Ｔ Ｕ Ｉ Ｓ

修士請求論文

ハイブリッドP2P型VPNに関する研究

大学院総合情報学研究科
総合情報学専攻
G05006
菊地　裕明

東京情報大学

内容梗概

従来、企業等において社外から社内のネットワークに接続する方法として PPP によるダイヤルアップ型の接続が中心となっていた。

ところが近年、IP 網の発達と計算機の演算能力向上により、IP パケットを暗号化して転送し、接続先と手元の端末の間で仮想的な専用線接続を得る VPN (Virtual Private Network)が普及している。VPN という言葉の示す分野は広く、大きく分けてインターネットを介するインターネット VPN と通信事業者の提供する IP-VPN があり、更に接続方法や利用されている技術などにより区分される。本研究では IP 網上に P2P による独立したネットワークを構築する事を目的とし、その方式としてクライアント同士の接続情報を仲介サーバを介して交換し相互の接続を確立するハイブリッド P2P 型 VPN というものを検討する。

第 1 章では本論文の概要を述べる。第 2 章では現在の VPN がどのような用途、仕組みによって運用されているかを述べる。第 3 章では従来の VPN の運用を参考にハイブリッド P2P 型 VPN に関して検討を行い、第 4 章ではハイブリッド P2P 型 VPN を実現するための基本的な構成について述べる。

目　次

第 1 章　緒論

従来、企業等において社外から社内のネットワークに接続する方法として PPP によるダイヤルアップ型の接続が中心となっていた。

ところが近年、IP 網の発達と計算機の演算能力向上により、IP パケットを暗号化して転送し、接続先と手元の端末の間で仮想的な専用線接続を得る VPN (Virtual Private Network)が普及している。VPN という言葉の示す分野は広く、大きく分けてインターネットを介するインターネット VPN と通信事業者の提供する IP-VPN があり、更に接続方法や利用されている技術などにより区分される。用途による区分として代表的なものは、拠点に対して端末がリモートアクセスを行うリモートアクセス VPN や、拠点同士を接続する拠点間 VPN などである。

また、個人でも高速な常時接続回線が利用できるようになったため、外出先から自宅への接続に VPN を使うといった用途や、サークルや研究室と言った少人数のグループによる VPN の利用が現実的になりつつある。

そのような場合に問題が起こるのが VPN を構成する機器の性能や回線の能力である。特にリモートアクセス VPN として運用する場合、リモートアクセスサーバは全てのクライアントの通信を一度受信し、内容を解釈して必要な宛先に対して再度送信する。回線の高速化とともに取り扱うコンテンツのサイズも大容量化する傾向がある中で、サーバの負荷分散は重要な課題となる。

本研究では IP 網上に P2P による独立したネットワークを構築する事を目的とし、その方式としてクライアント同士の接続情報を仲介サーバを介して交換し相互の接続を確立するハイブリッド P2P 型 VPN というものを検討する。ハイブリッド P2P 型 VPN を利用すると IP 網上に独立したネットワークを構築することが出来るほか、ハイブリッド P2P 型システムの特徴としてクライアント同士が直接に通信を行うため、サーバに対して負荷が集中しない点や認証などの情報を一元管理できる点などのメリットが考えられる。

第 2 章では現在の VPN がどのような用途、仕組みによって運用されているかを述べる。第 3 章では従来の VPN の運用を参考にハイブリッド P2P 型 VPN に関して検討を行い、第 4 章ではハイブリッド P2P 型 VPN を実現するための基本的な構成について述べる。

第 2 章　従来の VPN から P2P 型 VPN への発展

2.1　緒言

本章では現在の VPN がどのような用途、仕組みによって運用されているかを述べる。VPN とは Virtual Private Network の略であり、仮想専用線･仮想私設網等と訳すことが出来る。

従来、ある拠点とある拠点を結ぶ通信は、拠点同士に専用線を導入するか、必要に応じてダイヤルアップ接続を行うことで行われていた。しかし、拠点数が多い場合、多数の拠点同士を専用線で結ぶ事は工事費や回線維持費の面から現実的ではなく、ダイヤルアップ接続では、通信速度の面で大容量のファイルの転送に耐えられないなどの問題があった。近年の通信回線の高速化･費用の低下を受けて、これらの問題を解決する為に IP 網を使用した VPN サービスの利用が増加している。

ユーザは速度や利用頻度などの条件に合わせた通信回線を IP 網への接続回線とし通信を行う。この通信の内容は通信事業者の IP 網内をカプセル化したパケットとして通過し、目的となる宛先へと配送される。これにより通信を行う拠点からは IP 網に対するアクセスラインを用意するだけで通信を行う全ての拠点と専用線を接続したのとほぼ同等の通信を行うことが仮想的なネットワークが出来る。

現在も重要度の高い情報や速度に対する厳密な保証が必要な場合には専用線等の手段が採られているが、そのような環境が必ずしも必要のない場合がある。VPN の普及によってユーザの選択できる通信路の選択肢が増え、利用環境の多様化が進んでいる。

また、家庭でのインターネットの利用が一般的になり ADSL や光ファイバー接続が普及することで、個人でも比較的容易に常時接続を行うことが出来るようになった。これにより IP 網としてインターネットを利用するインターネット VPN が利用可能となり、ユーザ同士を直接結ぶ目的としての VPN が注目されるようになった。

2.2 IP-VPN とインターネット VPN

IP-VPN とは通信事業者の提供する IP 網を利用して接続を行う VPN であり、インターネット VPN とはインターネットを介して接続を行う VPN である。両者とも同じく VPN と呼ばれ IP 網を利用しているが、その利用目的や技術に相違がある。

IP-VPN は主に企業ユーザが利用するサービスで、通信事業者の用意する IP 網を利用して VPN を構築するものである。通信事業者による専用の IP 網を利用しているため通信速度などを保証することが出来る。このサービスが従来の専用線接続と異なるのは拠点間を直接に接続するのではなく IP 網を介することである。この際、一般的にはあらかじめ契約により定めたラベルを付加し、そのラベル情報を基に MPLS (Multi-Protocol Label Switching)と呼ばれる技術を利用してユーザを区別している。ラベルによって複数のユーザ、拠点を区別することが出来るため通信事業者は回線を共用することにより回線敷設コストなどを引き下げることが出来る。また、ユーザは通信事業者までの接続に関して設定を行うが、他の拠点までどのように転送されていくかを意識する必要がなく、あたかも通信相手が同一ネットワーク上に存在するかのように扱うことが出来る。

インターネット VPN はインターネットを介して接続を行う VPN サービスに対する呼称である。インターネットは様々な事業者のネットワークが相互に繋がっているものであるため、通信速度や品質に関する保証は行われず、特殊な仕掛けを用意することも出来ないが ISP (Internet Service Provider)を通して安価に通信を行うことが出来る。インターネット VPN ではカプセル化した VPN としての IP パケットをインターネットを通して送信し、受信側で解釈することで仮想的な通信路を得る。インターネット VPN を利用するには送信側と受信側双方でプロトコルを決め、設定を行って運用する必要がある。 ISP はあくまでインターネットへの接続を提供するだけであり、必要な設定はすべてユーザが行う。利用形態により様々なプロトコルがあり、用途に応じて使い分けられている。

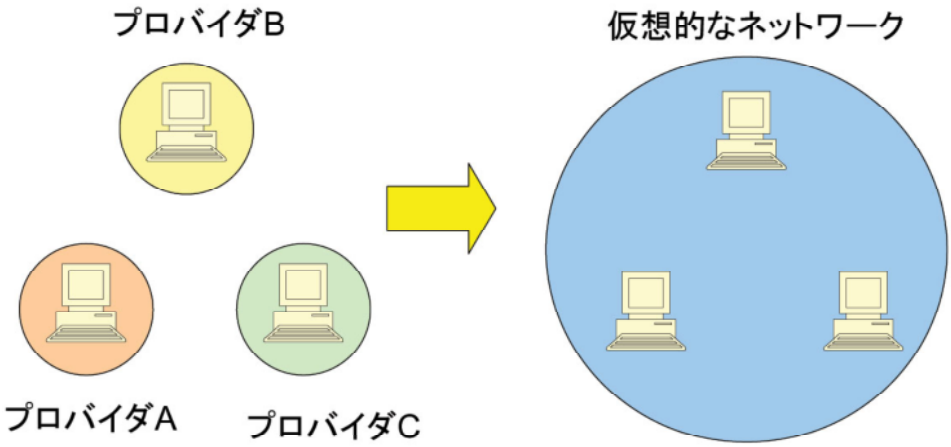

図 2-1　プロバイダの違いを超えて 1 つのネットワークとして動作する。

2.3　インターネット VPN の利用形態

インターネット VPN の一般的な利用形態として、拠点に対して端末が接続するリモートアクセス VPN と拠点同士を結ぶ拠点間接続 VPN に触れた上で、近年注目を集めつつある P2P 型 VPN について述べる。

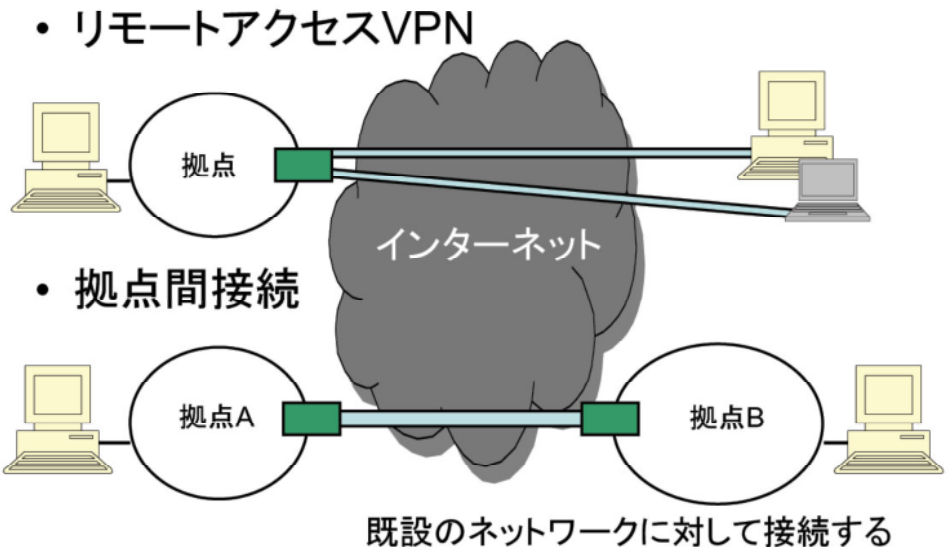

図 2-2　リモートアクセス VPN と拠点間接続 VPN

2.3.1 リモートアクセス VPN

リモートアクセス VPN とは、営業所などの拠点のネットワークと出張先などの出先にある端末を接続するような場合に利用されるＶＰＮである。従来ダイアルアップ IP 接続により提供されていたが、拠点側をサーバ、端末側をクライアントとする C/S (Client-Server)型の仕組みにより実現されるリモートアクセス VPN が利用される場合が増えてきた。その理由としてダイアルアップ IP 接続では通信速度の上限が低い事や回線数により同時利用可能なユーザの制約を受けてしまうこと、多くのユーザを同時に利用可能にするためには回線の維持コストが負担になることなどが挙げられる。代表的な例としては PPTP（Point-to-Point Tunneling Protocol）を用いた接続や SoftEther 社の SoftEther や PacketixVPN [1]を利用するもの、OpenVPN [2]を利用したものなどがある。

PPTP とは Point-to-Point Tunneling Protocol の略称であり、RFC2637 により規定されている VPN の為のプロトコルである。PPTP では PPP データグラムに GRE と呼ばれるヘッダを付けカプセル化する。一般的にはカプセル化したデータに対して MPPE (Microsoft Point-toPoint Encryption)を用いることで暗号化を行い送信し、受信側で復号を行うことで安全を保ちながらデータの転送を行う。

PPTP は主としてリモートアクセス VPN として用いられるが、本学では PPTP を用いて学生･教職員の利用する情報コンセント/無線 LAN システムを運用している。

2.3.2 拠点間接続 VPN

拠点間接続 VPN とは、営業所などの拠点のネットワークと別の拠点のネットワークを結ぶための VPN である。拠点と拠点を結ぶ VPN トンネルを作成し、それぞれの拠点に存在する端末からの通信を LAN 内の経路設定によりトンネルを使って行うため、クライアントとなる端末に特殊なソフトウェアの導入の必要はない。その代わりに接続先の拠点が増加した場合の設定変更や、障害発生時の影響範囲などは大きい。拠点間接続 VPN を利用する場合 IPSec に対応したルータを利用してネットワークを構築する場合が多い。

2.3.3　P2P 型 VPN

P2P とは Peer-to-Peer の通称であり、従来の C/S と対比される言葉である。 C/S 環境においては複数のクライアントがサーバを介して接続されるが、 P2P 環境では基本的にサーバを置かず端末同士が直接にデータのやりとりを行う。 P2P を実現する上で重要になる点は通信相手の端末を発見する方法である。従来の C/S 環境ではアクセスすべきサーバのアドレスというのは一意でありそのサーバに接続を行うことで確実に要件が満たされるが、 P2P によるネットワークでは端末同士が直接データをやりとりするという性質上、実際に通信を行う相手を特定しなければならない。現在、一般的に P2P と呼ばれるものは、この通信相手の特定を行う方式によってピュア P2P 型とハイブリッド P2P 型に分類される。

・ピュア P2P 型システム

ピュア P2P 型のシステムとは通信相手を特定するために P2P による通信を利用し、実際のデータ交換においても P2P による通信を用いるシステムのことである。 Winny などのファイル交換ソフトウェアがピュア P2P 型システムの代表例である。ピュア P2P 型のシステムでは通信先を特定するのに初期情報が必要となるが、初期接続以降の探索を自動的に続行し、ネットワークを維持し続ける。この方式を特定の通信先に対して接続するという目的に使用すると探索に時間がかかるなど無駄になる面が多いが、複数台から構成されたピュア P2P 型システムによって構築されたネットワークは、構成する端末が全て終了しない限り動作し続けるため障害に強いという面も持っている。

・ハイブリッド P2P 型システム

ハイブリッド P2P 型のシステムとは通信相手を特定するためにユーザ情報を管理するサーバを用意し、サーバの情報を参照して接続を行う方式で、接続の確立後は P2P によりクライアント同士で通信を行う方式である。音声通話ソフトの Skype [3]などが代表的な例であり、Skype では専用のサーバとプロトコルを用いて通話相手の情報を交換し、実際の通話に関しては直接データを送信する方式を取っている。この方式ではユーザ情報を管理するサーバで認証を行うことが出来るため、ユーザを識別する必要があるサービスなどでは有用であり、サーバを利用することで特定の相手

に確実に通信を行うことが出来る。ただし、ピュア P2P 型のシステムと違いサーバに依存すると言うことは、サーバがダウンすると全体が利用不可能になると言うことであり、耐障害性はピュア P2P 型のシステムに比べると弱くなる。

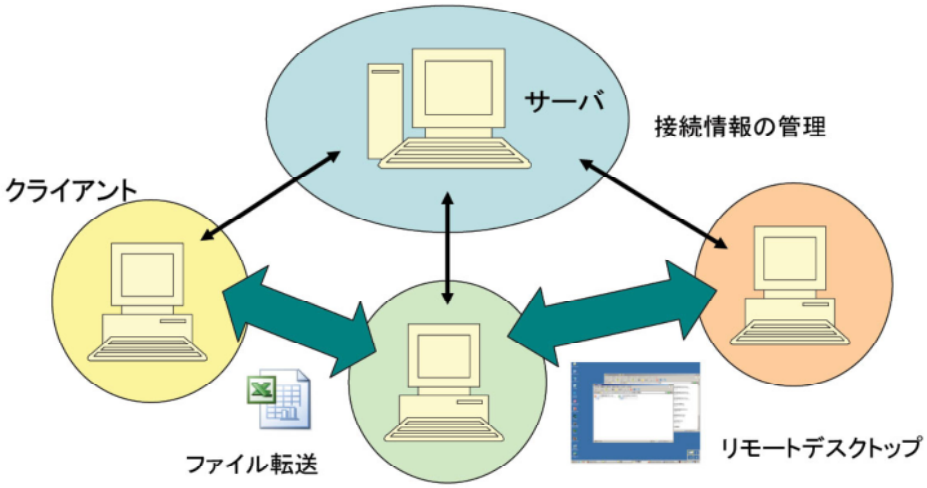

図 2-3　ハイブリッド P2P 型システムの概要

P2P 型 VPN はリモートアクセス VPN のようにクライアントが拠点に対して接続をかけるのではなく、拠点間接続 VPN のように拠点同士を結ぶわけでもない、新しい使い方である。この方法を用いると、データ転送の際に直接クライアント同士がデータをやりとりするため、通信時にトラフィックが集中する通信の中継用サーバを設ける必要がない。

ハイブリッド P2P 型の VPN では、通信先を特定し実際に接続を行うまでの前段階としてサーバを利用し、実際の通信においてはクライアント同士で行うという。通信に必要な情報はサーバに蓄えられ、要求に応じて送信される。

これら P2P 型のシステムを用いた VPN の例としては以下のようなものがある。

・ELA

第 6 回インターネットテクノロジーワークショップ（WIT2004）で慶應義塾大学の青柳 禎矩氏らが「ELA:分散型仮想ネットワークの設計と実装」という論文[4]を発表している。これは ELA (Everywhere Local Area network)という分散型仮想ネットワークを提案するもので、P2P 型のネット

ワークとしてノード間で即興的に分散型仮想ネットワークを形成する技術に関する内容である。

・**Hamachi**

Hamachi [5]はカナダの Applied Networking 社が開発したハイブリッド P2P 型の VPN サービスである。Applied Networking 社は 2006 年 8 月に買収され、現在は LogMeIn 社がこの事業を継続している。ユーザーは同社から提供されるクライアントソフトウェアを導入すると、同社の提供するサーバを介して接続相手との通信を設定し、P2P による VPN を確立する。同社では「zero-configuration virtual private networking (VPN)」と説明し導入が簡単であることを強調している。同社では無料で利用できる Free 版と有料だが詳細な設定を行うことが出来る Premium 版をリリースしている。

2.4 結言

本章では現在の VPN がどのような用途、仕組みによって運用されているかを述べた。これまでの経緯を踏まえ、本研究では接続のために仲介となるサーバを置き、各クライアント間が P2P で接続されるハイブリッド P2P 型の VPN に関して研究する。

想定する用途としては、サークルや研究室と言った少人数単位の学生の利用するコンピュータ同士を接続し、独立したネットワークとしてまとめることである。これを従来のリモートアクセス型の VPN で実現する場合、拠点となるリモートアクセスサーバに、全ての通信が集中してしまうため円滑なデータ送受信に支障を来す恐れがある。

