IPsec と IPv6

山本和彦 (IIJ 技術研究所)

1998 年 12 月 15 日

InternetWeek 98 国立京都国際会館

(社)日本ネットワークインフォメーションセンター編

1. 概要

2. IPsec

VPN と IPsec

セキュリティの階層と粒度

IPsec の用途、構成、モード、アーキテクチャ

インターネット・セキュリティの保護機能

暗号

IPsec と NAT

鍵配送

IPsec と IKE で利用可能な暗号

IPsec と IKE の実装状況

3. IPv6

次世代 IP の必要性

IPv6 は NAT のいらない世界

IPv6 の特徴

アドレス、プラグ&プレイ、到達不能検知、向け直しなど

IPv6 の現状

4. IPv6 への移行

移行のストーリー

デュアル・スタック

IPv6 in IPv4 トンネル

トランスレータ

プロトコル変換

アドレスの対応付け

トランスレータの分類

5. Q&A

ルーティングプロトコルの現状と Internet2 **との関係**

IPv6 の需要

1. 概要

IPsec と IPv6 について、その仕組みや設計思想、最近の動向などを説明します。IPsec では、インターネットにおけるセキュリティ、アーキテクチャ、認証ヘッダ、暗号ペイロード、IKE(鍵交換プロトコル)、また現在の実装状況などを説明しています。IPv6 では、その仕様と拡張ヘッダ、アドレス・アーキテクチャ、プラグ&プレイ、および現在の状況などを説明します。また、IPv6 への移行についてもあわせて説明します。

2. IPsec

VPN と IPsec

・TCP/IP レベルのセキュリティ
  暗号ペイロード(Encapsulating Security Payload)
  認証ヘッダ(Authentication Header)
・VPN(Virtual Private Network)
  専用線で各部所を接続するのは高価
  サイトの出口で暗号トンネルを張る
  安価なインターネット上にプライベート・ネットワークを構築

IPsec というと、VPN(Virtual Private Network)との関連性を思い浮かべる人が多いと思います。IPsec とは正確にいうと TCP/IP レベルのセキュリティです。
セキュリティという技術はいくつかありますが、どれも一長一短で完璧なものというのはありません。セキュリティには IPsec で守れるものもあれば、IPsec では難しいというものもありますので、必要に応じて使い分けることが大切です。
IPsec は大きくわけて「暗号ペイロード」というものと、「認証ヘッダ」というものからなっています。この IPsec の用途として、例えば電子メールのような、オブジェクトの保護というセキュリティの確保には使えませんが、TCP/IP レベルのネットワークを作る VPN には使用でき、これが IPsec の典型的な用途です。
二つのサイトを接続する場合、専用線を買えば本当に安全なネットワークを作ることができますが、高価なものになります。そこでインターネットという安価なネットワークの上に暗号のトンネルを引いて、あたかもプライベートなネットワークでつないでいるようなものを構築するというのが「VPN です。

セキュリティの階層と粒度

ネットワークはすべてそうですが、インターネットもいくつかの階層になっています。よく使われている OSI の 7 階層モデルというのはインターネットを説明するうえではあまり現実に即していませんので、われわれはこのような簡単な層に分けて考えています。データリンク層などもありますが、パケットを配送する「IP 層」、その上にバーチャルサーキットと呼ばれ、信頼性の保持をおこなう「TCP 層」、さらにそれを扱うアプリケーション・プログラム・インターフェースである「ソケット層」があり、「アプリケーション層」というのがあるわけです。それぞれをどういう技術で守れるかということで、一番わかりやすい例としては、アプリケーション層には「暗号メール」というものがあります。最近、S-MIME や PGP-MIME などというものをお聞きになったことがあると思いますが、それが暗号メールです。

ここで理解して頂きたいのはアプリケーション層というのはアプリケーションに特化したオブジェクトがあり、例えば電子メールなら電子メールを保護するものです。ですから通信中はオブジェクト以外のところは単なる平文で、守りたいオブジェクトのみが暗号化されていたりします。また受信した後も暗号化されたまま保存されるというような特徴があります。

