負荷分散型の大規模多人数参加型サービスにおける不正攻撃対策

遠藤慶一[†] 川原稔[††] 高橋豊[†]

本論文では，インターネットを通じてリアルタイムで双方向に情報を送受信する大規模多人数参加型サービスにおいて，ユーザが利用するマシンにサーバ機能の一部を委譲することによって，サーバ負荷を分散させる構成法を提案する．また，悪意のあるユーザによって行われる可能性のある不正攻撃を列挙し，提案法がそれらに対してどの程度の耐性があるかを評価する．サービスの信頼性を確保するための基本的なアイデアは，同じデータを複数のユーザマシンで管理して多数決をとるというものである．この仕組みによって，ユーザマシンに障害が起こった場合や悪意のあるユーザによるデータ改竄などの攻撃を受けた場合にも正常なサービス提供を続けることができる．

Cheat Prevention for Massively Multiplayer Online Distributed Services

Keiichi Endo,[†] Minoru Kawahara[††] and Yutaka Takahashi[†]

This paper deals with Massively Multiplayer Online Services, users of which communicate interactively with one another in real time. We propose an architecture for distributing loads by transferring part of the server's function to users' machines. We also enumerate possible attacks by malicious users, and evaluate the robustness against them. The main idea to solve security problems is "letting multiple machines manage the same data and applying a decision by majority rule." This mechanism adds robustness to the service against halts of users' machines and cheating such as data tampering by malicious users.

1. はじめに

近年，オンラインゲーム，オンラインチャット，オンラインオークション，オンライントレードなど，ユーザがリアルタイムで情報を送受信するサービスの利用者が増えている．本論文では，この種のサービスのことを MMO（Massively Multiplayer Online; 大規模多人数参加型）サービスと呼ぶことにする．

MMO サービスは多数のユーザが同時に利用するが，ユーザはつねに限られた範囲しか見ることができない．本論文ではこの範囲のことをサイトと呼ぶ．すなわち，MMO サービスにはいくつかのサイトがあり，ユーザはそれぞれ 1 つのサイトに属している．ユーザはあるサイトから別のサイトへ移動することが可能である．ユーザのアクションは，同じサイトにいるユーザのみにリアルタイムで伝えられる．各ユーザおよび各サイトはそれぞれ状態を持ち，それらは状態データによって表現される．たとえばロールプレイングゲームサービスの場合は，ユーザの位置，マップ（サイト）に置かれたアイテムの情報が，ユーザの状態データ，サイトの状態データにそれぞれ相当する．状態データは，ユーザがアクションを起こすたびに更新される．

現在，MMO サービスはほとんどの場合，サービス提供者が管理する中央サーバにユーザが直接接続するクライアント・サーバ（C/S）モデルによって運営されている（**図 1**）．しかしこのモデルには，計算負荷や通信負荷が中央サーバに集中するという欠点がある．このモデルで MMO サービスを運営するためには，高性能なサーバマシンや高速な通信回線が必要となる．そのため，Peer-to-Peer（P2P）モデルによってこのサービスを運営する手法に関する研究が近年活発に行われている．**図 2** は本論文で提案するモデルを示したものである．P2P モデルでは，ユーザの CPU 処理能力，メモリ，通信帯域をサービス運営に利用するので，サービス提供者が管理するサーバマシンにかかる負荷は軽減される．しかし，実用のためには，ユーザマシンの突然の離脱に対するロバスト性を確保し，ユーザの不正行為を防止することが必要である．P2P モデルではサーバ機能の一部をユーザマシンに委譲するため，一般にデータ改竄などの不正を行いやすい．本論文では，複数のユーザマシンに同じデータを管理させて，多数決方式を適用することにより，上述の問題の

† 京都大学大学院情報学研究科
Graduate School of Informatics, Kyoto University
†† 愛媛大学総合情報メディアセンター
Center for Information Technology, Ehime University

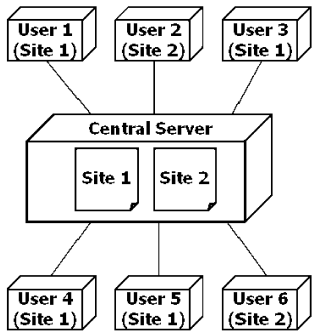

図 1 クライアント・サーバモデル
Fig. 1 Client-Server model.

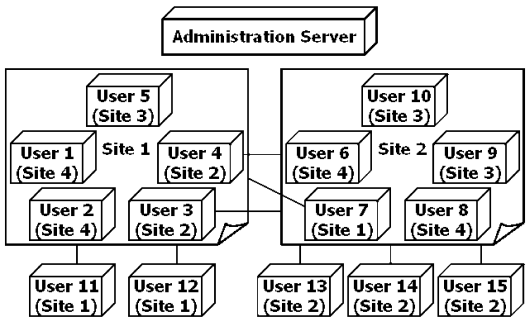

図 2 提案する Peer-to-Peer モデル
Fig. 2 Proposed Peer-to-Peer model.

解決を試みる．

以下，2 章では関連研究を紹介し，3 章で本論文で提案するモデルについて説明する．そして 4 章で提案法の不正攻撃への耐性を評価し，5 章で性能分析を行う．最後に 6 章でまとめと今後の課題を述べる．

2. 関 連 研 究

現在までに提案された，P2P モデルで MMO サービスを実現する構成法は，以下の 2 種類に大別される．

ユーザ中心構成法： 各ユーザがアクションを起こしたユーザからアクションメッセージを直接受け取って，自身が保持している状態データを更新する．

サーバ中心構成法： ある状態データの書き換えを 1 人のユーザが制御する．そのユーザはアクションメッセージを受け取って状態データを更新し，更新情報を必要とするユーザにアップデートメッセージを送信する役割を果たす（これらの通信は他ノードを経由して行われる場合もある）．

サーバ中心構成法では，ある状態データを書き換える権限を 1 人のユーザのみが持つことによって一貫性を確保しているが，そのユーザがデータ改竄などの不正を行うと，サービスに悪影響を及ぼす可能性がある．サーバ中心構成法にはこのような問題があるため，ユーザ中心構成法に分類される研究がより多く行われてきた．しかし，ユーザ中心構成法にも以下のような好ましくない点がある：

- 各ユーザから別々に直接メッセージを受け取るため，サーバ中心構成法に比べて通信負荷が高い．
- 各ユーザが，アクションメッセージを受信して状態データを更新するための計算に必要な状態データをすべて知っていなければならないので，情報の秘匿が難しい．

