送依頼などします
- TCPがバッファリングし、正しい順序に再構成後アプリケーションに渡します



TCP: 転送(転送速度の向上)とフローコントロール

- 一回毎に確認を待つのは非効率
  - 1往復どれくらいか?というと、 北海道〜東京で往復50msくらいかかるので1秒で20往復しか出来ません。 これでは転送速度が30KB/秒くらい…
- 確認を待たずに次々と送信しましょう
  - 確認を待たずに送れるサイズがTCPヘッダのwindowサイズ
  - そうそうエラーにはならずに次々とACKが返ってくると期待してますよ
- フローコントロールの基本
  - ヘッダのwindowでネゴしていきます
  - ホストが忙しいので少し待ってほしいならwindowは小さくしてほしい
  - 余裕が出来たらwindowを少し大きく



TCP/IP周辺の出来事史

年代	出来事	備考
1950年代	タイムシェアリングシステム(OS)研究	軍はCICもしくは防空センターの高度化に興味?
1951?	SAGE(半自動防空管制)開発開始	MITリンカーン研究所(1951)、運用開始は1958
1957	スプートニク１号打ち上げ	宇宙開発競争のはじまり、空から核ミサイル?
1958	ARPAとNASAの誕生	宇宙開発の主導権はNASA
1960年代	ARPAのIPTO部主導でARPANET開発	IPTOの予算はARPA全体の0.5%(期待されてない?)
	(ARPANETは60年代後半みたい?)	IPTO部長は対話的コンピューティング関係者
1969/10/29	ARPANETで最初のパケットが流れる	プロトコルはNCP, UCLA〜SRI間
1970年代	TCP/IPの設計と開発	(ARPAと無関係ながら)OSではUnixがブレイク
1970年代末	NCPからTCPへの移行を計画	UCBがBSD UnixにTCPを実装する案件を受注
1981-1983	4.2BSDでのTCP/IP実装	Bill Joy(pascal,vi,termcap,のちのSUN CTO)
1983/01/01	ARPANETがNCPからTCP/IPへ完全に移行	商用化とか対OSI(政治)とかいろいろあるみたい?
1991	HPC Act (by Al Gore)	のちのクリントン政権での情報ハイウエイ構想
		商用インターネットの躍進がはじまる
1995	Win95登場	一般人のインターネット率が急上昇していく

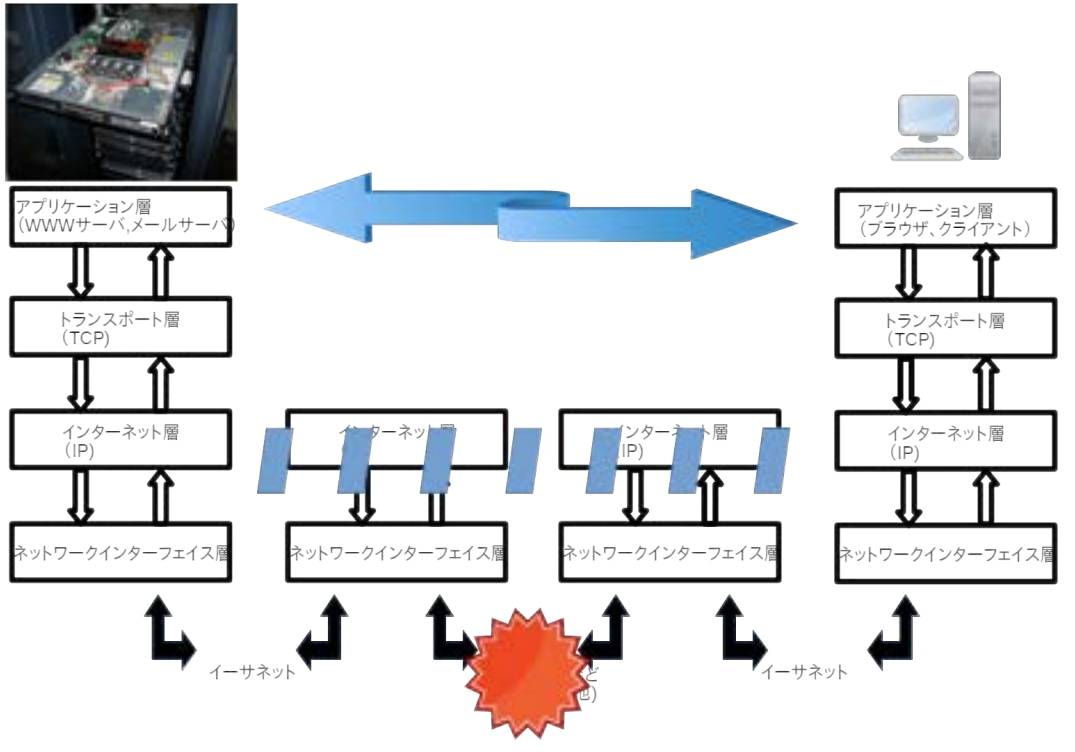
情報技術応用特論 第03回
TCP/IP(3) インターネット層

Copyright (C) Ken'ichi Fukamachi <fukachan@fml.org>, 2021-2025. CC BY-NC-SA 4.0

インターネット層の概要

インターネット層(IPとICMP)の役割

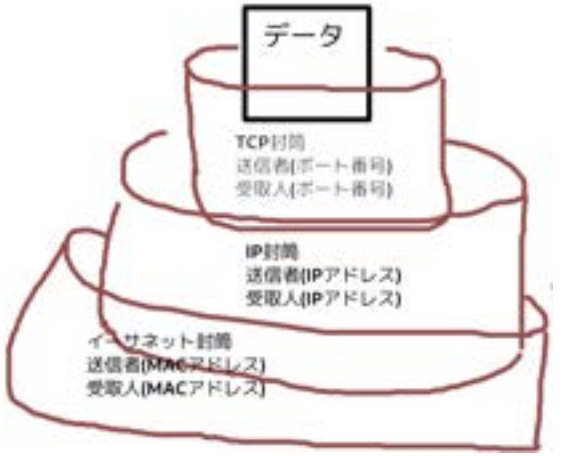
- IPは右図のようにインターネット層でのパケット転送を担当します
  - END-TO-ENDで、 あるホスト(IPアドレス)からホスト(IPアドレス)への転送
  - IPは**大域的な転送**の担当、 ネットワークインターフェイス(NI)層 (イーサネットなど)は**隣あった機器間の転送**
- TCPとIPはコンビです(再掲)
  - IPはIPアドレス(**住所**)間の転送を担当
  - TCPはアプリケーション(**誰**)宛の担当

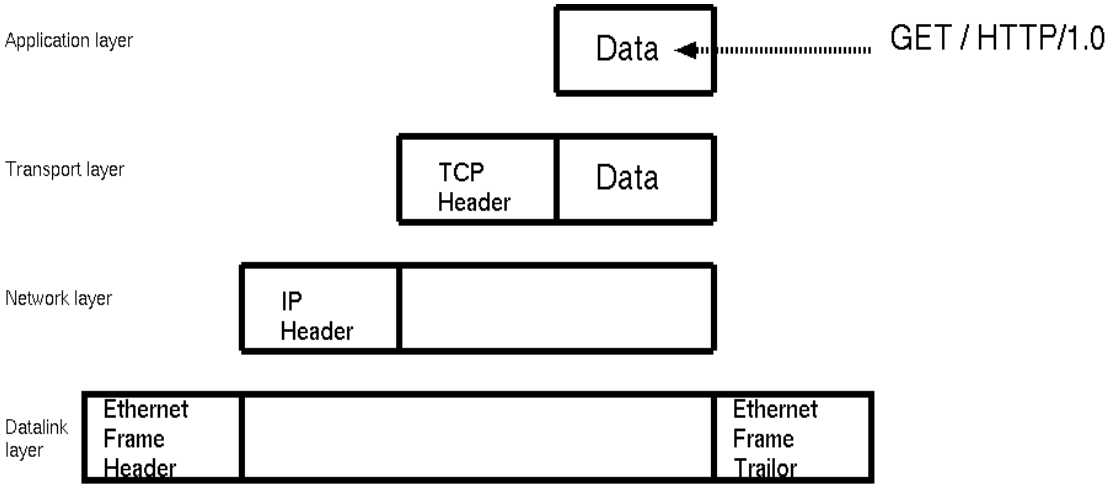


(脚注) ホスト=コンピュータ。 ホスト(host)というと英語では主人ですが、 かつて(大型コンピュータの時代)、 サービスを提供する(ユーザをおもてなしする)ことから**ホストコンピュータ(略してホスト)**と呼びました。 その名残り。 当時ユーザはホスト本体には触れず、ホストにつないだ**小型の端末(ターミナル)**でコンピュータを使っていました。 **ターミナル**という名のアプリは、その名残り

IPパケット(再掲)

- インターネット層(IP,ICMP)は、 トランスポート層(TCP,UDP)から渡されたパケットにIPヘッダを追加し、 IPパケットをネットワークインターフェイス層(NI層,Ethernetなど)へ渡します。 例:
  - TCPパケットにIPヘッダを追加してIPパケットを作り、 下層(NI層)に託します
    図(上)は、封筒に入れて入れて...のイメージ
- IPパケットサイズの上限も(TCPと同様で)下層の制限で決まります。例:
  - Ethernt(1500)
  - PPPoE(IP over PPP over Ethernet(1500))

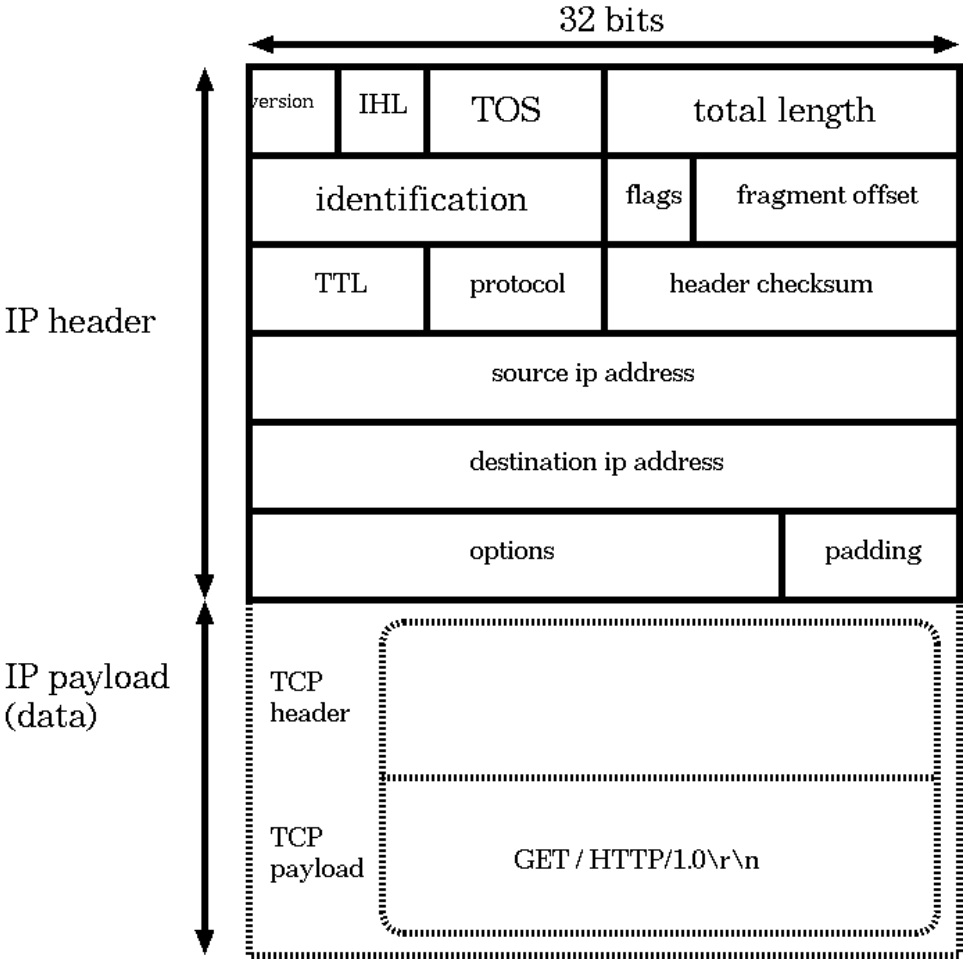




(脚注) [復習] パケットとは封筒のようなもので宛名書きがヘッダにあたります。 それを右下図のように書きます

IPv4ヘッダ

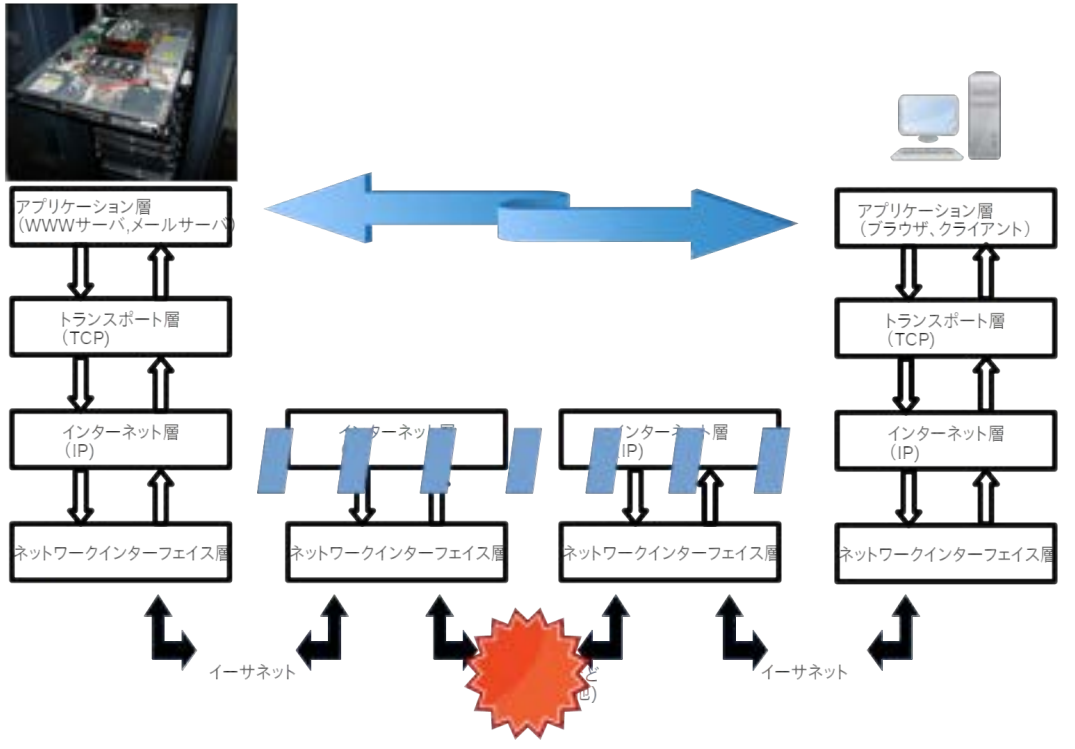
- ヘッダに**送信元と送信先のIPアドレス**(図の上から4行目と5行目)があります
- ペイロードにトランスポート層のTCPパケットなどが入ります。 3行目にあるprotocol(数字)がペイロードに入っているパケットの種類を表しています
  - 1 ... ICMP (後述)
  - 6 ... TCP
  - 17 ... UDP
  - 50 ... ESP (IPSec, VPNで利用)
  - 51 ... AH (IPSec, 最近使わない)
- あとの細かいところは省略します



(脚注1) IPSec = IP Security (脚注2) ヘッダを丸暗記する必要はありませんが、 **IPアドレスが書いてあることは覚えておいてください**

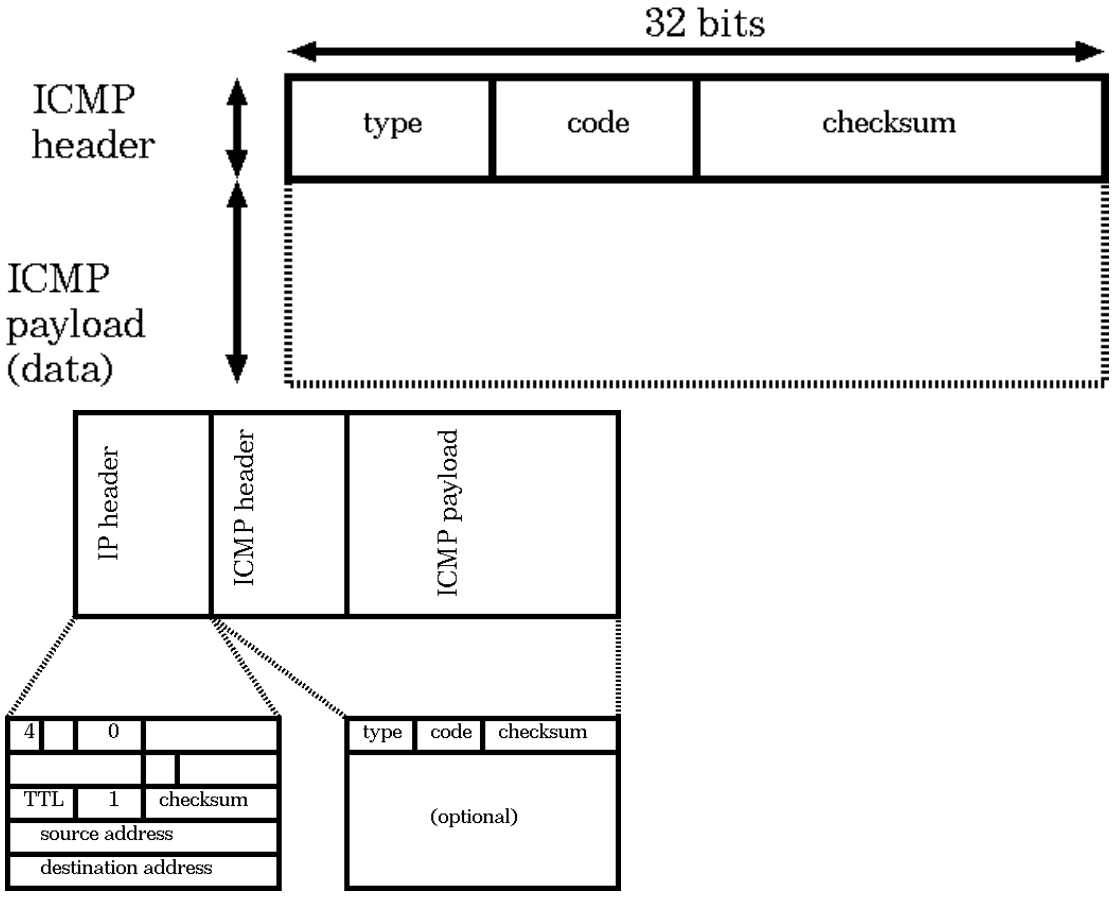
IPのエラー対応(ICMP)