ハイブリッド P2P 型 VPN においては、設置されるサーバで取り扱うのは制御情報であり、実際のデータ転送はクライアント間で行われる。この方式であれば比較的低負荷で独立したネットワークを形成できると考えられる。これにより従来のリモートアクセス型の VPN を利用する場合に比べて、サーバ側に発生するトラフィックを抑えながらネットワーク上に新たに独立したネットワークを形成する。

第 3 章　ハイブリッド P2P 型 VPN

3.1　緒言

本研究は仲介サーバを用いたハイブリッド P2P 型の VPN を構築することを目的としている。従来の VPN と P2P 型 VPN の違いは仮想ネットワークの形成である。これまでのリモートアクセス VPN や拠点間接続 VPN のような運用方法では、 VPN はあくまで固定的な拠点までの接続を行う伝送路として利用されていた。しかし、 P2P 型 VPN はデータ転送をクライアント相互で行うため拠点に依存することなく IP 網上に新たに独立したネットワークを構築することが出来る。

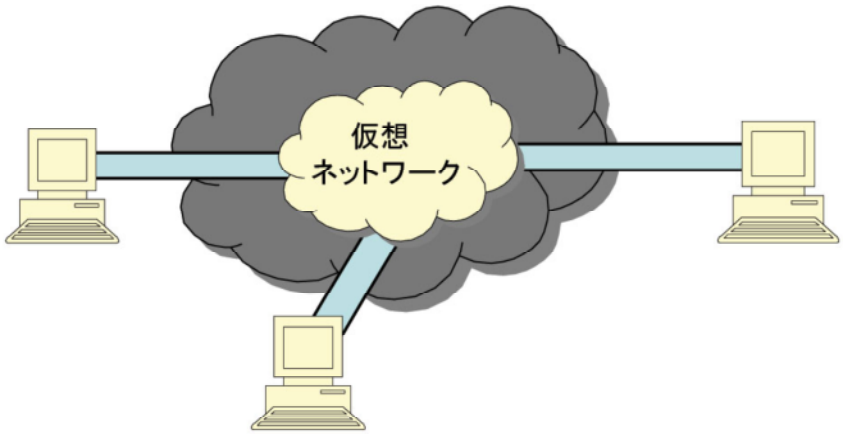

図 3-1　P2P 型 VPN のイメージ

このようなネットワークを利用することで、インターネットなどの IP 網を超えてサークルや研究室と言った少人数単位の学生の利用するコンピュータ同士を接続し、独立したネットワークとしてまとめることができる。

例えば講義毎に個別のネットワークを定義し教員と学生が必要な情報を交換する、自宅にいながらにして研究室と同様のネットワークに接続できる等といった運用方法が考えられる。

しかし、 P2P 型 VPN でこのような運用方法を実現するためにはいくつかの機能を備えていなければならない。

・相手の発見

従来の C/S システムと違い通信先と直接データを交換する仕組みとなるため、実際に通信を行う相手の IP アドレス等を把握していなければならない。

・認証

仮想ネットワークへのログインを考えた場合、正規利用者かどうかの識別を確実に行う必要がある。

・経路制御

P2P 型のネットワークは基本的にすべてのクライアントが直接データ交換できることを想定しているが、ルータの NAT 機能などがある正常に接続できない可能性が考えられる。こういった場合に間接的にデータを転送する為には、経路制御を行いデータを迂回させる必要がある。

これらの機能を実現するために本研究では仲介サーバを導入する。仲介サーバの役割としては、点在するクライアント同士の接続を円滑に行うための情報交換が挙げられる。クライアントはネットワークへ接続するために仲介サーバのアドレスを知っておく必要があるが、それ以外に情報については仲介サーバを参照することで解決できる。本研究では仲介サーバとの通信に SIP を用いる。

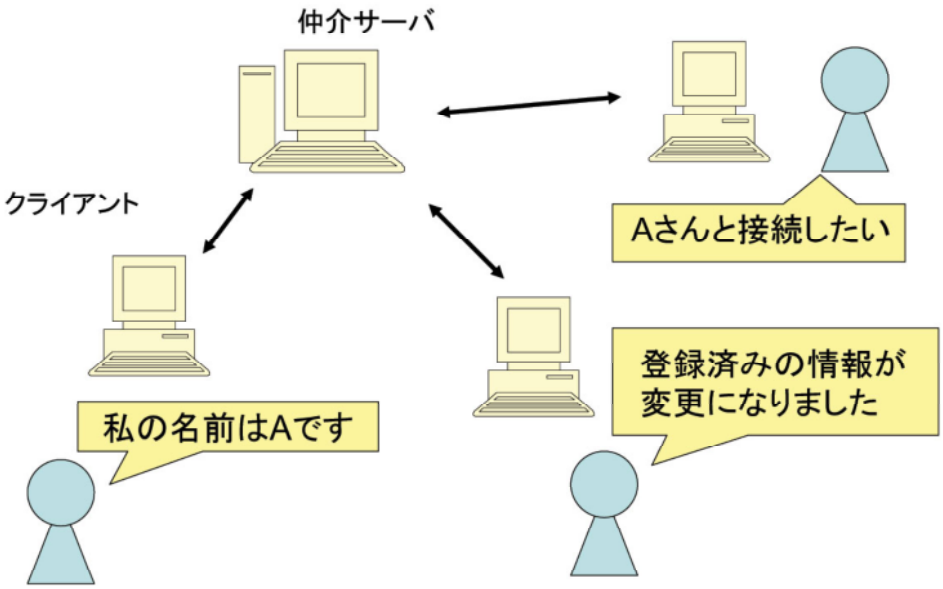

図 3-2　仲介サーバの役割

3.2　制御

3.2.1　SIP とは

SIP とは Session Initiation Protocol の略称で、RFC3261 等で規定されている。このプロトコルは通信におけるセッションの開始、終了などを管理するためのプロトコルであるため実際のデータ転送に関しては別途規定されることとなる。例えば SIP の主要なアプリケーションとして IP 電話が挙げられる。IP 電話における SIP の利用方法は SIP を用いた通信相手の特定に始まりセッションの開始、音声情報を転送するためのパラメータの設定、通話終了と言ったものであり、パラメータの設定のためには RFC 2327 で規定されている SDP（Session Description Protocol）を利用する。本研究では SDP を拡張し、VPN のセッション開始に必要な情報を送れるようにしている。

3.2.2　SIP を利用する利点

SIP はセッションを確立するために規定されたプロトコルであり、IP 電話による利用のみを目的としたプロトコルではない。 SIP にはユーザ同士でセッションを確立する機能や、着信転送と動的な登録により外出先へ移動しても同一手順で呼び出すことが出来る機能、 SIP サーバ-とユーザ間における認証機能などがある。 SIP を利用することで、それらの既存のサービスやネットワークを利用可能である。

例えば、既設の IP 電話の環境があれば SIP サーバを仲介サーバとして利用できる。

3.2.3　基本構成

・SIP アプリケーションとしての動作

ハイブリッド P2P 型 VPN を SIP を用いて実現する事を考えた場合、適切な動作は SIP メッセージによって情報を交換し、マルチメディアデータの代わりに VPN のパケットを送信する事であると考えられる。つまり、VPN 構成のためのパラメータを受信し、 VPN クライアントソフトウェアや必要であれば環境設定用のコマンドなどを発行して接続状態を維持できる環境を構築することでシステムとして成り立つのではないかと考えられる。

・複数台の接続

ユーザ名を指定すると指定した相手との間に VPN トンネルを追加することでネットワークとして動作する事となっている。しかし、実際には特定の相手に対して接続を行うのではなく、複数人を対象とした接続を行う必要がある。これは一つの VPN に属するユーザは同一 VPN に属する別のユーザに接続しているという状態が複数存在するということである。

本来 P2P 型の VPN という事であればクライアント間は全て直接に結ばれることが望ましいが、個人利用を考えた場合、一般家庭がインターネットへと接続する際にブロードバンドルータによって NAT が施されたプライベートネットワークに存在することが想定される。この場合、プライベートネットワーク内部から外部に向けて通信を行う場合には問題がないが、外部から内部に向けて送信する場合、特別な対策が取られない限り通信することは

出来ない。

解決方法としてはブロードバンドルータの設定を変更し、プライベートネットワーク内でデータを受信できるようにすることが挙げられる。一度設定すればその後も利用可能なため恒常的に利用できる確実な手段である。ただし、この方法ではユーザが直接に機器の設定を変更しなければならないため不便を強いる可能性がある。この設定をソフトウェアから行う方法として UPnP や UDP ホールパンチングなどの手法が考えられる。

それでも接続できないユーザに対しては経路制御を行い、ネットワーク中の接続可能であったユーザを経由して他のユーザと通信を行うことが出来ることが望ましい。

・認証方式

正規のユーザであるか確認を行うために認証は確実に必要である。また、一つの仲介サーバ上で複数の VPN を管理し、VPN に所属するためには識別のための ID とパスワードが別途必要になると考えられる。

大学や企業であれば認証情報の一元管理を行いやすく、ダイアルアップ接続やリモートアクセス VPN で利用されているように RADIUS のような認証機構を用いて認証することが可能であると考えられる。

個人利用を想定する場合、そこまで大規模な認証機構を導入することは一般的ではないため、ファイルなどに書きだしたデータを元に認証する仕組みも必要になる。

SIP では正規のユーザであるか確認を行う手順として、SIP サーバとユーザ間の認証について規定している。それによると SIP サーバとユーザ間の認証は HTTP ダイジェスト認証、または TLS(Transport Layer Security)による認証を行うこととなっている。この認証によりユーザや SIP サーバの成りすましを防止できる。

ユーザ間の認証を行う場合、接続の際に交換される SDP メッセージにパラメータを追加し認証情報の交換を行うことが考えられる。

・冗長性の確保

ハイブリッド P2P 型 VPN における仲介サーバの停止は、ネットワーク管理の停止と同義である。接続中の VPN はクライアント同士で通信しているため一時的な停止であれば影響を受けない可能性があるが、ユーザの参加離脱などの変更に関わる処理が不能になるため正常な運用は期待でき

ない。仲介サーバは故障すると問題となることから、 DNS におけるセカンダリサーバのように予備サーバを利用できる環境を整える事が望ましい。

SIP でこのような状況を想定する場合、クライアントの情報登録を行うレジストラの動作に関して工夫をする必要があると考えられる。

3.3 接続

SIP サーバをハイブリッド P2P 型 VPN サーバとして使用する場合、SIP により受理される SDP に VPN 接続のための情報を格納しなければならない。SDP で使用されるパラメータは以下の通りである。[6]

表 3-1 SDP のタイプ一覧（セッション記述）

タイプ	内容
v（必須）	プロトコルバージョン
o（必須）	セッションのオーナー
s（必須）	セッション名
l	セッション情報
u	記述の URL
e	電子メールアドレス
p	電話番号
c（必須）	接続情報
b	帯域情報
ひとつ以上の時間記述（必須）	
z	タイムゾーンの調整
k	暗号化キー
a	0 行以上のセッション属性行
0 個以上のメディア記述	

表 3-2 SDP のタイプ一覧（時間記述）

タイプ	内容
t（必須）	セッションの開始と終了時刻
r	0 個以上のセッション繰り返し時間

表 3-3　SDP のタイプ一覧（メディア記述）

タイプ	内容
m（必須）	メディアの名前とトランスポートアドレス
i	メディアのタイトル
c（必須）	接続情報 （セッション記述に書かれている場合はオプション）
b	帯域幅情報
k	暗号化キー
a	0 行以上のメディア属性行

VPN 接続に必要な情報はセッション情報のうち必須とされているものと、メディア記述に関する部分である。メディア記述とは音声又は映像と言ったマルチメディアデータを送受信する際に、コーデックなどの取り決めを行う為に用意されている設定項目である。メディア記述は複数あっても構わない。これは TV 会議のような音声と映像を同時に使い場面も存在するため、あらかじめ想定されている仕様である。

これらのあらかじめ定義された情報のうち、自由に付加して良いのはメディア属性行である a の項目である。メディア属性行とは音声や映像の場合、そのデータのコーデック情報などが具体的に割り振られている。同じ情報源からコーデックの違う複数の情報をデータを作成し、利用する側が対応する形式を選択できるようにするためである。

現在この方法で送信することを想定しているのは VPN で利用する接続先 IP アドレスやポート番号、 VPN で利用する仮想インターフェイス用の IP アドレスなどである。

3.4　結言

本章では従来の VPN の運用を参考にハイブリッド P2P 型 VPN の実現方法に関して検討を行った。ハイブリッド P2P 型 VPN を実現するためには VPN トンネルの接続を制御する部分と VPN トンネルの作成を行う部分の二つの機能が必要になることを示し、その制御に関してセッション管理用のプロトコルである SIP を用いる利点について検討した。実際に SIP を用いてネットワークを構成する方法と試験環境については次章で述べる。

第 4 章　実験システム

4.1　緒言

本章では実際に SIP を用いて VPN 接続の確立を行うアプリケーションを作成し、接続方法を検証する。 SIP を取り扱うアプリケーションを実装し、 VPN クライアントとして OpenVPN を利用したものを想定する。呼び出し、状態管理に SIP を用い、実際のトンネル接続に関しては OpenVPN を利用する形となる。 SIP では、音声通信の開始に SDP と呼ばれるプロトコルを用いて実際に通信する際のパラメータを設定するが、その機構を利用して OpenVPN の接続に必要なパラメータを転送し、外部アプリケーションとして OpenVPN を起動する形を取る。

4.2　実験環境

4.2.1　検証項目

今回の検証ではそれぞれ個別のネットワークに独立して存在する PC2 台に対して、それぞれユーザ名を割り当て、仲介サーバを通して実際の通信先となる端末の IP アドレス等を解決し、 VPN の確立を行う。今回の想定ではユーザ名について特に制約となる条件は無い。その為、任意の文字列をユーザ名として検証に利用する事が可能であるが本論文では PC1 をユーザ名 2500 、PC2 をユーザ名 2501 が利用しているものとする。

この環境において SIP を用いて実際に接続を行うことが出来るか確認する。

4.2.2　機材構成

ネットワークを構成する要素となるハードウェアの構成を提示する。これらのハードウェアを仲介サーバ、 PC1 、 PC2 とし、仲介サーバを利用して PC1 と PC2 の間に VPN 接続を構成する。

・仲介サーバ(202.26.159.131)

・ PC1(202.26.159.136)

・ PC2(172.22.1.28)

仲介サーバ及び PC1 を接続している 202.26.159.128/28 のネットワークをネットワーク 1 とし、 PC2 を接続している 172.22.1.0/24 のネットワークをネットワーク 2 としている。両者は L3 スイッチで相互に通信可能なように設定されている。 PC1 及び PC2 は、仲介サーバを利用して情報の交換を行い VPN トンネルを使った仮想的なネットワークを構築する。

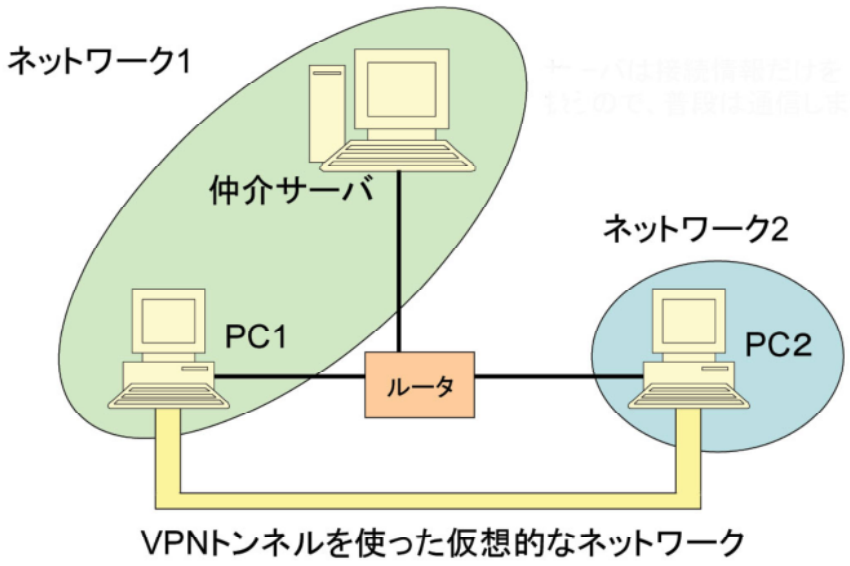

図 4-1　実験システム概要

4.3　ソフトウェア

検証環境で利用されているソフトウェアは仲介サーバとして SIP サーバソフトウェアである siproxd [7]、 SIP クライアントとして今回作成した sip_app 、 sip_app から起動する VPN クライアントとして OpenVPN となっている。

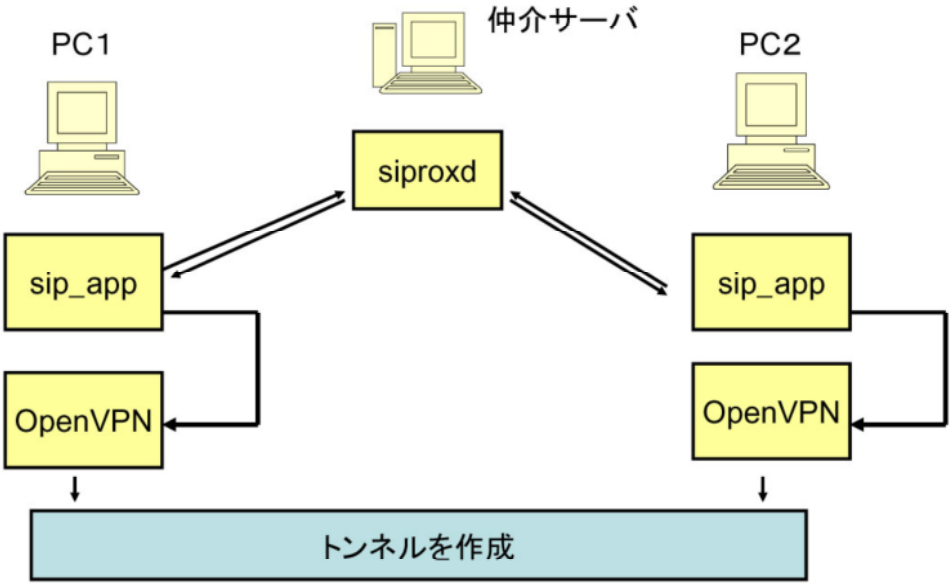

図 4-2　ソフトウェアの構成

4.3.1　sip_app とは

・動作

sip_app は今回検証するために作成したソフトウェアである。このアプリケーションは Linux 上で動作するアプリケーションとして提供される。実際の動作には大きく分けて 2 つのモードがあり、オプションスイッチにより区別される。オプションスイッチによって通信先のユーザが指定されなかった場合、他からの接続要求を待ち受けるサーバモードと、オプションスイッチによって通信先のユーザを指定した場合に接続にいくクライアントモードである。これらの機能を用いることで、 SIP を介した VPN コネクションの確立、管理を実現できる。