以下では，これまでに行われてきた研究をあげる．

2.1 ユーザ中心構成法

Bucket Synchronization [1] は，P2P 型の多人数参加型ゲームにおいて，パケット転送に遅延が発生してもユーザ間で状態の一貫性を保つことを可能にする分散同期メカニズムである．このメカニズムでは，時間を等間隔に区切り，それぞれの区間に bucket というものを割り当てる．ユーザが何らかの行動をとったときは，他のユーザにアクションメッセージを送信する．ユーザがアクションメッセージを受信したときは，それが送信された時間区間に対応する bucket に入れておく．bucket 中のアクションは，その bucket が割り当てられている時間区間の終点から一定時間経過してからそれぞれのユーザマシン上で実行される．このメカニズムを利用したアプリケーションの例としては，Age of Empires というゲームがある．しかしこのメカニズムには，様々な不正が可能であるという問題がある．たとえば，他のユーザのアクションを知ってから，同じ時間区間内に自分のアクションを決定して送信することにより優位に立つということができてしまう（これを先読みチートと呼ぶ）．

Lockstep Protocol [2] では，アクションの中身を送信する前にハッシュを送信するようにすることによってこの問題を解決している．しかしこのプロトコルでは，アクション遅延（ユーザがアクションを起こしてから，その結果がユーザの画面に表示されるまでにかかる時間）が最も長いユーザ間パケット転送遅延の 3 倍以上にもなってしまう．アクション遅延の平均を短くするために Asynchronous Synchronization というものも提案されているが，この方法ではジッタ（アクション遅延のばらつき）が大きくなる．この性質は，特にゲームアプリケーションでは好ましくない．

Adaptive Pipeline Protocol [3] では，ネットワークの状態に応じてパラメータを調整することによりジッタを抑える．しかし，アクション遅延はまだかなり大きい．

文献 4) では，ユーザ間で状態データの不整合が起こった場合にロールバックを行うことによって，アクション遅延を短くする手法が提案されている．しかし，一時的に誤った情報が表示されることは多くのアプリケーションにとって好ましいことではない．アクション遅延を犠牲にしてロールバックの発生頻度を下げることも可能だが[5]，ネットワークの混雑が起きやすいインターネットでは，発生頻度を十分低くしようとするとアクション遅延が長くなりすぎてしまう．

NEO Protocol[6] では，過半数のユーザがある時間区間内に受信できたアクションメッセージのみをそれぞれのユーザが受け入れるというルールにすることによって，ロールバックの必要なしに短いアクション遅延を達成している．このプロトコルは固定トポロジにおいては高性能で不正にも強いシステムを実現しているが，状態データの保存やユーザの他グループ（本論文でサイトと呼んでいるもの）への移動を，不正攻撃に影響されない方法でどのように行うのかは述べられていない．さらに，低速回線を利用しているユーザはサービスを利用することができない．

2.2 サーバ中心構成法

Mercury[7] は，ゲームアプリケーション向けの通信プロトコルであり，publish-subscribe システムを分散的に実現している．データを送信するとそれがリレーされ，そのデータを必要とするユーザ（購読者）の情報を管理しているユーザのもとに届いたとき，それが配信される仕組みになっている．しかしこのプロトコルのルーティング・アルゴリズムでは，平均 $O(n)$ ホップ（n はユーザ数）必要となるため，実用的ではない．

文献 8) および 9) は，分散ハッシュテーブルの利用により必要ホップ数を $O(\log n)$ に減らしている．さらに，文献 8) では，状態データを複製することによってデータが失われる確率を下げる方法も述べられている．文献 9) では，購読者情報を管理しているユーザを記憶し，次に同じ種類のデータを送るときはそのユーザに直接送信することによって，データの伝達にかかる時間を短縮する手法が提案されている．

3. 提案する P2P モデル

本論文で提案する P2P モデルは，サーバ中心構成法に近いが，複数のユーザが同じ状態データを管理するという点が既存のものと異なる．以下ではサイトを管理しているユーザマシンのことをサイトサーバと呼ぶことにする．

図 3 は，サイトサーバがどのように 1 つのサイトを管理するかを示したものである．ユーザは，自身がいるサイトを管理するすべてのサイトサーバと接続している．同じサイトを管理しているサイトサーバは，フルメッシュ型のネットワークを形成している．ユーザがアクションを起こしたときは，すべてのサイトサーバにアクションメッセージを送信する．アクションメッセージを受け取ったサイトサーバは状態データを更新する．その際には同期をとって，他のサイトサーバと同じデータを保持するようにする．最後に，サイトサーバはそのサイトにいるユーザに状態データの更新を伝える．

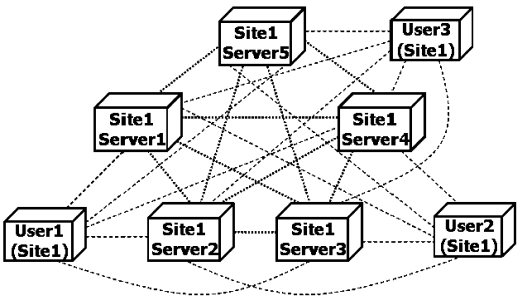

図 3 1 つのサイトを管理するモデル
Fig. 3 Model of managing a site.

このモデルでは，2 章であげた両構成法の問題点を解決している：

- 複数のユーザが同じ状態データを管理しているので，そのうちの過半数がデータ改竄などの不正を行わない限り，サービスに悪影響が及ぶことはない．
- サイトサーバが，複数のアクションの処理結果をまとめて送信できるので，ユーザ中心構成法に比べて通信効率が良い．
- あるサイトに関連する状態データを，そのサイトにいるユーザから隠すことができる．

以下では，アルゴリズムをより詳しく説明する．

3.1 ユーザのログイン

MMO サービスを利用するユーザは，まず初めに管理サーバに接続し，ユーザ ID とパスワードを送信して認証を受ける．

3.2 サイトのサービス機能の委譲

同時接続ユーザ数が少ないとき，MMO サービスは通常の C/S モデルで運営される．すなわち，すべてのサイトのサービス機能は管理サーバの中にある．ユーザ数が増えるにしたがって，管理サーバにかかる負荷が高くなっていく．そこで，ある条件でサイトのサービス機能はユーザのマシンに委譲される．ある条件とは，たとえば「サイトにいるユーザの人数が規定値 $\theta_{\max}$ に達したとき」などである．この仕組みによっ