下に下っていくと、ソケットのところでセキュリティを守っているものに、「SSL(Secure Socket Layer)」というものがあります。これは皆さんも多分使ったことがあると思います。web 上での買い物でクレジットカード番号などを入力するときに、Netscape の場合には鍵がつながったという状況になりますが、これがそうです。これはオブジェクトではなく、単なる通信を守っています。ですからその通信路を通っている間はクレジットカード番号は暗号化されていますが、着いた向こうの地点では平文に戻ってしまうというような特徴があります。

それから、さらに下っていきますと TCP 層や IP 層を守るのが **IPsec** です。具体的にどういうものを守れるかというのは、後で具体的に説明します。

この階層構造を考えると、対象としては上に行くほど粒度が細かくなって、オブジェクトとか、あるいは人とか、そういうものになってきますが、下に行けばネットワーク全体とかホスト全体のような粒度の大きなものになります。

暗号速度とか署名を取る速度というのがあるわけですが、下に行けば行くほど通信を守っ

ていますから、暗号速度が速いものでないと使えないということになります。上に行くと、電子メールというのは暗号化するのに 1 分かかっても 1 分後に届けばいいわけで、これは遅くても大丈夫というような特徴があります。

IPsec の用途

・ネットワーク単位の通信保護

VPN の構築

・ホスト単位の通信保護

・TCP の保護

トランザクション指向の TCP の保護

TCP リセット攻撃の防止

・UDP の保護

・IP に直接のっているプロトコルの保護

IPsec の用途ですが、セキュリティに中途半端に詳しい人で、IPsec は不要であるという意見を持つ人がいます。例えば、IPsec というのは結局 VPN(Virtual Private Network)にしか使えないとか、SSH があれば IPsec はいらない、などと言われるわけです。この意見は一面では合っていますが、一面では間違っています。SSH というのは telnet のようなもので、暗号化して telnet セッションを守るというようなものです。

IPsec で一番有効だろうと思われているのはネットワーク単位の通信の保護です。ネットワーク全体、例えば会社全体の通信を一括して暗号化したり認証したりということをやります。これがまさに Virtual Private Network なわけです。それから少し粒度を下げてホスト全体の通信保護というのがあります。

SSH というのは、少し語弊がありますが、結局人間が張りついているような通信パターンのものしか守れません。インタラクティブ指向とかといいますが、telnet とか FTP とかコピーのような感じのものしかできません。しかし、IPsec では、どういうアプリケーションかということを考えずに、すべてを保護できます。

ホスト単位の通信保護の典型的な例ですが、ノートブック PC を外に持ち出したときに、そこに鍵などを入れておくという場合がそうです。通信するときは SSH など何も気にせずに普通にアプリケーションを使うことで安全な通信路を確保するということができます。

それから TCP の保護というものがあります。SSH はもちろん TCP のみで、インタラクティブな通信の保護しかできません。これは少し語弊がありますが、大体そう思ってくださって結構です。

しかし IPsec では TCP であれば何でも可能です。トランザクション指向のもの、例えば web のようなものでもできますし、もちろんインタラクティブ指向の telnet のような通信も保護できます。それから SSH というのは TCP パケットの中身しか保護していないので、TCP のヘッダとか IP のヘッダというのは保護できません。そうするとやはり IPsec でなければ保護できないという攻撃もあるわけです。

端的な例に TCP リセット攻撃というのがあります。意識されないかもしれませんが、通信するときは TCP のコネクションを張っているわけで、それは第三者が非常に簡単に切断す

