

Chapter 2 Application Layer



A note on the use of these ppt slides:

We're making these slides freely available to all (faculty, students, readers). They're in PowerPoint form so you can add, modify, and delete slides (including this one) and slide content to suit your needs. They obviously represent a lot of work on our part. In return for use, we only ask the following:

- If you use these slides (e.g., in a class) in substantially unaltered form, that you mention their source (after all, we'd like people to use our book!)
- If you post any slides in substantially unaltered form on a www site, that you note that they are adapted from (or perhaps identical to) our slides, and note our copyright of this material.

Thanks and enjoy! JFK/KWR

All material copyright 1996-2002
J.F. Kurose and K.W. Ross, All Rights Reserved

*Computer Networking:
A Top Down Approach
Featuring the Internet,
2nd edition.*
Jim Kurose, Keith Ross
Addison-Wesley, July
2002.

2: Application Layer 1

Chapter 2: アプリケーション層

目標:

- ネットワークアプリケーションの概念と実装
 - トランスポート層サービスモデル
 - クライアント・サーバパラダイム
 - ピア・ツー・ピアパラダイム
- 有名なアプリケーションレベルプロトコルを調べることによって学ぶ
 - HTTP
 - FTP
 - SMTP / POP3 / IMAP
 - DNS
- ネットワークアプリケーションのプログラミング
 - socket API

2: Application Layer 2

Chapter 2 内容

- 2.1 アプリケーション層プロトコルの原理
- 2.2 Web と HTTP
- 2.3 FTP
- 2.4 電子メール
 - SMTP, POP3, IMAP
- 2.5 DNS
- 2.6 TCPによるソケットプログラミング(略)
- 2.7 UDPによるソケットプログラミング(略)
- 2.8 Web サーバの構築(略)
- 2.9 コンテンツ分配
 - ネットワークWebキャッシュ
 - コンテンツディストリビューションネットワーク
 - P2P ファイル共有

2: Application Layer 3

ネットワークアプリケーション: 用語

- プロセス:** ホスト内で実行されているプログラム
- ホスト内では(OSによって定義される) **プロセス間通信**により二つのプロセスが通信する
- 異なるホスト内で実行されているプロセスは、**アプリケーション層プロトコル**により通信する
- ユーザエージェント:** ユーザが“上に”, ネットワークが“下に”あるインタフェース
- ユーザインタフェースとアプリケーションプロトコルの実装
 - Web: ブラウザ
 - E-mail: メールリーダー
 - streaming audio/video: メディアプレイヤー

2: Application Layer 4

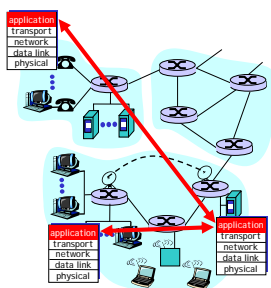
アプリケーションとアプリケーション層プロトコル

アプリケーション: 通信分散プロセス

- 例: e-mail, Web, P2P file sharing, instant messaging
- エンドシステム(ホスト)内で動作
- アプリケーション実装のためにメッセージを交換

アプリケーションプロトコル

- アプリケーションの“一部”
- アプリケーションが交換するメッセージとアプリケーションがとる動作を定義
- 下位層(TCP, UDP)によって提供される通信サービスを利用



2: Application Layer 5

アプリケーションプロトコルの定義

- 交換メッセージの種類
 - 例: 要求メッセージ, 応答メッセージ
- メッセージタイプのシンタックス: メッセージ内のどこをフィールドをどのように区切るか
- フィールドのシンタックス: フィールド内の情報の意味
- プロセスがいつどのようにメッセージを送受信するのかに対するルール
- 公開プロトコル:**
 - RFC で規定
 - 相互接続性の実現
 - 例: HTTP, SMTP
- 非公開プロトコル:**
 - 例: KaZaA

2: Application Layer 6

クライアント・サーバパラダイム

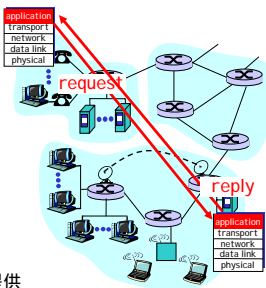
ネットワークアプリは、通常、クライアントとサーバから成り立つ

クライアント:

- サーバにコンタクトを開始(最初に話しかける)
- 通常、サーバにサービスを要求
- Web: クライアント機能は、ブラウザに実装, 電子メール: メールリーダーに実装

サーバ:

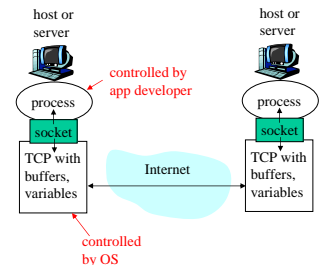
- クライアントに要求されたサービスを提供
- 例: Webサーバは、要求されたページを送信, メールサーバは、電子メールを配信



2: Application Layer 7

ネットワークを介したプロセス間通信

- プロセスは、ソケットを介してメッセージを送受信する
- ソケットとはドアに似ている
 - 送信プロセスは、メッセージをドアの外に押し出す
 - 送信プロセスは、ドアの裏側にあるトランスポート機能が受信プロセスのソケットにメッセージを運んでくれると仮定



- API: (1) トランスポート層の選択; (2) いくつかのパラメータを設定する機能 (その他は後述)

2: Application Layer 8

アドレッシング機能:

- メッセージ受信プロセスは、識別子を持たなければならない
- 各ホストは一意にあらわされる32ビットのIPアドレスを持つ
- 質問: ホストのIPアドレスは、プロセスの識別に十分か?
- 答え: ノー、多くのプロセスが同一ホストで実行されている可能性がある
- 識別子は、ホスト上のプロセスと関連付けられたIPアドレスとポート番号を含む
- ポート番号例:
 - HTTP server: 80
 - Mail server: 25
- その他は後述

2: Application Layer 9

アプリはどのようなトランスポート層が必要か?

データ廃棄

- いくつかのアプリ(例: 音声)は、ある程度のロスに対して耐性がある
- その他のアプリ(例: ファイル転送, telnet)は、100%の信頼性データ転送を要求する

適時性(遅延)

- いくつかのアプリ(例: インターネット電話, 対話型ゲーム)は低遅延であることが要求される

帯域

- いくつかのアプリ(例: マルチメディア)は、最小帯域が利用できることを要求する
- その他のアプリ(バースト型アプリ)は使用可能な帯域を利用する

2: Application Layer 10

一般的アプリのトランスポート層への要求条件

アプリケーション	データ廃棄	帯域	適時性
ファイル転送	不可	順応性あり	必要なし
電子メール	不可	順応性あり	必要なし
Webドキュメント	不可	順応性あり	必要なし
実時間音声/映像	許容	音声: 5kbps-1Mbps 映像: 10kbps-5Mbps	必要, 数百ミリ秒
蓄積型音声/映像	許容	同上	必要, 数秒
対話型ゲーム	許容	数kbps以上	必要, 数百ミリ秒
インスタントメッセージ	不可	順応性あり	時には必要

2: Application Layer 11

インターネットトランスポートプロトコルサービス

TCP サービス:

- **コネクション型**: クライアント・サーバ間でセットアップが必要
- **高信頼転送**: 送受信プロセス間
- **フロー制御**: 受信側の能力に合わせた転送
- **輻輳制御**: ネットワーク過負荷時に送信側を調整
- **未提供**: 遅延保証, 最小帯域保証

UDP サービス:

- 送受信間の低信頼性データ転送
- does not provide: コネクションセットアップ, 信頼性, フロー制御, 輻輳制御, 遅延・帯域保証

質問: なんでわざわざ? UDPが存在する理由は?