・サーバモードの動作

サーバモードでは、オプションスイッチとして仲介サーバのアドレスと仲介サーバに対して登録を行うユーザ名、接続の要求が正常に処理された場合に起動する外部プログラム（VPN クライアント）を必要とする。

・クライアントモードの動作

クライアントモードでは、サーバモードと同様にオプションスイッチとして仲介サーバのアドレスと仲介サーバに対して登録を行うユーザ名、接続の要求が正常に処理された場合に起動する外部プログラム（VPN クライアント）を必要とする。サーバモードとの違いはこれに接続先のユーザ名が加わることである。

・実装

sip_app の内部はいくつかの機能によって成り立っている。また、それらを実現するために外部ライブラリとしていくつかのライブラリを必要とする。使用するライブラリは oSIP2 [8]、eXosip2 [9]、TUIS_Lib [10]である。oSIP2 および eXosip2 は GPL で提供されている SIP ライブラリであり、これを利用することで SIP 機能を実現している。TUIS_Lib は東京情報大学井関ゼミで利用されている C 言語アプリケーション開発用のライブラリであり、ネットワークや文字列制御などの基本的な機能を提供している。

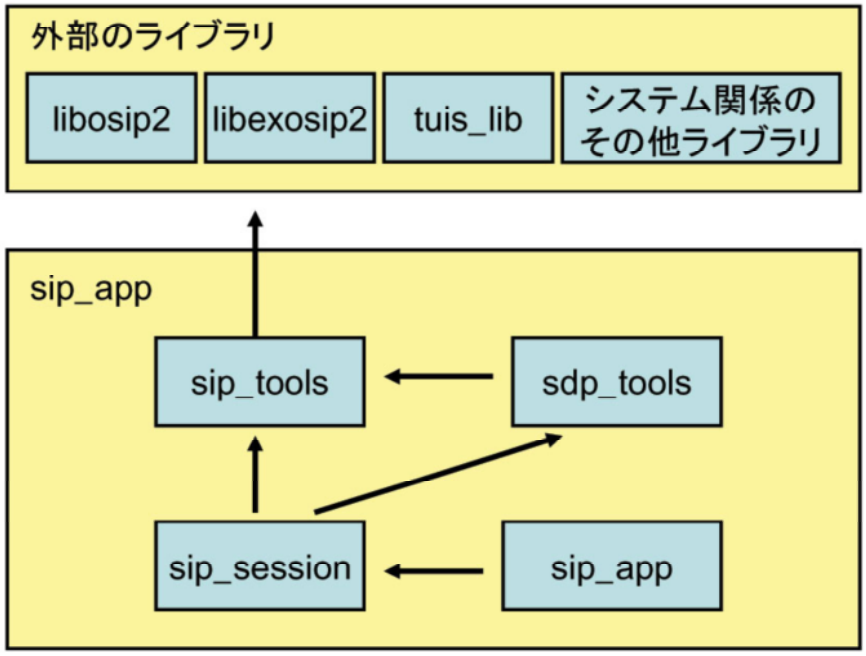

図 4-3　依存関係

sip_app 本体は SIP に関する命令を持つ sip_tools 、 SDP に関する命令を持つ sdp_tools 、それらを使って通信を行う sip_session 、そして命令の中心となる sip_app という機能から構成されている。sip_app には以下のようなコマンドオプションがある。

```
Usage... ./sip_app -s proxy[:port] -u user [-p port] [-t call_user] [-v vpn_wait_port] [-l
vpn_localip] [-r vpn_remoteip] [-f log_file] [-d] -m program [args]
```

s オプション

仲介サーバとなる SIP サーバを指定する。ポート番号は省略可能である。指定しない場合 SIP の標準であるポート番号 5060 を利用する。

u オプション

自分の利用するユーザ名を指定する。省略不可能である。、このオプションに基づいて仲介サーバに登録を行う。

p オプション

sip_app が受信に使うポート番号を指定する。省略可能である。

t オプション

接続を行うターゲットのユーザ名を指定する。省略可能である。このオプションを省略した場合、外部からの接続を待つサーバモードで待機を行う。オプションが指定された場合は、指定の相手に接続を行うクライアントモードとして動作する。

v オプション

VPN 接続を行う為のポート番号を指定する。省略可能である。省略した場合、アプリケーションの初期値としてポート番号 8000 を利用する。将来複数接続を同時に行う場合を考え、指定されたポートが使用中であればポート番号に+1 した値を利用するようになってる。

l オプション

サーバモードで待機する場合に使用できるオプションで、VPN トンネルを構築する場合にローカルの仮想インターフェイスに設定する IP アドレスを指定することが出来る。指定できるのは 192.168.0.1 などの 10 進数で表現された IPv4 アドレスに限られる。省略可能である。省略した場合の初期値は 192.168.234.1 となっている。

r オプション

サーバモードで待機する場合に使用できるオプションで、VPN トンネルを構築する場合にリモートの仮想インターフェイスに設定する IP アドレスを指定することが出来る。指定できるのは 192.168.0.1 などの 10 進数で表現された IPv4 アドレスに限られる。省略可能である。省略した場合の初期値は 192.168.234.2 となっている。

f オプション

デバッグモード中にデバッグデータを出力するファイル名を指定する。省略可能である。デバッグモードのオプションである d オプションと同時に設定されなければ有効にならない。

d オプション

デバッグモードを示すオプションである。このオプションが指定されている場合、 f オプションで指定されたファイルに対して動作のログを出力する。

m オプション

SIP による通信が確立した場合に起動される外部プログラムを指定する。省略不能である。 sip_app で指定されたプログラムを fork で子プロセスとして起動し、 SIP で交換した情報を引き渡す。引き渡される情報はサーバモードの場合、 VPN 接続を行う為のポート番号と VPN のローカル IP アドレス、リモート IP アドレスである。クライアントモードの場合、VPN 接続を行うためのポート番号の前に接続先の IP アドレスが追加される。また m オプションに対しては引数を取ることができ、その引数を与えて起動できる。

今回の検証においてはサーバモードの際に openvpn_server.sh 、クライアントモードの場合に openvpn_client.sh を呼び出し、そのスクリプト内で OpenVPN を起動するような仕組みとなっている。

4.3.2　siproxd とは

今回テストに利用した SIP のサーバプログラムを siproxd という。siproxd は Linux 、 FreeBSD 、 OpenBSD 、SunOS 、 Mac OS X 等の UNIX 系 OS 上で動作するフリーソフトウェアである。

siproxd はデバッグモードが使いやすく実験に適したサーバである。このデバッグモードでは詳細なデバッグ情報を出力可能である。

4.3.3　OpenVPN とは

OpenVPN とは暗号化に SSL を用いた VPN 環境の構築ソフトウェアである。本ソフトウェアでは「ブリッジ方式」「ルーティング方式」「Point-to-Point 接続」を利用した VPN を構成することが可能である。

・OpenVPN を利用する理由

接続に必要な情報の受け渡しを確認すること目的とするため、通信の暗号化などの部分に関しては新規に製作するよりも既存のソフトウェアを利用するほうが信頼性も高く、 OpenVPN であれば設定を柔軟に行うことが出来るため、確認を行うのに必要充分な性能を持っていると判断したためである。

・OpenVPN との連携

OpenVPN には事前に設定した設定ファイルを読み込むオプションと、各パラメータを起動時に指定するオプションがある。どの環境でも共通して利用できる基本設定をあらかじめファイルとして用意し、変動する部分を起動時に sip_app から与えることで適切な接続を行うことが出来る。

・キーの扱い

現在の段階では接続に必要な IP アドレスやポート番号と言ったパラメータを仲介サーバを介して交換することには成功しているが、暗号鍵の交換については行っておらず、共通暗号鍵となる情報が接続を行う各クライアントに設定済の状態で運用されている。

4.3.4 その他

確認作業を容易にするため、以下のような補助用のシェルスクリプトを用意した。これらのスクリプトのうち servertest.sh 及び openvpn_server.sh を PC1 において利用し、clienttest.sh 及び openvpn_client.sh を PC2 において利用している。

servertest.sh

```
# Test
./sip_app -s 202.26.159.131 -u 2500 -d -f ./log/server`date +%m%d%H%M`.txt -m
./openvpn_server.sh
```

clienttest.sh

```
# Normal
./sip_app -s 202.26.159.131 -u 2501 -t 2500 -d -f ./log/client`date +%m%d%H%M`.tx
-m ./openvpn_client.sh
```

openvpn_server.sh

```
# cat openvpn_server.sh
#!/bin/sh
# sip_app - OpenVPN(Server)  Connector
#default
/usr/local/sbin/openvpn --config ./conf/default.conf --lport $1 --ifconfig $2 $3
```

openvpn_client.sh

```
#!/bin/sh
#
# sip_app - OpenVPN(Client)  Connector
#default
/usr/local/sbin/openvpn --config ./conf/default.conf --remote $1 --rport $2 --ifconfig $
$3
```

また、全てのクライアントに共通の OpenVPN 用基本設定ファイルを用意した。

default.conf

```
#
# OpenVPN Config
#
dev tun

secret conf/default.key

user nobody
group nobody
persist-key

ping 10
ping-exit 30
persist-tun
float

# debug
verb 1
```

4.4 プロトコル

4.4.1 SIP による通信の確立

本プログラムにおいて SIP を用いて VPN を確立するまでの流れを説明する。本プログラムでは利用者 A 、B と仲介サーバという最小構成を想定してその接続を SIP を通して行う。

･通信の流れ

･接続

PC1 を sip_app のサーバモードで起動、 PC2 をクライアントモードで起動した場合の、接続までの動作の概略を図に示す。

サーバモードで起動した PC1 は仲介サーバに対して登録を行う。ここでは REGISTER の発行がそれに当たる。サーバモードでは INVITE を待つ。クライアントモードの場合 REGISTER の後、指定されたユーザ名に対して INVITE する。実際には SIP の接続シーケンスに従って動作するため、暫定応答の処理など手順がいくらか増える。

最終的にパラメータを交換し、接続に成功すると交換したパラメータを元に外部プログラムを起動する。この外部プログラムとして OpenVPN を利用している。 OpenVPN に対して接続に必要なパラメータを引き渡すことで、接続を確立すると、 VPN の仮想インターフェイスを通じた通信が可能になる。

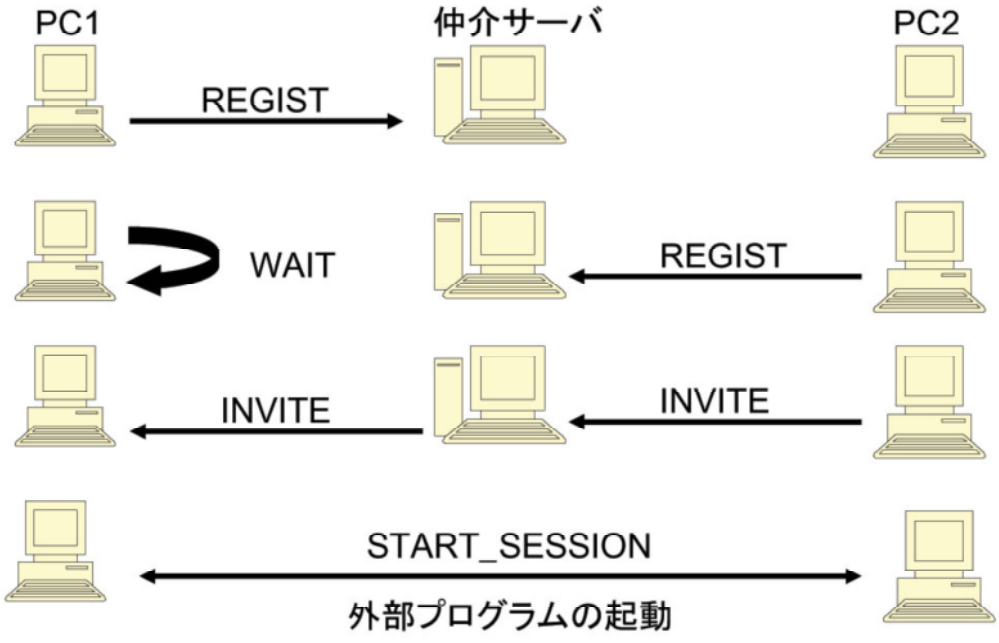

図 4-4　外部プログラムの起動

・切断

PC1 と PC2 の間で通信中に PC1 側で切断を行った場合の通信終了時の動作に関する概略を図に示す。

sip_app は外部プログラムの終了を検知すると通信先に対して終了を示す BYE 命令を発効して終了する。

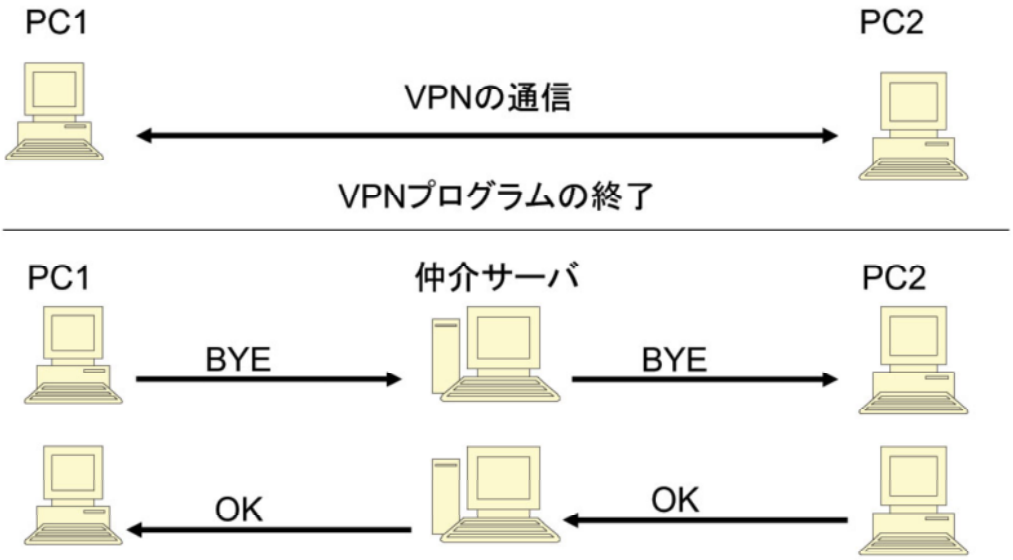

図 4-5　通信の終了処理

・受け渡すデータ

ここに示したのは sip_app が行った通信の一部である。実際に以下のようなパラメータが、SIP の情報交換によってやりとりされている。この例では待機中の PC1 のユーザ 2500 に対して PC2 のユーザ 2501 が接続する直前の応答である。

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP 172.22.1.28:5060;rport;branch=z9hG4bK1068437359
Record-Route: <sip:siproxd@202.26.159.131:5060;lr>
From: <sip:2501@202.26.159.131>;tag=1297171609
To: <sip:2500@202.26.159.131>;tag=1988095920
Call-ID: 1815073903@172.22.1.28
CSeq: 20 INVITE
Contact: <sip:2500@202.26.159.131>
User-agent: SIP for APP b1 rev.45
Allow: INVITE, ACK, OPTIONS, CANCEL, BYE, SUBSCRIBE, NOTIFY,
```

```
MESSAGE, INFO, REFER, UPDATE
Content-Type: application/sdp
Content-Length:    245

v=0
o=2500 1169538046 1169538046 IN IP4 202.26.159.131
s=-
t=0 0
m=application/vpn 7084 OpenVPN 0
c=IN IP4 202.26.159.131
k=DH:crypt code
a=IP4:202.26.159.136
a=PORT:8000
a=VPN_LOCAL_ADDR:192.168.234.1
a=VPN_REMOTE_ADDR:192.168.234.2
```

通信時に利用する MediaType として application/vpn 、 Protocol を OpenVPN としているが、この部分は独自に定義した内容である。一般的に SIP を利用する場合の利用するポート番号は M 、通信する IP アドレスは C に入れて渡すべきところであるが、独自の拡張をして A を利用している。 M と C の部分は SIP サーバを経由すると書き換えられてしまうため VPN を構成するに当たって利用できない情報であると判断したためである。この例においても本来 M のポート番号及び C のアドレスには別の情報が入っていたが SIP サーバによって変更がなされている。

SDP ではあらかじめ定義された項目以外の情報を転送する場合は A の項目を用いて拡張するよう定義されているため、今回もそれに従い必要な情報は A を拡張して格納している。

4.5 相互接続

実際に仲介サーバを介して PC1 と PC2 を接続した。

PC1 側

PC1 のコマンドラインから sip_app を呼び出す起動スクリプト servertest.sh を起動する。

その結果、仲介サーバへの登録が行われる。 servertest.sh はユーザ名

2500 として仲介サーバ 202.26.159.131 に対して接続するように設定が行われている。

```
[root@interlink SIP]# ./servertest.sh
PROXY     sip:202.26.159.131
FROM      sip:2500@202.26.159.131
CONTACT sip:2500@202.26.159.136:5060
TO        sip:
PROGRAM ./openvpn_server.sh
VPN_LOCAL 192.168.234.1
VPN_REMOTE 192.168.234.2
VPN_PORT 8000
sip_regist_answer: Event (type, did, cid) = (1, 0, 0)
send_regist: Registration Success.

recive_invite: new invite event wait start
debug:portnum:8000
```

画面上には表示されないが、デバッグ用のログには以下のように実際の通信内容が記録されている。

PC1 メッセージ

```
| INFO2 | <eXutils.c: 673> IPv4 address detected: 0.0.0.0
| INFO2 | <eXutils.c: 738> DNS resolution with 0.0.0.0:5060
| INFO2 | <eXconf.c: 658> eXosip: Reseting timer to 15s before waking up!
| INFO2 | <osip_transaction.c: 131> allocating transaction ressource 1 1946008052
| INFO2 | <nict.c: 34> allocating NICT context
| INFO2 | <eXutils.c: 673> IPv4 address detected: 202.26.159.131
| INFO2 | <eXutils.c: 738> DNS resolution with 202.26.159.131:5060
| INFO1 | <jcallback.c: 236> Message sent:
REGISTER sip:202.26.159.131 SIP/2.0
Via: SIP/2.0/UDP 202.26.159.136:5060;rport;branch=z9hG4bK17068792
From: <sip:2500@202.26.159.131>;tag=1399244863
To: <sip:2500@202.26.159.131>
Call-ID: 1946008052@202.26.159.136
CSeq: 1 REGISTER
Contact: <sip:2500@202.26.159.136:5060>
Max-Forwards: 70
```

