平成 18 年度

筑波大学第三学群情報学類

卒 業 研 究 論 文

題目 スケーラブルなクライアントサーバ型ストリーム中継システムの研究

主専攻 計算機システム主専攻

著 者 登 大遊

指導教員 板野肯三・新城 靖・佐藤 聡

要　　旨

インターネット上で、NAT やプロキシ等の、内側から外側に対する方向のみが可能なルータ等を経由してコンピュータを接続することが多くある。NAT 越えやプロキシ越えなどの複雑な処理が必要となるが、これらの処理をアプリケーションごとに個別に実装するのは効率が悪い。

複数台のコンピュータが NAT 等の内側にあっても、コンピュータ間で TCP のような双方向のストリームコネクションを確立し、自由に通信できれば、アプリケーションを開発する上で効率が良い。

本研究では、従来の関連する各種通信技術等について検証し、これらの問題点を解消するため、新しいクライアントサーバ型ストリーム中継システムを設計する。このシステムは、中継ゲートウェイ、サーバおよびクライアントの 3 つのモジュールに分かれており、中継ゲートウェイはスケーラビリティを確保するため、負荷分散を実現する形で実装する。サーバおよびクライアントは、ソケットと同等の API を提供するライブラリとして実装する。その後、性能測定等の実験を行う。また、実装した各モジュールの利用例として、新しいリモートデスクトップツールを実装する。

目　　次

第 1 章 序論

1.1 背景

インターネットの利用者は、増加傾向にあり、今後も利用者が増加するものと予想されているが、グローバル IP アドレス (IPv4 アドレス) の不足問題が生じてきている[1]。近年では、クライアントコンピュータにはプライベート IP アドレスを割り当てることが通常となり、複数台をまとめて 1 個のグローバル IP アドレスを共有させるために、NAT[2]やプロキシサーバ等の技術が利用されるようになってきた。また、近年インターネット上における不正アクセスやセキュリティホールを突く攻撃などが増加しているため、セキュリティを高めるために、プライベートネットワークとインターネットとの間にファイアウォールを設置することがある。

これらの接続方法の増加により、任意の 2 台のコンピュータの間で自由に IP パケットを送受信することができるという、TCP/IP を利用する上でのメリットを、インターネット上で享受できにくくなった。近年のインターネットの利用方法のうち、HTTP/HTTPS プロトコルを用いた Web サイトの閲覧や掲示板等への書き込み、オンラインショッピング等の電子商取引への参加や、SMTP や POP3 等のプロトコルを用いた電子メールの送受信、メッセージングサービスソフトウェアを利用した IM (インスタントメッセンジャー) サービスの利用などは、予めサービス提供者が設置したサーバに複数ユーザがアクセスする形態の、クライアント/サーバ型の通信である。

ソフトウェア開発者が、インターネット上の任意の 2 台のコンピュータ間で自由に通信を行うことができるという TCP/IP の性質を積極的に活用した通信アプリケーションを開発しても、その通信アプリケーションを NAT 等が障壁となり実際にインターネット上で使用することができないユーザが大半である。このような環境上でも、ユーザのコンピュータ間で 2 点間通信を行うための通信システムがあれば、インターネットをより活用したアプリケーションを開発することができる。

上記のような現象と並行して、インターネットのバックボーン回線や、ユーザがインターネットに接続するためのアクセス回線は、物理的に増強されてきている。同時に大量のユーザがアクセスし、それぞれのユーザが多量のトラフィックを消費するようなサーバをインターネット上に設置することも可能となった。例えば、米国の YouTube いうオンラインビデオ投稿サイトのシステムは、大容量の回線でインターネットに接続されており、その平均通信トラフィック量は 20Gbps 以上とされている。

太いバックボーン回線でインターネットに接続された場所にサーバを設置すれば、大量のトラフィックを集中させて一箇所で処理する中継サーバを設置できる。物理的な通信経路としてはクライアント/サーバ型であるにもかかわらず、論理的な通信経路としては、任意の 2 台のコンピュータ間で自由に通信を行うことができるという形態を採用した汎用的な通信システムがあれば望ましい。

1.2 目的

本研究の目的は、インターネット上で利用することができる、コネクション型の双方向通信を行うための通信の仕組みであって、かつ通信内容がインターネット上に設置したゲートウェイシステムを経由するような形態で動作するシステムを実現することである。通信を行おうとするコンピュータの台数が増えた場合でも支障なく動作するようにスケーラビリティを確保してあるような通信システムを作成する。また、その通信システムを利用した通信アプリケーションを開発者が容易に扱えるようにするためのライブラリを作成し、そのライブラリを実際に呼び出す実験用アプリケーションを作成し性能測定や実証実験等を行うことも目的に含まれる。

インターネット上で通信を行う際には、通信プログラムの用途や性質によって、大きく分けて、コネクション型のストリームを送受信することが望ましい場合と、コネクションレス型のデータグラムを送受信することが望ましい場合の 2 通りがある。

本研究では、コネクション型のストリームを送受信する仕組みをインターネット上で利用しようとすると発生することがあるいくつかの問題を解決する新しい仕組みが必要であると考え、それをシステムとして実装する。

インターネット上で使用されている TCP/IP プロトコルにおける TCP コネクションは、1 本のコネクションに同時に双方向に通信を行うことができるようになっている。本研究では、TCP コネクションを使用して通信を行う既存のプログラムを、本システムを使用して通信を行うように書き換えることを容易に行えるようにするため、TCP と同様に、1 本のコネクションにおいて双方向ストリーム通信を行うことを実現する。

現状、インターネットを利用する際には多くの場合 NAT やプロキシサーバ等を経由しなければならない環境が多数存在する。以下、これらを経由して支障無く通信を行う技術を総称して **NAT 越え**と呼ぶ。本研究では、NAT 越えを自動的に行うことを実現する。

インターネットはその性質上、中間者攻撃等のセキュリティ侵害行為を起こすことが可能である。これらの危険性を想定した上で、それらの攻撃を受けそうになった場合は自動的にそれを検出し、直ちに通信を中断するといった安全機能がプログラムに要求されている。本研究では、通信を行う際に自動的に十分なセキュリティの確保が行われることを実現する。

本研究によるシステムでは、インターネット上で複数台のコンピュータが通信をしようとするとき、その通信内容はインターネット上に予め設置されたゲートウェイシステムを経由することになる。この場合、通信システムの利用者が増加した際に、通信性能が低下したり、ゲートウェイシステムが破綻したりすることがないようにしなければならない。本研究では、利用者の増加に合わせていつでもゲートウェイシステムの処理能力を増強することができるようにするため、スケーラビリティおよびアベイラビリティを追求し、それを実現する。

優良な通信プロトコルを設計・実装したとしても、その通信プロトコルを利用する優良なアプリケーションがなければ、ユーザは利得を受けることができない。インターネットで使用されているアプリケーションの多くはソケット API を使用して TCP/IP 通信を行っているので、これらのアプリケーションを、容易に本通信システムを利用するように書き換え

ることができるのが望ましい。そこで、本研究で実現する通信システムを使用するためのソケット API と類似の通信 API を提供することを実現する。

本研究で作成する通信システムおよびライブラリを使用した実験用プログラムを作成すれば、実験環境での性能測定は実施できる。しかし、インターネット上で多数のコンピュータが同時に本通信システムを使用しても問題が生じないことを確認する必要があるので、実験環境よりもより大規模な負荷テストを行うことができるようにするため、本通信システムを利用して通信を行うプログラムを実装例として開発し、インターネット上で利用者を募集して配布し、実証実験を実施する。

1.3 構成

第 2 章では、本研究に関係する既存の技術およびそれらの問題点について述べる。第 3 章では、問題点を解決するための新しい通信システムの要件を定義し、全体の設計および各モジュールの詳細な設計を述べる。第 4 章では、設計した通信システムを実際のコンピュータ上で動作させるために行ったプログラミング方法およびその結果得られたプログラムについて述べる。第 5 章では、実装したプログラムを用いて通信性能の測定を行う。第 6 章では、より実用的なアプリケーションに組み込み、インターネット上で多数のユーザに対して配布し実際に利用してもらった結果について述べる。第 7 章で研究のまとめと今後の課題を述べる。

第 2 章 関連研究

本章では、本研究に関係する既存の研究や技術およびそれらの問題点について述べる。

2.1 TCP コネクション

インターネット上で使用されている TCP/IP プロトコルは、セッション層およびトランスポート層に位置するプロトコルとして TCP を使用している。TCP はコネクション志向の双方向通信が可能なストリームデータの送受信を行うためのプロトコルである。TCP によって 2 台のコンピュータが通信を行うときは、一方のコンピュータが予め決めてあるポート番号の TCP ポートを待ち受け状態にしておき、もう一方のコンピュータが、前者のコンピュータが待ち受け状態にしている TCP ポートに対して TCP コネクションの確立要求を行う。その後、スリーウェイハンドシェイクと呼ばれる手順に従って TCP コネクションが確立される。TCP コネクションが確立された後は、2 台のコンピュータはその TCP コネクションの中で、双方向でストリーム型のデータ転送を任意に行うことができる。TCP コネクションが不要になった場合は、それを切断することができる。また、ネットワークエラー等で通信が不能になった場合も、タイムアウト等により TCP コネクションが切断されたとみなすことができる。TCP コネクションの確立、データの送受信、および切断の手順を、図 1 に示す。

TCP を利用したプロトコルとしては、HTTP、HTTPS、FTP、Telnet、SSH、POP3、SMTP、IMAP4 等多くのものがあり、長年の間の実績がある。しかしながら、TCP を利用する場合は、同時にいくつかの制約事項がある。その主なものを以下に述べる。

1. TCP コネクションを接続しようとする側のコンピュータ (発信側) は、接続を待ち受ける側 (受信側) のコンピュータの IP アドレスを知らなければならず、また、発信側から発せられる接続要求パケットを受信側が受信できなければならない。
2. 発信側と受信側の IP ネットワークは、双方のコンピュータ同士が送受信しようとする IP パケットの伝送を実施できなければならない。
3. TCP は暗号化および電子署名をサポートしないため、TCP より下のレイヤが暗号化および電子署名を実施しない場合は、TCP を利用するアプリケーションは、セキュリティを実現するため、暗号化および電子署名を自ら処理しなければならない。

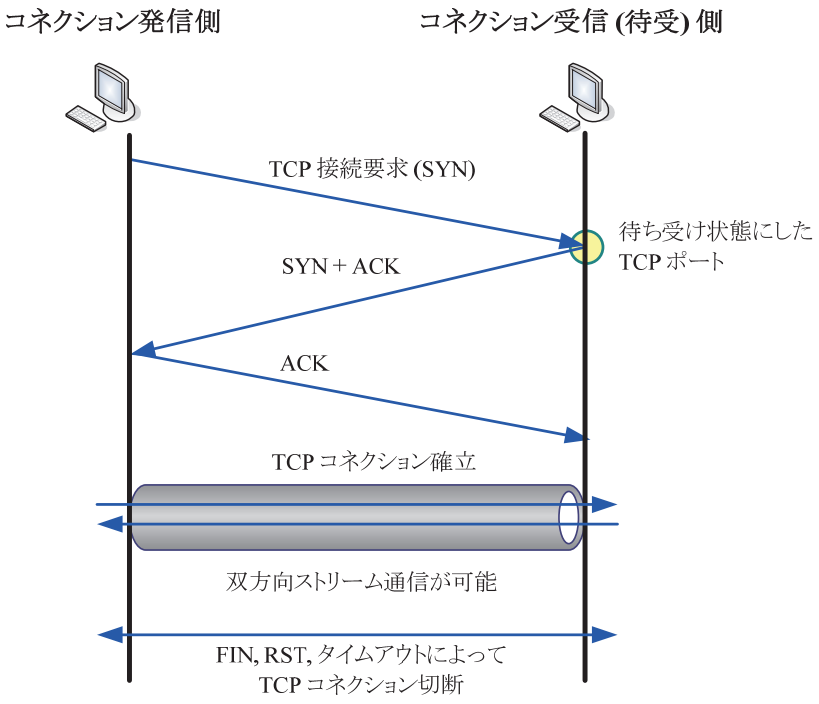

図 1 TCP コネクションの確立および通信の手順

2.2 グローバル IP アドレス

グローバル IP アドレスとは、インターネット上に接続されており、IP パケットの伝送を受けることができるコンピュータを識別するための IP アドレスである。グローバル IP アドレスを割り当てられているコンピュータは、インターネットに接続されている他のコンピュータから、当該グローバル IP アドレスを宛先として指定した IP パケットが発信されれば、その IP パケットを受信することができる。

2.1 で述べたように、TCP コネクションを確立するためには、発信側と受信側の双方のコンピュータが IP ネットワーク上で互いに IP パケットを送受信できる必要がある。また、インターネット上で IP パケットの送受信を行うコンピュータにはグローバル IP アドレスが割り当てられていなければならない。従って、インターネット上で TCP コネクションを確立して通信を行おうとする場合は、発信側のコンピュータと受信側のコンピュータはそれぞれグローバル IP アドレスを持っている必要がある。

しかしながら、現在広く利用されている IPv4 ネットワークにおいては、すべてのコンピュータにグローバル IP アドレスを割り当てることができない状態となっているため、NAT (2.4 で説明する) などの技術により、複数のコンピュータに対して 1 個のグローバル IP アドレスを割り当てる技術を使用することにより、グローバル IP アドレスの使用個数を節約することが一般的である。

なお、グローバル IP アドレスは IPv4 ネットワークにおいては、**130.158.80.244** のような、0 から 255 までの数字を 4 個並べてドットで区切った形式で記述することが一般的であるが、このような IP アドレスは人間にとっては覚えにくいので、代わりに

maple.cs.tsukuba.ac.jp のようにホスト名を付けて管理することが多い。

2.3 DDNS

インターネット上であるコンピュータが別のコンピュータに対して TCP コネクションの確立要求を行うときは、接続先のコンピュータのグローバル IP アドレスを知らなければならない。しかし、一般家庭等におけるインターネット接続形態である ISP (インターネットサービスプロバイダ) を経由した接続では、一度インターネットから切断し、再接続すると、コンピュータに割り当てられていた IP アドレスが変化することがある。また、企業や大学などで予めある程度の個数のグローバル IP アドレスを固定的に確保している場合でも、管理上の都合などにより、特定のコンピュータに割り当てられた IP アドレスが不定期に変化することもある。

このように特定の IP アドレスが変化した場合でも、そのコンピュータに対して必ず TCP コネクションを確立することができるようにするため、DDNS (ダイナミック DNS) と呼ばれる技術を利用することができる。DDNS の仕組みを、図 2 に示す。DDNS は、予め特定のコンピュータに特定のホスト名を割り当てることにしておき、DDNS のゾーン上の当該ホスト名を示す A レコードが示す IP アドレスに、当該コンピュータの最新のグローバル IP アドレスを反映し続けるように努力するシステムである。これによって、ユーザは DDNS 上のホスト名を覚えて置けば、インターネット上の別のコンピュータから、いつでもそのホスト名によって示される IP アドレスを持ったコンピュータにアクセスすることができる。

しかし、DDNS は DNS の仕組みを応用した技術であり、これを利用するためには、ユーザはある程度の IP アドレスおよび DNS に関する仕組みを理解している必要があり、難易度が高い。また、ISP 等がユーザ向けに提供する参照用 DNS サーバや、クライアントコンピュータにインストールすることができる通信高速化ソフトウェアなどは、元の DNS レコードの TTL (Time to live) 値よりも長い時間レコードをキャッシュするよう設定されているものがあり、クライアントが IP アドレスの変更を知るまでに時間がかかることがある。

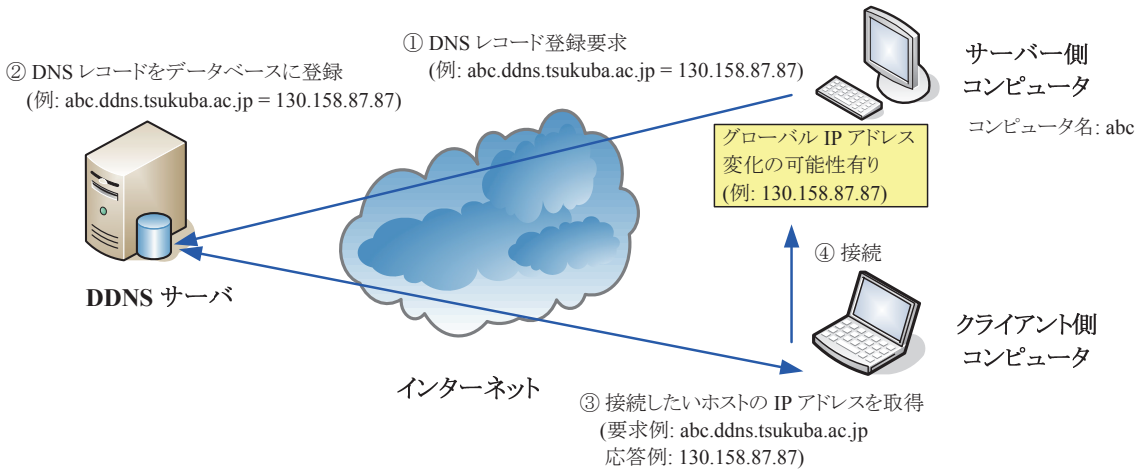

図 2 DDNS の仕組み

2.4 NAT

NAT とは、Network Address Translator (ネットワークアドレス変換) の略であり、一部では NAPT (Network Address Port Translation, ネットワークアドレスおよびポート変換) と呼ばれることもある。後者のほうが呼称としては正確であるが、近年はほとんどの場合ポート変換を伴うので、NAPT のことを NAT と呼ぶことが多い。

2.2 で述べたように、インターネット上で TCP 通信を行う場合は、原則として 1 台のコンピュータに 1 個のグローバル IP アドレスを割り当てなければならない。しかしながら、IP アドレス枯渇の問題等により、それぞれのコンピュータにグローバル IP アドレスを割り当てることができないことがある。この場合は、複数のコンピュータを NAT の内側に設置し、NAT にグローバル IP アドレスを 1 個割り当てることによって、複数のコンピュータに 1 個の IP アドレスを共有させることができる。この場合、NAT の内側にあるコンピュータは、NAT の内側でのみ通用するプライベート IP アドレスを持っており、NAT はグローバル IP アドレスとプライベート IP アドレス (および必要な場合はポート番号) の付け替えを行う。NAT によってグローバル IP アドレスを共有したネットワーク例を、図 3 に示す。

以下、NAT の内側にあり、NAT に割り当てられているグローバル IP アドレスを共有しているコンピュータの状態を **NAT の内側にある**、NAT の内側になく、グローバル IP アドレスを直接割り当てられているコンピュータを**グローバル上にある**と表現する。

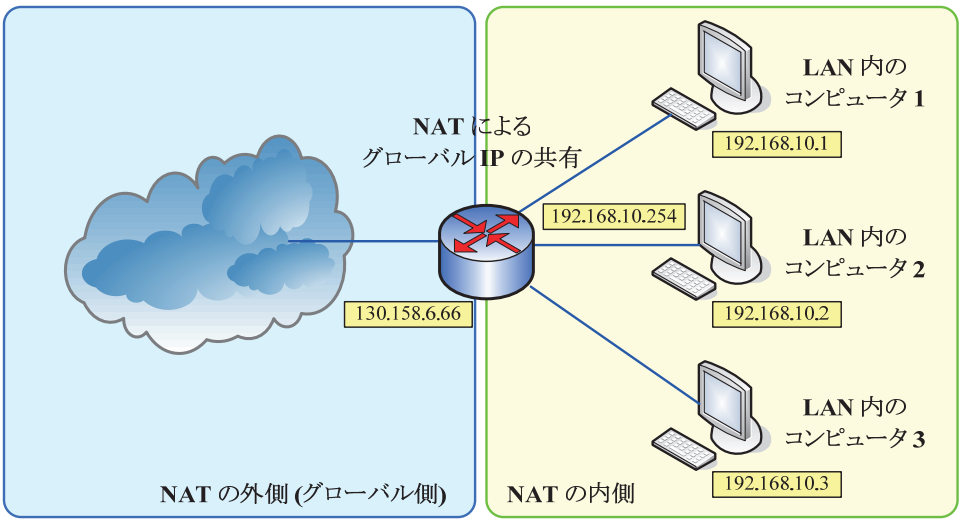

図 3 NAT によるグローバル IP アドレスの共有

NAT は、TCP 通信を正常に行うことができない原因となることがある。たとえば、コンピュータ A がグローバル上にあり、コンピュータ B が NAT の内側にある場合、コンピュータ B から発せられたコンピュータ A への TCP コネクション接続要求はコンピュータ A に届き、これにより TCP コネクションを確立させ、TCP 通信を行うことが可能である。しかし、逆にコンピュータ A からは、コンピュータ B に対して TCP コネクション接続要求を発することはできない。なぜならば、NAT の内側にあるコンピュータ B は NAT の内側でのみ通用するプライベート IP アドレスしか持っておらず、コンピュータ A は当該プライベート IP アドレスを指定して IP パケットを送信することができないからである。従って、コネクシ

ョンの確立要求を行うことができる方向が一方向となってしまう (図 4)。

さらに、インターネット上に 2 つの NAT があり、それぞれの NAT の内側に 2 台のコンピュータがある場合は、それらのコンピュータは、TCP コネクションの確立要求を互いに送信することができないので、後述するいくつかの仕組みがなければ、2 台のコンピュータ間で TCP 通信を行うことは不可能である。

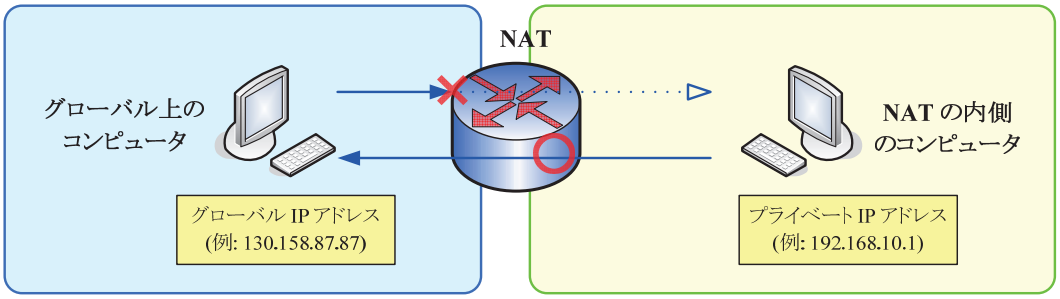

図 4 NAT によって生じるコネクション確立要求の一方向性

2.5 ポートマッピング

グローバル上のコンピュータ A が NAT の内側のコンピュータ B に対して TCP コネクションを確立させるための方法として、NAT に予めポートマッピングと呼ばれる設定を行っておくことができる。ポートマッピングとは、NAT に割り当てられているグローバル IP アドレスの特定のポート番号に対して TCP 接続要求が行われた場合、その接続要求パケットを自動的に NAT の内側にある特定のコンピュータの特定のポート番号に対して転送するという機能である (図 5)。これにより、事前設定により、コンピュータ A はコンピュータ B に対して TCP コネクションを接続することができる。

しかし、この方法を使用するためには、NAT の事前設定が必要になる。そのための設定インターフェイスは統一されておらず、NAT の実装によって様々であるので、自動的にこの設定を行うことは困難である。従って、NAT を設置するユーザによって事前設定を行う必要があるが、TCP/IP に関する知識が必要であり、難易度が高い。

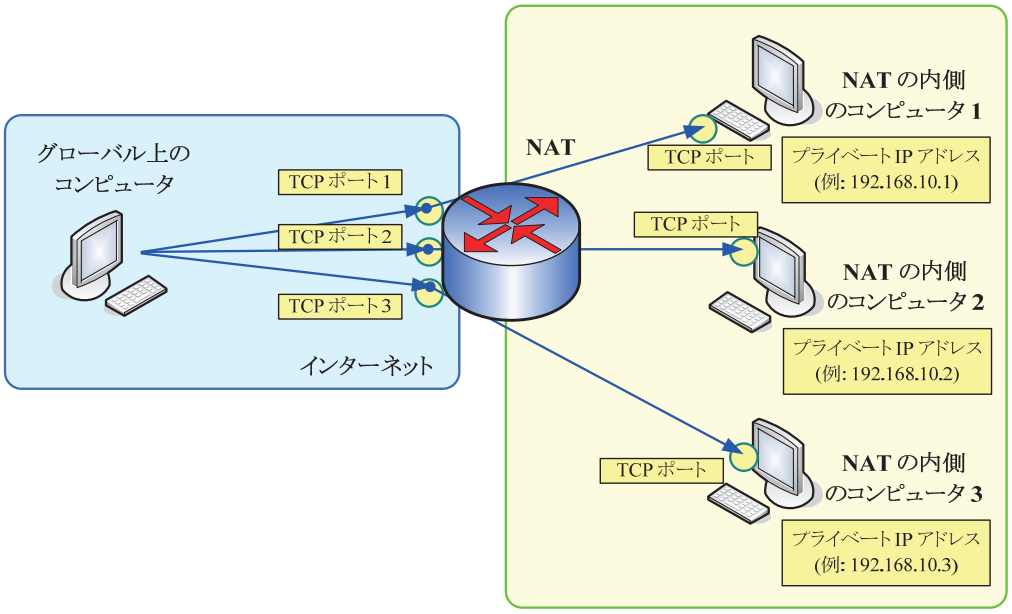

図 5 NAT におけるポートマッピング

2.6 Universal Plug and Play

Universal Plug and Play (UPnP) [3] とは、Microsoft によって提唱された、機器を接続しただけでネットワークに参加することを可能にするプロトコルである。

UPnP に対応した NAT は、UPnP の機能の一部である Simple Service Discovery Protocol (SSDP) を利用することにより、自動的にポートマッピングと同等の設定を行うことができ、グローバル上のコンピュータが NAT の内側のコンピュータに対して TCP コネクションを確立することができるようになる。

これにより、NAT の内側のコンピュータに対して、グローバル上のコンピュータから TCP 接続要求を行うことができないという問題が解決される。しかしながら、UPnP が組み込まれているのは NAT の実装のうち一部であり、UPnP が搭載されていない NAT も存在するので、汎用的な解決方法とはならない。

2.7 UDP ホール・パンチング

UDP ホール・パンチング[4] は、NAT の内側にあるコンピュータに対して、グローバル上のコンピュータから UDP パケットを送信するための手法の一つである。UDP は TCP と異なりデータグラム形のパケットを送受信する通信プロトコルであり、元々データグラム型である IP パケットに UDP ヘッダを付加しただけのものである。

UDP ホール・パンチングは、**Full-Cone NAT** と呼ばれる NAT 実装でのみ利用でき、**Restricted Cone NAT** または **Symmetric NAT** と呼ばれる NAT 実装では利用できないため、汎用的な解決方法とはならない[5]。

また、UDP ホール・パンチングが利用できる環境であっても、本研究の目的であるコネクション型のストリーム通信を実現するためには、TCP と同等のプロトコルを UDP よりも

上のレイヤで実装しなければならないため、技術的に利用が困難である。

同等の手法で、TCP フォーウェイ・アンドシェイクという特殊な手段で、グローバル上のコンピュータから NAT の内側のコンピュータに対して TCP コネクションを接続するという方法が利用できる場合がある。しかし、この方法も多くの場合 Full-Cone NAT でしか利用できない。

2.8 プロキシサーバ

プロキシサーバとは、通信を代理するサーバのことである。プライベートネットワーク内のコンピュータがインターネット上のコンピュータとの間の通信を行おうとするときに、直接インターネット上のコンピュータに接続して通信するのではなく、一旦プロキシサーバに接続し、そのプロキシサーバに目的の接続先コンピュータのアドレス等を通知し、接続要求を行う。接続が確立された後は、プライベートネットワーク内のコンピュータと、接続先のインターネット上のコンピュータとの間の通信は、すべてプロキシサーバが仲介して行うことになる。図 6 に、プロキシサーバを用いた例を示す。

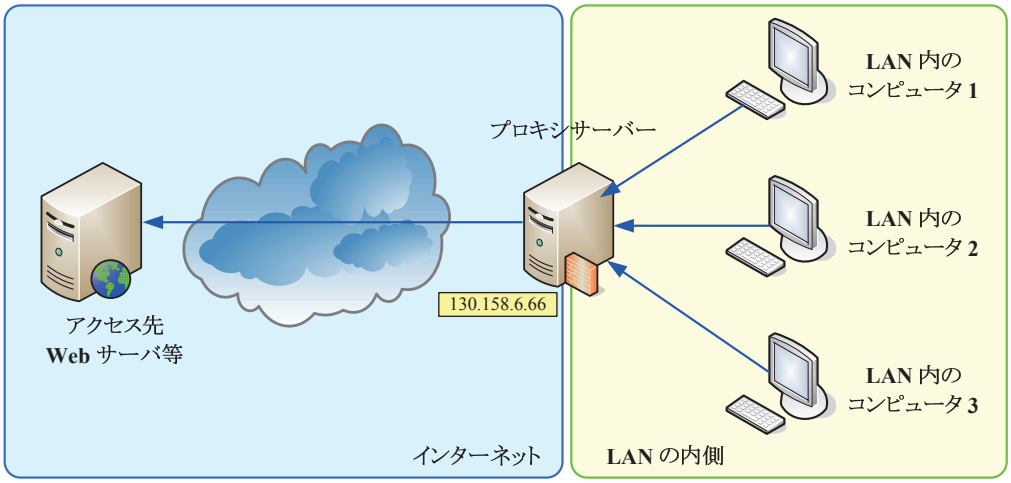

図 6 プロキシサーバを用いた例

プロキシサーバにはいくつかの種類があるが、企業ネットワーク等で使用されているプロキシサーバとしては、HTTP/HTTPS プロトコルを中継するためのものが多い。プロキシサーバを利用すると、NAT と同様に、複数のコンピュータで 1 個のグローバル IP アドレスを共有することができるという利点がある。また、プロキシサーバは通信内容を検査したり、通信内容によってフィルタリングを行ったりできる。しかし、プロキシサーバはそのプロキシサーバが想定しているプロトコルのみを中継することができる。HTTP/HTTPS プロキシサーバは、HTTP/HTTPS プロトコルを中継することができるが、他のプロトコル (例えば SMTP) を中継することができない。プロキシサーバを経由しなければインターネット上のコンピュータと通信することができない形態のネットワークで、プロキシサーバが利用したいプロトコルに対応していない場合は、プライベートネットワーク上のコンピュータで

は、原則としてそのプロトコルを利用することができない。

また、NAT と同様に、通常はプライベートネットワーク側にあるコンピュータに、グローバル上にあるコンピュータから接続を行うことができない。この問題を解決するためには、NAT におけるポートマッピングと同様に、プロキシサーバにおけるリバースプロキシと呼ばれる設定を行う必要があるが、この設定はネットワークに関する知識が必要であり、難易度が高い。また、リバースプロキシの設定に対応していないプロキシサーバの実装も存在する。

2.9 ファイアウォール

ファイアウォールとは、プライベートネットワーク上のコンピュータに対して、グローバル上のコンピュータからの不正な攻撃等の通信パケットが届かないように、そのようなパケットを遮断するためのシステムである。前述の NAT やプロキシサーバもファイアウォールの一部として使用することができる。

一般的に、ファイアウォールを企業等のネットワークに設置する場合、ファイアウォールの内側のコンピュータからグローバル上のコンピュータに対して接続要求が発せられる TCP コネクションの通信は許可し、逆に、グローバル上のコンピュータからファイアウォールの内側のコンピュータに対して接続要求が発せられる TCP コネクションの通信は遮断するというような運用が行われる。このような場合、ファイアウォールの内側のコンピュータに対してグローバル上のコンピュータから接続を行いたいようなときき、その都度、例外としてその通信パケットを経由させるためのルールをファイアウォールに追加する必要がある。しかし、ファイアウォールの実装の多くは、設定にあたって専門的な知識が必要であり、設定変更の難易度が高い。また、ネットワークセキュリティを高めるために適切に設置されているファイアウォールの設定を変更し、例外ルールを登録設定することは、その登録作業に間違いがあった場合に、逆にセキュリティを低下させる原因となる可能性があり、危険である。

また、一部のファイアウォールは、通信パケットをリアルタイムに解析し、アプリケーションレイヤまでその内容を解釈するものがあり、また解釈に失敗した通信パケットを遮断するような仕様となっているものがある。このような場合、ファイアウォールに通信内容の解析プログラムが組み込まれていないプロトコル等を、ファイアウォールを経由して利用することができない。

2.10 SSH ポート転送

SSH (Secure Shell) [6] とは、暗号や認証の技術を利用して、安全にリモートコンピュータと通信するためのプロトコルである。従来の Telnet に代わるセキュアなリモートログイン方法として、広く使用されている。

SSH ポート転送とは、SSH サーバに SSH クライアントが接続した状態で、SSH クライアント側で特定の TCP ポートを開き、その TCP ポートに対して接続要求があった場合は、SSH