2: Application Layer 12

インターネットアプリ: アプリケーション, トランスポートプロトコル

アプリケーション	アプリケーション層プロトコル	利用するトランスポートプロトコル
電子メール	SMTP [RFC 2821]	TCP
遠隔端末アクセス	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
ファイル転送	FTP [RFC 959]	TCP
マルチメディアストリーミング	専用プロトコル (e.g. RealNetworks)	TCP or UDP
インターネット電話	専用プロトコル (e.g., Dialpad)	多くの場合 UDP

2: Application Layer 13

Chapter 2 内容

- 2.1 アプリケーション層プロトコルの原理
- 2.2 Web と HTTP
- 2.3 FTP
- 2.4 電子メール
 - SMTP, POP3, IMAP
- 2.5 DNS
- 2.6 TCPによるソケットプログラミング
- 2.7 UDPによるソケットプログラミング
- 2.8 Web サーバの構築
- 2.9 コンテンツ分配
 - ネットワークWebキャッシュ
 - コンテンツディストリビューションネットワーク
 - P2P ファイル共有

2: Application Layer 14

Web と HTTP

いくつかの専門用語

- Web ページ はオブジェクトからなる
- オブジェクトとは, HTMLファイル, JPEG画像, Javaアプレット, audio ファイル, など
- Webページは, 参照オブジェクトを含む基本HTMLファイルから構成される
- 各オブジェクトは, URL (Uniform Resource Locator) によって指定される
- URLの例:

www.someschool.edu / someDept/pic.gif

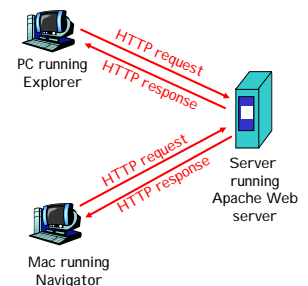
ホスト名 パス名

2: Application Layer 15

HTTP 概要

HTTP: hypertext transfer protocol

- Webのアプリケーション層プロトコル
- クライアント/サーバモデル
 - クライアント: オブジェクトをリクエスト, 受信, 表示するブラウザ
 - サーバ: リクエストに応答してオブジェクトを送信するWebサーバ
- HTTP 1.0: RFC 1945
- HTTP 1.1: RFC 2068



2: Application Layer 16

HTTP 概要 (続き)

TCPの仕様:

- クライアントは, サーバの80番ポートにTCPコネクション(ソケットの生成)を要求
- サーバは, クライアントからのTCPコネクションを受付
- HTTPメッセージ(アプリケーション層プロトコルメッセージ)をブラウザ(HTTPクライアント)とWebサーバ(HTTPサーバ)間で交換
- TCPコネクションのクローズ

HTTPは“ステートレス”

- サーバは, 過去のクライアントからの要求に関するいかなる情報も保持しない

余談
“状態”を維持するプロトコルは複雑である!
過去履歴(状態)が維持されなければならない
サーバまたはクライアントが故障した場合, 互いの“状態”が不整合になるかもしれないし, 整合をとらなければならない

2: Application Layer 17

HTTP コネクション

非継続型(nonpersistent) HTTP

- ひとつのオブジェクトを送信するたびにTCPコネクションをはる
- HTTP/1.0 は非継続型HTTPを使用

継続型(persistent) HTTP

- 複数のオブジェクトを単一のTCPコネクションを介して送信可能
- HTTP/1.1 は, デフォルトでは継続型HTTPを使用

2: Application Layer 18

非継続型 HTTP

ユーザが以下のURLを入力すると仮定

www.someSchool.edu/someDepartment/home.index

(テキストと10個のJpeg
画像への参照を含む)

- 1a. HTTPクライアントは
www.someSchool.edu の 80番ポ
ートの HTTP サーバ (プロセス)に
TCP コネクションを発行
- 1b. 80番ポートで TCP コネクショ
ン接続要求を待っているホスト
www.someSchool.edu のHTTP
サーバは、コネクションを“受け付
け”、クライアントに通知
2. HTTP クライアントはHTTP要求メ
ッセージ(URLを含む)を TCP コネ
クションソケットに送信。このメッ
セージは、オブジェクト
someDepartment/home.index
を要求していることを記述
3. HTTPサーバは、要求メッセ
ージを受け取り、要求されたオブ
ジェクトを含む応答メッセージ
を生成し、これを自分のソケッ
トに送信する

time

2: Application Layer 19

非継続型 HTTP(続き)

4. HTTP サーバはTCPコネクション
を閉じる
5. HTTP クライアントは、htmlフ
ァイルを含む応答メッセージを
受け取り、html ファイルを表
示する。Htmlファイルを分析し
、10個のjpeg オブジェクトが
参照されていることがわかる
6. ステップ1-5を、各10個のjpegオ
ブジェクトに対して繰り返す

time

2: Application Layer 20

応答時間のモデル化

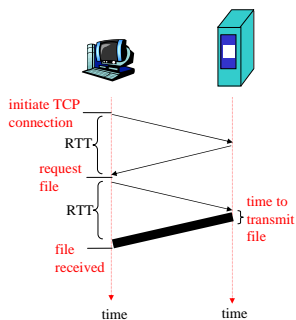
RTT (Round Trip Time)の定

義: 小バケットのクライアント・
サーバ間の往復遅延時間

応答時間:

- TCPコネクションを設定する
ための1 RTT
- HTTP要求とHTTP応答の
最初の数バイトのための1
RTT
- ファイル伝送時間

total = 2RTT+transmit time



2: Application Layer 21

継続型 HTTP

非継続型 HTTP issues:

- オブジェクトあたり2 RTT 必要
 - OSは各TCPコネクションに対し
、動作し、ホストのリソースを割
当なければならない
 - ブラウザは、参照オブジェクトの
取得のために平行して複数の
TCPコネクションをオープンする
- 継続型 HTTP
- サーバは、応答送信後、コネク
ションをオープンのままにする
 - 続く同一クライアント・サーバ間
のHTTPメッセージは、オープン
のままのコネクションを使って送
信される

継続型非パイプライン方式:

- クライアントは、前回の応答を
受け取った後で新しい要求を
送信できる
 - 各参照オブジェクトに対して1
RTT 必要
- 継続型パイプライン方式:
- HTTP/1.1におけるデフォルト
 - クライアントは、参照オブジェ
クトを見出すとすぐに要求メッ
セージを送信
 - 全参照オブジェクトに対して1
RTT程度の小ささ

2: Application Layer 22

HTTP 要求メッセージ

- 2種類の HTTP メッセージ: 要求, 応答
- HTTP 要求メッセージ:
 - ASCII (可読可能なフォーマット)

リクエスト行
(GET, POST,
HEAD commands)

ヘッダ行

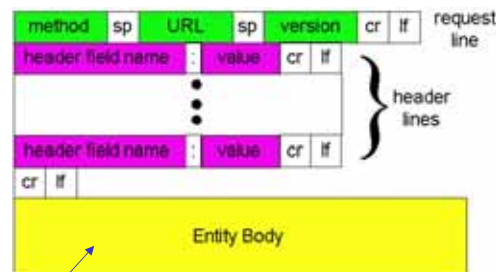
```
GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
User-agent: Mozilla/4.0
Connection: close
Accept-language: fr
```

改行 (キャリッジリターン、
ラインフィード)
メッセージの終了を表す

(extra carriage return, line feed)

2: Application Layer 23

HTTP 要求メッセージ: 一般フォーマット



Get method では利用されない

2: Application Layer 24

入力フォーム送信

- Post メソッド:

 - Web ページには入力フォームが含まれている
 - 入力は、エンティティボディを使ってサーバに送られる
- URL メソッド:

 - GETメソッドを利用
 - 入力内容は、要求行のURLフィールドを利用して送信される
- www.somesite.com/animalsearch?monkeys&banana