```
User-Agent: SIP for APP b1 rev.45
Expires: 3600
Content-Length: 0
```

PC1 メッセージ

```
 (to dest=(unknown):5060)
| INFO2 | <eXconf.c: 673> eXosip: timer sec:0 usec:100000!
| INFO1 | <udp.c: 1742> Received message:
SIP/2.0 200 OK
Via: SIP/2.0/UDP 202.26.159.136:5060;rport;branch=z9hG4bK17068792
From: <sip:2500@202.26.159.131>;tag=1399244863
To: <sip:2500@202.26.159.131>
Call-ID: 1946008052@202.26.159.136
CSeq: 1 REGISTER
Contact: <sip:2500@202.26.159.136:5060>
Expires: 3600
Content-Length: 0
```

REGISTER と OK の送受信により、ユーザ名と IP アドレスの情報を仲介サーバ上で対応づける。servertest.sh で sip_app を起動した場合、登録が終了すると INVITE が送られてくるまで待機状態に入る。 INVITE メッセージを受け付けると接続のための処理が開始される。

```
sip_invite_wait_event: Event (type, did, cid) = (5, 2, 1)
session_start: セッションがスタートしました
```

デバッグ用のメッセージには Received message として INVITE を受け取ったことが記録されている。 PC2 側のクライアントとの通信は実際に VPN を張る段階まで仲介サーバを通して行うため、受け取るデータは 202.26.159.131 を経由していることが判る。

PC1 メッセージ

```
| INFO1 | <udp.c: 1786> Message received from: 202.26.159.131:5060 (serv=)
| INFO1 | <udp.c: 1792> Message received from: 202.26.159.131:5060 (serv=)
| INFO2 | <eXconf.c: 673> eXosip: timer sec:5 usec:10000!
| INFO1 | <udp.c: 1742> Received message:
INVITE sip:2500@202.26.159.136:5060 SIP/2.0
Via: SIP/2.0/UDP
202.26.159.131:5060;branch=z9hG4bKd5494712271eafdca196759bbcd82500
Via: SIP/2.0/UDP
202.26.159.131:5060;branch=z9hG4bKdf1ab35628fe284df07a0549b85b5d31
Via: SIP/2.0/UDP 172.22.1.28:5060;rport;branch=z9hG4bK1068437359
Record-Route: <sip:siproxd@202.26.159.131:5060;lr>
From: <sip:2501@202.26.159.131>;tag=1297171609
To: <sip:2500@202.26.159.131>
Call-ID: 1815073903@172.22.1.28
CSeq: 20 INVITE
Contact: <sip:2501@202.26.159.131>
Max-forwards: 68
User-agent: SIP for APP b1 rev.45
Expires: 120
Allow: INVITE, ACK, UPDATE, INFO, CANCEL, BYE, OPTIONS, REFER,
SUBSCRIBE, NOTIFY, MESSAGE
Content-Length: 0
```

INVITE に対して応答を返している。この応答はステータスコードが 1xx となっており、暫定応答である。暫定応答とは処理の要求から終了するまで待つのではなく、要求を受け付けたことを暫定的に知らせると言った用途に用いられる。 SIP では暫定応答を受けた場合、暫定応答自体に対してはレスポンスを返さない。

PC1 メッセージ

```
| INFO1 | <udp.c: 1786> Message received from: 202.26.159.131:5060 (serv=)
| INFO1 | <udp.c: 1792> Message received from: 202.26.159.131:5060 (serv=)
| INFO1 | <udp.c: 1804> This is a request
| INFO2 | <osip_transaction.c: 131> allocating transaction ressource 2 1815073903
| INFO2 | <ist.c: 31> allocating IST context
| INFO2 | <eXutils.c: 673> IPv4 address detected: 202.26.159.131
```

| INFO2 | <eXutils.c: 738> DNS resolution with 202.26.159.131:5060
| INFO1 | <jcallback.c: 236> Message sent:
SIP/2.0 100 Trying
Via: SIP/2.0/UDP
202.26.159.131:5060;branch=z9hG4bKd5494712271eafdca196759bbcd82500
Via: SIP/2.0/UDP
202.26.159.131:5060;branch=z9hG4bKdf1ab35628fe284df07a0549b85b5d31
Via: SIP/2.0/UDP 172.22.1.28:5060;rport;branch=z9hG4bK1068437359
From: <sip:2501@202.26.159.131>;tag=1297171609
To: <sip:2500@202.26.159.131>
Call-ID: 1815073903@172.22.1.28
CSeq: 20 INVITE
User-Agent: SIP for APP b1 rev.45
Allow: INVITE, ACK, OPTIONS, CANCEL, BYE, SUBSCRIBE, NOTIFY, MESSAGE, INFO, REFER, UPDATE
Content-Length: 0

PC1 メッセージ

(to dest=(unknown):5060)
| INFO2 | <eXutils.c: 673> IPv4 address detected: 202.26.159.131
| INFO2 | <eXutils.c: 738> DNS resolution with 202.26.159.131:5060
| INFO1 | <jcallback.c: 236> Message sent:
SIP/2.0 101 Dialog Establishement
Via: SIP/2.0/UDP
202.26.159.131:5060;branch=z9hG4bKd5494712271eafdca196759bbcd82500
Via: SIP/2.0/UDP
202.26.159.131:5060;branch=z9hG4bKdf1ab35628fe284df07a0549b85b5d31
Via: SIP/2.0/UDP 172.22.1.28:5060;rport;branch=z9hG4bK1068437359
Record-Route: <sip:siproxd@202.26.159.131:5060;lr>
From: <sip:2501@202.26.159.131>;tag=1297171609
To: <sip:2500@202.26.159.131>;tag=1988095920
Call-ID: 1815073903@172.22.1.28
CSeq: 20 INVITE
Contact: <sip:2500@202.26.159.136:5060>
User-Agent: SIP for APP b1 rev.45
Allow: INVITE, ACK, OPTIONS, CANCEL, BYE, SUBSCRIBE, NOTIFY, MESSAGE, INFO, REFER, UPDATE
Content-Length: 0

PC1 メッセージ

```
 (to dest=(unknown):5060)
| INFO2 | <eXutils.c: 673> IPv4 address detected: 202.26.159.131
| INFO2 | <eXutils.c: 738> DNS resolution with 202.26.159.131:5060
| INFO1 | <jcallback.c: 236> Message sent:
SIP/2.0 100 Trying
Via: SIP/2.0/UDP
202.26.159.131:5060;branch=z9hG4bKd5494712271eafdca196759bbcd82500
Via: SIP/2.0/UDP
202.26.159.131:5060;branch=z9hG4bKdf1ab35628fe284df07a0549b85b5d31
Via: SIP/2.0/UDP 172.22.1.28:5060;rport;branch=z9hG4bK1068437359
From: <sip:2501@202.26.159.131>;tag=1297171609
To: <sip:2500@202.26.159.131>;tag=1988095920
Call-ID: 1815073903@172.22.1.28
CSeq: 20 INVITE
User-Agent: SIP for APP b1 rev.45
Allow: INVITE, ACK, OPTIONS, CANCEL, BYE, SUBSCRIBE, NOTIFY, MESSAGE,
INFO, REFER, UPDATE
Content-Length: 0
```

PC1 メッセージ

```
 (to dest=(unknown):5060)
 | INFO2 | <eXutils.c: 673> IPv4 address detected: 202.26.159.131
 | INFO2 | <eXutils.c: 738> DNS resolution with 202.26.159.131:5060
 | INFO1 | <jcallback.c: 236> Message sent:
 SIP/2.0 180 Ringing
 Via: SIP/2.0/UDP
 202.26.159.131:5060;branch=z9hG4bKd5494712271eafdca196759bbcd82500
 Via: SIP/2.0/UDP
 202.26.159.131:5060;branch=z9hG4bKdf1ab35628fe284df07a0549b85b5d31
 Via: SIP/2.0/UDP 172.22.1.28:5060;rport;branch=z9hG4bK1068437359
 Record-Route: <sip:siproxd@202.26.159.131:5060;lr>
 From: <sip:2501@202.26.159.131>;tag=1297171609
 To: <sip:2500@202.26.159.131>;tag=1988095920
 Call-ID: 1815073903@172.22.1.28
 CSeq: 20 INVITE
 Contact: <sip:2500@202.26.159.136:5060>
 User-Agent: SIP for APP b1 rev.45
```

```
Allow: INVITE, ACK, OPTIONS, CANCEL, BYE, SUBSCRIBE, NOTIFY,MESSAGE,
INFO, REFER, UPDATE
Content-Length: 0
```

暫定応答による対応を終了して接続の成功を示すステータスコード 200 で応答している。

この応答の際に、 PC1 に対して接続を行うためのパラメータを送信する。

PC1 メッセージ

```
(to dest=(unknown):5060)
| INFO2 | <eXutils.c: 673> IPv4 address detected: 202.26.159.131
| INFO2 | <eXutils.c: 738> DNS resolution with 202.26.159.131:5060
| INFO1 | <jcallback.c: 236> Message sent:
SIP/2.0 200 OK
Via: SIP/2.0/UDP
202.26.159.131:5060;branch=z9hG4bKd5494712271eafdca196759bbcd82500
Via: SIP/2.0/UDP
202.26.159.131:5060;branch=z9hG4bKdf1ab35628fe284df07a0549b85b5d31
Via: SIP/2.0/UDP 172.22.1.28:5060;rport;branch=z9hG4bK1068437359
Record-Route: <sip:siproxd@202.26.159.131:5060;lr>
From: <sip:2501@202.26.159.131>;tag=1297171609
To: <sip:2500@202.26.159.131>;tag=1988095920
Call-ID: 1815073903@172.22.1.28
CSeq: 20 INVITE
Contact: <sip:2500@202.26.159.136:5060>
User-Agent: SIP for APP b1 rev.45
Allow: INVITE, ACK, OPTIONS, CANCEL, BYE, SUBSCRIBE, NOTIFY,MESSAGE,
INFO, REFER, UPDATE
Content-Type: application/sdp
Content-Length:    245

v=0
o=2500 1169538046 1169538046 IN IP4 202.26.159.136
s=-
t=0 0
m=application/vpn 8000 OpenVPN 0
k=DH:crypt code
```

```
c=IN IP4 202.26.159.136
a=IP4:202.26.159.136
a=PORT:8000
a=VPN_LOCAL_ADDR:192.168.234.1
a=VPN_REMOTE_ADDR:192.168.234.2
```

PC1 から PC2 に対して送信したパラメータが受理された場合、PC2 から ACK が送信される。この ACK の受信により実際の VPN 接続が開始される。

PC1 メッセージ

```
(to dest=(unknown):5060)
| INFO1 | <jcallback.c: 413> cb_ist_kill_transaction (id=2)
| INFO2 | <eXconf.c: 673> eXosip: timer sec:1 usec:10000!
| INFO2 | <eXconf.c: 673> eXosip: timer sec:1 usec:10000!
| INFO1 | <udp.c: 1742> Received message:
ACK sip:2500@202.26.159.136:5060 SIP/2.0
Via: SIP/2.0/UDP
202.26.159.131:5060;branch=z9hG4bKa23e5c0385e03ef4fd3c3c3d38f095e6
Via: SIP/2.0/UDP
202.26.159.131:5060;branch=z9hG4bKf90be6f905d9f03db481bfac5f9ba2ab
Via: SIP/2.0/UDP 172.22.1.28:5060;rport;branch=z9hG4bK573479532
Record-Route: <sip:siproxd@202.26.159.131:5060;lr>
From: <sip:2501@202.26.159.131>;tag=1297171609
To: <sip:2500@202.26.159.131>;tag=1988095920
Call-ID: 1815073903@172.22.1.28
CSeq: 20 ACK
Contact: <sip:2501@202.26.159.131>
Max-forwards: 68
User-agent: SIP for APP b1 rev.45
Content-Length: 0
```

ACK 受信後、送信したパラメータに合わせて VPN クライアントを起動する。この動作は sip_app が行う。INVITE を待ち受けてから起動する場合、VPN クライアントも起動して接続待ちを行うモードで起動する。この場合は 192.168.234.1 を自分の IP アドレス、192.168.234.2 を対向アドレスとする VPN トンネルを作成するため、プログラム内で取得した自らの IP アド

レスのポート 8000 番にて待機している。

```
./openvpn_server.sh 8000 192.168.234.1 192.168.234.2
Tue Jan 23 16:40:46 2007 OpenVPN 2.0.9 i686-pc-linux [SSL] [EPOLL] built on Oct 18 2006
Tue Jan 23 16:40:46 2007 TUN/TAP device tun0 opened
Tue Jan 23 16:40:46 2007 /sbin/ifconfig tun0 192.168.234.1 pointopoint 192.168.234.2 mtu 1500
Tue Jan 23 16:40:46 2007 GID set to nobody
Tue Jan 23 16:40:46 2007 UID set to nobody
Tue Jan 23 16:40:46 2007 UDPv4 link local (bound): [undef]:8000
Tue Jan 23 16:40:46 2007 UDPv4 link remote: [undef]
sip_session_term_wait: Event (type, did, cid) = (15, 2, 1)
Tue Jan 23 16:40:46 2007 Peer Connection Initiated with 172.22.1.28:1194
Tue Jan 23 16:40:47 2007 Initialization Sequence Completed
```

待機していると PC2 側からの VPN 接続が行われる。この場合は PC2 は 172.22.12.28 のポート 1194 を利用して通信してきたことが判る。これによりインターフェイス tun0 を利用した VPN トンネルの作成が完了する。通信の終了は VPN クライアントの終了に合わせて行われる。 PC1 または PC2 において VPN クライアントが終了すると、 sip_app は相手に対して BYE を送信して通信を終了する。今回の場合 PC2 側でクライアントを終了したため Received message として BYE を受け取ったログが残る。BYE を受け取った側は応答として OK を返して終了する。

PC1 メッセージ

```
| INFO1 | <udp.c: 1786> Message received from: 202.26.159.131:5060 (serv=)
| INFO1 | <udp.c: 1792> Message received from: 202.26.159.131:5060 (serv=)
| INFO1 | <udp.c: 1804> This is a request
| INFO2 | <eXconf.c: 673> eXosip: timer sec:1 usec:10000!
| INFO1 | <jcallback.c: 436> cb_nict_kill_transaction (id=1)
| INFO2 | <eXconf.c: 658> eXosip: Reseting timer to 15s before waking up!
| INFO2 | <eXconf.c: 658> eXosip: Reseting timer to 15s before waking up!
| INFO2 | <eXconf.c: 658> eXosip: Reseting timer to 15s before waking up!
| INFO2 | <eXconf.c: 658> eXosip: Reseting timer to 15s before waking up!
| INFO2 | <eXconf.c: 658> eXosip: Reseting timer to 15s before waking up!
| INFO2 | <eXconf.c: 658> eXosip: Reseting timer to 15s before waking up!
```

| INFO2 | <eXconf.c: 658> eXosip: Reseting timer to 15s before waking up!
| INFO2 | <eXconf.c: 658> eXosip: Reseting timer to 15s before waking up!
| INFO1 | <udp.c: 1742> Received message:
BYE sip:2500@202.26.159.136:5060 SIP/2.0
Via: SIP/2.0/UDP
202.26.159.131:5060;branch=z9hG4bK57910ad4aa45a3169c85f1a6a49a9ce3
Via: SIP/2.0/UDP
202.26.159.131:5060;branch=z9hG4bK87281fde72b832d952ce12852d3293f9
Via: SIP/2.0/UDP 172.22.1.28:5060;rport;branch=z9hG4bK695439551
Record-Route: <sip:siproxd@202.26.159.131:5060;lr>
From: <sip:2501@202.26.159.131>;tag=1297171609
To: <sip:2500@202.26.159.131>;tag=1988095920
Call-ID: 1815073903@172.22.1.28
CSeq: 21 BYE
Contact: <sip:2501@202.26.159.131>
Max-forwards: 68
User-agent: SIP for APP b1 rev.45
Content-Length: 0

PC1 メッセージ

| INFO1 | <udp.c: 1786> Message received from: 202.26.159.131:5060 (serv=)
| INFO1 | <udp.c: 1792> Message received from: 202.26.159.131:5060 (serv=)
| INFO1 | <udp.c: 1804> This is a request
| INFO2 | <osip_transaction.c: 131> allocating transaction ressource 3 1815073903
| INFO2 | <nist.c: 32> allocating NIST context
| INFO2 | <eXutils.c: 673> IPv4 address detected: 202.26.159.131
| INFO2 | <eXutils.c: 738> DNS resolution with 202.26.159.131:5060
| INFO1 | <jcallback.c: 236> Message sent:
SIP/2.0 200 OK
Via: SIP/2.0/UDP
202.26.159.131:5060;branch=z9hG4bK57910ad4aa45a3169c85f1a6a49a9ce3
Via: SIP/2.0/UDP
202.26.159.131:5060;branch=z9hG4bK87281fde72b832d952ce12852d3293f9
Via: SIP/2.0/UDP 172.22.1.28:5060;rport;branch=z9hG4bK695439551
From: <sip:2501@202.26.159.131>;tag=1297171609
To: <sip:2500@202.26.159.131>;tag=1988095920
Call-ID: 1815073903@172.22.1.28
CSeq: 21 BYE

```
User-Agent: SIP for APP b1 rev.45
Allow: INVITE, ACK, OPTIONS, CANCEL, BYE, SUBSCRIBE, NOTIFY, MESSAGE,
INFO, REFER, UPDATE
Content-Length: 0
```

PC2 側

PC2 のコマンドラインから sip_app を呼び出す起動スクリプト clienttest.sh を起動する。

その結果、仲介サーバへの登録が行われる。 clienttest.sh はユーザ名 2501 として仲介サーバ 202.26.159.131 に対して接続するように設定が行われている。

```
[root@commander SIP]# ./clienttest.sh
PROXY     sip:202.26.159.131
FROM      sip:2501@202.26.159.131
CONTACT sip:2501@172.22.1.28:5060
TO        sip:2500@202.26.159.131
PROGRAM ./openvpn_client.sh
VPN_LOCAL (null)
VPN_REMOTE (null)
sip_regist_answer: Event (type, did, cid) = (1, 0, 0)
send_regist: Registration Success.
```

PC2 メッセージ