て，ユーザマシンは徐々に（ハイブリッド）P2P ネットワークを形成する．

管理サーバがサイトのサービス機能を委譲することを決めたときは，サービス機能の委譲先として N_{server} 個のユーザマシンを選ぶ．ここで N_{server} はあらかじめ定められた奇数である．移譲先の選び方としては様々なものが考えられるが，単純な方法としてはまだサイトを管理していないユーザマシンの中からランダムに選ぶというものがある．ただし不正行為を難しくするため，サービス機能を委譲しようとしているサイトにいるユーザは選ばないようにする．

移譲先のユーザマシンを決定した後，管理サーバはそのサイトにいるすべてのユーザに対し，新しいサイトサーバの情報を送る．ユーザがその情報を受け取ったときは，それをもとにサイトサーバに接続する．そして，それまでに送信したアクションメッセージ（詳細は次節参照）のうち，対応するアップデートメッセージが返ってきていないものがあれば，新しいサイトサーバに再送する．

3.3 アクションメッセージの処理

ユーザによるアクションは，以下のような流れで処理される．

Step 1 ユーザが，すべてのサイトサーバにアクションメッセージを送信する．

Step 2 サイトサーバが，タイムスロットメッセージを相互に送受信する．

Step 3 すべてのサイトサーバが，各ユーザにアップデートメッセージを送信する．

なお，それぞれのアクションメッセージには識別のためにアクション番号が付加されている．また，サイトサーバの時計は NTP [10]（Network Time Protocol）のような仕組みによってあらかじめ同期がとられているものとする．

Step 2 および Step 3 は，ある時間間隔 T_{timeslot} ごとに実行される．Step 2 におけるタイムスロットメッセージとは，アクションメッセージの受信時刻を他のサイトサーバに知らせるために送るものであり，以下の情報から構成される：

- そのタイムスロットメッセージの送信時刻（タイムスタンプ）．
- 前回タイムスロットメッセージを送信した後に受信したすべてのアクションメッセージに関する {ユーザ識別番号，アクション番号，到着時刻} の組．

本論文のシステムでは，それぞれのサイトサーバへの到着時刻の中央値を基準としてアクションメッセージの処理順序を定めることにより，サイトサーバ間での状態データの一貫性を保つ．中央値を基準とする理由は，特定ユーザが不当に有利あるいは不利になる状況を作り出すような受信時刻改竄などの，不正行為による影響を抑えるためである．N_{majority} 個未満のサイトサーバが受信時刻を偽っても，中央値は必ず正規の受信時刻となる．

具体的には，以下に示すアルゴリズムで状態データを更新する．

(1) N_{majority} 個以上のサイトサーバ（自分自身を含む）から情報を受け取っているアクションの中から，N_{majority} 番目に早い到着時刻が，どのサイトサーバからの最新タイムスロットメッセージのタイムスタンプよりも早いアクションを選ぶ（複数の場合もあり）．ただし N_{majority} は，同じサイトを管理するサイトサーバの数の半分より大きい最小の整数，すなわち
$$N_{\mathrm{majority}} = (N_{\mathrm{server}} + 1)/2$$
である．このステップは，到着時刻の中央値が確定したアクションを選ぶことを意味する．もしそのようなアクションがなければ，状態データの更新は行われない．

(2) サイトサーバは N_{majority} 番目に早い到着時刻をもとにした順序で，選ばれたアクションに対応するアクションメッセージを処理することにより，状態データを更新する．もしまだ受信していないアクションメッセージがあれば，そのアクションを起こしたユーザからそのメッセージが到着するまで待つ．

(3) ユーザにアップデートメッセージを送信する．

アクションの順番付けの具体例を**表 1** に示す．この表において括弧付きの箇所は，該当アクションメッセージの到着時刻が該当サイトサーバからまだ知らされていないところである．ただし，その到着時刻は，そのサイトサーバから最後に来たタイムスロットメッセージの送信時刻よりは後であるはずなので，「(> 最後に来たタイムスロットメッセージの送信時刻)」という形で表してある．この例の時点で N_{majority}（= 3）番目に早い到着時刻，すなわち順番付けの基準となる時刻が確定しているアクションは，Action A および Action C であり，基準時刻はそれぞれ 7.8，7.3 である．したがって，この場合は Action C，Action A の順に処理を行う．Action B については，基準時刻が表 1 で網がけされている 2 つのどちらにもなりうるので，この時点では処理を行わない．

サイトサーバはアクションを処理して状態データを

表 1 アクションの順番付けの例
Table 1 Example of ordering actions.

	Server1	Server2	Server3	Server4	Server5
Timestamp of the Last Timeslot Mes.	9.0	8.0	9.0	8.0	8.0
Arrival Time of Action Mes. A	7.6	(>*8.0*)	8.2	**7.8**	7.2
Arrival Time of Action Mes. B	8.3	(>*8.0*)	(>*9.0*)	7.4	7.9
Arrival Time of Action Mes. C	**7.3**	7.2	6.8	(>*8.0*)	(>*8.0*)

更新した後，アップデートメッセージによって各ユーザに状態データの更新を伝える（Step 3）．そして，ユーザは N_{majority} 個の同じアップデートメッセージを異なるサイトサーバから受け取った後に，アップデートメッセージの内容を受け入れ，画面を更新する．この仕組みにより，半数未満のサイトサーバが不正を行っていても影響を受けない．なお，いくつかのサイトサーバはアップデートメッセージのハッシュのみを送るようにすることによって通信量を削減することもできる．たとえば N_{majority} 個のサイトサーバがアップデートメッセージを，$(N_{\mathrm{majority}}-1)$ 個のサイトサーバがそのハッシュを送信するようにした場合は，それらのうち N_{majority} 番目が届くまでの間に必ず 1 個はメッセージ本体が届くため，全サイトサーバがメッセージ本体を送る場合と比べて遅延が長くなるということはない．

ユーザが各サイトサーバから受け取ったアップデートメッセージが一致していない場合は，管理サーバに通報する．通報を受けた管理サーバは，サイトサーバや該当サイトにいるユーザに対して送受信メッセージのログを要求し，悪意のあるユーザを正確に特定する．ログの改竄を防ぐため，ユーザが送信するメッセージにはデジタル署名が付加されているものとする（メッセージには通し番号の情報を含んでいるものとする）．管理サーバがサイトの管理機能を委譲した時点の状態データと，ユーザが送信したアクションメッセージのログなどを利用して，管理サーバ内でアクションメッセージの処理などを正しくシミュレートすれば，その結果と異なるメッセージを送信したサイトサーバが不正を行っていると判断することができる．このようにすれば，たとえ正常なサイトサーバが少数派であったとしても，悪意のあるユーザを正しく特定し，ロールバックによって正常な状態に戻すことができる．なお，不正を行おうとしたユーザは，発見次第何らかの方法でサービスの利用を禁止することが望ましい．