クライアントの指示によって予め SSH サーバに対して設定された宛先ホストおよび TCP ポートに対して、SSH サーバから TCP コネクションを確立し、その後は SSH クライアントの開いている TCP ポートに接続されている TCP コネクションと、SSH サーバが確立した宛先ホストとの間の TCP コネクション間の通信が SSH プロトコル上にカプセル化されて伝送されることになる、VPN (Virtual Private Network, 仮想プライベートネットワーク) の一種である (図 7)。これにより、SSH クライアントは、直接接続することはできないが、SSH サーバから接続することができる宛先ホストに対して、TCP コネクションを確立して通信することができる。

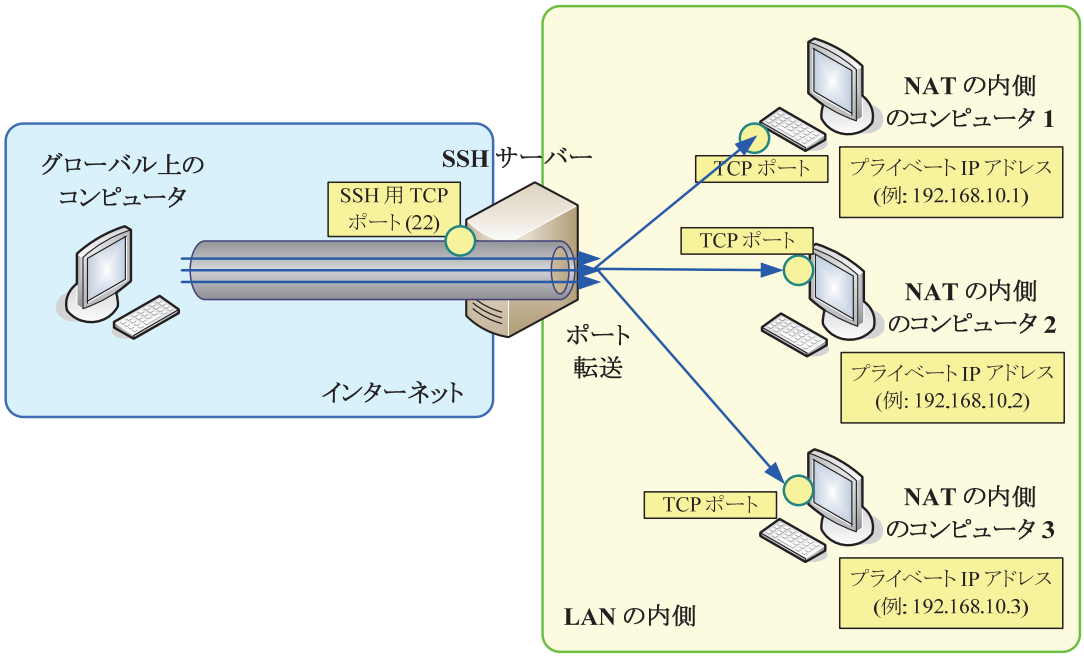

図 7 SSH ポート転送の仕組み

前述した NAT やファイアウォールの多くは、SSH パケットを安全であると見なすので、SSH プロトコルにカプセル化した SSH ポート転送を使用することにより、NAT やファイアウォールがある環境でも、任意の TCP 通信アプリケーションを利用できる場合がある。

しかし、SSH ポート転送を使用する場合は、接続先コンピュータに対して TCP コネクションを確立することができる位置に SSH サーバを予め設置しておくことが必須であり、接続先コンピュータに対して接続することができる SSH サーバが設置されていない場合は SSH ポート転送を使用することができない。また、NAT に設定を行って、グローバル上から NAT の内側の SSH サーバへの SSH 接続要求を転送するように設定しておく必要がある。さらに、ほとんどのプロキシサーバの実装は、SSH プロトコルの中継を行うことができない。

逆に、SSH ポート転送の転送処理を SSH サーバ側ではなく SSH クライアント側で行うこともできる。この場合は、SSH サーバを予めグローバル上に置いておき、NAT の内側の SSH クライアントでポート転送処理を行うように設定しておくことにより、別の場所にあるコンピュータが SSH サーバに接続すると、NAT の内側の SSH クライアントを経由して NAT

の内側の別のコンピュータと通信することができる。この方法では、NAT の内側に SSH サーバを常駐させておく必要はなく、また NAT に特別な設定を行う必要はない。しかし、グローバル上に事前に SSH サーバを常駐させておく必要があることと、HTTP プロトコル用のプロキシサーバを経由して通信することが困難であるという問題点がある。

2.11 VPN

VPN は、インターネット等の公衆ネットワーク上に、本来専用線等のプライベートネットワーク上を流れるべき通信データをカプセル化して流す通信技術である。通信データをカプセル化するというのは、通信データを暗号化・電子署名し、その通信データがカプセル化されたものである旨を示すヘッダを付加するという作業である。この通信を、トンネリングとも呼ぶ。ネットワーク上で 2 点間通信を行うとき、データの送信側がカプセル化を行い、受信側がカプセル化を解除すれば、その 2 点間だけで安全で盗聴困難なデータ通信が可能になる (図 8)。

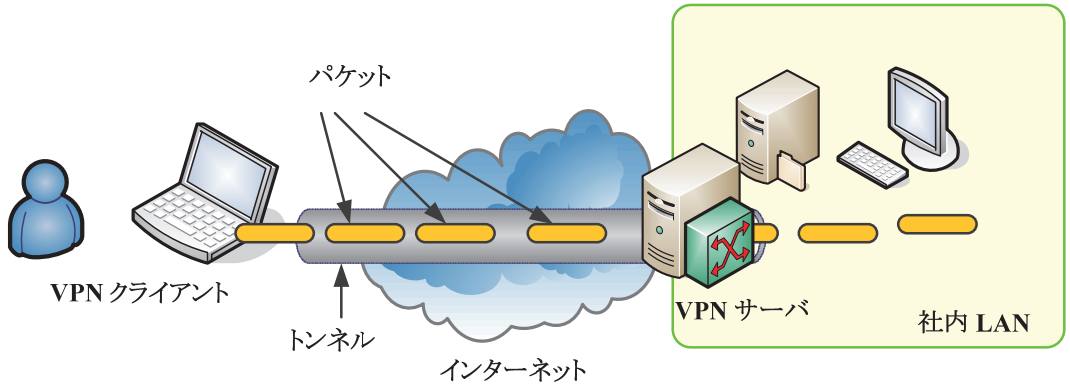

図 8 VPN によるトンネリング通信

VPN にはレイヤ 2 やレイヤ 3 をカプセル化の対象とするネットワーク型のものと、それ以上のレイヤをカプセル化するアプリケーション型のものがある。ネットワーク型の VPN を利用する場合は、その VPN がサポートしているレイヤより上のプロトコルはすべて VPN 経由で通信することができる。アプリケーション型の VPN を利用する場合は、その VPN がサポートしているアプリケーションのみを VPN 経由で通信させることができる。ネットワーク型の VPN としては、IPsec [7] や PPTP [8]、著者が開発した SoftEther [9] や PacketiX VPN [10] 等がある。アプリケーション型の VPN としては、一般的に SSL-VPN と呼ばれている技術がある。

NAT やプロキシサーバ等を経由して通信できるネットワーク型の VPN を利用することにより、これまでに述べたような NAT やプロキシサーバ等の内側にあるコンピュータに対して、グローバル上のコンピュータから TCP の接続要求を行うことができないという問題を解決することができる。しかし、ネットワーク型の VPN を利用するためには、ネットワーク上に VPN ブリッジやゲートウェイと呼ばれるコンピュータを設置するか、もしくは VPN 通信を行おうとするコンピュータそれぞれに VPN クライアントをインストールする必要が

ある。VPN クライアントのインストールおよび起動にはシステム管理者権限が必要であり、常駐ソフトウェアが管理者権限で動作するということはセキュリティ上問題となる場合がある。

2.12 P2P

P2P (Peer-to-peer) とは、ネットワーク上に固定されたサーバを持たず、ネットワーク上の各コンピュータがサーバおよびクライアントの両方の機能を有するようなコンピュータの集合によって構成される通信上の形態のことである。クライアント/サーバ方式と比較した場合、P2P 方式の利点は、クライアント数が膨大に増加しても、特定台数のサーバにクライアントすべてが依存しないため、サーバが高負荷となりシステム全体が破綻することがなく、スケーラビリティが保てるという点にある。また、P2P のうち、NAT を経由して通信を行うために特化した機能を持つプロトコルは、NAT の内側にあるコンピュータも、グローバル上のコンピュータと対等に通信を行うことができ、これによって NAT 越えを実現することができる。例として、Skype [11] という IP 電話ソフトウェアがある。

P2P の欠点として、安定した P2P のソフトウェアの実装は技術的難易度が高いため、開発コストがかかるという問題点がある。また、中継ノードとなるコンピュータも一般のユーザのコンピュータであるため、突然中継ノードが停止したり、ネットワークから切断されてしまったりする可能性があり、他の中継ノードを経由するように通信経路が再構成されるまでに時間がかかる場合があることもある。さらに、P2P で NAT 越えを実現しようとする場合、別々の NAT の内側のコンピュータ同士の通信を中継する場合は、グローバル上のコンピュータ動作が通信する場合と比較して、経由しなければならない中継ノードの数が増え、またネットワークトポロジが複雑になる。これにより、通信スループットや遅延などが、NAT の内側のコンピュータとグローバル上のコンピュータの間で異なるという現象が生じることになり、NAT の内側か否かによりユーザが得られる体感速度やサービス品質に差が生じてしまう。

加えて、特別な管理用のサーバが無いピュア P2P の形態では、通信しようとしている相手のコンピュータが意図したコンピュータ本体であるか、もしくは成り済まされた別のコンピュータであるかといった本人性格人のための検証を、事前に公開鍵やそのハッシュ等を知っておくか、信頼できる証明機関によって相手が提示した証明書が署名されているかどうかを確認する等の方法を使用しない限り困難である。同時に、P2P のプロトコルに弱点がある場合は、正規の P2P ソフトウェアの振りをして、虚偽の制御情報を流す等により、P2P ネットワーク全体を混乱させることができるという可能性が指摘されており、プロトコルが公開され、長年に渡る運用実績がなければ、その安定性を検証することは困難である。

2.13 問題点のまとめ

ここまでで挙げた従来技術およびそれに関する問題点をまとめると、以下のようになる。

まず、インターネットに接続されているコンピュータ間で通信を行おうとしたとき、コンピュータのうちすくなくとも片方が NAT やプロキシサーバ、ファイアウォール等 (以下

NAT 等と表現する) の内側にある場合は、NAT 越え等の方法でそれらを経由して通信を行う必要がある。NAT 越え等を行うための手法として、ポートマッピングや UPnP、プロキシサーバのリバースプロキシの使用、例外ルールの追加、UDP ホール・パンチング、TCP フォーウェイハンドシェイクなどは、対応していない NAT 等があったり、設定方法などが NAT 等の実装によって異なったり、設定を行う際ことがセキュリティを低下させる原因となる可能性があったりするので、汎用的に使用することができる手法ではない。

SSH サーバや VPN ソフトウェア等を NAT 等の内側に設置しておく方法は、NAT 等の設定を変更してグローバル上からの SSH 等の接続要求に内部のコンピュータが応答できるようにしておくか、もしくは VPN クライアンソフトウェアを NAT 等の内側のコンピュータで常駐させておく必要がある。前者は NAT 等の設定変更作業を必要とする点で、前段で述べた方法と同一の問題点があり、後者は NAT の内側のコンピュータで管理者権限によって常駐するソフトウェアが必要になるので、セキュリティ上安全とはいえない場合がある。

P2P 技術を用いて NAT 越えを実現する方法は、安定した P2P ソフトウェアの実装は技術的簡易度が高く、また、通信プロトコルに相当な工夫をしなければ、途中の中継ノードの役割を担っているコンピュータが停止したりネットワークが切断されたりした場合に通信が不安定となるという問題点がある。さらに各コンピュータが中継ノードになり得るため、プロトコルの抱える弱点を突かれて、虚偽の制御情報を流す等により P2P ネットワークが混乱し通信が不安定になる可能性もある。

以上により、インターネット上で NAT 等の内側のコンピュータとグローバル上のコンピュータ、もしくは別々の NAT 等の内側にある複数のコンピュータ同士が、いずれを TCP コネクション接続要求の発信元としなければならないかという制約を受けずに TCP コネクションまたはそれと同等の双方向ストリーム型コネクションを確立し、安定して自由に通信することを必要とする場合、それを実現するためのいくつかの技術にはそれぞれ問題点があり、様々なネットワーク環境で、汎用的に利用することができる手段が存在しないことがわかった。そこで、本研究の目標としては、これらの問題点を解決する新しい通信システムの実現を目指すこととした。

第 3 章 要件および設計

本章では、本研究において実現する新しい通信システムの要件を定義し、全体の設計および各モジュールの詳細な設計を述べる。

3.1 通信に関する要件

本通信システムによって利用できるようにする通信機能の要件は、次のとおりである。

- コネクション型通信
- 双方向ストリーム通信
- ソケット API との互換
- 同期モードおよび非同期モード
- 通信相手の識別
- ゲートウェイシステムによる中継
- 暗号化および電子署名
- NAT 越え
- プログラミングの容易性

以下にこれらの項目について詳細に述べる。

3.1.1 コネクション型通信

本通信システムを利用して行うことができる通信の形態は、コネクション型とする。これは、インターネット上で利用されている既存のコネクション型のアプリケーションの数が多く、これらのアプリケーションを、本通信システムを利用できるように書き換えることを容易に可能とするためである。

通信を行おうとする 2 台のコンピュータ間でコネクション型通信を行うことができるということは、どちらかのコンピュータのアプリケーションが接続元、もう一方のコンピュータのアプリケーションが接続先となり、接続先が予め新規コネクションを待ち受けている状態で、接続元が接続先に対してコネクションの接続要求を行うと、接続先がこの要求を受信し、それを受諾することにより、2 点のアプリケーション間で 1 本の新しいコネクションが確立され、以後このコネクションを用いて通信ができ、また、これ以上通信を行う必要が無いという状態になった場合は、いつでもどちらかの側からコネクションを切断することができるというものである (図 9)。

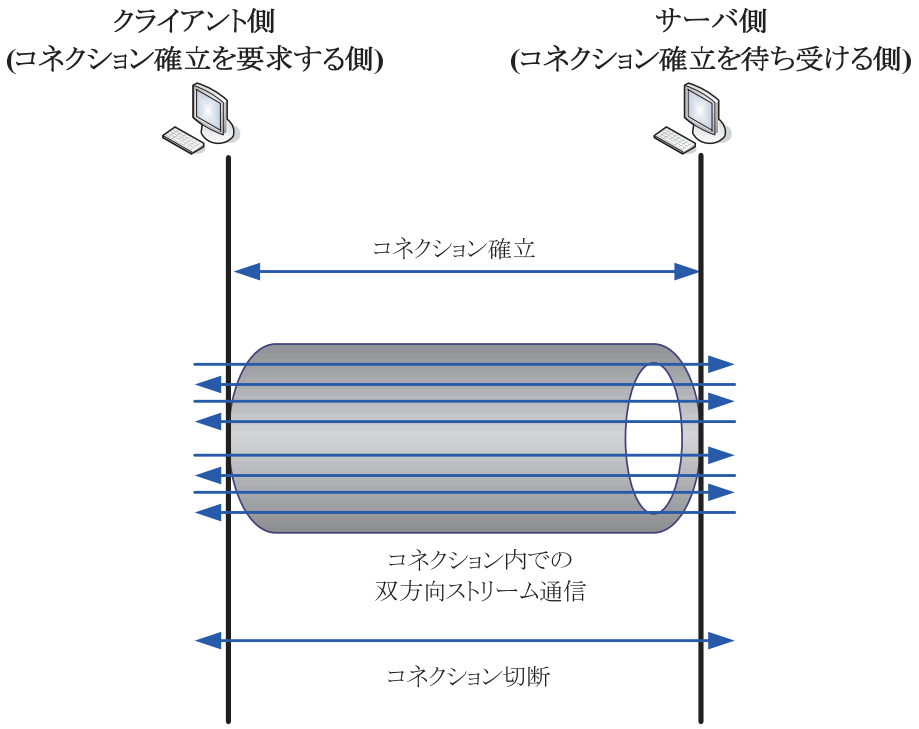

図 9 コネクション型通信の概要

3.1.2 双方向ストリーム通信

コネクションを確立した後は、コネクションの両端を保有しているアプリケーション間で自由に通信を行うことができるべきであるが、その通信形態は、双方向であって、また任意のストリームを送受信することができるということにした。双方向とは、1 本のコネクションに対して、片側からそれを見たときに、送信方向のデータ伝送と受信方向のデータ伝送が同時に行えるという意味である。任意のストリームの送受信ができるとは、送信側がコネクションを用いて送信しようとしたデータのビット列が、送信しようとした順番にネットワーク上を伝送され、受信側にその順番で正確に届くという意味である。これらは、TCP/IP における TCP コネクションの性質と同一である。

3.1.3 ソケット API との互換

既存の通信アプリケーションのうち、ソケット API [12] を用いてプログラミングされているものを容易に書き換えて本通信システムに対応することができるようにするため、本通信システムがアプリケーション開発者に対して提供する通信のための API は、可能な限り、ソケット API と同等のインターフェイスを持つものとする。ソケット API の概要を、図 10 に示す。

本通信システムを利用するアプリケーションを開発するために使用できる API を用いたプログラミングは、できる限り容易に行うことができるようにするため、API を呼び出す側が特殊なデータ構造を取り扱ったり、複雑な処理を行ったりしなくても良いようにすることとする。

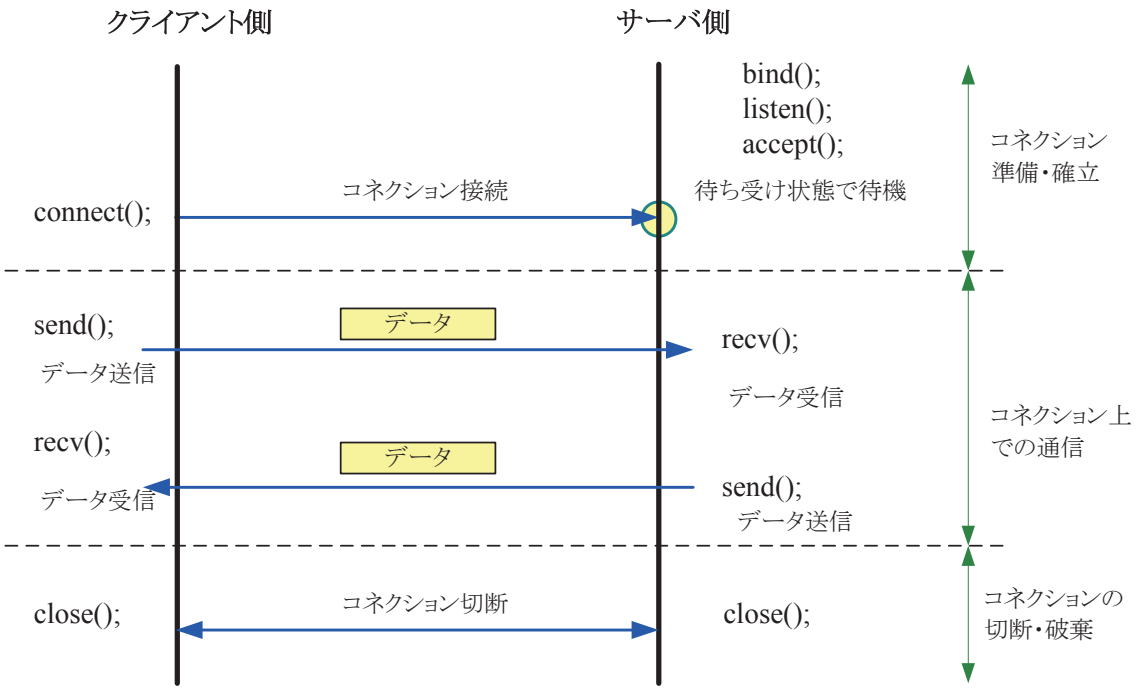

図 10 ソケット API による通信の手順

3.1.4 同期モードおよび非同期モード

アプリケーション開発者が本通信システムの API を呼び出して通信を行おうとする場合、データの送受信においては、ソケット API において同期ソケットと非同期ソケットの 2 種類があることと同様に、同期モードと非同期モードの 2 種類のモードを利用してデータの送受信を行うことができるようにする。

これにより、1 つのスレッドで同時に複数のコネクションを管理し、効率的に通信を行うことができるプログラムを、アプリケーション開発者が簡単にプログラミングできる。

3.1.5 通信相手の識別

接続元 (コネクションの接続および確立を要求する側) が接続先 (コネクションの接続および確立を受諾する側) に対して接続要求を行うためには、接続先のコンピュータ内のアプリケーションを識別するためのアドレスが必要である。インターネット上ではこのためのアドレスとして IP アドレスおよびポート番号が使用されるが、これらは覚えにくく、また、1.1 等で述べたように、インターネット上でコンピュータを識別するためのグローバル IP アドレスが利用できない場合も多いことから、IP アドレスを使用することはできない。

本通信システムでは、コンピュータを識別するための識別子として、任意の文字列を使用できることとし、この任意の文字列を、接続先のコンピュータで予めユーザが指定することができるようにした。これにより、接続先のアプリケーションを起動したユーザは、自らが覚えやすく、他者と重複しない任意の文字列を識別子として利用することができるようになる。

3.1.6 ゲートウェイシステムによる中継

本通信システムは、すべてのコネクションの通信を、インターネット上に設置したゲートウェイシステムを経由して行う (図 11)。物理的な通信 (実際にユーザのコンピュータとインターネットとの間で流れる通信) は TCP コネクションとし、ゲートウェイシステム側で待ち受けている TCP ポートに対して、各アプリケーション側から TCP 接続する。その後、各アプリケーションとゲートウェイシステムとの間で確立された TCP コネクションの内部に論理的な仮想通信路 (トンネル) を構成する。この論理的な仮想通信路を**セッション**と呼ぶ。

コネクションを接続しようとする側は、新しいコネクションを確立しようとする都度、ゲートウェイシステムに対して、接続先のアプリケーションの識別子を指定し、新しいコネクションを確立したい旨の要求を送信する。ゲートウェイシステムは接続先のアプリケーションの識別子を検索し、その識別子を持っているアプリケーションのセッションを特定する。そして、接続元とのセッションおよび接続先とのセッションの内部に 1 本の新しい論理的なコネクションを確立し、以後その論理的なコネクションを用いて接続元と接続先との間での自由な通信が行えるようになる。

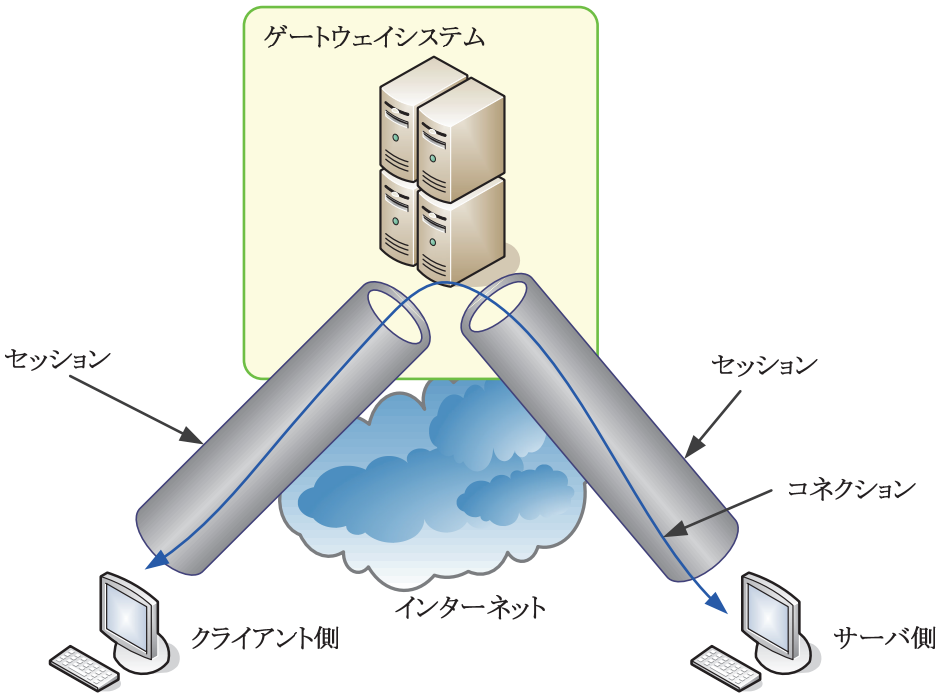

図 11 ゲートウェイシステムによるすべてのコネクションの中継

3.1.7 暗号化および電子署名

本通信システムはインターネット上の 2 台のコンピュータ間で通信を行えるようにするためのものである。インターネット回線上の通信は、常に盗聴されている危険性があるという前提で扱うのが望ましい。そこで、本通信システムによる通信データの保護のため、

すべての通信データは暗号化アルゴリズムによって自動的に暗号化され、かつ電子署名が付加され、通信内容を第三者が盗聴したり、書き換えたりすることができないようにする必要がある。

本通信システムでは、まず、コネクションを受け付ける側のアプリケーション、コネクションを接続する側のアプリケーションの双方が、経由するゲートウェイシステムが本物であるかどうかの正当性検証を PKI によって行い、その後、コネクションを接続する側から見て、コネクションを受け付ける側が本物であるかどうかの正当性検証を、経由するゲートウェイシステムが本物であるということが前提で検証するという手順を実施するようにする。また、同時にすべてのセッションにおいて、セッション内を流れるすべてのデータが暗号化アルゴリズムおよび電子署名アルゴリズムによって暗号化・電子署名されるようにする。

3.1.8 NAT 越え

本通信システムでは、1.1 等で述べたように、現状の多くの環境で NAT やプロキシサーバ、ファイアウォールなどの、内側から外側に対する通信のみを通過させることができる中継装置が設置されていることを鑑み、これらの中継装置を経由して、アプリケーションからゲートウェイシステムまでの間のセッションを確立することができるように工夫することにする。

一度、アプリケーションとゲートウェイシステムとの間のセッションが確立した後は、そのセッションの内側で論理的なコネクションを確立し任意のストリームデータを送受信することは支障無く可能となる。

3.2 スケーラビリティに関する要件

本システムの物理的な通信経路においては、すべての通信はゲートウェイシステムを経由して行われるため、全通信の負荷がゲートウェイシステムに集中することとなる。従って、多くのユーザが同時に本システムを利用してもシステムが破綻しないようにするため、スケーラビリティが重要である。サーバサービスを提供するコンピュータは物理的な CPU やメモリの処理速度の上限があり、1 台のサーバコンピュータで処理することができるスループット以上のリクエストを処理することができない。

スケーラビリティを実現するための本システムにおける要件は、次のとおりである。

- 負荷分散
- 障害検出
- 動的なゲートウェイの増減
- ゲートウェイのインターネット上への分散配置

以下にこれらの項目について詳細に述べる。

3.2.1 負荷分散

クライアント/サーバ型システムでスケーラビリティを向上するための方法の一つとして、一度サービスの提供を開始した後、利用ユーザ数等の増加にあわせてサーバの台数を増加していき、クライアントからのリクエストを各サーバに負荷分散する方法がある。負荷分散とは、サーバコンピュータを複数台設置しておき、クライアントからのリクエストを、予め定めておいたアルゴリズムによって、各サーバに対して転送する方法である。これにより、他のオーバーヘッドが無い場合は、1 台のサーバコンピュータで処理することができるスループットにサーバ台数を乗じたスループットをシステム全体で処理することができる。

本システムのゲートウェイシステムでは、ユーザ側アプリケーション間の通信処理を中継する役割を持つゲートウェイ機能を提供するサーバコンピュータ (以下**ゲートウェイ**という) を複数台設置することができるようにし、クライアントからのリクエストを、それらのゲートウェイに対して分散して転送する (図 12)。

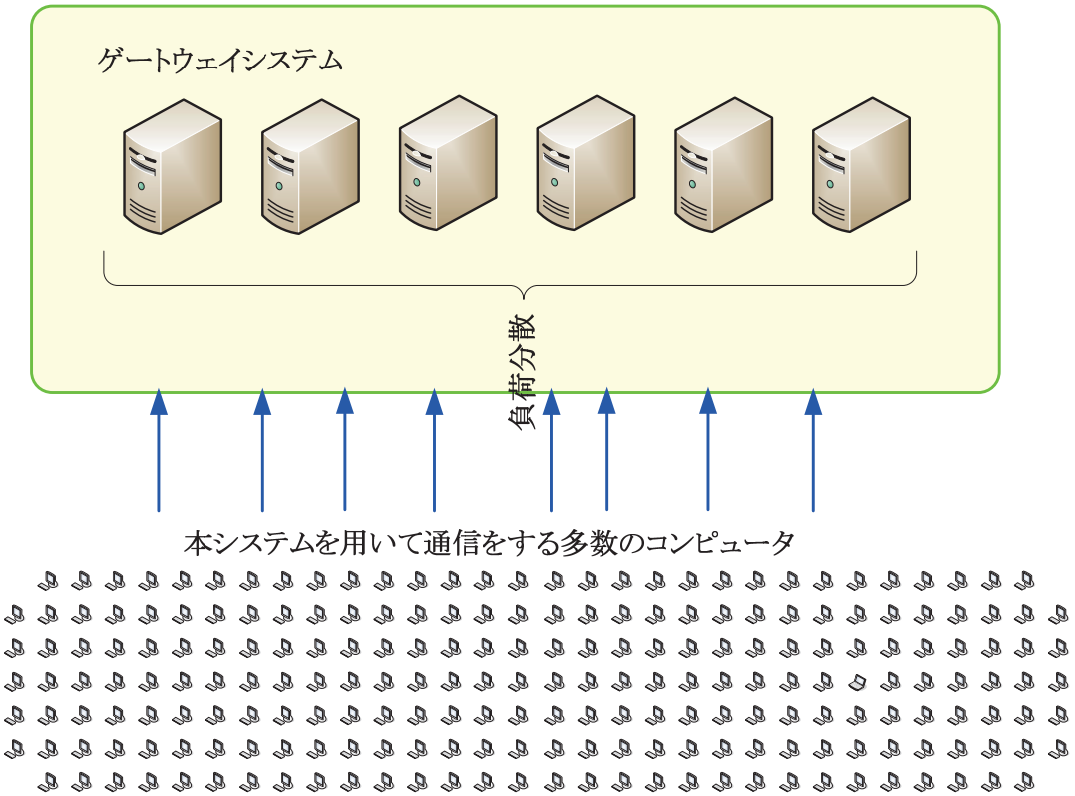

図 12 複数台のゲートウェイによる負荷分散処理

3.2.2 障害検出

本システムにおけるゲートウェイへの負荷分散においては、ゲートウェイ停止時の障害検出が重要である。各ゲートウェイは物理的には汎用的で安価なサーバコンピュータによって稼動させることを想定すると、各ゲートウェイが突然物理的に故障する可能性がある。このような場合は、負荷分散を行うコンピュータであるロードバランサが直ちに当該ゲー

トウェイの停止を検出し、以降のリクエストは、停止していない別のゲートウェイに対して転送するようにしなければならない。

また、本システムを利用して通信を行うアプリケーションのうち、コネクションを待ち受ける側のアプリケーションは、常にゲートウェイシステム内の 1 台のゲートウェイとの間でセッションを張っていることになる。この状態でセッションを張っている先のゲートウェイが停止したときは、自動的に別の正常稼動しているゲートウェイに接続し直さなければならない。

本システムでは、上記のような障害検出機能を実現する。

3.2.3 動的なゲートウェイの増減

予めゲートウェイシステムが処理することになるコネクションの総数や総スループットがわかっている場合は、それを処理できるだけのゲートウェイを設置しておけばよい。しかし、将来的にユーザ数が増加し、それに伴いコネクションの総数や総スループットが増加し続ける可能性がある場合は、必要な都度、新しいゲートウェイをゲートウェイシステムに追加する必要が生じる。

一度インターネット上でゲートウェイシステムの稼動を開始し、ユーザに対してゲートウェイサービスの提供を始めた後は、多数のユーザによるコネクションがゲートウェイシステムを経由して確立され通信が行われることになるため、ゲートウェイシステムを頻繁に停止しなければならない事態となることは望ましくない。そのため、ゲートウェイシステムに新しいゲートウェイを追加する必要があるときは、ゲートウェイシステム全体を停止させることなく、新しいゲートウェイを追加することができることが必要である。メンテナンス等のために、1 台のゲートウェイをシステムから削除する際も、同等である。

本システムでは、ゲートウェイシステムの稼働中における、システム全体を停止させることのない動的なゲートウェイの増減を実現する。

3.2.4 ゲートウェイのインターネット上への分散配置

ゲートウェイシステムはロードバランサと複数台のゲートウェイとによって構成されるが、それぞれを稼動させている物理的なサーバコンピュータは、グローバル IP アドレスを割り当てられた状態で、物理的回線によってインターネットに接続されていなければならない。しかしながら、ゲートウェイの台数が増加した場合、各サーバコンピュータを 1 本のインターネット接続回線に収容することができなくなることが考えられる。たとえば、ISP によって同時に 1 回線契約に割り当てられるグローバル IP アドレスの数に上限がある場合で、それを超える数のゲートウェイを設置したい場合などである。また、例えば物理的に 1 箇所の拠点にすべてのゲートウェイを設置することができなくなる可能性もある。たとえば、1 拠点における電力不足や空間不足が発生した場合などである。

