Computer Networking
A Top-Down Approach pp.144-169

James F. Kurose, Keith W. Roth

情報工学科 3 年 寺岡研究室
学籍番号 61619027 名前 安森涼

2019 年 2 月 14 日

1　Peer to Peer アプリケーション

Web, Email, DNS のようにクライアントとサーバの関係にあるものは基盤サーバ側に信頼が必要である．一方で Peer to Peer(P2P) アプリケーションは基盤サーバ側に信頼を必要としない．その代わり，ピアという断続的につながっているホストの組が必要で，これはサービスプロバイダではなく，ユーザのデスクトップやノートパソコン間で提供されるものである．この節では，P2P によく適している二つの異なるアプリケーションを説明する．一つ目がファイル分配で単一のソースから多くのピアにファイルを送信するものである．この例として，最も一般的な P2P ファイル分配プロトコルである BitTorrent を紹介する．二つ目が分散データベースで，これは巨大なピアの共同体により配信されるデータベースである．ここで Database Hash Tables (DHT) について述べる．

1.1　ファイル分配

P2P では Linux OS のアップデート，ソフトウェア，MP3 音楽ファイル，MPEG ビデオファイルなどの大容量のファイルを単一のサーバから多数のホストに送ることを考える．クライアントサーバ構造でのファイル分配ではサーバがピアに大きなファイルを送る．この際，サーバのチア息を大きく占有してしまうという問題がある．P2P 構造のファイル分配ではピア側のどの部分でも再分配できる．そのためサーバの帯域を占有せずにファイルの送信が可能である．Web ブラウザが HTTP を形成しているように，BitTorrent がプロトコルを形成している．

1.1.1　スケーラビリティ

スケーラビリティとは利用者や仕事の増大に適応できる能力度合いのことである．これを説明するのに後述の図 1 のようなピアとサーバがリンクでつながっている構造を考え，P2P 構造とクライアントサーバ構造で，全てのピアへの送信にかかる時間を比較する．ここでサーバのアップロード速度を u_s，i 番目のピアのアップロード速度を d_i，i 番目のピアのダウンロード速度を d_i，ファイルの大きさを F，送信するピアの数を N とする．本文章では N 個全てのピアがダウンロードするのにかかる時間を元にスケーラビリティを比較する．この時間を以降分配時間と呼ぶ．また，条件を公平にするために，仮定としてインターネットが十分な帯域を保持していると考え、サーバとクライアントが他のネットワークに参加していないと考える．まず，クライアントサーバ構造について分配時間を D_{cs} と考える．サーバがピアへ送信することとピアが受信すること 2 つの面を考える．以降，データの容量の単位を bit，時間の単位を (hour)，送信速度を　(bit/hour) として考える．まず，前者について，送信するデータの総容量は NF であり，送信速度は前述の通り，u_s である．ゆえに，送信時間は NF/u_s で表される．続いて，後者については多数のピアの中で最短の受信速度は $d_{min} = \min\{d_1 \sim d_N\}$ であるため，受信時間は F/d_{min} で表される．これらより送信にかかる時間は前述の 2 側面のうち長い方なので

