

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 (**U**ser **D**atagram **P**rotocol)

- 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 (packet)

- 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 $ns \sim \mu s$ のオーダー)
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 $R[\text{bps}]$, パケットサイズを $L \text{ bit}$ 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 ルーティング

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 ルーティングテーブル 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)

ルーティングテーブルの自動作成

⇒ ルーティングプロトコル

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、トラフィック量に応じて料金を課す (従量課金)

ISP: Internet service 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

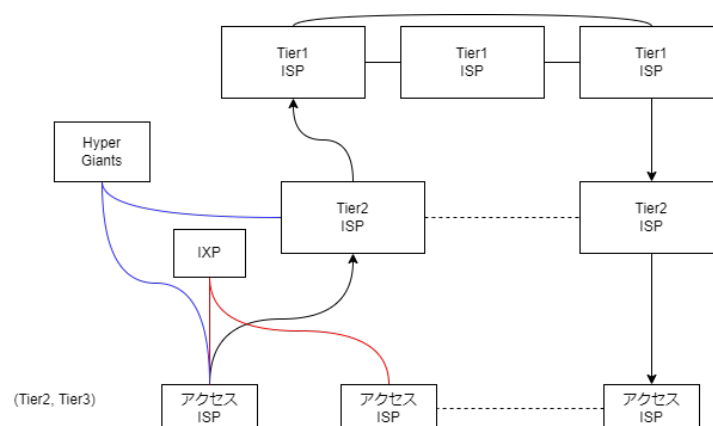
PoP を提供する事業者

- 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

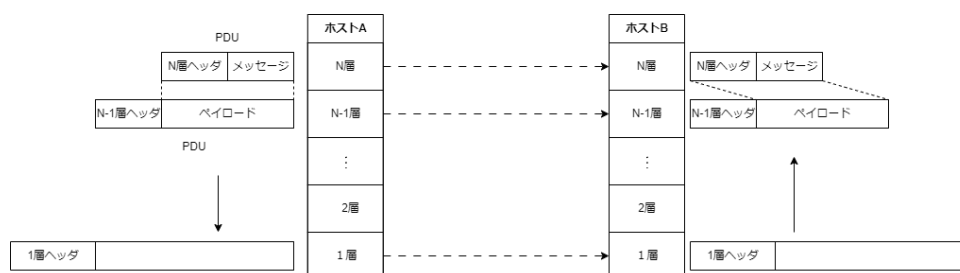
モジュール化、個別に設計

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 Reference 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

– ルーティングプロトコル:

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/](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

2.2.4 HTTP/2

2015年5月にRFC文書化された新しいHTTPのバージョン(5年ぶりのアップデート)

- ストリーム: コネクション上にレスポンスとリクエストが多重化された仮想的な双方向シーケンス
ストリームの多重化により、オブジェクトの同時送受信が可能
→ HoL (Head of Line) ブロッキング^{*1}の解消
(例えば Apache では、100 ストリームの多重化が可能)
- HPACK による HTTP ヘッダの圧縮 (二回目以降は差分で通信)
サーバなど大規模なものであればヘッダの圧縮の効果が大きい
※ HTTP/1.1 ではコンテンツのみ.gzip 圧縮
- HTTP/2 フレーム: フレームという形式のバイナリデータによる通信 (これまでは全てテキストベース)
 - HTTP ヘッダとデータをそれぞれフレーム単位で分割 (ヘッダ圧縮にも対応)
 - フレーム分割や圧縮は HTTP/1.X のメッセージに対し透過的 (HTTP の下層に配置。アプリケーションの製作者は通常どおりに HTTP を利用すればよい)
アプリケーション層とトランスポート層の間で動作。アプリケーション層では通常の HTTP として処理されるため、HTTP/1.1 でも動作する
- サーバプッシュ (レスポンスに必要なデータを事前にサーバからクライアントへ送信)
GET リクエストが来た際に必要そうなものを予め送信
- ストリームの優先制御
- HTTP/2 は、ほとんどの主要ブラウザがサポート
10 年前の技術であるためサポートされているが、そこまで早くする必要がない場合もあり HTTP/1 も現役
- TCP 以外でも動作を想定