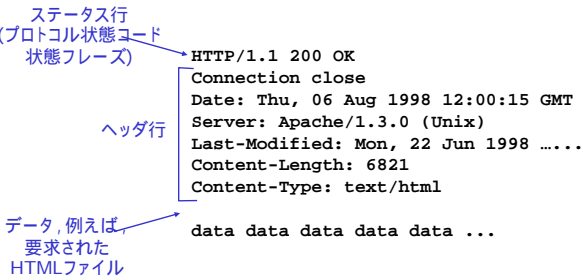
メソッドタイプ

- HTTP/1.0

 - GET
 - POST
 - HEAD
 - 要求オブジェクトを応答メッセージに含めないように通知
- HTTP/1.1

 - GET, POST, HEAD
 - PUT
 - URLフィールドに記述したパスにエンティティボディ内のファイルをアップロード
 - DELETE
 - URLフィールドに記述したファイルを消去

HTTP 応答メッセージ



HTTP 応答ステータスコード

- サーバからクライアントへの応答メッセージ内の第1行目例:
- 200 OK**
 - 要求成功, 要求オブジェクトがメッセージ後半に添付
 - 301 Moved Permanently**
 - 要求メッセージは移動, 新規移動先をメッセージ内に記述 (Location:)
 - 400 Bad Request**
 - 要求メッセージ理解不能
 - 404 Not Found**
 - 要求ドキュメントがサーバ内にない
 - 505 HTTP Version Not Supported**

HTTP (クライアント)の実験的使用

- 対象とするWebサーバに telnet する:

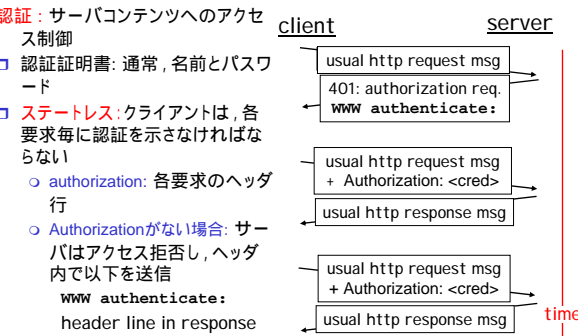
```
telnet www.eurecom.fr 80
```

 [www.eurecom.fr の80番ポート(デフォルト HTTP サーバポート)に対してTCPコネクションをオープン, 入力内容はこのポートに向けて送信される]
- GET HTTP 要求の入力:

```
GET /~ross/index.html HTTP/1.0
```

 [これを入力することで(最後に2回改行を入力), もっとも簡単かつ完全な GET 要求を HTTPサーバに送信]
- HTTPサーバから送信された内容を見よう!

ユーザ・サーバ間のやり取り: 認証



Cookie: 状態の保持

多くの Web サイトは
Cookie を使用

4つのコンポーネント:

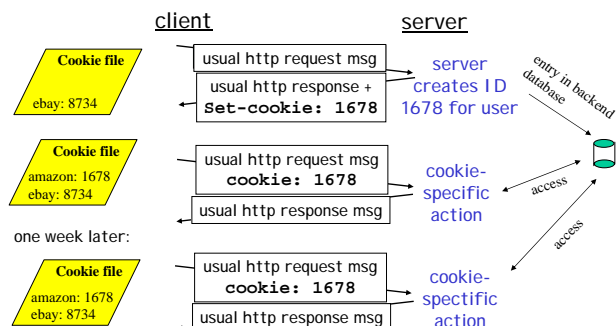
- 1) HTTP応答メッセージ内の cookie ヘッダ行
- 2) HTTP要求メッセージ内の cookieヘッダ行
- 3) ユーザホストのブラウザによって管理される cookie ファイル
- 4) Webサイトのバックエンドデータベース

例:

- スーザンはいつも同じ PC からインターネットにアクセス
- 彼女はある特定の e-commerce サイトにはじめて訪問
- サイトに最初の HTTP 要求が到着したときに、サイトは一意的 ID を生成し、バックエンドデータベースにこのIDのためのエントリを作成

2: Application Layer 31

Cookie: 状態の保持 (つづき)



2: Application Layer 32

Cookie (つづき)

クッキーがもたらすもの:

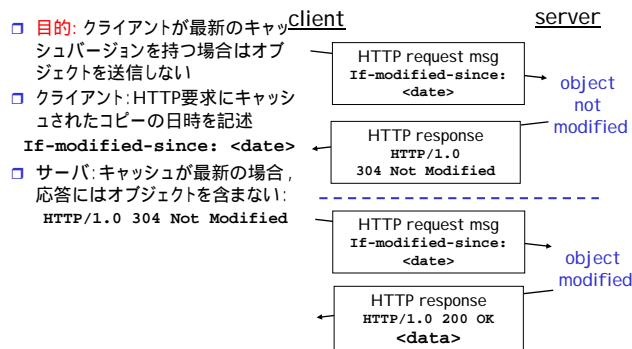
- 認証
- ショッピングカート
- お勧め
- ユーザセッション状態 (Web e-mail)

Cookieとプライバシー:

- Cookieによりサイトはあなた自身について多くを学ぶ
- 名前と電子メールをサイトに教えることもある
- サーチエンジンは、リダイレクションと cookie を利用してより多くの情報を収集できる
- 広告会社はサイトを通じて情報を収集

2: Application Layer 33

条件付GET:クライアントサイドキャッシング



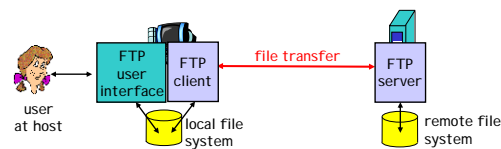
2: Application Layer 34

Chapter 2 内容

- 2.1 アプリケーション層プロトコルの原理
- 2.2 Web と HTTP
- 2.3 FTP
- 2.4 電子メール
 - SMTP, POP3, IMAP
- 2.5 DNS
- 2.6 TCPによるソケットプログラミング
- 2.7 UDPによるソケットプログラミング
- 2.8 Web サーバの構築
- 2.9 コンテンツ分配
 - ネットワークWebキャッシュ
 - コンテンツディストリビューションネットワーク
 - P2P ファイル共有

2: Application Layer 35

FTP: ファイル転送プロトコル

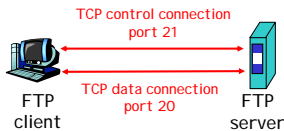


- 遠隔ホストに / からファイルを転送
- クライアント / サーバ モデル
 - クライアント: 転送開始 (遠隔ホストに / から)
 - サーバ: 遠隔ホスト
- ftp: RFC 959
- ftp server: 21番ポート

2: Application Layer 36

FTP: 別々の制御・データコネクション

- FTPクライアントは、21番ポートでFTPサーバにコンタクトし、TCPを転送プロトコルとすることを指定
- クライアントは、制御チャンネルを介して認証を得る
- クライアントは、制御コネクションを介してコマンドを送信し、遠隔ディレクトリをブラウズ
- サーバは、ファイル転送に対するコマンドを受信すると、TCPデータコネクションをクライアントに対してオープンする
- 1ファイルを転送後、サーバはコネクションをクローズする
- サーバは、次のファイル転送のために、TCPデータコネクションをオープンする
- 制御コネクション: "out of band"
- FTPサーバは、"状態"を維持: カレントディレクトリ, 認証



2: Application Layer 37

FTPコマンド, 応答

コマンド例:

- 制御チャンネルを介してASCIIテキストとして送信
- USER *username*
- PASS *password*
- LIST は、カレントディレクトリのファイルリストを返信
- RETR *filename* で、ファイルを獲得 (get)
- STOR *filename* で、遠隔ホストにファイルを格納 (put)

リターンコード例

- 状態コードとフレーズ (HTTPと同様)
- 331 Username OK, password required
- 125 data connection already open; transfer starting
- 425 Can't open data connection
- 452 Error writing file

2: Application Layer 38

Chapter 2 内容

- 2.1 アプリケーション層プロトコルの原理
- 2.2 Web と HTTP
- 2.3 FTP
- 2.4 電子メール
 - SMTP, POP3, IMAP
- 2.5 DNS
- 2.6 TCPによるソケットプログラミング
- 2.7 UDPによるソケットプログラミング
- 2.8 Web サーバの構築
- 2.9 コンテンツ分配
 - ネットワークWebキャッシュ
 - コンテンツディストリビューションネットワーク
 - P2P ファイル共有

