コンピュータネットワーク

1　コンピュータネットワーク概観

1.1　コンピュータネットワークの構成

1.1.1　インターネットの構成

- エンドシステム (end system)
  - ネットワークに接続しているコンピュータ
  - ホストともいう
  - クライアントとサーバに分類されることが多い
- クライアント
  サービス要求を出す
- サーバ
  サービス要求を待ち受ける
- 通信リンク
  光ファイバ、電波、同軸ケーブル、etc...
- ルータ
  パケットの中継装置
- ここまでがモノ
- プロトコル
  二つ以上の通信エンティティ間でやりとりされるメッセージの形式と順序などを取り決める規約
  ネットワークの機器間でのやり取りにおけるルール

1.1.2 通信サービス

エンドシステム間で情報をやり取りするための仕掛け

- コネクション指向型サービス

  クライアントとサーバは、通信を始める前に相互に制御パケットを送信

  通信前にハンドシェイク

  - 高信頼データ転送

    順序通り、誤りなく伝送

  - フロー制御

    受信側のバッファを溢れさせない

    空きバイト数をフィードバック

  - 輻輳制御

    ネットワークの混雑を防ぐ

  代表的なプロトコルは、TCP (Transmission Control Protocol)

- コネクションレス型サービス

  事前の制御パケットのやり取りなしにいきなりデータを送る

  代表的なプロトコルは

  - UDP (User Datagram Protocol)
  - リアルタイムアプリケーション向き
  - 高い自由度をもつ

  高信頼性が無駄なときや、トランスポート層をアプリケーションからいじることができる (TCP をいじりたければ OS アップデートが必要)

1.2 ネットワークコア

エンドシステム間の相互接続を担うルータ群

1.2.1 回線交換

エンドシステム間の通信のために経路に沿った通信資源 (バッファや帯域の一部) をセッション中常時常時占有

予め通信経路を予約して占有

1.2.2 パケット交換

パケット (oacket)

- アプリケーションレベルのメッセージを分割したもの
- ネットワークコアでの伝送単位

蓄積交換伝送

- 各ルータで到着したパケットを一旦、バッファへ格納
- 予め定められた順序 (先着順など) に従い、順次パケットを伝送

1.2.3 回線交換 vs パケット交換

パケット交換の長所

- 伝送容量を効率的に利用可能
- 実装が容易 ルータに送るだけ

回線交換の長所

- 通信品質が安定、リアルタイムアプリケーション向き
  通信を占有するため、安定性が求められるとき

表 1　回線交換とパケット交換の長所

	回線交換	パケット交換
長所	通信品質が安定 リアルタイムアプリケーション向き	伝送容量を効率的に利用可能 実装が容易

1.2.4 遅延とパケット損

各ノード (ルータ) における遅延

- 処理遅延: パケットのヘッダを読み、出力リンクを決定する時間
  伝送誤りチェックも含む
  (通常 $ns \sim \mu s$ のオーダ)
  パリティチェック等
- 待ち行列遅延: 送信待ちに要する時間
  (通常 $100ms$ 程度まで、バッファサイズに依存)
  LAN 出力ポートに複数の出力が来た際の待ち時間、バッファサイズによってパケットを保持できる (より情報を保持できるが待ち時間も増える)
- 伝送遅延: パケットを通信リンクに送り出す時間
  (リンクの容量を R[bps], パケットサイズを L bit とすると, $\frac{L}{R}$)
- 伝搬遅延: 送信された 1 ビット目の情報が次のノードに到達するまでの時間
  (光速より少し遅め、 $2 \times 10^8 m/s \sim 3 \times 10^8 m/s$ 程度)
  ex.) 衛星通信など。送信元と送信先の距離によって変化

パケット損

待ち行列 (バッファ) に入ることのできるパケットの数は有限

⇒ パケット損が生じる

一般に、「バッファサイズ大」 ⇔ 「待ち行列遅延大 かつ パケット損小」
というトレードオフが存在
パケットの損失は TCP の場合、輻輳制御によってサービスが低下する

1.2.5 ルーチング

送信ホストは終点ホストのアドレス (IP アドレス) をパケットのヘッダに書き込んで送信
ルータは、終点アドレスを出力リンクに対応付けた ルーチングテーブル を持ち、検索して転送
ネットワークグラフの情報を持っていて、適切な経路を返すイメージ？
ルータはコネクション情報を管理しない (ヘッダに書かれたアドレスを読むだけ)
ルーチングテーブルの自動作成

⇒ ルーチングプロトコル

1.2.6 ネットワークのネットワーク

インターネット (the Internet) は、複数の ISP 同士が階層的に接続することで構成

ネットワークのネットワーク (Network of network)