- IPはUDPと同様でエラーに対処しません
  TCP/IPのエラー対策は (1)**TCP**に頼るか (2)**各ソフトウエアの独自対策**のいづれか
- **ICMP (Internet Control Message Protocol)**
  - インターネット層の各種情報を通知する仕組みとしてICMPがあります
    - 用途: デバッグ、エラー、おすすめの情報などの通知
  - ただし悪用を恐れて**実際の運用ではICMPを許可しないことが多い**ため、 かなり使えないプロトコルです
  - 通信相手の生死確認程度であれば使えることが**多い** e.g. pingコマンド



ICMPパケットとICMPヘッダ

- **ICMPパケットを転送するのはIP**
- IPヘッダの**プロトコル番号は1**
- IPパケットのペイロードがICMP(右下図)
- ICMPヘッダはシンプルに3要素32ビット
  - typeとcode次第でヘッダの次にオプション要素があることもあります
  - ICMPペイロードでデータも運べます
    - 中身はtypeとcode次第
    - もしくは独自拡張



(脚注) ヘッダを丸暗記する必要はありません

(アーカイブ動画で見ている人も)一時停止して休憩

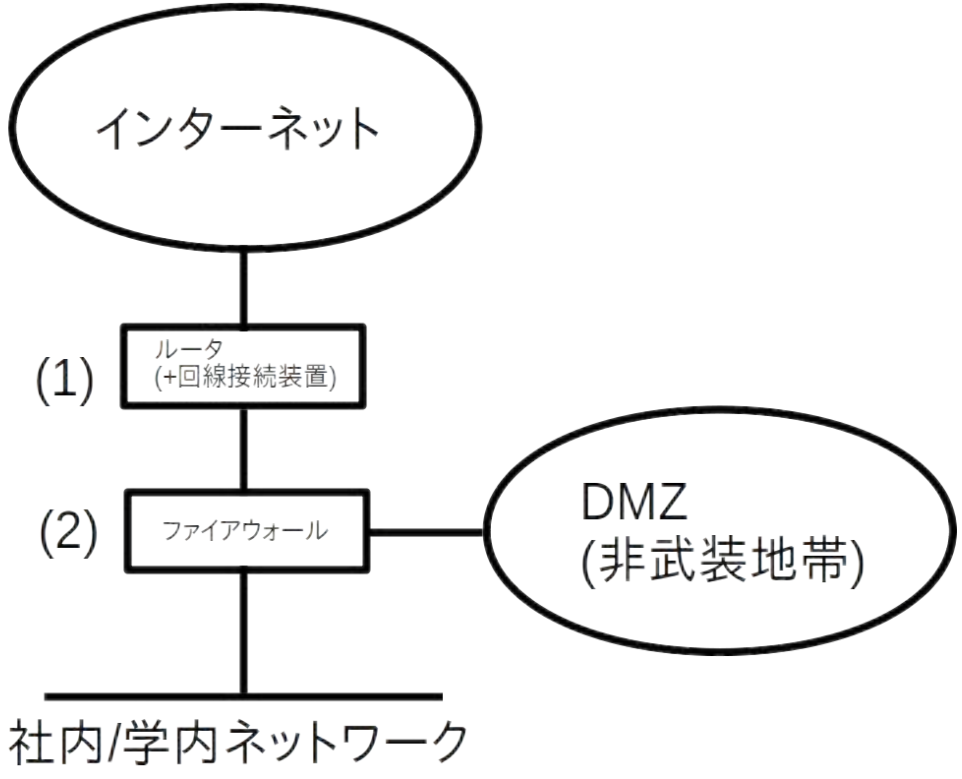
30分ごとに雑談（休憩）をしろというのが指導教官の教え

正確には「アメリカでは30分ごとにジョークを言わないといけない」だけれど、そんな洒落乙なこと無理:-)

ネットワーク構成

ネットワークの基本構成

- **社内/学内**と**社外/学外**という用語があるのは、1990年代半ば以降、 図のようなネットワーク構成が基本だからです
- この基本構成図を覚えてください
  (家庭のネットワークは右図の劣化版)

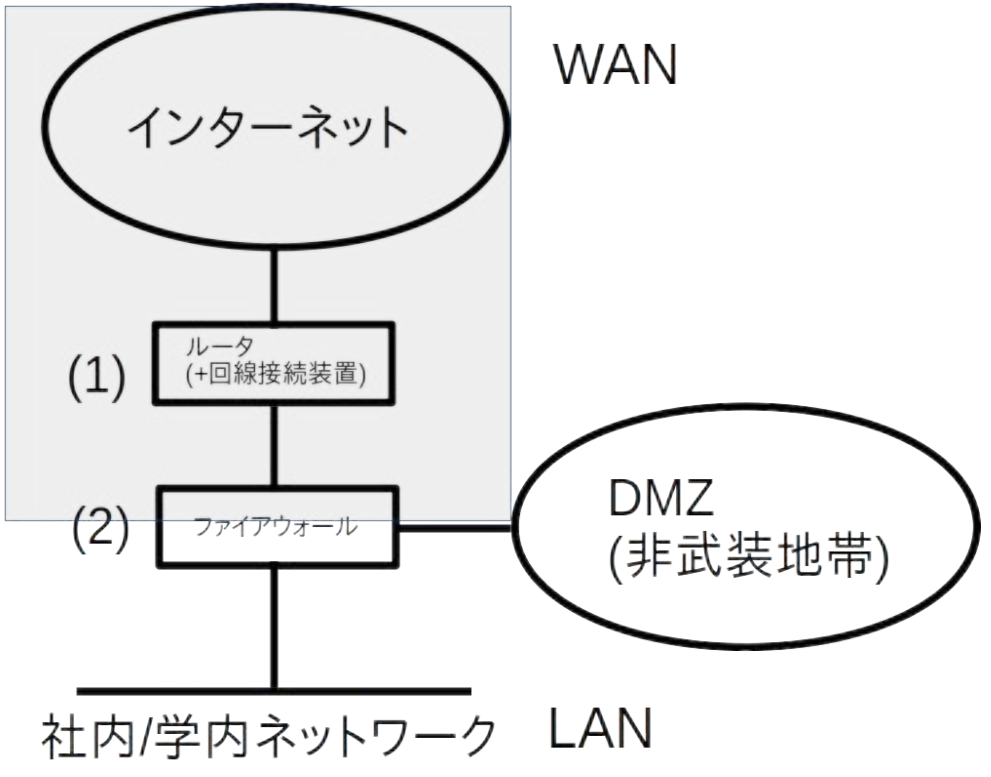


(脚注1) 正確には、 ルータの先(インターネット側)に通信事業者(キャリア)の機材 (回線終端装置,ONUなど)があるのですが、 (そこはプロバイダ屋の仕事の範囲外、キャリアさんの仕事範囲なので)省略してます

(脚注2) 情報システム工学科3年春学期の後半「設計編」でやりましたね？ね？

用語: ネットワークの基本構成(論理的な区分?or文脈依存?)

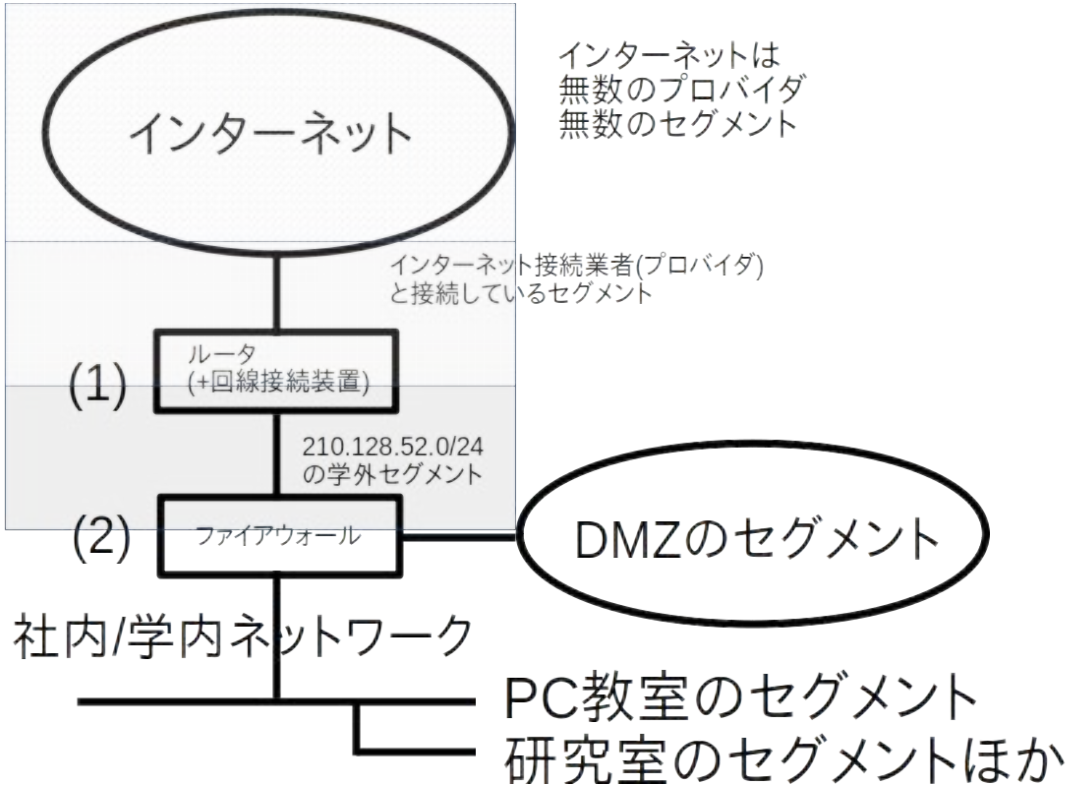
場所	用語	備考
学外	WAN	いわゆるインターネットですが(2)より上は世界中と通信できる領域
学内	LAN	イントラネットと呼ばれることも
DMZ	DMZ	インターネット向けサーバなどを置く場所。学外からの通信は厳しく制限しつつ、学内むけサービスも行う
(1)		ルータ(+通信事業者との接続に必要な装置)でインターネットに接続
(2)		ファイアウォールという装置を置きます(業務用,一般家庭には無し)



(脚注1)物理的には(1)(2)も学内 (脚注2) 90年代なかばからDMZ(=非武装地帯)という用語を使っていますが違和感しかない。 ここ別に平和な場所ではないです。 個人的には**区画(frame/compartment)**と**隔壁(bulkhead)**という用語を使いたいところです。 なぜなら、われわれは、**やられる前提でサーバを置いて**いて、 非常時には切り離す場所(**ダメージコントロール**の対象)だからです

用語: ネットワークの基本構成 (セグメント)

- **セグメント**(segment)
  - ネットワークの断片、範囲
  - 正しくはnetwork segmentです
  - **ブロードキャストが届く範囲**がセグメント
  - **ブロードキャスト**は (Ethernetで後述
- ネットワーク機器の区別
  - **ルータ**は異なるセグメント間の接続
  - **スイッチ**は本来**同一セグメント**をつなぐ機材 (学内はスイッチが基本、詳細は3年春学期後半「設計編」を参照)



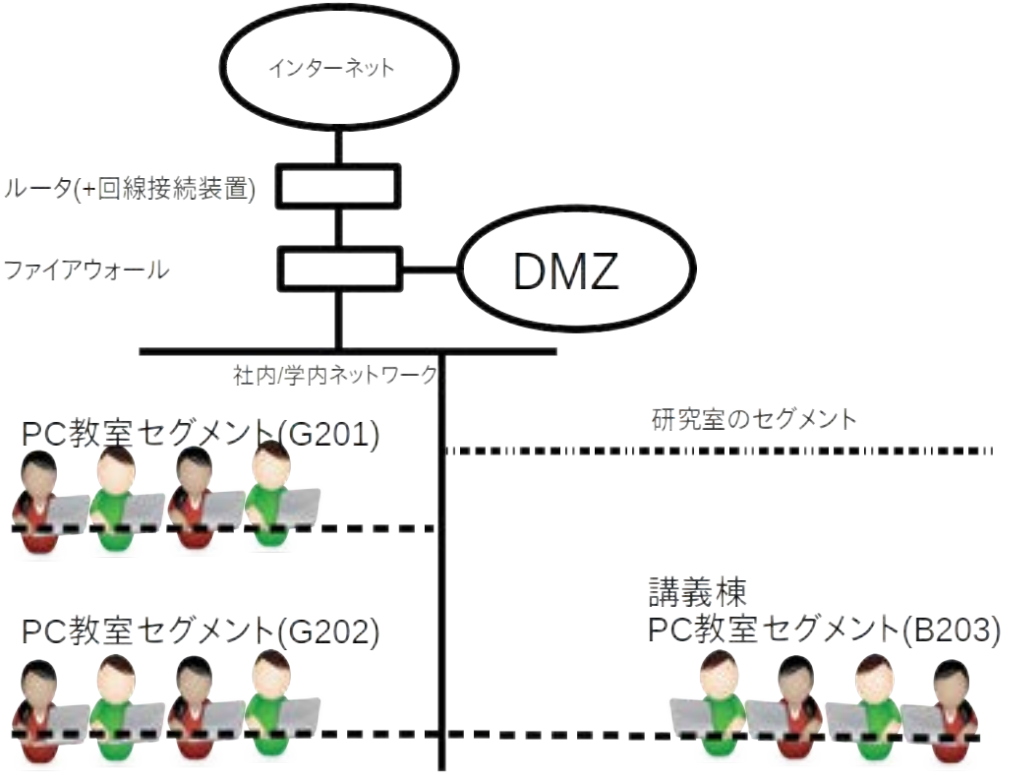
(脚注1) セグメントは連続したIPアドレス帯。 例: 192.168.10.0〜192.168.10.255が使えるセグメント

セグメントとネットワークは同義語ですがネットワークも曖昧な用語で困りますね

用語: ネットワークの基本構成 (学内ではVLANを利用する話)

- 大昔は「**一つの部屋**の中にあるPC群は**一つの同じ(物理)セグメント**に接続」されていましたが、1990年代以降のイントラネットでは、セグメントを**論理的に延長するVLAN(Virtual LAN)**という技術を使うことが普通です

- PC教室のセグメントは建物を越えて複数の部屋にまたがっています(右図)
  - PC教室セグメントのPCは、すべて172.23.x.yというIPアドレスです
  - とりあえずPCの引越しは楽ですね...



(脚注1) VLANを使うと一つのスイッチから複数のセグメントが伸びます (論理的な構成図と物理的な構成図が異なります)

(脚注2) 上の図は論理構成図です

(アーカイブ動画で見ている人も)一時停止して休憩

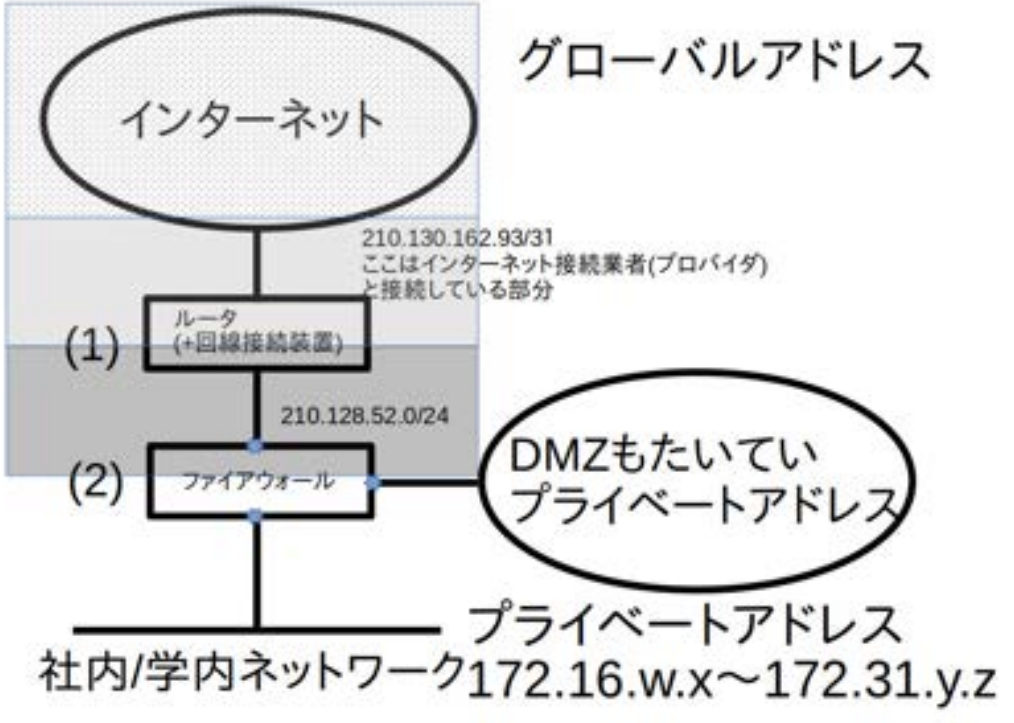
30分ごとに雑談（休憩）をしろというのが指導教官の教え

正確には「アメリカでは30分ごとにジョークを言わないといけない」だけれど、そんな洒落乙なこと無理:-)

IPアドレス

ネットワークの基本構成とIPアドレス

種類	説明
グローバルアドレス	住所としての意味があるIPアドレスで、ホストを世界で一意に識別するための数字です。(2)より上側で利用します
プライベートアドレス	ユーザが自由に**利用してよい**IPアドレスで、(2)より下側つまり社内/学内/家庭内ネットワークで使います。次の3つの範囲が利用可能(アドレスの表記法は後述)
	10.0.0.0/8
	172.16.0.0/12
	192.168.0.0/16



(脚注1) AWSはglobal IP (addres)のことをPublic IPと称しています。 まぁ普通に英語なので意味は通じますよね?