$$D_{cs} = \max\{\frac{NF}{u_s}, \frac{F}{d_{min}}\} \tag{1}$$

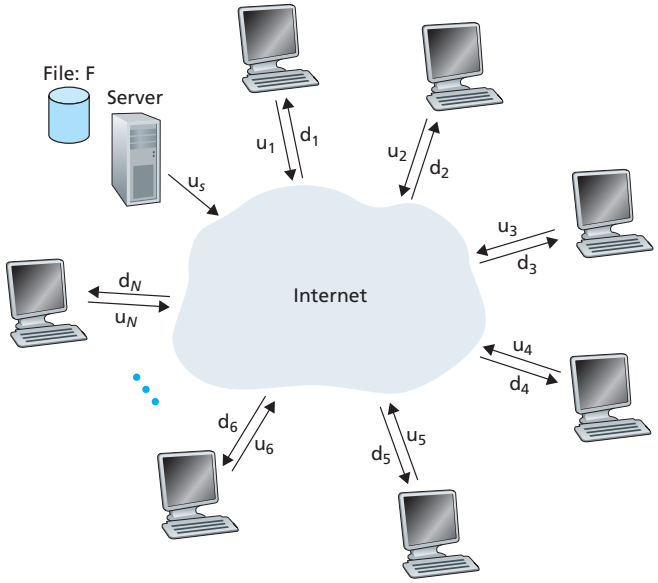

図 1　ピアとサーバがリンクでつながっている構造

と表される．これより N が十分に大きいときに N の大きさに伴い線形に増加することがわかる．

続いて，P2P 構造の分配時間を D_p と考える．サーバのピアへの送信にかかる時間，ピアの受信時間とシステム全体の送信能力から考える時間の 3 つの観点から考える．まず，1 つ目の観点において，サーバは 1 度だけアクセスリンクに送信するので最短の送信にかかる時間は F/u_s である．また，2 つ目の観点についてはサーバクライアント構造と同様に F/d_{min} である．最後の観点について，サーバとここのピアノ送信能力の和とシステム全体の送信能力は同等と考えられる．送信するデータの総容量は NF，送信速度は $u_s + \sum_{i=1}^{N}$ である．ゆえにこの観点からは少なくとも分配時間が $\frac{NF}{u_s+\sum_{i=1}^{N}}$ と表される．故に分配時間が

$$D_p = \max\{\frac{F}{u_s}, \frac{F}{d_{min}}, \frac{NF}{u_s + \sum_{i=1}^{N}}\} \tag{2}$$

と表される．

式 (1)，(2) より送信時間の比較を行う．ピアの送信速度を一律 u (bit/hour)，$F/u + 1$ (hour)，$u_s = 10u$，$d_{min} \geqq u_s$ とすると図 2 より常に P2P の送信時間が短く，N がどんな値でも P2P の送信時間が 1 時間未満であることがわかる．

1.1.2　BitTorrent

続いて BitTorrent について記述する．これはファイル分配における有名な P2P プロトコルである．このプロトコルの名前になっているトレントというものはあるファイルの分配に参加している全てのピアの集合である．このトレント内のピアがファイルを分割したチャンクを受信し，ピアが受信すると同時にチャンクの送

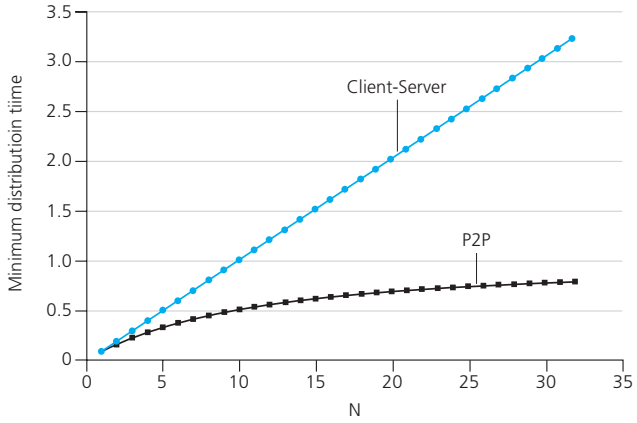

図 2　クライアントサーバ構造と P2P 構造の送信時間の比較

信も行う．ピアが全体のファイルを取得するとトレントを離脱する，またはトレントに残り他のピアにチャンクを送信する．また，トレント内のピアの特徴としていつでもトレントの離脱と加入が可能である．また，トレントがトラッカというノードを所持していることで，トラッカはトレントに加入したピアの IP アドレスを保持することで，トレント内のピアの存在の是非をトラッカに定期通知することでトレント内のピアの追跡を行うことができる．例えば Alice というピアを考慮すると，Alice は他のピアと通信を行い，トラッカから通信に成功したピアノ IP アドレスを取得する．このように通信に成功したピアを以降隣接ピアと呼ぶ．Alice は隣接ピアの通信確立を繰り返し行い，隣接ピアが所持しているチャンクのリストを入手し，これを元に所持していないチャンクの要求をする．このチャンクの要求方法として，rarest first と unchoked がある．前者は隣接ピアに存在する複製が少ない貴重なチャンクから要求を行う．隣接ピアの中でコピーされている回数が少ないものから貴重なチャンクを判断する．後者はデータの割合が高いペア (以下 unchoked と示す) を 4 つ選択肢，この 4 つのピアに自身のチャンクを送信し，一定間隔ごとに新しい相手を探し，チャンクを送信を繰り返す．他のピアの unchoked の 1 つとなったらチャンクを取得できる．これによりトレント内のチャンクのコピーの個数が一定に近づく．