2: Application Layer 39

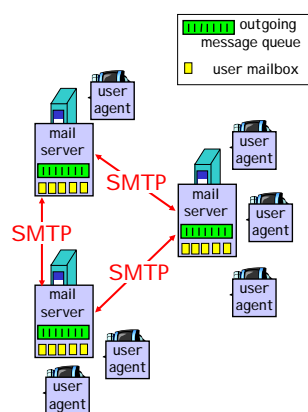
電子メール

3つの主要コンポーネント:

- ユーザエージェント
- メールサーバ
- simple mail transfer protocol: SMTP

ユーザエージェント

- "メールリーダー"のこと
- メールメッセージを作成, 編集, 読む
- 例: Eudora, Outlook, elm, Netscape Messenger
- 差出, 受信メッセージをサーバに保存

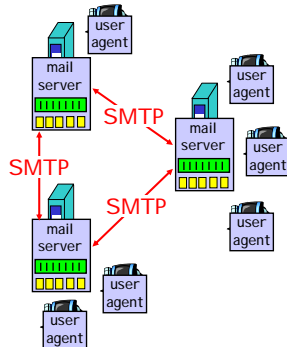


2: Application Layer 40

電子メール: メールサーバ

メールサーバ

- メールボックスは、受信メッセージを保存
- 送信すべきメールメッセージのメッセージキュー
- 電子メールメッセージを送信するメールサーバ間のSMTPプロトコル
 - クライアント: 送信メールサーバ:
 - サーバ: 受信メールサーバ



2: Application Layer 41

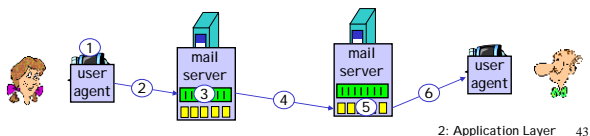
電子メール: SMTP [RFC 2821]

- クライアント・サーバ間高信頼電子メールメッセージ転送のために25番ポートのTCPを使用
- 直接転送: 送信サーバから受信サーバへ (中継サーバが入る場合がある)
- 3段階転送
 - ハンドシェイク (挨拶)
 - メッセージ転送
 - 終了
- コマンド / 応答インタラクション
 - commands: ASCII テキスト
 - response: 状態コードとフレーズ
- メッセージは7-bit ASCIIの必要がある

2: Application Layer 42

シナリオ: アリスがボブにメール送信

- 1) アリスは, メッセージとあて先 bob@someschool.edu を作成するためにユーザエージェントを使用する.
- 2) アリスのユーザエージェントは, メッセージを彼女のメールサーバに送信し, メッセージはメッセージキューに置かれる.
- 3) SMTPのクライアント側は, ボブのメールサーバにTCPコネクションをオープンする.
- 4) SMTPクライアントは, TCPコネクションを介してアリスのメッセージを送信する.
- 5) ボブのメールサーバは, ボブのメールボックスにメッセージをおく.
- 6) ボブは, メッセージを読むために自分のユーザエージェントを起動する.



簡単なSMTPインタラクション

```
S: 220 hamburger.edu
C: HELO crepes.fr
S: 250 Hello crepes.fr, pleased to meet you
C: MAIL FROM: <alice@crepes.fr>
S: 250 alice@crepes.fr... Sender ok
C: RCPT TO: <bob@hamburger.edu>
S: 250 bob@hamburger.edu ... Recipient ok
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: Do you like ketchup?
C: How about pickles?
C: .
S: 250 Message accepted for delivery
C: QUIT
S: 221 hamburger.edu closing connection
```

2: Application Layer 44

SMTPインタラクションを自分で試そう

- サーバの25番ポートにtelnetコマンドを実行
 - サーバからの220 replyを確認
 - HELO, MAIL FROM, RCPT TO, DATA, QUITコマンドを入力
- これにより, 電子メールクライアントを使用しないでもメール送信できる

2: Application Layer 45

SMTP: 最後に

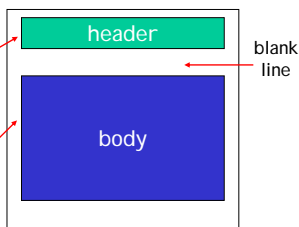
- SMTPは, 継続型コネクションを使用
 - SMTPではメッセージ(ヘッダと本文)が7-bit ASCIIである必要がある
 - SMTPサーバは, メッセージの終了を決定するために CRLF, CRLF を使用
- HTTPとの比較:**
- HTTP: pull
 - SMTP: push
 - 両者とも ASCII コマンド/レスポンス インタラクション, ステータスコードを使用
 - HTTP: 各オブジェクトは, 応答メッセージ自身にカプセル化される
 - SMTP: 複数のオブジェクトは, マルチパートメッセージで送信される

2: Application Layer 46

メールメッセージフォーマット

SMTP: 電子メールメッセージを交換するためのプロトコル
RFC 822: テキストメッセージフォーマットの標準:

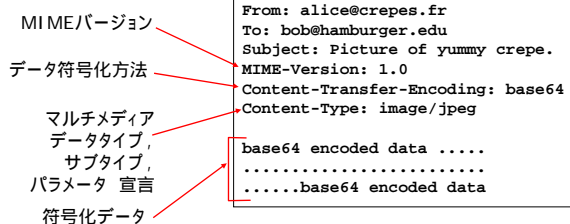
- ヘッダ行: 例
 - To:
 - From:
 - Subject:
- 本文
 - "メッセージ", ASCII文字のみ



2: Application Layer 47

メッセージフォーマット: マルチメディア拡張

- MIME: multimedia mail extension, RFC 2045, 2056
- メッセージヘッダの追加行によりMIMEコンテンツタイプを宣言



2: Application Layer 48

MIME型

Content-Type: type/subtype; parameters

Text

- サブタイプ例: plain, html

Image

- サブタイプ例: jpeg, gif

Audio

- サブタイプ例: basic (8-bit mu-law encoded), 32kadpcm (32 kbps coding)

Video

- サブタイプ例: mpeg, quicktime

Application

- 可視化の前にリーダによって処理されなければならない他のデータ
- サブタイプ例: msword, octet-stream

2: Application Layer 49

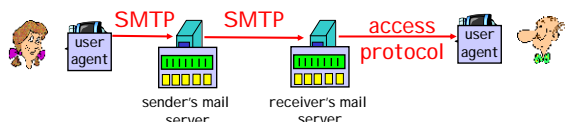
マルチパートタイプ

```
From: alice@crepes.fr
To: bob@hamburger.edu
Subject: Picture of yummy crepe.
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary=StartOfNextPart
```

```
--StartOfNextPart
Dear Bob, Please find a picture of a crepe.
--StartOfNextPart
Content-Transfer-Encoding: base64
Content-Type: image/jpeg
base64 encoded data .....
.....base64 encoded data
--StartOfNextPart
Do you want the recipe?
```

2: Application Layer 50

メールアクセスプロトコル



- SMTP: 受信サーバへのメール配信 / 蓄積
- メールアクセスプロトコル: サーバからのメール取得
 - POP: Post Office Protocol [RFC 1939]
 - ・ 認証 (エージェント <--> サーバ) とダウンロード
 - IMAP: Internet Mail Access Protocol [RFC 1730]
 - ・ より多くの機能 (より複雑)
 - ・ サーバに蓄積されたメッセージの操作
 - HTTP: Hotmail, Yahoo! Mail, etc.