(脚注2) global IPという用語のほうが広く使われている気がしますが、 private IPの反対はpublic IPでしょうからAWSの方が正しい？

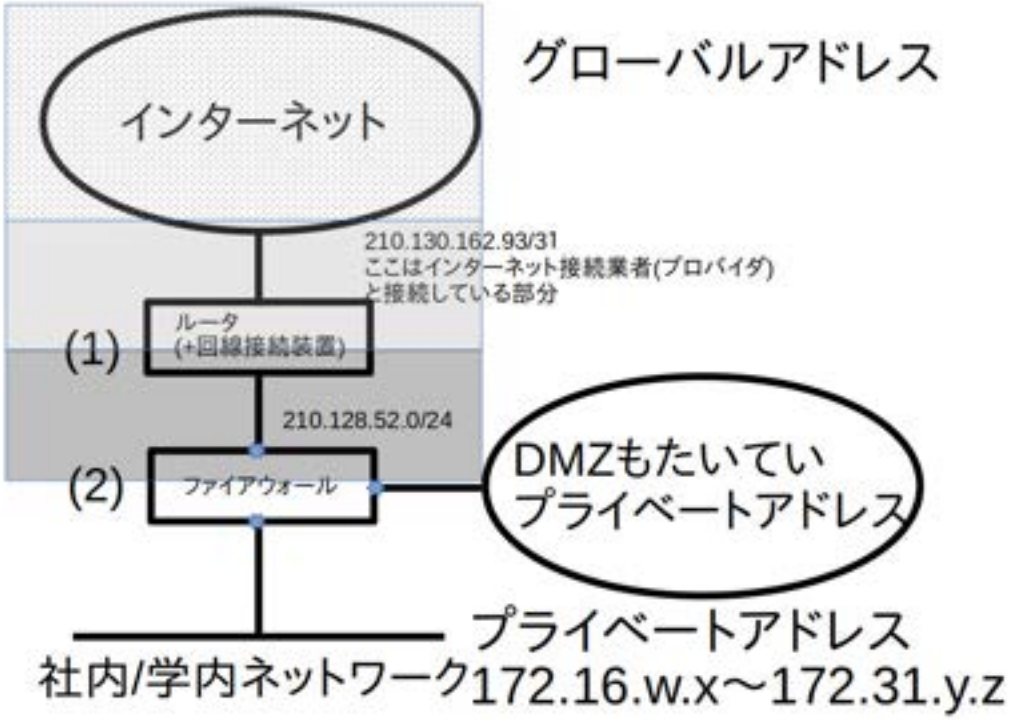
(脚注3) 世界的には無意味でも、 プライベートIPを学内で識別できる数字(住所）として使うので、 当然、学内LANの中で一意に割当

IPアドレスはネットワークインターフェイスの識別子

- 正しくは、「IPアドレスとは**ネットワークインターフェイスにつける識別子**」です
- PCの識別子ではありませんが、**たいていPCのインターフェイスは1つなので住所とみなせます**
- 図(右)の業務用サーバのイーサネットは3口
  - 中央付近の緑色の養生テープの下の3口
  - どれもケーブルが1つだけ刺さっています…
  - 口ごと（接続先セグメント）ごとに異なるIPアドレスを設定していきます

IPアドレスはネットワークインターフェイスの識別子

- 図の(1)や(2)の機材には複数のIPアドレスがついています (例: 図中(2)の＊3ヶ所)
- 接続しているセグメントごとに異なるIPアドレスになります
  - たとえば(2)のファイアウォールは、 上側の学外側インターフェイスが210.128.52.4、 下側の学内側インターフェイスが172.16.0.4、 右側のDMZ側インターフェイスが192.168.1.4



IPv4とIPv6アドレス体系の大まかな概要と比較

- IPv4 (IP version 4)
  - 大きさは**32ビット**
  - 表記法:**8ビット**ずつ**.(dot)**で区切り**10進数表記**(詳しくは後述)。例：
    - 210.128.52.45
    - 192.168.10.1
  - グローバルIPアドレスはJPNIC に申請しJPNICから割り当ててもらいます (ドメインと同様に申請した組織がIPアドレスの管理を委任されると考えてください)
  - プライベートアドレスの利用は自由 (使い方は次節NATを参照)

- IPv6 (IP version 6)
  - 大きさは**128ビット**
  - 表記法:**16ビット**ごとに:で区切り**16進数表記**
    - 2403:3a00:202:1209:49:212:144:112
    - 2001:240::105
      0000を0、さらに連続した0を::で省略可 (::は一ヶ所限定)
  - 申請手続きは同様です
  - IPv6のアドレスの**種類**は複数あり、**一つのインターフェイスに複数種**つけられます。
    その中にはイントラネットのみで使うプライベートっぽいアドレスもありますがIPv4のようなNATは出来ません(NATは次節参照)

(脚注1) 実際のJPNIC申請はISPが代行します。 ユーザが直接JPNICとやりとりすることはありません

(脚注2) そもそもIPv6にはNATの仕様がありません。 アドレス空間が巨大なのでインチキせずに全機器にグローバルIPを割り当てます

IPv4の表記法

- クラス (classfull)
  - 旧表現 (1993年以前?)
  - IPアドレスを二つに分けて考えます
    - **ネットワーク部(network part)**
    - **ホスト部(host part)**
  - 分け方は**サブネットマスク**(subnetmask)で指定します
    - 分け方は3種類のみで、それぞれ**クラスA,B,C**に対応します(後述)
  - **IPアドレス/サブネットマスク**という表記をすることもあります
  - 例: 192.168.10.1/255.255.255.0
    - IPアドレスが192.168.10.1
    - サブネットマスクが255.255.255.0
- CIDR (Classless Inter Domain Routing)
  - 新表現(1993〜)、発音は「さいだ〜」
  - 分け方は可変長で、分け方(ネットワーク側の大きさ)を/数字 (network prefix、host prefixもしくは単にprefix)で表現します
  - **VLSM(Variable Length Subnet Mask)** つまり可変長のサブネットマスクと呼んでいます
  - **IPアドレス/prefix**と表記します
  - 例: 192.168.10.1/24
    - **IPアドレスが192.168.10.1**
    - **network prefixが24**

IPv4の分類(クラス)

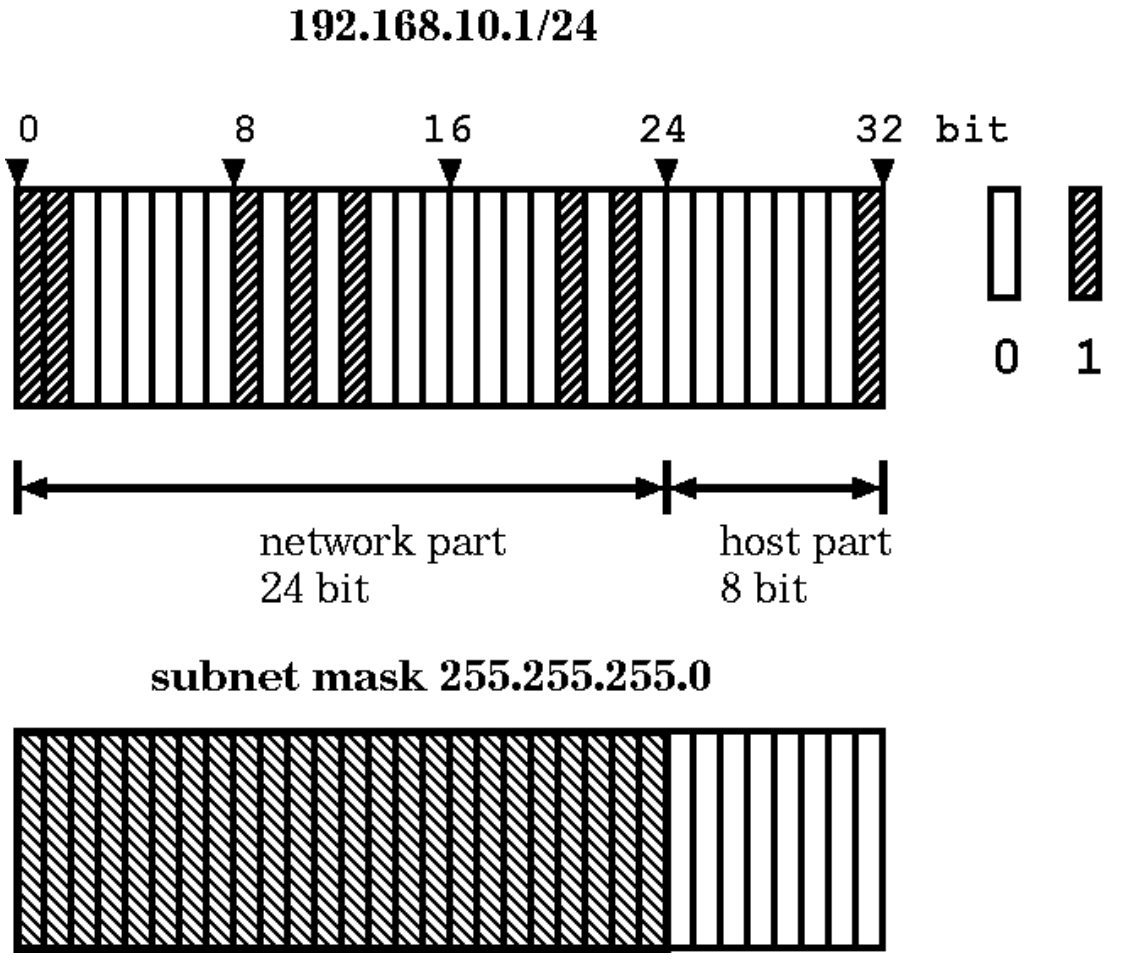
クラス	役割	ネットマスク(/prefix)	IPアドレスの範囲	通信の仕方
A	大規模組織用	255.0.0.0 (/8)	0.0.0.0 〜 127.255.255.255	ユニキャスト(1対1)
B	中規模組織用	255.255.0.0(/16)	128.0.0.0 〜 192.255.255.255	ユニキャスト(1対1)
C	小規模組織用	255.255.255.0(/24)	192.0.0.0 〜 223.255.255.255	ユニキャスト(1対1)
D	マルチキャスト		224.0.0.0 〜 239.255.255.255	マルチキャスト(1対多)
E	予約済(未使用)		240.0.0.0 〜 255.255.255.255	-

- **クラス**は、 1990年代なかばには旧式になった用語ですが、 大昔のネットワークがあるかぎり現場では使いますので、 おぼえてください(基本情報でも出ます)
- **ユニキャスト**は**1対1**の通信です。 IPアドレスは住所に相当します
- **マルチキャスト**の場合、IPアドレスに**住所の意味はありません。** 通信は**1対複数**で、**IPアドレスはチャンネルにあたります。** そのチャンネルに参加しているホストすべてに通信が届きます

(脚注1) multicastを使った会議中継を90年代前半までは見かけました。 認証や課金が出来ないため商用インターネットではmulticastの使いどころがないです。 ちなみにIPv6の裏側ではmulticastを多用しています(なぜなら作者が同じだから:-)

例: 192.168.10.1/24の新旧表記法

- 旧表現 192.168.10.1/255.255.255.0
- 新表現 192.168.10.1/24
- 旧表現では**ネットワーク部が24ビット、ホスト部が8ビット**です
- /24 (network prefix が 24)はネットワーク部が24ビットという意味です
- 255.255.255.0を2進数で書くと**左から1が24個**ならびます(右下図)。 network prefixの24は、この24を意味しています



(脚注) /28とか/14とか半端なprefixにすると分かりにくいですが、 10進数表記にするから分かりづらいだけ

IPv4アドレスの使い方(割り当て方)

- 割り当てられたIPアドレス群の中で、両端はホストの割り当てに使えません
  - 最小のIPアドレスを**ネットワークアドレス**、最大のIPアドレスを**ブロードキャストアドレス**と呼びます
- 残りの部分はPCに割り当てられます
  - つまり住所としてPC（ホスト）に使えるIPアドレスの数は「**ホスト部の大きさ(2^ホスト部のビット数) - 2**」個が上限
  - CIDR表現なら**2^(32 - network prefix)**個

- 例: 研究室では192.168.10.0/24を使ってください
  - IPアドレスは192.168.10.0 〜 192.168.10.255 までの256個
  - ただし両端（次の２つ）は使えません
    - 192.168.10.0(一番小さいIPアドレス)は**ネットワークアドレス**
    - 192.168.10.255(一番大きいIPアドレス)は**ブロードキャストアドレス**
  - ホストに使えるIPアドレスは
    - 192.168.10.1 〜 192.168.10.254 までの254個
    - ホスト部は8(32-24)ビットなので
      (2^8=256) - 2 = 254

IPv4アドレスの使い方で特別なもの(1)

- ブロードキャストアドレス(既出)
  - **同一セグメントにあるホスト(群)へパケットを一斉に送信したい**場合に使う宛先
  - インターネット層では、IPパケットにブロードキャストアドレスを宛先として指定して送信すれば、 全ホストへ届く。 その動作原理は次節「イーサネット」で解説
  - **一番大きなIPアドレス**を使う約束
  - 例: 192.168.10.0/24 セグメントの場合、192.168.10.255がブロードキャストアドレス
- ネットワークアドレス(既出)
  - **一番小さなIPアドレス**で、 利用するIPアドレス帯を**代表**するIPアドレスです。 「192.168.10.0/24のセグメント」といった表現をします。 例: 192.168.10.1/24は192.168.10.0/24の中でホストに割り振るIPアドレスの1つ
  - 例: 192.168.10.0/24 セグメントの場合
    - 192.168.10.0 がネットワークアドレス
  - 実際のところ特別な用途はないのですが、後方互換性のため利用しません

(脚注)BSD Unixの初期実装ではネットワークアドレスがブロードキャストでした

IPv4アドレスの使い方で特別なもの(2)

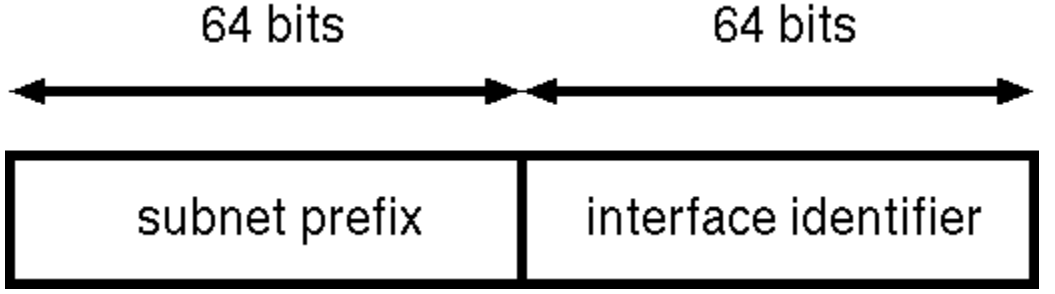
- プライベートアドレス(既出)
  - 自由につかってよいIPアドレスなので**世界中で重複**しています
  - このアドレスをつけているホストは**インターネットに直結**できません (同じIPアドレスのホストが世界中に無数にあるから)
  - この変なアドレスをどう使うのか?は、 次節「NAT」を参照
  - 各クラスから1領域ずつ割いています
    - 10.0.0.0/8
    - 172.16.0.0/12
    - 192.168.0.0/16

- 127.0.0.1 (loopback address)
  - このIPのホスト名は**localhost**
  - **自分自身**を意味します。とうぜん全コンピュータに存在します。 自分宛=他のホストと通信しないため、127.0.0.1は重複していても問題ありません
  - デモで127.0.0.1を使っているので既出
  - この特別な127.0.0.1があるので、 実は127.0.0.0/8は一般のIPアドレス割当に使っていません (もったいない)

(脚注) ふつう127.0.0.1に物理的な実体はありません。 OS(カーネル)では127.0.0.1をソフトウエアで処理する実装なのでsoftware loopbackという表現もあります

IPv6アドレス

- 前述の大きさやアドレス表記法といった基本的な事柄は覚えてください
  - 大きさは**128**ビット
  - **16ビットごと**に**16進数**、:つなぎ
  - 連続した0は省略可
- IPv4と同じ**CIDR表記**です
  - 2403:3a00:202:1209:49:212:144:112/128
  - 2403:3a00:202::/64
- IPv6のユニキャストアドレス(住所の意味合いを持つアドレス)は、 左側64ビットが**サブネットプレフィックス**、 右側64ビットが**インターフェイス識別子** (MACアドレスを元に自動生成したホストごとに異なる値)



- たとえば70:85:c2:da:bb:72という MACアドレスのネットワークインターフェイスに自動でつくIPv6アドレスは fe80::7285:c2ff:feda:bb72 になります。ここで fe80::/64 が prefix で、7285:c2ff:feda:bb72が自動生成(Modified EUI-64という規則で変換) された右側64ビットです

IPv6のメリット

- IPv6の必要性
  - IPv4ではIPアドレスが足りないため
  - v4の最大43億程度では地球人口以下
  - そこで1990年代前半にはv6開発開始
- 128ビットの巨大なアドレス空間
  - IPv4の96ビット倍は簡単に数えられない巨大さで、 アドレスが足らないという話は当面ありません

- IPv6の開発開始と同時に、 **IPv4延命策としてプライベートアドレスとNAT**を導入
- **IPv6ではTCP/IPの原点へ回帰**
  - **IPv6ではグローバル**アドレスで通信
  - **正しいEND-TO-END通信**へ回帰
  - NATはしません、出来ません
- ヘッダの整理など細かな技術的改良
- 新機能
  - IPSec (IP Security, VPNなどで利用)
  - IP Anycast

(脚注1) 参考:(ネットワーク運用) IPv6を使うユーザが少なくv6ネットワークが空いている (俗に言うスカスカの状態な)ので、 自宅のフレッツをIPv4 over IPv6で接続すると速くなるらしいです (という噂に引っかかって...)みんなでv6へ切り替えるとv6劇遅へ(w)

(脚注2) 43億は単なる理論上の数字で、実際にはIPv4のクラスA,B,Cしかユニキャストがないので有効なIPは半分くらいでしょう

IPv6のデメリット(1):技術仕様（私見）

- 新機能の魅力が無い ... IPv6の新機能のほとんどはIPv4でも実装されました(back import to IPv4)
  - よってIPv4でもIPv6の新機能のほとんどが使えるため、移行するべき動機が希薄です
- 巨大なIPアドレス空間と言われても ... この利点だけでは移行の動機として不十分です
  - IPv4が売りきれて久しいので、インターネット後発の国はIPv6を使わざるを得ない状態ですが...
- みんなで欧米王手クラウドサービスに頼るなら、あまり関係ない？
  - 個人利用の環境がIPv6でもクラウドを使うかぎり関係ないかも？なぜなら、 大手ISPはIPv4もたくさん握りしめているし、 IPv4/IPv6どちらでもアクセスできるでしょうから
  - 個人vs個人(peer to peer)は問題になるかもしれませんが（大手サービスでは）主流でないのでok?