1.2　分散データベース

1.2.1　Distributed Hash Tables (DHT)

続いて DHT について述べる．これは (キー，値) のペアを持つ単純なデータベースで P2P 構造は多数のピアにペアの一部を保存することでどのピアにも特定のキーで問い合わせや挿入が可能になるものである．ピアに nbit 表現で ID を割り振り，キーにも同じ範囲の整数で ID の割り振りを実施する．そしてキーを自身と最も近い ID を持つピアに格納をすることでペアを保存している．元々のキーが数字以外の場合はハッシュ化し，数値化を行う．ハッシュ化は 7 章で取り扱うため深くは述べないが，これに用いられるハッシュ関数は多対 1 関数であるため，出力が等しくなることがあるが，出力が同じ場合の入力の差は非常に少ないので同じものとみなすことができる．キーに自身と最も近いピアの選択を行うさい，全てのピアを探索するのは不可能である．本レポートでは円形 DHT について説明する．

1.2.2　円形 DHT

円形 DHT とは図 3 のように円形に配置されたピアのことである．それぞれのピアは前後のピアの情報を保持し，ピアにキーの情報がないとき，次のピアを調査することで探索を行うものである．ピアが自由に参加離脱することから削除挿入において配列より計算が早いリストを利用しており，リストの途中の要素からたどり始めたとしても、リスト全体を一周することが容易であるため循環リストを利用し，突然の挿入削除に対して対応できる双方向型のリストを採用している．例えば図 3 のように [0,15] であるこの中