このような場合に対応するため、本システムでは、ゲートウェイシステムを構成する各サーバコンピュータは、物理的に分散した拠点に配置することができるようにするとともに、同時にインターネット上に分散配置することができるようにする。すなわち、ロードバランサを稼動させるサーバコンピュータと、ゲートウェイを稼動させるサーバコンピュータ

は、物理的には全く別の場所にあり、全く別の ISP によってグローバル IP アドレスを割り当てられていても良いものとする。

ゲートウェイのインターネット上への分散配置の例を、図 13 に示す。

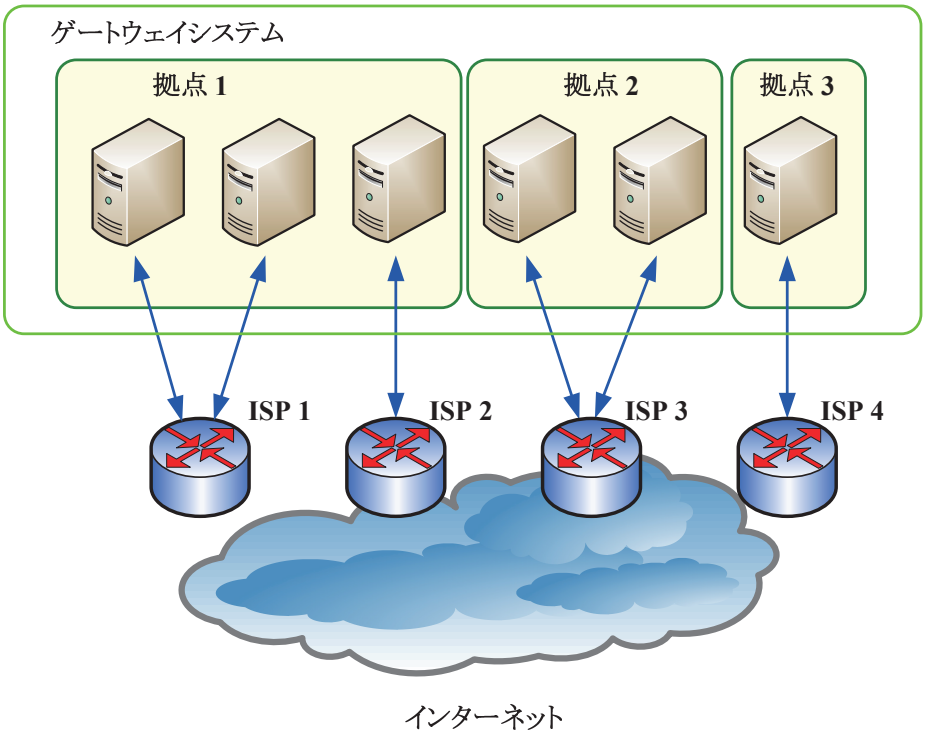

図 13 ゲートウェイのインターネット上への分散配置の例

3.3 全体の設計

具体的なゲートウェイシステムおよび通信アプリケーション側で利用する通信ライブラリの設計を行うために、まず、モジュールの名称および機能を定義し、それぞれのモジュール間の関係を決定する。

3.3.1 モジュール一覧

本システムは、4 種類のモジュールに分類される。モジュール一覧を、表 1 に示す。

表 1 本システムを構成するモジュールの一覧

分類	モジュール名
ゲートウェイシステム側モジュール (中継システム)	コントローラ
	ゲートウェイ
アプリケーション側モジュール (中継システムを利用するための通信 API)	サーバ
	クライアント

まず、本システムの構成要素を大きく二分し、ゲートウェイシステム (ストリーム中継システム) と、そのゲートウェイシステムを利用するための通信 API 等を提供する通信アプリケーション用ライブラリとする。

ゲートウェイシステムは、3.1 で述べたように、ロードバランサとゲートウェイの 2 種類のモジュールによって構成される。なお、ロードバランサのことを**コントローラ**と呼ぶ。

ゲートウェイシステムを利用する通信アプリケーションは、3.1 で述べたように、**コネクションを待ち受ける側**と**コネクションを接続する側**のように、二分することができるため、それぞれのためのライブラリとして、サーバ側およびクライアント側の 2 種類のモジュールによって構成される。

各モジュールおよびモジュール間の関係を、図 14 に図示する。

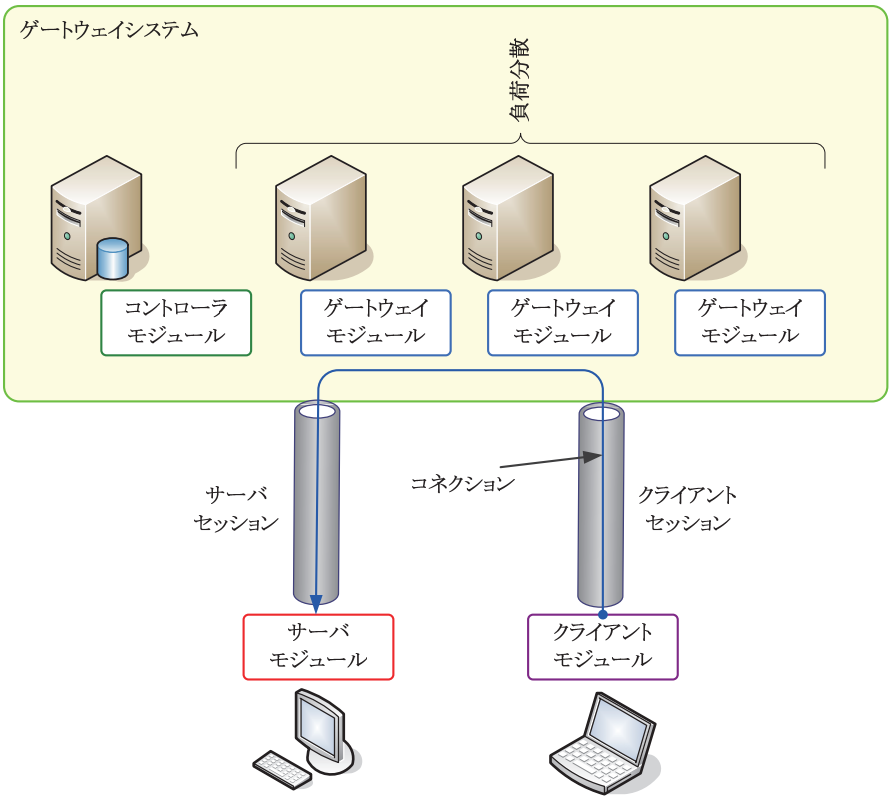

図 14 本システムを構成する 4 種類のモジュール

3.3.2 コントローラ

コントローラは、クライアントもしくはサーバによるゲートウェイシステムへの接続要求を処理し、予め設定されている負荷分散アルゴリズムに基づき、その接続要求を処理させるべきゲートウェイにリダイレクトする処理を行うプログラムである。コントローラの役割を、図 15 に示す。

3.2 で述べたように、コントローラは、現在正常稼動しているゲートウェイの一覧を常に保持しており、また、一部のゲートウェイとの間の障害を自動検出し、そのゲートウェイを負荷分散の対象から除外する処理を行う。また、3.1 で述べたように、本システムではクライアントからサーバに接続する際のサーバを識別するための識別子として、予めユーザが指定した任意の文字列を使用することができるようにするため、コントローラはこれまでに登録された識別子の一覧と、現在ゲートウェイシステムに対してセッションを確立しているサーバとの間の各セッションが対応している識別子の一覧のテーブルを持つ。

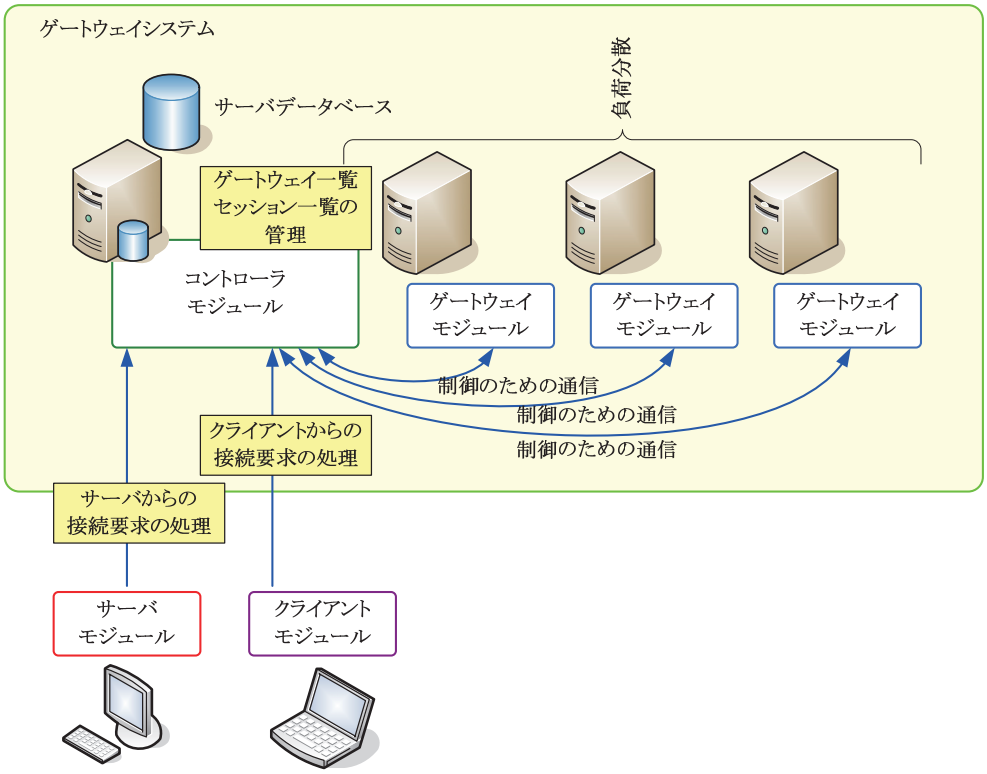

図 15 コントローラの役割

3.3.3 ゲートウェイ

ゲートウェイは、本システムにおいてサーバとクライアントとの間のコネクションにおけるストリーム中継を行う役割を担うプログラムである。サーバはまずコントローラにアクセスし、接続先となるべきゲートウェイを指定してもらい、次にそのゲートウェイに接続する。一度サーバがゲートウェイとの間のセッションを確立した後は、ゲートウェイが停止するか、インターネット回線上でエラーが発生するか、サーバが停止するかしない限り、

そのセッションは維持される。この状態で、クライアントからのコネクションの接続要求があれば、ゲートウェイを経由してその接続要求がサーバに伝送され、コネクションが確立される。ゲートウェイの動作の概要を、図 16 に示す。

ゲートウェイはサーバから自らに対して確立されているセッションの一覧をコントローラに対して報告することにより、コントローラは、現在どのサーバがどのゲートウェイに接続しているかという情報を把握することができるようになっている。

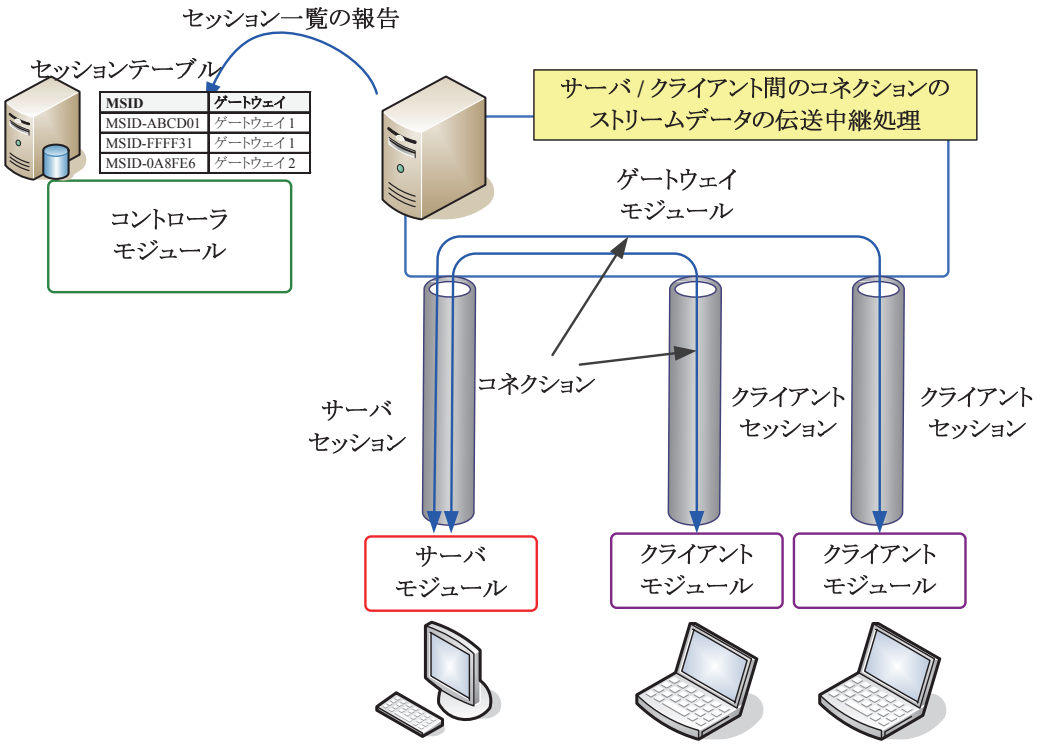

図 16 ゲートウェイの動作

3.3.4 サーバ

サーバは、本システムを経由して通信を行おうとする通信アプリケーションのうち、コネクションの接続を待ち受ける側のプログラム内で動作する、本システムを利用するための通信 API によって操作可能なライブラリである。開発者はサーバのライブラリが提供する通信 API を用いて通信アプリケーションを開発することができる。

サーバが起動すると、まずコントローラへ接続する。コントローラは、そのサーバが接続すべきゲートウェイを指定し、リダイレクトするように指示する。サーバは指示されたリダイレクト先のゲートウェイに接続し、セッションを確立する。サーバ機能をシャットダウンするまでの間は、常にゲートウェイとの間のセッションを確立し続ける。

ゲートウェイの停止やインターネット上の通信エラーなどで、ゲートウェイとの間のセッションが切断された場合は、サーバは再度コントローラに接続し、前述と同様の動作を行う。これを繰り返すので、コントローラおよび接続可能なゲートウェイが稼動している限り、常にサーバはゲートウェイシステムと接続され続けることになる。

3.3.5 クライアント

クライアントは、本システムを経由して通信を行おうとする通信アプリケーションのうち、コネクションの接続を行う側のプログラム内で動作する、本システムを利用するための通信 API によって操作可能なライブラリである。

クライアントはサーバとの間で通信を行うために、コネクションを確立する必要がある。コネクションの確立のためには、まずクライアントがコントローラに接続し、希望する接続先のサーバの識別子を指定する。コントローラは、指定された識別子を持つサーバが現在いずれかのゲートウェイに接続されているかどうかを調べ、いずれかのゲートウェイに接続されている場合は、そのゲートウェイに対して接続するための情報をクライアントに返す。クライアントはその情報を使用してゲートウェイに接続し、セッションを確立し、目的のサーバへのコネクションの接続を要求する。

コネクションが接続された後は、クライアントはサーバとの間で任意のストリームデータの送受信が可能になる。

3.4 ゲートウェイシステム

ゲートウェイシステムの全体およびそれを構成するコントローラとゲートウェイの間のデータのやりとりの設計について述べる。

3.4.1 疎結合クラスタ

ゲートウェイシステムでは、1 台のコントローラ、および複数台のゲートウェイの間で、疎結合クラスタが組まれて動作する。疎結合クラスタとは、各ノード間で共有メモリや共有ディスクを持たない、コンピュータ群のことを指す。

コントローラと各ゲートウェイでは、共有メモリやデータベース等を持たず、必要最小限の制御情報の通信のみが行われるだけである。これにより、コントローラと各ゲートウェイは物理的に離れた場所に分散して設置されていても支障が無い。

3.4.2 負荷分散アルゴリズム

コントローラは、新しいサーバがゲートウェイシステムに接続しようとした際に、セッションを確立させるゲートウェイを選択する作業を行う。選択は、予めプログラミングされた負荷分散アルゴリズムによって実行される。

本研究において実装したコントローラでは、以下のような単純なアルゴリズムを使用する。

1. 各ゲートウェイは、ハードウェアの**性能基準値**と呼ばれる、自らが処理できる同時接続数の目安となる数値 (他ゲートウェイと比較した際の相対的な能力値) をパラメータとして持っている。このパラメータは、ゲートウェイシステムの管理者が設

定する。

2. コントローラは、ゲートウェイの一覧と、各ゲートウェイに対して現在サーバから接続されているセッション数を把握している。
3. コントローラは、各ゲートウェイにかかっている負荷を、セッション数 ÷ 性能基準値 で算出する。
4. 新しいサーバがゲートウェイシステムに対して接続しようとすると、コントローラは各ゲートウェイのうち、3 で算出した負荷値が最も低いものを選択する。

上記のアルゴリズムを使用すると、各ゲートウェイのハードウェア性能が同一である場合は、各ゲートウェイの中から、最もサーバからのセッション数が少ないものを選択することになる。実運用上は、各ゲートウェイにかかっている負荷状況は、必ずしもセッション数のみで表されるものでなく、各ゲートウェイがその瞬間に処理しているスループットと、確立されているセッション数、およびセッションの内部で確立されているコネクション数、および各ゲートウェイの CPU 使用率やメモリ空き容量などといったいくつかのパラメータを元に数式化して判断するべきである。しかしながら、これらのパラメータから正確な負荷状況を算出するための数式を導出することは難易度が高いので、本研究では見送り、今後の課題とすることにし、上記のような単純なアルゴリズムを使用すること、および当該アルゴリズム部分は後に容易に置換可能とする。

3.4.3 ゲートウェイ登録

ゲートウェイは、継続して稼動するサーバコンピュータ上で動作する常駐サーバプログラムである。ゲートウェイのプログラムがサーバ上で開始されると、予め管理者が設定しておいたコントローラの IP アドレスに対して通信を行い、自らがインターネット上で動作を開始したことをコントローラに対して登録する。これにより、コントローラの持つゲートウェイの一覧に新しいゲートウェイが登録される。

3.4.4 キープアライブ通信

ゲートウェイは、自らが正常に動作しており、新しいサーバからのセッションの接続要求の受け入れが可能である旨を、管理者によって設定された時間 (T1) ごとにコントローラに対して通知する。これをキープアライブ通信と呼ぶ。

コントローラは、管理者によって設定された時間 (T2, ただし T1 < T2) の間に 1 度もゲートウェイからのキープアライブ通信を受信することができない場合は、そのゲートウェイは停止したものとみなし、ゲートウェイの一覧から削除するとともに、負荷分散先としてそのゲートウェイを選択しなくなる。

キープアライブ通信の動作を、図 17 に示す。

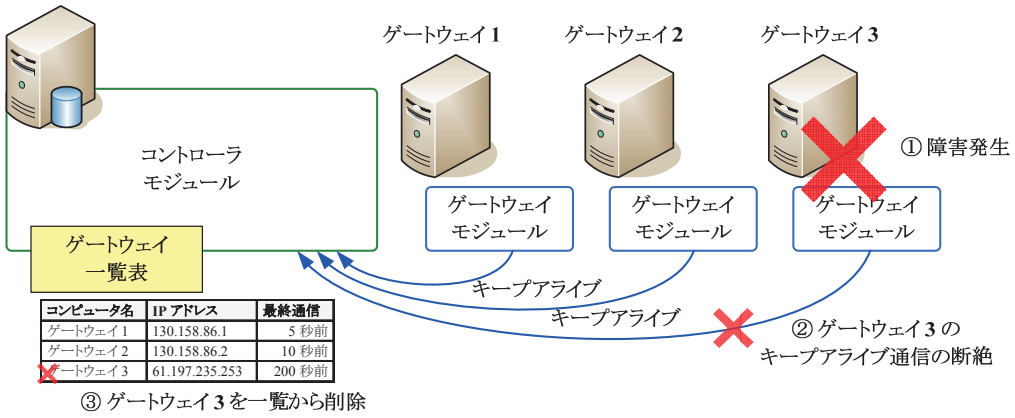

図 17 ゲートウェイの生存確認のためのキープアライブ通信

3.4.5 セッション管理

サーバがコントローラに対してセッション確立要求を行うと、コントローラがそのセッションを受け入れるべきゲートウェイを指定し、サーバがそのゲートウェイに接続すると、ゲートウェイ上で新しいセッションが開始されることになる (このセッションを、以下**サーバセッション**と呼ぶ)。

ゲートウェイは、自らが処理しているサーバセッションの一覧に変化があったとき、その差分データをコントローラに対して通知する。コントローラは、メモリ上に、どのゲートウェイ上にどのサーバからのサーバセッションが接続されているかという**セッションテーブル**を保持しており、ゲートウェイからの通知に基づき、このセッションテーブルを最新の状態に更新しておく。

これにより、クライアントがコントローラに対して接続してきて、コネクションを確立したい接続先のサーバの識別子の指定があった場合は、セッションテーブルから、その接続先のサーバが接続されているサーバセッションがどのゲートウェイ上に存在するかを直ちに検索することができる。

このセッションデータベースは、携帯・自動車電話システムにおいて、待ち受け状態の携帯電話等が現在カバーされている基地局の ID と、その携帯電話等の識別子を関連付ける、ロケーションレジスタのような役目を果たす [13]。

3.4.6 サーバデータベース

コントローラは、メモリ上にゲートウェイの一覧およびセッションテーブルを保持しているだけでなく、ディスク上にデータベースとしてサーバデータベースを保持している。

サーバデータベース内のテーブルには、後述する、各サーバの保持している秘密鍵に対応する公開鍵の SHA-1 ハッシュデータと、ユーザフレンドリーなサーバ識別子 (PCID) および内部的なサーバ識別子 (MSID) との対応表が格納されている。これらの詳細については後述する。

3.5 サーバおよびクライアント

ゲートウェイシステムを経由して通信を行うアプリケーション側で使用されるサーバモジュールおよびクライアントモジュールの設計について述べる。

3.5.1 全体のシーケンス

サーバが新しいコネクションを待ち受けており、クライアントがサーバに対してコネクション確立要求をし、これによりコネクションが確立され、ストリーム通信が行えるようになるという一連の流れをより詳しく示すと、以下のようになる。

A. ゲートウェイシステムへのサーバの接続処理 (図 18 に図示)
   1. サーバは、独自に生成した秘密鍵、公開鍵およびその公開鍵を格納した自己署名証明書を持つ (初回起動時に自己生成し、ディスク等に記録しておく)。
   2. サーバがコントローラに対して、サーバセッションの確立要求を行う。この際、コントローラに対して証明書を提示する。
   3. コントローラは、提示されたサーバの証明書の SHA-1 ハッシュを計算し、その SHA-1 ハッシュから生成される内部識別子を生成する。この内部識別子を、MSID (Machine-System ID) と呼ぶ。サーバの証明書はサーバごとに異なるので、MSID はそのサーバ固有の値となる。同一サーバが複数回ゲートウェイシステムに接続しても、MSID は変化しない。
   4. コントローラは、サーバデータベースを検索し、MSID に対応するレコードが存在するかどうか調べる。レコードが存在しない場合は、新しいレコードを作成し、サーバ識別子を、乱数を用いて決定する。サーバ識別子を、ユーザにわかりやすいように**コンピュータ ID と**呼ぶ。以下では、コンピュータ ID を、実装時に用いた内部名である PCID (Personal-Computer ID) という。なお、このシーケンス外であるが、サーバはいつでも自らの PCID を取得したり、他のサーバと重複しない任意のものに変更したりすることができる。
   5. コントローラは、3.4.2 で述べた方法でゲートウェイを 1 台選択する。
   6. 選択したゲートウェイの IP アドレスおよび付加情報 (MSID が含まれる) をサーバに返す。
   7. サーバはコントローラとの間の接続を一旦切断してから、コントローラによって返されたゲートウェイの IP アドレスおよび付加情報を用いて、ゲートウェイに対して接続する。
   8. ゲートウェイは、サーバから渡された付加情報をもとに、サーバとの間で新しいセッションを確立する。
   9. ゲートウェイは、コントローラに対して、新しいサーバセッションが確立された旨を通知する。これにより、コントローラは当該ゲートウェイがそのサーバとの間のセッションを確立しているということを知ることができるようになる。なぜならば、ゲートウェイから通知されるサーバセッション情報には、そのサーバの

MSID が含まれるからである。この状態になると、新しいクライアントがこのサーバに対してコネクション接続を行うことができるようになる。

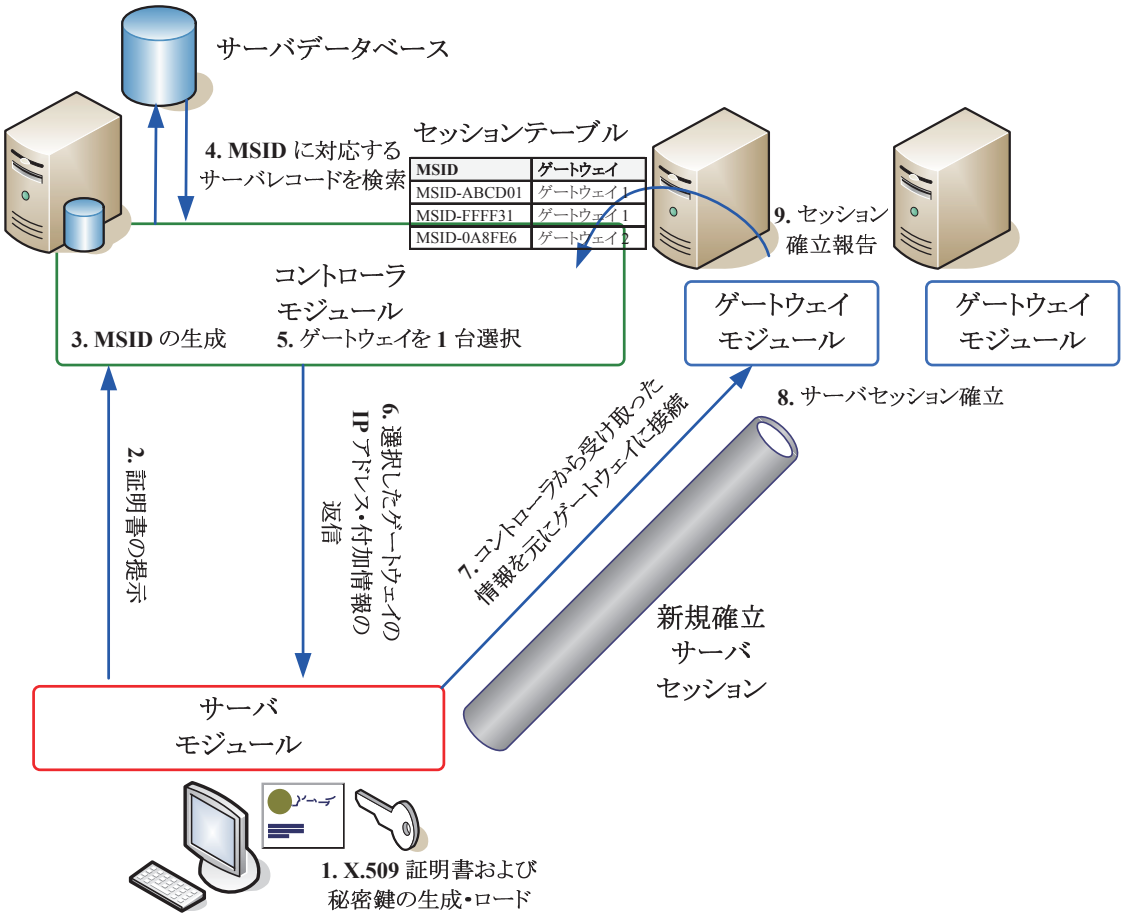

図 18 ゲートウェイシステムへのサーバの接続処理

B. ゲートウェイシステムへのクライアントの接続処理 (図 19 に図示)

1. クライアントがコントローラに対して、セッション (以下**クライアントセッション**という) の確立要求を行う。この際、コントローラに対して、接続したいサーバの PCID を指定する。
2. コントローラは、サーバデータベースを検索し、指定された PCID に対応するレコードがあるかどうか調べる。レコードが存在しない場合は、PCID が間違っている旨のエラーを返す。レコードが存在する場合は、そのレコードから MSID を取得する。
3. コントローラは、取得した MSID のサーバとの間で確立されているサーバセッションを、いずれかのゲートウェイが保持しているかどうかを、セッションテーブルを検索することによって確認する。
4. 検索の結果、いずれのゲートウェイもサーバセッションを保持していない場合は、指定された接続先は現在ゲートウェイシステムに接続されていない旨のエラーを返す。サーバセッションを保持しているゲートウェイが見つかった場合は、そ

のゲートウェイの IP アドレスおよび付加情報 (MSID が含まれる) をクライアントに対して返す。

5. クライアントは、受け取ったゲートウェイの IP アドレスを使用して、ゲートウェイに対して接続し、コントローラより受け取った付加情報を渡して、サーバとの間のコネクションの確立を要求する。
6. ゲートウェイは、クライアントより受け取った付加情報に含まれている MSID に対応するサーバセッションが接続されているかどうかを確認し、接続されている場合は、クライアントとの間にクライアントセッションを確立してから、クライアントセッションをサーバセッションに関連付け、そのセッションの内部で新しいコネクションを確立する。
7. クライアントとサーバの間で確立されたコネクションを用いて、自由なストリームデータの送受信が可能となり、本通信システムの目的が達成される。

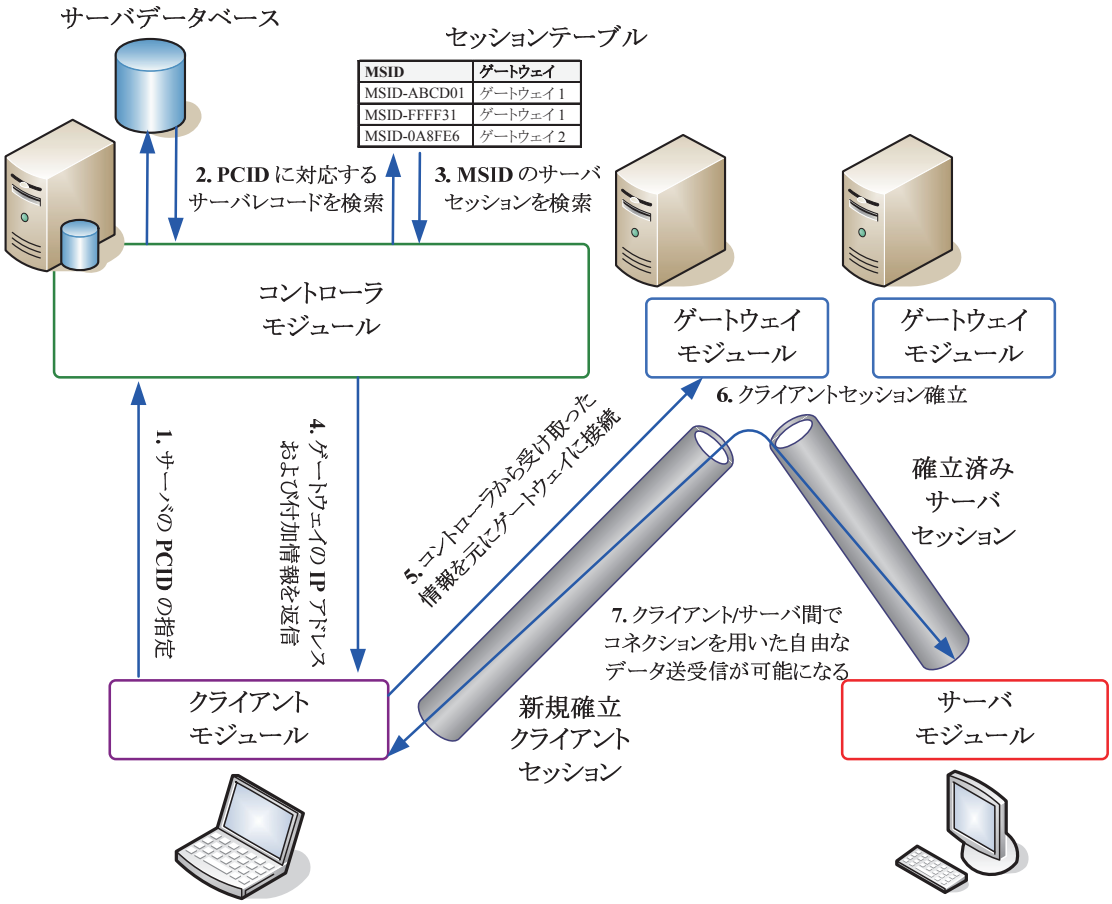

図 19 ゲートウェイシステムへのクライアントの接続処理

3.5.2 MSID と PCID

サーバを識別するための識別子として、MSID (Machine-System ID) および PCID (Personal-Computer ID) を用いる。

MSID はゲートウェイシステムの内部で用いられる識別子である。プログラム内部でのみ使用され、ユーザには表示されない。本システムにおけるサーバを利用する通信アプリケーションがコンピュータで初めて起動されたときに、サーバモジュールは、まず RSA 1024 bit の秘密鍵および公開鍵を生成する。次に、それを用いて自己署名証明書を作成する。この自己署名証明書は X.509 形式であり、証明書のデータから SHA-1 ハッシュを計算することができるようになっている。サーバがゲートウェイシステムのコントローラにアクセスする際、この証明書を提示する。コントローラは受け取った証明書の SHA-1 ハッシュを計算し、これを 16 進数で表示したものを用いて MSID を決定する。たとえば、証明書の SHA-1 ハッシュが **1D3BBE170F621E9A193296FACA589A0D783380F1** のとき、MSID は **MSID-1D3BBE170F621E9A193296FACA589A0D783380F1** となる (図 20)。