イントラネット (企業内などの閉じたインターネット)、ARPANET から始まったインターネットが大きな塊になっていった

- アクセス ISP: DSL, FTTH, Wi-Fi, セルラ, ビジネス LAN などによるエンドシステムからのアクセスを提供

  DSL: 日本では ADSL(asymmetric DSL)
- Tier1 ISP: 他の Tier1 ISP および下位 ISP と接続し、国際的エリアをカバー

  日本では NTT コミュニケーションズとソフトバンクの 2 社
- Tier2 ISP(広域), Tier3 ISP(地域):

  Tier2 ISP は、グローバル通信を行うとき、Tier1 ISP を介してトランジット通信

  上位 ISP はサービスプロバイダ、下位 ISP はカスタマーという関係となり、トラヒック量に応じて料金を課す (従量課金)

  ISP: Internet survice provider, 通信の企業のことみたい。kddi → so-net → user みたいな感じ？
- ピアリング:

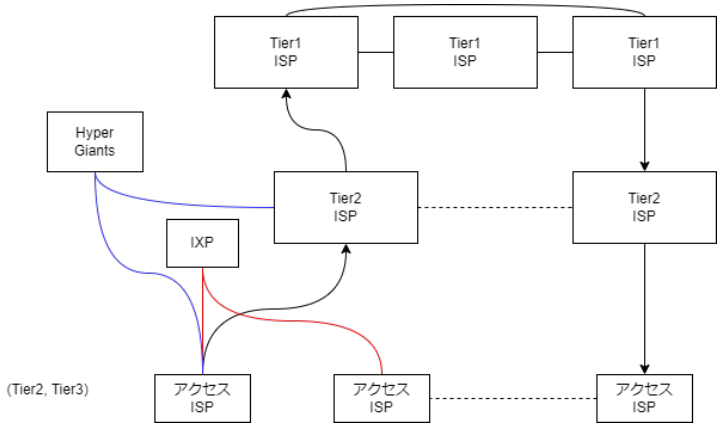
  同層 ISP 間で接続すること。

  上位 ISP へのトランジット料金の支払いを軽減
- 相互接続点 (PoP: Point of Presence): ISP 間が接続するときの接続点 (複数のルータから構成)

  複数の事業者間で通信するには物理的にルータが繋がっている必要がある
- IXP(Internet Exchange Point): ピアリングする ISP がつなぎこむ箇所を提供する、独立した組織

  PoP を提供する事業者
- コンテンツプロバイダ:
  - Google など、非常に大きなリソースを持つサードパーティ (Hypergiants などとも呼ばれる)
  - Tier2, Tier3 ISP と直接ピアリングを行う

    従量課金がなくなるため、下位 ISP としても Win-Win



1.3 プロトコル階層とサービスモデル

1.3.1 階層化アーキテクチャ

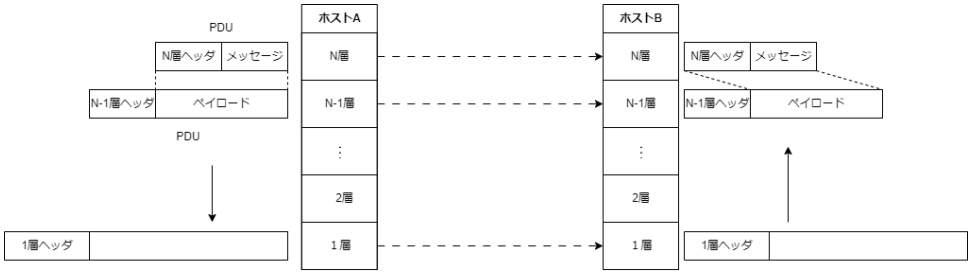
インターネットは極めて複雑なシステム

プロトコルならびに、それを動作させるハードウェア、ソフトウェアを階層化して設計
⇒ ネットワーク設計の複雑さを軽減し、各構成要素間の役割や関係を明確にする
モジュール化、個別に設計

各プロトコルはある一つの階層 (layer) に属する

第 n 層のプロトコルは第 n 層同士でメッセージを交換
第 n 層のプロトコルデータユニット (n-PDU): 第 n 層で交換されるメッセージ

- プロトコルスタック (protocol stack): これらのプロトコルが構成する階層全体
  - OSI 参照モデル (Open System Interconnection Refference Model): 7 層構造
  - インターネット (TCP/IP): 5 層構造