```
| INFO2 | <eXutils.c: 673> IPv4 address detected: 0.0.0.0
| INFO2 | <eXutils.c: 738> DNS resolution with 0.0.0.0:5060
| INFO2 | <eXconf.c: 658> eXosip: Reseting timer to 15s before waking up!
| INFO2 | <osip_transaction.c: 131> allocating transaction ressource 1 1422549109
| INFO2 | <nict.c: 34> allocating NICT context
| INFO2 | <eXutils.c: 673> IPv4 address detected: 202.26.159.131
| INFO2 | <eXutils.c: 738> DNS resolution with 202.26.159.131:5060
| INFO1 | <jcallback.c: 236> Message sent:
REGISTER sip:202.26.159.131 SIP/2.0
Via: SIP/2.0/UDP 172.22.1.28:5060;rport;branch=z9hG4bK1923825125
From: <sip:2501@202.26.159.131>;tag=399182884
```

```
To: <sip:2501@202.26.159.131>
Call-ID: 1422549109@172.22.1.28
CSeq: 1 REGISTER
Contact: <sip:2501@172.22.1.28:5060>
Max-Forwards: 70
User-Agent: SIP for APP b1 rev.45
Expires: 3600
Content-Length: 0
```

PC2 メッセージ

```
 (to dest=(unknown):5060)
| INFO2 | <eXconf.c: 673> eXosip: timer sec:0 usec:100000!
| INFO1 | <udp.c: 1742> Received message:
SIP/2.0 200 OK
Via: SIP/2.0/UDP 172.22.1.28:5060;rport;branch=z9hG4bK1923825125
From: <sip:2501@202.26.159.131>;tag=399182884
To: <sip:2501@202.26.159.131>
Call-ID: 1422549109@172.22.1.28
CSeq: 1 REGISTER
Contact: <sip:2501@172.22.1.28:5060>
Expires: 3600
Content-Length: 0
```

clienttest.sh によって起動した sip_app はユーザ名 2501 として仲介サーバに自身を通知した後、ユーザ名 2500 に対して INVITE を送信します。送信すると暫定応答を経て呼出が行われ、接続情報が交換される。

```
sip_invite_answer: Event (type, did, cid) = (8, 0, 1)
sip_invite_answer: Remote App.  8
sip_invite_answer: Event (type, did, cid) = (8, 2, 1)
sip_invite_answer: Remote App.  8
sip_invite_answer: Event (type, did, cid) = (8, 2, 1)
sip_invite_answer: Remote App.  8
sip_invite_answer: Event (type, did, cid) = (9, 2, 1)
sip_invite_answer: Call Ringing.  9
```

```
sip_invite_answer: Event (type, did, cid) = (10, 2, 1)
sip_invite_answer: Call Answer.  10
sip_invite_send_and_answer = 10
```

接続行うため、仲介サーバ 202.26.159.131 に対してユーザ名 2500 として登録されている相手に対して Message sent を行う。

PC2 メッセージ

```
| INFO1 | <udp.c: 1786> Message received from: 202.26.159.131:5060 (serv=)
| INFO1 | <udp.c: 1792> Message received from: 202.26.159.131:5060 (serv=)
| INFO2 | <eXconf.c: 673> eXosip: timer sec:5 usec:10000!
| INFO2 | <osip_transaction.c: 131> allocating transaction ressource 2 1815073903
| INFO2 | <ict.c: 34> allocating ICT context
| INFO2 | <eXutils.c: 673> IPv4 address detected: 202.26.159.131
| INFO2 | <eXutils.c: 738> DNS resolution with 202.26.159.131:5060
| INFO1 | <jcallback.c: 236> Message sent:
INVITE sip:2500@202.26.159.131 SIP/2.0
Via: SIP/2.0/UDP 172.22.1.28:5060;rport;branch=z9hG4bK1068437359
From: <sip:2501@202.26.159.131>;tag=1297171609
To: <sip:2500@202.26.159.131>
Call-ID: 1815073903@172.22.1.28
CSeq: 20 INVITE
Contact: <sip:2501@172.22.1.28:5060>
Max-Forwards: 70
User-Agent: SIP for APP b1 rev.45
Expires: 120
Allow: INVITE, ACK, UPDATE, INFO, CANCEL, BYE, OPTIONS, REFER,
SUBSCRIBE, NOTIFY, MESSAGE
Content-Length: 0
```

送信した内容に関して暫定応答を受信しました。正式な応答が送信されるまで待機を続行する。

PC2 メッセージ

```
 (to dest=(unknown):5060)
| INFO2 | <eXconf.c: 673> eXosip: timer sec:0 usec:100000!
| INFO1 | <udp.c: 1742> Received message:
SIP/2.0 100 Trying
Via: SIP/2.0/UDP 172.22.1.28:5060;rport;branch=z9hG4bK1068437359
Record-Route: <sip:siproxd@202.26.159.131:5060;lr>
From: <sip:2501@202.26.159.131>;tag=1297171609
To: <sip:2500@202.26.159.131>
Call-ID: 1815073903@172.22.1.28
CSeq: 20 INVITE
User-agent: SIP for APP b1 rev.45
Allow: INVITE, ACK, OPTIONS, CANCEL, BYE, SUBSCRIBE, NOTIFY, MESSAGE,
INFO, REFER, UPDATE
Content-Length: 0
```

PC2 メッセージ

```
| INFO1 | <udp.c: 1786> Message received from: 202.26.159.131:5060 (serv=)
| INFO1 | <udp.c: 1792> Message received from: 202.26.159.131:5060 (serv=)
| INFO2 | <eXconf.c: 673> eXosip: timer sec:5 usec:10000!
| INFO1 | <udp.c: 1742> Received message:
SIP/2.0 101 Dialog Establishement
Via: SIP/2.0/UDP 172.22.1.28:5060;rport;branch=z9hG4bK1068437359
Record-Route: <sip:siproxd@202.26.159.131:5060;lr>
From: <sip:2501@202.26.159.131>;tag=1297171609
To: <sip:2500@202.26.159.131>;tag=1988095920
Call-ID: 1815073903@172.22.1.28
CSeq: 20 INVITE
Contact: <sip:2500@202.26.159.131>
User-agent: SIP for APP b1 rev.45
Allow: INVITE, ACK, OPTIONS, CANCEL, BYE, SUBSCRIBE, NOTIFY, MESSAGE,
INFO, REFER, UPDATE
Content-Length: 0
```

PC2 メッセージ

```
| INFO1 | <udp.c: 1786> Message received from: 202.26.159.131:5060 (serv=)
| INFO1 | <udp.c: 1792> Message received from: 202.26.159.131:5060 (serv=)
| INFO2 | <eXconf.c: 673> eXosip: timer sec:5 usec:10000!
| INFO1 | <udp.c: 1742> Received message:
SIP/2.0 100 Trying
Via: SIP/2.0/UDP 172.22.1.28:5060;rport;branch=z9hG4bK1068437359
Record-Route: <sip:siproxd@202.26.159.131:5060;lr>
From: <sip:2501@202.26.159.131>;tag=1297171609
To: <sip:2500@202.26.159.131>;tag=1988095920
Call-ID: 1815073903@172.22.1.28
CSeq: 20 INVITE
User-agent: SIP for APP b1 rev.45
Allow: INVITE, ACK, OPTIONS, CANCEL, BYE, SUBSCRIBE, NOTIFY, MESSAGE,
INFO, REFER, UPDATE
Content-Length: 0
```

PC2 メッセージ

```
| INFO1 | <udp.c: 1786> Message received from: 202.26.159.131:5060 (serv=)
| INFO1 | <udp.c: 1792> Message received from: 202.26.159.131:5060 (serv=)
| INFO2 | <eXconf.c: 673> eXosip: timer sec:5 usec:10000!
| INFO1 | <udp.c: 1742> Received message:
SIP/2.0 180 Ringing
Via: SIP/2.0/UDP 172.22.1.28:5060;rport;branch=z9hG4bK1068437359
Record-Route: <sip:siproxd@202.26.159.131:5060;lr>
From: <sip:2501@202.26.159.131>;tag=1297171609
To: <sip:2500@202.26.159.131>;tag=1988095920
Call-ID: 1815073903@172.22.1.28
CSeq: 20 INVITE
Contact: <sip:2500@202.26.159.131>
User-agent: SIP for APP b1 rev.45
Allow: INVITE, ACK, OPTIONS, CANCEL, BYE, SUBSCRIBE, NOTIFY, MESSAGE,
INFO, REFER, UPDATE
Content-Length: 0
```

接続可能であることを示すレスポンスコード　200 がメッセージとして返信された。このメッセージには VPN 接続に必要となるパラメータが付加されており、このパラメータに従って接続可能であると判断する場合 ACK を送信して接続を確立する。その後 VPN クライアントにパラメータを引き渡し VPN トンネルの作成を開始する。

PC2 メッセージ

```
| INFO1 | <udp.c: 1786> Message received from: 202.26.159.131:5060 (serv=)
| INFO1 | <udp.c: 1792> Message received from: 202.26.159.131:5060 (serv=)
| INFO2 | <eXconf.c: 673> eXosip: timer sec:5 usec:10000!
| INFO1 | <udp.c: 1742> Received message:
SIP/2.0 200 OK
Via: SIP/2.0/UDP 172.22.1.28:5060;rport;branch=z9hG4bK1068437359
Record-Route: <sip:siproxd@202.26.159.131:5060;lr>
From: <sip:2501@202.26.159.131>;tag=1297171609
To: <sip:2500@202.26.159.131>;tag=1988095920
Call-ID: 1815073903@172.22.1.28
CSeq: 20 INVITE
Contact: <sip:2500@202.26.159.131>
User-agent: SIP for APP b1 rev.45
Allow: INVITE, ACK, OPTIONS, CANCEL, BYE, SUBSCRIBE, NOTIFY, MESSAGE,
INFO, REFER, UPDATE
Content-Type: application/sdp
Content-Length:    245

v=0
o=2500 1169538046 1169538046 IN IP4 202.26.159.131
s=-
t=0 0
m=application/vpn 7084 OpenVPN 0
c=IN IP4 202.26.159.131
k=DH:crypt code
a=IP4:202.26.159.136
a=PORT:8000
a=VPN_LOCAL_ADDR:192.168.234.1
a=VPN_REMOTE_ADDR:192.168.234.2
```

PC2 メッセージ

```
| INFO1 | <udp.c: 1786> Message received from: 202.26.159.131:5060 (serv=)
| INFO1 | <udp.c: 1792> Message received from: 202.26.159.131:5060 (serv=)
| INFO1 | <jcallback.c: 394> cb_ict_kill_transaction (id=2)
| INFO2 | <eXutils.c: 673> IPv4 address detected: 202.26.159.131
| INFO2 | <eXutils.c: 738> DNS resolution with 202.26.159.131:5060
| INFO1 | <jcallback.c: 236> Message sent:
ACK sip:2500@202.26.159.131 SIP/2.0
Via: SIP/2.0/UDP 172.22.1.28:5060;rport;branch=z9hG4bK573479532
Route: <sip:siproxd@202.26.159.131:5060;lr>
From: <sip:2501@202.26.159.131>;tag=1297171609
To: <sip:2500@202.26.159.131>;tag=1988095920
Call-ID: 1815073903@172.22.1.28
CSeq: 20 ACK
Contact: <sip:2501@172.22.1.28:5060>
Max-Forwards: 70
User-Agent: SIP for APP b1 rev.45
Content-Length: 0
```

sip_app が接続先を正常に確認したことを示すため、PC2 のコンソール画面上には受信した内容を示すメッセージが表示される。このメッセージによると接続先は 202.26.159.136 でありポートは 8000 番、対向を 192.168.234.1、自分を 192.168.234.2 とする VPN トンネルを作成する。

```
-----------------------------------------
v_version   -> 0
o_username -> 2500
o_sess_id   -> 1169538046
o_sess_ver -> 1169538046
o_nettype   -> IN
o_addrtype -> IP4
o_addr       -> 202.26.159.131
s_name       -> -
i_info       -> (null)
u_uri        -> (null)
z_adjust     -> (null)
```

```
Media 0
m_media      ->  application/vpn
m_port       ->  7084
m_num_port ->  (null)
m_protocol ->  OpenVPN
m_keytype   ->  DH
m_keydata   ->  crypt code
m_nettype   ->  IN
m_addrtype ->  IP4
m_addr       ->  202.26.159.131
m_att_field ->  IP4
m_att_value ->  202.26.159.136
m_att_field ->  PORT
m_att_value ->  8000
m_att_field ->  VPN_LOCAL_ADDR
m_att_value ->  192.168.234.1
m_att_field ->  VPN_REMOTE_ADDR
m_att_value ->  192.168.234.2
-----------------------------------------

sdp_get_body: 202.26.159.136:8000   DH:crypt code

session_start: セッションがスタートしました
```

実際に仲介サーバから得られた情報を元に VPN クライアントの起動指示を行う。

```
./openvpn_client.sh 202.26.159.136 8000 192.168.234.1 192.168.234.2
Tue Jan 23 16:36:06 2007 OpenVPN 2.0.9 i686-pc-linux [SSL] built on Nov 10 2006
Tue Jan 23 16:36:06 2007 TUN/TAP device tun0 opened
Tue Jan 23 16:36:06 2007 /sbin/ifconfig tun0 192.168.234.2 pointopoint 192.168.234.1
mtu 1500
Tue Jan 23 16:36:06 2007 GID set to nobody
Tue Jan 23 16:36:06 2007 UID set to nobody
Tue Jan 23 16:36:06 2007 UDPv4 link local (bound): [undef]:1194
Tue Jan 23 16:36:06 2007 UDPv4 link remote: 202.26.159.136:8000
Tue Jan 23 16:36:16 2007 Peer Connection Initiated with 202.26.159.136:8000
Tue Jan 23 16:36:17 2007 Initialization Sequence Completed
```

　今回はVPNによる通信の終了をPC2側で行った。通信の終了はVPNクライアントを終了することで自動的に行われる。sip_appはVPNクライアントの終了を確認すると、接続先であるユーザ名2500に対してBYEを送信して終了する。

PC2メッセージ

```
 (to dest=(unknown):5060)
| INFO2 | <eXconf.c: 673>  eXosip: timer sec:4 usec:100000!
| INFO2 | <eXconf.c: 673>  eXosip: timer sec:0 usec:100000!
| INFO2 | <eXconf.c: 673>  eXosip: timer sec:0 usec:100000!
| INFO2 | <eXconf.c: 673>  eXosip: timer sec:0 usec:100000!
| INFO2 | <eXconf.c: 673>  eXosip: timer sec:0 usec:100000!
| INFO2 | <eXconf.c: 673>  eXosip: timer sec:0 usec:100000!
| INFO2 | <eXconf.c: 673>  eXosip: timer sec:0 usec:100000!
| INFO2 | <eXconf.c: 673>  eXosip: timer sec:0 usec:100000!
| INFO2 | <eXconf.c: 673>  eXosip: timer sec:0 usec:100000!
| INFO1 | <jcallback.c: 436>  cb_nict_kill_transaction (id=1)
| INFO2 | <eXconf.c: 658>  eXosip: Reseting timer to 15s before waking up!
| INFO2 | <eXconf.c: 658>  eXosip: Reseting timer to 15s before waking up!
| INFO2 | <eXconf.c: 658>  eXosip: Reseting timer to 15s before waking up!
| INFO2 | <eXconf.c: 658>  eXosip: Reseting timer to 15s before waking up!
| INFO2 | <eXconf.c: 658>  eXosip: Reseting timer to 15s before waking up!
| INFO2 | <eXconf.c: 658>  eXosip: Reseting timer to 15s before waking up!
| INFO2 | <eXconf.c: 658>  eXosip: Reseting timer to 15s before waking up!
| INFO2 | <eXconf.c: 658>  eXosip: Reseting timer to 15s before waking up!
| INFO2 | <osip_transaction.c: 131>  allocating transaction ressource 3 1815073903
| INFO2 | <nict.c: 34>  allocating NICT context
| INFO2 | <eXutils.c: 673>  IPv4 address detected: 202.26.159.131
| INFO2 | <eXutils.c: 738>  DNS resolution with 202.26.159.131:5060
| INFO1 | <jcallback.c: 236>  Message sent:
BYE sip:2500@202.26.159.131 SIP/2.0
Via: SIP/2.0/UDP 172.22.1.28:5060;rport;branch=z9hG4bK695439551
Route: <sip:siproxd@202.26.159.131:5060;lr>
From: <sip:2501@202.26.159.131>;tag=1297171609
To: <sip:2500@202.26.159.131>;tag=1988095920
Call-ID: 1815073903@172.22.1.28
CSeq: 21 BYE
Contact: <sip:2501@172.22.1.28:5060>
```

```
Max-Forwards: 70
User-Agent: SIP for APP b1 rev.45
Content-Length: 0
```

4.5.1 改善点

現在の段階では複数台に対する接続が考慮されていないので、接続先を増やした場合について検討されていない。もし複数ユーザを対象とする場合は、経路情報の伝達について考えるユーザ認証を行う機構についても検討を行う必要がある。経路制御の為の情報を取り扱う必要がある。

4.6 結言

本章では SIP を使った VPN クライアントの制御方法及び実装について述べた。これにより SIP を利用して、 VPN 構築に必要な情報を交換し VPN トンネルが接続可能であることを示したが、現在の方法では接続時に認証を行う機能やファイヤーウォールや NAT と言った接続制限のある環境での動作が実装されていないため、一般家庭のような比較的制約の多い条件下では同様の検証が出来ない。実際の運用に関する機能については更なる改良と検証の余地がある。

第 5 章　結論

本研究では接続のために仲介となるサーバを置き、各クライアント間が P2P で接続されるハイブリッド P2P 型の VPN に関して研究を行った。

仲介サーバを介したデータ交換の為のプロトコルには SIP を用い、SIP で IP 電話などのマルチメディアデータの通信を開始する前のパラメータ設定に用いられている SDP を拡張して VPN の構成情報を転送する方法について検討した。

検討に基づいて実際に作成したクライアントソフトが仲介サーバを経由して得た情報を元に外部アプリケーションとなる VPN クライアントを起動し、仲介サーバ及び PC2 台を利用した環境において VPN 接続が可能である事を確認した。

実際に一般家庭のような環境で利用する為には、様々な点に関して更なる検討や改良が必要となるが SIP を利用したハイブリッド P2P 型 VPN の基本的な事項に関しては確認できたと考えている。

謝　　辞

本研究に際し適切なる御指導ならびに御助言をいただいた東京情報大学大学院総合情報学研究科　池田博昌教授に心から深謝致します。

本研究を進める過程において終始有益な御助言と励ましをいただいた東京情報大学総合情報学部情報システム学科　井関文一助教授には、厚く感謝致します。

また、東京情報大学大学院総合情報学研究科　木ノ内康夫教授、東京情報大学大学院総合情報学研究科　平野正則教授をはじめ有益な助言、討論、協力をいただいた関係各位に厚くお礼を申し上げます。また、日頃から活発な討論を頂いたネットワークシステム研究室の諸氏にも感謝いたします。

参　考　文　献

[1] PacketixVPN 、http://www.softether.com/jp/

[2] OpenVPN 、http://openvpn.net/

[3] Skype 、http://www.skype.com/intl/ja/

[4] ELA 、http://www.ht.sfc.keio.ac.jp/~sada/etc/pdf/wit2004.pdf

[5] Hamachi 、http://www.hamachi.cc/

[6] ソフトフロント著、「エッセンシャル SIP 」、日経 BP 社、2005 年

[7] siproxd 、http://siproxd.sourceforge.net/

[8] oSIP2 、http://www.gnu.org/software/osip/

[9] eXosip2 、http://www.antisip.com/as/en/products.php

[10]TUIS_Lib 、 http://www.solar-system.tuis.ac.jp/SoftWare/index.html

[11] 澤田　拓也他著、「実践 SIP 詳解テキスト」、リックテレコム、 2005 年

[12] ケイズプロダクション著、「 OpenVPN で構築する 超簡単 VPN 入門」、株式会社ラトルズ、2006 年

付録

sip_app ソースコード

・sdp_tools.h

```
/*


*/


#ifndef   _SDP_TOOLS_H
#define   _SDP_TOOLS_H


/**/

int     sdp_set_body (sip_param* param, osip_message_t* mesg);
int     sdp_set_body2(sip_param* param, osip_message_t* mesg);

sdp_message_t*   sdp_get_body(sip_param* param);
char* sdp_get_media_keytype(sdp_message_t* sdp, int pos);
char* sdp_get_media_keydata(sdp_message_t* sdp, int pos);
char* sdp_get_media_attr(sdp_message_t* sdp, char* key, int pos);

void   sdp_print_body(sdp_message_t* sdp);
void   sdp_show_body(sdp_message_t* sdp);


/**/

#endif
```

・sdp_tools.c

```
 /**
  SDP 用ツール                                    by Fumi.Iseki
                                                 （C）2006 9/6
*/



#include "sip_tools.h"
#include "sdp_tools.h"




////////////////////////////////////////////////////////////////////////////////////////////////////
//
//          SDP
//                              注意： t は必須．パラメータの順序も考慮する．m　は一番最後.
//

/**
int   sdp_set_body(sip_param* param, osip_message_t* mesg)

          機能：param の変数に従って，mesg へ SDP ボディを格納する.