^{*1} HoL ブロッキング問題: レスポンスにおいて、リクエストの順に返す必要があるため、大きなオブジェクトの伝送が詰まると待ち時間が発生してしまう

2.2.5 HTTP ストリーミングと DASH

DASH(Dynamic Adaptive Streaming over HTTP)

※正式には MPEGDASH

- 動画ファイルを異なるビットレートを持つ複数のバージョンにエンコード (圧縮)
動画の場合は画像毎の差分をとったりして圧縮
- セグメントに分割され、サーバ側で保持 (数秒から数十秒単位)
- クライアントは、帯域に応じて、(動的に) 適切な圧縮率のセグメントをリクエスト
(混んでいるときは、低品質を選ぶ)
- HTTP サーバは MPD (Media Presentation Description) 保持 (MPD という設定ファイル)
 - 各バージョンのセグメントの URL を記載
 - クライアントは、最初に MPD をリクエスト
- 各セグメントの取得は、通常の HTTP GET でリクエスト
- apple は MPEG DASH には対応せず HLS という別の規格が存在
<https://www.cloudflare.com/ja-jp/learning/video/what-is-mpeg-dash/>

2.3 FTP

- 個別帯域 (out-of-band) 方式のプロトコル
- データ転送に利用
- 二つのコネクションを利用
 - 制御コネクション: ポート 21、継続型
 - * ユーザ認証、cd、put、get などのコマンド
 - データコネクション: ポート 20、非継続型
 - * 実際のファイル転送を担う
 - * 1 ファイルずつ、必要に応じて

2.4 電子メール

構成要素

- ユーザエージェント (メーラ)
- メールサーバ (postfix など)
- プロトコル: SMTP(Simple Mail Transfer Protocol)
 メールアクセスプロトコル
- 始点エージェントサーバ間およびサーバ間の通信は SMTP を使用
- メールサーバ終点エージェント間にはメールアクセスプロトコルを使用
 - POP3 (Post Office Protocol ver.3)
 - IMAP (Internet Mail Access Protocol)

2.4.1 SMTP

- 始点側メールサーバ (クライアント) から終点側メールサーバへのメッセージ転送
- データ形式は 7 ビット ASCII コード
 1. クライアントはサーバにポート 25 番で TCP 接続
 2. アプリケーション層でハンドシェイク
 - (a) クライアントが送信元アドレスを知らせ、サーバ送信許可を返す
 - (b) クライアントは宛先アドレスをサーバへ知らせ、サーバは、宛先アカウントの有無をクライアントに通知
 3. クライアントはデータ送信開始を通知し、サーバの返答を受け取ると送信開始
 4. 送信終了後、TCP コネクションを切断
- TCP コネクションは継続型
 - 同じサーバ宛のメールが複数あれば、同じコネクションを使って送信可能

2.5 DNS (Domain Name System)

- ネームサーバによる階層的データベース
- ホストとネームサーバ間の通信を担うアプリケーション層プロトコル

2.5.1 提供されるサービス

- ホスト名を IP アドレスに変換 (名前決定)
- ホスト名の別名 (alias: エイリアス) の登録
- メールサーバ名の別名登録
- DNS ラウンドロビン: 負荷分散
 - － 公式ホスト名と複数のミラーサーバの IP アドレスの集合を対応づけ問い合わせ毎に IP アドレスをシフトして通知
(クライアントはリストの一番上を使う)