3.4 ネットワーク混雑の対策

前節で述べたアルゴリズムだけでは，アクション遅延がネットワークの混雑に大きく影響されてしまう．なぜならば，たとえばあるサイトサーバから N_{majority} 個以上のサイトサーバへのタイムスロットメッセージが遅れると，それらのサイトサーバ上ではアクションの順番付けが行えず，アクションの処理も行えないため，そのサイトのサービス機能が一時停止するからである．以下に述べる方法でこの問題を解決する．

T_{timeout} を T_{timeslot} より大きい定数とする．もしあるサイトサーバが，他のサイトサーバからのタイムスロットメッセージを，最後に来たタイムスロットメッセージのタイムスタンプより T_{timeout} だけ後の時刻になっても受け取っていないときは，同じサイトを管理するサイトサーバのうち，届いていないタイムスロットメッセージの送信元サーバを除いた全サーバにタイムスロット・リクエストメッセージ（TRM）を送信する．TRM を受け取ったサイトサーバは，要求されているタイムスロットメッセージを以前に受け取っていれば，要求されているメッセージを返送する．

すべての TRM 送信先サイトサーバから TRM を受信したとき，または TRM を送信してからある時間 $T_{\mathrm{req\_timeout}}$ が経過したときは，届いていないタイムスロットメッセージの送信元サイトサーバをある時間 T_{ignore} だけ無視する．サイトサーバを無視するとは，そのサーバからのタイムスロットメッセージを無視し，状態データ更新時にもそのサーバを除外してユーザアクションの順番付けを行うという意味である．このアルゴリズムによって，あるサイトサーバから無視されたサイトサーバは，同じサイトを管理する他のすべてのサイトサーバにも無視されることになる．時間 T_{ignore} が経過する前に，無視されているサイトサーバからメッセージが届いた場合は，そのサイトサーバがサイト管理に復帰するが，届かない場合は，管理サーバに代役を選ぶよう要求する．この要求の正当性を確認するため，管理サーバは同じサイトを管理する N_{majority} 個以上のサイトサーバから同じ要求が来た後に代役を選び，サイトの管理機能を委譲する．その際には，元の（無視されていた）サイトサーバにも代役が立てられたことを通知しておく．もし代役が選ばれた後に，元のサイトサーバから他のサイトサーバに

メッセージが届いても，サイト管理には復帰させない．これらの仕組みによって，サービスはユーザマシンの停止に対するロバスト性を持つ．

3.5 サイトの移動

ユーザがサイトを移動する場合は，管理サーバから移動先サイトを管理しているサイトサーバの情報を取得する．また，管理サーバは移動元サイトを管理しているすべてのサイトサーバからそのユーザの状態データを受け取り，それらの比較により正当性を確認した後，移動先サイトを管理しているサイトサーバに引き渡す．このとき，移動元サイトを管理しているサイトサーバのうちいくつかに対しては，ユーザの状態データの代わりにそのハッシュを要求するようにすれば，通信量を削減することができる．

すでに述べたように，サイトがそのサイトにいるユーザによって管理されることは望ましくない．そこで，もしユーザがそのユーザによって管理されているサイトに移動するときは，管理サーバが代替のサイトサーバを選び，新しいサイトサーバにそのサイトに関するすべてのデータを送信するようサイト移動するユーザに対して命じる．管理サーバは他のサイトサーバやそのサイトにいる全ユーザに対しても，サイトサーバが変わったことを知らせる．

3.6 ユーザのログアウト

ユーザが MMO サービスの利用を終了するときは，管理サーバにその意向を伝える．その際，そのユーザがサイトのサービス機能を持っている場合は，そのサイトに関するすべての状態データを管理サーバに送信する．そしてユーザは全サーバとの接続を切断し，サービスのソフトウェアを終了させる．

管理サーバはさらに次のような処理を行う必要がある．

(1) ログアウトしたユーザがサイトを管理していた場合：
  (a) そのサイトを管理している他のサイトサーバから状態データのハッシュを受け取り，ログアウトしたユーザが持っていた状態データから計算したハッシュと比較することにより，状態データの正当性を確認する．
  (b) サイトサーバとなるユーザマシンを新たに選び，上記の状態データを送信する．もし総ユーザ数が少なすぎて代替のサイトサーバを見つけることができない場合は，管理サーバがそのサイトの管理を始める．
  (c) そのサイトの他のサイトサーバや，そのサイトにいる全ユーザにサービス機能の移動を伝える．
(2) ログアウトしたユーザがサイトサーバによって管理されているサイトにいた場合：
  (a) ログアウトしたユーザがいたサイトにいるユーザの数が規定値 $\theta_{\min}$ 以下になった場合は，そのサイトを管理しているすべてのサイトサーバから，そのサイトに関するすべての状態データを受け取り（ハッシュの利用により通信量削減可能），それらの比較によって正当性を確認した後，管理サーバがそのサイトの管理を始める．
  (b) 上記 (a) 以外の場合は，ログアウトしたユーザの状態データのみを受け取り，正当性を確認する．
(3) ログアウトしたユーザの状態データを管理サーバの記憶装置に保存する．

4. 不正攻撃への耐性

本章では，悪意のあるユーザによって行われる可能性のある不正攻撃を列挙し，提案法がそれらに対してどの程度の耐性があるかを述べる．不正攻撃には主に以下のようなものがある．

不正接続 ユーザ ID を偽ってサイトサーバに接続する．

不正送信 不正なデータを送信する．

送信抑制 送信すべきデータを送信しない．

DoS 攻撃 パケットを大量に送信してサービス不能に陥らせる．

データ盗聴 サイトサーバとなっているユーザがメモリ中の状態データを覗く．

このうち DoS（Denial of Service）攻撃（分散 DoS 攻撃を含む）については，現在までも文献 11)〜15) のような研究が行われており，今後も活発な研究が行われると期待される．したがって，本論文では DoS 攻撃については考察の対象外とする．

また，本研究のシステムにおいて，ユーザは自身がいるサイトをサイトサーバとして管理することはできないようになっているが，自身が管理しているサイトに移動して（この時点でサイトの管理機能は他のユーザマシンに移る），過去に持っていた情報（状態データ）を利用することにより優位に立つことは可能である．この問題の解決法としては，検証可能秘密分散法や，マルチパーティ・コンピューテーションなどのような技術を利用することにより，過半数のサイトサー

表 2 不正攻撃への耐性
Table 2 Robustness against cheating.