          引数：param : SIP 用パラメータ
                    mesg　: 転送用の SIP メッセージへのポインタ.

          戻り値：0： 正常終了.
                              0 未満：エラー.

*/
int   sdp_set_body(sip_param* param, osip_message_t* mesg)
{
          time_t tsec;
          char* buf;
          char   proto_fmtlst[LDATA];
          sdp_message_t* sdp = NULL;
```

```
        time(&tsec);                                    //  1900 年からではない
(1970). 要修正?.
        memset(proto_fmtlst, 0, LDATA);
        snprintf(proto_fmtlst, LDATA-1, "%s %d", param->protocol, 0);

        sdp_message_init(&sdp);
        sdp_message_v_version_set(sdp, "0");
        sdp_message_o_origin_set(sdp, param->username, ultostr(tsec), ultostr(tsec), "IN",
"IP4", param->localip);
        sdp_message_s_name_set(sdp, "-");
        sdp_message_t_time_descr_add(sdp, "0", "0");

        sdp_message_m_media_add(sdp, param->media, ltostr(param->localport), NULL,
proto_fmtlst);
        sdp_message_k_key_set(sdp, 0, param->keytype, param->keydata);
        sdp_message_c_connection_add(sdp, 0, "IN", "IP4", ltostr(param->localport), NULL,
NULL);
        sdp_message_a_attribute_add (sdp, 0, "IP4", param->localip);
        sdp_message_a_attribute_add (sdp, 0, "PORT", ltostr(param->localport));
        sdp_message_a_attribute_add (sdp, 0, "VPN_LOCAL_ADDR", param->vpn_localip);
        sdp_message_a_attribute_add (sdp, 0, "VPN_REMOTE_ADDR", param->
vpn_remoteip);

        sdp_message_to_str(sdp, &buf);
        sdp_print_body(sdp);
        osip_message_set_body(mesg, buf, strlen(buf));
        osip_message_set_content_type(mesg, "application/sdp");

        free(buf);
        sdp_message_free(sdp);

        return 0;
}



/**
int  sdp_set_body2(sip_param* param, osip_message_t* mesg)
```

機能：param の変数に従って，mesg へ SDP ボディを格納する もう一つの方法.
分かり易いがスマートではない.

引数：param : SIP 用パラメータ
mesg ：転送用の SIP メッセージへのポインタ.

戻り値：0： 正常終了.
0 未満：エラー.

```
*/
int   sdp_set_body2(sip_param* param, osip_message_t* mesg)
{
        time_t tsec;
        char buf[BUFSZ];


        time(&tsec);                                    //  1900 年からではない
(1970). 要修正?.

        snprintf (buf, BUFSZ,
                        "v=0¥r¥n"
                "o=%s %ld %ld IN IP4 %s¥r¥n"
                "s=-¥r¥n"
                                "t=0 0¥r¥n"
                                "m=%s %d %s %s¥r¥n"
                                "k=%s:%s¥r¥n"
                                "c=IN IP4 %s¥r¥n"
                                "a=IP4:%s¥r¥n"
                                "a=PORT:%d¥r¥n"
                                "a=VPN_LOCAL_ADDR:%s¥r¥n"
                                "a=VPN_REMOTE_ADDR:%s¥r¥n",
                                param->username, tsec, tsec, param->localip,
                                param->media, param->localport, param->protocol, param-
>fmtlist,
                                param->keytype, param->keydata,
                                param->localip,
                                param->localip,
                                param->localport,
                                param->vpn_localip,
```

```
                                        param->vpn_remoteip);

        osip_message_set_body(mesg, buf, strlen(buf));
        osip_message_set_content_type(mesg, "application/sdp");

        return 0;
}




/**
sdp_message_t*   sdp_get_body(sip_param* param)

        機能：相手の返答から SDP ボディを取り出す.

        引数：param : SIP 用パラメータ

        戻り値：SDP ボディへのポインタ.
                        エラーの場合は NULL.

*/
sdp_message_t*   sdp_get_body(sip_param* param)
{
    sdp_message_t* sdp = NULL;

    sdp = eXosip_get_remote_sdp(param->did);
        if (sdp==NULL) {
        sdp = eXosip_get_remote_sdp_from_tid(param->tid);
        }

        return sdp;
}
```

```
/**
char*   sdp_get_media_keytype(sdp_message_t* sdp, int pos)

        機能：SDP ボディから media の keytype を取り出す.

        引数：sdp : SDP ボディへのポインタ.
                pos : 何番目のメディアかを指定する．通常は 0.

        戻り値：keytype を示す文字列．free する必要あり.
                        エラーの場合は NULL

*/
char*   sdp_get_media_keytype(sdp_message_t* sdp, int pos)
{
        char* ret = NULL;
        sdp_media_t* med;

        if（sdp==NULL || sdp->m_medias==NULL）return NULL;
        if（pos<0）pos = 0;
        med =（sdp_media_t*）osip_list_get(sdp->m_medias, pos);
        if（med==NULL || med->k_key==NULL）return NULL;

        ret = dup_str(med->k_key->k_keytype);

        return ret;
}




/**
char*   sdp_get_media_keytype(sdp_message_t* sdp, int pos)

        機能：SDP ボディから media の keydata を取り出す.

        引数：sdp : SDP ボディへのポインタ.
                pos : 何番目のメディアかを指定する．通常は 0.

        戻り値：keydata を示す文字列．free する必要あり.
```

エラーの場合は NULL

```
*/
char*   sdp_get_media_keydata(sdp_message_t* sdp, int pos)
{
        char* ret = NULL;
        sdp_media_t* med;

        if (sdp==NULL || sdp->m_medias==NULL) return NULL;
        if (pos<0) pos = 0;
        med = (sdp_media_t*)osip_list_get(sdp->m_medias, pos);
        if (med==NULL || med->k_key==NULL) return NULL;

        ret = dup_str(med->k_key->k_keydata);

        return ret;
}
```

```
/**
char*   sdp_get_media_attr(sdp_message_t* sdp, char* key, int pos)

        機能：SDP ボディから media の 属性の値を取り出す.

        引数：sdp : SDP ボディへのポインタ.
                key : 属性(attribute) を指定する.
                pos : 何番目のメディアかを指定する. 通常は 0.

        戻り値：属性の値を示す文字列. free する必要あり.
                        属性が無い. またはエラーの場合は NULL

*/
char*   sdp_get_media_attr(sdp_message_t* sdp, char* key, int pos)
{
        int     i;
        char* ret = NULL;
        sdp_media_t* med;
```

```
        if（sdp==NULL || key==NULL || sdp->m_medias==NULL） return NULL;
        if（pos<0） pos = 0;
        med =（sdp_media_t*）osip_list_get（sdp->m_medias, pos）;
        if（med==NULL || med->a_attributes==NULL） return NULL;

        i = 0;
        while（!osip_list_eol（med->a_attributes, i））｛
                sdp_attribute_t* attr;
                attr =（sdp_attribute_t*）osip_list_get（med->a_attributes, i）;
                if（!strcmp（key, attr->a_att_field））｛
                        ret = dup_str（attr->a_att_value）;
                        break;;
                }
                i++;
        }

        return ret;
}
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////
//
//      for Debug
//

/**
void   sdp_show_body（sdp_message_t* sdp）
```

機能：SDP ボディの内容の一部を取り出してプリントする．デバッグ用．
取り出し方法の確認用．

引数：sdp：SDP ボディへのポインタ．

see /usr/local/include/osipparser2/sdp_message.h

```
*/
void   sdp_show_body(sdp_message_t* sdp)
{
        int i, j;

        if (sdp==NULL) return;

        print_message("¥n-----------------------------------------¥n");
        print_message("v_version   -> %s¥n", sdp->v_version);
        print_message("o_username -> %s¥n", sdp->o_username);
        print_message("o_sess_id   -> %s¥n", sdp->o_sess_id);
        print_message("o_sess_ver -> %s¥n", sdp->o_sess_version);
        print_message("o_nettype   -> %s¥n", sdp->o_nettype);
        print_message("o_addrtype -> %s¥n", sdp->o_addrtype);
        print_message("o_addr       -> %s¥n", sdp->o_addr);
        print_message("s_name       -> %s¥n", sdp->s_name);
        print_message("i_info       -> %s¥n", sdp->i_info);
        print_message("u_uri        -> %s¥n", sdp->u_uri);
        print_message("z_adjust     -> %s¥n", sdp->z_adjustments);

        if (sdp->c_connection!=NULL) {
                print_message("c_nettype   -> %s¥n", sdp->c_connection->c_nettype);
                print_message("c_addrtype -> %s¥n", sdp->c_connection->c_addrtype);
                print_message("c_addr       -> %s¥n", sdp->c_connection->c_addr);
        }


        // Media
        i = 0;
        while(!osip_list_eol(sdp->m_medias, i)) {
                sdp_media_t*   med;
                med = (sdp_media_t*)osip_list_get(sdp->m_medias, i);

                print_message("¥nMedia %d¥n", i);
                print_message("m_media      -> %s¥n", med->m_media);
                print_message("m_port       -> %s¥n", med->m_port);
                print_message("m_num_port -> %s¥n", med->m_number_of_port);
                print_message("m_protocol -> %s¥n", med->m_proto);
```

```
		if (med->k_key!=NULL) {
			print_message("m_keytype    -> %s¥n",  med->k_key->k_keytype);
			print_message("m_keydata   -> %s¥n", med->k_key->k_keydata);
		}

		j = 0;
		while(!osip_list_eol(med->c_connections, j)) {
			sdp_connection_t* con;
			con = (sdp_connection_t*)osip_list_get(med->c_connections, j);
			print_message("m_nettype   -> %s¥n", con->c_nettype);
			print_message("m_addrtype -> %s¥n", con->c_addrtype);
			print_message("m_addr        -> %s¥n", con->c_addr);
			j++;
		}

		j = 0;
		while(!osip_list_eol(med->a_attributes, j)) {
			sdp_attribute_t* buf;
			buf = (sdp_attribute_t*)osip_list_get(med->a_attributes, j);
			print_message("m_att_field -> %s¥n", buf->a_att_field);
			print_message("m_att_value -> %s¥n", buf->a_att_value);
			j++;
		}

		i++;
	}

	print_message("----------------------------------------¥n¥n");

	return;
}
```

```
/**
int   sdp_print_body(sdp_message_t* sdp)
```

機能：SDP ボディの内容をプリントする．デバッグ用.

sdp_show_body() と違い，ただ垂れ流すだけ.

引数：sdp：SDP ボディへのポインタ.

see /usr/local/include/osipparser2/sdp_message.h

```
*/
void   sdp_print_body(sdp_message_t* sdp)
{
        char* str = NULL;

        if (sdp==NULL) return;
        sdp_message_to_str(sdp, &str);

        if (str==NULL) return;
        print_message("¥n-----------------------------------------¥n");
        print_message(str);
        print_message("-----------------------------------------¥n¥n");
        free(str);

        return;
}
```

```
////////////////////////////////////////////////////////////////////////////////
//
//      Junk Code
//
#if 0

        char* buf;
        sdp_message_t* sdp = NULL;

        sdp_message_init(&sdp);
```

```
        sdp_message_v_version_set(sdp, "0");
        sdp_message_o_origin_set(sdp, "9000", "1111", "1111", "IN", "IP4", "202.26.158.1");
        sdp_message_s_name_set(sdp, "-");
        sdp_message_t_time_descr_add(sdp, "0", "0");
        sdp_message_m_media_add(sdp, "application", "5060", NULL, "APP 0");
        sdp_message_k_key_set(sdp, 0, "-", "-");
        sdp_message_c_connection_add(sdp, 0, "IN", "IP4", "202.26.158.1", NULL, NULL);
        sdp_message_a_attribute_add(sdp, 0, "IP", "202.26.158.1");
        sdp_message_a_attribute_add(sdp, 0, "PORT", "5011");

        sdp_message_to_str(sdp, &buf);
        sdp_print_body(sdp);
*/



/*
        if (sdp->k_key!=NULL) {
                print_message("k_keytype  -> %s¥n", sdp->k_key->k_keytype);
                print_message("k_keydata  -> %s¥n", sdp->k_key->k_keydata);
        }
*/









#endif
```

' sip_app.h

```
#include   "sip_session.h"


#define   MAX_ARGS     100
#define   SIP_PORTNO "5060"
#define   VPN_PORTNO "8000"
#define   VPN_IPADDR "192.168.234.1"

int     main(int, char**);

void   sip_main_term(int signal);
void   sip_child_term(int signal);
```

' sip_app.c

```
#include "sip_app.h"


extern int   DebugMode;


sip_param   SIP_param;

FILE*   Logf = NULL;



int   main(int argc, char** argv)
{
        int   i, j, err, invite=OFF;
        struct sigaction ca, sa;
        char*   localip;
        char*   args[MAX_ARGS];

        int         mport=0;
        int         vport=0;
        int         raddr=0;
        Buffer progname;
        Buffer hostname;
        Buffer username;
        Buffer invtname;
        Buffer portnum;
        Buffer vpn_portnum;
        Buffer vpn_localip;
        Buffer vpn_remoteip;
        Buffer logfile;
        Buffer raddr_temp;

        Buffer sip_to;
        Buffer sip_from;
        Buffer sip_contact;
        Buffer sip_proxy;
```

```
////////////////////////////////////////////////////////////
//          引数処理
//
          progname    = make_Buffer(LNAME);
          hostname    = make_Buffer(LNAME);
          username    = make_Buffer(LNAME);
          invtname    = make_Buffer(LNAME);
          portnum     = make_Buffer(LNAME);
          vpn_portnum      = make_Buffer(LNAME);
          vpn_localip      = make_Buffer(LNAME);
          vpn_remoteip     = make_Buffer(LNAME);
          logfile          = make_Buffer(LNAME);
          raddr_temp               = make_Buffer(LNAME);
          DebugMode   = OFF;

          for (i=1; i<argc; i++) {
                    if                    (!strcmp(argv[i],"-p")) { if (i!=argc-1) copy_s2Buffer
(argv[i+1], &portnum);}
                    else if (!strcmp(argv[i],"-s")) { if (i!=argc-1) copy_s2Buffer(argv[i+1],
&hostname);}
                    else if (!strcmp(argv[i],"-u")) { if (i!=argc-1) copy_s2Buffer(argv[i+1],
&username);}
                    else if (!strcmp(argv[i],"-t")) { if (i!=argc-1) copy_s2Buffer(argv[i+1],
&invtname);}
                    else if (!strcmp(argv[i],"-v")) { if (i!=argc-1) copy_s2Buffer(argv[i+1],
&vpn_portnum);}
                    else if (!strcmp(argv[i],"-l")) { if (i!=argc-1) copy_s2Buffer(argv[i+1],
&vpn_localip);}
                    else if (!strcmp(argv[i],"-r")) { if (i!=argc-1) copy_s2Buffer(argv[i+1],
&vpn_remoteip);}
                    else if (!strcmp(argv[i],"-f")) { if (i!=argc-1) copy_s2Buffer(argv[i+1],
&logfile);}
                    else if (!strcmp(argv[i],"-d")) { DebugMode = ON;}
                    else if (!strcmp(argv[i],"-m")) {
                              if (i!=argc-1) copy_s2Buffer(argv[i+1], &progname);
                              for (i=i+1, j=0; i<argc; i++, j++) {
                                        if (j>=MAX_ARGS) {
                                                  print_message( "main: Too many args....
```

```
Exit!!¥n");
                                        exit(1);
                                }
                                args[j] = argv[i];
                        }
                        args[j] = NULL;
                }
        }
        if (hostname.buf[0]=='¥0' || username.buf[0]=='¥0' || progname.buf=='¥0') {
                print_message("Usage... %s -s proxy[:port] -u user [-p port] [-t call_user]
[-v vpn_wait_port] [-l vpn_localip] [-r vpn_remoteip] [-f log_file] [-d] -m program [args]
¥n", argv[0]);
                exit(1);
        }

        //ローカル IP アドレスの取得 get_myipaddr()は使わない
        localip = get_localip();
        if (localip==NULL) {
                print_message("main: Failure get my IP address!!¥n");
                exit(1);
        }

        // setting sip_wait port
        if (portnum.buf[0]!='¥0') {
                mport = atoi(portnum.buf);
        }
        if (mport==0) {
                mport = atoi(SIP_PORTNO);
                copy_s2Buffer(SIP_PORTNO, &portnum);
        }