2.5.2 DNS の仕組み

- なぜ集中型でないか
- 分散型モデル (ネームサーバの分類)
 - － 障害発生時の信頼性欠如 (single point of failure)
 - － サーバへのトラヒック集中
 - － 遠隔地との通信遅延が大きい
 - － 保守更新の煩雑さ
- 分散型モデル (ネームサーバの分類)
 - － ルート DNS サーバ (世界で 13 系統)
 - * トップレベル DNS サーバの IP アドレスを把握
 - － トップレベルドメイン (top-level domain)DNS サーバ
 - * jp, com, edu などのトップレベルドメインを管理
 - － 権威 (Authoritative)DNS サーバ
 - * ホスト名 (ドメイン名) と IP アドレスの変換機能を持つ
 - * 各ホストは必ず一つ以上の権威 DNS サーバに登録
- ※ ローカル DNS サーバ:DNS キャッシュサーバ、DNS リゾルバ
 - * ISP や各企業が個別に持つ
 - * ユーザは通常、ローカル DNS サーバに名前解決を依頼
- DNS 反復クエリと DNS 再帰クエリ
 - － DNS 反復クエリ
 - * 問い合わせたネームサーバへの名前の解決を依頼
 - － DNS 再帰クエリ
 - * 問い合わせるべきネームサーバの情報を要求

- ローカル DNS サーバはホストへ転送した情報をキャッシュする:DNS キャッシュ
- DNS レコード
 - － リソースレコードを格納する分散データベース
 - － (name, value, type, ttl) の 4 フィールドで構成
 - － Type A レコード
 - * name はホスト名, value は IPアドレス (IPv6 の場合、AAAA レコード)
 - － Type NS レコード
 - * name はドメイン名, value はその権威 DNS サーバホスト名
 - － Type CNAME レコード
 - * ホスト名の別名
 - － Type MX レコード
 - * name に関するメールサーバ情報を格納
- TTL は秒単位で記載
 - － この期間を過ぎるとキャッシュから削除

2.5.3 DNS サーバへの攻撃

- DNS キャッシュポイズニング
 1. 攻撃者が DNS キャッシュサーバへ名前解決を依頼
 2. DNS キャッシュサーバは再帰的に問合せ
 3. 攻撃者は即座に、偽装した UDP パケットで返答
 4. キャッシュサーバは、誤ったドメイン情報を記憶

※ 攻撃要件

1. 発信元 DNS サーバの IP アドレスに偽装
2. 問い合わせと同じ UDP ポート番号を使用
3. 問い合わせと同じ ID(6/bit) を使用
4. 本来の DNS サーバより先に応答
 - － さらに、攻撃のチャンスは TTL 毎に一度だけ

2.6 コンテンツ分散

- コンテンツが一つのサーバ上のみにあると
 - － 伝送遅延が増大
 - － サーバへの負担が大きい
- コンテンツ分散
 - － コンテンツをインターネット上の複数サーバにコピー
 - － ユーザに最も迅速に配送できるサーバを見つけ、コンテンツを配信

2.6.1 CDN (Content Distribution Network)

- 90 年代後半から登場したビジネスモデル
- CDN サービス企業は、インターネット上に多数のサーバ (CDN サーバ) を配置
 - － (Netflix や Youtube は自社で CDN サーバを設置)
 - － 顧客のコンテンツを CDN サーバにコピー (各更新毎に)
 - － ユーザのコンテンツを要求に対して、適切な CDN サーバを選択

→ ユーザごとに DNS の返答を変える

3 トランスポート層

3.1 トランスポート層サービス

アプリケーションプロセスへ、通信サービスを直接提供

- アプリケーションを実行するエンドシステムに組み込まれている
- ネットワーク内のルータには通常組み込まれない

アプリケーションプロセス 間に、論理的通信路 を提供

- アプリケーションは、物理的通信インフラの構造を意識しない

3.1.1 トランスポート層とネットワーク層の関係

ネットワーク層: ホスト間 の 論理的通信路 を提供

インターネット (IP) では、ベストエフォート *2型転送サービス

アプリケーションから見たとき、トランスポート層がないと、

- アプリケーション固有の要求に応じた適切な通信サービスが受けられない
- 物理的特性あるいはネットワークの輻輳により、通信インフラの信頼性が低い場合に対処できない

3.1.2 インターネットのトランスポート層

最小限度の機能 (UDP, TCP とともに備える)