	Step 1 (Action)	Step 2 (Timeslot)	Step 3 (Update)
不正接続	A	A	A
不正送信	B	C	B
送信抑制	A	C	C

（上表の A，B，C は下表の耐性ランクを表す）

耐性ランク	$1 \leq x < N_{\text{majority}}$	$N_{\text{majority}} \leq x < N_{\text{server}}$	$x = N_{\text{server}}$
A	問題なし	問題なし	問題なし
B	影響なし（検出可）	ロールバックで復旧可	影響あり
C	影響なし（検出不可）	影響あり	影響あり

（x は同じサイトを管理する不正サイトサーバの数）

バが結託しない限り状態データの内容を知ることができないようにする方法が考えられる[16]．しかし，本論文ではこのようなデータ盗聴に関する問題を扱わず，今後の課題としておくことにする．

以下では，上述の 2 種類を除いた不正接続，不正送信，送信抑制について考察する．**表 2** は，3.3 節の冒頭で示したそれぞれのステップで不正が行われた場合，サービスに影響が及ぶ可能性があるかどうかを示したものである．

4.1 不正接続

不正接続を防ぐためには，サイトサーバに接続しようとしているユーザの認証を行えばよい．サイトサーバはユーザのパスワードを知らない（知ってはならない）ので，次のようにして，管理サーバにより認証済みのユーザであることを確認する．まず，ユーザのサイト移動などで新たなユーザ間の接続が必要になるたびに，管理サーバが十分な長さを持つランダムな数（以下ではセッション ID と呼ぶ）を生成し，それを接続元および接続先のユーザに知らせる．そして，接続元ユーザはセッション ID を接続先ユーザに送信し，接続先ユーザはそのセッション ID が管理サーバから受け取ったものと一致する場合のみ接続を許可する．なお，セッション ID などの機密情報を送受信する際には，適宜 SSL などの暗号通信方式を利用するものとする．

4.2 不正送信

3.3 節の冒頭で示したそれぞれのステップで不正送信が行われた場合について考察する．

Step 1 不正なアクションメッセージをユーザが送信した場合，正常なサイトサーバは不正を検出できるので，過半数のサイトサーバが正常ならばそのアクションが実行されることはない．また，各サイトサーバに異なるアクションメッセージが送信された場合にも整合性が保たれるようにするには，タイムスロットメッセージにアクションメッセージのハッシュ値を含めればよい．

Step 2 不正なタイムスロットメッセージを送信するサイトサーバが半数未満の場合は，アクションメッセージの受信時刻がどのように改竄されていても，中央値は適正な時刻となる．各サイトサーバに異なるタイムスロットメッセージが送信されたことによって状態データに不整合が生じた場合でも，不正サイトサーバが半数未満であればロールバックすることなく解決できる．

Step 3 ユーザはアップデートメッセージの多数決をとるので，不正なアップデートメッセージを送信するサイトサーバ（状態データの改竄を行うサイトサーバなども含む）が半数未満であれば影響を受けない．また，半数以上であっても，すべてのサイトサーバが不正処理を行わない限り，システムが不正を検出して悪意のあるユーザを特定し，ロールバックによって正常な状態に戻すことができる．ロールバックは多くのサービスにとって好ましくない操作であるが，この操作が必要となる確率は非常に低いので，実用上は問題にならない．

4.3 送信抑制

アクションメッセージの送信を行わない場合（Step 1）は，単にそのユーザがアクションを実行できないだけであるから，問題は起こらない．サイトサーバがタイムスロットメッセージやアップデートメッセージを送信しない場合（Step 2，3）も，そのようなサイトサーバが半数未満であればサービスに影響は及ばない．しかし，過半数のサイトサーバが送信抑制を行うと，サービスが停止する可能性がある．また，不正送信の場合と異なり，「送信すべきデータが送信されていない」という証拠を提示することはできないので，悪意のあるユーザを特定することは難しい．

5. 性能分析

本章では，提案した P2P モデルの性能分析を行い，

C/S モデルや，代表的なユーザ中心構成法の 1 つである NEO Protocol と比較する．

5.1 通信負荷

本節では，通信負荷の比較を行う．ユニキャストのみを利用する場合，それぞれのモデルにおいて 1 アクションあたりにユーザが送らなければならないメッセージの合計サイズは：

C/S：$S_{\mathrm{header}} + a$,

NEO：$(n-1)\{S_{\mathrm{header}} + a + \lceil (n-1)/8 \rceil + S_{\mathrm{key}}\}$,

提案法：$N_{\mathrm{server}}(S_{\mathrm{header}} + a)$.

上式において，a はアクションメッセージのサイズ，n は同じサイトにいるユーザの数，S_{header} はメッセージヘッダのサイズ，S_{key} は NEO Protocol において先読みチートを防ぐためアクションメッセージに施している暗号の復号鍵のサイズをそれぞれ表す．

同じサイトにいる他ユーザ全員のアクションを知るためにユーザが受信しなければならないメッセージの合計サイズは：

C/S：$S_{\mathrm{header}} + na$,

NEO：$(n-1)\{S_{\mathrm{header}} + a + \lceil (n-1)/8 \rceil + S_{\mathrm{key}}\}$,

提案法：$N_{\mathrm{majority}}(S_{\mathrm{header}} + na) + (N_{\mathrm{server}} - N_{\mathrm{majority}})(S_{\mathrm{header}} + S_{\mathrm{hash}})$, (1)

ただし S_{hash} はアップデートメッセージから計算されるハッシュのサイズを表す．双方の P2P モデルにおいて，それぞれのユーザはアクションメッセージが処理されるのと同じペースでアクションメッセージを送信するものと仮定した．たとえば，提案法において T_{timeslot} が 100 ミリ秒であるとすると，それぞれのユーザは 100 ミリ秒ごとにアクションメッセージを送信する．さらに，アップデートメッセージのサイズは，アクションメッセージのサイズに，同じサイトにいるユーザの数を掛けたものに等しいとした．それぞれのモデルにおいて，以下の条件 (2) の成立を仮定し☆，N_{server} の値を $\{3, 5\}$，n の値を $\{20, 40\}$ とそれぞれ変化させ，a を横軸にとって描いた受信データ量のグラフを**図 4** に示す．

$S_{\mathrm{header}} = 66\,\mathrm{Bytes}$, $S_{\mathrm{hash}} = 8\,\mathrm{Bytes}$,