ホスト A の第 n 層がホスト B の第 n 層に n-PDU を送信

1. ホスト A の第 n 層がホスト B の第 $n-1$ 層に n-PDU を渡し、ホスト B の第 n 層への送信を依頼
2. 第 n 層は第 $n-1$ 層のサービスを受ける (第 $n-1$ 層は第 n 層にサービスを提供)
3. 第 n 層は第 $n-1$ 層がどのようにサービスを実現しているかを意識しない
   ⇒ 層間のインターフェースが定義されていれば差し替え可能
   抽象メソッドみたい

層が深くなるにつれ、誤り訂正などの処理が行われる

- プロトコル階層化の欠点

- 同じ機能を複数の層が持つ場合がある (誤り制御など)
  データ量的に無駄
- 上位層は下位層の情報を利用できない (柔軟性の欠如)

1.3.2 インターネットプロトコル階層

表 2

		PDU の呼び方
Layer 5	アプリケーション	メッセージ
Layer 4	トランスポート	セグメント
Layer 3	ネットワーク データグラム (パケット)	
Layer 2	データリンク	フレーム
Layer 1	物理	1-PDU

- アプリケーション層
  ネットワークアプリケーションをサポート
  例) HTTP: web
  SMTP: 電子メール
  FTP: ファイル転送
- トランスポート層
  アプリケーションプロセス間のメッセージ転送サービスで提供
  - TCP: 高信頼データ転送、フロー制御、輻輳制御
  - UDP: コネクションレス型サービス
- ネットワーク層
  始点・終点ホスト間でデータグラムの転送を行う IP・インターネットの 3 層プロトコル (IP 層ともいう)
  IP データグラムの定義と、それに基づくエンドシステムやルータの動作を規定
  - ルーチングプロトコル:
    始点ホストから終点ホストまでの経路を設定
    複数のプロトコルが存在
- データリンク層
  ノード間の 1 ホップ の通信を担う
  例) Wi-Fi, イーサネット (有限 LAN) など
- 物理層
  - フレーム内の各ビットを伝送
  - 物理メディア: より対線、同軸ケーブル
    光ファイバ、無線など

ネットワークエンティティ
ネットワークの構成要素 (エンドシステムと中継器)

- 中継器: ルータ (3 層まで)
  ブリッジ、スイッチ (2 層まで)
- エンドシステム: 5 層すべて

2　アプリケーション層プロトコル

2.1　基礎

2.1.1　アプリケーション層プロトコル

- ネットワークアプリケーション: ネットワークの使用を前提とするソフトウェア
  例) Web アプリケーションは、ブラウザ (Chrome, Safari など) と Web サーバ (Apache など) の複数のソフトウェアから構成
  Web サーバは他にも nginx とか

- アプリケーション層プロトコルは。メッセージ交換方法やメッセージのタイプ・フォーマットを規定
  例) web では HTTP を使用
- ネットワークアプリケーションは一般にクライアントとサーバの両方の側面をもつ (Web ブラウザと Web サーバの関係)
- ネットワークを介した プロセス間通信
  プロセス間通信は実際には 1 つ下の層であるトランスポート層が提供 (自由度が高いらしい)
  - 二つのエンドシステムにあるプロセス間でネットワークを介してやり取り
  - ソケット (Socket) と呼ばれる仮想的なインターフェースを用いる
    * ソケットは API (Application Programming Interface) の一種
    * トランスポート層とのインターフェースの役割
- コネクション (フロー) の単位
  ※ アプリケーション層が割り当てる情報
    * 送受信ホストの　IP アドレス
    * プロセスの送受信　ポート番号
      → ホストのどのプロセスかを特定する番号
    * サーバ側プロセスは多くの場合ポート番号が固定
      · 1023 番までは well-known ポートとして予約 (HTTP:80. SMTP:25, DNS:53 など)
- トランスポート層プロトコル　(TCP, UDP)
  同じポート番号でもプロトコルが違えば使用可能

2.1.2 トランスポート層プロトコルから提供されるサービス

アプリケーション製作者は、「データ損失」「帯域幅」「遅延」の観点から選択

- TCP サービス
  - コネクション指向型: 全 2 重コネクション。ハンドシェイクを待ってから通信
  - 高信頼データ伝送: 誤りのない、正しい順序のデータ送信を保証
  - 輻輳制御: ネットワークの混雑に合わせて伝送レートを変更
- UDP サービス
  - 最小限のデータ伝送: メッセージの到達性や正確性は保証しない
  - コネクションレス型: ハンドシェイクは行わない
  - 輻輳制御なし: 伝送レートは始点プロセスで指定