(脚注1) IPv6の闇を語るよ〜まぁ、デメリットについては異論もありましょう

(脚注2) 欧米諸国(1990年代初頭にインターネットを導入した国々)では、 IPv4アドレスをわりとたくさん確保しているため、あえてIPv6に移行する動機が希薄です。 後発の中国やロシア、インド、中東、アフリカはIPv6を使いたいでしょうけれど

IPv6のデメリット(2):運用設計 （私見）

- IPv6主体のネットワーク設計のベストプラクティスが分かりません
  - 「（NAT無しの）ネットワーク設計」の良い提案を聞きません
  - 「原点にもどれ」という趣旨は分かりますが、 いまさら30年もやってきた（ファイアウォールをはさむ）設計ポリシーをくつがえすのは難しいです
- IPv6が普及しないのでIPv6用の機器が進化しませんが、機器がないからIPv6が使えないわけで…
  - 膠着状態が20年以上つづく所謂「**鶏が先か、卵が先か**」問題です ;-)

(脚注1) 近年「家庭のBフレッツをIPv6経由にすると高速」という営業をしてますが、 v6の技術が優位だからではなく、 v6の設備投資がダブついているからv6へ誘導したいキャリア都合でしょう (フレッツのIPv6バックボーンがスカスカの間だけ真実なので、 買ってみたけど速くない！だまされた！と思う人が出てきてて当然の案件; みんなで使えば遅くなる;実際そういう人が出てきてます)。 我々の産業は巨額の先行設備投資ビジネスなわけで、その負の側面とも言えます

(アーカイブ動画で見ている人も)一時停止して休憩

30分ごとに雑談（休憩）をしろというのが指導教官の教え

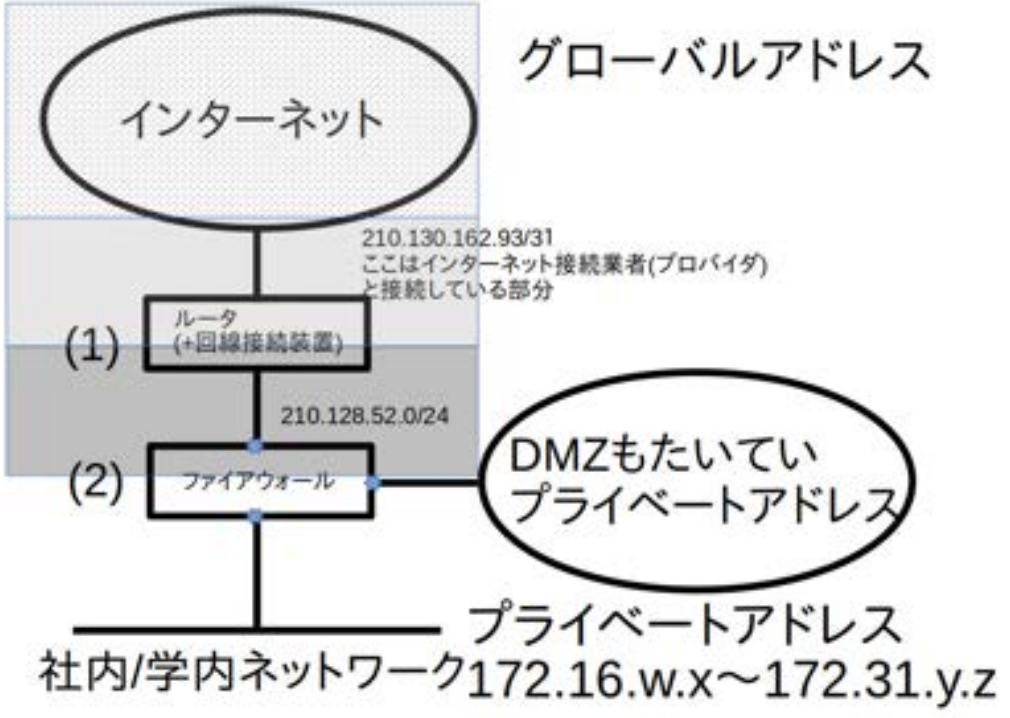
正確には「アメリカでは30分ごとにジョークを言わないといけない」だけれど、そんな洒落乙なこと無理:-)

NAT
(Network Address Translation)

IPv4延命策としてのプライベートアドレスとNAT

- IPv6の開発開始と同時に**IPv4延命策としてプライベートアドレスとNAT**を導入
- 図の(2)でNATを行っていて、 (2)より上はグローバルアドレス、 (2)より下はプライベートアドレス (色がついているところはグローバルIP)

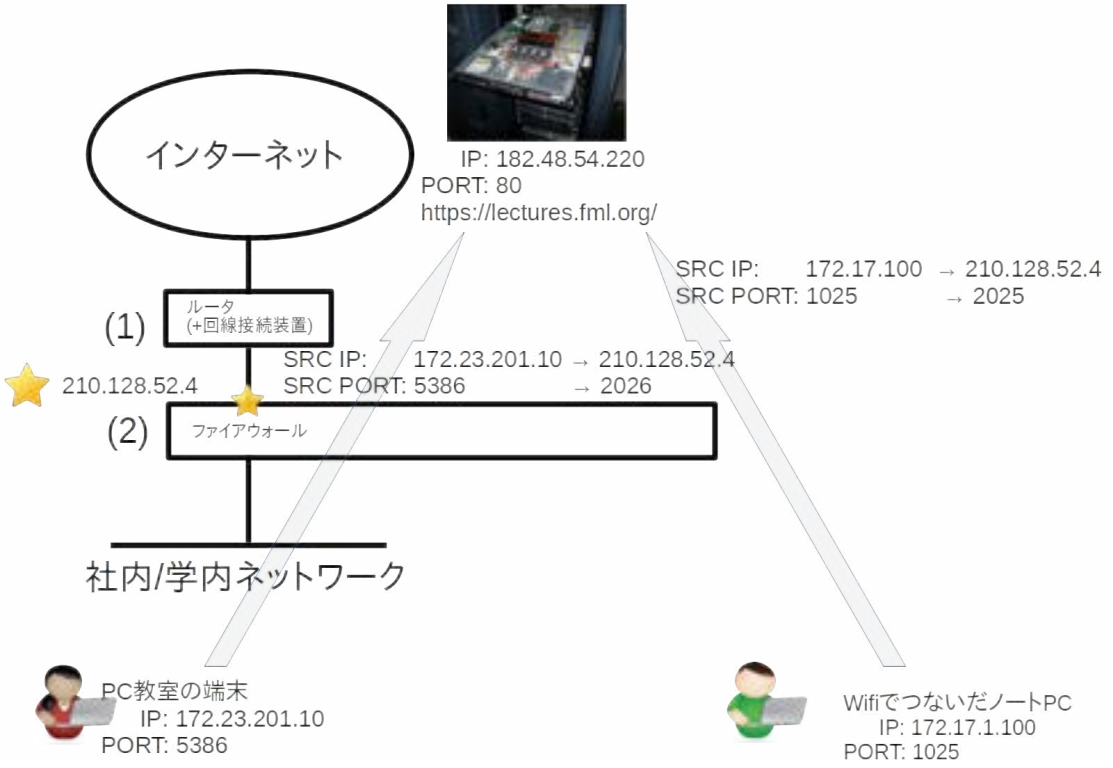
用語	
NAT	Network Address Translation
NAPT	Network Address Port Translation
IPマスカレード	NAPTと同じ意味



(脚注1) 総称がNAT (脚注2) 本来のNATはポート変換なし (脚注3) まずIP masquerade機能がLinuxカーネルに登場し、 後に、それをNAPTと呼んだため名称が二つあります。 設定にnatではなくmasqueradeという名称を使うことがあるため、 どの用語も覚えるべきです

NAT 行きの通信

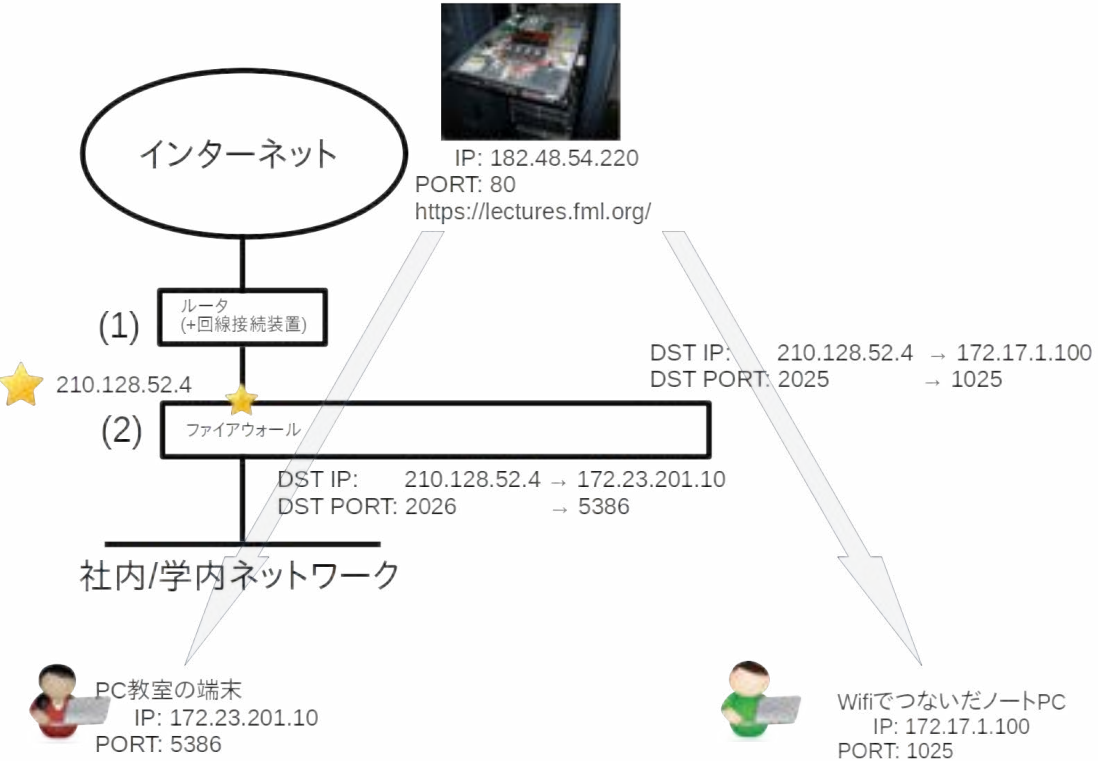
- 例: 学内のホストからWWWサーバ(IPは182.48.54.220)へのアクセスを考えます
- 図(2)の機器のインターネット側インターフェイスのIPアドレスは210.128.52.4です
- 図(2)のファイアウォールを通過時、IPヘッダとTCPヘッダを書き換えます
  - 送信先(DST)のIPアドレスとポートは変えません。 送信元(SRC)のIPアドレスを210.128.52.4へ書き換え、 送信元ポート番号を適当な番号(注: 通信ごとに異なる番号)へ書き換えます
  - この書換えルールを機器は覚えておきます
- WWWサーバ(IP:182.48.54.220)には送信元IP(SRC IP)がグローバルIPアドレス=210.128.52.4に見えるので、 普通に通信できます



(脚注1) DST = destination, SRC = source (脚注2) Global IPとPublic IPは同じ意味です

NAT 返りの通信

- 返りのパケットは、 182.48.54.220(送信元=SRC)から210.128.52.4(送信先=DST)へ送られてきます。 当然ポート番号はHTTPコネクションごとに異なります
- 図(2)の機器は（行きの）各HTTPコネクションの書き換えルールを覚えているので、 IPアドレスとポート番号の組み合わせを照らし合わせ、IPヘッダとTCPヘッダを逆に書き換えます
- 書き換え後、学内側へ送信します
- こうすれば、プライベートアドレスでもインターネットのサーバと通信ができるわけです



NAT メリット

- 前頁で説明した例では、アドレスとポート番号の両方を変換するので、 **NAPT**もしくは**IPマスカレード**と呼ばれます。
- **1つのGlobal IPで同時に多くの通信を処理可能**
  - 通信(例: TCPコネクション、UDPパケット)ごとに、 送信元ポート番号(SRC PORT)を変えていくことで、 ポート番号の大きさ、 つまり16ビット(65536個)分の書き換えルールが(最大)使えます
    (理論上は使えるという話です;脚注1も参照)
- 例: 大学のユーザ数は1000人強ですが、 短時間のTCPコネクションも多いため、 1つのグローバルIP(210.128.52.4)で十分に対応できています
  (SRCポート番号変換だけで処理できています)
  - 16ビット以上に書き換えルールを増やすには、 書き換え先のIPアドレス候補も追加します。 例: SRC IPとして210.128.52.4と210.128.52.5の2つを使えば、 最大で約13万とおりの変換が可能となります

(脚注1) 16ビットの使い方にも流儀があり、 実際には16ビットすべてをNAT変換に利用していません ->JPNICの解説

(脚注2) アドレスとポートを書き換えてゴマかす小手先芸を、 (NAPTも含めて)総称でNATと呼ぶことが普通ですが、 本来のNATはIPアドレスだけを変換する規格のことです

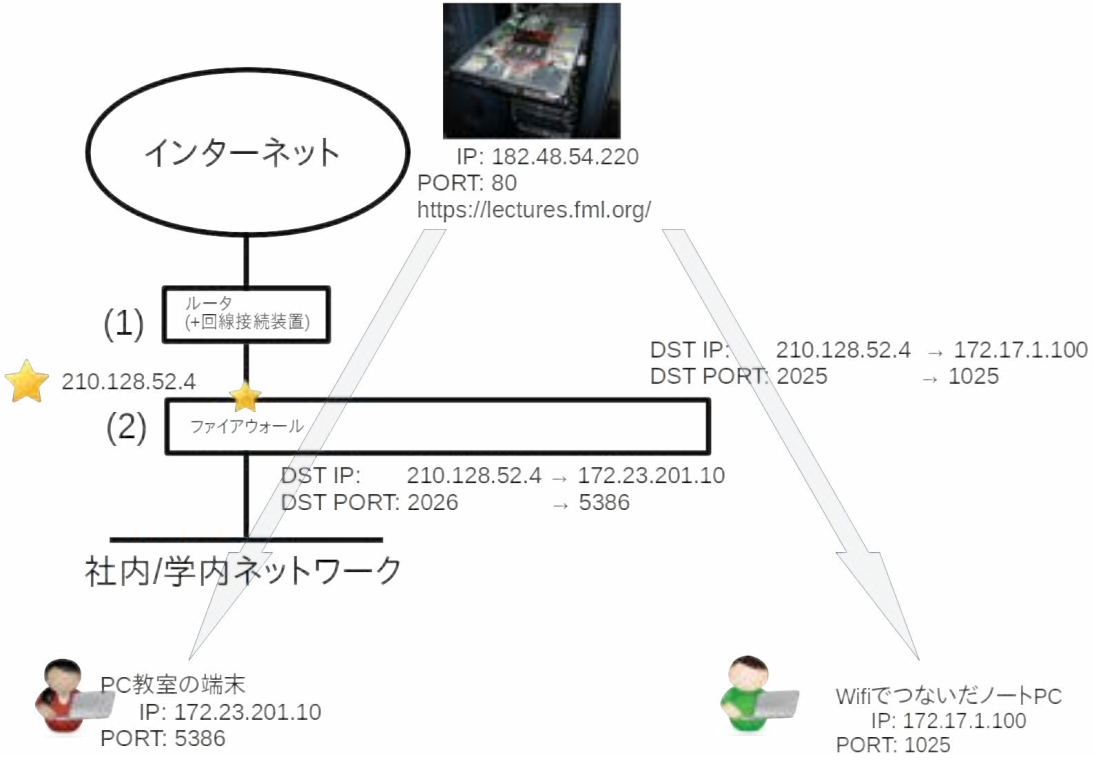
(脚注3) なおIPv6ではNATを使いません。あくまでもNATはIPv4の延命策で邪道な技です

NAT デメリット

- **IPアドレスに住所の意味が無くなります**
  - **一意な数字ではなくなるため**
  - ホストにつけるプライベートアドレスは、世界で一意にホストを識別する数字になっていないので、 正しい意味での住所(ユニキャストアドレス)になっていません
- 本来のEND-TO-END通信ではないです
  - 必ず途中にNATをする機材が挟まります。この機材のせいで、通信がうまくいかないことがあります

NAT デメリット

- **自由な双方向通信が出来ません**
  - 本来のEND-TO-END通信なら特別な制限なく出来るはずです
  - たとえばIP電話で受信する場合を考えます。 インターネット側から来た(いわば一見さんの)初パケットに対しては、 NATの書き換えルールが分からないので変換できません
- 右図は返りのNATの説明(再掲):
  - あくまでも、まず最初に学内から学外へ出て行く通信があり、 そのときに書換えルールが決まります。 そのあとで、はじめて逆変換が出来るようになるわけです。 **「外->内」から始まる通信には対応不可**



(アーカイブ動画で見ている人も)一時停止して休憩

30分ごとに雑談（休憩）をしろというのが指導教官の教え

正確には「アメリカでは30分ごとにジョークを言わないといけない」だけれど、そんな洒落乙なこと無理:-)

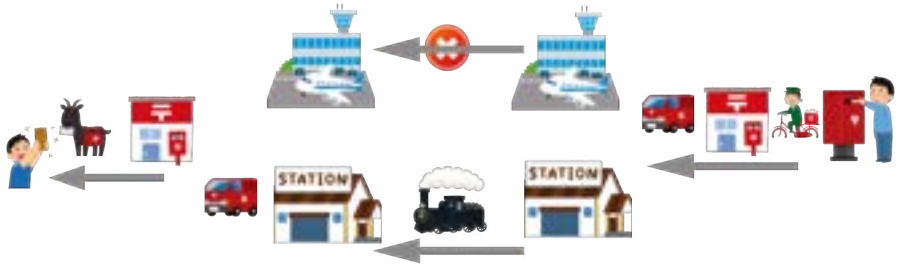
ルーティング

たとえ話: 郵便の配送におけるルーティング

1. （郵便物を投函すると、郵便局員さんが収集して）最寄りの郵便局に集めます。 郵便物をトラックに載せて飛行場へ向けて出発！

2. 飛行場まで輸送してきました。いつもは飛行機に載せるのですが、今日は天気が悪くて飛行機が飛びません

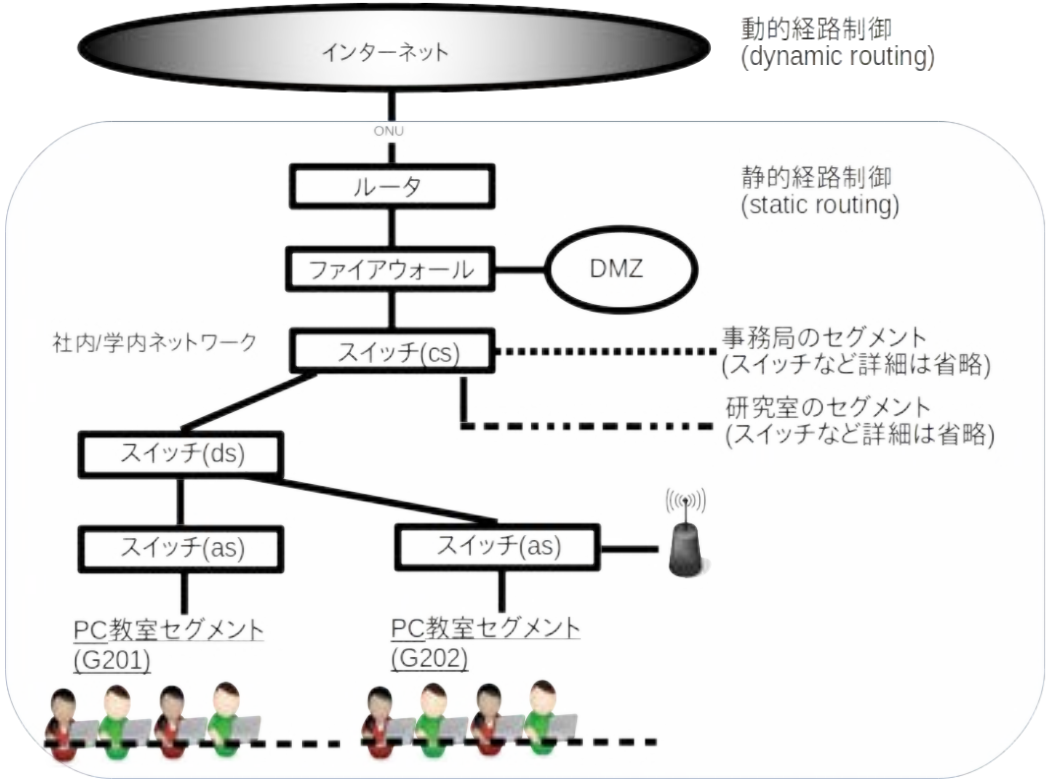
3. そこで鉄道輸送に切り替えます。 これが**動的経路選択(dynamic routing)**です



