2004－OS－96　(9)
2004／6／17

Tree構造とMesh構造に対応した大規模ネットワークゲームAgent

小杉 隆二[†]　河野 真治[††]

我々は数百万人規模のネットワークゲームを実現するために、Agent を用いたネットワークゲーム用インフラを提案している。ここでは、Agent を実装し、実際に、ネットワークトポロジーとして Tree 構造と Mesh 構造を持つゲームを提案しているインフラ上に作成し、人数的スケーラビリティの有無を検証する。

The large-scale online network game Agent supporting a tree structure and a mesh structure

Ryuji Kosugi[†] and Shinji Kono[††,?]

We have proposed a infrastructure for online network game in order to realize the online network game of millions of people scale. In this paper, an agent based infrastructure is implemented. For games which have tree structure and mesh structure as a network topology, number-scalability is measured.

1. はじめに

ここ数年で、ネットワークゲームを取り巻く環境は大きく変化した。PC 用ネットワークゲームの台頭、有名タイトルのネットワークゲームへの進出、小型ゲーム機の増加、そして、家庭用ゲーム機のネットワークへの対応などが挙げられる。また、インターネットの普及･発達と共に、ネットワークゲームも世界的規模で増加している。これにより、数万人規模の多人数同時参加型ゲーム、MMOG(Massive Multiplayer Online Games) と呼ばれるジャンルも生まれた。

現在ある主な大規模ネットワークゲームの例として以下のようなものが挙げられます。

- 囲碁、将棋、麻雀
- ファイナルファンタジ XI
  スクウェア社が開発[1)]
  MMORPG
- エイジ・ オブ・エンパイア・シリーズ
  Microsoft[2)]
  リアル・タイム・ストラテジ
  (ストラテジー：戦略ゲーム系)
- ボンバーマンオンライン[3)]
  HUDSON SOFT[4)]/MGAME.Corp が開発
  アクション

これらのゲームは同時ログイン数は数万人でも同時にインタラクションするのは数人から数十人である。また、現在のネットワークゲームのシステムは、集中サーバーを用いて管理を行うのが一般的であるため、参加者が増えるに従って様々な障害が起こりやすくなっている。例えば、数万人が同時にアクセスすることにより集中サーバーへの負荷が大きくなり、重くなることや、集中サーバーが落ちるとゲームも全て止まってしまうことなどが挙げられる。また、開発されているゲーム自身も、特定のゲームマシンに依存して作られており、プレイヤーはソフトウェアが要求する、ハードウェアを購入しなければならない。

そこで、我々は集中サーバーが存在しないネットワーク構成で、数百万人規模のネットワークゲームを実現する事を最終的な目標として、Agent を用いたネットワークゲーム用フレームワークを提案している。

これまで、我々は PlayStation2-Agent 間の通信プロトコルに関する研究発表[5)]、また、Java での Agent-Agent 間プロトコルの提案に関する考察を行ってきた。[6)] 本論文では、実際に Java で実装した Agent-Agent 間プロトコルについて述べる。

2. Agent システムの概要

Agent とは、それ自身が情報処理能力を持ち、送受

† 琉球大学理工学研究科情報工学専攻
Interdisciplinary Infomation Engineering, Graduate School of Engineering and Science, University of the Ryukyus.
†† 琉球大学工学部情報工学科
Information Engineering, University of the Ryukyus.

信されるデータを元に家庭用ゲーム機とネットワークに対して、様々な働きかけを行うものである。Agent 間でネットワークを築くことにより並列分散型のネットワークゲームを構築する要素となる。(図 1 参照) Agent は、各家庭用ゲーム機の持つアーキテクチャの特性を吸収する。それにより、プラットフォームに依存しないゲームプログラミングを提供したり、ゲーム機と市場 PC との性能差を緩和できる等の利点がある。