$S_{\mathrm{key}} = 8\,\mathrm{Bytes}$. (2)

たとえば

$N_{\mathrm{server}} = 5$, $n = 40$ (3)

のときは，$a = 10\,\mathrm{Bytes}$（ほとんどの場合ユーザの動きを伝えるにはこれで十分）とすると，式 (1) の値は

☆ S_{header} = 20 Bytes (*TCP Header*) + 20 Bytes (*IP Header*) + 26 Bytes (*Ethernet Header and Trailer*) = 66 Bytes（TCP/IP Ethernet の利用を仮定）．

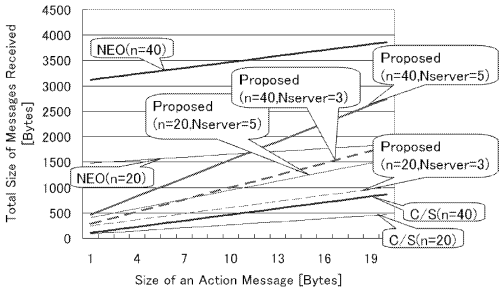

図 4 ユーザの受信データ量

Fig. 4 Total size of messages a user receives.

それぞれ 466 Bytes，3,471 Bytes，1,546 Bytes となる．条件 (2)，(3) のもとでは，$a \leq 33\,\mathrm{Bytes}$ であれば提案モデルは NEO Protocol よりユーザの通信負荷を少なくすることができる．これは，複数ユーザのアクションの結果を同時に受け取ることができるという利点によるものである．

n ユーザがいるサイトのサービス機能を担っているユーザ（すなわちサイトサーバ）には，さらに以下の通信負荷がかかる：

送信：$n(S_{\mathrm{header}} + na)$ (アップデートメッセージを送る場合),

$n(S_{\mathrm{header}} + S_{\mathrm{hash}})$ (メッセージのハッシュを送る場合),

受信：$n(S_{\mathrm{header}} + a)$.

加えて，サイトサーバは自分が管理しているサイトにいるユーザがログイン，ログアウト，サイト移動を行うたびに状態データ（またはそのハッシュ）を管理サーバと交換しなければならない．しかし，これらはサイトを管理する余裕のあるユーザのみが行えばよいことであるので，大きな問題とはならない．

5.2 アクション遅延

それぞれのモデルにおける平均アクション遅延は以下のように概算できる：

C/S：$2l_{\mathrm{cu}}$,

NEO：$2l_{\mathrm{uu}} + (T_{\mathrm{arrival}}/2)$,

提案法：$2l_{\mathrm{su}} + l_{\mathrm{ss}} + (T_{\mathrm{timeslot}}/2)$,

ただし l_{cu}，l_{uu}，l_{su}，l_{ss} はそれぞれ，中央サーバとユーザの間，2 人のユーザの間，サイトサーバとユーザの間，2 つのサイトサーバの間の片道転送遅延を表す．T_{arrival} は文献 6) で arrival delay と呼ばれているもので，提案法における T_{timeslot} に相当するパラメータである．T_{arrival} や T_{timeslot} は，通信負荷を犠牲にすれば限りなく 0 に近づけることができる．提案モデルは NEO Protocol に比べて 1 ホップ多く必要だが，実際にはその差は小さい．なぜならば，高速回

線を利用しているユーザをサイトサーバとして選ぶようにすれば，それぞれの転送遅延には平均的に以下のような関係があるからである：

$$l_{\mathrm{uu}} > l_{\mathrm{su}} > l_{\mathrm{ss}}.$$

これは，ユーザの地理的な位置だけでなく，インターネットへの接続に利用している回線（ラスト・ワン・マイル）の性能も遅延に大きな影響を及ぼすことによる．

5.3 ネットワークの混雑に対するロバスト性

本節では悪意のあるユーザはいないと仮定して，ネットワークの混雑に対するロバスト性を調べる．そのために，確率 p で 2 マシン間のリンクが一時的に不通になるという条件のもとで，あるユーザのアクションが正常に処理されない確率を求めると，

C/S：p,

NEO：$\displaystyle\sum_{k=\lceil n/2\rceil}^{n-1}\binom{n-1}{k}\cdot p^k(1-p)^{n-1-k}$,

提案法：$\displaystyle\sum_{k=N_{\mathrm{majority}}}^{N_{\mathrm{server}}}\binom{N_{\mathrm{server}}}{k}\cdot p^k(1-p)^{N_{\mathrm{server}}-k}$

となる．ここで

$$\binom{n}{k}=\frac{n!}{k!(n-k)!}.$$

図 5 は，この確率をグラフで表したものである．ただし，P2P モデルについては，提案法において $N_{\mathrm{server}}=5$ とした場合について示した．NEO Protocol においても $n=6$ のときは同じ確率になる．このグラフから，C/S モデルに比べて P2P モデルは，パケット遅延に対するロバスト性が高いといえる．C/S モデルでは，ユーザが中央サーバに送信したアクションメッセージが届かなければ，そのアクションメッセージは処理されないが，P2P モデルではアクションメッセージの送信先ユーザのうち過半数に届けば正常に処理されることから，このような差が現れているといえる．たとえば $p=0.01$ とすると，アクションが処理されない確率は C/S モデルでは 10^{-2} であるのに対し，提案モデルでは $N_{\mathrm{server}}=5$，$N_{\mathrm{server}}=7$ のときそれぞれおおむね 10^{-5}，3×10^{-7} となる．提案モデルでは，N_{server} の設定によって，大きい遅延が発生する確率を調整することができる．

5.4 スケーラビリティ

本節では，ユーザ数が増えるとどのようになるかを考察する．

図 6 は，総ユーザ数と，最も人数の多いマップ（本論文でサイトと呼んでいるもの）にいるユーザの人数との関係を示したものである．このグラフは，C/S モデルで提供されている実際の MMO ゲームサービスから 2004 年 8 月 13 日に収集したデータをもとに作成している．そのゲームの世界には 53 個のマップがあり，これは同時接続人数が 400 人程度であれば十分な数であるといえる．1 つのサイトにいるユーザの最大人数には飽和点があると考えられる．

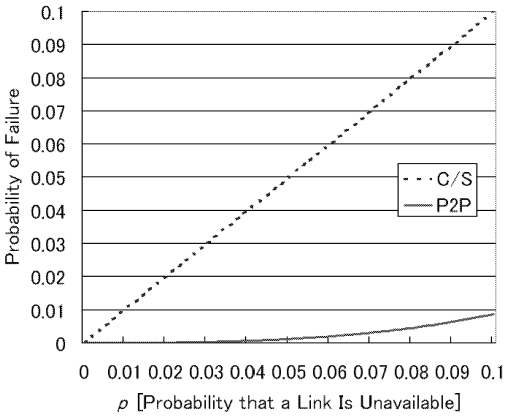

図 5 あるユーザのアクションが処理されない確率
Fig. 5 Probability that a user fails to have his action processed.