2.1.3 アプリケーション層プロトコルの分類

プル型、プッシュ型

- プル型プロトコル
  - HTTP, FTP など
  - ユーザが必要に応じて情報を引き出す (サーバが情報通信)
  - 情報取得を希望するクライアントがコネクションを開始
- プッシュ型プロトコル
  - SMTP など: (メールを送る側が相手へ情報送信)
  - 情報発信する側がコネクションを開始

個別帯域、共通帯域

- 個別帯域 (out-of-band) 方式
  - 制御用とデータ用で別のコネクションを使用
    大きなファイルのアップロードの際に制御情報を別ルートで送ることができる
  - 代表例は FTP Web ページの更新など
- 共通帯域 (in-band 方式)
  - 制御用とデータ用で同じコネクションを使用 (TCP を想定)
  - 代表例は HTTP

個別帯域の例

- 個別帯域の考え方は色々な場合に存在
- OpenFlow
- 5G の C/U 分離 (フェムトセル) control, user https://jirei.bzlog.jp/5g/information_14/
  - マクロセルで制御信号をやり取り
  - フェムトセルで高速データ転送

2.2 Web と HTTP

2.2.1 HTTP の概要

- HTTP(Hypertext Transfer Protocol):
  Web 用のアプリケーション層プロトコル
- Web ページ:
  基本となる HTML ファイルと、いくつかの Web オブジェクトの集合体
  URL にはホスト名やパス名、ファイル名が含まれる
- Web オブジェクト: HTML ファイル, 各種画像, JavaScript, 音声データ, 動画など
- Web ブラウザ: Web ページを表示する HTTP クライアント (Chrome, Safari, Edge, Firefox など)
- Web サーバ: Web オブジェクトを蓄える HTTP サーバ (Apache など)

- HTTP は、Web ブラウザが Web サーバに対し Web ページを要求する方法やサーバがそれに対して返送するための方法を定義
  - 基本的に、クライアントが HTTP request メッセージを送り、サーバが HTTP response メッセージを返す
- トランスポート層プロトコルは TCP
  (ソケットに渡したあとは到達性が保証される)
  1. クライアントがサーバのポート 80 へ TCP コネクションを要求
  2. サーバとクライアントの間で TCP コネクションを確立
  3. HTTP メッセージをクライアント・サーバ間で交換
  4. TCP コネクションをクローズ

- HTTP はステートレス (stateless) なプロトコル
  - サーバはクライアントの履歴情報を記憶しない
    例えば同じクライアントから、同じ要求が 2 回届くとサーバは同じレスポンスを 2 回返す
  - 制御が軽く、同時に多数の HTTP 要求に対応可能
    状態を記憶する場合、状態の維持や故障・切断時の処理定義が別途必要

  ステートレス: 履歴をもたない (一昔前のチャットボットみたいな感じ？)
  web の情報はクライアント側が保持 (cookie?)

2.2.2 非継続型コネクションと継続型コネクション

- 非継続型コネクション (non-persistent connection)
  - 一つの TCP コネクションは一つの Web オブジェクトを転送
  - 通常ブラウザは並列して 5 から 10 の TCP コネクションをはれるが埋まっていれば、各オブジェクトは直列に処理される
  - HTTP/1.0 で使用
  - 非継続型の欠点
    * 多くの TCP コネクションをサーバが管理する必要

* 各オブジェクトの転送に、2RTT 必要 (TCP コネクションの確立に 1RTT 要する)
※ RTT(Round Trip Time: 往復遅延時間)

- 継続型コネクション (Persistent connection)
  - 一つのコネクションで複数の Web オブジェクトを転送
  - サーバは応答を返した後も TCP コネクションを継続サーバで認定された一定時間 (タイムアウト時間) 使用されたなければコネクションを切断 HTTP/1.1 のデフォルト

パイプライン処理

- 非パイプライン処理型 (without pipelinig)
  - クライアントは応答メッセージの受信を待ってから、次の要求メッセージを送信
- パイプライン処理型 (with pipelinig)
  - クライアントは応答メッセージを受け取るまえに、次々と要求メッセージを送信可能

2.2.3 HTTP メッセージ

- 2 種類のメッセージ: リクエスト (要求), レスポンス (応答)
- HTTP リクエストメッセージ
  - リクエスト行: HTTP メソッド (GET, POST など)
    リクエスト対象 (URL)
    HTTP バージョン
  - ヘッダ行 : user agent などを key:value 形式でストア
  - 本文
- HTTP 応答メッセージ
  - ステータス行: ステータスコード (200, 403, 404, 503 など)
    ステータス文字列 (OK, Permission Denied, Not Found, Service Unavailable など)
  - ヘッダ行: Via や Content-length などを key:value 形式でストア
  - 本文