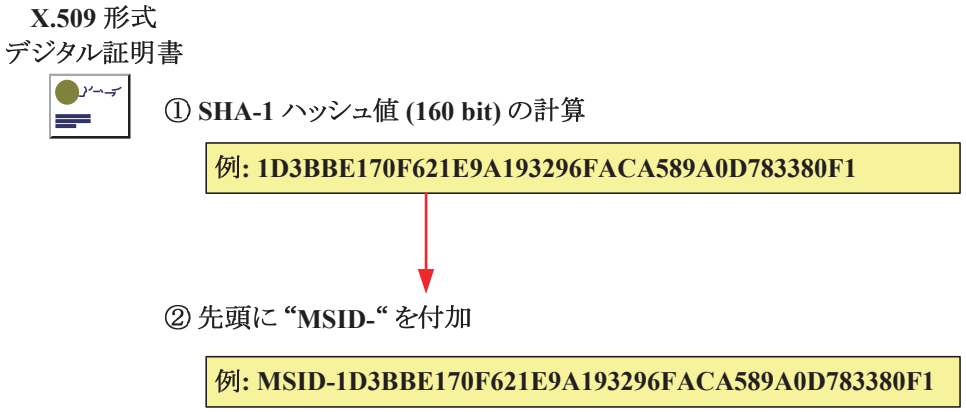

図 20 証明書のハッシュから MSID を生成する方法

PCID はインターネット上に存在し、現在ゲートウェイシステムにサーバセッションが接続されているサーバプリケーションのインスタンスを、ユーザが識別するために、任意の 31 文字以下の半角英数字および記号で構成される文字列である。たとえば、**PC1** や **TSUKUBA-CS-SOFTLAB-3** といったような、任意の文字列を設定できる。

MSID と PCID との間の対応テーブルは、コントローラの持っているサーバデータベースに格納されている。PCID はいつでもサーバプリケーション側から変更することができる。つまり、サーバプリケーションを実行しているユーザは、自らが好む覚えやすい文字列を PCID として使用することができる。ただし、他のサーバが使用している PCID を使用することはできない。

3.5.3 サーバの認証

コントローラは、サーバセッションの確立要求のためにアクセスしてきたサーバから証明書の提示を受け、MSID を決定する。本システムにおいては、MSID がインターネット上でのサーバのインスタンスを識別するための識別子である。しかし、MSID は秘密とすべき値ではない。なぜならば、クライアントが指定した PCID に対してコネクションを張るためにコントローラに対して接続要求を行ったとき、コントローラはサーバセッションが接続さ

れているゲートウェイの IP アドレス等と共に MSID を返すので、この時点で、サーバプリケーションの設置者以外の者が、そのサーバの MSID を知ることができるためである。

コントローラは、アクセスしてきたサーバが提示する証明書を受け取ると共に、サーバに対して、乱数メッセージを送信する。サーバは受信した乱数メッセージを、自らの持っている秘密鍵によってデジタル署名し、署名メッセージをコントローラに送信する。コントローラは、乱数メッセージに対するデジタル署名を、そのサーバから受け取った証明書に含まれる公開鍵によって検証し、サーバが提示してきた証明書の公開鍵に対応する秘密鍵をサーバが本当に保有しているかどうかを確認する。これにより、コントローラはアクセスしてくるサーバが偽者でないことを確認し、提示された証明書の SHA-1 ハッシュをもとに MSID を生成して負荷分散処理を行うことができる。

公開鍵アルゴリズムによるサーバの認証の動作を、図 21 に示す。

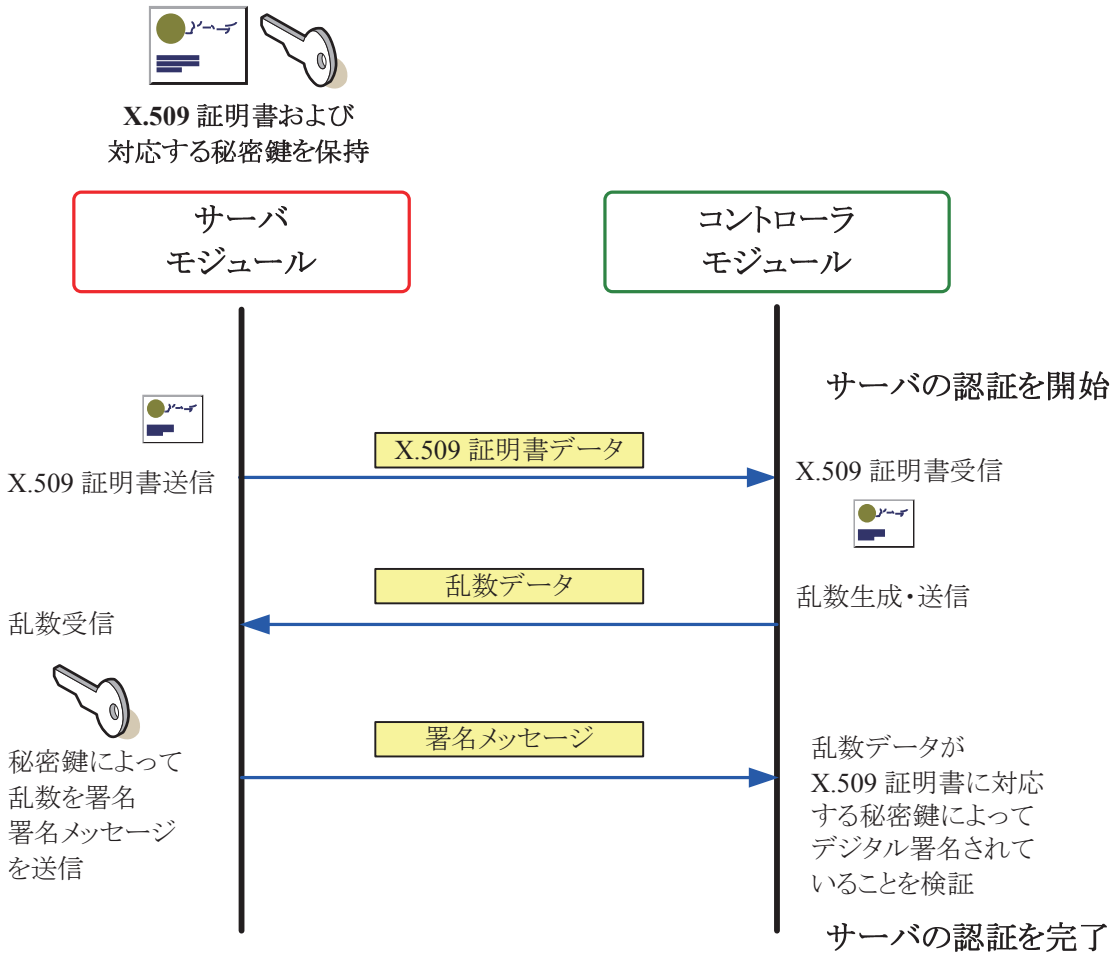

図 21 公開鍵アルゴリズムによるサーバの認証

3.5.4 サーバの収容先ゲートウェイの決定および指定

次に、コントローラはサーバとの間のセッションを確立させるべきゲートウェイを選択する。ここでの選択は、3.4.2 によって決定される。

接続先のゲートウェイが決定されたら、そのゲートウェイの IP アドレス、アクセスしてきたサーバの MSID が含まれるメッセージを、そのメッセージの有効期限と共に、コントローラの保有する秘密鍵によってデジタル署名し、サーバに返す。コントローラの保有する秘密鍵や、メッセージの有効期限については、3.6 で詳しく述べる。

3.5.5 サーバセッションの確立

コントローラから接続先のゲートウェイの IP アドレスや自らの MSID 等のメッセージが含まれる署名済みのデータを受け取ったサーバは、指定されたゲートウェイに対して接続し、その署名済みメッセージを提示する。ゲートウェイは受け取ったメッセージの電子署名を確認し、そのメッセージに含まれる MSID 等のデータや有効期限がコントローラによって認証されたものであることを検証する。そのために、あらかじめゲートウェイはコントローラの公開鍵が含まれた証明書を保有している。また、有効期限が現在時刻よりも後であることも確認する。これらの確認手続きが完了すれば、コントローラとサーバとの間でサーバセッションが確立される。

3.5.6 クライアントの接続要求

すでにゲートウェイシステムに対してサーバセッションを確立しているサーバとの間でコネクションを確立したいクライアントは、まずコントローラに対して接続要求を行う。この際、クライアントはサーバの PCID を指定する。

3.5.7 クライアントの接続先ゲートウェイの指定

コントローラはサーバの PCID をもとにそのサーバの MSID を検索する。次に、この MSID のサーバに対するサーバセッションが確立されているゲートウェイを、セッションテーブルを検索して取得する。そして、そのゲートウェイの IP アドレスおよび接続先サーバの MSID をクライアントに返す。

3.5.8 クライアントセッションの確立

クライアントは、指定されたゲートウェイに対して接続し、接続したい相手のサーバの MSID を指定する。ゲートウェイは、自らが現在処理しているサーバセッションのうち、その MSID を持ったサーバセッションを検索し、そのサーバセッションが存在する場合は、クライアントとゲートウェイの間でクライアントセッションが確立され、同時に、そのクライアントセッションとサーバセッションとの間で関連付けが行われる。

3.5.9 コネクション ID の割り当て

ゲートウェイは、接続しているサーバセッションに対して、新しいクライアントが接続してきたときに、コネクション ID を生成し、これをクライアントセッションに割り当てる。同時に、サーバに対してコネクション ID を通知し、新しいコネクションが確立されたこと

を通知する。コネクション ID は 32 bit で、同一サーバセッション内で重複しない値である。

1 個のサーバに対して複数のクライアントがコネクションを張って通信を行うとき、サーバとゲートウェイとの間で接続されているサーバセッションの内部は、複数本のコネクションが多重化されて流れることになる。従って、サーバの側から見たとき、ゲートウェイを経由して多重化され受信したストリームデータが、どのコネクションのものであるかを識別するための重複しない ID が必要である。この ID がコネクション ID である。また、サーバはゲートウェイに対して特定のコネクションに対するストリームデータの送信を行うとき、このコネクション ID を付加してそのデータを送信する (図 22)。

クライアントから見れば、ゲートウェイとの間で確立されているクライアントセッションには、1 本のコネクションしか流れない。なぜならば、特定のサーバに対して複数本のコネクションを確立したい場合は、その都度新しいクライアントセッションを確立すれば良いため、たとえば 1 個のクライアントから特定のサーバに対して 3 本のコネクションを確立したい場合は、ゲートウェイに対して 3 本の個別のクライアントセッションを確立すれば足りるからである。そのため、クライアントは、ゲートウェイとサーバが知っているコネクション ID を知る必要は無く、また通知も行われない。

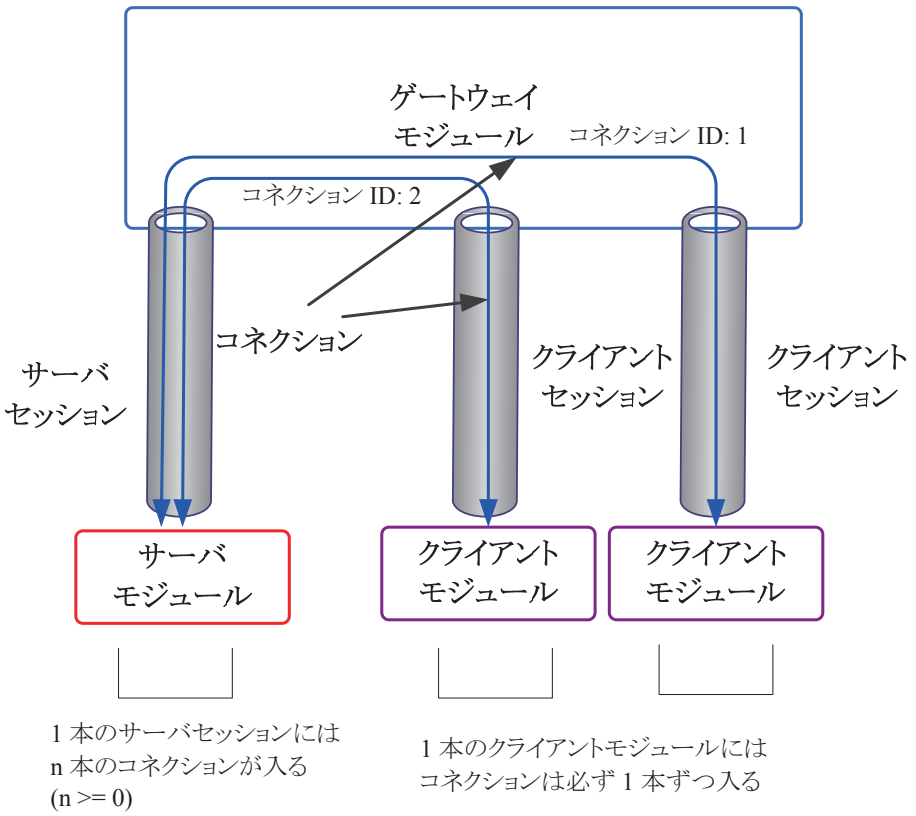

図 22 セッションとコネクションの関係

3.5.10 サーバ/クライアント間の通信

クライアントとサーバとの間でゲートウェイを通して通信するためのコネクションが確

立された後は、サーバとクライアントの間で、そのコネクションを経由して、任意のストリームデータの双方向送受信が可能となる。これは、あたかも TCP アプリケーションが TCP コネクションを用いてソケット通信を行うのと同等である。

クライアントがサーバに対してデータを送信しようとするときは、クライアントとゲートウェイ間で確立されているクライアントセッションを用いてサーバに対してペイロードのみを送信すればよい。受信する場合も同等である。

これに対して、サーバがクライアントに対してデータを送信しようとするときは、サーバとゲートウェイ間で確立されているサーバセッションに、宛先のクライアントを識別するコネクション ID をラベルデータとして付加したペイロードを送信する必要がある。逆に、クライアントからサーバがデータを受け取るときは、ゲートウェイはサーバに対して送信元のクライアントを識別するコネクション ID をラベルデータとして付加したペイロードを送信するので、これを受信したサーバは、コネクション ID を見て、どのクライアントからのデータであるのかを判別しなければならない。これらの、1 本のサーバセッションへの複数のコネクションの多重化処理および分離処理は、サーバおよびゲートウェイが行う。

サーバ/クライアント間のデータ通信の概要を、図 23 に示す。

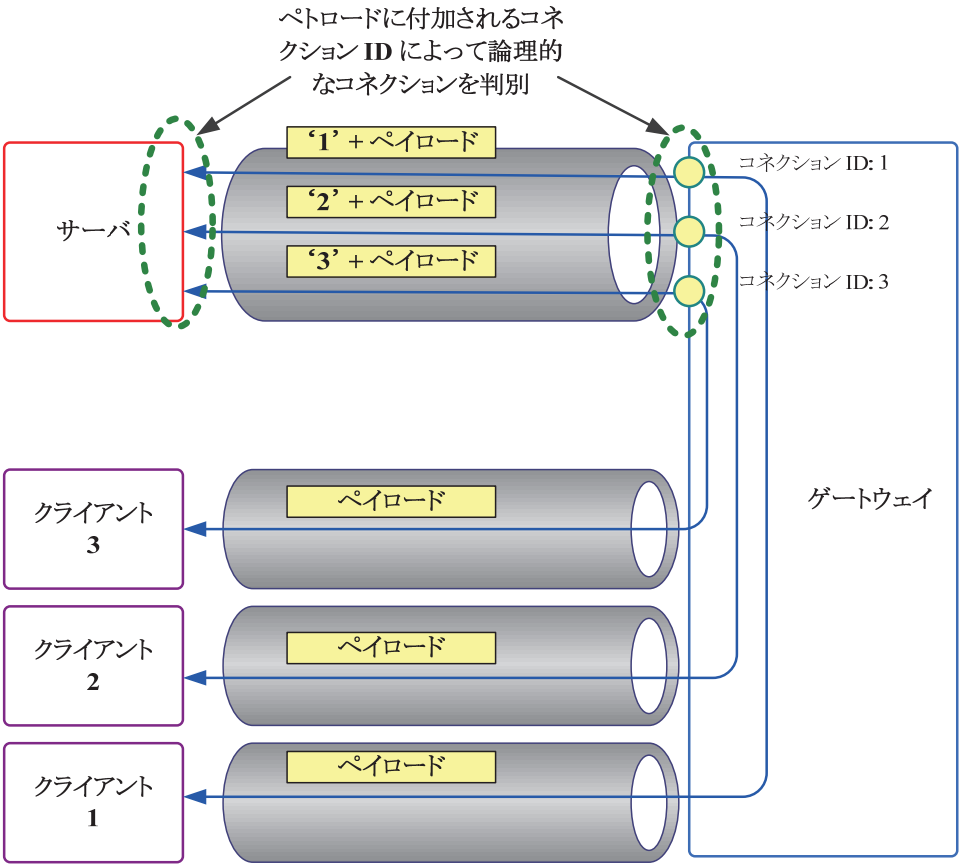

図 23 サーバ/クライアント間の通信

3.5.11 クライアントによる切断

クライアントがサーバとの間で確立したコネクションが切断される場合は 3 通りある。クライアントがコネクションを切断した場合、サーバがコネクションを切断した場合、または通信経路にエラーが発生した場合、の 3 通りである。

クライアントがコネクションを切断したい場合は、単純に、クライアントがゲートウェイとの間で確立しているクライアントコネクションを切断すれば良い。ゲートウェイはクライアントコネクションが切断されたことを検出すると、直ちにサーバに対して、そのクライアントコネクションに関連付けられていたコネクション ID と共に、切断された旨のメッセージを送信する。これにより、サーバは当該コネクションが切断されたことを知ることができる。

3.5.12 サーバによる切断

サーバが特定のクライアントとの間のコネクションを切断したい場合は、ゲートウェイに対して、切断したいコネクションのコネクション ID と共に、切断する旨のメッセージを送信する。これにより、ゲートウェイは当該コネクション ID に関連付けられているクライアントコネクションを切断するので、クライアントはサーバとの間のコネクションが切断されたことを知ることができる。

また、サーバとゲートウェイとの間のセッションが切断された場合は、そのサーバに対してコネクションを確立していたすべてのクライアントとの間のクライアントセッションはすべて切断されるので、クライアントはサーバとの間のコネクションが切断されたことを知ることができる。

3.5.13 通信経路上のエラーによる切断

通信系路上のエラーを検出するために、ゲートウェイとクライアント間、およびゲートウェイとサーバ間で確立されるそれぞれのセッション上では、キープアライブ通信を行うこととした。すなわち、セッション内で最後に何らかのデータが通信されてから一定時間無通信状態が経過したら、キープアライブのための、ペイロードを持たないメッセージを送信するのである。

これにより、一定時間 (例えばキープアライブ間隔の 2 倍の時間) 無通信状態であった場合は、通信相手との間の回線でエラーが発生したか、もしくは通信相手がクラッシュしたものとみなして、そのセッションが切断されたものとする。

3.6 認証

本システムには、通信を行うとする相手が偽者ではないことを確認するための認証の仕組みを組み込む。ここでは、認証の仕組みについて詳しく述べる。

3.6.1 ルート証明書

本システムは、大きく分けて、ゲートウェイシステムと、ゲートウェイシステムを利用するアプリケーション側モジュールに分離することができる。ゲートウェイシステムはストリーム中継サービスを提供しようとするサービスプロバイダー等によって設置・運用され、アプリケーション側モジュールは、インターネット上の色々なユーザによって起動され、運用されることを想定すると、まず、アプリケーションのユーザから見たとき、通信しようとしているゲートウェイシステムに属するサーバコンピュータ (コントローラおよびゲートウェイ) が本当に意図しているゲートウェイシステム設置者によって設置されているものであるかということを認証しなければ、そのシステムを安心して使用することができない。

本システムにおいては、このためのユーザ認証を確実かつシンプルに行うため、X.509 証明書[14] における PKI の仕組みの一部を利用する。まず、ゲートウェイシステム設置者がルート証明書を生成し保有する。このルート証明書に対応する秘密鍵は、ゲートウェイシステム設置者が秘密に保管する。ルート証明書は、ゲートウェイシステムを経由して通信を行うサーバプリケーションおよびクライアントアプリケーションと共に配布する。

ゲートウェイシステム設置者は、次に、1 個のコントローラ用証明書、および複数個のゲートウェイ用証明書を生成し、これらの証明書をルート証明書の秘密鍵によってデジタル署名したものを作成し保有する。

証明書の種類の一覧と認証の関係を、図 24 に示す。

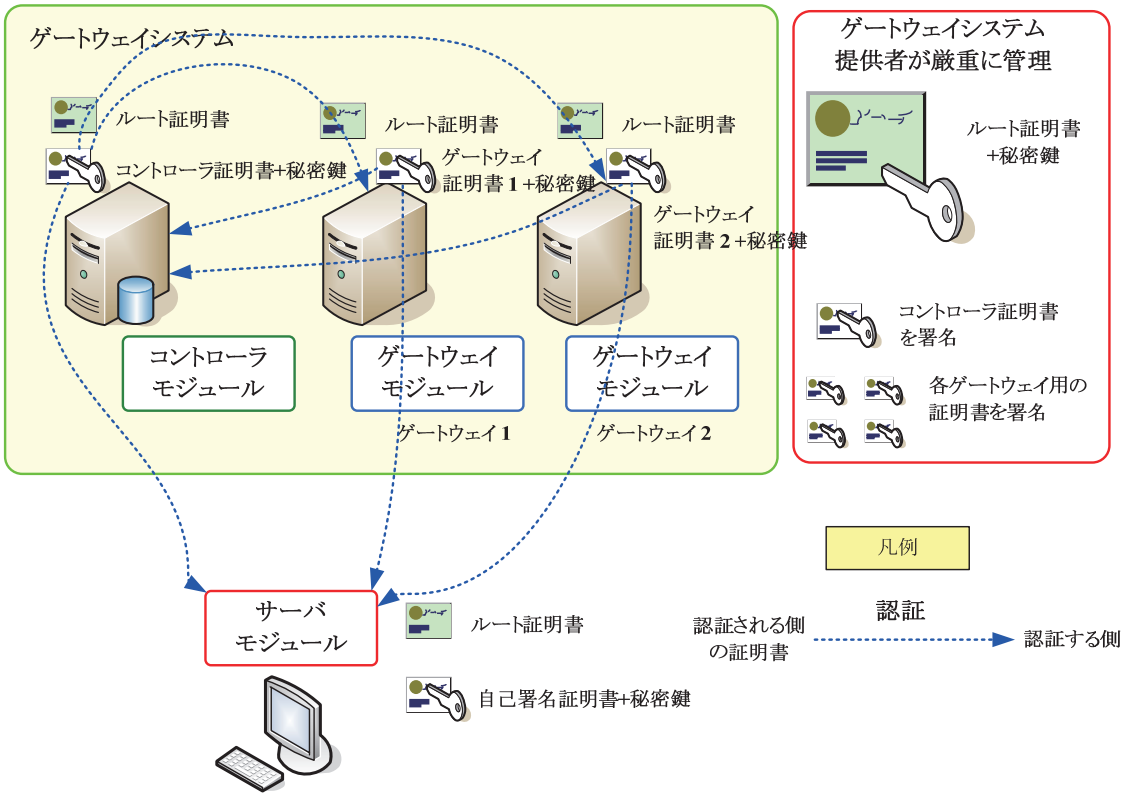

図 24 証明書の一覧と認証の関係

3.6.2 コントローラ証明書

コントローラとなるサーバコンピュータは、コントローラ証明書およびそれに対応した秘密鍵を持つ。他のコンピュータがコントローラとの間で通信を行う場合は、まず、SSL プロトコル [15] によってコントローラがルート証明書によって署名された証明書を提示したかどうか、およびその証明書の公開鍵に対応する秘密鍵をコントローラが持っているかどうかを検証する。同時にこのパラメータを用いて SSL プロトコル上でセッション鍵の交換を行い、以後の暗号化通信に利用する。これにより、コントローラと通信する他のコンピュータは接続先のコントローラが偽者でないことを検証できると共に、以後のコントローラとの間の通信内容の盗聴・改ざんを防ぐことができる。

3.6.3 ゲートウェイ証明書

ゲートウェイとなるサーバコンピュータは、ゲートウェイ証明書およびそれに対応した秘密鍵を持つ。他のコンピュータがゲートウェイとの間で通信を行おうとするときに、ゲートウェイがこの証明書を提示することにより、他のコンピュータは通信相手のゲートウェイが偽者でないことを検証できること等は、コントローラにおけるコントローラ証明書と同等である。

加えて、ゲートウェイはコントローラに対して 3.4 および 3.5 で述べたようなゲートウェイ登録、セッションの増減の通知およびキープアライブのための通信を行う。この際、ゲートウェイはコントローラに対してゲートウェイ証明書を提示し、自らがコントローラに対して送信するメッセージをデジタル署名するので、コントローラはこれらのメッセージの送信元が本物のゲートウェイであることを検証することができる。

3.6.4 サーバ証明書

サーバは、3.5.2 および 3.5.3 で述べたように、初回起動時に自らのための秘密鍵および公開鍵を生成し、これらによって自己署名証明書を生成する。2 回目以降の起動時には、初回に生成したこれらの鍵および証明書を利用する。

3.6.5 コントローラ/ゲートウェイ間の認証

コントローラ証明書およびゲートウェイ証明書を基に、コントローラ/ゲートウェイ間は、相互に相手を認証することができる。

コントローラはゲートウェイから、必要に応じて、もしくは定期的に送られてくるゲートウェイ登録メッセージ、キープアライブメッセージおよびセッションの増減の通知メッセージを受信する。

この際、コントローラはゲートウェイが提示するゲートウェイ証明書を検証することにより、それらのメッセージが、自らが保有するルート証明書によって署名されたゲートウェイ証明書および秘密鍵を持ったゲートウェイから発せられたものであることを認証する。

また、ゲートウェイは通信先のコントローラが提示するコントローラ証明書を検証することにより、同様に、コントローラを認証する。

これらの仕組みにより、3.2.4 で述べたように、コントローラとゲートウェイとの間の通信を、通信内容が盗聴・改ざんされる可能性のあるインターネット回線によって行っても安全である。

3.6.6 コントローラ/サーバ間の認証

コントローラ証明書およびサーバ証明書を基に、コントローラ/サーバ間は、相互に相手を認証することができる。

コントローラはサーバから、新しいサーバセッションの確立要求や、PCID の変更要求などのメッセージを受け取る。この際、サーバがコントローラに対して送信するすべてのメッセージは、サーバ証明書が付加され、またサーバ証明書に対応するサーバの秘密鍵によってデジタル署名されている。これにより、コントローラはアクセス元のサーバが本当にサーバ証明書および秘密鍵を持っていることが認証でき、同時にサーバ証明書の SHA-1 ハッシュをもとに、アクセス元サーバの MSID を決定することができる。

サーバは通信先のコントローラが提示するサーバ証明書を検証することにより、コントローラを認証し、コントローラがゲートウェイシステム設置者によって設置されているものであるということを認証することができるとともに、コントローラとサーバとの間で行われるすべての通信は盗聴・改ざんから保護される。

3.6.7 ゲートウェイ/サーバ間の認証

ゲートウェイ証明書を基に、サーバは、接続先のゲートウェイがゲートウェイシステム設置者によって設置されているものであるということを認証することができるとともに、ゲートウェイとサーバとの間で行われるすべての通信は盗聴・改ざんから保護される。

また、サーバがコントローラから受け取る、接続先ゲートウェイの IP アドレスおよび自らの MSID が格納されたメッセージは、有効期限が付加されており、コントローラの持つ秘密鍵によってデジタル署名されている。そのメッセージをサーバはそのままゲートウェイに対して提示するため、ゲートウェイから見れば、自らに接続しサーバセッションの確立を要求しようとしているサーバは、正規のコントローラによって自らに対して負荷分散されようとしているサーバであるということを認証することができる。

3.6.8 有効期限付き署名メッセージ

3.6.7 で述べたメッセージに付加された有効期限について詳しく述べる。ゲートウェイは、一旦コントローラを介して負荷分散されたサーバからのサーバセッション確立要求にのみ応答するべきである。なぜならば、コントローラを介さずに直接ゲートウェイに対してサーバセッションの確立要求が行われることは、コントローラによる負荷分散アルゴリズムに従わずに各サーバセッションが特定のゲートウェイに多数確立されるという状況を発生し得るものであり、一旦コントローラを経由してリダイレクトされるという 3.5.1 で述べた手順に従わないサーバプリケーションの開発・動作を許してしまうことになり、システム全体の公平な負荷分散が侵害されることになるためである。

そのため、ゲートウェイは、サーバから提示されるリダイレクトのためのメッセージに、コントローラによるデジタル署名が付加されていることを確認するとともに、そのメッセージの有効期限を確認する。有効期限とは、コントローラが、サーバが収容されるべきゲートウェイに対するリダイレクトメッセージをサーバに対して返したときから一定期間内に速やかに指定されたゲートウェイに対してリダイレクトしサーバセッションの確立要求を行わなければならないようにするための、コントローラによって指定される時刻である (例えば 60 秒間)。この有効期限データもまたメッセージの一部でありコントローラによって署名されるので、サーバはメッセージを改ざんしたり、有効期限を変更したり、もしくは有効期限が切れた古いメッセージを用いてゲートウェイに対して接続要求を行ったりすることはできなくなる。すなわち有効期限付きの署名されたリダイレクトのためのメッセージは、コントローラによって一時的に発行されるトークンとして動作し、ゲートウェイは、接続しようとしてきたサーバがこのトークンを所有しているかどうかによって、サーバが自らに接続する資格を持っていることを認証するのである。

3.6.9 クライアントによるコントローラの認証

接続先のコントローラが提示するコントローラ証明書を基に、クライアントはコントローラがゲートウェイシステム設置者によって設置されたものであることを認証することができるとともに、クライアントとコントローラとの間で行われるすべての通信は盗聴・改ざんから保護される。

3.6.10 クライアントによるゲートウェイの認証

接続先のゲートウェイが提示するゲートウェイ証明書を基に、クライアントはゲートウェイがゲートウェイシステム設置者によって設置されたものであることを認証することができるとともに、クライアントとゲートウェイとの間で行われるすべての通信は盗聴・改ざんから保護される。

3.6.7 で述べたように、サーバとゲートウェイとの間で行われるすべての通信は盗聴・改ざんから保護されるため、クライアントとゲートウェイ間の通信回線、およびサーバとゲートウェイ間の通信回線上に悪意を持った者が存在しても、その者は、クライアントとサーバ間を流れるメッセージを盗聴・改ざんすることはできない。

3.6.11 アプリケーション上での認証

ここまでで述べた認証は、ゲートウェイシステムと、ゲートウェイシステムを利用するアプリケーション側モジュールの間での認証である。

本システムは、サーバとクライアントとの間のコネクションの確立と、そのコネクションの内部での通信を実現しようとするものであり、サーバがクライアントからのコネクションを確立された際の、コネクション確立元クライアントをサーバが認証する作業、およびサーバをクライアントが認証する作業という 2 つの認証については提供しない。なぜならば、それらの認証で求められるセキュリティのレベルや暗号強度、認証方式 (例えばパスワ

ードによる認証を行うのか、それとも PKI による認証を行うのか) といったことは、各サーバおよびクライアントアプリケーションの性質に依存するため、本システムで一元的に決定してしまうべきではないと考えたためである。

従って、サーバはクライアントから確立されようとしたすべてのコネクションを一旦受け入れ、その上でアプリケーションレイヤのユーザ認証が必要であれば、自由にどのような認証でも行うことができる。

第 4 章 実装

本章では、設計した通信システムを実際のコンピュータ上で動作させるためのプログラミング方法およびその結果得られたプログラムについて述べる。

4.1 実装の準備

本システムを実装する上で、まず、実装したシステムをどのようなシステム上で動作させるかを決定し、使用すべきプログラミング言語や開発環境などを用意する。

4.1.1 プラットフォーム

本研究において実装を行うプラットフォームについては、まずは Windows 上で動作することを目指し、また、ゲートウェイ、サーバおよびクライアント側のライブラリモジュールについてはできる限りプラットフォーム非依存とする。これにより、今回は表 2 に示すの環境で動作するモジュールを実装する。