2: Application Layer 51

POP3プロトコル

認証フェーズ

- クライアントコマンド:
 - user: ユーザ名の宣言
 - pass: パスワード
- サーバレスポンス
 - +OK
 - -ERR

処理フェーズ, クライアント:

- list: メッセージ番号のリスト
- retr: 番号によるメッセージ取得
- dele: 削除
- quit

```
S: +OK POP3 server ready
C: user bob
S: +OK
C: pass hungry
S: +OK user successfully logged on

C: list
S: 1 498
S: 2 912
S: .
C: retr 1
S: <message 1 contents>
S: .
C: dele 1
C: retr 2
S: <message 1 contents>
S: .
C: dele 2
C: quit
S: +OK POP3 server signing off
```

2: Application Layer 52

POP3(さらに)とIMAP

POP3についてもっと

- 先の例では, “ダウンロード・消去”モードを利用
- ポップは, クライアントを変えたと再度電子メールを読むことができない
- “ダウンロード・保存”: 異なるクライアント上でメッセージのコピー
- POP3は, セッションをまたいでステートレス

IMAP

- 全メッセージを一箇所に保存: サーバ
- ユーザがフォルダないにメッセージをおくことを許す
- IMAPは, セッションをまたいで状態を保存:
 - フォルダ名とメッセージIDとフォルダ名間の対応付け

2: Application Layer 53

Chapter 2 内容

- 2.1 アプリケーション層プロトコルの原理
- 2.2 Web と HTTP
- 2.3 FTP
- 2.4 電子メール
 - SMTP, POP3, IMAP
- 2.5 DNS
- 2.6 TCPによるソケットプログラミング
- 2.7 UDPによるソケットプログラミング
- 2.8 Web サーバの構築
- 2.9 コンテンツ分配
 - ネットワークWebキャッシュ
 - コンテンツディストリビューションネットワーク
 - P2P ファイル共有

2: Application Layer 54

DNS: ドメインネームシステム

ピープル: 多くの識別子:

- SSN, 名前, パスポート番号
- インターネットホスト・ルータ:
- IP アドレス(32 bit) - データグラムのアドレッシングのために使用
 - “名前”, 例えば, `gaia.cs.umass.edu` が人によって使用される

疑問: IPアドレスと名前の間のマップは?

ドメインネームシステム:

- 多数の**ネームサーバの階層として実装された分散データベース**

- アプリケーション層プロトコル**

ホスト, ルータ, ネームサーバは, 名前を**解決**する(アドレス / 名前変換)ために通信する

- 注意: アプリケーション層プロトコルとして実装されるコアネットワーク機能
- ネットワークの“エッジ”における複雑さ

2: Application Layer 55

DNS ネームサーバ

なぜ中央型DNSでないのか?

- 単一故障ポイント
- トラフィック量
- 中央型データベースまでの距離
- 管理

スケールしない!

- サーバは, 全名前・IPアドレス対応をもたない

ローカルネームサーバ:

- 各ISP, 企業は, **ローカル (デフォルト)ネームサーバ**をもつ
- ホストのDNSクエリは, 最初にローカルネームサーバに問い合わせられる

オーソリタティブネームサーバ:

- あるホストに対し, ホストのIPアドレスと名前を保存
- そのホスト名に対する名前 / IPアドレス変換を実行できる

2: Application Layer 56

DNS: ルートネームサーバ

- 名前解決できないローカルネームサーバが問い合わせる
- ルートネームサーバ:
 - ネームマッピングが未知の場合, オーソリタティブネームサーバにコンタクト
 - マッピングの獲得
 - ローカルネームサーバにマッピングをリターン



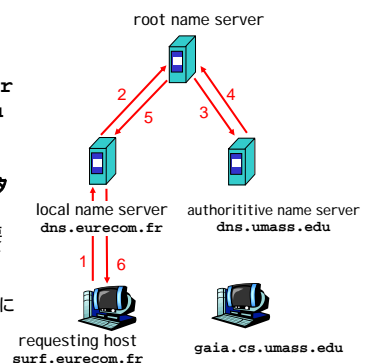
世界中には13のルートサーバがある

2: Application Layer 57

簡単な DNS 例

ホスト `surf.eurecom.fr`
`gaia.cs.umass.edu`
のIPアドレスがほしい

- ローカルDNSサーバ `dns.eurecom.fr` にコンタクト
- `dns.eurecom.fr` は, 必要に応じてルートネームサーバに問い合わせ
- ルートネームサーバは, 必要に応じてオーソリタティブネームサーバ `dns.umass.edu` に問い合わせ

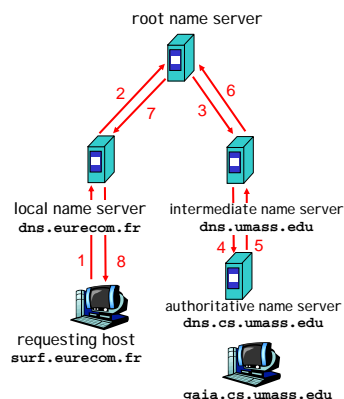


2: Application Layer 58

DNS 例

ルートネームサーバは

- オーソリタティブネームサーバを知らないかもしれない
- オーソリタティブネームサーバに問い合わせる**仲介ネームサーバ**を知っているかもしれない



2: Application Layer 59

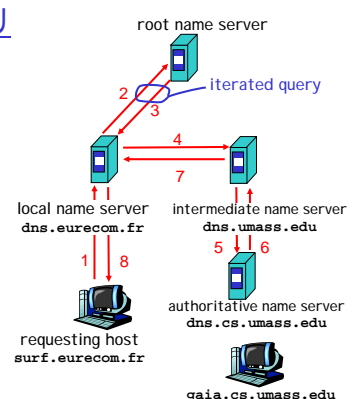
DNS: 反復クエリ

再帰クエリ:

- 問い合わせられたネームサーバ側に名前解決の負担をおわせる
- 高負荷?

反復クエリ:

- 問い合わせられたサーバは, 問い合わせすべきサーバ名を応答する
- “この名前を知りませんが, このサーバに聞いてください”



2: Application Layer 60

DNS: キャッシュとレコードの更新

- いったん(ある)ネームサーバがマッピングを知ると、マッピングをキャッシュする
 - ある時間の後、キャッシュエントリはタイムアウトする(消える)
- IETFによる設計にもとづく更新/通知メカニズム
 - RFC 2136
 - <http://www.ietf.org/html.charters/dnsind-charter.html>

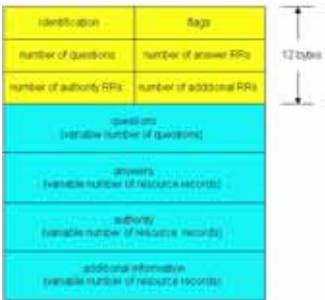
DNS レコード

- DNS: リソースレコード(RR)を格納する分散データベース
- RR フォーマット: (name, value, type, ttl)
- Type=A
 - name: ホスト名
 - value: IP アドレス
 - Type=CNAME
 - name: 公式名("canonical")に対する別名
 - value: 正式名
www.ibm.com は実際には servereast.backup2.ibm.com
 - Type=NS
 - name: ドメイン (例 foo.com)
 - value: このドメインのオーソリティティブネームサーバのIPアドレス
 - Type=MX
 - value: name に関するメールサーバ

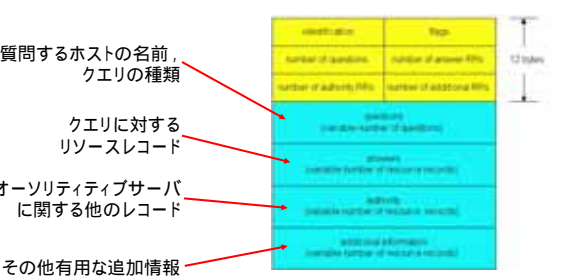
DNS プロトコル、メッセージ

DNS プロトコル: クエリと 応答 メッセージはともに同じメッセージフォーマットをもつ

- メッセージヘッダ
- 識別子(identification): クエリに対する16 bit番号, 応答は同一番号を使用
 - フラグ(flags):
 - クエリか応答か
 - 再帰要望
 - 再帰利用可能
 - 応答がオーソリティティブによる