        //setting vpn_localip addr
        if (vpn_localip.buf[0]=='¥0') {
                copy_s2Buffer(VPN_IPADDR, &vpn_localip);
        }

        //setting vpn_wait port
        if (vpn_portnum.buf[0]!='¥0') {
                vport = atoi(vpn_portnum.buf);
        }
```

```
        if（vport==0）{
                vport = atoi（VPN_PORTNO）;
                copy_s2Buffer（VPN_PORTNO, &vpn_portnum）;
        }

        // ログファイル
        if（DebugMode==ON && logfile.buf[0]!='¥0'）{
                Logf = fopen（logfile.buf, "w"）;
                TRACE_INITIALIZE（6, Logf）;
        }
        free_Buffer（&logfile）;

        //setting vpn_remoteip addr
        if（vpn_remoteip.buf[0]=='¥0'）{
        raddr_temp = awk_Buffer（vpn_localip, '.', 1）;
        cat_Buffer（&raddr_temp, &vpn_remoteip）;
        cat_s2Buffer（".", &vpn_remoteip）;
        raddr_temp = awk_Buffer（vpn_localip, '.', 2）;
        cat_Buffer（&raddr_temp, &vpn_remoteip）;
        cat_s2Buffer（".", &vpn_remoteip）;
        raddr_temp = awk_Buffer（vpn_localip, '.', 3）;
        cat_Buffer（&raddr_temp, &vpn_remoteip）;
        cat_s2Buffer（".", &vpn_remoteip）;

        raddr = atoi（awk_Buffer（vpn_localip, '.', 4）.buf）;
        if（raddr < 255）{
                raddr++;
        }else{
                fprintf（stderr,"main:assign remoteip failed¥n"）;
                exit（1）;
        }
        cat_s2Buffer（ltostr（raddr）, &vpn_remoteip）;
        }
        free_Buffer（&raddr_temp）;



////////////////////////////////////////////////////////////
//          シグナルハンドリング
//
```

```
        // for Child Process
    ca.sa_handler = sip_child_term;
    ca.sa_flags    = SA_NOCLDSTOP | SA_RESTART;
    sigemptyset(&ca.sa_mask);
    sigaction(SIGCHLD, &ca, NULL);

        // for Parent Process
//      signal(SIGINT, sip_main_term);
        sa.sa_handler = sip_main_term;
        sa.sa_flags    = 0;
        sigemptyset(&sa.sa_mask);
        sigaction(SIGINT,   &sa, NULL);
        sigaction(SIGHUP,   &sa, NULL);
        sigaction(SIGTERM, &sa, NULL);



////////////////////////////////////////////////////////////////
//      パラメータ設定
//
        sip_param_init(&SIP_param);

        sip_proxy    = make_Buffer_str("sip:");
        sip_to         = make_Buffer_str("sip:");
        sip_from  = make_Buffer_str("sip:");
        sip_contact = make_Buffer_str("sip:");

        cat_Buffer(&hostname, &sip_proxy);

        cat_Buffer(&username, &sip_from);
        cat_s2Buffer("@", &sip_from);
        cat_Buffer(&hostname, &sip_from);

        cat_Buffer(&username, &sip_contact);
        cat_s2Buffer("@", &sip_contact);
        cat_s2Buffer(localip, &sip_contact);
        cat_s2Buffer(":", &sip_contact);
        cat_Buffer(&portnum, &sip_contact);

        if (invtname.buf[0]!='¥0') {
```

```
            invite = ON;
            SIP_param.server = OFF;
            cat_Buffer(&invtname, &sip_to);
            cat_s2Buffer("@", &sip_to);
            cat_Buffer(&hostname, &sip_to);
        }

//      SIP_param.expires    = 60;
        SIP_param.proxy       = dup_str(sip_proxy.buf);
        SIP_param.from          = dup_str(sip_from.buf);
        SIP_param.contact    = dup_str(sip_contact.buf);
        SIP_param.to           = dup_str(sip_to.buf);
        SIP_param.prog        = dup_str(progname.buf);
        SIP_param.argv          = args;
        SIP_param.username  = dup_str(username.buf);
        SIP_param.localip    = localip;
        SIP_param.vpn_defaultport = vport;
        SIP_param.localport = vport;

        SIP_param.media         = dup_str("application/vpn");
        SIP_param.protocol  = dup_str("OpenVPN");
        SIP_param.fmtlist    = dup_str("0");

        // for Server
        if (SIP_param.server==ON) {
            SIP_param.localport = vport;
            SIP_param.vpn_localip  = dup_str(vpn_localip.buf);
            SIP_param.vpn_remoteip = dup_str(vpn_remoteip.buf);
            SIP_param.keytype    = dup_str("DH");
            SIP_param.keydata    = dup_str("crypt code");
        }


        if (file_exist(SIP_param.prog)==FALSE) {
            print_message("main: プログラムファイルが存在しない．¥n");
            exit(1);
        }

        SIP_param.ppid = getpid();
```

```
//          不要変数の開放
            free_Buffer(&progname);
            free_Buffer(&hostname);
            free_Buffer(&username);
            free_Buffer(&invtname);
            free_Buffer(&portnum);
            free_Buffer(&vpn_localip);
            free_Buffer(&vpn_remoteip);
            free_Buffer(&vpn_portnum);
            free_Buffer(&sip_proxy);
            free_Buffer(&sip_to);
            free_Buffer(&sip_from);
            free_Buffer(&sip_contact);

            err_message("PROXY     %s¥n", SIP_param.proxy);
            err_message("FROM      %s¥n", SIP_param.from);
            err_message("CONTACT %s¥n", SIP_param.contact);
            err_message("TO        %s¥n", SIP_param.to);
            err_message("PROGRAM %s¥n", SIP_param.prog);
            err_message("VPN_LOCAL %s¥n", SIP_param.vpn_localip);
            err_message("VPN_REMOTE %s¥n", SIP_param.vpn_remoteip);
            if (SIP_param.server==ON) {
                        err_message("VPN_PORT %s¥n", ltostr(SIP_param.localport));
            }


////////////////////////////////////////////////////////////
//          eXosip 初期化
//
            err = eXosip_init();
            if (err!=0) {
                        if (Logf!=NULL) fclose(Logf);
                        print_message("main: eXosip_init: ERROR!!¥n");
                        exit(1);
            }

            err = eXosip_listen_addr(IPPROTO_UDP, NULL, mport, AF_INET, 0);
            if (err!=0) {
                        print_message("main: eXosip_listen_addr: ERROR!!¥n");
```

```
                eXosip_quit();
                exit(1);
        }

        eXosip_set_user_agent("SIP for APP b1 rev.45");
        eXosip_add_authentication_info("infosys", "infosys", "infosysipass", NULL, NULL);



////////////////////////////////////////////////////////////
//      SIP
//
        // REGISTER
        do {
                err = send_regist(&SIP_param);
        } while(err!=EXOSIP_REGISTRATION_SUCCESS);


        // INVITE 送信
        if (invite==ON) {
                err = send_invite(&SIP_param);
                if (err==0) {
                        err = session_start(&SIP_param);      // ここで子プロセスを起動
                        if (err>0) {
                                session_term_wait(&SIP_param);
                                session_terminate(&SIP_param);
                        }
                }
                else {
                        print_message("main: 相手が呼び出しに応じません．INVITE 失
敗!!¥n");
                }
        }


        // INVITE 待ち
        else {
                Loop {
                        err = recive_invite(&SIP_param);      // ここで子プロセスを起動
                        if (err>0) {
```

```
                                        err = session_start(&SIP_param);
                                        if (err==0) {
                                                session_term_wait(&SIP_param);
                                                session_terminate(&SIP_param);
                                        }
                                }
                        }
                }

                sip_main_term(0);
}

////////////////////////////////////////////////////////////////////////
// プログラムの終了
//

//
// for メインプロセス
//
void   sip_main_term(int signal)
{
        int err;

        err = sip_terminate(&SIP_param);
        if (err==1) return;             // スレッドはそのままリターンさせる.

        if (Logf!=NULL) {
                fclose(Logf);
                Logf = NULL;
        }
        exit(signal);
```

```
}



//
// for 子プロセス
//
void   sip_child_term(int signal)
{
        pid_t pid = 0;
    int ret;

    do {
        pid = waitpid(-1, &ret, WNOHANG);
    } while(pid>0);

        SIP_param.inChild = OFF;
}
```

· sip_session.h

```
#include   "sip_tools.h"
#include   "sdp_tools.h"



int   send_regist(sip_param* param);

int   send_invite(sip_param* param);
int   recive_invite(sip_param* param);


int   session_start(sip_param* param);
int   session_term_wait(sip_param* param);
int   session_terminate(sip_param* param);

int   get_sdp_paramter(sip_param* param, sdp_message_t* sdp);
```

' sip_session.c

```
#include "sip_session.h"




////////////////////////////////////////////////////////////////////
//
//          REGISTER
//

int   send_regist(sip_param* param)
{
          int err;

          err = sip_regist_send_and_answer(param);
          if (err==EXOSIP_REGISTRATION_SUCCESS) {
                    err_message("send_regist: Registration Success.¥n");
          }
          else {
                    err_message("send_regist: Registration Failure.¥n");
                    return -1;
          }

          sip_regist_thread(param);

          return err;
}
```

・sip_tools.h

```
/**
  SIP 用ツール                                    by Fumi.Iseki
                                                  （C）2006 9/1


        注意：
                eXosip_lock(), eXosip_unlock() の間で sleep() しないこと.
*/

#ifndef _SIP_TOOLS_H
#define _SIP_TOOLS_H




#include "common.h"
#include "tools.h"
#include "network.h"
#include "buffer.h"

#include <strings.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/ioctl.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <signal.h>
#include <syslog.h>
#include <eXosip2/eXosip.h>



/**/

#define   SIP_USLEEP_TIME 10000
```

```
typedef struct _sip_param
{
        int tid;                        // ID for transactions（to be used for answers）
        int rid;                        // REGISTER ID
        int iid;                        // INVITE ID
        int cid;                        // CALL ID
        int did;                        // DIALOG ID

        int type;                       // 受信した Event Type
        int auth;                       // 認証を行うか？　デフォルトで ON
        int server;                             // サーバか？
        int ppid;                       // 親プロセス（メインプロセス）の PID
        int cpid;                       // 子プロセス（起動したプログラム）の PID

        char*   prog;                   // 起動するプログラムのパス
        char** argv;                    // 起動するプログラムへの引数
        char** env;                             // 起動するプログラムへの環境変数

        int inRegist;                   // （再）REGIST 中
        int inInvite;                   // INVITE 中
        int inWait;                             // 待機中
        int inSession;                  // セッション中
        int inExit;                             // 終了作業中
        int inChild;                    // 子プロセス作動中

        char* from;                             // SIP from
        char* to;                       // SIP to
        char* proxy;                    // SIP Proxy
        char* contact;                  // SIP contact
        int     expires;                // SIP expires

        char*   localip;                        // 自分の IP アドレス
        char*   remoteip;               // 相手の IP アドレス
        char*   vpn_localip;            // 自分の VPN IP アドレス
        char*   vpn_remoteip;           // 相手の VPN IP アドレス
        char*   username;               // 自分のユーザ名（番号）
        char*   media;                          // SDP メディア情報
        char*   protocol;               // SDP プロトコル
        char*   fmtlist;                        // SDP フォーマットリスト
        char*   keytype;                        // 暗号用キータイプ
```

```
        char*   keydata;                    // 暗号用キー
        int     vpn_defaultport;    // 自分の子プロセスの Default ポート番号.
        int     localport;          // 自分の子プロセスのポート番号．または自分のメイ
ンプロセスのポート番号.
        int     remoteport;         // 相手の子プロセスのポート番号．または相手のメイ
ンプロセスのポート番号.

        osip_message_t* mssg;       // 汎用メッセージ
        osip_message_t* rgst;       // レジスト用メッセージ
        osip_message_t* invt;       // インバイト用メッセージ

} sip_param;



void sip_param_init(sip_param* param);
void sip_param_free(sip_param* param);
int   sip_terminate(sip_param* param);


int   sip_regist_send(sip_param* param);
int   sip_regist_answer(sip_param* param);
int   sip_regist_send_and_answer(sip_param* param);
int   sip_regist_resend(sip_param* param);

int   sip_regist_thread(sip_param* param);
static void*   sip_regist_thread_loop(void* arg);


int   sip_invite_send(sip_param* param);
int   sip_invite_answer(sip_param* param);
int   sip_invite_send_and_answer(sip_param* param);

int   sip_invite_wait(sip_param* param);
int   sip_invite_accept(sip_param* param);

int   sip_session_term_wait(sip_param* param);
int   sip_session_terminate(sip_param* param);
int   sip_session_start(sip_param* param);
```

```
int   sip_send_ack(sip_param* param);
int   sip_event_get(sip_param* param);
int   sip_process_fork(sip_param* param);


/**/



#endif
```

・sip_tools.c

```
/**
  SIP 用ツール                                    by Fumi.Iseki
                                                  （C）2006 9/1
*/



#include "sip_tools.h"
#include "sdp_tools.h"




/**
void   sip_param_init（sip_param* param）

     機能：SIP 用パラメータを初期化する.

          引数：param   SIP 用パラメータ

          戻り値：なし

*/
void   sip_param_init（sip_param* param）
{
          param->tid    = -1;
          param->rid    = -1;
          param->iid    = -1;
          param->cid    = 0;
          param->did    = 0;

          param->type    = -1;
          param->auth    = ON;
          param->server = ON;
          param->cpid    = -1;
          param->ppid    = -1;

          param->prog = NULL;
```

```
        param->argv = NULL;
        param->env  = NULL;

        param->inInvite  = OFF;
        param->inRegist  = OFF;
        param->inSession = OFF;
        param->inWait    = OFF;
        param->inExit    = OFF;
        param->inChild   = OFF;

        param->from      = NULL;
        param->to        = NULL;
        param->proxy     = NULL;
        param->contact   = NULL;
        param->expires   = 3600;

        param->localip   = NULL;
        param->remoteip  = NULL;
        param->vpn_localip   = NULL;
        param->vpn_remoteip  = NULL;
        param->username  = NULL;
        param->media     = NULL;
        param->protocol  = NULL;
        param->fmtlist   = NULL;
        param->keytype   = NULL;
        param->keydata   = NULL;
        param->vpn_defaultport = 0;
        param->localport = 0;
        param->remoteport= 0;

        param->mssg      = NULL;
        param->rgst      = NULL;
        param->invt      = NULL;
}



/**
```

```
void   sip_param_free(sip_param* param)

    機能：SIP 用パラメータ変数をクリアする.
          ただし，eXosip のメッセージ変数はクリアされないので
          eXosip_quit と併用する.

        引数：param   SIP 用パラメータ

        戻り値：なし

*/
void   sip_param_free(sip_param* param)
{
        if (param->prog!=NULL)        free(param->prog);
//      if (param->argv!=NULL)        free(param->argv);
//      if (param->env!=NULL)         free(param->env);

        if (param->from!=NULL)       free(param->from);
        if (param->to!=NULL)         free(param->to);
        if (param->proxy!=NULL)      free(param->proxy);
        if (param->contact!=NULL)   free(param->contact);

        if (param->localip!=NULL)   free(param->localip);
        if (param->remoteip!=NULL) free(param->remoteip);
        if (param->vpn_localip!=NULL)   free(param->vpn_localip);
        if (param->vpn_remoteip!=NULL) free(param->vpn_remoteip);
        if (param->username!=NULL) free(param->username);
        if (param->media!=NULL)      free(param->media);
        if (param->protocol!=NULL) free(param->protocol);
        if (param->fmtlist!=NULL)   free(param->fmtlist);
        if (param->keytype!=NULL)   free(param->keytype);
        if (param->keydata!=NULL)   free(param->keydata);

/*
        if (param->invt!=NULL) osip_free(param->mssg);
        if (param->rgst!=NULL) osip_free(param->rgst);
        if (param->invt!=NULL) osip_free(param->invt);
*/
        sip_param_init(param);
}
```

```
/**
int   sip_terminate(sip_param* param)

    機能：parm->ppid で指定したプロセスの SIP セッションを終了する.

        引数：param   SIP 用パラメータ
                    param->ppid      セッションを停止させるプロセスの PID. 通常は一番上（親）の PID.

    戻り値：0:  していされたプロセスのセッションを停止させた. 呼び出し側でそのまま終了させる.
                        1:  指定されたプロセスではないので何もしなかった. 呼び出し側でそのままリターンさせる.

*/
int   sip_terminate(sip_param* param)
{
        int   pid;

        pid = getpid();

    if (param->inExit!=ON && param->ppid==pid) {
        param->inExit = ON;

                //debug_message("sip_terminate:¥n");

                // BYE を送信
                if (param->inInvite==ON || param->inSession==ON) {
                        if (param->inInvite ==ON) param->inInvite   = OFF;
                        if (param->inSession==ON) param->inSession = OFF;
                        sip_session_terminate(param);
                }

                // REGIST を抹消
                param->expires = 0;
```

```
                sip_regist_resend(param);

                eXosip_lock();
        eXosip_register_remove(param->rid);
                eXosip_unlock();
                eXosip_quit();

        sip_param_free(param);
                return 0;
        }
    return 1;
}
```

```
////////////////////////////////////////////////////////////////////////////////////////////////
//
// REGISTER
//

/**
int sip_regist_send_and_answer(sip_param* param)

    機能：パラメータの内容に従って，サーバへレジストして返答を待つ.

        引数：param   SIP 用パラメータ

    戻り値： 0 以上:  レジストに対する  eXosip メッセージ
                                                    see  /usr/local/include/eXosip2/eXosip.h
                        0 未満:  エラー
*/
int sip_regist_send_and_answer(sip_param* param)
{
        int err;
```

```
	while（param->inInvite==ON）{
		err_message("Sleeping in sip_regist_send_and_answer.¥n");
		usleep(SIP_USLEEP_TIME);
	}
	param->inRegist = ON;

	err = sip_regist_send(param);
	if（err<0）{
		param->inRegist = OFF;
		return err;
	}

	err = sip_regist_answer(param);
	param->inRegist = OFF;

	return err;
}



/**
int  sip_regist_send(sip_param* param)

    機能：パラメータの内容に従って，サーバへレジストメッセージを送信する.

        引数：param  SIP 用パラメータ

    戻り値： 0 以上： 正常にレジストメッセージをサーバに送信した.
                    0 未満： エラー
*/
int  sip_regist_send(sip_param* param)
{
	int err;

	eXosip_lock();
	err = eXosip_register_build_initial_register(param->from, param->proxy, param->
contact, param->expires, &(param->rgst));
	param->rid = err;
```

```
        if (err<0) {
                eXosip_unlock();
                return err;
        }

        err = eXosip_register_send_register(param->rid, param->rgst);
        eXosip_unlock();
        return err;
}




/**
int   sip_regist_resend(sip_param* param)

    機能：パラメータの内容に従って，サーバへレジストメッセージを再送信する.