(脚注1) 冒頭シーン「～収集～」はネットワークと少し違いますが、 そこは例え話なので目をつむってください;-) (脚注2) 現実的な設定は、いろいろ難しいですが、ルーティングの本質は、これだけです。 ちなみにルーティング設定はバックボーン担当の醍醐味です

二種類のルーティング(経路選択)

- **動的経路制御** (dynamic routing)
  - ルータ群が**自律**的に考え経路変更
  - 図のインターネット部分
  - **大域的な経路制御**はdynamic
- **静的経路制御** (static routing)
  - 設定されたとおりに動作
    障害時は人間が対応します
  - 大学敷地内はstaticで十分 (注:インターネットへの出口が一つなのでdynamic routingする理由も手段もないから)です

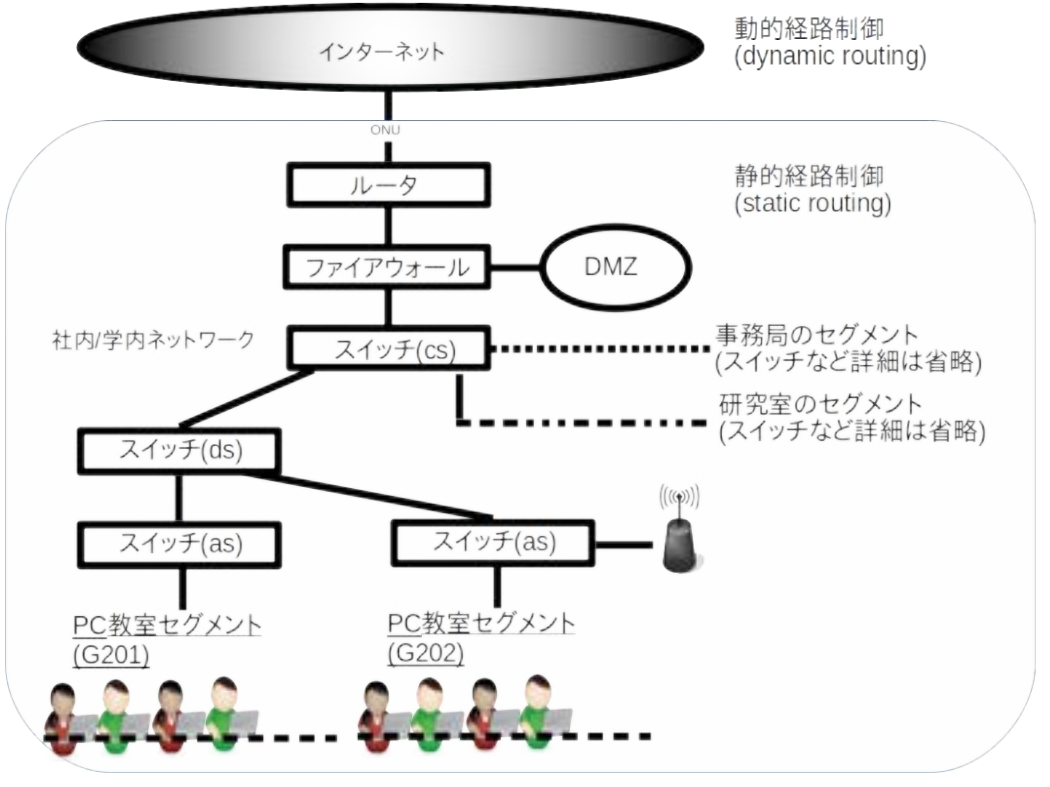


(脚注1) 今回は大域的な動的経路制御つまりインターネット側がメインテーマです。 インターネットの醍醐味

(脚注2) 大きめの社内LANや学内LANでは動的経路制御を行うこともありますが、 技術的にはISPのやることと同じなので省略

図に登場する機器の説明

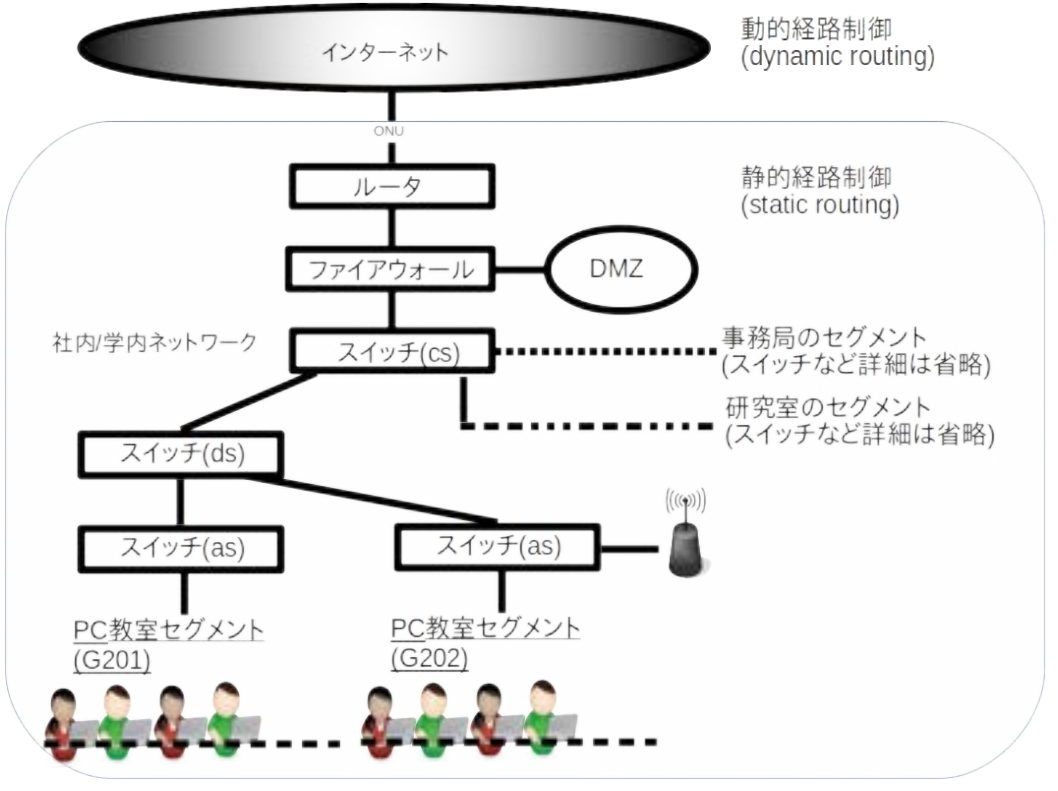
機材	説明
ルータ	インターネットへの経路制御担当 この構成では仕事のしがい無し
スイッチ(cs)	**core switch** **L3スイッチ** VLAN間ルーティングを行う
スイッチ(ds)	**distribution switch** 建物の出口を担当する**L2スイッチ** 建物内のアクセススイッチ(as)群への接続を集約する役目 coreまでVLANを中継
スイッチ(as)	**access switch (edge switch)** 末端に置く**L2スイッチ** PCや無線LAN機器などをつなぐ先



(脚注1) L3スイッチとルータは似ていますが、 ルーティング機能(頭のよさ)に力点がある機材がルータ、 スイッチング機能(パケット転送能力)に力点がある機材がL3スイッチ (だと思う) (脚注2) cs,ds,asはCisco用語 (だと思う)

学内ルーティング(static routing)

- 最初に設定したまま動作しつづけます
  - **学内**側:**そうそう学内の構成は変わらない**ので、これで問題ありません
  - 学外側:インターネット接続回線が一つなので**敷地内の学外側もstatic**です (注:複数のインターネット接続回線があるならdynamicを検討します)
- **障害時は人間が対応**する必要あり
- coreは**VLAN間ルーティング**をします
  - 異なるセグメント間(例:PC教室～研究室)や、 学内セグメントとインターネット側とのパケットのやりとり
- 学内 -> cs -> 学内(別セグメント)
- 学内 -> cs -> ファイアウォール -> インターネット,DMZ

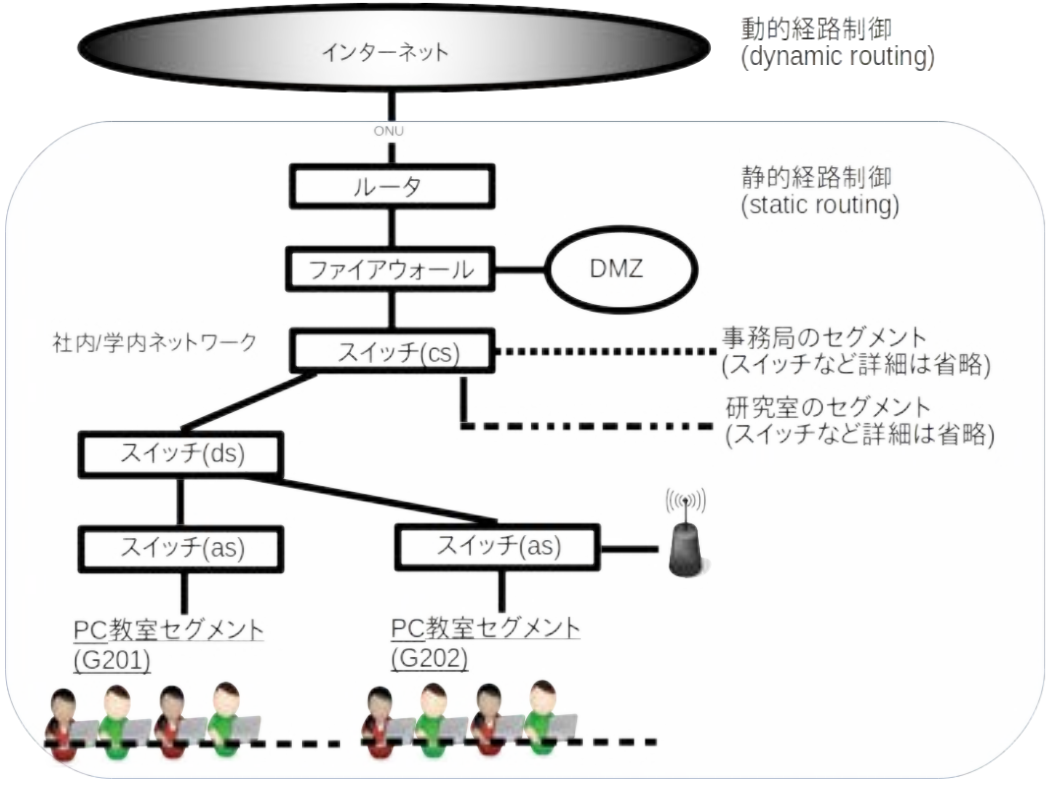


(脚注1) 用語「学内」が曖昧です。 実際、図の大枠の中は**物理的に学内**です。 でも、ネットワーク的(or 論理的)には、 ファイアウォールの上(インターネット)側を**学外**と言うことが多いと思います。 このへん普段から文脈とあうんの呼吸で互いに理解している感じです

(脚注2) つまり異なるセグメント間の通信は一度coreまで行って戻ってきます。 例：研究棟の研究室～大学院棟のPC教室

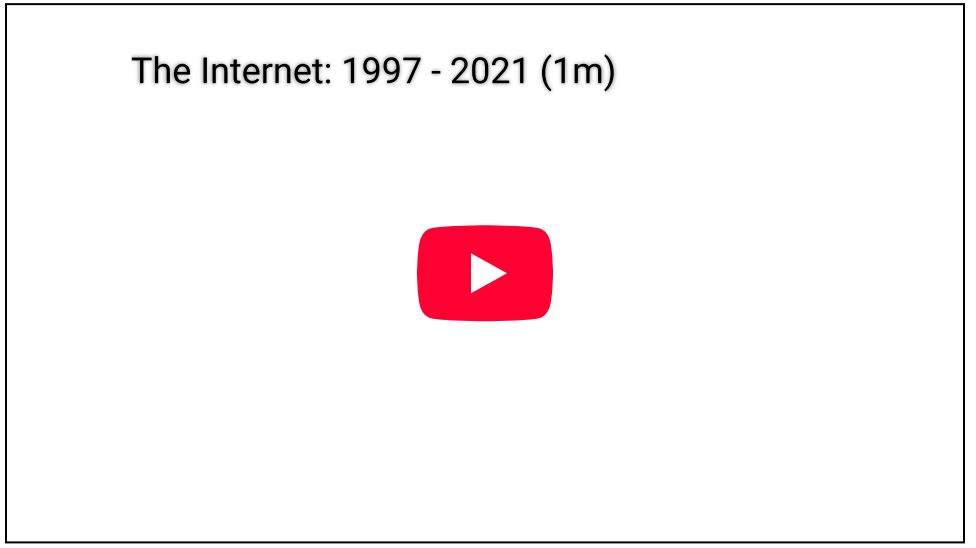
学外(敷地内インターネット側)ルーティング

- ファイアウォール～ルータ～ISP
  - ファイアウォール～ルータ
    (ISP接続ルータ)間はstatic
  - ISP接続ルータ～ISP間もstatic
    (この図ではdynamicの必要が無い)
- 図中一番上のインターネットの**ごく一部**がISP(大学の契約先ISP)に相当しています



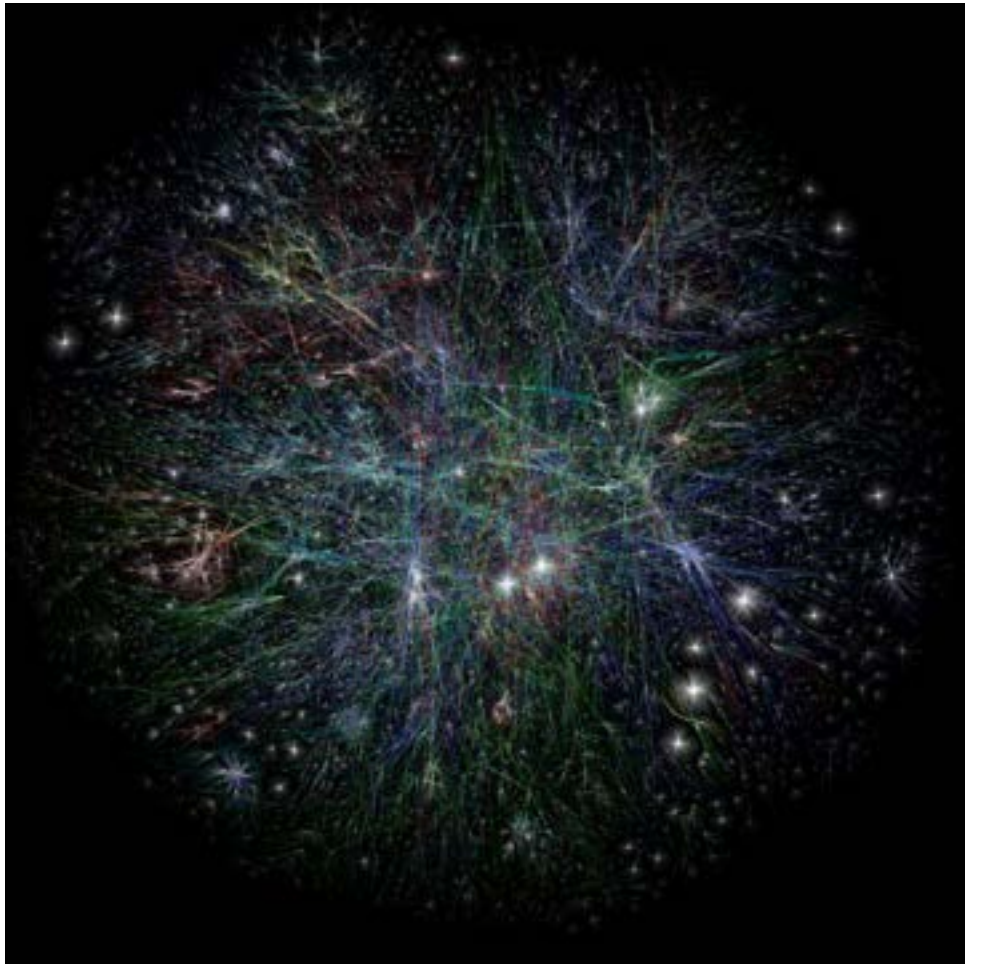
(脚注) さて、次頁ではISP内のdynamic routing(Internetの正体)へ進むことにしましょう

インターネットは数万のプロバイダの集合体です



“The Internet: 1997 - 2021 (1m)” by Barret Lyon, CC-BY 3.0

ルーティング情報をもとに可視化したもの
(生物的なイメージ?生命の樹?)