- プロセス間の転送サービス
- セグメント *3の伝送誤り検査

UDP (User Datagram Protocol): 信頼性を保証しないコネクションレス型サービス

最小限度の機能のみを持つ

TCP (Transmission Control Protocol): 信頼性の高い、コネクション指向型サービス

最小限度の機能に加えて

- － 誤りのない、正しい順序でのセグメント転送を保証
- － 輻輳制御とフロー制御による、送信レート制御を提供

*2 何も保証しない

*3 トランスポート層 PDU

3.2 多重化と逆多重化

多重化 (multiplexing): 様々なソケット (アプリケーションとのインターフェース) からトランスポート層へ送出されたセグメントをまとめてネットワーク層へ引き渡す
 逆多重化 (demultiplexing): ネットワーク層から取り出されたセグメントを適切なソケットへ渡す

ポート番号: ソケットを識別する番号 (0 から $65535 (2^{16} - 1)$)

- 0 から 1023 までは well-known ポート番号として予約
- TCP と UDP で区別 (両者で同じ番号が使える)

クライアントが UDP あるいは TCP ソケットを作り、ポート番号を付加
 (通常は 1024 番号以降からランダムに選択)

サーバ側では well-known ポート番号を使用

3.2.1 UDP

クライアント: 始点ポート番号と終点ポート番号をヘッダに付加し、(ソケットへ bind し) 多重化

サーバ: 終点ポート番号 (と IP アドレス) のみに基づいてソケットを選択
 (異なるクライアントからのセグメントでも同じソケットへ渡される)
 始点ポート番号は、返信の際の終点ポート番号として使用

3.2.2 TCP

クライアント: UDP と同じ

サーバ: 始点 IP アドレス、始点ポート番号、終点 IP アドレス、終点ポート番号でソケットを選択

3.3 UDP

- 8 バイトのヘッダ + ペイロード (= アプリケーション層メッセージ)

← 32 ビット →	
始点ポート番号	終点ポート番号
セグメントの長さ	チェックサム
アプリケーション層メッセージ	

- 受信側でチェックサムを用いて誤り検出を行う
 チェックサム: 16 ビット語を全て足したものの 1 の補数
- これ以外は何もせず、必要最低限のトランスポート層サービスを提供
 - メッセージを直接 IP 層へ渡す
 使用例: DNS, SNMP など

- UDP を用いる利点

1. 送るべきデータとその送出タイミングをアプリケーションから制御可能
 - TCP の制約から解放、高信頼データ転送は、必要に応じてアプリケーション側で実装
2. コネクションの確立が不要: 遅延の減少
3. コネクションの管理が不要
 - サーバでのオーバーヘッドが小さく、多くのクライアントと通信可能
4. ヘッダが小さく、伝送効率が高い (UDP 8 バイト, TCP 20 バイト)

3.4 高信頼データ転送

高信頼データ転送 (reliable data transmission)

- 誤り (0, 1 の反転) がなく、かつ、送信された順序通りに届く
- エンド-エンドの制御でどうやって実現するのか
 - ⇒ **ARQ (Automatic Repeat reQuest)** プロトコルによる制御
 - 誤り検知・送達確認を行いながら、必要に応じて再送