ることができます。プログラムと TCP の知識があれば簡単に 200 行ぐらいで書けるものを使って、何の詳しい情報を得ることなしに、送りつけるだけで切れてしまいます。この場合、SSH というのは TCP の上に乗っていますが、SSH は TCP の通信は保護していませんから簡単に切れてしまいます。だから嫌がらせというのは保護できません。しかし IPsec を使うと TCP リセット攻撃からは守れます。
もちろん IPsec でも守れない攻撃方法というのはたくさんあります。端末をそのままにして席を離れたりした場合に、その端末を乗っ取られたら終わりですし、それから同じネットワーク、例えば同じハブの中にいるときにできる攻撃というのがいくつかありますが、そういうものを守るのは IPsec でも難しいわけです。例えば会社の中に悪者がいたら結局守りきれないので、そういうものはやはり IPsec では守れません。
UDP の保護については、SSH では UDP の保護はできませんが、IPsec ではできます。ここでは詳しいことは説明しませんが、IP に直接乗っているようなプロトコルの保護もできます。

インターネット・セキュリティの保護機能

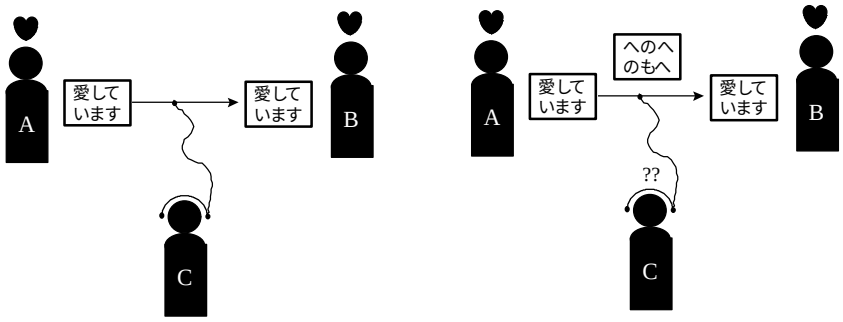
・保護できること
　機密性(confidentiality)
　完全性(integrity)
　認証(authentication)
　否認防止(non-repudiation)
・保護できないこと
　トラフィック解析の防止(traffic analysis protection)
　いやがらせの防止(denial-of-service protection)

インターネット・セキュリティという言葉はよく聞きますが、いったい何を守ろうとしているのか、何が守れるのかということになるとあやふやな場合が多いと思います。インターネットのセキュリティ技術で守れるというものには、まず機密性、次が完全性、認証、最後に忘れがちな否認防止という 4 つのものがあります。もちろん、これらは IPsec で守ることができますし、SSH や S-MIME もそうです。これは後で詳しく説明します。
インターネット・セキュリティの何らかの技術を使うとインターネットの生活は安全かということですが、まず実質的な現実世界よりははるかに安全で、この 4 つを完全に守れるということが数学的に保証できます。現実社会の印鑑を押す認証には、全然意味がありません。何か認証したような気になっていますが、これは後で書き換えることが簡単にできてしまいます。ですから、使い方を間違わなければインターネットというのは現実世界よりも安全ということになります。
しかし、インターネットのセキュリティ技術でもほぼできないというものはたくさんあり、

例えばトラフィック解析というのがそうです。これは通信の内容自体は見ませんが、通信が起こったという事実、そのパターンを見て有益な情報を得るという攻撃です。一例をあげると、アリスという女の子とボブという男の子が 1 日に 50 回ぐらい電子メールをやり取りしているというのを、内容は見えないけれどもわかったとします。そうするとその 2 人は怪しいということになり、内容は見ていませんが、有益な情報を得られるわけです。インターネットというのはオープンな世界ですから、通信の発生を相手に悟られないということはできません。ですからトラフィック解析の防止ということはできません。

嫌がらせの防止というのも、工夫はできますが 100 パーセント防止はできません。嫌がらせというのは英語で、"denial of service"といいますが、こちらの方がよく知られているかもしれません。例としては SPAM メールというものがあります。SPAM というのはアメリカに SPAM というコンビーフのようなものがあって、その CM の中で SPAM、SPAM、SPAM とうるさいぐらいに連呼するので、何かしつこいものを SPAM と呼んでいます。そのしつこい、コンタクトレンズを買いませんか、のようなメールがたくさん来るわけですが、そういうものが嫌がらせの典型例です。例えばこの対策として、SPAM メールをリレーしてしまうような電子メールの中継システムを片端から列挙して、そこからのアクセスは受けつけないというようなサービスも最近始まっています。このような工夫はできますが、100 パーセント完璧な対策というのはありませんので、やはりこれは保護できない方に入れられてしまいます。