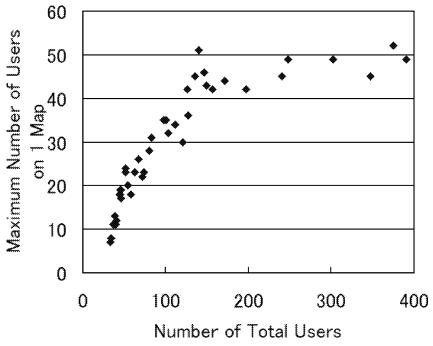

図 6 最も人数の多いマップにいるユーザの人数
Fig. 6 Number of users on the most popular map.

図 7 は，総ユーザ数と，1 時間で最も発言が多かったチャットチャネルでの発言数との関係を示したものである．このデータは上述の MMO ゲームサービスにおいて 2004 年の 1 月から 5 月にかけて収集した．チャットチャネルとは，集まって会話するための部屋のことであり，これもサイトと見なすことができる．そのサービスでは，ユーザが必要に応じて新たなチャットチャネルを作成することができる．図 7 のグラフの傾きがユーザ数の増加とともに小さくなっていることから，多くのユーザがサービスを利用しているとき，1 つのチャットチャネルでの最大発言数は総ユーザ数とはほぼ無関係であるといえる．

これらのデータから，総ユーザ数が増加していくと，

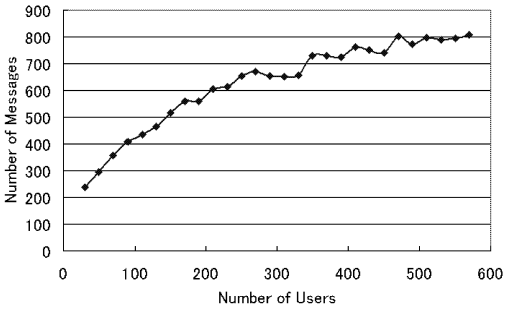

図 7 1 時間で最も発言が多かったチャットチャネルでの発言数（20 ユーザ間隔で平均をとったもの）

Fig. 7 Number of messages sent to the most popular chat channel in an hour (averaged at intervals of 20 users).

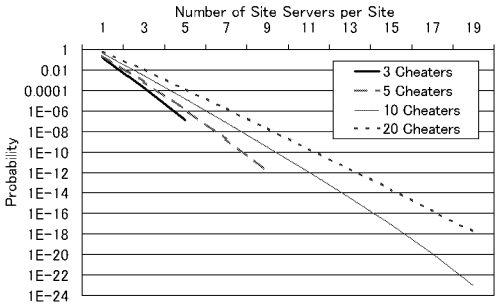

図 8 結託した悪意あるユーザがサービスに影響を与えることができる確率

Fig. 8 Probability that colluding cheaters can affect the service.

最も人気のあるサイトのユーザ数はほとんど変わらなくなっていくことが推測される．これは，通信回線，CPU，人間などの処理能力には限界があるためであるといえる．たとえば MMO ゲームサービスにおいて 1 つのマップにユーザが過度に集中していると，画面描画が遅くなる，アクション遅延が長くなるなどサービス品質の低下が起こる．チャットサービスにおいても，発言数が多すぎると，ユーザが発言をすべて読んでその場の話題についていくことが難しくなる．これらのことが，他サイトへの移動をユーザが考えるきっかけとなる．したがって，アプリケーションに十分な数のサイトが用意されていれば（あるいはユーザが自由に新しいサイトを作ることができる仕組みになっていれば），サイトサーバに負荷がかかりすぎてしまうことはほとんどない．もし何らかの理由で一時的に 1 つのサイトに多数のユーザが集中した場合は，（回線・CPU などの面で）より高性能なマシンにサイト管理機能を移す仕組みにするか，あるいはあらかじめサイトのサービス機能を何らかの形で分割できるような設計にしておき，新たに N_{server} 個サイトサーバを増やして負荷を分散させるようにするなどの対策が考えられる．また，サイトあたりのユーザ数に制限を設けるのも良い方法である．なぜならば，上述したように，混雑しすぎているサイトにいるユーザはいずれにしてもサービスを快適に利用することができないからである．

提案した P2P モデルにおいて管理サーバにかかる通信負荷は総ユーザ数に比例する．しかし，ユーザマシンにサイトのサービス機能を委譲すれば，管理サーバはユーザがログイン・ログアウト・サイト移動などを行ったときのみ処理を行えばよいため，計算負荷や通信負荷が大幅に削減される．したがって，C/S モデルよりはるかに多くのユーザにサービスを提供することができる．分散ストレージシステムなど他の技術と組み合わせれば，管理サーバを必要としない完全にスケーラブルなシステムにすることも可能である．

5.5 不正に対するロバスト性

提案した P2P モデルでは，複数の悪意あるユーザが結託しなければ，不正行為を成功させることができない（$N_{\mathrm{server}} > 1$ のとき）．サイトの数が i 個，悪意のあるユーザが c 人，総ユーザ数が N 人であるときに，サイトサーバの過半数が悪意のあるユーザによって操作されているようなサイトが 1 つ以上ある確率を $C_i(N, c)$ で表すとする．するとこの確率は以下の漸化式によって計算できる：

$$C_i(N,c) = 1 - \sum_{k=0}^{N_{\mathrm{majority}}-1} \frac{\binom{c}{k} \cdot \binom{N-c}{N_{\mathrm{server}}-k}}{\binom{N}{N_{\mathrm{server}}}} \times \{1 - C_{i-1}(N - N_{\mathrm{server}}, c-k)\},$$

$$C_0(N,c) = 0.$$

一例として $N = 5000$, $i = 250$ と仮定し，N_{server} を変えると $C_{250}(5000,3)$，$C_{250}(5000,5)$，$C_{250}(5000,10)$，$C_{250}(5000,20)$ がどのように変化するかを表したグラフを**図 8** に示す．結託した悪意あるユーザがサービスに影響を与えることができる確率は，1 つのサイトを管理するサイトサーバの数を増やせば指数的に減少する．