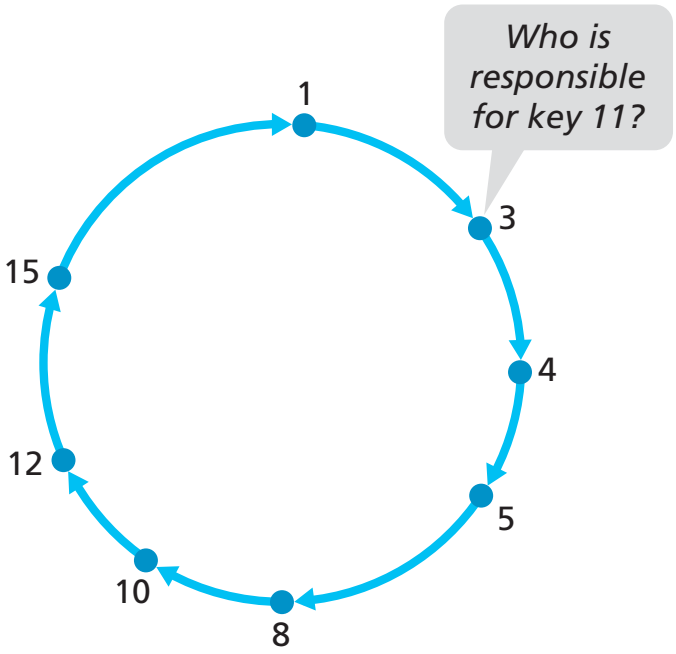

図 3　円形 DHT

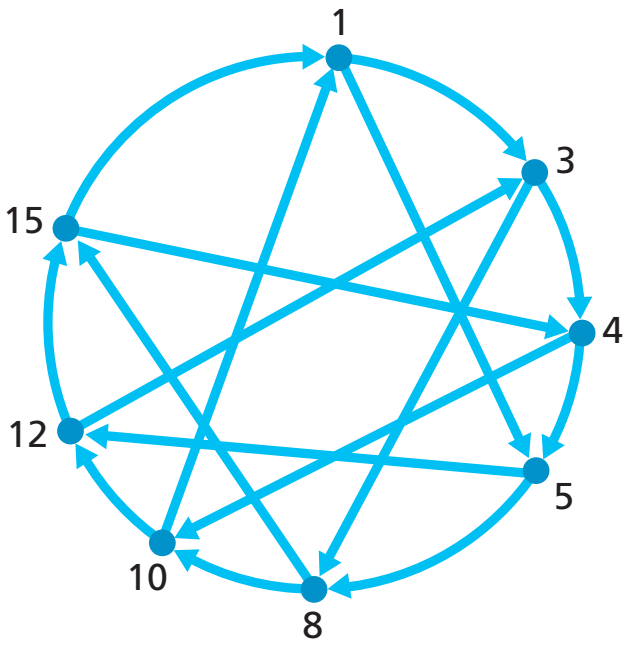

図 4　ショートカット付き円形 DHT

に 8 つのピアが存在し，その ID を 1,3,4,5,8,10,12,15 とする．今回，ピア 3 から探索をし，挿入するペアのキーが 11 とする．探索を ID3,4,5,8,10,12 の順に行う．今回は自身に続く最も近いピアを選択するという規則にすると，12 の ID が割り振られたピアにペアを格納することとなる．この方式によりピアは 2 つの隣接ピアの情報のみ保持する．このため保持する情報を削減することができる．ただしこの方法には欠点があり，ノード数を N とすると最悪の場合 N ノード全てに問い合わせを行い，平均で $N/2$ ノードに問い合わせを行う必要がある．このように保持する隣接ピアの情報の個数と問い合わせの回数はトレードオフの関係にある．この問い合わせの回数を減らすために図 4 のようにショートカットを追加することが必要である．

1.2.3　ピアチャーン

また P2P においてピアがトレントから自由に参加や離脱が可能という点を DHT の設計において考慮が必要である．これをピ

UDPClient.py

```
from socket import *
serverName = 'hostName'
serverPort = 12000
clientSocket = socket(socket.AF_INET, socket.SOCK_DGRAM)
message = raw_input('Input lowercase sentence:')
clientSocket.sendto(message,(serverName, serverPort))
modifiedMessage, serverAddress = clientSocket.recvfrom(2048)
print modifiedMessage
clientSocket.close()
```

図 5　UDP クライアントのコード

UDPServer.py

```
1. from socket import *
2. serverPort = 12000
3. serverSocket = socket(AF_INET, SOCK_DGRAM)
4. serverSocket.bind(('', serverPort))
5. print "The server is ready to receive"
6. while 1:
7.     message, clientAddress = serverSocket.recvfrom(2048)
8.     modifiedMessage = message.upper()
9.     serverSocket.sendto(modifiedMessage, clientAddress)
```

図 6　UDP サーバのコード

アチャーンと呼ぶ．離脱時にその次にピアの情報を取得し，円形 DHT を保持することである．例えば図 3 においてピア 5 が離れた時にピア 4 はピア 8 の情報を保持しているのでピア 8 にピア 10 の情報を請求し，ピア 4 の保持している情報に加える．これにより，ピアの自由な参加離脱に対応ができるようになる．

2　ソケットプログラミング

ソケット API(Socket Application Programming Interface) は，アプリケーションに対して，異なるホスト間での通信をサポートするためのトランスポート層以下の機能を提供するプログラミングインタフェースで，クライアント/サーバモデルを実現するために用いられるものである．UDP と TCP にてソケットプログラミングを実施するため，それぞれについて確認する．これらはトランスポート層のプロトコルであり，前者はコネクションレスで信頼性がなく輻輳制御を行わないため容量が小さいプロトコルであり，後者はコネクション指向通信を行い，信頼性があり，輻輳制御を行うため容量が大きいプロトコルである．これらを用いたプログラムを Python のプログラムを元に説明する．これは鍵ソケットの考えが明瞭であり，また JAVA や C,C++ に記法が似ているのでプログラムが読みやすいという点を考慮した結果である．今回ソケットプログラミングを行う上で UDP，TCP 共に
1. クライアントが一連の文字入力をサーバに送信
2. サーバがサーバが受信した文字入力を大文字に変換
3. サーバが変換後のデータをクライアントに送信
4. クライアントは受信データをスクリーンに表示
の流れでプログラミングを行う．

2.1　UDP ソケットプログラミング

UDP のクライアント，サーバプログラムはそれぞれ UDPClient.py，UDPServer.py で表される．UDP では送信前にパケットに宛先アドレスを付加し，受信プロセスまでの道を決定する．その後パケット受信時にソケットから抽出を行い，内容を確認し適切な振る舞いを実施する．宛先アドレスの内部には目的地の IP アドレスとポート番号の情報が含まれている．IP アドレスだけで目的のホストに行くことが可能であるが，ホストは多数のネットワークプロセルを実行しているため，ホストが持つ複数のソケットを区別するために存在する．

2.1.1　UDP クライアント

UDP クライアントについてのコードは図 5 のように示す．1 行目は socket モジュールのインクルードを行い，2 行目でサーバの名前の決定により IP アドレスを設定している．3 行目でポート番号を 12000 に指定し，4 行目でソケット生成を行い，受信プロセスまでの道を決定している．続いて 5 行目で，プロンプトからの文字入力を変数 message 代入している．6 行目ではメッセージ付パケットをクライアントソケットに送信し，7 行目でパケット到着後データを modifiedMessage に格納し，ソースアドレスを serverAddress に格納している．8,9 行目でスクリーンに表示しソケットを閉鎖している．