シナリオ 1. パケットが壊れる (ビットエラーが発生する) 通信路

誤り検知と送達確認

- ACK (Acknowledgement): 正しく受信されたことを伝える
- NACK (NegativeACK): 正しく受信されなかったことを伝える

ストップアンドウェイト (SW: Stop and Wait) プロトコル

- パケットは一つずつ送信 (受信バッファサイズ 1)
- ACK/NACK を受け取ったら、次のパケットを送信
 - NACK の場合、再送

シナリオ 1.1 ACK/NACK も壊れる場合

- ACK/NACK が壊れると、正しく受け取れたか判断できない
- ACK を再送したとしても、それが「どのパケットの ACK か」がわからない

⇒ パケットに **シーケンス番号** を付与

- SW の場合、1 ビットで十分
- 再送中であることを表すフラグ

シナリオ 1.1': 1.1 と同様の状況で、NACK を使用しない

同じシーケンス番号の ACK が 2 度届けば、NACK と同じ効果

1,0 が交互にくるはずだから、連続で同じ数値の場合は再送とわかる

シナリオ 2.0; パケットロスが発生する通信路

パケットがそもそも到着しないので、エラーが検知できない

⇒ **タイムアウト時間** を設定

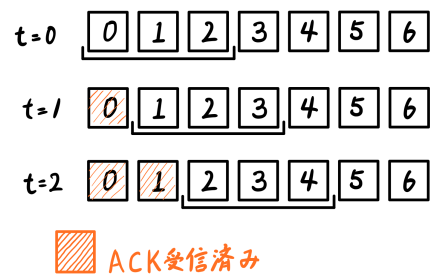
- タイムアウト時間内に ACK を受信したら、次のパケットを送信
- タイムアウト時間内に ACK を受信しなければ、再送

※ 遅延 > タイムアウト時間のとき、ロスがないのに再送してしまう

3.4.2 Go-back-N (GBN)

- ACK の受信を待たずに、連続するパケットを最大 N 個まで送信
(スライディングウィンドウプロトコル)
 - 送信失敗を確認すると、そのパケットから全てやり直し
 - 受信した順序が入れ替わっていたら、パケットを捨てる
- ACK が来ないとウィンドウが進めなくなる

受信側が簡単な仕様、TCP はざっくり Go-back-N だけど、すぐに捨てるわけじゃないみたい



3.4.4 SR (selective-repeat)

ACK の受信を待たずに連続するパケットを最大 N 個まで送信可

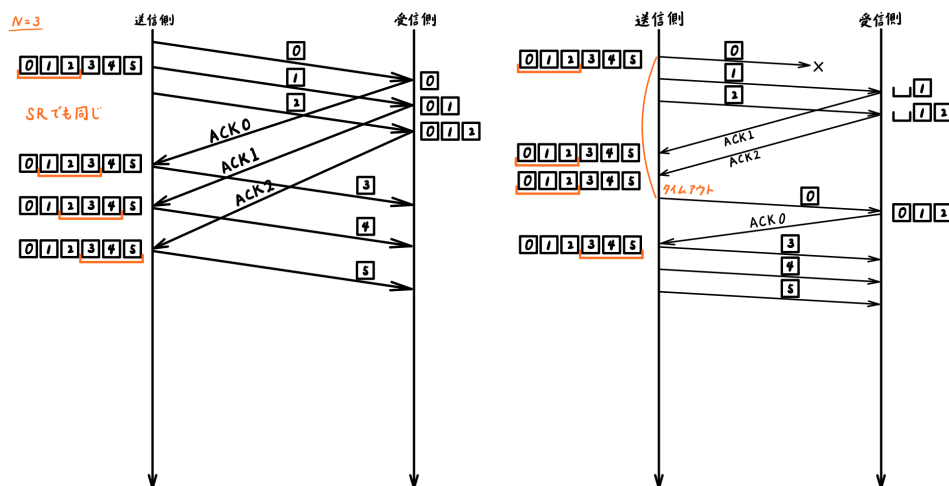
送信失敗が確認されると、そのパケットのみ再送 (受信バッファサイズ N)

- 送信側

- パケットにシーケンス番号を付加 (k ビット)
- ウィンドウサイズ ($N \leq 2^{k-1}$) とタイムアウト時間を設定
 - * n : 送信したが、ACK 未受信のパケットの最小シーケンス番号
- 1. 次に送るべきパケットのうち、 $n + N - 1$ 番までを連続的に送信し、2 へ
- 2. 各パケットに対する ACK の受信を待つ
 - タイムアウト時間内に ACK を受信したら、1 へ
 - タイムアウト時間内に ACK を受信しなければそのパケットを再送し、2 へ

- 受信側

- パケットを受け取ったら、誤りがないか確認
- 誤りがなければ、そのシーケンス番号を付加した ACK *4 を送信
(順序通りに上位層へ渡すため、バッファリング)



*4 選択的 (Selective) ACK