表 2 実装するモジュールの動作環境

モジュール名	プラットフォーム
コントローラ	Windows 2000 以降の OS および IIS (Internet Information Service) (ASP.NET 2.0 アプリケーションとして動作) および SQL Server 2000 以降の RDBMS CPU には非依存
ゲートウェイ	Windows 2000 以降の OS (Win32 ネイティブプログラムとして動作) Intel x86 CPU
サーバ クライアント	Windows 98, ME, NT 4.0, 2000, XP, Server 2003 および Vista 上で動作する Win32 アプリケーションに組み込むことができるスタティックリンクライブラリ (Win32 ネイティブプログラムとして動作) Intel x86 CPU

4.1.2 プログラミング言語

これらのモジュールを開発するためのプログラミング言語は、表 3 のように選択する。

表 3 モジュールを開発するためのプログラミング言語

モジュール名	プログラミング言語
コントローラ	C#.net 2.0 および Transact-SQL 言語
ゲートウェイ	C 言語

サーバ	C 言語
クライアント	C 言語

4.1.3 開発環境

これらのモジュールを開発するための開発環境は、表 4 のように選択する。

表 4 モジュールを開発するための開発環境

モジュール名	開発環境
コントローラ	Visual Studio 2005 および SQL Server Management Studio
ゲートウェイ	Visual Studio 2005
サーバ	Visual Studio 2005
クライアント	Visual Studio 2005

4.2 使用システムソフトウェア

4.1.1 とも関連するが、本システムの実装および実装したものの実行にあたっては、以下の既存のシステムソフトウェアを利用する。

オペレーティングシステム

コントローラおよびゲートウェイについては、Windows Server 2003 Service Pack 1 を使用する。

サーバおよびクライアントについては、各種 Windows で動作することを検証するため、Windows 98、Windows 98 SE、Windows ME、Windows NT 4.0、Windows 2000、Windows XP、Windows Server 2003 および Windows Vista を使用する。なお、プログラミング環境としては、Windows XP を使用する。

ランタイム環境

コントローラについては、.NET Framework 2.0 を使用する。

ゲートウェイ、サーバおよびクライアントについては、Win32 API を使用する。ただし、将来的に Linux や Darwin 等の他の OS で動作するように移植しやすくするため、Win32 API を呼び出す場所をラッピングし、簡単に他 OS のシステムコールを使用するように改変することができるようにする。

暗号化エンジン

3.6 におけるデジタル署名の生成や暗号化等を行うため、既存の暗号化エンジンを使用する。

コントローラについては、.NET Framework 2.0 経由で呼び出す暗号化 API (System.Security

以下) を使用する。
ゲートウェイ、サーバおよびクライアントについては、OpenSSL 0.9.8a を使用する。

アプリケーションプラットフォーム

コントローラは、サーバコンピュータ上で常時動作すべき常駐プログラムであるが、開発を簡単にするため、実際には Windows 上の Internet Information Service (IIS) 上で動作する ASP.NET 2.0 アプリケーションとして実装する。

データベースエンジン

データベースエンジンは、SQL Server 2005 Service Pack 1 を使用する。

4.3 コントローラの実装

コントローラの実装について述べる。

4.3.1 システム概要

コントローラは、IIS 上で動作する ASP.NET アプリケーションとして動作する。コントローラはいつでもゲートウェイ、サーバおよびクライアントからアクセスされ、要求を受けることができるようにするため、サーバプリケーションとして特定の TCP ポートを開いておく必要があるが、この新しいリクエストの受け付け処理は IIS に任せている。
IIS は HTTPS のポート (TCP ポート番号 443) を常に待ち受けており、ここに接続したゲートウェイ、サーバおよびクライアントは、SSL コネクションを確立した上で、HTTP POST メソッドによってリクエストをコントローラプログラムに送信する。コントローラプログラムは、そのリクエストを処理した結果のレスポンスを返信する。
IIS 上で予め設置者が決めたパスがエントリポイントとなり、そのエントリポイントがコントローラ本体のプログラムに対してリクエストを転送し、またレスポンスをクライアントに返す。例えば、/control/entrance.aspx というパスがエントリポイントとなっている場合は、https://IP アドレス/control/entrance.aspx に対してアクセスすることにより、コントローラに対してリクエストを発することができる。
コントローラ本体のプログラムは、内部で.NET Framework 2.0 が提供するクラスライブラリの提供する機能を呼び出し、デジタル署名の処理を行ったり、サーバデータベースが格納されている SQL Server にアクセスしてクエリーを送信したりする。
コントローラの実装概要図を、図 25 に示す。

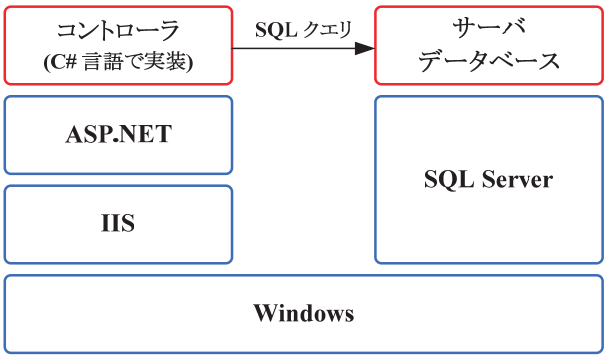

図 25 コントローラの実装概要図

4.3.2 コンフィグレーション項目

コントローラプログラムは、起動時に、予め設定ファイルに書かれている以下の項目を読み込む。また、コントローラプログラムの実行中に設定ファイルの内容が変化した場合は、再度読み込む。

- ルート証明書のファイル名
- コントローラ証明書ファイルのファイル名
- コントローラ証明書に対応する秘密鍵ファイルのファイル名
- SQL Server に対してアクセスするためのデータベース接続文字列

4.3.3 データベース構築

コントローラが使用するサーバデータベースを格納するため、SQL Server 上に新しいデータベースを作成し、表 5 のようなテーブルを定義する。

表 5 コントローラが使用するサーバデータベースのテーブルの定義

列名	データ型	内容
MSID	nvarchar(128)	サーバの MSID が格納される。なお、MSID は通常は ”MSID-ハッシュ” のような形で、サーバの持つ証明書の SHA-1 ハッシュを “MSID-“ の後に付加した文字列となる。
PCID	nvarchar(31)	サーバの PCID が格納される。新しいレコードを登録する際は、乱数から生成した他のレコードと重複しない ID が設定されるが、あとからサーバのユーザによっていつでも変更可能である。
CERT_HASH	nvarchar(40)	サーバの持つ証明書の SHA-1 ハッシュが格納される。
CREATE_DATE	datetime	レコードが作成された日時が格納される。
UPDATE_DATE	datetime	レコードの内容が更新された日時が格納される。

4.3.4 プログラミング

コントローラプログラムは C#言語で記述する。また、コントローラからサーバデータベースにアクセスする際の言語としては、Transact-SQL で記述する。

4.4 ゲートウェイの実装

ゲートウェイの実装について述べる。

4.4.1 システム概要

ゲートウェイは、Windows 上でシステムサービスとして動作する。Windows におけるシステムサービスとは、UNIX におけるデーモンプロセスのようなもので、Windows が起動している間は常にバックグラウンドで稼動し続けるものである。

当初はゲートウェイをプログラミングの容易性から .NET Framework 2.0 上で動作するアプリケーションとして記述することを検討したが、将来的に、本ゲートウェイプログラムを Linux や Mac OS X 等の OS 上で稼動させることを考慮し、移植が容易になるように、ネイティブコードで実装することとした。

Windows 版のゲートウェイにおいては、インターネットとの間の TCP/IP 通信は WinSock API を用いた。ゲートウェイは、稼動時は常にポート 443 を待ち受け状態にしており、新しいサーバセッションまたはクライアントセッションの確立要求が届くのを常に待機しているサーバとして動作する。

ゲートウェイはまた、コントローラに対してはクライアントとして動作する。すなわち、一定間隔または必要に応じてコントローラのポート 443 に対して接続し、ゲートウェイ登録やセッションの増減等の報告を行うのである。これらのポート 443 を用いた通信はすべて SSL コネクションを確立した上で行われる。

ゲートウェイの実装概要図を、図 26 に示す。

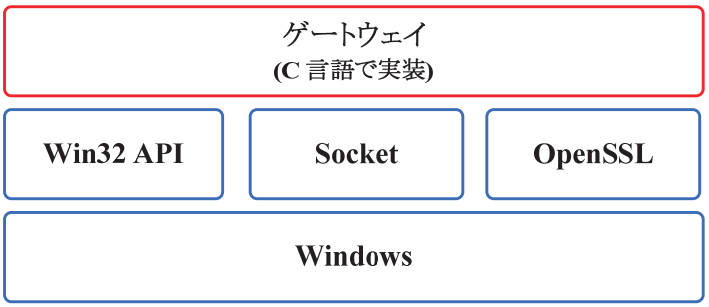

図 26 ゲートウェイの実装概要図

4.4.2 コンフィグレーション項目

ゲートウェイプログラムは、起動時に、予め設定ファイルに書かれている以下の項目を読み込む。また、ゲートウェイプログラムの実行中に設定ファイルの内容が変化した場合は、

再度読み込む。

- ルート証明書のファイル名
- コントローラの IP アドレス
- コントローラに対してキープアライブ通信を行う際の間隔

4.4.3 プログラミング

ゲートウェイプログラムは C 言語で記述する。SSL 通信を必要とするところでは、OpenSSL を使用する。

4.5 サーバおよびクライアントの実装

サーバおよびクライアント (通信ライブラリ) の実装について述べる。

4.5.1 ライブラリ化

3.3.4 および 3.3.5 で述べたように、サーバおよびクライアントは、本システムを用いたストリーム通信を行おうとするアプリケーションの開発者が呼び出すことができる、汎用的な通信 API を提供するため、関数ライブラリの形で実装する。具体的には、今回実装した Windows 版ライブラリは、C 言語のスタティックリンクライブラリおよびヘッダファイルによって構成される。

なお、サーバおよびクライアントのための通信 API は、共通して実装すべき部分が多いため、同一のライブラリにサーバとしての機能およびクライアントとしての機能を共存させる。ゲートウェイプログラムは C 言語で記述した。SSL 通信を必要とするところでは、OpenSSL を使用する。

サーバおよびクライアントの実装概要図を、図 27 に示す。

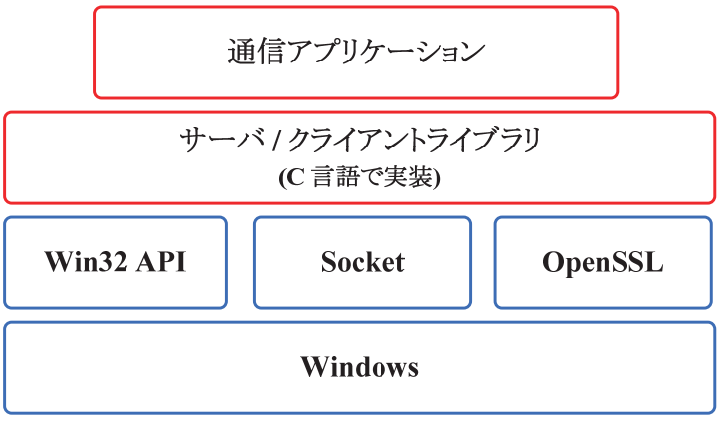

図 27 サーバおよびクライアントの実装概要図

4.5.2 接続待機 API

サーバ側で、クライアントからの接続を待機させるための API を、以下に示す。

```
void ServerStart(ACCEPT_PROC *accept_proc, void *accept_param);
```

ただし、ACCEPT_PROC *は関数ポインタであり、以下のように定義される。

```
typedef void (ACCEPT_PROC)(THREAD *thread, SOCKIO *sock, void
  *param);
```

これにより、新しいサーバ機能の動作が開始され、クライアントからのコネクションが張られた場合は、accept_proc で指定されたコールバック関数が呼ばれる。

SOCKIO は本ライブラリで提供されるソケットを指す内部的な構造体のインスタンスへのポインタである。サーバプリケーションおよびクライアントアプリケーションは、ライブラリによってコネクションが確立された後は、ライブラリから SOCKIO 型の新しいポインタを受け取り、ソケット API におけるソケットのようなものであるとして取り扱うことになる。以後、SOCKIO を用いて、そのコネクションに対してデータを送信したり受信したりすることになる。

4.5.3 接続 API

クライアント側で、サーバに対してコネクションを接続するための API を、以下に示す。

```
UINT ClientConnect(char *pcid, SOCKIO **sockio);
```

pcid には文字列で接続先のサーバの PCID を指定する。接続に成功した場合は、確立されたコネクションを使用するための SOCKIO 型のポインタを受け取ることができる。

4.5.4 同期送信 API

サーバおよびクライアントにおける、コネクションを使用したデータの送受信のための API は同一である。データの送受信には同期と非同期がある。同期送信 API は、指定されたデータのうち 1 バイト以上を相手に送信し、送信が完了したバイト数を返す。必ずしも、一度に指定したすべてのデータが送信されるわけではない。また、ネットワークが混雑しているなどの場合でデータの送信に時間がかかった場合は、データの一部の送信が実行されるまで制御を返さない。

同期送信の API を、以下に示す。

```
UINT SockIoSend(SOCKIO *io, void *data, UINT size);
```

4.5.5 同期受信 API

同期受信 API は、相手から 1 バイト以上のデータが送られてくるのを待って、データが届いたらデータとともに制御を返す API である。同期受信の API を、以下に示す。

```
UINT SockIoRecv(SOCKIO *io, void *data, UINT size);
```

4.5.6 非同期イベントメカニズム

本ライブラリでは、非同期でデータの送受信を行うための API も実装する。

同期送受信 API を用いる場合は、1 つのスレッドで複数のコネクションを同時に扱うのは難しい。なぜならば、1 つのコネクションでブロッキングが発生した場合、ブロッキングが発生した原因が解消されるまでの間は同期 API の内部で制御がブロックされるため、複数のコネクションを管理しておき、それぞれのコネクションのうち一部が送信可能もしくは受信可能となったときに直ちにそのコネクションを用いてデータ転送を行うことができず、各コネクションを同時に用いた並列通信ができないためである。

そのため、本 API では既存のソケット API を用いた非同期プログラミングを行う際に用いるべき select() システムコールや poll() システムコールに似た、コネクションの状態が変化するまでの間待機する API を提供し、同時に非同期送受信を実行するための API も提供する。

まず、コネクションを指し示す SOCKIO 型に関連付けられている待機イベントを取得するための API として、以下の API を提供する。

```
SOCK_EVENT *SockIoGetSockIoEvent(SOCKIO *io);
```

上記の API を呼び出すと、SOCK_EVENT 型のポインタが返却される。SOCK_EVENT 型の実装は、OS によって異なるが、Windows の場合は、HANDLE 型で定義される Windows のイベントオブジェクトへのハンドルが入っている。このイベントハンドルを Win32 API の WaitForSingleObject 関数または WaitForMultipleObjects 関数などで使用することにより、イベントがセットされるまでの間、CPU を解放し待機することができる。

また、逆に、既存のイベントオブジェクトをコネクションに関連付けることができるようにもした。このための API として、以下の API を提供する。

```
void SockIoReplaceIoEvent(SOCKIO *io, SOCK_EVENT *e);
```

4.5.7 非同期送信 API

非同期送信 API は、指定されたデータのうち 1 バイト以上を相手に対して送信しようとし、送信が完了した場合は送信されたバイト数を返し、もしネットワークが混雑しているなどの理由で送信が直ちにできない場合は、内部でブロックせずに、すぐに 0xFFFFFFFF を返り

値として制御を戻す API である。

非同期送信の API を、以下に示す。

```
UINT SockIoSendAsync(SOCKIO *io, void *data, UINT size);
```

4.5.8 非同期受信 API

非同期受信 API は、相手から 1 バイト以上のデータが送られてきている場合はそのデータを返し、もしデータがまだ届いていない場合は、内部でブロックせずに、すぐに 0xFFFFFFFF を返り値として制御を戻す API である。

非同期受信の API を、以下に示す。

```
UINT SockIoRecvAsync(SOCKIO *io, void *data, UINT size);
```

4.5.9 タイムアウト設定 API

これまでに述べた API のうち、同期送受信 API については、ネットワークが混雑しているために送信不能であったり、まだデータが 1 バイトも届いていないため受信不能であったりする場合は、その状況が変化し、データが送信できたり、もしくはデータを受信したりするまでの間にブロッキングする。

デフォルトでは、ブロッキングは、コネクションが切断されるか、ブロッキングの原因が解消されるまでの間、無制限に実行されるが、予めタイムアウト API を設定しておくことにより、ブロッキング状態が指定された時間以上経過したときは直ちにブロッキングを解除して制御を戻すことができるようにした。

タイムアウト設定の API を、以下に示す。

```
void SockIoSetTimeout(SOCKIO *io, UINT timeout);
```

4.5.10 状態取得 API

一度確立されたコネクションは、切断されるまでの間、自由なストリームデータの送受信に利用することができる。コネクションが切断された場合は、それ以降に送受信 API を呼び出すと、返り値として 0 が返却されるため、それによりコネクションの切断を検出することができる。

また、コネクションの状態取得 API を呼び出すことにより、いつでもそのコネクションが現在切断されているかどうかを検出することができる。

コネクションの状態取得の API を、以下に示す。

```
bool SockIoIsConnected(SOCKIO *io);
```

4.5.11 切断 API

コネクションは、サーバ側からもクライアント側からもいつでも切断することができる。切断の API を、以下に示す。

```
void SockIoDisconnect(SOCKIO *io);
```

4.5.12 PCID の取得および変更 API

サーバは、自らが使用している PCID をいつでも取得することができる。また、PCID をいつでも任意のものに変更することができる。これらの取得および変更処理は、実際にはゲートウェイシステム側に登録されているサーバデータベースにアクセスする必要があり、コントローラとの間の通信が行われる。

PCID の取得は以下の API によって実行することができる。

```
UINT ServerGetLoginInfo(LOGIN_INFO *info);
```

ここで、LOGIN_INFO 構造体は以下のように定義されている。

```
typedef struct LOGIN_INFO
{
  char Msid[129];
  char Pcid[32];
  UINT64 CreateDate;
  UINT64 UpdateDate;
} LOGIN_INFO;
```

上記のうち、Msid[129] は MSID、Pcid[32] は PCID、CreateDate はサーバデータベースに初めてこの MSID のレコードが登録された日時、UpdateDate はサーバデータベースのレコードが最後に更新された日時を示す。

PCID の変更は以下の API によって実行することができる。

```
UINT ServerRenameMachine(char *pcid);
```

4.5.13 API の使用方法のまとめ

本節で述べた、本システムを用いて通信を行う際に呼び出すべき代表的な API を、コネクションの確立、データ通信および切断手順に分けて、図 28 に示す。ソケット API における通信の手順 (図 10) と比較して、ソケット API を用いた場合とほぼ同一の API 呼び出しで、コネクションの確立、データ通信および切断を行うことができる。

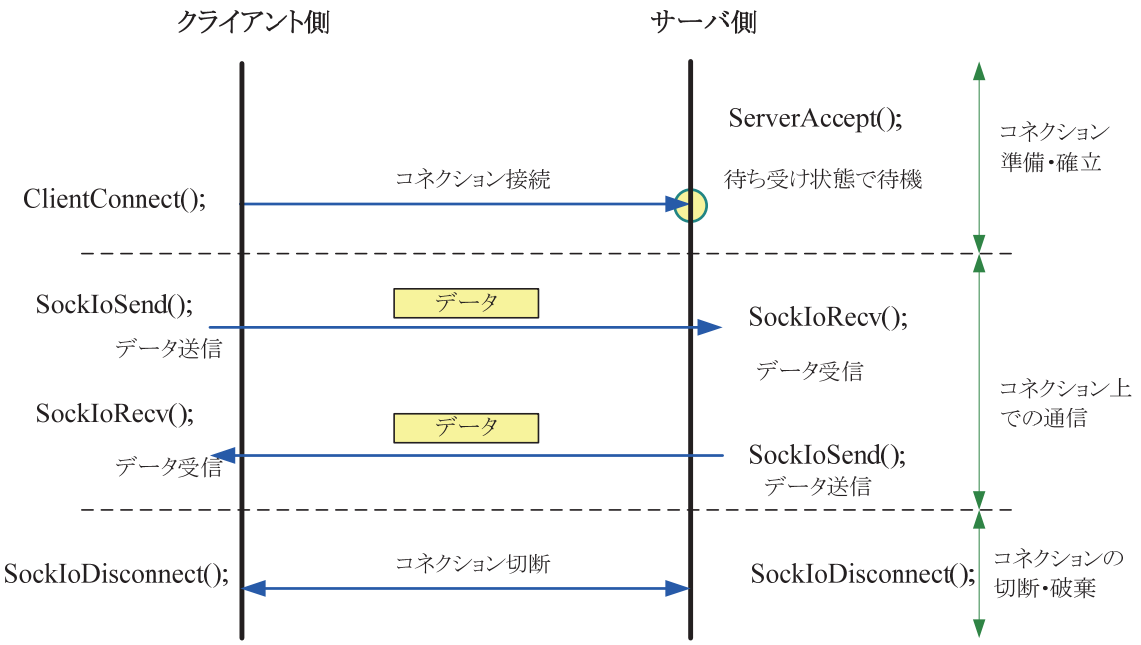

図 28 本システムの API における通信の手順

第 5 章 通信実験

本章では、実装した本システムのプログラムを用いて通信性能の測定を行い、その結果について述べる。

5.1 実験概要

実装した本システムの性能を評価するための通信実験の概要について述べる。

5.1.1 実験の目的

本実験の目的は、本システムを使用したデータ通信の性能評価を行うことである。すなわち、本システムを使用せずに物理的なネットワーク上で通信を行った場合と比較して、本システムを用いることによりどの程度のオーバーヘッドが生じるかということを知るために実験を行う。

本システムを使用した通信では、各アプリケーションは、コントローラに対する接続要求処理を行った後に、ゲートウェイを経由してコネクションを確立させ、アプリケーションデータの送受信を行うことになるが、本実験ではゲートウェイを経由してコネクションが確立した後のクライアント/サーバ間の通信についてのみ評価を行う。そのため、ゲートウェイシステムはコントローラ 1 台とゲートウェイ 1 台の 2 台のみによって構成されることになる。

ゲートウェイを複数台設置し、多数の接続要求がコントローラに対して行われるような環境における実証実験は、第 6 章で述べる。

5.1.2 遅延およびスループット

本実験において評価の対象とするのは、2 台のコンピュータ間で通信を行う際に測定できる、遅延とスループットの変化である。本実験ではいくつかのネットワーク環境で実験を行う。それぞれの環境で、まず本システムを用いずにネットワーク上で物理的な通信を行い、遅延およびスループットを測定する。次に、本システムを用いてコネクションを確立し、そのコネクション上で遅延およびスループットを測定する。その後、物理的な通信を行った場合と、コネクション上で通信を行った場合の 2 者を比較する。

5.1.3 コンピュータ環境

実験に用いるためのコンピュータを 4 台用意した。それぞれの概要は、表 6 のとおりである。

表 6 実験に用いるコンピュータ

コンピュータ名	役割	スペック等
CONTROLLER	本システムにおけるコントローラ	DELL PowerEdge SC430 CPU: Pentium D 2.8GHz (2 CPU) RAM: 512 Mbytes LAN: Gigabit Ethernet OS: Windows Server 2003
GATE	本システムにおけるゲートウェイ	同上
PC1	通信者	Supermicro PDSLA CPU: Pentium 4 3.0GHz (2 CPU) RAM: 2048 Mbytes LAN: Gigabit Ethernet OS: Windows XP Professional
PC2	通信者	ThinkPad X60s CPU: Intel Centrino Duo L2400 1.66GHz (2 CPU) RAM: 2048 Mbytes LAN: Gigabit Ethernet PHS: NEC AX520N 256Kbps (WILLCOM) OS: Windows Vista Ultimate

5.1.4 ネットワーク環境

実験に用いるために、いくつかのネットワーク環境を用意した。
それぞれのネットワークの概要は、以下のとおりである。

1. LAN
   Gigabit Ethernet の単一セグメントで構成されるネットワークである。
2. 学内ネットワーク
   筑波大学内に敷設されている学内ネットワークである。130.158.0.0/16 のグローバル IP アドレスで運用されている。物理的には、Gigabit Ethernet および Gigabit レイヤ 3 スイッチによって構成されている。
3. 光ファイバーによるインターネット接続
   NTT 東日本の B フレッツニューファミリータイプおよび NTT PC コミュニケーションズの InfoSphere IP8 プランを用いてインターネットに接続されているネットワークである。最大 100Mbps、常時約 10Mbps 程度の通信が可能である。
4. PHS によるインターネット接続

株式会社ウィルコムの PHS 網および IIJ の接続サービスを用いてインターネットに接続されているネットワークである。最大 256Kbps、常時約 32Kbps〜64Kbps 程度の通信が可能である。

5.2 遅延測定実験

遅延測定実験では、まず PC1-GATE 間および PC2-GATE 間の RTT を測定し、次に PC1 と PC2 の間で GATE を経由して本システムによるコネクションを確立して、そのコネクション上でデータを送受信することにより RTT を測定した。

5.2.1 遅延測定の方法

本システムを用いない場合の 2 台のコンピュータの間の RTT の測定は、OS に付属の ping コマンドを用いて ICMP Echo Request メッセージ (データは 1 byte) を送信してから、応答パケットである ICMP Echo Response メッセージを受信するまでの時間を 10 回測定し、その平均値を算出する方法で行った。

なお、Windows に付属の ping コマンドでは、結果の精度は 1 ミリ秒であり、それ未満の場合は **1 ミリ秒未満**と表示されるため、正確な値がわからない。そこで、NDIS レイヤに挿入するパケットキャプチャプログラムである WinPcap [16] を用いて ICMP パケットをキャプチャし、キャプチャ時に精度の高い内部時計 [17] を用いて測定される時刻をもとに、RTT を算出した。

本システムを用いてコネクションが確立された 2 つのアプリケーション間のコネクション上での RTT の測定は、片方を発信側とすると、発信側から受信側に対して現在時刻をマイクロ秒精度で表した 64 bit の整数値 (8 bytes) を送信し、これを受信した受信側は直ちにそのままそのデータを返信することにより、発信側が返信されたデータ内に含まれる整数値と、返信データを受け取った瞬間に取得した時刻との間の差を求める方法で行うこととした。精度は 1 マイクロ秒以下である。これを 10 回測定し、その平均値を算出する方法で行った。

5.2.2 遅延測定の評価方法

本遅延測定実験においては、LAN 環境において、まず PC1-GATE 間および PC2-GATE 間の RTT を測定する。次に、PC1 と PC2 の間でゲートウェイである GATE を経由して本システムによりコネクションを確立し、PC1 と PC2 の間で遅延測定を行う。遅延測定の評価方法を、図 29 に示す。その際、GATE に接続する前に一旦コントローラである CONTROLLER に接続して接続要求を行うが、コネクションを確立した後は、CONTROLLER は実験にとって無関係となる。すなわち、表 7 のような実験結果を得ることができることになる。

表 7 遅延測定実験によって得ることができる結果

記号	測定方法	通信者
A	ICMP による RTT 測定	PC1 <--> GATE
B	ICMP による RTT 測定	PC2 <--> GATE
C	本システム上で確立されたコネクション上での RTT 測定	PC1 <--> GATE <--> PC2 (PC1 と PC2 間で GATE を経由してコネクションを確立)

上記の 3 つの測定結果から、以下の式により、本システムによって発生した遅延が算出できる。

本システムによって発生した遅延 = (C – (A + B)) ÷ 2

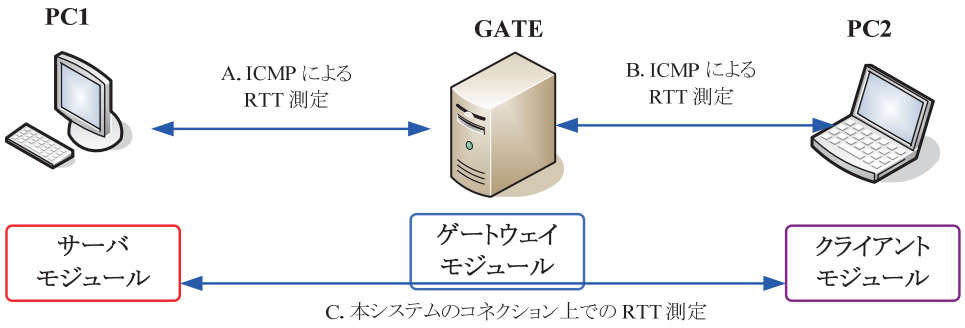

図 29 遅延測定の評価方法

5.2.3 実験結果

PC1、PC2、CONTROLLER および GATE の 4 台のコンピュータを同一の Gigabit Ethernet セグメントに接続して実験を行ったところ、表 8 のような結果となった。

表 8 遅延測定実験の結果

記号	測定方法	通信者	結果 (ミリ秒)
A	ICMP による RTT 測定	PC1 <--> GATE (通信回線: LAN)	0.044
B	ICMP による RTT 測定	PC2 <--> GATE (通信回線: LAN)	0.044
C	本システム上で確立されたコネクション上での RTT 測定	PC1 <--> GATE <--> PC2 (PC1 と PC2 間で GATE を経由してコネクションを確立)	0.511
D	本システムによって発生した遅延 = (C – (A + B)) ÷ 2		0.212

5.2.4 考察

実験結果により、本システムを用いて確立されたコネクション上で通信を行う場合は、そのコネクションのためのデータが物理的に流れるパス上で本システムを用いずに直接通信を行う場合と比較して、ごくわずかの遅延が生じることがわかった。本システムによって発生する遅延の原因は、ゲートウェイおよびアプリケーション側のライブラリで行われる SSL のデータ暗号化/解読、カプセル化/カプセル化解除などの処理によって発生する CPU 等における処理時間であると推測できる。

しかし、本システムによって発生したこの遅延は 0.212 ミリ秒である。システムのプログラムによって生じる遅延は、途中のネットワークの種類によって影響を受けないので、通常 RTT が数ミリ秒以上あるようなインターネット環境では、上記程度の遅延は問題とならない。

結論として、本システムを用いることによって発生する遅延は小さいことが評価できた。

5.2.5 補足

本システムは、インターネット上に設置されているゲートウェイシステムを中継して、クライアントとサーバ間が通信を行う仕組みである。第 1 章で述べたような、インターネットに高速で接続されているバックボーンに近いネットワーク上にゲートウェイシステムを設置している場合は、インターネットに別々の回線で接続されている 2 台のコンピュータ間で本システムを用いて通信を行う際に発生する遅延と本システムを用いずに直接通信を行う際に発生する遅延との差は小さい。

たとえば、コンピュータ 1 が ISP1 に、コンピュータ 2 が ISP2 に接続されている場合、コンピュータ 1 とコンピュータ 2 の間で直接通信を行う場合は、ISP1 <--> バックボーン回線 <--> ISP2 のようにパケットがルーティングされる。本システムを用いて通信を行う場合は、ISP1 <--> バックボーン回線 <--> ゲートウェイシステム <--> バックボーン回線 <--> ISP2 のようにパケットがルーティングされる。このとき、バックボーン回線が高速で、ゲートウェイシステムが高速回線でバックボーンに接続されていると想定すると、主な遅延の発生箇所は、ISP1 および ISP2 のネットワーク内である。したがって、直接通信を行う場合と、本システムのゲートウェイを経由して通信を行う場合は、遅延の差は小さい (図 30)。

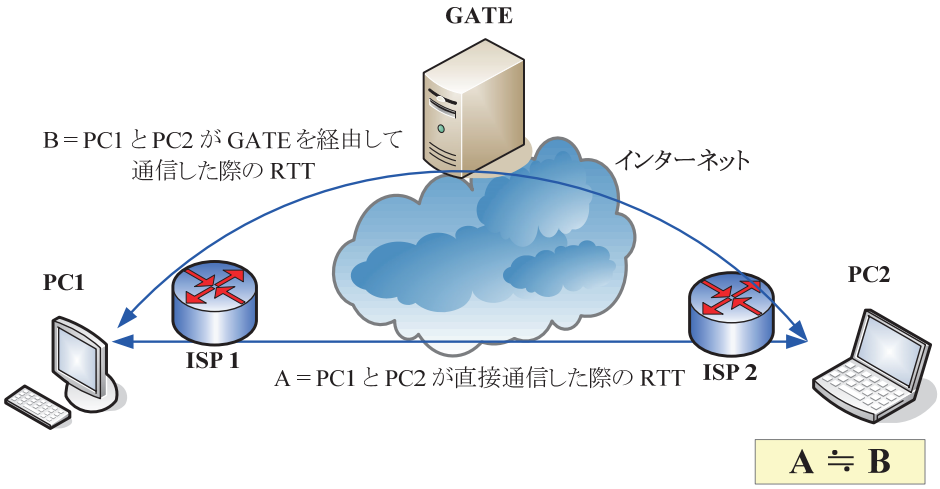

図 30 PC1 と PC2 が別々の ISP に接続している場合

しかし、コンピュータ 1 とコンピュータ 2 がインターネットを経由せずに通信できる場所にある場合、すなわち同じ LAN 経由でインターネットに接続されていたり、別々の場所にあっても同一の ISP によってインターネットに接続されていたりするような場合は、コンピュータ間で直接通信を行おうとすると、インターネットまでデータが出て行く必要がないので、非常に低遅延で通信を行うことができる。このような環境では、本ゲートウェイシステムを経由して通信をすると、遅延が相対的に大きくなり、不利となることがある (図 31)。