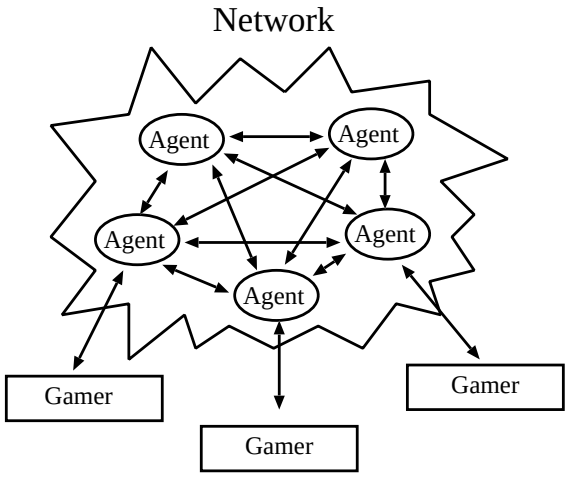

図 1 Agent を用いた通信システム

我々が目指している Agent を用いたネットワークゲームとは、まず、数百万人規模のネットワークゲームで、かつ、プレイヤーがゲームを常に監視する必要が無いゲームである。現存するネットワークゲームは、たいてい常にゲーム機となるマシンの前で、そのゲームを操作し進行状況を見ていなければならない。よって、かなりの時間が必要とされる。時間がない人には、そういったゲームを続けていくことは難しい。しかし、我々が目指すゲームは、ゲームの進行方法についての指示を設定しておけば、あとは Agent が指示通りにゲームを進めてくれるというものである。その間、ゲームを常に監視する必要はなく、時間が空いた時にだけゲームの様子を見ればよい。

また、分散 Agent によるネットワークゲームインフラを構築し、そのインフラ上にはゲームが複数個干渉しながら存在するものを目指している。

これらを実現するためには、現状のインフラには頼れず、新しいシステムが必要となった。そこで以下のようなインフラを提案している。

- 集中サーバーを必要としない。
  集中サーバーを用いたネットワークゲームの問題点は Section1 で既に述べた。数百万人規模のゲームを実現するには、分散 Agent によるネットワークゲームインフラの構築が必要となる。
- ゲームの必要に応じて動的かつ自律的にネットワークを再構築する。
  同じインフラ上に複数ゲームが存在する場合、各ゲームに最適なネットワークを構築していかなければならない。また、ネットワークトラフィックなどからトラフィックを抑えるために再構築をしていく必要がある。
- Agent ネットワークがプログラムの流通も担当。
  これにより、User はゲームをするために、必要なハードウェアやソフトウェアを準備する必要がない。

3. Agent のネットワーク構成

Agent が構成するネットワークは物理的なものと論理的なものの２種類ある。Agent 起動時に与えられるユニークなノード ID を基に Agent 同士がコネクションを確立し、作られる物理ネットワーク。もう一つは、物理ネットワーク上に仮想的に表現した論理ネットワークがある。(図 2 参照) 実際にゲームの通信は、論理ネットワークを基に行われる。

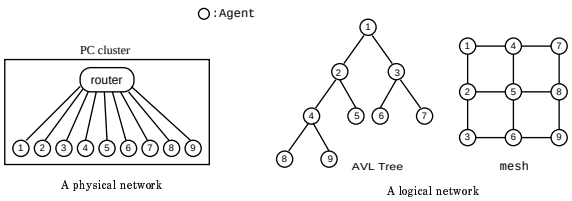

図 2 物理ネットワーク (左) と論理ネットワーク (右)

4. Agent が持つ 3 形態

Agent は通信する相手、また通信相手の要求に応じた 3 つの形態を持つ。
詳細を以下に示す。

4.1 Portal Agent

新しく接続要求を送って来た Agent の受け付け処理を行う。(図 3 参照) 要求に対して Portal Agent は、所持しているゲームリストを渡し、初めにどのゲームネットワークへ参加するのかを選択させ、実際にそのゲームを実行している Agent を検索して、そのアドレスを返す。アドレスを受け取った Agent は、その情報を基にゲームネットワークへ直接接続する。そして、ゲームネットワークの再構築を行う。この時、隣接する Agent からゲームネットワークの状態を表すデー

タ、ゲームリスト、ゲームに必要なデータを取得する

これにより、どの Agent からでもネットワークへ参加出来るので、集中サーバーを必要としない。Portal Agent では、1 つの Portal Agent に対して複数の接続要求が起こることを想定しており、よって 1 対多の通信となる。