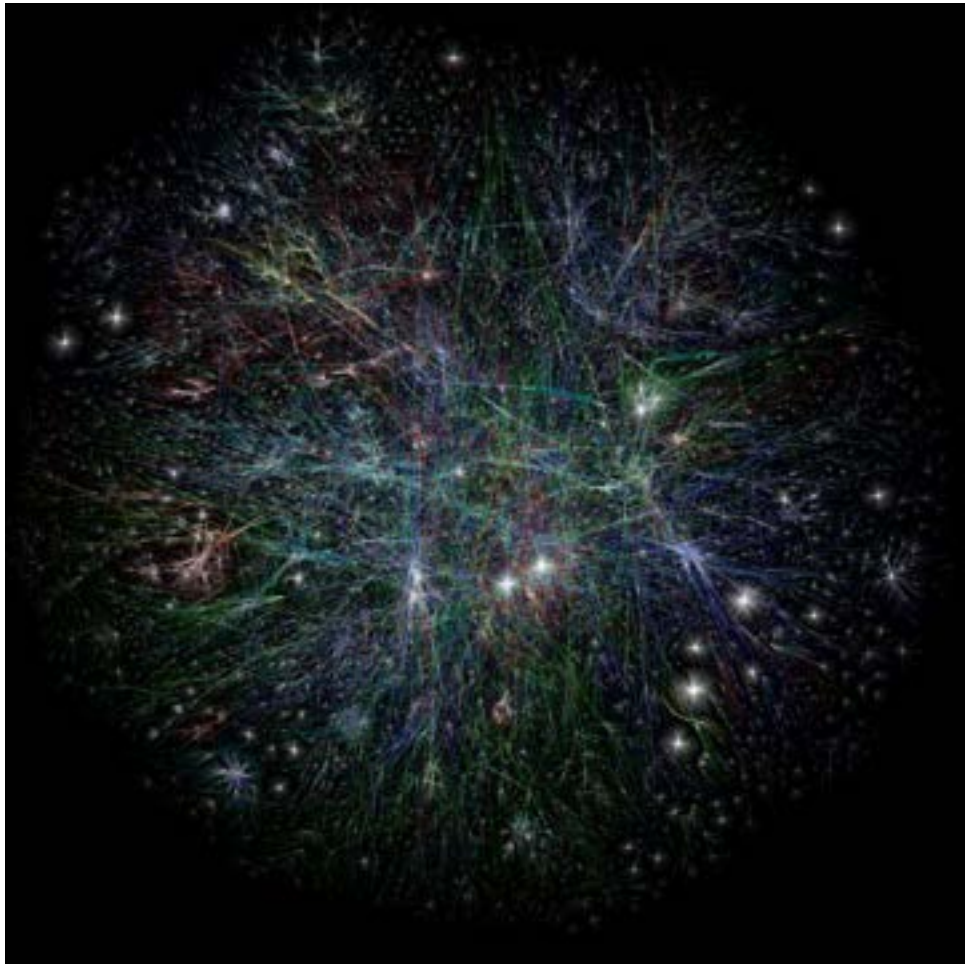


The Internet 2003 by Barret Lyon, CC BY 4.0

インターネットは数万のプロバイダの集合体です

$$\text{インターネット} = \sum_{AS} \text{プロバイダ}(AS)$$

- **インターネットは数万のISPの集合体**
  - **ISP (Internet Service Provider)**
  - 単に**プロバイダ**とも言います
  - ISPのほぼすべてが民間企業
- 大手ISPは**AS番号**を持っています
  - AS番号は**16ビット**
  - 32ビットAS番号というのもあります
    - 16ビットでは足りなくなったのでBGP4(後述)のオプションで拡張

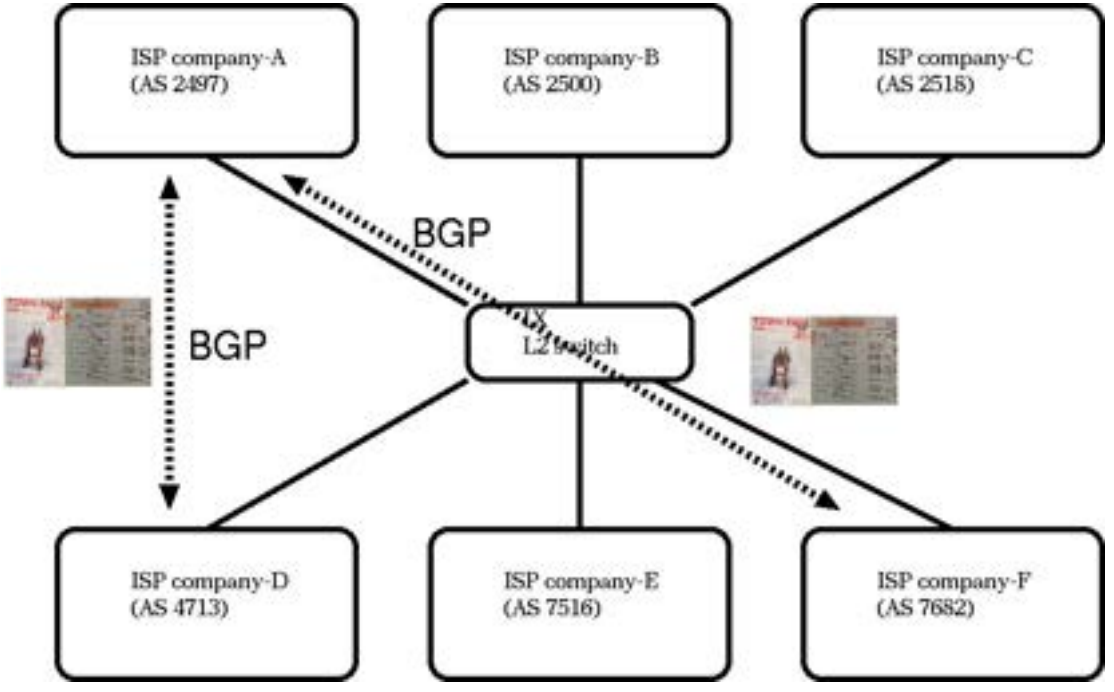


(脚注) BGP情報から推測すると、AS番号を持つ大手ISPの総数は約8万？

インターネットルーティングアーキテクチャ: IX

$$\text{インターネット} = \sum_{AS} \text{プロバイダ}(AS)$$

- 大手ISP同士は**IX**という施設に各自のルータを持ち寄り相互接続しています
  - IX = Internet eXchange
  - **IXの実態はL2スイッチの提供**です
  - 各ISPが自社の責任で
    - 各ISPがIXに自分のルータを設置し、
    - ISP〜IX間の回線調達もします



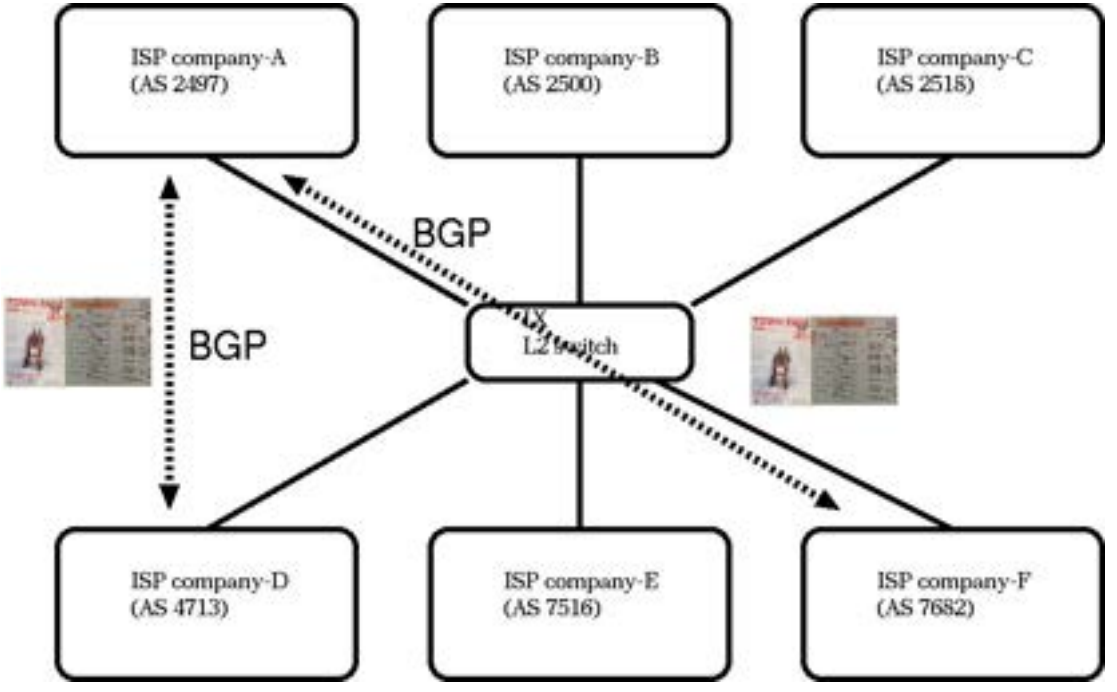
(脚注1) BGPで渡す経路情報は、 自ASのIPアドレス一覧(と付随するAS情報) = いわば自ASの電話帳です

(脚注2) IXではISPのルータ同士がBGP4で経路情報を交換します。 どのISP間で経路を交換するか（ピアリング）？は別途ISP間で話し合います (IXに来ているISP間でfull meshではありません)。 ピアリングには会社間の力関係や政治力学、お金など様々な要素が絡みます

インターネットルーティングアーキテクチャ: ISP間

$$\text{インターネット} = \sum_{AS} \text{プロバイダ}(AS)$$

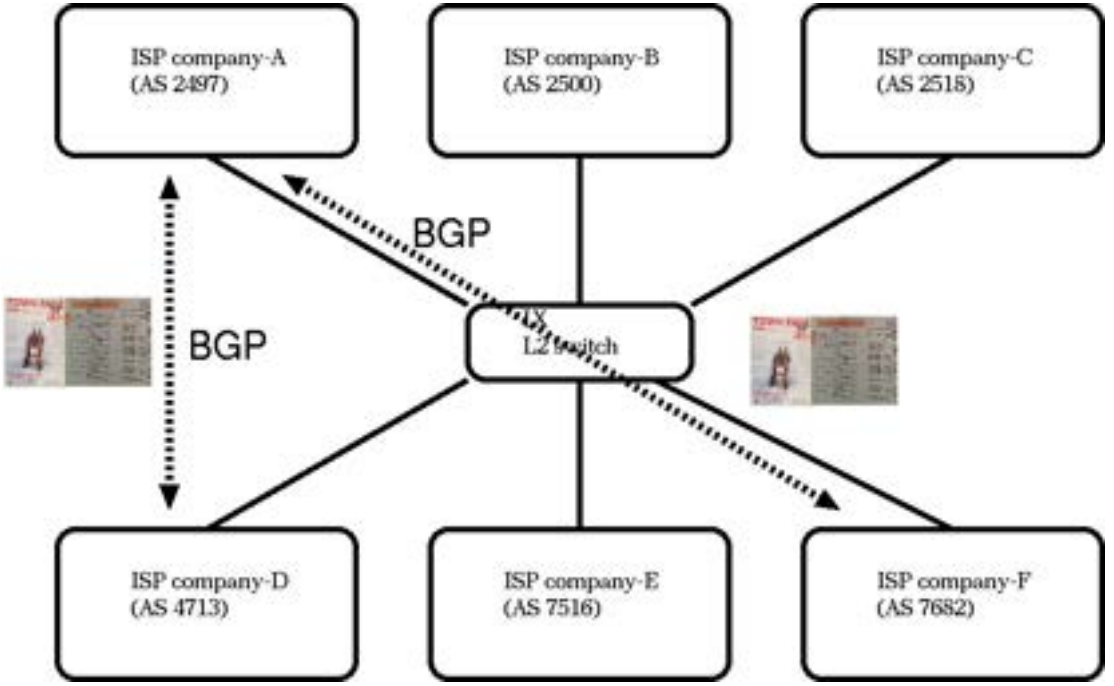
- 大手ISPは互いのルータ間でBGP4を使い経路情報を交換します
- **BGP4** = Border Gateway Protocol 4
  - 4 は version 4 です
- ルータ間のBGP接続(179/tcp)を設定することを「BGP **ピア(peer)を張る(ピアリング(peering)する)**」と言います (図の点線部分)



(脚注) BGPで渡す経路情報は、 自ASのIPアドレス一覧(と付随するAS情報) = いわば自ASの電話帳です

インターネットルーティングアーキテクチャ: ISP間

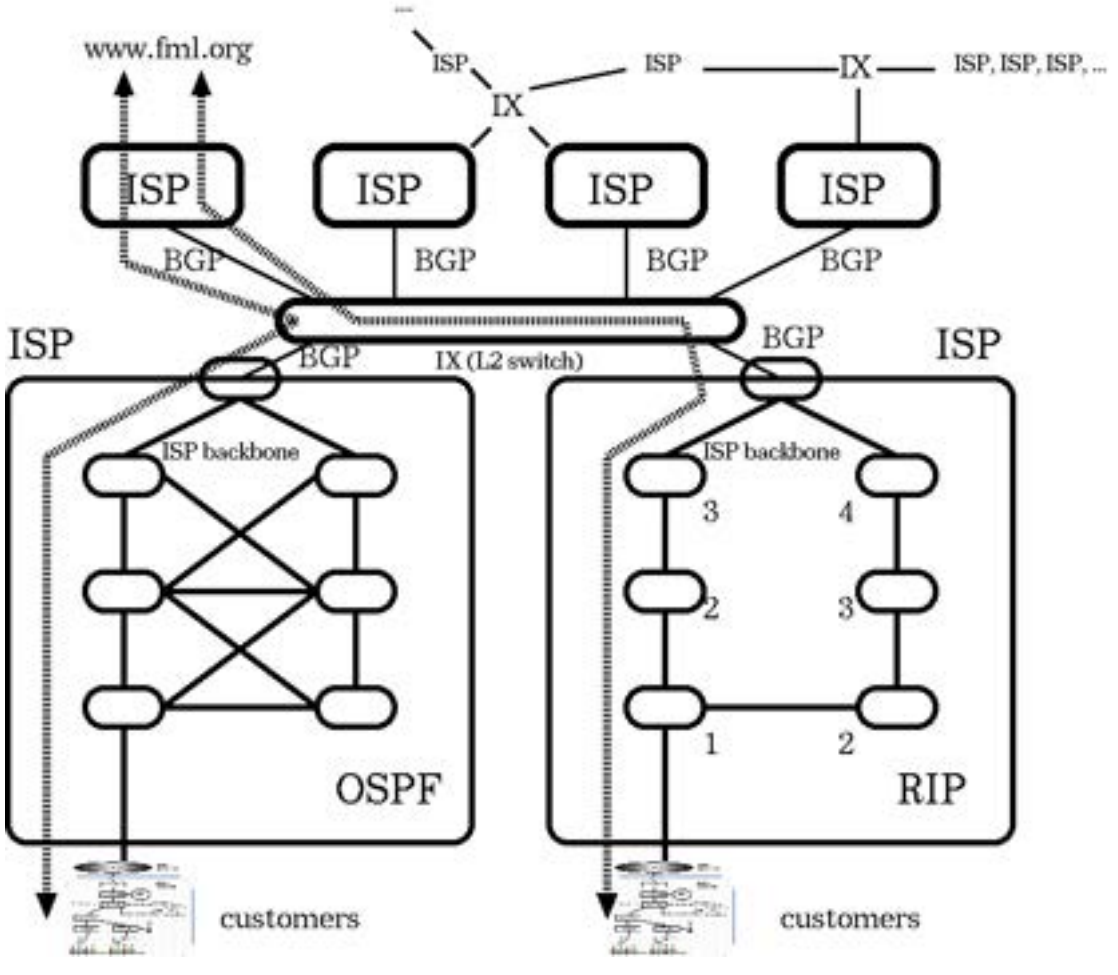
- IXにあるルータの動作
  - つまりISPの出口のルータの動作
  - どのISPへパケットを転送すればよいのか？
  - BGPで集めた情報を元に考えます
- **パスベクトル型**: 基本戦略は「**目的地までに通過するASの数が少ない経路の選択**」です (後述するディスタンスベクトルの応用と考えてよい)



インターネットルーティングアーキテクチャ: ISP内外

$$\text{インターネット} = \sum_{AS} \text{プロバイダ}(AS)$$

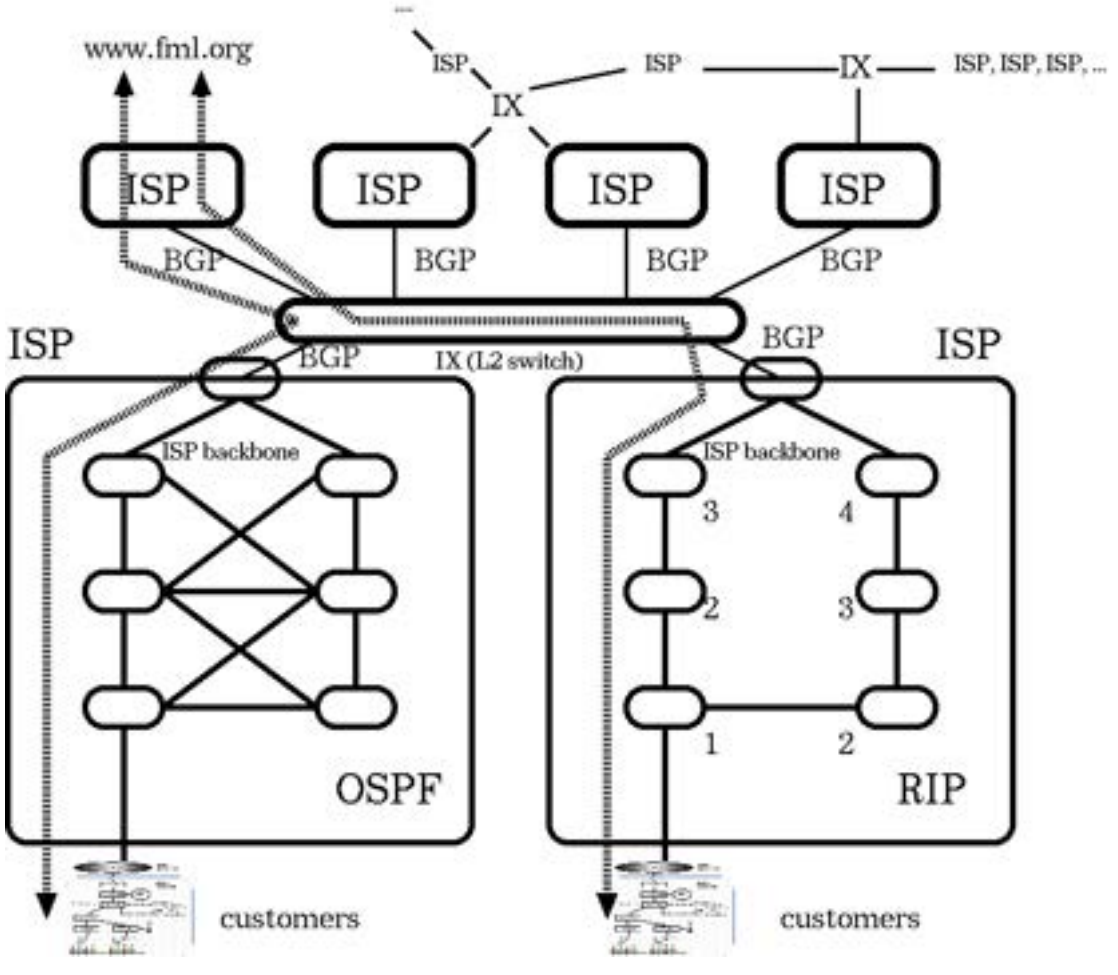
- ISP間のBGP peeringは比較的タイムスケールの長い処理と思ってよいですが
- **ISP内**でのルーティングでは**高速な(秒単位の)障害回避**が必要です
  - ISP内のルーティングには **OSPF**や **IS-IS**を使うことが普通です。 ちなみにOSPFはIETF、IS-ISはOSIの規格です
  - 商用インターネット以前の古えの時代には **RIP** という単純なルーティングで運用できていました (例:かつて日本全体をRIPで運用:-)



(脚注) 図の下側にあるISPという箱の中の説明です。 左側はOSPF、右側はRIPで運用されている想定になっています

インターネットルーティングアーキテクチャ: ISP内

- RIP (Routing Information Protocol)
  - **ディスタンスベクトル**型
  - 複数経路がある場合、目的地までのルータの数が少ない方を選択(図(右))
- OSPF (Open Shortest Path First)
  - **リンクステート**型
  - 参加している全ルータが定期的に情報交換し、 互いの**状態**や構成について知っています。 そのため障害時には高速な経路変更が可能(図(左))
  - 基本的にfull meshを組んで情報交換するので、 ルータ数が増えると急激に計算量が増えます(O(N^2))。 そのため巨大なネットワークでは使えません



(脚注1)OSPFはstatefull (脚注2)RIPは切替が遅い (脚注3)雑なたとえをすれば、RIPはUDPぽくて、OSPFはTCPぽいイメージ

インターネットルーティングまとめ

ケース	ルーティングプロトコル
イントラネット	基本は静的(static)、大きな組織ではRIPやOSPFで動的経路制御あり
ISP内	OSPFやIS-IS、ある程度大きいとBGP、小規模組織ならRIPもありうる
ISP間	BGP

- OSPF(IETF)やIS-IS(OSI)
  - 賢くて高速な判断が出来るかわりに負荷は高い傾向があります。 よって巨大なネットワークでは使えません。 そのため大きなISPでは、 ISP内もBGPでミニインターネットのように運用することがあります (e.g. BGP confederation)
- BGP
  - 基本戦略は簡単ですが、実際には... (脚注を参照)
    -> いろいろあるけど、でも、ルーティング屋さんは楽しそうにみえますね〜:-)?