TCPClient.py

```
1. from socket import *
2. serverName = 'servername'
3. serverPort = 12000
4. clientSocket = socket(AF_INET, SOCK_STREAM)
5. clientSocket.connect((serverName,serverPort))
6. sentence = raw_input('Input lowercase sentence:')
7. clientSocket.send(sentence)
8. modifiedSentence = clientSocket.recv(1024)
9. print 'From Server:', modifiedSentence
10. clientSocket.close()
```

図 7　TCP クライアントのコード

2.1.2　UDP サーバ

UDP サーバについてのコードは図 6 のように示す．1,2,3 行目はクライアントと同じ動作を行い，4 行目でサーバのソケットにポート番号を対応させ，6 行目でクライアントのパケットを待機させ，7 行目でデータとソー 8 行目でメッセージを大文字に変換し 9 行目でアドレスからパケットをサーバのソケットに送信している．その後，インターネットがクライアントのアドレスに配達し，パケット送信後はサーバ側は他の UDP パケットを受信するまで待機状態を継続する．

2.2　TCP ソケットプログラミング

TCP は先述の通りコネクション指向でデータ送信前にハンドシェイクと TCP 接続を確立するものである．ハンドシェイクとは通信路確立後に通信規則を自動的に決定するものである．TCP 接続にはクライアントとサーバのソケットのアドレスを利用する．TCP 接続確立後はソケット経由でデータ送信を行う．UDP と同様に接触開始前にサーバのプロセスを先に実行し，データの送信はバイト単位で送信している．サーバプログラムは特別なソケットを保持していて，このソケットはクライアントの最初の接触を受け入れるためにある．これを別名歓迎用の扉と呼ぶ．TCP ソケット生成時，クライアントは歓迎用の扉のアドレスを指定し，これを用いてデータをやりとり捨．これにより，接続を特定のクライアントに制限している．TCP ソケット生成後 3 ウェイハンドシェイクを行う．これは TCP 接続を確立後にトランスポート層で実行されるもので，クライアントはサーバの歓迎用の扉をノックし，サーバはノックを検知し，ノックしたクライアントのみを通す新しいソケットを生成する．つまり通信を行うクライアントごとに新しい扉の作成を行っている．

2.2.1　TCP クライアント

TCP クライアントについてのコードは図 7 のように示す．1 から 4 行目,9 行目,10 行目は UDP クライアントとほぼ同じ動作で，5 行目ではサーバのアドレスを指定し，TCP 接続を開始し，5 行目実行後 3 ウェイハンドシェイク 6 行目で sentence に改行コー

TCPServer.py

```
1. from socket import *
2. serverPort = 12000
3. serverSocket = socket(AF_INET,SOCK_STREAM)
4. serverSocket.bind(('',serverPort))
5. serverSocket.listen(1)
6. print 'The server is ready to receive'
7. while 1:
8.      connectionSocket, addr = serverSocket.accept()
9.      sentence = connectionSocket.recv(1024)
10.     capitalizedSentence = sentence.upper()
11.     connectionSocket.send(capitalizedSentence)
12.     connectionSocket.close()
```

図 8　TCP サーバのコード

ドまでの文字列を代入している．7 行目では TCP 接続に sentence の文字列のバイトを送信する．この際，UDP と異なり，パケット背作成と宛先アドレスは必要がない．その後クライアントはサーバからのバイト返還を待機する．サーバがバイトを返還した後，modifiedSentence にバイトの文字列を格納している．

2.2.2　TCP サーバ

TCP サーバについてのコードは図 8 のように示す．1 から 4 行目,6 行目,9 から 11 行目は UDP サーバとほぼ同じ動作で，5 行目でクライアントからの TCP 通信リクエストの受信を行う．5 行目の関数の引数はリクエストを受け付けているキューの数である．8 行目でノックしたクライアントのみを通すソケットを作成し，その後 3 ウェイハンドシェイクを行い TCP 接続を確立する．12 行目で 8 行目に生成した通信用ソケットを閉鎖する．しかし serverSocket は開いたままで他のクライアントが文字を送信することが可能である．