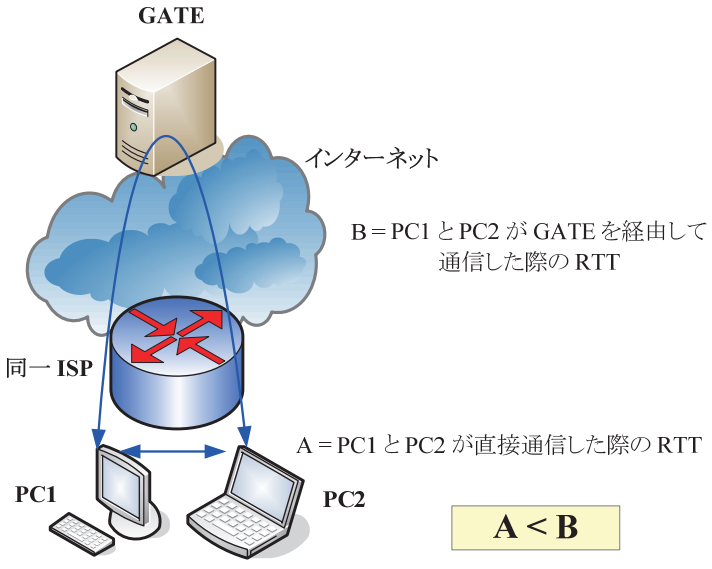

図 31 PC1 と PC2 が同一の ISP に接続している場合

5.3 スループット測定実験

スループット測定実験では、まず PC1-GATE 間および PC2-GATE 間の TCP 通信における

スループットを、アップロード方向・ダウンロード方向を別々に測定し、次に PC1 と PC2 の間で GATE を経由して本システムによるコネクションを確立して、そのコネクション上でデータを送受信することによりスループットを測定した。

5.3.1 スループット測定の方法

本システムを用いない場合の 2 台のコンピュータの間のスループットの測定は、2 台のコンピュータのうち、1 台を TCP におけるサーバ、もう 1 台を TCP におけるクライアントとし、この間で TCP コネクションを確立した後、その TCP コネクション上でデータを一方からもう一方に対して 120 秒間送信することにより行った。受信側で最初のデータを受信してから 120 秒間が経過したときまでに受信したデータの合計バイト数を 120 で割り 8 を乗じたものをスループットとした。

本システムを用いてコネクションが確立された 2 つのアプリケーション間のコネクション上でのスループットの測定は、片方のアプリケーションからもう一方に対してコネクション上で 120 秒間送信することにより行った。受信側で最初のデータを受信してから 120 秒間が経過したときまでに受信したデータの合計バイト数を 120 で割り 8 を乗じたものをスループットとした。

この方法により、本システムを用いない場合のコンピュータの間のスループットの測定と本システムを用いてコネクションが確立された 2 つのアプリケーション間のコネクション上でのスループットの測定の双方とも、それぞれ両方向 (アップロード方向/ダウンロード報告) について別々にスループットを測定した。

なお、各コンピュータの TCP スタックにおけるウインドウサイズの値は 65,535 bytes を用いた。

5.3.2 スループット測定の評価方法

本スループット測定実験においては、それぞれのネットワーク環境において、まず PC1-GATE 間および PC2-GATE 間のスループットを測定する。

次に、PC1 と PC2 の間でゲートウェイである GATE を経由して本システムによりコネクションを確立し、PC1 と PC2 の間でスループット測定を行う。スループット測定の評価方法を、図 32 に示す。その際、GATE に接続する前に一旦コントローラである CONTROLLER に接続して接続要求を行うが、コネクションを確立した後は、CONTROLLER は実験にとって無関係となる。

すなわち、各ネットワーク環境において、表 9 のような実験結果を得ることができることになる。

表 9 スループット測定実験によって得ることができる結果

記号	測定方法	通信者および通信方向
A	TCP による物理的なスループット測定	PC1 <-- GATE
B	TCP による物理的なスループット測定	PC1 --> GATE

記号	測定方法	通信者および通信方向
C	TCP による物理的なスループット測定	PC2 <-- GATE
D	TCP による物理的なスループット測定	PC2 --> GATE
E	本システム上で確立されたコネクション上でのスループット測定	PC1 <-- GATE <-- PC2 (PC1 と PC2 間で GATE を経由してコネクションを確立)
F	本システム上で確立されたコネクション上でのスループット測定	PC1 --> GATE --> PC2 (PC1 と PC2 間で GATE を経由してコネクションを確立)

上記の 6 つの測定結果から、以下の式により、本システムによって発生したスループット低下を算出することができる。

本システムによって発生したスループット低下 (PC1 <-- PC2 方向)
= MIN(A, D) - E

本システムによって発生したスループット低下 (PC1 --> PC2 方向)
= MIN(B, C) - F

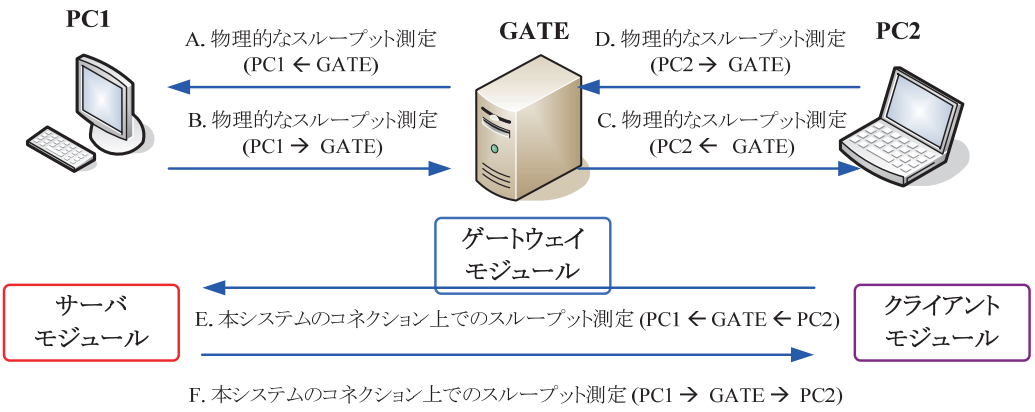

図 32 スループット測定の評価方法

5.3.3 LAN での実験

PC1、PC2、CONTROLLER および GATE の 4 台のコンピュータを同一の Gigabit Ethernet セグメントに接続して実験を行ったところ、表 10 のような結果となった。

表 10 LAN におけるスループット測定実験結果

記号	測定方法	通信者および通信方向	スループット (bps)
A	TCP による物理的なスループ	PC1 <-- GATE	37,109,104

記号	測定方法	通信者および通信方向	スループット (bps)
	ット測定		
B	TCP による物理的なスループット測定	PC1 --> GATE	35,913,728
C	TCP による物理的なスループット測定	PC2 <-- GATE	36,804,670
D	TCP による物理的なスループット測定	PC2 --> GATE	35,344,861
E	本システム上で確立されたコネクション上でのスループット測定	PC1 <-- GATE <-- PC2 (PC1 と PC2 間で GATE を経由してコネクションを確立)	23,608,352
F	本システム上で確立されたコネクション上でのスループット測定	PC1 --> GATE --> PC2 (PC1 と PC2 間で GATE を経由してコネクションを確立)	23,694,785
G	本システムによって発生したスループット低下 (PC1 <-- PC2 方向) = MIN(A, D) – E		11,736,509
H	本システムによって発生したスループット低下 (PC1 --> PC2 方向) = MIN(B, C) - F		12,218,943

5.3.4 学内ネットワークでの実験

CONTROLLER および GATE の 2 台のコンピュータを筑波大学の学術情報メディアセンターのネットワークに接続し、グローバル IP アドレスを割り当てた。また、PC1 および PC2 の 2 台のコンピュータを筑波大学の産学リエゾン共同研究センターのネットワークに接続し、グローバル IP アドレスを割り当てた。

この環境で実験を行ったところ、表 11 のような結果となった。

表 11 学内ネットワークにおけるスループット測定実験結果

記号	測定方法	通信者および通信方向	スループット (bps)
A	TCP による物理的なスループット測定	PC1 <-- GATE	13,491,211
B	TCP による物理的なスループット測定	PC1 --> GATE	12,027,636
C	TCP による物理的なスループ	PC2 <-- GATE	14,022,763

記号	測定方法	通信者および通信方向	スループット (bps)
	ット測定		
D	TCP による物理的なスループット測定	PC2 --> GATE	12,921,804
E	本システム上で確立されたコネクション上でのスループット測定	PC1 <-- GATE <-- PC2 (PC1 と PC2 間で GATE を経由してコネクションを確立)	11,464,839
F	本システム上で確立されたコネクション上でのスループット測定	PC1 --> GATE --> PC2 (PC1 と PC2 間で GATE を経由してコネクションを確立)	10,177,794
G	本システムによって発生したスループット低下 (PC1 <-- PC2 方向) = MIN(A, D) - E		1,456,965
H	本システムによって発生したスループット低下 (PC1 --> PC2 方向) = MIN(B, C) - F		1,849,842

5.3.5 光ファイバーによるインターネットでの実験

CONTROLLER および GATE の 2 台のコンピュータを筑波大学の学術情報メディアセンターのネットワークに接続し、グローバル IP アドレスを割り当てた。また、PC1 を筑波大学の産学リエゾン共同研究センターのネットワークに接続し、グローバル IP アドレスを割り当て、PC2 を光ファイバーによるインターネット接続回線に接続し、グローバル IP アドレスを割り当てた。

つまり、5.3.4 と比較すると、PC1-GATE 間の回線は変更されていないが、PC2-GATE 間の回線は光ファイバーによるインターネット接続回線となり、スループットが低下することが事前に予想される。

この環境で実験を行ったところ、表 12 のような結果となった。

表 12 光ファイバーにおけるスループット測定実験結果

記号	測定方法	通信者および通信方向	スループット (bps)
A	TCP による物理的なスループット測定	PC1 <-- GATE	13,491,211
B	TCP による物理的なスループット測定	PC1 --> GATE	12,027,636
C	TCP による物理的なスループット測定	PC2 <-- GATE	5,316,989
D	TCP による物理的なスループ	PC2 --> GATE	5,611,949

記号	測定方法	通信者および通信方向	スループット (bps)
	ット測定		
E	本システム上で確立されたコネクション上でのスループット測定	PC1 <-- GATE <-- PC2 (PC1 と PC2 間で GATE を経由してコネクションを確立)	4,953,258
F	本システム上で確立されたコネクション上でのスループット測定	PC1 --> GATE --> PC2 (PC1 と PC2 間で GATE を経由してコネクションを確立)	4,961,456
G	本システムによって発生したスループット低下 (PC1 <-- PC2 方向) = MIN(A, D) – E		658,691
H	本システムによって発生したスループット低下 (PC1 --> PC2 方向) = MIN(B, C) – F		355,533

5.3.6 PHS によるインターネットでの実験

CONTROLLER および GATE の 2 台のコンピュータを筑波大学の学術情報メディアセンターのネットワークに接続し、グローバル IP アドレスを割り当てた。また、PC1 を筑波大学の産学リエゾン共同研究センターのネットワークに接続し、グローバル IP アドレスを割り当て、PC2 を PHS によるインターネット接続回線に接続し、グローバル IP アドレスを割り当てた。

つまり、5.3.5 と比較すると、PC1-GATE 間の回線は変更されていないが、PC2-GATE 間の回線は PHS によるインターネット接続回線となり、スループットがさらに低下することが事前に予想される。

この環境で実験を行ったところ、表 13 のような結果となった。

表 13 PHS におけるスループット測定実験結果

記号	測定方法	通信者および通信方向	スループット (bps)
A	TCP による物理的なスループット測定	PC1 <-- GATE	13,491,211
B	TCP による物理的なスループット測定	PC1 --> GATE	12,027,636
C	TCP による物理的なスループット測定	PC2 <-- GATE	69,383
D	TCP による物理的なスループット測定	PC2 --> GATE	72,398
E	本システム上で確立されたコ	PC1 <-- GATE <-- PC2	51,887

記号	測定方法	通信者および通信方向	スループット (bps)
	ネクション上でのスループット測定	(PC1 と PC2 間で GATE を経由してコネクションを確立)	
F	本システム上で確立されたコネクション上でのスループット測定	PC1 --> GATE --> PC2 (PC1 と PC2 間で GATE を経由してコネクションを確立)	58,280
G	本システムによって発生したスループット低下 (PC1 <-- PC2 方向) = MIN(A, D) – E		17,496
H	本システムによって発生したスループット低下 (PC1 --> PC2 方向) = MIN(B, C) – F		14,118

5.3.7 考察

上記の 4 個の実験結果をまとめると、表 14 のようになった。

表 14 実験結果のまとめ

実験名	PC1-GATE 間のスループットおよび PC2-GATE 間のスループットのうち値が低いもの	PC1-PC2 間で GATE を経由して本システムによるコネクションを確立した際のコネクション上でのスループット	本システムによって発生したスループット低下 (スループット低下率)
LAN での実験 (PC1 <-- PC2 方向)	35,344,861	23,608,352	11,736,509 (33.2%)
LAN での実験 (PC1 --> PC2 方向)	35,913,728	23,694,785	12,218,943 (34.0%)
学内ネットワークでの実験 (PC1 <-- PC2 方向)	12,921,804	11,464,839	1,456,965 (11.3%)
学内ネットワークでの実験 (PC2 <-- PC1 方向)	12,027,636	10,177,794	1,849,842 (15.3%)
光ファイバーでの実験 (PC1 <-- PC2 方向)	5,611,949	4,953,258	658,691 (11.7%)
光ファイバーでの実験 (PC1 --> PC2 方向)	5,316,989	4,961,456	355,533 (6.7%)
PHS での実験	72,398	51,887	17,496

実験名	PC1-GATE 間のスループットおよび PC2-GATE 間のスループットのうち値が低いもの	PC1-PC2 間で GATE を経由して本システムによるコネクションを確立した際のコネクション上でのスループット	本システムによって発生したスループット低下 (スループット低下率)
(PC1 <-- PC2 方向)			(24.2%)
PHS での実験 (PC1 --> PC2 方向)	69,383	58,280	14,118 (20.3%)

(単位: bps)

これによると、回線の種類などによって大小はあるものの、本システムのコネクションで通信を行うことにより、物理的な通信回線上で TCP コネクションを用いる場合と比較して、10%〜30%程度のスループットの低下がみられることがわかる。

本システムのコネクション上を流れるデータは、実際には SSL によって TCP にカプセル化されて伝送される。TCP のスループット低下の原因の一つとして、遅延の発生が挙げられる。しかしながら、5.2 の実験結果より、本システムを使用する場合においてほとんど遅延は発生していないので、スループットの低下の主な要因が遅延ではないことが推測できる。

そこで、その他の可能性を考える。スループットの低下の原因の一つに、送受信しようとするデータをカプセル化/カプセル化解除したり、暗号化/解読したりする際に発生する CPU 処理やメモリ転送処理のための時間の増加の可能性を挙げることができる。既存の物理的なネットワーク上に TCP コネクションを張って通信を行うネットワークソフトウェアで、データの SSL による暗号化の有効/無効を切り替えることができるソフトウェアにおいては、暗号化を有効にすると、無効にしている場合と比較して、20%〜50%程度のスループット低下が発生するというデータがある [18]。

また、その他の原因として、ゲートウェイのプログラム内部において、サーバが接続されているコネクションと、クライアントが接続されているコネクションとの間で、伝送データを交換する際に、一時的にデータをメモリにストアし、その後にフォワードするという処理を行っている部分がボトルネックとなっている可能性を挙げることができる。例えば、ネットワークのスイッチにおいては、ストアアンドフォワード方式は一時的にバッファにデータを格納するための処理時間のため、カットスルー方式等と比較して低速であるというデータがある。これと同等の原因によって、スループット低下が発生している可能性がある。

実験結果では、スループットの低下率は、同一セグメント内での実験時という極端な場合を除くと、最大でも 20%程度であり、システムを利用する上で致命的な問題ではないと考えることもできる。しかしながら、ユーザにとっては、スループットは高ければ高いほど望ましい場合が多いので、このスループット低下の原因を探求することが必要である。

5.3.8 補足

インターネットに高速で接続されているバックボーンに近いネットワーク上にゲートウェイシステムを設置している場合は、インターネットに別々の回線で接続されている 2 台のコンピュータ間で本システムを用いて通信を行う際のスループットは、本システムを用いずに直接通信を行う際のスループットと比較して、差は小さい。

たとえば、コンピュータ 1 が ISP1 に、コンピュータ 2 が ISP2 に接続されている場合、コンピュータ 1 とコンピュータ 2 の間で直接通信を行う場合は、ISP1 <--> バックボーン回線 <--> ISP2 のようにパケットがルーティングされる。本システムを用いて通信を行う場合は、ISP1 <--> バックボーン回線 <--> ゲートウェイシステム <--> バックボーン回線 <--> ISP2 のようにパケットがルーティングされる。このとき、バックボーン回線が高速で、ゲートウェイシステムが高速回線でバックボーンに接続されていると想定すると、主なスループット低下原因の発生箇所は、ISP1 および ISP2 のネットワーク内である。したがって、直接通信を行う場合と、本システムのゲートウェイを経由して通信を行う場合は、スループットの差は小さい (図 33)。

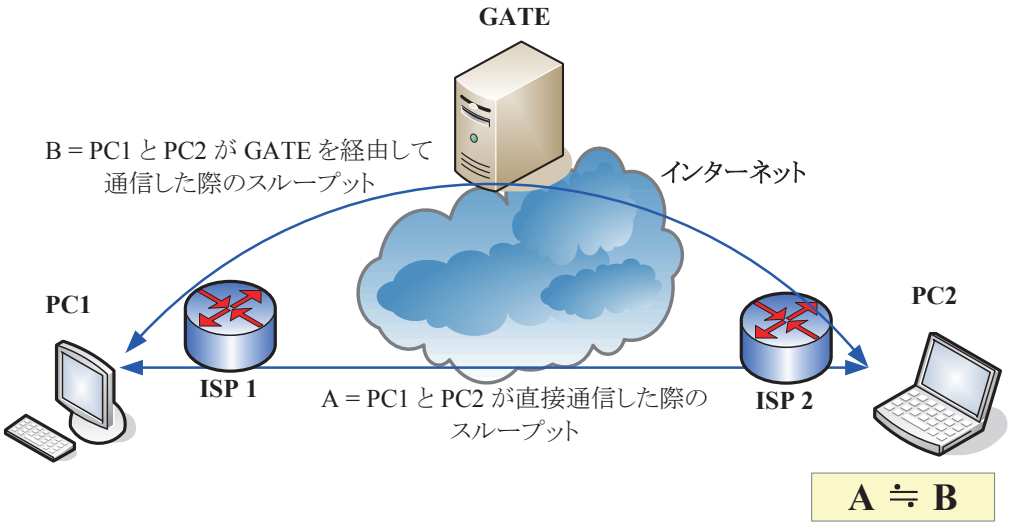

図 33 PC1 と PC2 が別々の ISP に接続している場合

しかし、コンピュータ 1 とコンピュータ 2 がインターネットを経由せずに通信できる場所にある場合、すなわち同じ LAN 経由でインターネットに接続されていたり、別々の場所にあっても同一の ISP によってインターネットに接続されていたりするような場合は、コンピュータ間で直接通信を行おうとすると、インターネットまでデータが出て行く必要がないので、非常に高スループットで通信を行うことができる。このような環境では、本ゲートウェイシステムを経由して通信をすると、スループットが相対的に低下し、不利となることがある (図 34)。

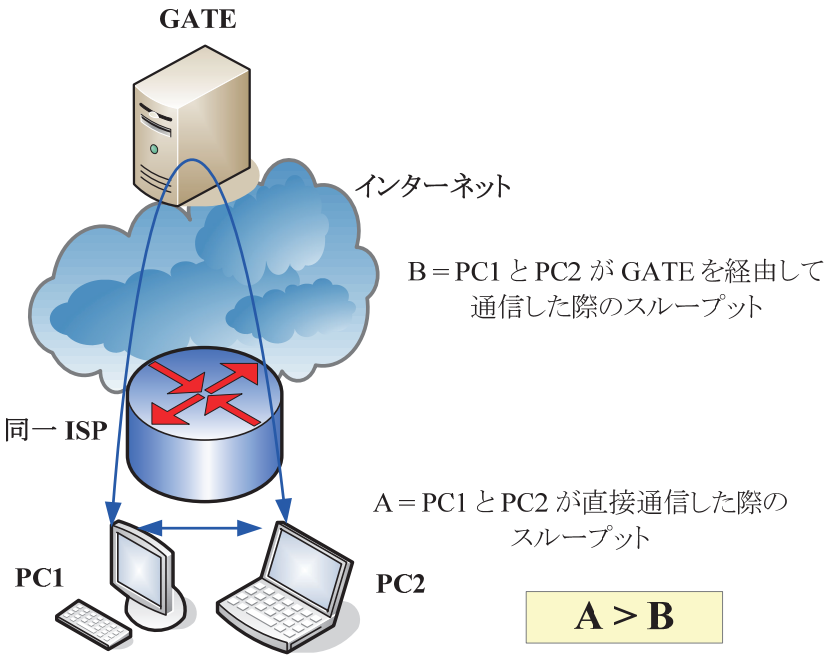

図 34 PC1 と PC2 が同一の ISP に接続している場合

第 6 章 実用的アプリケーションによる実証実験

本章では、本システムをより実用的なアプリケーションに組み込み、インターネット上で多数のユーザに対して配布し、同時にインターネット上でゲートウェイシステムを稼動させ、実際に通信に利用してもらった結果について述べる。

6.1 目的

この実証実験の目的は、以下のとおりである。

- インターネット上の高速なバックボーンで接続された場所に、数台のゲートウェイおよび 1 台のコントローラを実際に設置し、利用可能にする。
- インターネット上で、不特定多数のユーザに、本システムのライブラリを用いて通信を行うアプリケーションを無償で配布し、実際に利用してもらう。
- その結果、常に 1,000 セッション以上のサーバセッションがゲートウェイシステムに接続されている状態を目指す。そのような状態でシステムが安定して稼動するかどうか検証する。
- ユーザ数 (サーバデータベースへの MSID の登録数) としては、5,000 人程度を目指す。
- コントローラの CPU 使用率などの負荷状況、ゲートウェイシステムにおける同時接続サーバセッション数、各ゲートウェイの接続サーバセッション数、ゲートウェイシステムが処理しているスループットなどの使用状況を随時記録しグラフ化する。

6.2 アプリケーションの実装および配布

本システムは、多数のユーザによって同時に使用されることを想定して設計および実装したものである。そこで、実際に多数のユーザに本システムを使用してもらい、ゲートウェイシステムに負荷がかかっても支障が発生しないかどうかを検証するとともに、どのような現象が発生するのかを調べるため、より実用的なアプリケーションを本システムのサーバおよびクライアントのライブラリを用いて実装し、インターネット上で配布したのち、2 週間かけて実証実験を行った。

6.2.1 アプリケーションの種類

本実証実験の目的を達成するためには、できるだけ多数のユーザに便利に使ってもらうことが重要である。実証実験のためのアプリケーションとして、**コンピュータのデスクトップへリモートアクセスし、離れている場所からでもコンピュータを自由に遠隔操作することができるソフトウェア**を実装する。この機能の概要を、図 35 に図示する。

このアプリケーションの目的を、以下に示す。

1. サーバをあらかじめインストールしておいたコンピュータのデスクトップに、インターネット経由で、クライアントを用いていつでもリモートアクセスし、自由に遠隔操作を行うことができる。
2. サーバをインストールしたPCとクライアントをインストールしたPCの双方が、NATやプロキシサーバ等の内側にあり、プライベート IP アドレスしか持っていなくても、簡単に利用することができる。
3. 複雑なネットワークの知識や、ファイアウォールやルータ等の設定変更、固定グローバル IP アドレスの取得、および DDNS 等の利用が一切不要である。
4. すべての通信内容は、本システムによって暗号化され、インターネットの回線上を伝送される。
5. サーバ側によるクライアント認証 (ユーザ認証) が実施される。
6. 通信の内容としては、既存のリモートデスクトップのためのプロトコルをトンネリングして (ポート転送して) 通信する。
7. 常時、管理者権限でサーバソフトウェアを起動しておくことがセキュリティ上不安なユーザのために、一般ユーザ権限でもサーバソフトウェアを起動しておくことができるようにする。

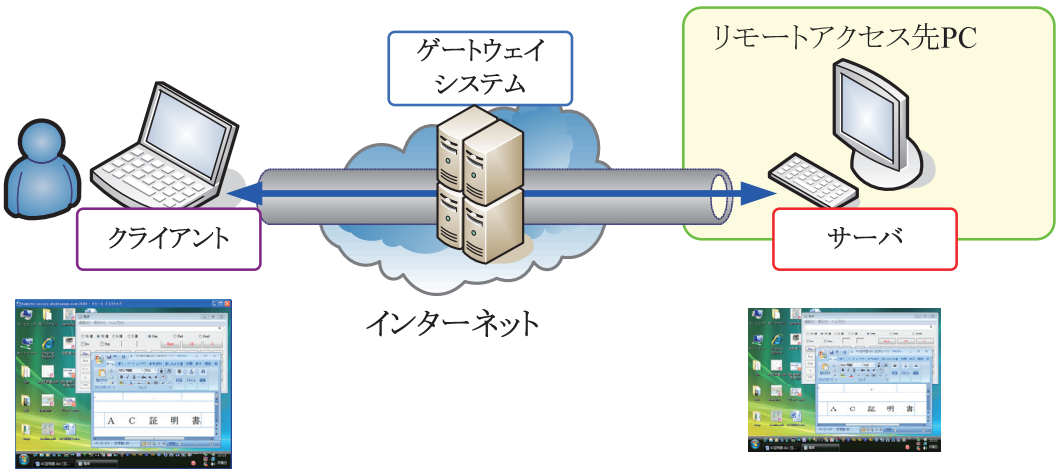

図 35 デスクトップへリモートアクセスするソフトウェア

6.2.2 Windows XP リモートデスクトッププロトコル (RDP)

Windows XP 以降のバージョンの Windows にはリモートデスクトッププロトコル (RDP) [19]およびこれに対応したリモート操作ソフトウェアのクライアントおよびサーバが搭載されている。RDP を用いることにより、ユーザは別のコンピュータから自分のコンピュータにリモートログオンして、まるでコンピュータを直接操作しているかのように、遠隔操作を行うことができる。

しかし、RDP は通常の TCP コネクションを確立して通信を行うので、NAT の内側にあるコンピュータに、その NAT の外側にあるコンピュータから接続することができない。

そこで、アプリケーションに、RDP の使用する TCP コネクションを転送するための機能

を追加し、RDP の通信データを本システムにおけるコネクションを使用して転送することにより、NAT の内側にあるコンピュータに対して、NAT の外側にあるコンピュータからの RDP 接続を実現する。

6.2.3 VNC プロトコル

VNC [20] は RDP と同様に遠隔のコンピュータをリモート操作するためのプロトコルである。RDP と同じく、TCP コネクションを確立することにより通信を行うので、NAT 越えの問題を解決するために、VNC の使用する TCP コネクションを本システムにおけるコネクションを使用して転送することにより、NAT の内側にあるコンピュータに対して、NAT の外側にあるコンピュータからの VNC 接続を実現する。

6.2.4 サーバプリケーション

サーバプリケーションは、常にコネクションを待ち受ける。新しいコネクションがクライアントから接続された場合は、サーバ側で予めユーザが選択しているプロトコル (RDP もしくは VPN のいずれか) に応じた動作を行う。RDP プロトコルを使用する場合は、localhost の TCP ポート (Windows におけるデフォルトの RDP サーバの待ち受けポートは 3389 である) に対して接続する。VNC プロトコルを使用する場合は、起動した VNC サーバが待ち受ける localhost の TCP ポートに対して接続する。その後、コネクションから伝送されてきたデータをそのまま当該ポートにリレーし、当該ポートから受け取ったデータをコネクションに対して送信するという転送処理を行うソフトウェアである。ユーザインターフェイスのスクリーンショットを、図 36 に示す。

なお、本システムを使用してサーバ処理を行うアプリケーションは、一般ユーザ権限でも、管理者権限でもどちらでも動作するので、ユーザはサーバプリケーションをインストールする際に、管理者権限としてインストールを行うか、それとも一般ユーザ権限でインストールを行うかを選択することができるようにした。管理者権限でインストールすると、Windows のシステムサービスとして登録され、ユーザが誰もコンピュータにログオンしていなくてもいつでもそのコンピュータにリモートアクセスできる。一般ユーザ権限でインストールすると、そのユーザのスタートアップ項目として登録され、そのユーザがコンピュータにログオンしている間だけ、そのコンピュータにリモートアクセスできる。

6.2.5 クライアントアプリケーション

クライアントアプリケーションは、ユーザが指定した PCID を持つサーバに対してコネクションを接続する。コネクションの接続が完了したら、RDP もしくは VNC のクライアントソフトウェアを起動する。次に、localhost 上の TCP ポートのうち空いているものを 1 つ選択し、その TCP ポートを待機状態にしてから、リモートデスクトップクライアントソフトウェアをその TCP ポートに接続させる。そして、TCP ポートに対してリモートデスクトップクライアントソフトウェアが送信しようとしたデータをそのままコネクションに対して送信し、コネクションから受け取ったデータをリモートデスクトップクライアントソフト

ウェアに対して渡すという転送処理を行う。

ユーザインターフェイスのスクリーンショットを、図 37 に示す。

図 36 サーバのユーザインターフェイス

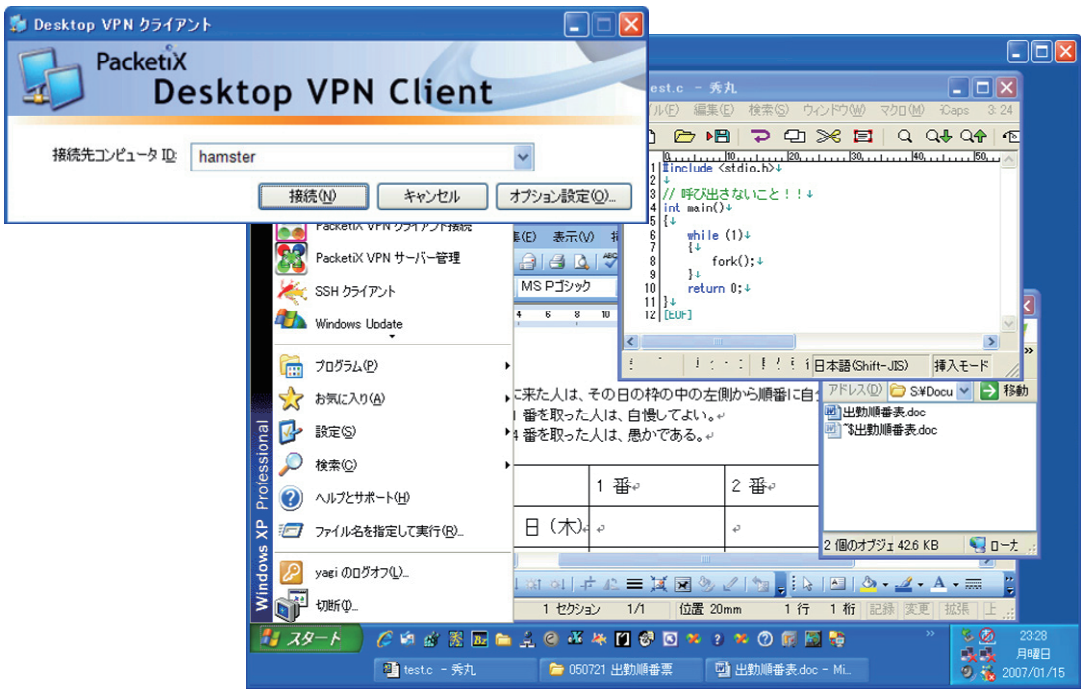

図 37 クライアントのユーザインターフェイス

6.2.6 Web サイトでの配布

サーバプリケーションおよびクライアントアプリケーションを実装した後、短期間でできるだけ多くのユーザにこれらのソフトウェアを使用してもらい、よって本システムに負荷をかけた状態を実現したいと考えたので、ソフトウェアは、筑波大学発ベンチャー企業であるソフトイーサ株式会社の Web サイトで、**Desktop VPN** [21] という名称で 2007 年 1 月 16 日から配布を開始した。

6.3 実証実験

本アプリケーションによる実証実験を行うため、アプリケーション本体を Web サイト上で配布するに先立ち、ゲートウェイシステムをインターネット上に設置した。本実証実験の目的は、このゲートウェイシステムの動作状況を記録し、そのデータを分析して、システムの性能を評価するというものである。

6.3.1 実験環境の概要

本アプリケーションの配布による実証実験の構成要素は二分することができる。一方は Web サイトで大量に配布された本アプリケーションをインストールし自らのコンピュータで使用するユーザであり、もう一方はそれらのユーザによるリモートデスクトップ通信を中継するために設置されたゲートウェイシステムである。