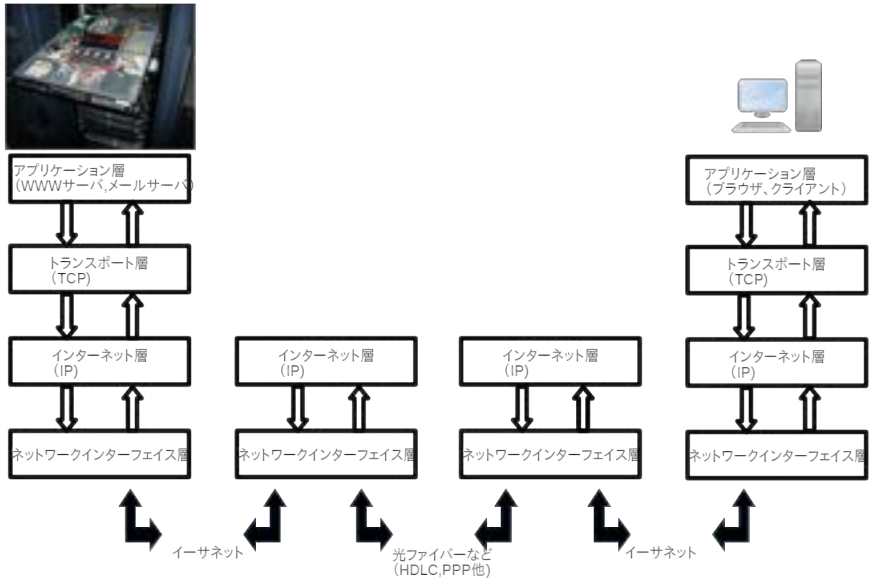
(脚注1) 実際の運用では、 会社間の力関係、 大域的なISPの接続関係、 回線やピアリングの費用など、 多くの要素を考慮し、複雑な条件を満たすよう設定することになります (脚注2) 資格よりBGPの経験値のほうが重要（現場の意見）

情報技術応用特論 第04回
TCP/IP(4) ネットワークインターフェイス層

Copyright (C) Ken'ichi Fukamachi <fukachan@fml.org>, 2021-2025. CC BY-NC-SA 4.0

全体像（郵便たとえ話、半ふりかえり）

- アプリケーション層：手紙を開けて**内容を読んで、（必要な処理をし、）結果を返します**
- トランスポート層： **書留郵便**。 郵便屋さんは確実に相手に届けることが必要です。 お届けの際には受領確認のハンコ(ACK)をもらいます
- インターネット層： **普通郵便**。 郵便屋さんは各家のポストに投函して終了です。 相手が受け取ったか？は関知しません。 そもそも（投函以前に）配送途中で行方不明もありえます
- ネットワークインターフェイス層： **回覧板方式**。 郵便物をご近所もしくは隣の町内へ渡していく方式。 目的地まで所謂バケツリレーです

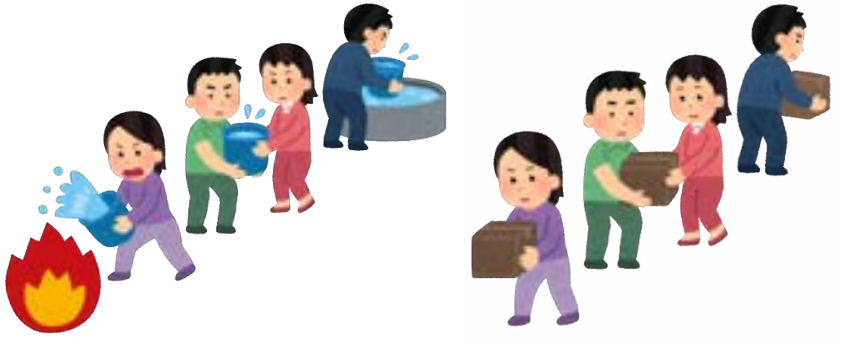


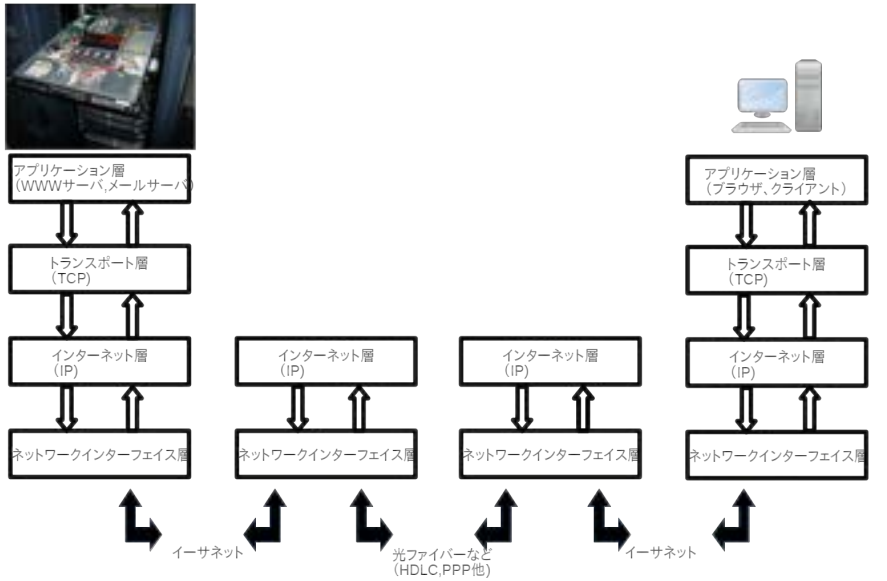
(脚注1) いつもの郵便たとえ話でいきます (脚注2) アプリケーション層の（）内「処理をする」はデジタルデータ転送の話ではなく、 実際の処理をおこなうWeb APIサーバ（アプリケーション）の動作の話です。だから（）でくくっています

イーサネットの動作原理

ネットワークインターフェイス層はバケツリレー

- ネットワークインターフェイス(以下NI)層を代表するプロトコルが**イーサネット(Ethernet)**です
- NI層は、**となりあうネットワーク機器間のやりとり**を規定しています
  - 隣同士と**バケツリレー**です(図上側)
  - 右図の一番下では上の層と異なり隣同士の矢印が書いてあります。 これは**隣り合う機器のやりとり**を意味しています





(脚注) IP層より上では世界のどことでもEND-TO-END通信できることが目的です(再掲)

イーサネットの動作原理の全体像

- 共通のバスに全デバイスが接続されています
- だから**一斉に会話（通信）**できます
    - 【応用】目的地は誰ですか？を全デバイスに尋ねる（～へ転送したいのですが、誰が渡すべき相手（目的地）でしょうか？）
    - 【応用】～サーバさんは誰？を全デバイスに尋ねる
- 素のままでは効率よく一斉通信できませんけど…

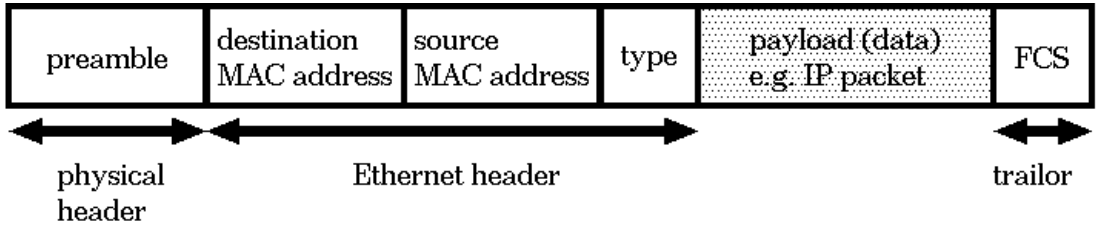
(脚注1) 図は同軸ケーブルで各デバイスから電気信号を流す想定です。 一斉に通信をする＝各自が勝手に電気信号を流す -> 全電気信号が重なり合う状態になります。 それを分解して読むとか出来ません。 【電磁気学】マクスウエル方程式は線形方程式だからね、ok?

(脚注2) 同軸ケーブルの写真は、付録を参照 -> こちら

イーサネットフレーム

- NI層のデータ塊はパケットではなく**フレーム**と呼びます
  - ヘッダ(header) ... 先頭部分
  - トレイラ(trailor) ... 最後尾
- 下図(左)はフレーム全体、 下図(右)は同じくフレーム全体ですがイーサネットヘッダ詳細版

- イーサネットつまりネットワークインターフェイスの識別子が**MACアドレス**
- **MACアドレス**
  - **48ビット**の「イーサネットの識別子」
  - 工場出荷設定で一意
  - **48ビット**を**8ビットごとに:区切り16進数**表記
    例: 12:34:56:78:90:AB



(脚注1) 図は覚えなくて大丈夫です。 図は両方ともイーサネットフレーム全体を表しています。 図(右)ではイーサネットのヘッダとトレイラを拡大し具体的に書いてあります。 どちらの図でも**左側が先頭**(headerが来る側)です

(脚注2) 仮想環境では管理システムがMACアドレスを生成・割当します (また本来は一意のはずですが書き換えるツールとかも...;-)

(脚注3) イーサネットは物理的な動作も規定しているのでOSIモデルではレイヤー1と2にまたがります

一斉通信できることを応用する

ブロードキャストの原理

- **同一セグメントにあるホスト(群)へパケットを一斉に送信したい**場合に使うIPアドレスが**ブロードキャスト**でしたね？
- MACアドレスが**FF:FF:FF:FF:FF:FF**宛のイーサネットフレームを、 ネットワーク機器は**必ず受信しなければなりません**。これを利用してブロードキャストを実現します

(脚注) 左上のPCからイーサネットフレームを送信すると、 全PCが受け取っている様子を書いているつもりです。 ヤギさんが封筒(フレーム)を運んでいます。 ヤギさんが各PCに**同時に届けている**様子までは書いてないんですけど、 手紙の絵で察してほしい:-)

IPパケットとブロードキャストの関係

- インターネット層でIPパケットをブロードキャストアドレスへ送信すると、 **NI層では、 そのIPパケットを包んだイーサネットフレームを作り、FF:FF:FF:FF:FF:FF宛に送信**します
- MACアドレスが**FF:FF:FF:FF:FF:FF**宛のイーサネットフレームを、 ネットワーク機器は**必ず受信しなければなりません**
- 各デバイスで、ブロードキャストを受信後、中身を見て、適宜すてるなり反応するなり各自に対応してもらいます

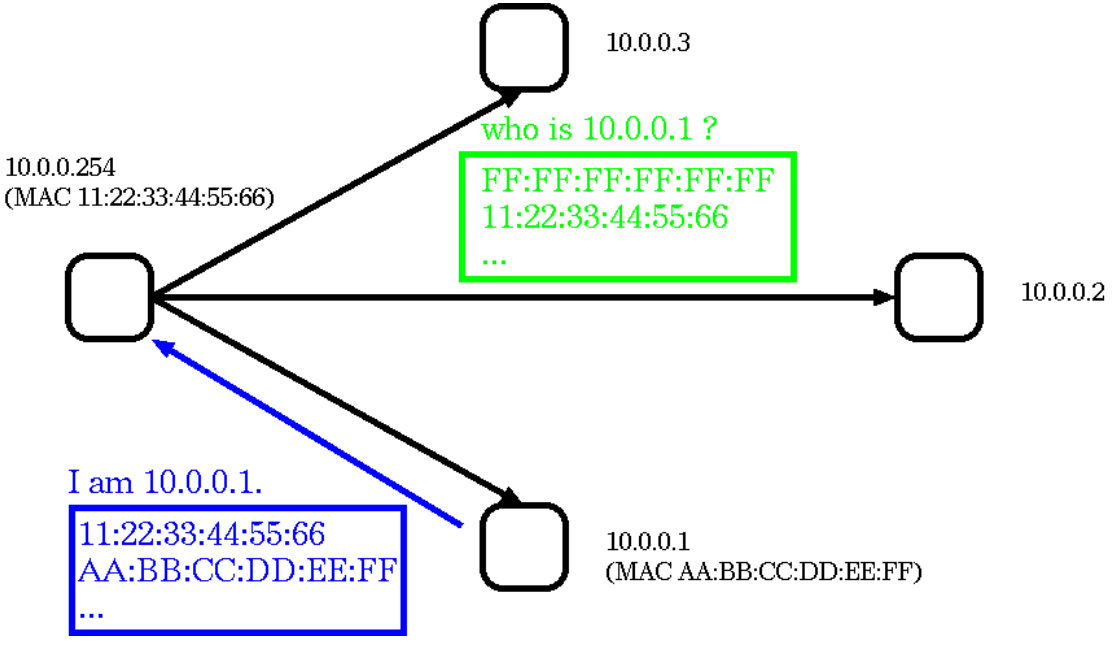
応用: ARP (Address Resolution Protocol)

- NI層では通信相手(のMACアドレス)を自動探索
  - **「ブロードキャストで全ホストに尋ねれば該当ホストから答えがもらえる」**ことを利用します。 このしくみが**ARP**です
- 手順
  - ARPパケット(「IPが〜の方こたえてください」)をブロードキャスト
  - 該当するIPアドレスのホストが返事を返す（この返事のイーサネットフレームの送信元MACアドレスが探していた情報 (**該当ホストのMACアドレス**)です)

(脚注) 毎回ARPするのは無駄なので得られたIPは一定期間記憶します(arp cache)。 初心者は障害対応で機器を入れ替え時にcacheを忘れがち。 **知識ではなく、 どういう理屈で動いているかが頭に入っていけないといけない典型例**

応用: ARP (Address Resolution Protocol)

- ブロードキャストでARPを送信
  - 送信先FF:FF:FF:FF:FF:FF
  - 送信元11:22:33:44:55:66
    (自分のMACアドレス)
  - ARPパケットの中身には「IPアドレスが
    10.0.0.1の方は誰ですか?」と書いてあります
- 10.0.0.1のデバイスが返事をします
- 返事のイーサネットフレームは
  - 送信先11:22:33:44:55:66
  - 送信元AA:BB:CC:DD:EE:FF
    (知りたかった**該当デバイスのMACアドレス**)



(脚注) ふつうARPはカーネルのネットワーク機能の一部です

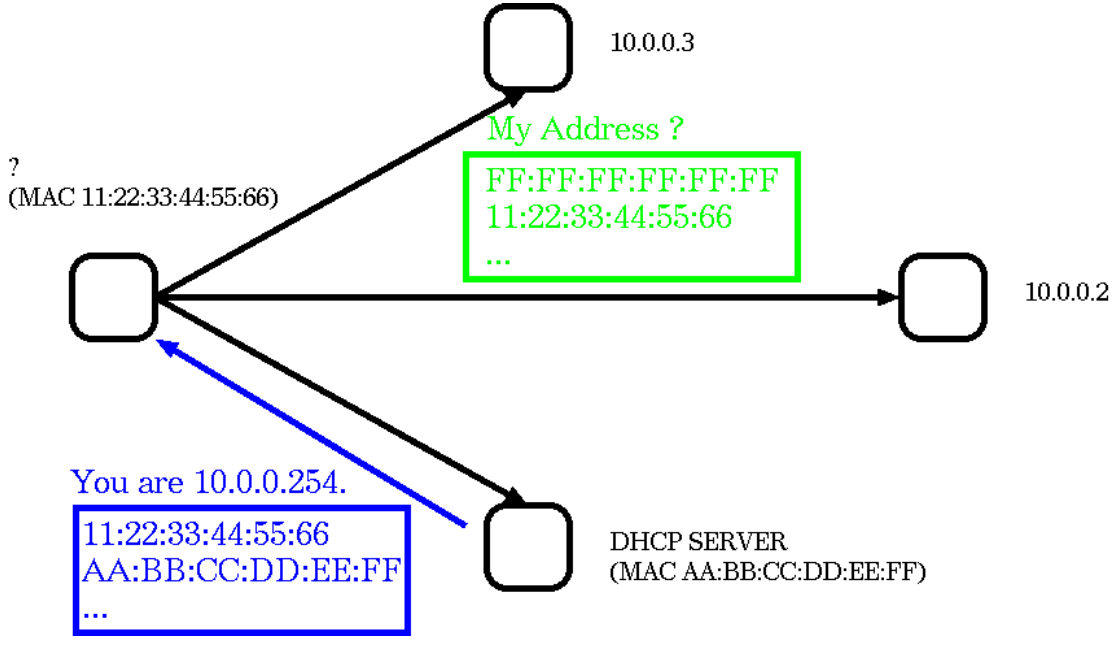
応用: DHCP(Dynamic Host Configuration Protocol)

- 自動でOSのネットワーク設定をする仕組み
- PCやデバイスの起動時に、ブロードキャストを利用してOSの設定情報をDHCPサーバに問い合わせます。
  - **起動時のPCは自分のIPアドレスすら分かりません。** もちろん問い合わせるべきサーバのIPアドレスやMACアドレスも知りません
  - そこで**ブロードキャスト**を使います

(脚注1) PC教室や家庭、無線LANなどあらゆるところで動作しています。 DHCPのおかげでOSを手動設定せずに利用できるわけです

応用: DHCP(Dynamic Host Configuration Protocol)