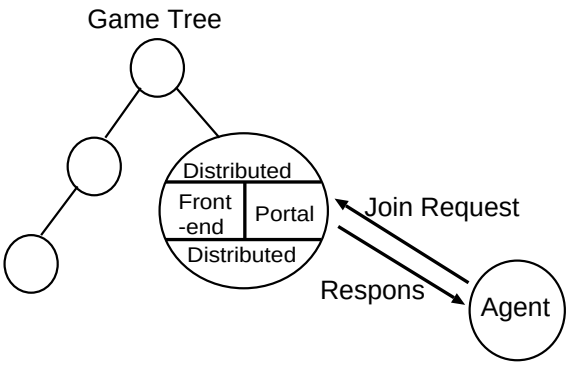

図 3 Portal Agent

4.2 Distributed Agent

ゲーム進行に必要なデータのやり取りを行う。他の Agent から受信したパケットを解析し、自分宛のデータなら適所に格納し、自分宛以外なら宛先によって適切な Agent に送り、ルーティングを行う。(図 4 参照) よって、多対多の通信となる。

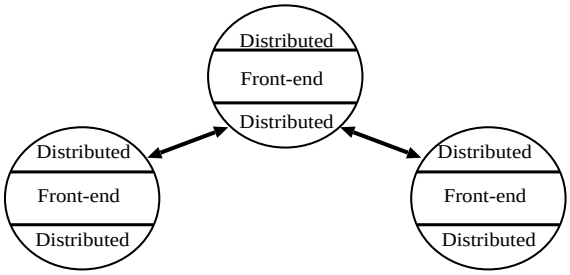

図 4 Distributed Agent

4.3 Front-end Agent

Agent とゲーム機のインターフェースとなる部分で、プレイヤーの操作をゲームの状態へ反映させたり、ゲームの状態に応じた描画抑制を行う。一つの Front-end Agent に対して、複数のゲーム機が接続可能である。(図 5 参照) よって、1 対多の通信となる。

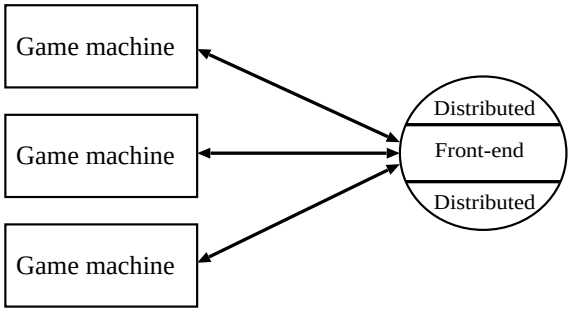

図 5 Front-end Agent

5. Agent の具体的な実装

Agent は Java で実装した。Java を用いることによって、以下のような利点が挙げられる。

- 様々なプラットフォームで動作する
- オブジェクト指向による開発
- オブジェクトシリアライズによるプログラム転送ができる

今回実装した Agent は投票ゲーム専用で、ネットワークトポロジーは AVL ツリー[7] を使用した。AVL ツリー 3 人の研究者 (Ade'son-Vel'skii-Landis) の名前が由来となっており、各ノードの高さがたかだか 1 という性質を持つ。また、ノードの挿入、探索、削除が全て O(log n) となっている。

実装した Agent の詳細を以下に述べる。

5.1 Node Manager class

接続要求を送ってきた Agent の受け付け処理を行う。Portal Agent の役割を果たす。今回は、Agent と切離し Node Manager として実装した。Node Manager はネットワーク構成も管理している。(図 6 参照)

5.2 Agent class

Distributed Agent、Front-end Agent の機能を実装している。Agent は、起動時に Node Manager に接続する。Node Manager は新規 Agent を追加したネットワークトポロジーを代表的な Agent(今回はツリー構造なので、ルート) に渡す。次に、代表的なノードは、自分と直接関係のある各ノードに新規 Agent が追加された事を通知する。通知された Agent は、通知してきた Agent 以外の直接関係のある Agent に通知する。このように新規 Agent が通知される過程で、最終的に新規 Agent と直接接続される Agent まで通知されると、その Agent が新規 Agent に ID を渡す。(図 7 参照)