ユーザはインターネットを経由して匿名で本アプリケーションをダウンロードし使用するため、各ユーザのコンピュータのデータを記録・分析して評価することはできない。そのため、本実証実験における観察・評価の対象は、インターネット上に設置したゲートウェイシステムのみである。

6.3.2 ゲートウェイシステムの設置

本実験のためのゲートウェイシステムは、筑波大学の学術情報メディアセンター内のインターネット側セグメント (130.158.6.0/24) に Gigabit Ethernet で接続することにより設置した。このセグメントは、SINET (学術情報ネットワーク) を経由して、インターネットに対して最低でも 1 Gbps で接続されている。設置図を、図 38 に示す。また、設置した写真を、図 39 に示す。

ゲートウェイシステムのうち、コントローラの台数は 1 台、ゲートウェイの台数は 5 台である。それぞれ、CONTROLLER、GATE1〜GATE5 という名前を付けた。

各コンピュータは同一スペックである。その詳細を、以下に示す。

スペック等
DELL PowerEdge SC430
CPU: Pentium D 2.8GHz (2 CPU)
RAM: 512 Mbytes
LAN: Gigabit Ethernet
OS: Windows Server 2003

それぞれのコンピュータに割り当てた IP アドレスは、表 15 のとおりである。

表 15 それぞれのコンピュータに割り当てた IP アドレス

コンピュータ名	IP アドレス
CONTROLLER	130.158.6.64
GATE1	130.158.6.68
GATE2	130.158.6.69
GATE3	130.158.6.70
GATE4	130.158.6.71
GATE5	130.158.6.72

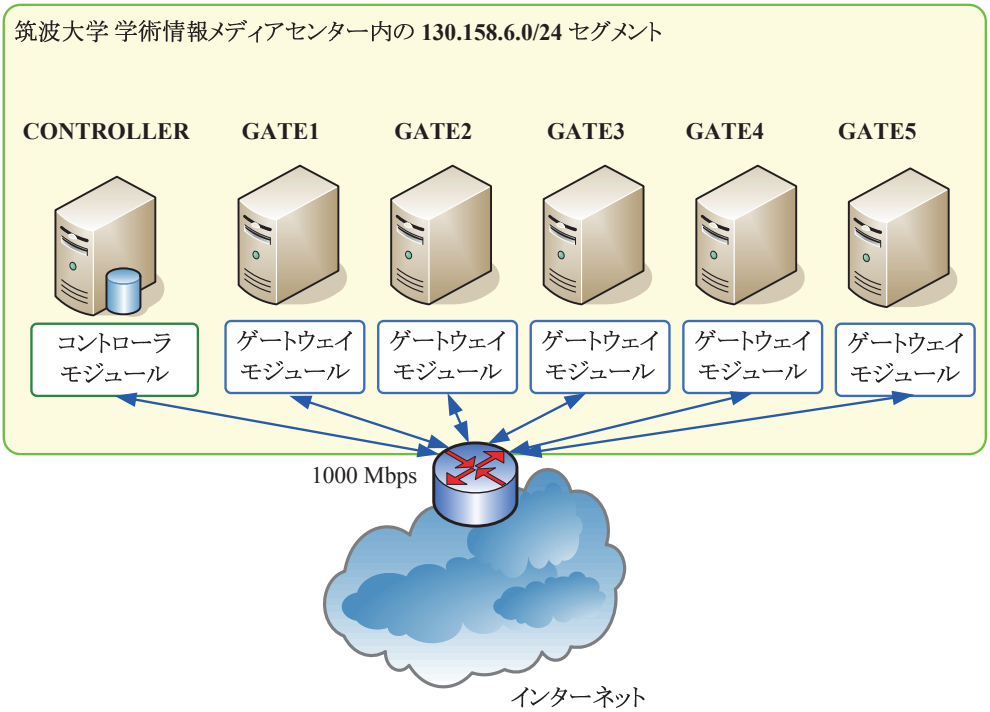

図 38 ゲートウェイシステムの各サーバコンピュータの設置図

図 39 ゲートウェイシステムの各サーバコンピュータを設置した際の写真

6.3.3 Web サイトでの配布の開始

設置したゲートウェイシステムを経由して通信を行うようにプログラミングした、ユーザ向けに配布するサーバソフトウェア **Desktop VPN サーバ**およびクライアントソフトウェア **Desktop VPN Client** を 2007 年 1 月 16 日午前 5 時に Web サイト上に掲載し、配布を開始した。

6.3.4 サーバデータベースレコード数の推移

ユーザが Desktop VPN サーバをコンピュータにインストールし起動すると、初回起動時に、そのコンピュータに対応する MSID およびそれに関連付けられたデータが格納されたレコードが、コントローラ上のサーバデータベーステーブルに作成される。このサーバデータベーステーブルのレコード数は、世界中のインターネットに接続されており Desktop VPN サーバがインストールされ 1 度以上起動されたコンピュータの台数を示す。

2007 年 1 月 16 日 (以下、同年) から 1 月 30 日までのサーバデータベースレコード数の推移をグラフにしたところ、図 40 のようになった。

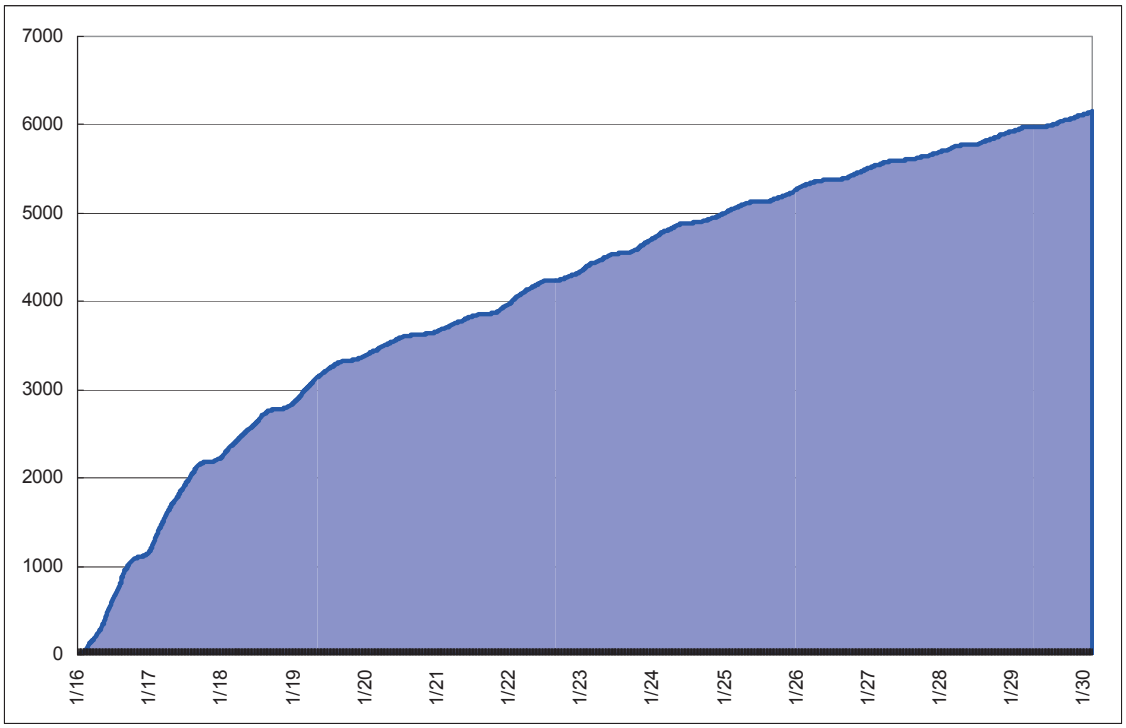

図 40 サーバデータベースレコード数の推移

これによると、サーバソフトウェアの配布を開始した 1 月 16 日から 1 週間で約 4,000 レコード、2 週間で約 6,000 レコードが登録されていることがわかる。

6.3.5 ゲートウェイシステム全体での接続サーバセッション数の推移の観測

1 月 18 日から 1 月 27 日までの 10 日間、GATE1～GATE5 の 5 台のゲートウェイに接続されているサーバセッション数の合計を 1 時間ごとに測定して記録したところ、その合計値は図 41 のようになった。

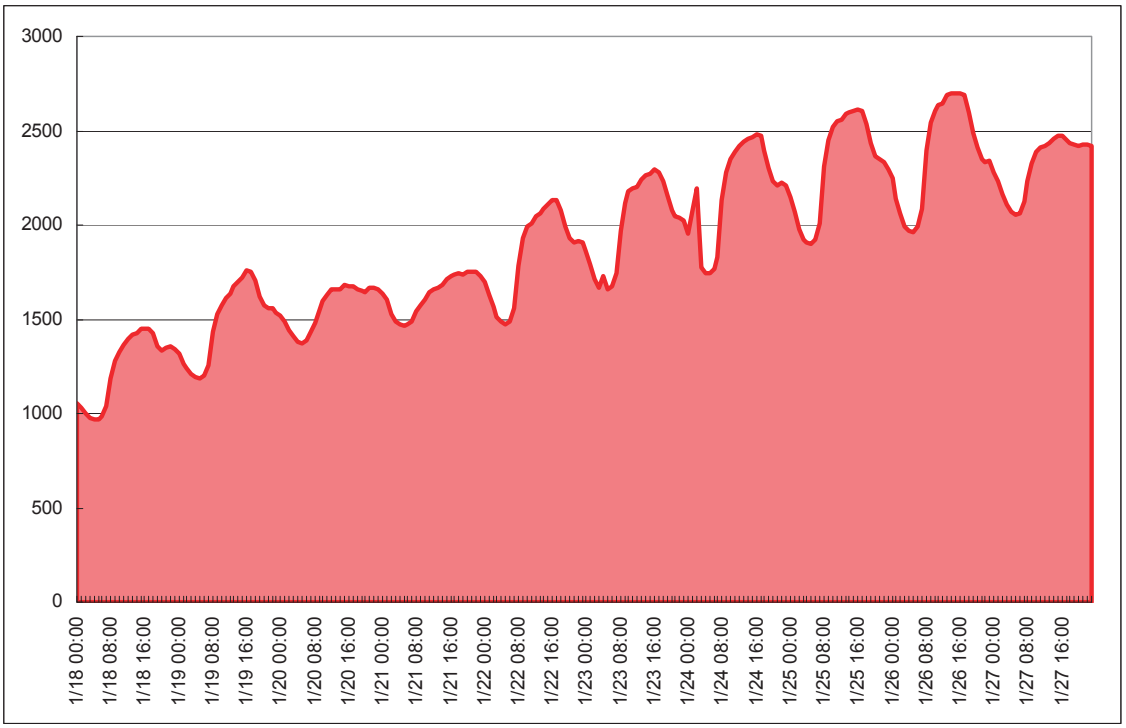

図 41 接続サーバセッション数の推移 (ゲートウェイシステム全体)

これによると、毎日、8 時頃から 16 時頃までに接続サーバセッション数が増加し、その後翌日の午前 1 時頃までかけて減少するというような規則性が見られる。また、平均約 1,900 サーバセッションが接続されていることがわかる。

6.3.6 コントローラに対するサーバセッション接続要求数の推移の観測

本システムでは、サーバは、ゲートウェイシステムに対してサーバセッションを確立しようとするときは、まずコントローラに対してサーバセッションの接続要求を行う。そこで、コントローラで 1 時間ごとに処理したサーバセッション接続要求数を測定したところ、以下のようになった。

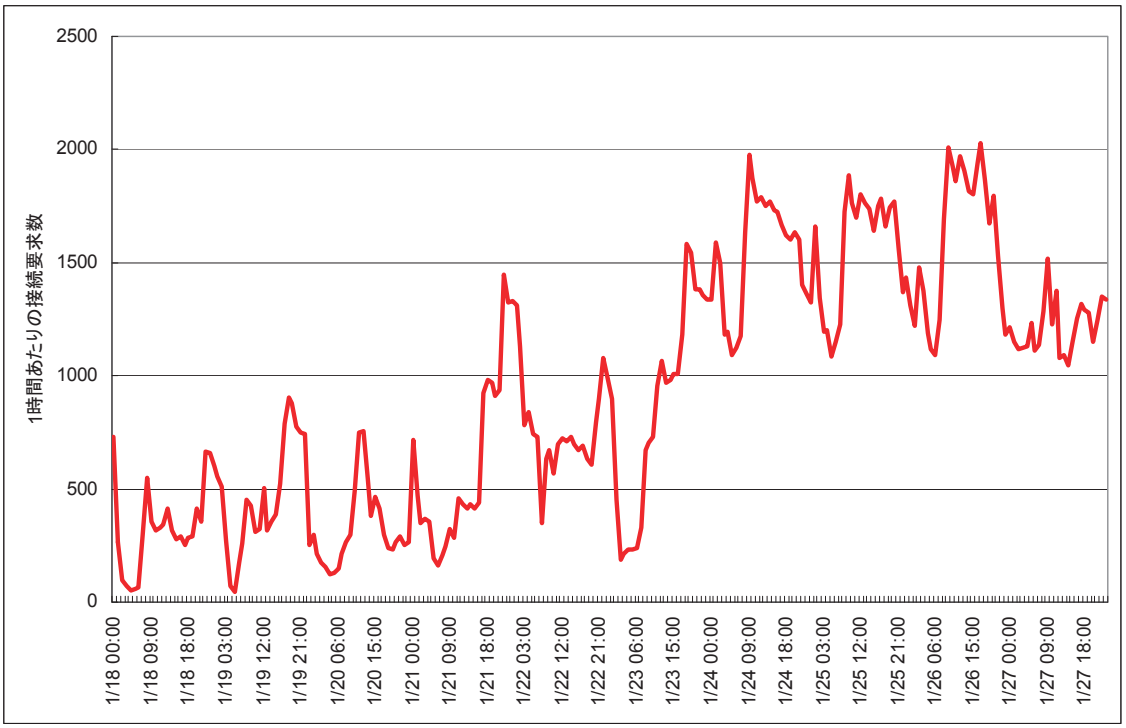

図 42 サーバセッションの接続要求数の推移

これによると、コントローラは、サーバセッションの接続要求を、最大で 1 時間あたり 2,027 回、平均で 1 時間あたり約 924.8 回処理していることがわかる。

6.3.7 コントローラの CPU 使用率の観測

1 月 25 日 0 時から 1 月 27 日 48 時までの 24 時間の間、1 時間ごとにコントローラの平均 CPU 使用率 (精度は 1.0%) を記録したところ、以下のようになった。

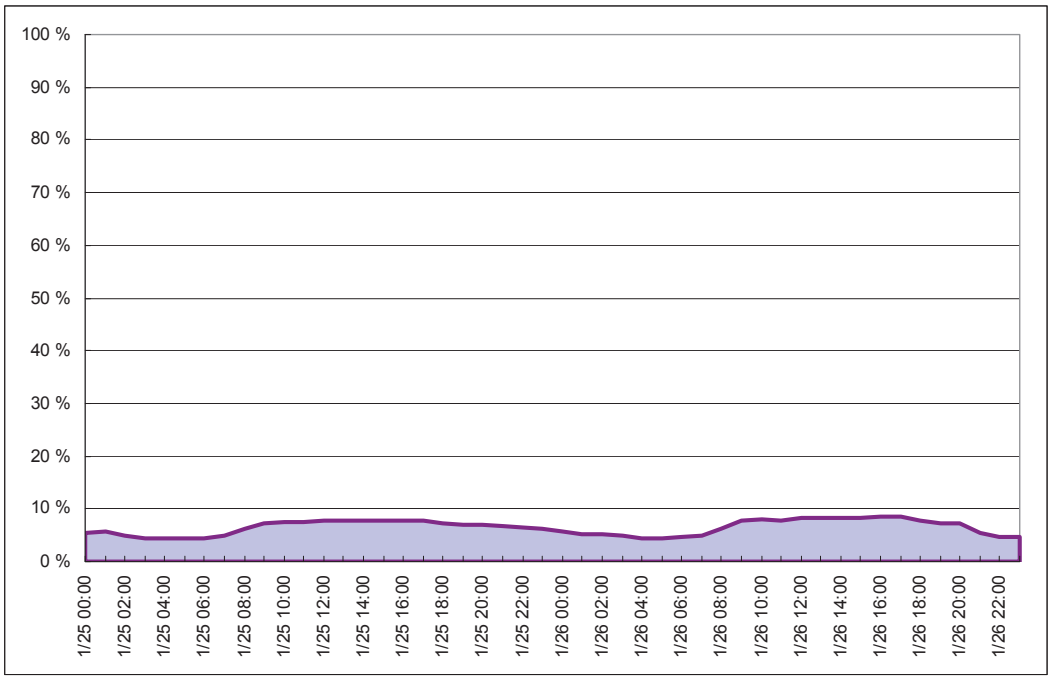

図 43 コントローラ CPU 使用率の推移

同時期におけるサーバセッションの接続要求数を、図 42 から抽出すると、図 44 のようになった。

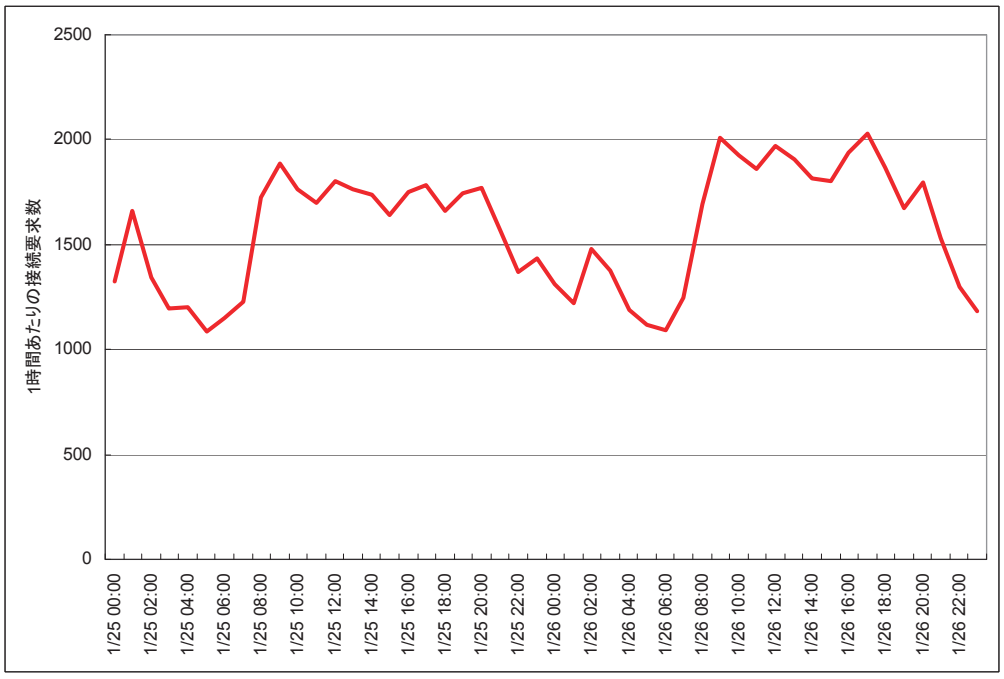

図 44 サーバセッションの接続要求数の推移 (図 43 と同一の期間)

図 43 と図 44 を比較すると、コントローラ CPU 使用率の増減は、コントローラが処理す

るサーバセッションの接続要求数の増減とともに起こっていることがわかる。

コントローラの CPU 使用率がサーバセッションの接続要求数に正比例すると仮定して、コントローラの CPU 使用率をサーバセッションの接続要求数で割ったところ、図 45 のようになった。

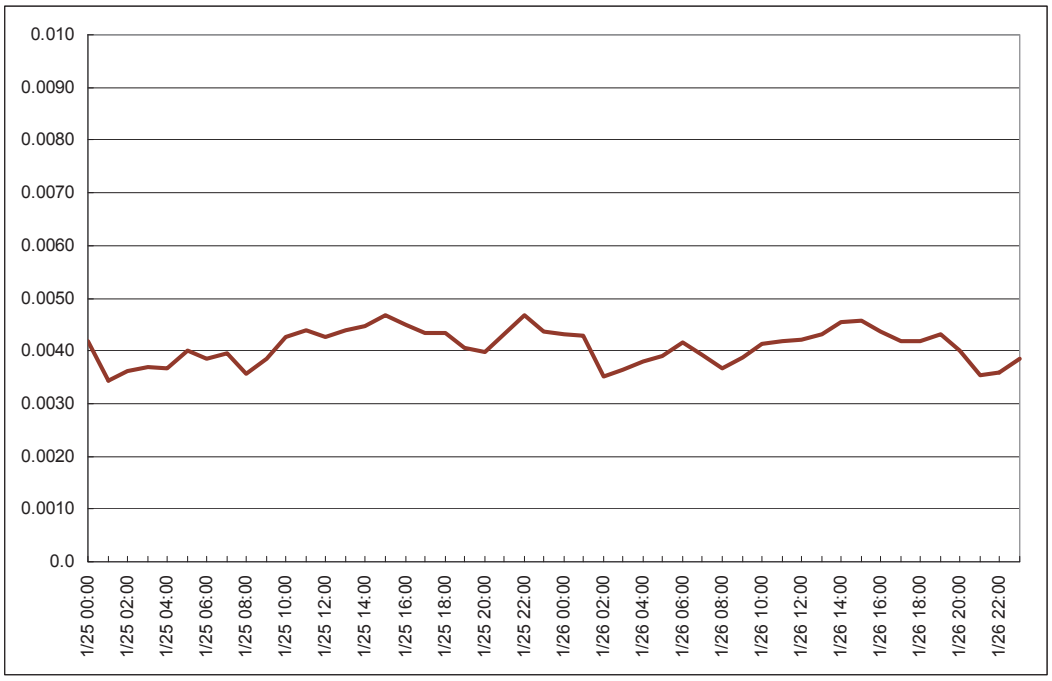

図 45 コントローラの CPU 使用率をサーバセッションの接続要求数で割った値の推移

これによると、多少の誤差はあるものの、サーバセッションの接続要求数 (1 時間あたり) に対するコントローラの CPU 使用率の比例定数は、0.004 を中心として、おおよそ 0.003〜0.005 程度に収まっている。CPU 使用率の最大値は 100%であるので、現在のコントローラのスペックでは、おおよそ 100÷0.004=25,000 サーバセッションの接続要求を 1 時間で処理することができる (1 分あたり、417 サーバセッションの接続要求を処理することができる) ということが予想できる。

6.3.8 負荷分散状況の観測

ゲートウェイシステムは 5 台のゲートウェイから構成されるが、それぞれのゲートウェイにおけるサーバセッションの接続数を 1 時間ごとに測定して記録したところ、その合計値は図 46 のようになった。

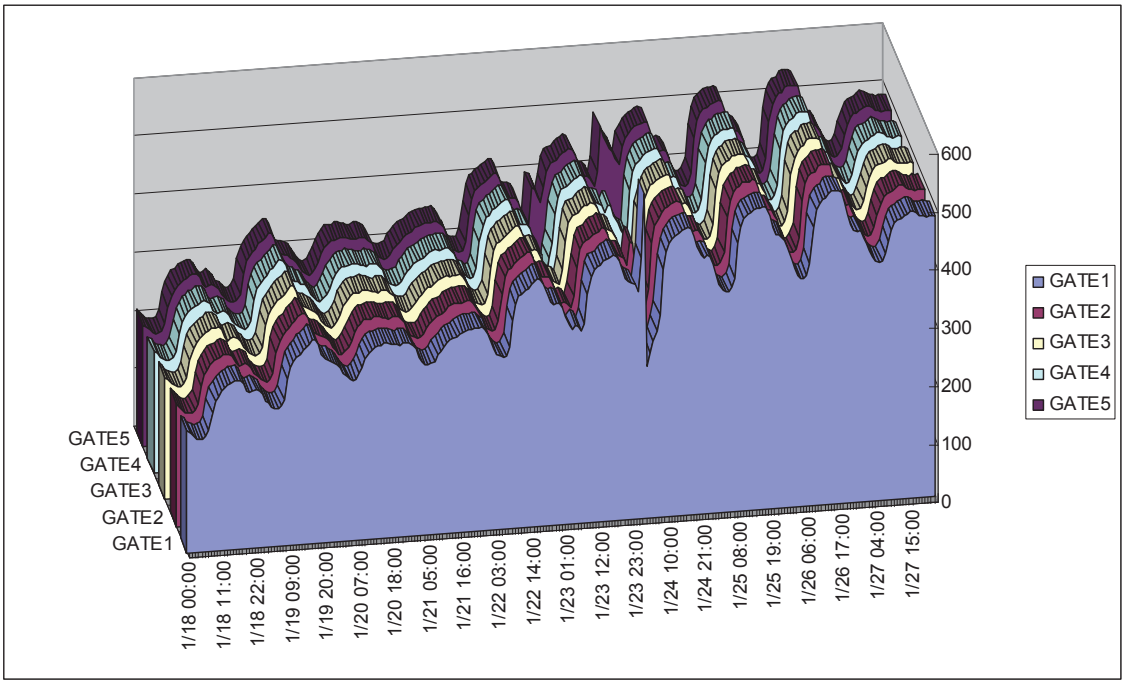

図 46 接続サーバセッション数の推移 (ゲートウェイ別)

これによると、GATE1〜GATE5 の 5 台の各ゲートウェイが、ほぼ同数のサーバセッションを処理していることがわかり、コントローラにおいてプログラミングした負荷分散処理が理想的に動作していることがわかる。

6.3.9 ゲートウェイのうち 1 台を停止させた際の挙動の観測

本システムでは、サーバは常にゲートウェイに対してサーバセッションを確立し続けようとする。サーバが接続しているゲートウェイが停止した場合は、サーバセッションが切断されるので、サーバはすぐに再度コントローラに対してサーバセッション確立要求を行うはずである。

そこで、GATE1 で動作しているゲートウェイプログラムを強制終了し、3 分後に再度起動することによって、作為的にゲートウェイのうち 1 台を停止させた場合のゲートウェイシステム全体の挙動を観測することにした。実際に、1 月 28 日の午前 3 時 45 分から 3 分後の午前 3 時 48 分の間の 3 分間、GATE1 を停止させた際の、GATE1〜GATE5 の 5 台のゲートウェイに接続されているサーバセッション数を測定すると、表 16 のようになった。

表 16 サーバセッション数の推移

時刻	GATE1	GATE2	GATE3	GATE4	GATE5	合計
午前 3 時 42 分	427	429	428	427	429	2,140
午前 3 時 43 分	427	429	429	427	429	2,141
午前 3 時 44 分	428	428	427	427	429	2,139
午前 3 時 45 分	0	428	430	427	429	1,714
午前 3 時 46 分	0	488	536	539	570	2,133
午前 3 時 47 分	0	494	535	537	570	2,136
午前 3 時 48 分	3	495	534	536	568	2,136
午前 3 時 49 分	4	492	534	536	568	2,134
午前 3 時 50 分	7	492	534	535	567	2,135

前後の期間を含めてグラフ化したものが、図 47 である。

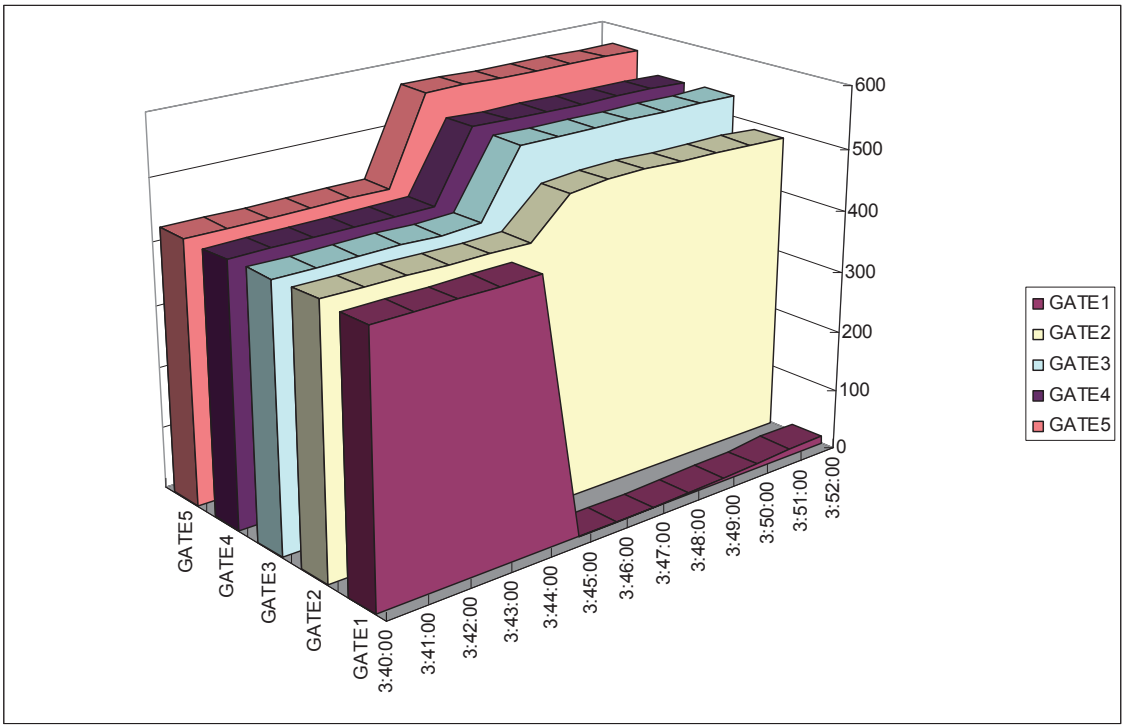

図 47 GATE1 を 3 分間停止させた際の各ゲートウェイの接続サーバセッション数の推移

この結果によると、GATE1 が停止した午前 3 時 45 分の瞬間には、合計サーバセッション数は一時的に GATE1 が停止する直前に処理していたサーバセッション数とほぼ同一の 425 セッションが減少しているが、その 1 分後の測定時 (午前 3 時 46 分) には、合計セッション数は元の合計サーバセッション数とほぼ同等の 2,133 セッションに復活していることがわかる。

これを同日午前 2 時から午前 6 時までの 4 時間に渡って記録したもののグラフが、図 48 である。

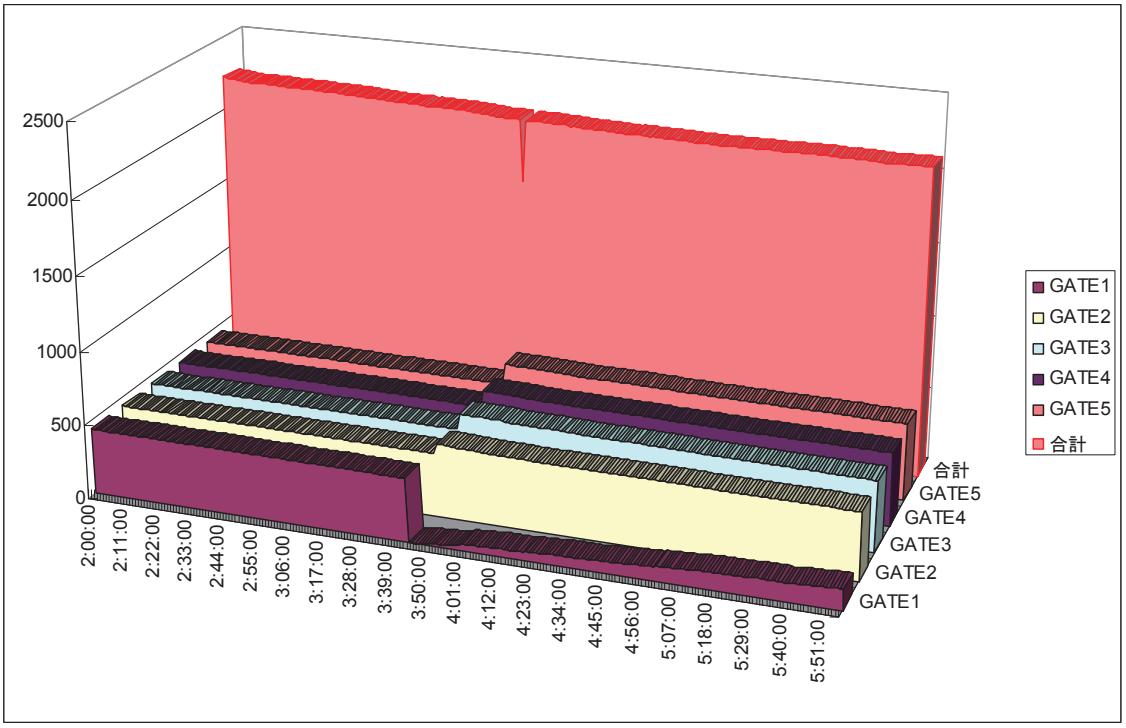

図 48 午前 2 時から午前 6 時までの各ゲートウェイおよび
合計の接続サーバセッション数の推移

この結果によると、GATE1 が動作を再開した後は、コントローラが新しいサーバセッションの接続要求をコントローラが受信すると、最も接続数の少ないゲートウェイである GATE1 に対してそのセッションを割り当てようとするので、GATE1 のセッション数が、明け方にかけて緩やかに増加していることがわかる。

6.3.10 ゲートウェイのうち 1 台を停止させた直後、サーバセッション接続要求が集中した瞬間における 1 分あたりの接続要求数およびコントローラの CPU 使用率の観測