- ブロードキャストでDHCPリクエストを送信
  - 送信先FF:FF:FF:FF:FF:FF
  - 送信元11:22:33:44:55:66
    (自分のMACアドレス)
- 全デバイスが、このリクエストを見ますが、
  DHCPサーバが自分宛だと判断し、
- DHCPサーバが返事をくれます
  - 送信先11:22:33:44:55:66
  - 送信元AA:BB:CC:DD:EE:FF
    (サーバのMACアドレス)
  - **DHCPサーバからの返事(パケット)にネットワーク設定情報が含まれている**ので、 それらをPCに設定します （e.g. IP, network prefix, デフォルトルート, DNSサーバ, ...）



(脚注) OSの起動シークエンスの途中で、DHCP clientが起動され、それがブロードキャストで問い合わせを行います。 ISCのdhcpが古典ですが、 LinuxではDNSマスカレードというソフトウエアもあります。 最近では、また別のソフトウエアを使い始めているような？

素のままでは、うまく動きません
(通信の調停と効率化)

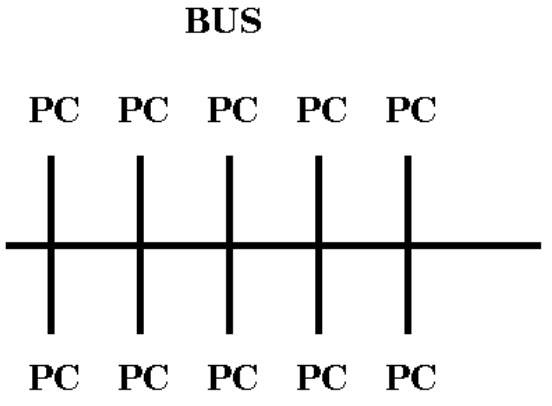
CSMA/CD(Carrier Sense Multiple Access with Collision Detection)

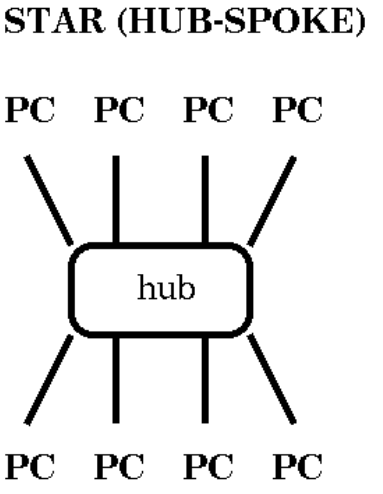
- 複数のデバイスが同時に送信すると、電気信号が重なりあい、解読できません;-)
- よって、ある瞬間に送信してよいデバイスを一台に限定することにします！
- **送信するホスト(一台)を決める**には**CSMA/CD**というプロトコルで**調停**します
  - CS = 他に通信しているホストがいないことを確認、信号の感知(CS)
  - MA = 複数のホストが共有している回線を使い通信(=access)
  - CD = 衝突(Collision)の検知(Detection)

(脚注1) 物理な話をしていますがNI層は、そういうものです (脚注2) 一台に限る！とか当たり前な話をしていますね、でも、そういうものです (脚注3) /は発音しないか with (with を w/ と略しますね) (脚注4) ちなみに無線LANは**CSMA/CA**という方式です

転送効率の向上、スター型への変更

- **バス型**(図左)では、ホストが増えるほど衝突が増えていきます (CSMA/CDの理屈を思い出せ！)
- 商用インターネット時代に入ると、イーサネットの規格も10BASE-Tが主流になり、 それ以降は**スター型の接続**になります(図右)
- どの型でも、ブロードキャストの動作原理は同じですが、 スター型の中心に置く機材(図のHUB、現代では**スイッチングハブ(スイッチ)**) が通信の様子を学習して、**無関係なホストへ通信を見せない**ようにしています。 つまり**衝突を回避**しているため、 **転送効率が向上**しています



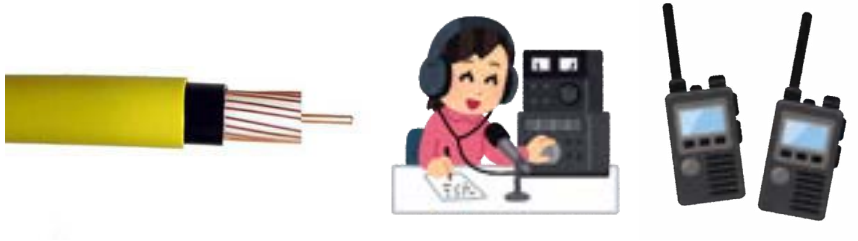


(脚注1) スター型はハブ-スポーク型(自転車の車輪)とも言う (脚注2) 図のHUB部分に使うネットワーク機材の名称と役割も調べてみよう (基本情報に出ます！): ハブ、リピーター、ブリッジ、ラーニングブリッジ、スイッチングハブ(=スイッチ)は、 何が違いますか?

参考: ブロードキャストが動作する理由

ブロードキャストが動作する理由

- なぜブロードキャストが出来るのかというと、もともと接続している全ホストに通信が見える想定だから。ここに注目
- もともと一つの同軸ケーブル(図左下の黄色いケーブル)に全ホストが接続しているので、 全デバイスに電気信号が見えています
- イーサネットのアイデアには無線ネットワークのalohanetが影響を与えているそうなので、 通信が全員に見えるという想定に納得ですよね



(脚注) alohanet = 「アロハ〜」で分かるとおり、ハワイの(研究用)無線ネットワーク。 N. Abelsonが「ハワイでサーフィンしたいなぁ、ハワイの大学に転職するか〜」と、 ハワイに行って無線ベースで作りあげたネットワークのこと。 当時(ARPA初期)のIMP(ルータの先祖)はコンピュータ直結型の違う原理で動いています。 では一つのネットワークに複数のコンピュータをつなぐとどうなるの? を実際にやってみた -> かなり困るぞ...どう解決する?という話です。 R.M.MetcalfeはAlohanetを見てイーサネットを開発しました

ケーブル類の紹介

10Base-5 (通称イエローケーブル)。 最初に使われていた同軸ケーブルで、太くて引き回しが大変

10Base-2 (5の後に使われていた同軸ケーブル)

ONUと光ファイバー(single mode)。 これは家庭用フレッツの機材、 業務用とは違います

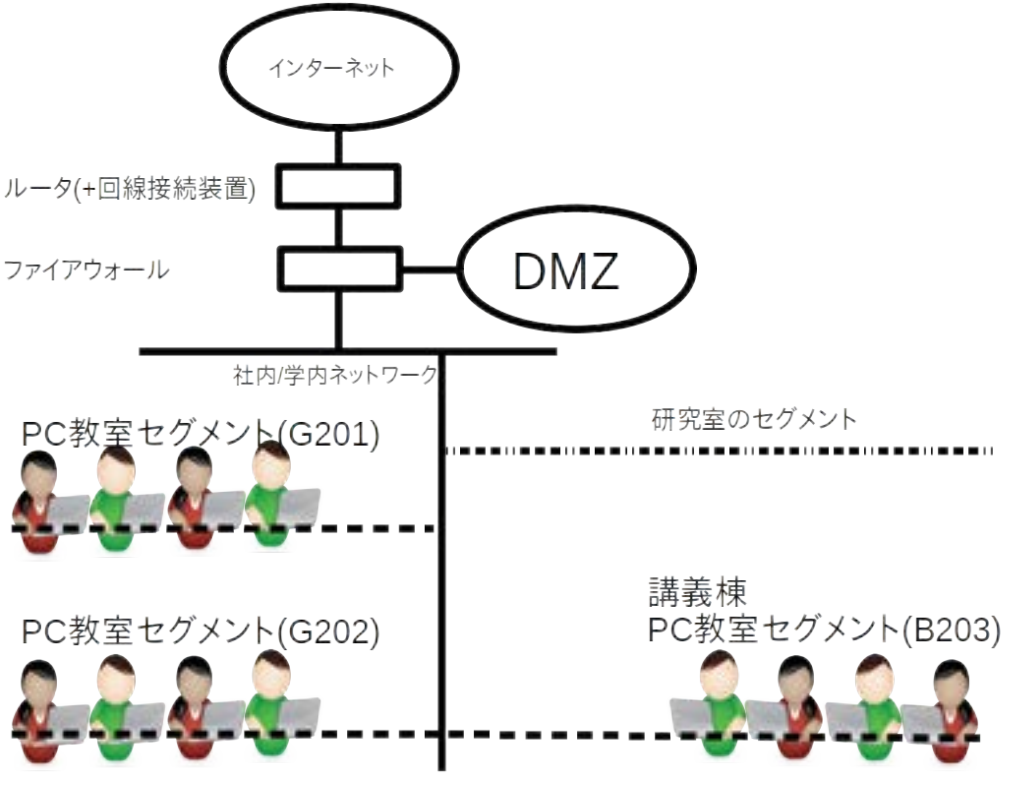
参考: Bフレッツや学内の物理構成について少々

物理構成

後半の設計編で必要な知識を少し先出しで紹介

実際の物理構成(概略)

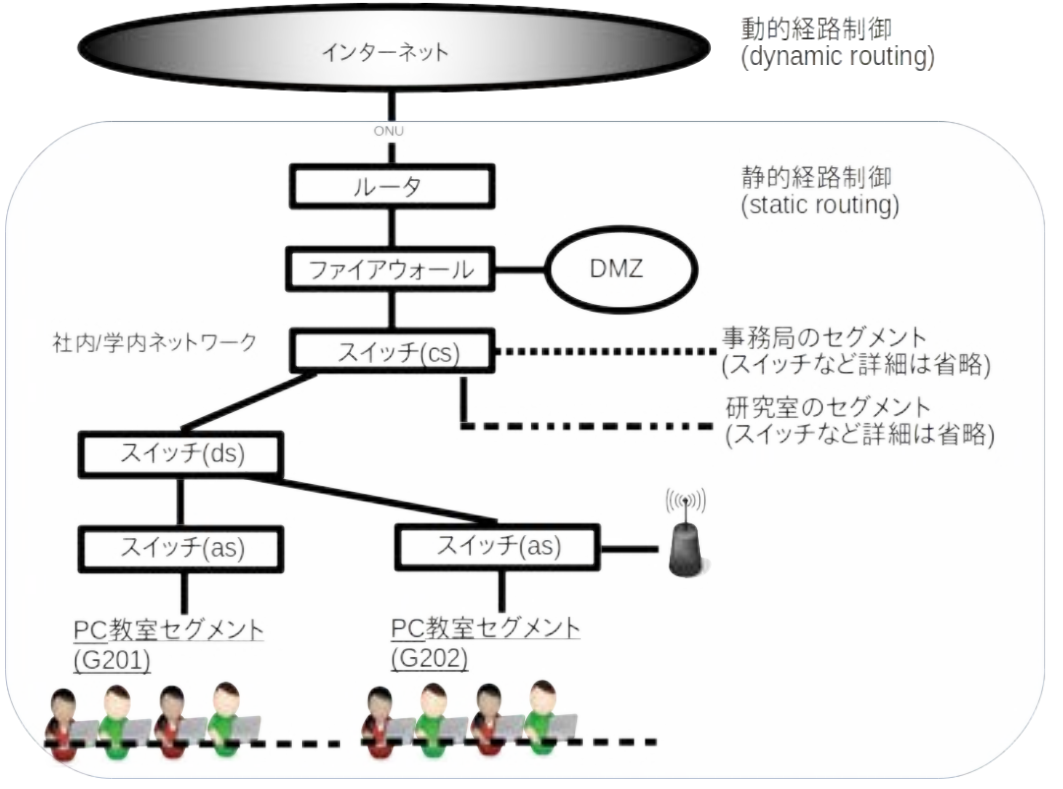
- ネットワーク構成の回に、 右の図を「学内の**論理構成**図」として使いました
- 実際の**物理構成**はどうなっているの?という解説を少しだけします
- 用語
  - ネットワーク機器(既出)
  - 機器同士の配線に使うもの
    - イーサネット
    - 光ファイバ (single-mode,長距離)
    - 光ファイバ (multi-mode,短距離)



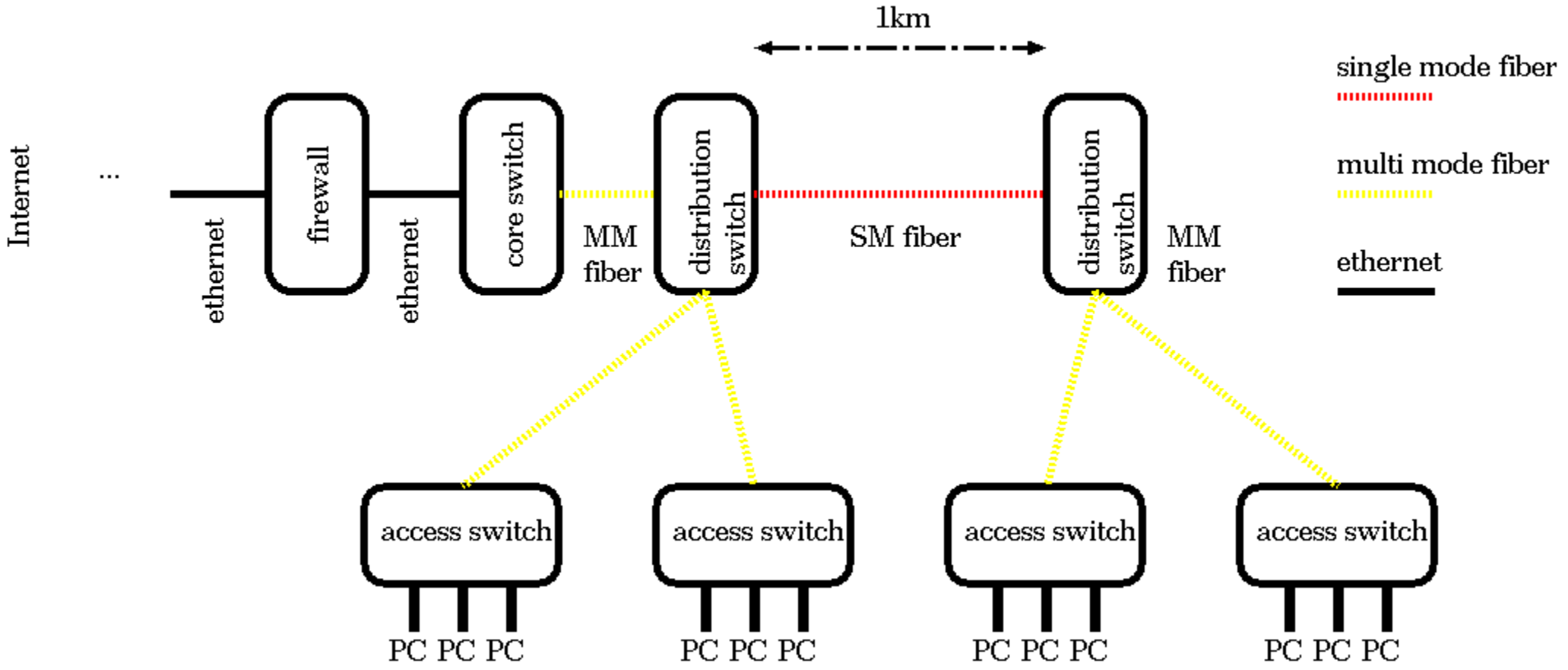
(脚注) 光ファイバの詳細は他の授業に譲ります。 singleとmultiは反射の仕方など動作自体が異なります。 実務的にはファイバの曲げやすさが違うので(折らないように)注意が必要です

ネットワーク機器の構成(復習)

- ルーティングの回のスイッチ群
  - cs (core)
  - ds (distribution)
  - as (access)
- 次頁では、実際の cs, ds, as 間の配線の使い分け方を説明



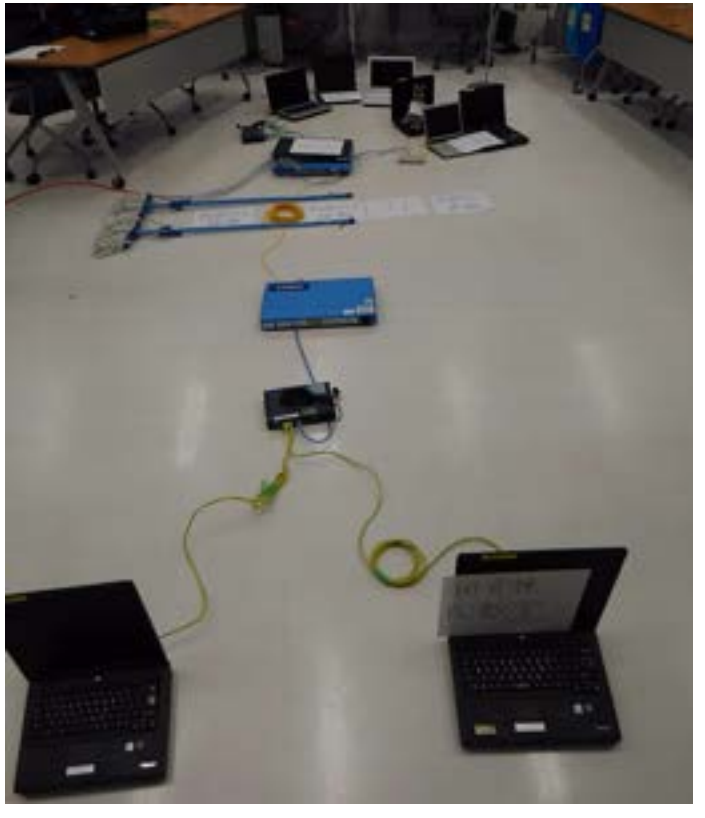
実際の光ファイバ配線構成図



(脚注) 春学期後半の設計編の「設計ガイド」より

物理構成(イメージ)図

- 物理的にPCとスイッチと光ファイバーとケーブルと…いろいろ並べて写真と動画をとりました (次頁の動画も堪能してね)
  - 構築するだけで5人で2時間くらい使いました (本来は、このあと設定もあるので全部やったら半日仕事)
- いやぁ物理作業って大変ですね;)
  - ブラウザの上でクリックするだけのクラウドは楽ですね?
  - でも、**クラウドを構築して売っている側の人たち**がいるから使えることを忘れずに



(脚注1) 春学期後半の設計編の「設計ガイド」より抜粋 (脚注2)「クラウドの上で何か作って売る職業」と「インフラを作って売る職業」どちらが**安定して儲かる**の?って、 それはAmazonの売上を見れば一目瞭然だよね?

物理構成の解説動画(再生リスト)

(左)E208のBフレッツを例に「ルータ～L3スイッチ～ファイアウォール」の説明 (右)学内全体のイメージ
ちなみに再生リストの3番め以降は、光ファイバなどの解説動画(他作)です。
8番目にあるGoogleの気球ネットワークは良いアイデア！ 人工衛星より低遅延・低価格なので普段から使いたい。 災害時などの緊急展開にも役立ちます。 でもイマイチ流行してないみたい...

再生できない方は こちら からどうぞ