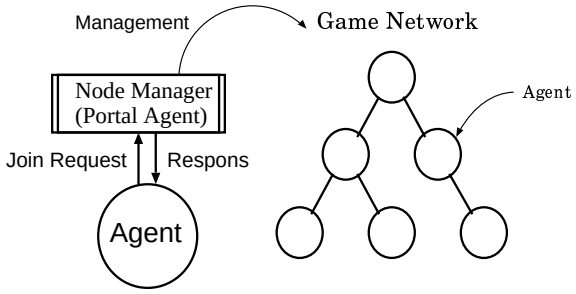

図 6 Node Manager class の構成

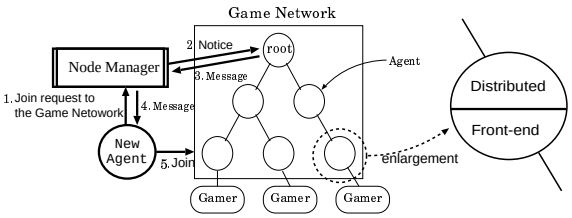

図 7 Agent class の構成

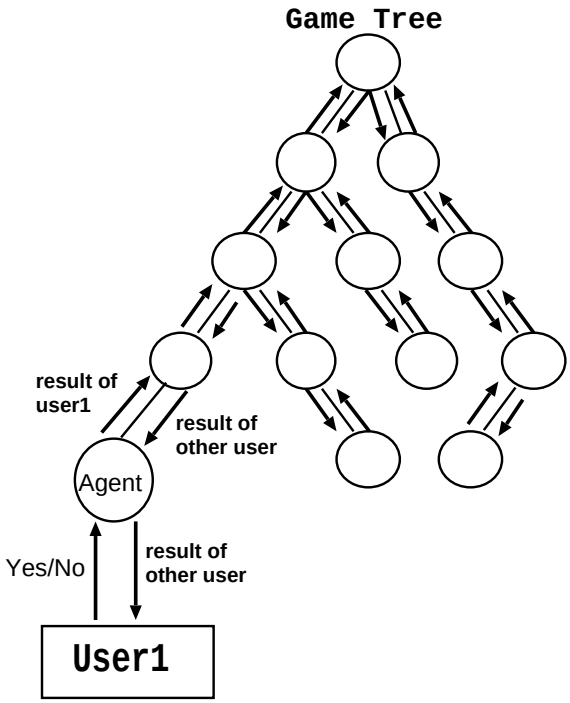

図 8 投票全体の流れ

6. 投票ゲーム

実装した Agent を用いて投票ゲーム用アプリケーションを作成した。投票ゲームとは、User がある質問に対して「Yes」か「No」を投票するゲームである。(図 8 参照) 投票した結果は、ツリーに沿って全ての Agent に反映される。

アプリケーションは起動時に最初に Node Manager に接続し、直ちに自分が接続する Agent の ID、IP アドレス、ポート番号を保持しているインスタンスを取得する。そして、その情報をもとに Agent に接続する。接続された Agent は保存しているアプリケーションのデータを直ちにアプリケーションに送信する。アプリケーションは受け取ったデータを処理し、ゲームがスタートする。スタート後のデータのやり取りは、全て最初に接続した Agent を介して行われる。

7. シミュレーションによる通信量計測

PC クラスタ 40 台を用いてポート毎の平均通信量 (byte/sec)、全体の平均通信量 (byte/sec) を計測するためにシミュレーションを行った。シミュレーションを行うにあたって、質問を投入するエミュレータと投票をするエミュレータを作成した。

学科のマシン 40 台を用いて、Agent50 台、質問 20 問、User200 人でポート毎の平均通信量を計測、また、Agent50 台、質問数 20 問は固定、User 数を 10 人から 80 人まで 10 人ずつ増やしていき、それぞれの全体の平均通信量を計測を行った。

シミュレーション時の投票データの流れ (図 9 参照) を以下に述べる。

1 投票の対象となる問題が、エミュレータから投入される
2 データを受け取った Agent は問題をハッシュテーブルに格納する
3 同時に、Agent は直接接続している Agent にデータをマルチキャストする。最終的に全ての Agent に反映される
4 投票エミュレータは Agent が保持している問題をとってきて、質問に対して「Yes」か「No」を選択し、Agent に投票データを送信する
5 投票データを受け取った Agent は、またそれをハッシュテーブルに格納する
6 3 と同様にマルチキャストし、Agent に反映させる