DNS プロトコル、メッセージ



Chapter 2 内容

- 2.1 アプリケーション層プロトコルの原理
- 2.2 Web と HTTP
- 2.3 FTP
- 2.4 電子メール
 - SMTP, POP3, IMAP
- 2.5 DNS
- 2.6 TCPによるソケットプログラミング
- 2.7 UDPによるソケットプログラミング
- 2.8 Web サーバの構築
- 2.9 コンテンツ分配
 - ネットワークWebキャッシュ
 - コンテンツディストリビューションネットワーク
 - P2P ファイル共有

ソケットプログラミング

目標: ソケットを使って通信するクライアント/サーバアプリケーションの構築方法を学ぶ

- Socket API
- BSD4.1 UNIXによって1981に導入
 - アプリにより陽に生成、使用、開放される
 - クライアント/サーバパラダイム
 - ソケットAPIを介した2種類のトランスポートサービス:
 - 低信頼、データグラム
 - 高信頼、バイトストリーム

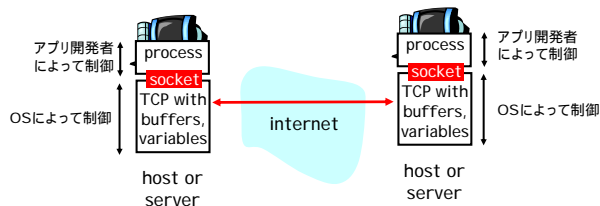
ソケット(socket)

ホスト内で、アプリケーションにより生成され、OSによりコントロールされるインタフェース(ドア)であり、これを介してアプリケーションが、他のアプリケーションプロセスにメッセージを送受信できる。

TCPを使ったソケットプログラミング

Socket: アプリケーションプロセスとトランスポートプロトコル(UDP,TCP)との窓口

TCPサービス: プロセスからプロセスへのバイトの高精度信頼性転送



2: Application Layer 67

TCPによるソケットプログラミング

クライアントがサーバにコンタクトしなければならない

- サーバはすでに稼働している
- サーバはクライアントのコンタクトを受け付けるソケットをすでに生成

クライアントはサーバに次のようにコンタクトする:

- クライアント内のTCPソケットを生成
- サーバのIPアドレスとプロセスのポート番号を記述
- クライアントがソケットを生成すると、クライアントTCPはサーバTCPにコネクションを生成

□ クライアントによってコンタクトされると、サーバプロセスがクライアントと通信するために、**サーバTCPはあたかもソケットを生成**

- サーバが複数のクライアントと通信することを可能にする
- ソースのポート番号がクライアントを識別するために使用される (詳しくは3章)

アプリケーションの観点から

TCPは、クライアントサーバ間に、高精度、順序保存バイト転送("パイプ")を提供

2: Application Layer 68

ストリームに関する専門用語

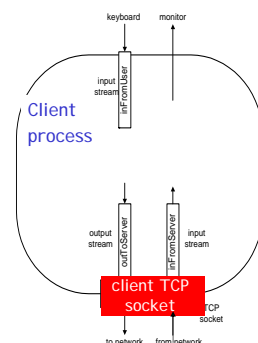
- **ストリーム**とは、プロセスに流入、流出する文字列
- **入力ストリーム**は、プロセスに対する入力ソース(例えば、キーボードやソケット)に取り付けられる
- **出力ストリーム**は、出力ソース(例えば、モニタやソケット)に取り付けられる

2: Application Layer 69

TCPによるソケットプログラミング

クライアント・サーバアプリの例:

- 1) クライアントは、標準入力 (inFromUser stream) から行を読み、ソケット(outToServer stream)を介してサーバに送信する
- 2) サーバはソケットから行を読み込む
- 3) サーバは読み込んだ行を大文字に変換し、クライアントに送信する
- 4) クライアントは、ソケット (inFromServer stream) から、変更された行を読み込み、表示する

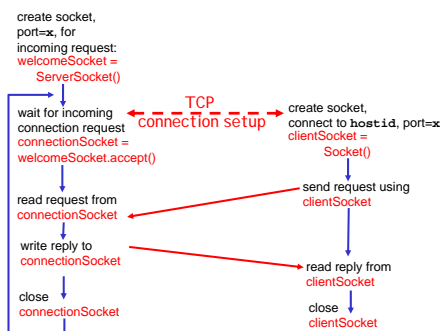


2: Application Layer 70

クライアント/サーバ ソケット インタラクション : TCP

Server (running on hostid)

Client



2: Application Layer 71

Java クライアントの例 (TCP)

```
import java.io.*;
import java.net.*;
class TCPClient {

    public static void main(String argv[]) throws Exception
    {
        String sentence;
        String modifiedSentence;

        Create input stream → BufferedReader inFromUser =
                               new BufferedReader(new InputStreamReader(System.in));
        Create client socket, connect to server → Socket clientSocket = new Socket("hostname", 6789);
        Create output stream attached to socket → DataOutputStream outToServer =
                                                  new DataOutputStream(clientSocket.getOutputStream());
```

2: Application Layer 72

Java クライアントの例 (TCP) (つづき)

```
        }
        Create input stream attached to socket }
        BufferedReader inFromServer =
        new BufferedReader(new
        InputStreamReader(clientSocket.getInputStream()));

        sentence = inFromUser.readLine();

        Send line to server }
        outToServer.writeBytes(sentence + '\n');

        Read line from server }
        modifiedSentence = inFromServer.readLine();

        System.out.println("FROM SERVER: " + modifiedSentence);

        clientSocket.close();
    }
}
```

2: Application Layer 73

Java サーバの例 (TCP)

```
import java.io.*;
import java.net.*;

class TCPServer {

    public static void main(String argv[]) throws Exception
    {
        String clientSentence;
        String capitalizedSentence;

        Create welcoming socket at port 6789 }
        ServerSocket welcomeSocket = new ServerSocket(6789);

        Wait, on welcoming socket for contact by client }
        while(true) {
            Socket connectionSocket = welcomeSocket.accept();

            Create input stream, attached to socket }
            BufferedReader inFromClient =
            new BufferedReader(new
            InputStreamReader(connectionSocket.getInputStream()));
        }
    }
}
```

2: Application Layer 74

Java サーバの例 (TCP) (つづき)

```
        }
        Create output stream, attached to socket }
        DataOutputStream outToClient =
        new DataOutputStream(connectionSocket.getOutputStream());

        Read in line from socket }
        clientSentence = inFromClient.readLine();

        capitalizedSentence = clientSentence.toUpperCase() + '\n';

        Write out line to socket }
        outToClient.writeBytes(capitalizedSentence);
    }
}
End of while loop,
loop back and wait for
another client connection
```

2: Application Layer 75

Chapter 2 内容

- 2.1 アプリケーション層プロトコルの原理
- 2.2 Web と HTTP
- 2.3 FTP
- 2.4 電子メール
 - SMTP, POP3, IMAP
- 2.5 DNS
- 2.6 TCPによるソケットプログラミング
- 2.7 UDPによるソケットプログラミング
- 2.8 Web サーバの構築
- 2.9 コンテンツ分配
 - ネットワークWebキャッシュ
 - コンテンツディストリビューションネットワーク
 - P2P ファイル共有

2: Application Layer 76

UDPによるソケットプログラミング

UDP: クライアント、サーバ間に“コネクション”なし

- ハンドシェイクなし
- 送信側は、各パケットごとに陽にあて先のIPアドレスとポート番号を付加
- サーバは、受信パケットから送信側のIPアドレスとポート番号を抽出する必要あり

UDP: 送信データは、順序どおりに来ないかもしれないし、失われるかもしれない

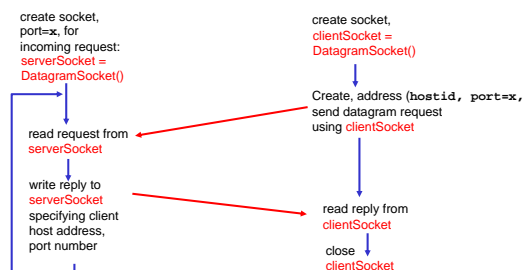
アプリケーションの観点から
UDP は、クライアント、サーバ間の、低信頼な、バイトのグループ(“データグラム”)の転送を提供する