6. ま と め

本論文で提案したモデルには既存の研究と比較して以下のような特徴がある．

(1) 同じデータを管理する複数のマシンで多数決をとるという方法を MMO サービスに適用した

こと．この仕組みによってデータの改竄や消失に対する信頼性を高めることができる．さらに，サービスはネットワーク混雑に対するロバスト性を持っている．

(2) サイトを管理するユーザは，そのサイトにいないユーザの中から管理サーバが選択すること．これによって不正を行う利点が減り，結託も難しくなる．

(3) アクションメッセージの処理順序が，それぞれのサイトサーバへの到着時刻の中央値を基準として決定されること．これは，メッセージに含まれるタイムスタンプの改竄による影響を少なくする効果がある．

(4) すべてのユーザマシンがサイトを管理しなければならないわけではないということ．したがって，サイトのサービス機能を委譲するルールを適切に決めれば，低速回線，低性能マシンを利用しているユーザや，NAT ルータの裏にいるユーザなどでもサービスを利用することができる．

今後の課題としては，まず実際の MMO ゲームサービスを利用した実験により，実用性を検証することがあげられる．5.4 節であげた実際の MMO ゲームサービスは C/S モデルで運営されているため，総ユーザ数が多くなるとしばしばアクション遅延が長くなるなどの不具合が生じている．提案モデルを適用することによってこの不具合が解消され，より多くのユーザにサービスを提供することができるようになることを確かめたいと考えている．

また，4 章で述べたように，状態データの盗聴防止に関する研究を行うことも計画している．

参 考 文 献

1) Diot, C. and Gautier, L.: A Distributed Architecture for Multiplayer Interactive Applications on the Internet, *IEEE Network*, Vol.13, No.4, pp.6–15 (1999).
2) Baughman, N.E. and Levine, B.N.: Cheat-Proof Playout for Centralized and Distributed Online Games, *Proc. Infocom 2001* (2001).
3) Cronin, E., Filstrup, B. and Jamin, S.: Cheat-Proofing Dead Reckoned Multiplayer Games, *Proc. ADCOG 2003* (2003).
4) Cronin, E., Kurc, A.R., Filstrup, B. and Jamin, S.: An Efficient Synchronization Mechanism for Mirrored Game Architectures, *Multimedia Tools and Applications*, Vol.23, No.1, pp.7–30 (2004).
5) Brun, J., Safaei, F. and Boustead, P.: Tailoring Local Lag for Improved Playability in Wide Distributed Network Games, *Proc. NSIM 2004* (2004).
6) GauthierDickey, C., Zappala, D., Lo, V. and Marr, J.: Low Latency and Cheat-Proof Event Ordering for Peer-to-Peer Games, *Proc. NOSSDAV 2004* (2004).
7) Bharambe, A.R., Rao, S. and Seshan, S.: Mercury: A Scalable Publish-Subscribe System for Internet Games, *Proc. NetGames 2002* (2002).
8) Knutsson, B., Lu, H., Xu, W. and Hopkins, B.: Peer-to-Peer Support for Massively Multiplayer Games, *Proc. Infocom 2004* (2004).
9) Iimura, T., Hazeyama, H. and Kadobayashi, Y.: Zoned Federation of Game Servers: A Peer-to-Peer Approach to Scalable Multi-Player Online Games, *Proc. NetGames 2004* (2004).
10) Mills, D.L.: Network Time Protocol (Version 3) Specification, Implementation and Analysis, RFC 1305 (1992).
11) Park, K. and Lee, H.: On the Effectiveness of Route-Based Packet Filtering for Distributed DoS Attack Prevention in Power-Law Internets, *Proc. SIGCOMM 2001* (2001).
12) Anderson, T., Roscoe, T. and Wetherall, D.: Preventing Internet Denial-of-Service with Capabilities, *ACM SIGCOMM Computer Communications Review*, Vol.34, No.1, pp.39–44 (2004).
13) Chang, R.K.C.: Defending against Flooding-Based Distributed Denial-of-Service Attacks: A Tutorial, *IEEE Communications Magazine*, Vol.40, No.10, pp.42–51 (2002).
14) Krishnamoorthy, S. and Dasgupta, P.: Tackling Congestion to Address Distributed Denial of Service: A Push-Forward Mechanism, *Proc. GLOBECOM 2004* (2004).
15) Handley, M. and Greenhalgh, A.: Steps Towards a DoS-resistant Internet Architecture, *Proc. SIGCOMM 2004* (2004).
16) Rabin, T. and Ben-Or, M.: Verifiable Secret Sharing and Multiparty Protocols with Honest Majority, *Proc. STOC '89* (1989).

(平成 17 年 7 月 8 日受付)
(平成 18 年 2 月 1 日採録)

遠藤　慶一（学生会員）

昭和 55 年生．平成 15 年京都大学工学部情報学科卒業．平成 17 年京都大学大学院情報学研究科システム科学専攻修士課程修了．現在，同専攻博士後期課程在学中．大規模ネットワーク・アプリケーションの負荷分散およびセキュリティに関する研究に従事．電子情報通信学会，ゲーム学会各学生会員．

川原　　稔（正会員）

昭和 63 年早稲田大学理工学部電気工学科卒業．平成 2 年京都大学大学院工学研究科応用システム科学専攻修士課程修了．同年京都大学大型計算機センター助手．平成 16 年愛媛大学総合情報メディアセンター助教授，現在に至る．京都大学博士（情報学）．情報検索，データマイニング，情報ネットワークの研究に従事．IEEE，人工知能学会各会員．

高橋　　豊（正会員）

昭和 26 年生．昭和 55 年京都大学大学院工学研究科数理工学専攻博士課程単位取得退学．同年同大学工学部数理工学教室助手．昭和 58 年 4 月より 1 年間，フランス INRIA（国立情報制御研究所）客員教授．平成元年同助教授．平成 8 年奈良先端大情報科学研究科教授．平成 11 年より京都大学大学院情報学研究科システム科学専攻教授．工学博士．待ち行列理論・トラヒック理論，情報システム・情報ネットワーク等のモデル化と性能評価に関する研究に従事．平成 13 年 IFIP（International Federation for Information Processing）Silver Core 受賞，Telecommunication Systems 等の Associate Editor．IFIP WG6.3 Former Cochairman．Elected Full Member of IFIP WG6.3 and WG7.3．日本オペレーションズ・リサーチ学会，電子情報通信学会，応用数理学会，システム制御情報学会各会員．