Agent が保持しているハッシュテーブルには、質問、Yes の総数、No の総数が格納されている。また、User は質問一つに対して、投票は一回だけ行う。

7.1 計測目的

特定のポートに集中していないかを調べるために、ポート毎の平均通信量の計測を行った。ばらつきが少ないほどネットワーク構成が良く、スケーラビリティ

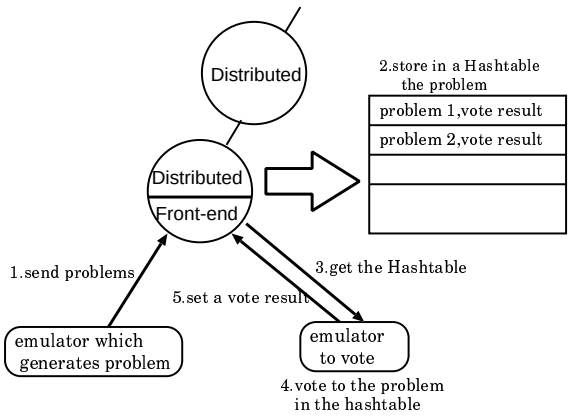

図 9 投票データの流れ

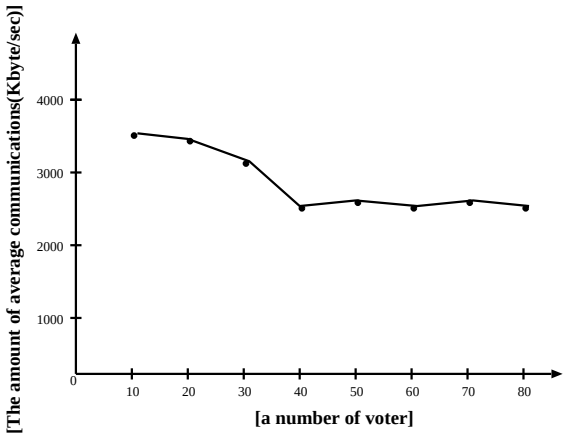

図 11 投票者数に対する全体の平均通信量

があるといえる。また、User 数を徐々に増やしていった時の各々の全体の平均通信量を計測することによって、その増減から人数的スケーラビリティがあるかどうかを調べた。

7.2 計測結果

シミュレーションからポート毎の平均通信量 (図 10 参照)、全体の平均通信量の計測 (図 11 参照) を行った。

図 10 は、横軸が各々のポートの平均通信量 (`byte/sec`) を表しており、縦軸がポートの個数を表している。

図 11 は、横軸が投票者数を表しており、縦軸が全体の平均通信量 (`Kbyte/sec`) を表している。

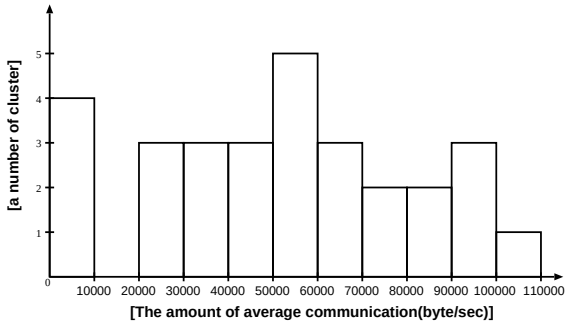

図 10 ポート毎の平均通信量

7.3 比較

7.3.1 Globus

Globus[8] とは、Globus Toolkit といわれる基本のグリッドサービスをまとめて開発されたものである。Globus Toolkit は、グリッドコンピューティングの中核をなすソフトウェアの 1 つで、オープンソースのミドルウェアである。

Globus Toolkit は大きく分けて資源管理を行うサービスである GRAM、グリッドマシンの情報を提供する MDS、高性能で安全にデータを転送するプロトコルである GridFTP と呼ばれる 3 つの柱と、それらのサービスを支えるセキュリティ機構である GSI と呼ばれるもので構成されている。

7.3.2 JXTA

JXTA とは、一般的な P2P アプリケーションを構築するためのフレームワークを提供する。JXTA プロトコルは、PDP(Peer Discovery Protocol)、PRP(Peer Resolver Protocol)、RVP(Rendezvous)、PIP(Peer Infomation Protocol)、PBP(Pipe Binding Protocol)、ERP(Endpoint Routing Protocol) の 6 つのプロトコルから構成されている。これらのプロトコルは TCP/IP などのトランスポートプロトコルの上に実装されることができる。

7.4 考察

図 10 では、平均通信量が`10,000byte/sec` 以下のポートもあれば、`100,000byte/sec` 以上のポートもあり、最大通信量の最小通信量の差が約 10 倍になっていることが読み取れる。これは、通信量が多い Agent と少ない Agent が存在していることを意味しており、負荷が分散されていないと言える。我々が目指している Agent システムは、全ての Agent になるべく均等に負荷を分散させることを目的としているので、今回実装した Agent では、ネットワーク部分を再構成する必要があると考えられる。