1 月 28 日の午前 3 時 45 分から 3 分後の午前 3 時 48 分の間の 3 分間、GATE1 を停止させた際、GATE1 に接続されていたサーバ全てが、再度コントローラに対してサーバセッションの接続要求を行ったので、コントローラの CPU 使用率の値は、常時と比較して大きい値となるはずである。

そこで、上記付近の期間、コントローラで 1 分ごとに CPU 使用率を測定したところ、図 49 のようになった。

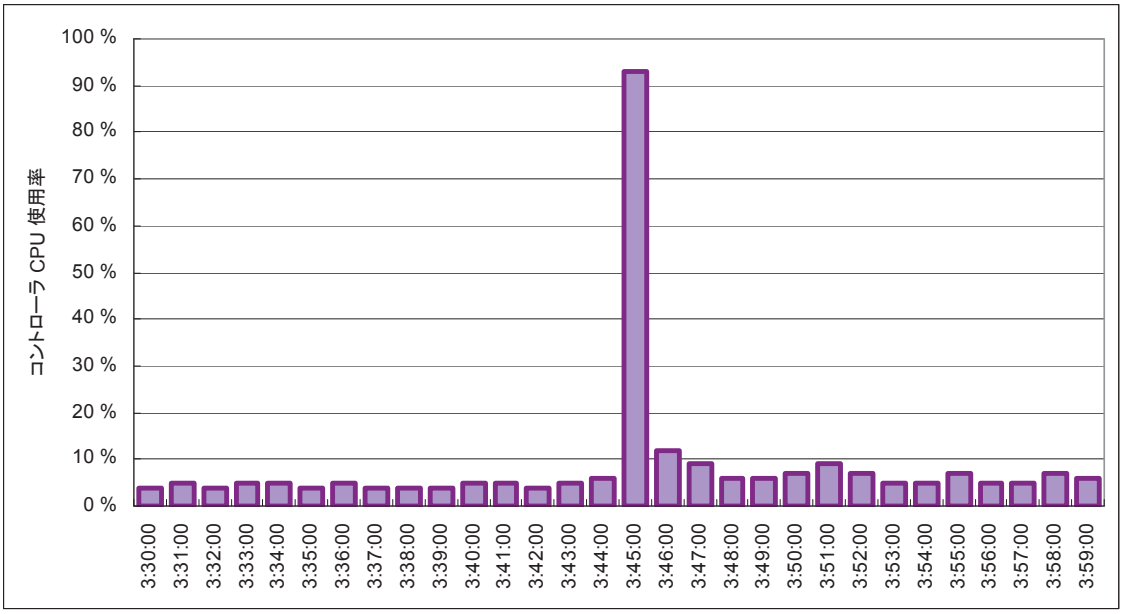

図 49 GATE1 の停止時刻付近のコントローラの CPU 使用率 (1 分ごと)

この結果によると、午前 3 時 45 分頃に、CPU 使用率が 93%という値をとっている。これを、6.3.7 における、1 分あたり 417 サーバセッションの接続要求を処理すると、CPU 使用率が 100%になるという予想と比較する。実験結果では、午前 3 時 45 分から 1 分間でコントローラが処理したサーバセッションの接続要求数は、425 回で、CPU 使用率は 93%であったので、この予想はおおむね正しかったと言える。

なお、サーバセッション数の合計値の記録では、GATE1 の停止前の午前 3 時 44 分には 2,139 セッション、GATE1 の停止後の午前 3 時 46 分には 2,133 セッションが接続されている。ここで、GATE1 の停止によって 6 セッションが減少したのは、一見、再接続要求を中断したサーバが存在するのではないかと考えることもできる。しかし、前後の合計サーバセッション数の推移を見れば明らかなように、合計サーバセッション数はそもそも数分ごとに増減しているのと同等の自然な増減であり、6 セッション減少していることが、GATE1 を停止したことによって発生するゲートウェイシステムへの再接続要求を中断したサーバが存在するということを示すのではない。コントローラにおいて過負荷が発生し、それが原因で再接続に失敗したサーバは存在しないと考えられる。

6.3.11 接続クライアントセッション数の推移 (ゲートウェイシステム全体)

1 月 21 日から 1 月 27 日までの 7 日間、GATE1〜GATE5 の 5 台のゲートウェイに接続されているクライアントセッション数の合計を 1 時間ごとに測定して記録したところ、その合計値は図 50 のようになった。

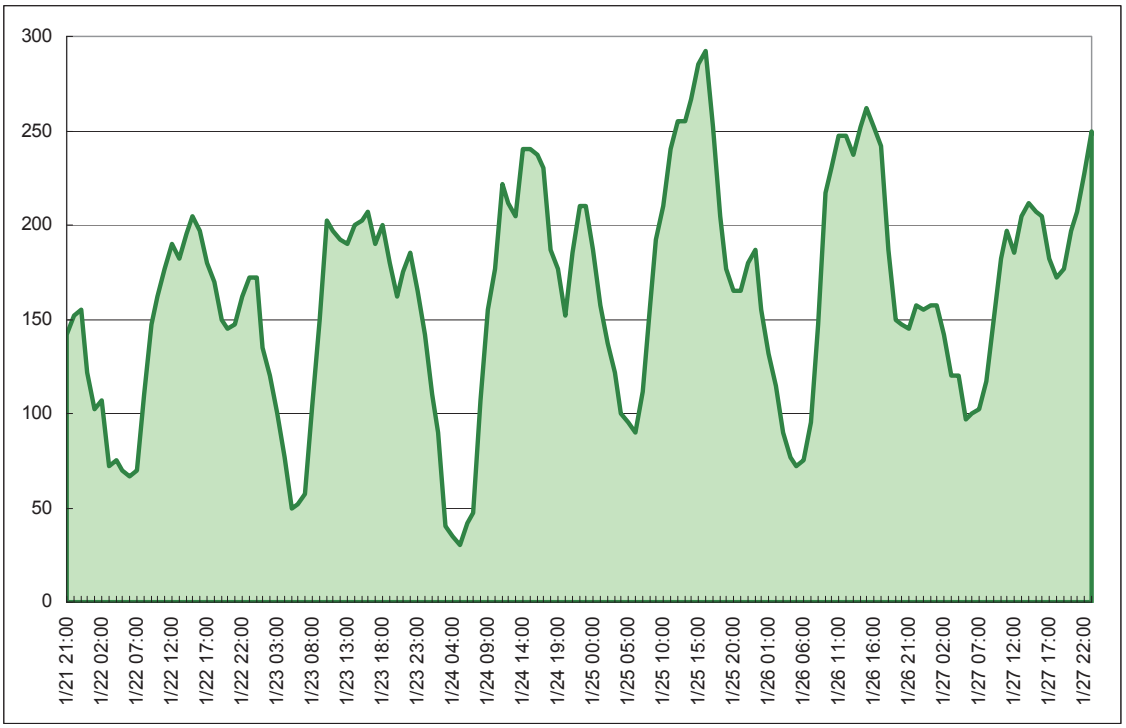

図 50 接続クライアントセッション数の推移 (ゲートウェイシステム全体)

これによると、毎日、8 時頃から 16 時頃までに接続クライアントセッション数が増加し、その後翌日の午前 1 時頃までかけて減少するというような規則性が見られる。また、平均約 162 クライアントセッションが接続されていることがわかる。

図 50 のデータと、図 41 のうち同一の期間内のデータを比較したグラフが図 51 である。

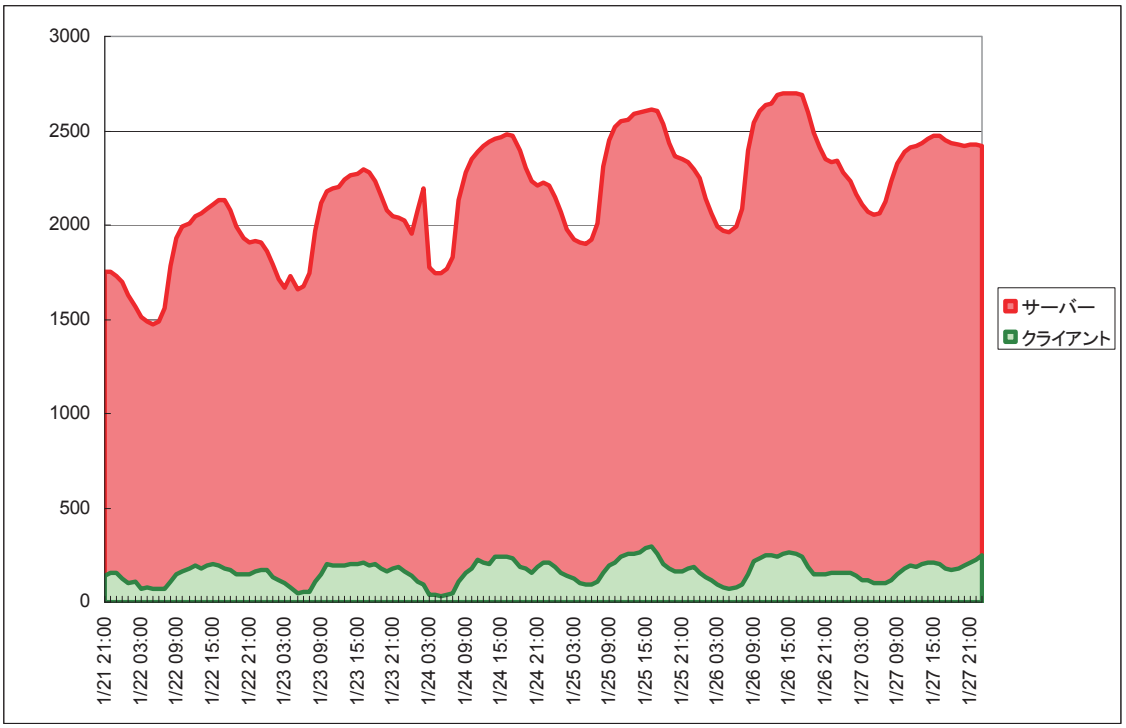

図 51 接続サーバセッション数およびクライアントセッション数の推移
(ゲートウェイシステム全体)

これによると、クライアントセッション数は、サーバセッション数の増減に比例して増減しているように見える。しかし、クライアントセッション数は午前 3 時から 6 時の間はとても少なくなるにもかかわらず、サーバセッション数は同時刻でも大半が接続されたままになっている。これによって、本実証実験のアプリケーションの機能である、リモートデスクトップの接続を待ち受けるサーバ側のコンピュータは常に電源を入れているユーザが多いこと、およびそれらのユーザがクライアントソフトウェアを使用してサーバに対してリモートアクセスしようとするのは朝から深夜 (午前 3 時頃) にかけてが大半であり、午前 3 時から朝までの間はほとんど誰もクライアントソフトウェアを使用していないということを推測することができる。

6.3.12 各ゲートウェイのトラフィック

各ゲートウェイ (GATE1 から GATE5) が処理しているトラフィックを、1 月 27 日から 2 月 1 日まで 5 分間隔で測定し、グラフ化したところ、図 52 から図 56 までのようになった。

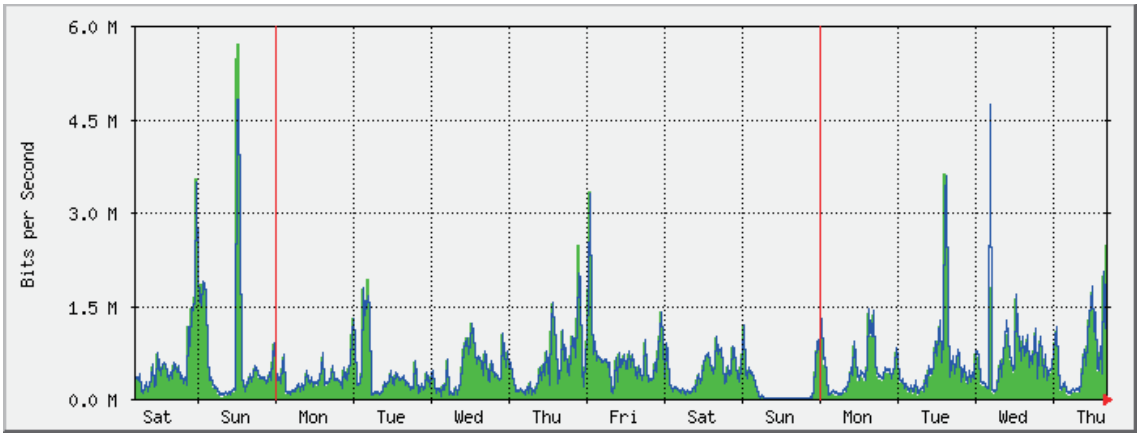

図 52 GATE1 のトラフィック

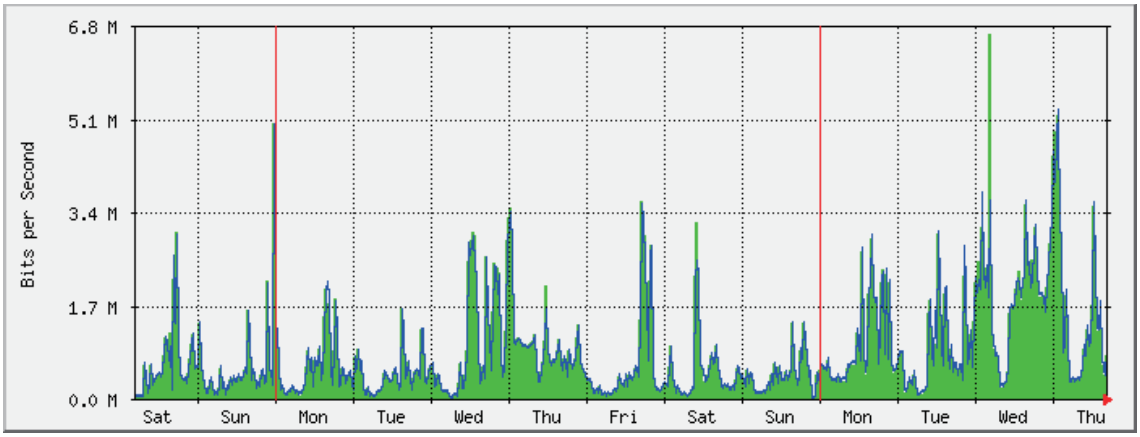

図 53 GATE2 のトラフィック

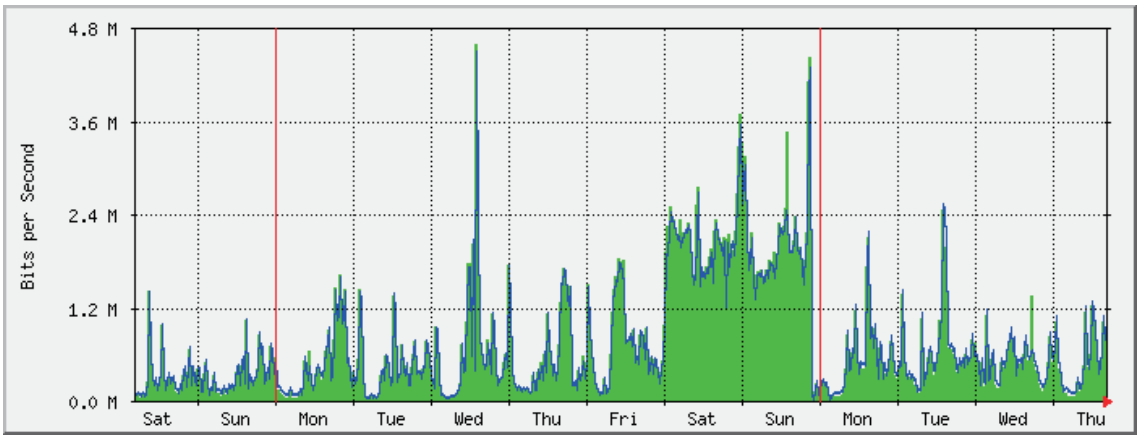

図 54 GATE3 のトラフィック

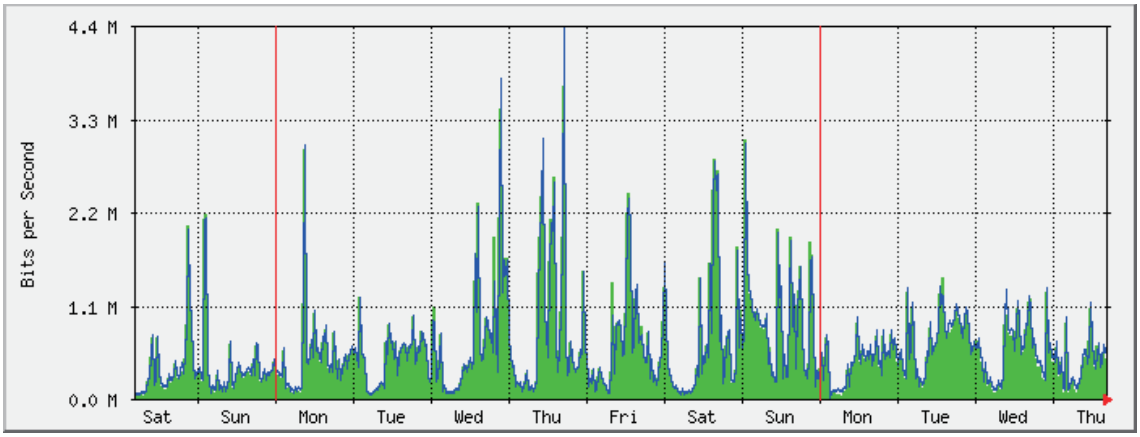

図 55 GATE4 のトラフィック

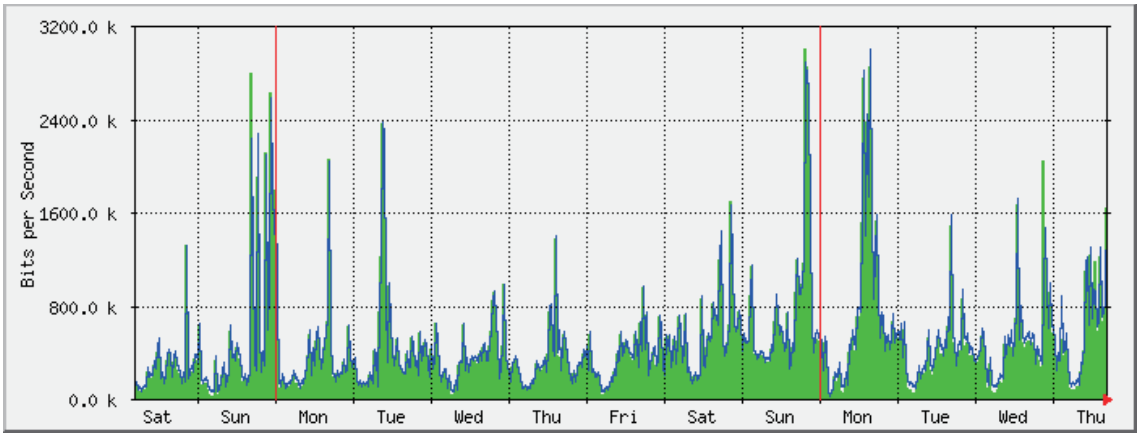

図 56 GATE5 のトラフィック

また、各ゲートウェイのプログラムが 1 月 16 日午前 5 時から 2 月 1 日午後 5 時までの稼働中に処理したトラフィックの合計量および平均値を取得すると、表 17 のようになった (ただし、測定結果は、各ゲートウェイプログラムがインターネットに対して送受信した IP パケットのヘッダを含めた合計サイズを含むため、ヘッダサイズ分および再送処理分が含まれる)。

表 17 トラフィックの合計量および平均値

コンピュータ名	受信合計 (bytes)	受信平均 (bps)	送信合計 (bytes)	送信平均 (bps)
GATE1	81,583,211,403	444,337	86,127,193,124	469,085
GATE2	125,050,880,790	738,876	128,280,119,941	757,956
GATE3	111,564,393,866	607,501	114,047,004,956	621,020
GATE4	91,170,926,649	496,453	95,419,608,387	519,588
GATE5	72,751,490,610	396,153	77,459,839,971	421,792
合計	482,120,903,318	2,683,320	501,333,766,379	2,789,441

理想的には、各ゲートウェイが受信したデータサイズと、送信したデータサイズは同一となるはずである。測定結果において、受信合計と比較して送信合計のほうが 3.9%多い理由は、パケットをインターネット経由で送受信した際に、パケットロスなどによって TCP がデータを再送した回数が、送信方向のほうが多かったためであると推測できる。測定方法やこれらの再送などによる誤差を無視すると、約 16.5 日間でゲートウェイシステムを経由してユーザが転送したトラフィックは、449.01 Gbytes 程度であることがわかる。

ゲートウェイによって処理したデータ量に、最大約 71.9%の差があることがわかる。具体的には、GATE2 が最も多く、GATE5 が最も少ない。この差は、本システムの設計段階から予期していたものである。本システムは、第 3 章で述べたように、各ゲートウェイのサーバセッション数によって各ゲートウェイの負荷を計算するという単純なアルゴリズムで負荷分散を行うので、各ゲートウェイの直前のトラフィックや CPU 負荷などは考慮されていないためである。

6.3.13 アプリケーションを使用した際の体感速度

1 月 16 日から 2 月 1 日の間に、本実証実験で設置したゲートウェイシステムを用いて、サーバプリケーションおよびクライアントアプリケーションを用いて、遠隔地にあるコンピュータのデスクトップにリモートアクセスして作業を行ったが、その際に得た主観的な体感速度について、クライアントを使用したコンピュータのアクセス回線ごとにまとめた。なお、サーバ側のコンピュータは、ゲートウェイとの間で RTT が 3 ミリ秒以下の高速回線上に接続しておいた。また、使用したプロトコルは RDP である。

実験を行った環境は、表 18 における A〜D の 4 箇所である。

表 18 ネットワーク環境の一覧

記号	使用したアクセス回線	ゲートウェイとの間のおおよその RTT	ゲートウェイとの間で測定した TCP コネクション 1 本あたりのおおよそのスループット
A	研究室の LAN 経由のインターネットアクセス	3 ミリ秒	13 Mbps
B	B フレッツ経由のインターネットアクセス	18 ミリ秒	5 Mbps
C	PHS によるインターネットアクセス (電波良好)	230 ミリ秒	80 Kbps
D	PHS によるインターネットアクセス (電波悪化)	500 ミリ秒	25 Kbps

C では、コンピュータを大学の建物の窓際に設置したところ、電波状況が良好であった。これに対して、D では、コンピュータを廊下側に設置したところ、電波状況が悪化し、RTT

が増大し、スループットが低下した (電波状況は、PHS 端末の LED によって、良好 / 悪化 / 最悪の 3 段階しか判別できなかった)。

体格速度の結果は、表 19 のとおりである。

表 19 体感速度の結果一覧

記号	体感速度の結果
A	デスクトップをスムーズにリモート操作することができ、ワープロソフトウェア等を使用したり、インターネットブラウザで Web サイトをブラウズしたりする程度の作業では、ローカルコンピュータを操作している場合と比較して、ストレスをほとんど感じなかった。
B	A と同等。
C	マウスをクリックしたり、キーボードから文字を入力したりした結果が、画面に反映されるまでに 1 秒未満のタイムラグが生じ、多少のストレスを感じたが、長時間作業することに苦痛は感じなかった。ただし、サイズの大きいビットマップ画像の表示等は、ビットマップの上端から順に描画されるので、違和感があった。
D	C と比較して 3 倍程度のタイムラグを感じたり、クリックやキータイプが無視されたりすることがあった。多大なストレスを感じ、長時間作業することに苦痛を感じた。

従って、本ゲートウェイシステムを経由して RDP を用いてリモートデスクトップ操作を行う場合は、B フレッツ等の高速回線であれば、ユーザはストレスなく使用できるが、PHS 回線の場合は、通信状況が悪化すると、ユーザに多大なストレスがかかることがわかった。

6.3.14 まとめ

本実証実験により、ゲートウェイシステムに接続するサーバセッション数が 2,000 セッション程度の場合は安定して動作することがわかった。また、実験に用いたスペックのサーバコンピュータで稼動させているコントローラが処理することができるサーバセッションの接続要求数は、1 分あたり約 400 回程度であることがわかった。さらに、ゲートウェイシステムの稼働中に、ゲートウェイのうち 1 台を停止させた場合は、そのゲートウェイに接続していたサーバは直ちに別のゲートウェイに対してサーバセッションを確立したこと、およびその後にゲートウェイを復活させた場合は、その後のサーバセッションは自動的にそのゲートウェイに接続するようになったことから、本システムの目標の 1 つであった、ゲートウェイシステムを停止させることのないゲートウェイの減少・増加が正しく動作することがわかった。

しかしながら、本実証実験における接続数の規模は、配布したサーバプリケーションをインストールして起動しているユーザ数によって決定される。今回の観測期間では、サーバ数は合計 6,000 程度、最大同時接続サーバセッション数は 2,700 程度であった。そのため、より大量のセッションが本システムに確立された場合の評価を行うことはできなかった。これは、今後の課題のひとつである。

第 7 章 結論

本章では、この研究の内容および結果をまとめるとともに、今後の課題について述べる。

7.1 まとめ

本研究では、インターネット上で利用することができる、スケーラブルなクライアント/サーバ型ストリーム中継システムを提案した。インターネット上で問題となる NAT 越え、ファイアウォール越えなどの処理を自動的に行い、固定グローバル IP アドレスの取得や DDNS サービスの利用などを必要とせずに、インターネットに接続された特定の 2 台のコンピュータ同士を接続して、自由な双方向ストリーム通信を可能とする。また、本システムによって通信を行うアプリケーションを開発する際は、既存のソケット API を利用して TCP 通信を行う際と同等のプログラミングを行うことができ、利用が容易である。

次に、提案したシステムの実装を、C、C# および SQL 言語によって行った。実装したプログラムによる遅延測定実験の結果は、本システムによる通信遅延はごくわずかであり、通常のネットワーク上で固定的に発生する遅延と比較するととても小さく問題とはならないことを示している。また、スループット測定実験の結果は、本システムを用いずに TCP 通信を行う場合と比較して、低下率は 10%〜30%であることを示している。従って、本実装は、インターネットで利用する場合、十分実用に耐えうるシステムである。

最後に、実装した本システムを利用してデスクトップのリモート操作を行う通信アプリケーションを開発し、インターネット上で配布して、多数のユーザに利用してもらったことにより、登録サーバレコード数 6,000 が程度、同時接続セッション数が 2,700 程度の状態で安定して動作すること、および動作中のゲートウェイシステムを構成する数台のコンピュータのうち一部が停止しても、システム全体は動作し続けることを示すことができた。この実証実験の結果は、本システムがスケーラビリティとアベイラビリティを有することを示している。

7.2 今後の課題

ゲートウェイシステムにおいては、ゲートウェイを複数台設置し、サーバセッションを各ゲートウェイに負荷分散したり、ゲートウェイのうち一部が停止した場合は自動的に別のアクティブなゲートウェイにセッションが移動したりすることを実現したが、これらの負荷分散処理を行っているコントローラは 1 台である。従って、コントローラが停止するか、コントローラとインターネットとの間の回線が不能となれば、ゲートウェイシステム全体が停止するため、コントローラが単一障害点となっている。今後の課題の一つとして、コントローラ自体も複数台用意し、負荷分散およびフォールトトレランスを実現したい。

その他の課題として、サーバセッションが接続されているゲートウェイが停止した場合は、一旦そのサーバセッションは切断された後、コントローラを介して別のゲートウェイに接続されるが、この際に、セッション内で確立されているコネクションが一旦切断される。

これを、サーバセッションが別のゲートウェイに移動しても、内部のコネクションは切断されずそのまま通信が継続できるような仕組みを実現したい。

最後に、本研究における実証実験においては、開始日から、本論文のために結果をまとめるまで 2 週間程度しか期間がなかったため、インターネット経由で集めることができたユーザ数 (登録サーバ台数) はわずか 6,000 程度であった。今後、引き続き継続実験を行い、ゲートウェイシステムにさらに負荷をかけることにより、10 万〜20 万セッション程度が同時にゲートウェイシステムに接続されていても安定して動作するかどうか検証したい。

謝辞

本研究を行うにあたり、まず、指導教員である筑波大学大学院システム情報工学研究科の板野肯三先生、新城 靖先生および佐藤 聡先生には、多くのご助言およびご指導をいただきました。ここに深く感謝申し上げます。特に、新城 靖先生には、本研究期間のみではなく、大学に入学してから現在までの 4 年間に、頻繁に技術的なご助言をいただきましたが、本研究を構成する要素には、それらを基礎に考えたアイデアによるものが多数含まれています。ありがとうございました。

また、著者が所属しているソフトウェア研究室のメンバーの皆様、筑波大学学術情報メディアセンターおよび産学リエゾン共同研究センターの関係者の皆様には大変お世話になりました。

本研究におけるゲートウェイシステムの実装の一部は、独立行政法人情報処理推進機構 (IPA) の平成 15 年度未踏ソフトウェア創造事業未踏ユース部門において著者が実施した開発プロジェクトで制作した VPN 技術をもとに行いました。同プロジェクトにおいてプロジェクトマネージャーとしてご指導いただいた東京大学の竹内郁雄先生にも、大変お世話になりました。

上記を含めた多くの方々にご協力いただけたことにより、本研究を円滑に実施し、まとめることができたと思います。誠にありがとうございました。

参考文献

[1] **社団法人 日本ネットワークインフォメーションセンター**: "IPv4 **アドレス枯渇に向けた提言**", http://www.nic.ad.jp/ja/research/ipv4exhaustion/ipv4exh-report.pdf, Mar 2006.

[2] Kjeld Borch Egevang and Paul Francis: "RFC1631: The IP Network Address Translator (NAT)", http://rfc.net/rfc1631.html, May 1994.

[3] UPnP Forum: "UPnP Standards", http://www.upnp.org/standardizeddcps/, November 2001.

[4] Jonathan Rosenberg, Joel Weinberger, Christian Huitema and Rohan Mahy: "RFC3489: STUN - Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs)", http://rfc.net/rfc3489.html, Mar 2003.

[5] **須之内雄司**: "NAT **越えに関する技術とその仕組み**", http://www.skame.nu/P2Pmeeting/0409sunouchi.pdf, Aug 2004.

[6] Tatu Ylonen and Chris Lonvick: "RFC4251: The Secure Shell (SSH) Protocol Architecture", http://rfc.net/rfc4251.txt, Jan 2006.

[7] Stephen Kent and Karen Seo: "RFC4301: Security Architecture for the Internet Protocol", http://rfc.net/rfc4301.html, Dec 2005.

[8] Kory Hamzeh, Gurdeep Singh Pall, William Verthein, Jeff Taarud, W. Andrew Little and Glen Zorn: "RFC2637: Point-to-Point Tunneling Protocol (PPTP)", http://rfc.net/rfc2637.html, Jul 1999.

[9] **登 大遊**: "SoftEther **の内部構造**", **情報処理** Vol.45, No.10, pp.1057-1062, Oct 2004.

[10] **ソフトイーサ株式会社**: "PacketiX VPN 2.0", http://www.softether.com/jp/vpn2/, Dec 2005.

[11] Salman A. Baset and Henning Schulzrinne: "An Analysis of the Skype Peer-to-Peer Internet Telephony Protocol", http://arxiv.org/pdf/cs.NI/0412017, Sep 2004.

[12] Michael J. Donahoo and Kenneth L. Calvert: "The Pocket Guide to TCP/IP Sockets: C Version", http://weblog.cs.uiowa.edu/22C178f02/uploads/PocketSocketC.pdf, Sep 2002.

[13] NTT **ドコモ** R&D **センター**: "**移動体通信のセル方式について**", http://www.nttdocomo.co.jp/corporate/rd/tech/04_01.html, Mar 1999.

[14] Carlisle Adams and Stephen Farrell: "Internet X.509 Public Key Infrastructure Certificate Management Protocols", http://rfc.net/rfc2510.html, Mar 1999.

[15] Alan O. Freier, Paul C. Kocher and Philip L. Karlton: "The SSL Protocol Version 3.0", http://wp.netscape.com/eng/ssl3/draft302.txt, Nov 1996.

[16] Politecnico di Torino: "WinPcap: The Windows Packet Capture Library", http://www.winpcap.org/.

[17] Microsoft Corporation: "QueryPerformanceCounter", http://msdn.microsoft.com/library/default.asp?url=/library/en-us/winui/winui/windowsuserinterface/windowing/timers/timerreference/timerfunctions/queryperformancecounter.asp.

[18] **筑波大学学術情報メディアセンター**: "**各種** VPN **ソフトウェアと** 64bit CPU **および** 10 Gigabit Ethernet **ハードウェアを用いた** VPN **通信の速度測定による性能比較実験の報告書**", Jun 2006.

[19] Microsoft, PictureTel and Polycom: "ITU-T Recommendation T.128 - Application Sharing", http://www.rdesktop.org/docs/t128.zip, Mar 1997.

[20] AT&T Laboratories Cambridge: "Virtual Network Computing", http://www.cl.cam.ac.uk/research/dtg/attarchive/vnc/, Mar 2003.

[21] **ソフトイーサ株式会社**: "Desktop VPN", http://www.softether.com/jp/desktop/, Jan 2007.