        引数：param   SIP 用パラメータ

    戻り値： 0 以上： 正常に再レジストメッセージをサーバに送信した.
                    0 未満： エラー
*/
int   sip_regist_resend(sip_param* param)
{
        int err;

        eXosip_lock();
        err = eXosip_register_build_register(param->rid, param->expires, &(param->rgst));
        if (err<0) {
                eXosip_unlock();
                return err;
        }

        err = eXosip_register_send_register(param->rid, param->rgst);
        eXosip_unlock();
        return err;
}
```

```
/**
int   sip_regist_answer(sip_param* param)

      機能：レジストに対するサーバからの返答を待つ.

            引数：param   SIP 用パラメータ

      戻り値： 0 以上:  レジストに対する eXosip メッセージ
                                                      see  /usr/local/include/eXosip2/eXosip.h
                        0 未満:  エラー
*/
int   sip_regist_answer(sip_param* param)
{
            int   err, etype;
            int   brk_flg = OFF;

      eXosip_event_t *event;

            Loop {
            event = eXosip_event_wait(0, 0);
            if (event==NULL) {
                        usleep(SIP_USLEEP_TIME);
                        continue;
                  }
                  etype = event->type;
            debug_message("sip_regist_answer: Event (type, did, cid) = (%d, %d, %d)¥n", event-
>type, event->did, event->cid);

                  if (etype == EXOSIP_REGISTRATION_SUCCESS) {
                  //debug_message("sip_regist_answer: Registration Success.¥n");
                        brk_flg = ON;
                  }
                  else if (etype == EXOSIP_REGISTRATION_FAILURE) {
               if (param->auth==ON) {
                              err = sip_regist_resend(param);        // 認証を行う場合
はもう一度送る
                              param->auth = OFF;
```

```
                        if（err<0）brk_flg = ON;
                    }
                    else {
                        debug_message( "sip_regist_answer: Registration
Failure.¥n");
                        brk_flg = ON;
                    }
        }
            else {
            err_message("sip_regist_answer: Unexpected Event. event(type, did, cid) =
(%d, %d, %d)¥n",
                                                    event->type, event-
>did, event->cid);
            }

            param->type = etype;
            param->cid  = event->cid;
            param->did  = event->did;
        eXosip_event_free(event);
            if（brk_flg==ON）break;
        }

        param->auth = ON;// デフォルトは ON という事で....
        return etype;
}




/**
int  sip_regist_thread(sip_param* param)

    機能：再レジスト用のスレッド生成を行う.

        スレッドの機能：
            再レジストに失敗した場合，試行時間間隔を短縮(1/2)して再試行する.
            最小の再レジスト間隔は 10s
```

```
        引数：param   SIP 用パラメータ

    戻り値： スレッド生成 pthread_create() の結果. pthread_create のマニュアルを見よ.

*/
int   sip_regist_thread(sip_param* param)
{
        int err;
        pthread_t reg_thread;

        err = pthread_create(&reg_thread, NULL, sip_regist_thread_loop, (void*)param);
        if (err!=0) {
                err_message("sip_regist_thread: Thread Create Failure.¥n");
        }

        return err;
}




/**
static void*   sip_regist_thread_loop(void* arg)

    機能：sip_regist_thread() から呼び出されるスレッド本体.
          通常は直接呼び出されることは無い.

                再レジストに失敗した場合，試行時間間隔を短縮(1/2)して再試行する.
                最小の再レジスト間隔は 10s

        引数：param   SIP 用パラメータ void* へキャスト.

*/
static void*   sip_regist_thread_loop(void* arg)
{
        int err;
        int exp;
```

```
	sip_param *param = arg;

	exp = param->expires/2;


	Loop {
		sleep(exp);

		if (param->inExit==ON) return;			// 終了処理中なので中断してリターンする.
		param->inRegist = ON;
		while (param->inInvite==ON || param->inWait==ON) {  // 他の仕事が終わるまで待つ.
			err_message("Sleeping in sip_regist_thred_loop¥n");
			usleep(SIP_USLEEP_TIME);
		}

		err = sip_regist_resend(param);

		if (err==0) {
			err = sip_regist_answer(param);
			if (err==EXOSIP_REGISTRATION_SUCCESS) {
				debug_message("sip_regist_thread_loop: Re-Registed.¥n");

				exp = param->expires/2;
			}
			else {
				debug_message("sip_regist_thread_loop: Re-Registration Failure. %d¥n", err);
				exp = exp/2;
				if (exp<10) exp = 10;
			}
		}
		else {
			debug_message("sip_regist_thread_loop: Re-Registration Send error. %d¥n", err);
			exp = exp/2;
			if (exp<10) exp = 10;
		}
		param->inRegist = OFF;
```

```
        }

        return;
}
```

```
////////////////////////////////////////////////////////////////////////////////////////////////
//
// INVITE
//


/**
int sip_invite_send_and_answer(sip_param* param)

    機能：パラメータの内容に従って，インバイトを送信して返答を待つ.

        引数：param   SIP 用パラメータ

    戻り値： 0 以上:  インバイトに対する eXosip メッセージ
                                                    see  /usr/local/include/eXosip2/eXosip.h
                    0 未満:  エラー
*/
int sip_invite_send_and_answer(sip_param* param)
{
        int err;

        while（param->inRegist==ON）{
                err_message("Sleeping in sip_invite_and_anser.¥n");
                param->inInvite = OFF;        // Dead Lock 防止   Register 優先.
                usleep(SIP_USLEEP_TIME);
        }
        param->inInvite = ON;
```

```
	err = sip_invite_send(param);
	if (err<0) {
		param->inInvite = OFF;
		return err;
	}
	param->iid = err;

	err = sip_invite_answer(param);
	param->inInvite = OFF;

	return err;
}
```

```
/**
int  sip_invite_send(sip_param* param)

    機能：パラメータの内容に従って，サーバへインバイトメッセージを送信する.

        引数：param  SIP 用パラメータ

    戻り値： 0 以上： INVITE ID, 正常にインバイトメッセージをサーバに送信した.
                    0 未満： エラー
*/
int  sip_invite_send(sip_param* param)
{
	int err;

	eXosip_lock();
	err = eXosip_call_build_initial_invite(&(param->invt), param->to, param->from,
NULL, NULL);
	if (err<0) {
		eXosip_unlock();
		return err;
	}
```

```
    err = eXosip_call_send_initial_invite(param->invt);
    eXosip_unlock ();

        return err;
}




/**
int   sip_invite_answer(sip_param* param)

    機能：インバイトに対するサーバからの返答を待つ.

        引数：param   SIP 用パラメータ

    戻り値： 0 以上:  インバイトに対する eXosip メッセージ
                                                    see  /usr/local/include/eXosip2/eXosip.h
                        0 未満:  エラー
*/
int   sip_invite_answer(sip_param* param)
{
        int   err, etype;
        int   brk_flg = OFF;
        int   suc_flg = OFF;

    eXosip_event_t *event;

        Loop {
        event = eXosip_event_wait(0, 100);
        if (event==NULL) continue;
                etype = event->type;
        err_message("sip_invite_answer: Event (type, did, cid) = (%d, %d, %d)¥n", event->
type, event->did, event->cid);


                // 無視するイベント
            if (etype==EXOSIP_CALL_INVITE) {                    // a new
call
```

```
                debug_message("sip_invite_answer: Call Invite.  %d¥n", etype);
                }
        else if (etype==EXOSIP_CALL_REINVITE) {                    // a new INVITE
within call
                debug_message("sip_invite_answer: Call ReInvite.  %d¥n", etype);
                }
                else if (etype==EXOSIP_CALL_ACK) {
// ACK received for 200ok to INVITE
                debug_message("sip_invite_answer: Call ACK.  %d¥n", etype);
                }
                else if (etype==EXOSIP_CALL_PROCEEDING) {                  //
processing by a remote app
                debug_message("sip_invite_answer: Remote App.  %d¥n", etype);
                }
            else if (etype==EXOSIP_CALL_RINGING) {                         // 呼び出
し中.
                debug_message("sip_invite_answer: Call Ringing.  %d¥n", etype);
                }


                // ループを終了するイベント（接続）
            else if (etype==EXOSIP_CALL_ANSWERED) {              // 通話開始
                debug_message("sip_invite_answer: Call Answer.  %d¥n", etype);
                        suc_flg = ON;
                        brk_flg = ON;
                }


                // ループを終了するイベント（エラー）
            else if (etype==EXOSIP_CALL_REQUESTFAILURE) {      // 電話に出ない
                debug_message("sip_invite_answer: Request Failure.  %d¥n", etype);
                        brk_flg = ON;
                }
            else if (etype==EXOSIP_CALL_NOANSWER) {                        //  no
answer within the timeout
                debug_message("sip_invite_answer: Time out.  %d¥n", etype);
                        brk_flg = ON;
                }
            else if (etype==EXOSIP_CALL_SERVERFAILURE) {       // a server failure
                debug_message("sip_invite_answer: Server Failure.  %d¥n", etype);
```

```
                brk_flg = ON;
            }
        else if（etype==EXOSIP_CALL_GLOBALFAILURE）{        // a global failure
            debug_message("sip_invite_answer: Global Failure.  %d¥n", etype);
                brk_flg = ON;
            }
    else if（etype==EXOSIP_CALL_CANCELLED）{                // that call has been
cancelled
            debug_message("sip_invite_answer: Call Cancelled.  %d¥n", etype);
                brk_flg = ON;
            }
    else if（etype==EXOSIP_CALL_CLOSED）{                    // a BYE was
received for this call
            debug_message("sip_invite_answer: Call Closed.  %d¥n", etype);
                brk_flg = ON;
            }
            else if（etype==EXOSIP_CALL_RELEASED）{
// call context is cleared.
            debug_message("sip_invite_answer: Call Released.  %d¥n", etype);
                brk_flg = ON;
            }


            // その他のイベントは無視する
            else {
            debug_message("sip_invite_answer: Continue Event. event(type, did, cid) =
(%d, %d, %d)¥n",
                                                    event-> type,  event-> did,
event->cid);
    }


            param->type = etype;
            param->cid  = event->cid;
            param->did  = event->did;

    eXosip_event_free(event);
            if（brk_flg==ON）break;
    }
```

```
        if（suc_flg==OFF）{
        debug_message("sip_invite_answer: END Event = %d.¥n", etype);
        }

        return etype;
}
```

```
/**
eXosip_event_t*   sip_invite_wait(sip_param* param)

    機能：INVITE イベントを待つ

        引数：param   SIP 用パラメータ

    戻り値： 0： INVITE メッセージを受信.
                    0 未満： エラー

//      戻り値： イベント変数(eXosip_event_t) へのポインタ. 失敗した場合は NULL.

//                          イベント変数はどこかでフリーしないといけない.
eXosip_event_free()

*/
int   sip_invite_wait(sip_param* param)
{
        int   err, etype;
    eXosip_event_t *event;


        Loop {
                param->inWait = OFF;
                while（param->inRegist==ON）{
```

```
                        err_message("Sleeping in sip_invite_wait.¥n");
                        usleep(SIP_USLEEP_TIME);
                }
                param->inWait = ON;

        event = eXosip_event_wait(0, 100);

        if (event==NULL) {
                        continue;
                }
                etype = event->type;
        err_message("sip_invite_wait_event: Event (type, did, cid) = (%d, %d, %d)¥n",
event->type, event->did, event->cid);

                param->type = etype;
                param->tid  = event->tid;
                param->cid  = event->cid;
                param->did  = event->did;
        eXosip_event_free(event);

                if (etype==EXOSIP_CALL_INVITE || etype==EXOSIP_CALL_REINVITE)
{
                        param->inInvite = ON;
                        break;
                }
        //eXosip_event_free(event);
        }

        param->inWait = OFF;
        return 0;
        //return event;
}




/**
int  sip_invite_accept(sip_param* param)

        機能：INVITE 要求を処理する.
```

```
        引数：param   SIP 用パラメータ

    戻り値： 0 以上 ：子プロセスのプロセス ID. 正常終了
                    0 未満 ：エラー

*/
int   sip_invite_accept(sip_param* param)
{
        int i, err, pid;


        // 100 Trying 送信
        eXosip_lock();
        eXosip_call_send_answer(param->tid, 100, NULL);
        eXosip_unlock();


        // 180 Ringing 送信
        eXosip_lock();
        eXosip_call_send_answer(param->tid, 180, NULL);
        eXosip_unlock();


        // プロセスの起動
        pid = sip_process_fork(param);
        if (pid<0) {
                debug_message("sip_invite_accept: 子プロセスの起動に失敗.¥n");
                sip_session_terminate(param);
                return -1;
        }
        param->cpid = pid;


        // 200 OK 送信
        eXosip_lock();
        err = eXosip_call_build_answer(param->tid, 200, &param->mssg);
        if (err==0) {
                sdp_set_body2(param, param->mssg);
                err = eXosip_call_send_answer(param->tid, 200, param->mssg);
```

```
                //osip_message_free(param->mssg);
                //param->mssg = NULL;
        }
        eXosip_unlock();
        param->inInvite = OFF;


        if (err!=0) {
                debug_message("sip_invite_accept: 200 OK の送信に失敗.¥n");
                sip_session_terminate(param);
                return -1;
        }
        else {
                param->inSession = ON;
        }

        return pid;
}




////////////////////////////////////////////////////////////////////////////////
//
// SESSION
//


/**
int   sip_session_term_wait(sip_param* param)

        機能：切断要求を待つ.

        引数：param   SIP 用パラメータ

    戻り値： 0 以上:  インバイトに対する eXosip メッセージ
                                                see  /usr/local/include/eXosip2/eXosip.h
                        0 未満:  エラー
```

```
*/
int   sip_session_term_wait(sip_param* param)
{
        int   err, etype;
      eXosip_event_t *event;

      //debug_message("sip_session_term_wait: Wait termination¥n");

        Loop {
                param->inSession = OFF;
                while (param->inRegist==ON) {
                        err_message("Sleeping in sip_session_term_wait.¥n");
                        usleep(SIP_USLEEP_TIME);
                }
                param->inSession = ON;

                if (param->inChild==OFF) break;      // 子プロセスは終了している.

        event = eXosip_event_wait(0, 100);

        if (event==NULL) {
                        continue;
                }
                etype = event->type;
        err_message("sip_session_term_wait: Event (type, did, cid) = (%d, %d, %d)¥n",
event->type, event->did, event->cid);

                param->type = etype;
                param->tid   = event->tid;
                param->cid   = event->cid;
                param->did   = event->did;
        eXosip_event_free(event);

                 if (etype==EXOSIP_CALL_CLOSED ||
etype==EXOSIP_CALL_CANCELLED || etype==EXOSIP_CALL_RELEASED) {
                        break;
                }

        }
```

```
        param->inSession = OFF;
        return etype;
}
```

```
/**
int   sip_session_terminate(sip_param* param)

        機能：接続を切断する.

        引数：param   SIP 用パラメータ

    戻り値： 0 : 正常終了
                    0 未満 ：エラー

*/
int   sip_session_terminate(sip_param* param)
{
        if (param->cpid>0) {
                param->inChild = OFF;
                kill(param->cpid, SIGTERM);
        }

        // BYE を送信
        eXosip_lock();
        eXosip_call_terminate(param->cid, param->did);
        eXosip_unlock();
        sleep(2);

        return 0;
}
```

```
/**
int   sip_session_start(sip_param* param)

      機能：セッションを開始する.
      引数：param   SIP 用パラメータ

      戻り値： 呼び出し（クライアント）の場合は 子プロセスの PID. サーバの場合は 0
が返る．i
                  0 未満 ：エラー

*/
int   sip_session_start(sip_param* param)
{
      int pid = 0;
      if (param->server==OFF) {
            pid = sip_process_fork(param);

            if (pid<0) {
                  debug_message("sip_session_start: 子プロセスの起動に失敗.¥n");
                  sip_session_terminate(param);
                  return -1;
            }
            param->cpid = pid;
      }
            return pid;
}
```

```
////////////////////////////////////////////////////////////////////////////////
//
//
```

```
/**
int   sip_send_ack(sip_param* param)


        戻り値： 0: 正常終了
                 0 未満: エラー
*/
int   sip_send_ack(sip_param* param)
{
        int err;

    eXosip_lock();
    err = eXosip_call_build_ack(param->did, &param->mssg);
    if (err!=0) {
                sip_session_terminate(param);
        }
        else {
                err = eXosip_call_send_ack(param->did, param->mssg);
        }
    eXosip_unlock();

        return err;
}




/**
int   sip_event_get(sip_param* param)

    機能：

        引数：param   SIP 用パラメータ

    戻り値： 0 以上:  インバイトに対する eXosip メッセージ
                                                    see. /usr/local/include/eXosip2/eXosip.h
                    0 未満:  エラー
*/
```

```
int   sip_event_get(sip_param* param)
{
        int   err, etype;
      eXosip_event_t *event;


        Loop {
                param->inWait = OFF;
                while (param->inRegist==ON) {
                        err_message("Sleeping in sip_get_event.¥n");
                        usleep(SIP_USLEEP_TIME);
                }
                param->inWait = ON;

        event = eXosip_event_wait(0, 100);

        if (event==NULL) {
                        continue;
                }
                etype = event->type;
        err_message("sip_event_get: Event (type, did, cid) = (%d, %d, %d)¥n", event->type,
event->did, event->cid);

                param->type = etype;
                param->tid   = event->tid;
                param->cid   = event->cid;
                param->did   = event->did;
        eXosip_event_free(event);


                if (etype==EXOSIP_CALL_INVITE) {
                        param->inInvite = ON;
                        err = sip_invite_accept(param);
                        param->inInvite = OFF;
                }

        }

        return etype;
}
```

```
int   sip_process_fork(sip_param* param)
{
        int pid;

        if (file_exist(param->prog)==FALSE) return -1;

        param->inChild = ON;

        pid = fork();
        if (pid==0) {
                execve((const char*)(param->prog), param->argv, param->env);
          //fprintf(stderr, "RemoteIP:%s RemotePort:%s ¥n", param->remoteip, param->
remoteport);
                print_message("sip_process_fork: 致命的：子プロセスを起動できない. %s
%s %s¥n", param->prog, param->argv, param->env);
                print_message("sip_process_fork: param->argv：%s %s %s %s %s¥n",
param->argv[0], param->argv[1], param->argv[2],param->argv[3],param->argv[4]);

                exit(1);
        }
        return pid;
}
```