2: Application Layer 77

クライアント/サーバ ソケット インタラクション : UDP

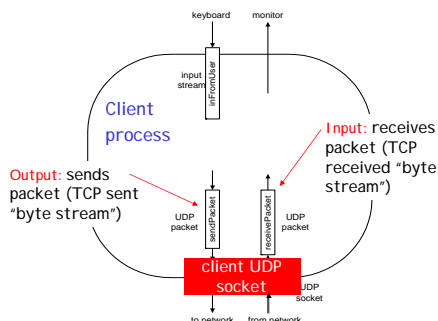
Server (running on *hostid*)

Client



2: Application Layer 78

Java クライアントの例 (UDP)



2: Application Layer 79

Java クライアントの例 (UDP)

```
import java.io.*;
import java.net.*;

class UDPClient {
    public static void main(String args[]) throws Exception
    {
        // Create input stream
        BufferedReader inFromUser =
            new BufferedReader(new InputStreamReader(System.in));

        // Create client socket
        DatagramSocket clientSocket = new DatagramSocket();

        // Translate hostname to IP address using DNS
        InetAddress IPAddress = InetAddress.getByName("hostname");

        byte[] sendData = new byte[1024];
        byte[] receiveData = new byte[1024];

        String sentence = inFromUser.readLine();
        sendData = sentence.getBytes();
```

2: Application Layer 80

Java クライアントの例 (UDP) (つづき)

```
        // Create datagram with data-to-send, length, IP addr, port
        DatagramPacket sendPacket =
            new DatagramPacket(sendData, sendData.length, IPAddress, 9876);

        // Send datagram to server
        clientSocket.send(sendPacket);

        // Read datagram from server
        DatagramPacket receivePacket =
            new DatagramPacket(receiveData, receiveData.length);
        clientSocket.receive(receivePacket);

        String modifiedSentence =
            new String(receivePacket.getData());

        System.out.println("FROM SERVER:" + modifiedSentence);
        clientSocket.close();
    }
}
```

2: Application Layer 81

Java サーバの例 (UDP)

```
import java.io.*;
import java.net.*;

class UDPServer {
    public static void main(String args[]) throws Exception
    {
        // Create datagram socket at port 9876
        DatagramSocket serverSocket = new DatagramSocket(9876);

        byte[] receiveData = new byte[1024];
        byte[] sendData = new byte[1024];

        while(true)
        {
            // Create space for received datagram
            // Receive datagram
            DatagramPacket receivePacket =
                new DatagramPacket(receiveData, receiveData.length);
            serverSocket.receive(receivePacket);
```

2: Application Layer 82

Java サーバの例 (UDP) (つづき)

```
        String sentence = new String(receivePacket.getData());

        // Get IP addr, port #, of sender
        InetAddress IPAddress = receivePacket.getAddress();
        int port = receivePacket.getPort();

        String capitalizedSentence = sentence.toUpperCase();

        sendData = capitalizedSentence.getBytes();

        // Create datagram to send to client
        DatagramPacket sendPacket =
            new DatagramPacket(sendData, sendData.length, IPAddress,
                               port);

        // Write out datagram to socket
        serverSocket.send(sendPacket);
    }
}
```

End of while loop, loop back and wait for another datagram

2: Application Layer 83

簡単なWebサーバの構築

- ひとつのHTTPリクエストを扱う
- リクエストを受け付ける
- ヘッダを解析する
- サーバのファイルシステムから要求されたファイルを取り出す
- HTTP 応答メッセージを生成する
 - ヘッダ行 + ファイル
- クライアントに応答を返す
- サーバを生成後, ブラウザ (例, IE explorer) を使ってファイル要求を出すことができる
- 詳細はテキストを参照のこと

2: Application Layer 84

ソケットプログラミング: 参考文献

C-language tutorial (audio/slides):

- "Unix Network Programming" (J. Kurose),
<http://manic.cs.umass.edu/~amldemo/courseware/>.

Java-tutorials:

- "All About Sockets" (Sun tutorial),
<http://www.javaworld.com/javaworld/jw-12-1996/jw-12-sockets.html>
- "Socket Programming in Java: a tutorial,"
<http://www.javaworld.com/javaworld/jw-12-1996/jw-12-sockets.html>

2: Application Layer 85

Chapter 2 内容

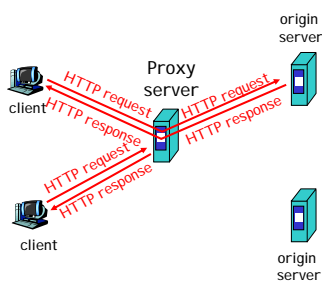
- 2.1 アプリケーション層プロトコルの原理
- 2.2 Web と HTTP
- 2.3 FTP
- 2.4 電子メール
 - SMTP, POP3, IMAP
- 2.5 DNS
- 2.6 TCPによるソケットプログラミング
- 2.7 UDPによるソケットプログラミング
- 2.8 Web サーバの構築
- 2.9 コンテンツ分配
 - ネットワークWebキャッシュ
 - コンテンツディストリビューションネットワーク
 - P2P ファイル共有

2: Application Layer 86

Webキャッシュ(プロキシサーバ)

目的: オリジナルサーバを必要とせずクライアントの要求を満足させる

- ユーザは、キャッシュを介してWebアクセスするようブラウザに設定
- ブラウザは、全てのHTTP要求をキャッシュに送信
 - キャッシュ内にオブジェクトあり: キャッシュはオブジェクトを返信
 - キャッシュ内にオブジェクトなし: キャッシュはオリジナルサーバにオブジェクトを要求し、クライアントにオブジェクトに送信する



2: Application Layer 87

Webキャッシュについてもっと

- キャッシュは、クライアントとサーバとして動作
- キャッシュは、HTTPヘッダのIf-modified-sinceを使って最新かどうかを評価
 - 問題: キャッシュはリスクをとってチェックすることなくキャッシュされたオブジェクトを配送するべきか?
 - 発見的方法を使用
- 通常、ISP(大学、企業、地域ISP)によりキャッシュが実装される

なぜWebキャッシュか?

- クライアント要求に対する応答時間の減少
- アクセスリンクのトラフィックを削減
- インターネット内におかれたキャッシュにより、“ブアー”なコンテンツプロバイダが効率よくコンテンツ配信をすることが可能になる

2: Application Layer 88

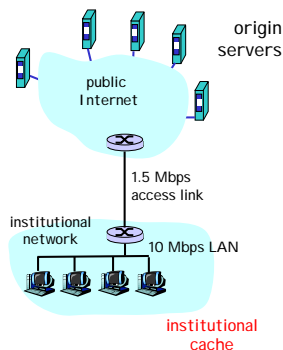
キャッシュ例 (1)

仮定

- 平均オブジェクトサイズ= 100,000 bits
- ブラウザからオリジナルサーバへの平均要求レート= 15/sec
- 組織のルータからオリジナルサーバへの往復遅延 = 2 sec

結果

- LAN利用率 = 15%
- アクセスリンク利用率 = 100%
- トータル遅延 = インターネット遅延 + アクセス遅延 + LAN 遅延 = 2 秒 + 数分 + 数ミリ秒



2: Application Layer 89

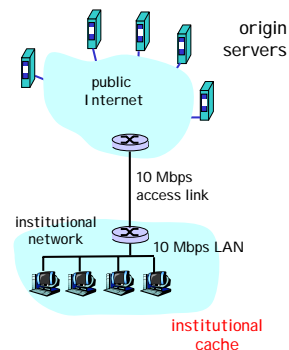
キャッシュ例 (2)

可能な解

- アクセスリンクの増強, たとえば 10 Mbps

結果

- LAN利用率 = 15%
- アクセスリンク利用率 = 15%
- トータル遅延 = インターネット遅延 + アクセス遅延 + LAN 遅延 = 2 秒 + 数ミリ秒 + 数ミリ秒
- 回線増強はコストがかかる