図 11 では、投票者数を 30 人から 40 人に増やしたところでの全体の平均通信量の低下が顕著に現れており、40 人以上からの通信量は一定を保っている。これは、やりとりされるデータ量が増えたことにより、

Agent の処理が低下し、それにより全体の平均通信量も減少したと考えられる。そして、40 人以上からは、データ量が増えても処理速度が一定になっているので、全体の平均通信量も一定になってしまっていると考えられる。

今回実装した Agent は、プログラムの設計不足、特に Node Manager がネックとなった。現段階では、Node Manager は Agent の数だけソケットを開く仕様になっており、Agent の数がソケットの限界数に依存している。また、Agent は起動時に Node Manager に接続するので、タイミングによっては Node Manager に一気に負荷がかかることも考えられ、それによって、Node Manager が機能しなくなることも考えられる。

スケーラビリティを上げるには、まずこれらのネックの解消が必須である。

8. まとめと今後の課題

8.1 ま と め

本研究では、現在のネットワークゲームにおける問題点を挙げ、その解決方法として Agent による、並列分散型のネットワーク構築システムを提案した。また、それらを用い、複数台の PC 上でシミュレーションを行い、ポート毎の平均通信量、投票者数に対する全体の平均通信量の計測を行った。計測により、スケーラビリティに問題があることが明確となった。

現在、実装途中である Agent システムは、まだ実装されていない部分がたくさんあり、まだ細かく決めなければならない仕様やデータフォーマット等がある。今後に残された課題を示す。

8.2 今後の課題

8.2.1 NodeManager のネック解消

第一に考察で述べた、Node Manager のネックの解消をしなければならない。その上で、Node Manager への集中的負荷が起こることを防ぐために、今回実装した Node Manager を Agent に組み込む。Agent 全てに Node Manager の機能を実装することにより、Node Manager にかかる負荷を分散させる。

8.2.2 Agent 間の更に詳しい通信プロトコル

現段階の Agent システムでは、まだ Agent 間の通信について、定まっていない所が多い。例えば、ネットワークトポロジーの表現方法やネットワーク構築の手順など、仕様を作成して実装しなければならない。

8.2.3 動的なゲームの実行

Agent システムは本来、ゲームに必要なデータを全て Agent から支給する。ゲームマシン側で動くゲームプログラム自体も、それに含まれる。ゲームプログラム自体をダウンロードして、実行するゲームプログラムの実装が必要である。この際、Java のオブジェクトシリアライズを、ゲームマシン上で利用出来ることが望ましい。

Java がサポートされていない、もしくは、できないゲームマシンの場合は、Agent 側でゲームマシンのアーキテクチャに合うプログラムを生成する必要がある。これを実現するには、高レベルのゲーム記述言語が必要である。

参 考 文 献

1) http://www.square-enix.co.jp/
SQUARE ENIX 社
2) http://www.microsoft.com/
Microsoft 社
3) http://www.bomberman-online.com/
ボンバーマンオンライン公式サイト
4) http://www.hudson.co.jp/index.cgi/
HUDSON 公式サイト
5) 佐渡山 陽, 河野真治. PlayStation 2Linux 上のネットワークゲーム・フレームワークの提案. 日本ソフトウェア科学会第 19 回大会論文集 September, 2002
6) 佐渡山 陽, 小杉 隆二, 河野 真治. 大規模ネットワークゲームのインフラを自律的に構築するシステムに関する考察. 日本ソフトウェア科学会第 19 回大会論文集 September, 2003
7) http://ciips.ee.uwa.edu.au/morris/Year2/PLDS210/AVL.html
AVL Trees
8) http://www.globus.org/
the globus alliance
9) Robert Lafore(著), 岩谷 宏 (訳). Java で学ぶアルゴリズムとデータ構造. ISBN 4-7973-0694-7
10) Michael J. Donahoo/Kenneth L. Calvert(共著), 小高 知宏 (訳). TCP/IP ソケットプログラミング (Java 編). ISBN 4-274-06520-0
11) 小高 知宏 (著). 基礎からわかる TCP/IP Java ネットワークプログラミング. ISBN 4-274-06486-7