2: Application Layer 90

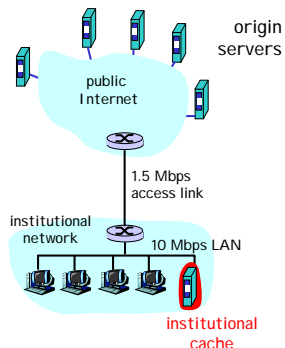
キャッシュ例 (3)

キャッシュの設置

- ヒット率 = 0.4 と仮定

結果

- 40% のリクエストがキャッシュにより満たされる
- 60% のリクエストがオリジナルサーバに満たされる
- アクセスリンク利用率 60% に削減, 遅延は無視できる (10ミリ秒)
- トータル遅延 = インターネット遅延 + アクセス遅延 + LAN遅延
= $0.6 \times 2 \text{ 秒} + 0.6 \times 0.01 \text{ 秒} + \text{数ミリ秒} < 1.3 \text{ 秒}$



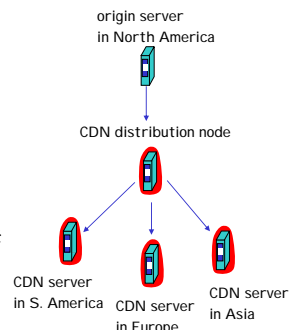
2: Application Layer 91

コンテンツ分散ネットワーク (CDN)

- コンテンツプロバイダは, CDNの顧客

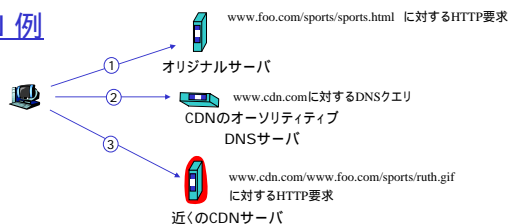
コンテンツ複製

- CDN会社は, インターネット内に数百のCDNサーバを設置
 - より低レベルISP, ユーザの近くに
- CDNサーバ内に顧客のコンテンツを複製. プロバイダがコンテンツを更新すると, CDNはサーバを更新する



2: Application Layer 92

CDN 例



オリジナルサーバ

- www.foo.com
- HTMLの配信
- $\text{http://www.foo.com/sports.ruth.gif}$ を $\text{http://www.cdn.com/www.foo.com/sports/ruth.gif}$ で置換

CDN会社

- cdn.com
- GIFファイルの配信
- オーソリティティブDNSがリクエストを適切なCDNサーバに向ける

2: Application Layer 93

CDNについてもっと

リクエストの経路制御

- CDN は, ISPとCDNノード間の距離を示す“マップ”を生成
- クエリがオーソリティティブDNSに到着したとき:
 - サーバは, クエリ送信元からのISPからかを決定
 - 最適なCDNサーバを“マップ”を使って決定

Webページだけではない

- 蓄積ストリーミング音声/映像
- 実時間ストリーミング音声/映像
 - CDNノードは, アプリケーション層オーバーレイネットワークを構築

2: Application Layer 94

P2P ファイル共有

例

- アリスは, P2PクライアントアプリをノートPCで実行
- 断続的にインターネットに接続, 各コネクションに対する新しいIPアドレスを獲得
- “Hey Jude”が欲しい
- アプリケーションは, “Hey Jude”をもつピアを表示

- アリスはピアの中からボブを選択
- ボブのPCからアリスのノートPCにファイルコピー: HTTP
- アリスがダウンロード中, 他のユーザがアリスからダウンロード
- アリスのピアは, Webクライアントであり一時的なWebサーバである

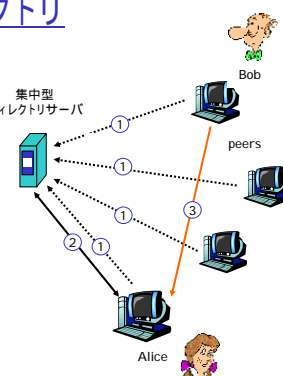
全ピアがサーバ = スケーラブル!

2: Application Layer 95

P2P: 集中型ディレクトリ

オリジナルな “Napster” の設計

- 1) ピアがコネクトすると, 中央サーバに通知:
 - IPアドレス
 - コンテンツ
- 2) アリスは “Hey Jude” に対するクエリをだす
- 3) アリスはボブにファイルを要求する



2: Application Layer 96

P2P: 集中型ディレクトリの問題

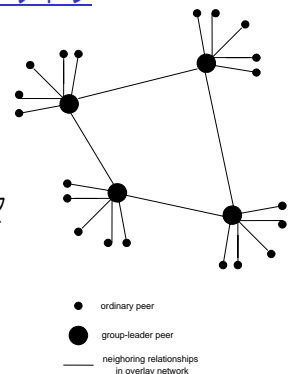
- 単一故障ポイント
- 性能ボトルネック
- 著作権侵害

ファイル転送は分散化されるが、ファイル位置特定は極度に集中化されている

2: Application Layer 97

P2P: 分散型ディレクトリ

- 各ピアはグループリーダであるか、グループリーダが割り当てられる
- グループリーダは、全ての子のコンテンツをトラック
- ピアは、グループリーダにクエリをだす; グループリーダは他のグループリーダにクエリを出すかもしれない



2: Application Layer 98

分散型ディレクトリについてもっと

オーバーレイネットワーク

- ピアがノード
- ピアと自分のグループリーダとの間のエッジ
- いくつかのグループリーダ間のエッジ
- 仮想ネイバー
- ブートストラップノード
- 接続ピアはあるグループリーダが割り当てられるか、リーダーとして選定される

長所

- 非集中型サーバ
 - ピアにわたって分散されたロケーションサービス
 - シャットダウンがより困難になる

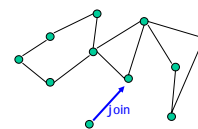
短所

- ブートストラップノードが必要
- グループリーダが過負荷になりうる

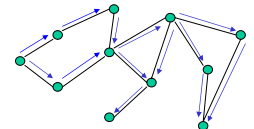
2: Application Layer 99

P2P: クエリ・フラッディング

- Gnutella
- 非階層
- 他ノードについて知るためにブートストラップノードの使用
- 参加メッセージ



- 隣接ノードにクエリを送信
- 隣接ノードはクエリをフォワード
- クエリを受けたピアがオブジェクトを持つ場合、クエリを出したピアにメッセージを送信



2: Application Layer 100

P2P: クエリ・フラッディングについてもっと

長所

- ピアは同様な責任をもつ: グループリーダなし
- 高度に分散化
- どのピアもディレクトリ情報を維持しない

短所

- 過度のクエリトラフィック
- クエリ半径: あるコンテンツが見つからないかもしれない
- ブートストラップノードが必要
- オーバレイネットワークのメンテナンスが必要

2: Application Layer 101

Chapter 2: まとめ

ネットワークアプリケーションについての学習はあわり！

- アプリケーションサービスの要求:
 - 信頼性, 帯域, 遅延
- クライアント・サーバパラダイム
- インターネットトランスポートサービスモデル
 - コネクション指向型, 高信頼: TCP
 - 低信頼, データグラム: UDP
- 特定のプロトコル:
 - HTTP
 - FTP
 - SMTP, POP, IMAP
 - DNS
- ソケットプログラミング
- コンテンツディストリビューション
 - キャッシュ, CDN
 - P2P

2: Application Layer 102

Chapter 2: まとめ

もっとも重要なこと: プロトコルについて学んだこと

- 典型的リクエスト/リプライメッセージ交換:
 - クライアントは、情報またはサービスを要求
 - サーバは、データ、ステータスコードにより応答
- メッセージフォーマット:
 - ヘッダ: データに関する情報を与えるフィールド
 - データ: 通信される情報
- 制御メッセージ 対 データメッセージ
 - in-band, out-of-band
- 集中型 対 分散型
- ステートレス型 対 ステート型
- 高信頼 対 低信頼型メッセージ転送
- ネットワークエッジにおける複雑さ
- セキュリティ: 認証