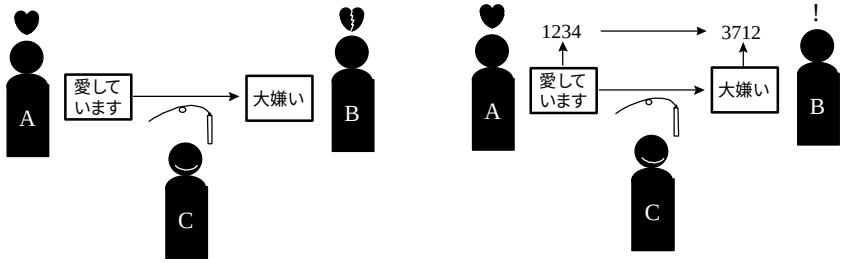
(1) **機密性**



・意図した相手のみが内容を理解できること
・第三者に内容を知られないこと
・暗号で実現する
鍵を共有することが本質

まず機密性ですが、これは簡単で、意図した相手のみが内容を理解できることを機密性といいます。説明するときに否定表現を使うことは好きではないのですが、あえて否定表現を使い、第三者に内容を知られないことを機密性といいます。
暗号屋さんは登場人物を A とか B とか C とかで始めますが、最初の人物 A はアリス、B はボブ、C になるとクリスとか、いろいろ人によって流儀は違いますが、暗号の世界の人はアリスとかボブという名前をよく使います。だから逆にアリスとかボブなどという表現をしている人には、暗号のバックグラウンドがあるのではと勘ぐることもできるわけです。
この例では、アリスがボブにラブレターを出していて、クリスが盗聴しているわけです。
これは暗号を使うと保護できるというようにいわれており、逆に暗号以外ではこのような機密性を実現できないだろうともいわれています。
暗号については、後程説明しますが、「鍵を共有する」ということが本質です。機密性は暗号でしか実現できないということと、暗号を使うときには鍵を共有することが難しいが、それが暗号の本質であるということが重要です。

(2) 完全性



・送信者が書いた内容がそのまま受信者に伝わること
・第三者による内容の改竄を検知できること
・実現方法
　電子署名
　MAC(Message Authentication Code)

次に完全性ですが、これも簡単で、送信者が書いた内容がそのまま受信者に伝わることです。否定文を使って説明すると、第三者による内容の改竄、悪意を持った改変を検知できることです。

この例ではアリスが愛していますと書いたのに、途中のクリスという女の子が嫉妬に狂って大嫌いと書き換えてしまったので、こちらでめげているわけで、こういうのは一種の嫌がらせです、嫌がらせを防止することはできませんが、発見する、検知するということが重要になってきます。

その実現方法としては電子署名と MAC(Message Authentication Code)の二つがありますが、電子署名の方ばかり強調されているので、MAC というのは知らない方も多いと思います。どちらもチェックサムを使って改竄の有無を調べるわけで、電子署名ではなくても完全性というのは実現できるということです。

(3) 認証

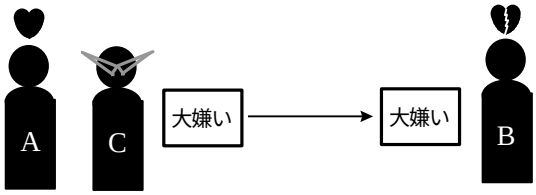
・ある名前を名乗る人が本当にその人だと確認すること

・その人だけが知っている秘密を確認する

電子署名: 公開鍵で秘密鍵を確認

・共有している秘密を確認する

MAC(Message Authentication Code)



これもよく聞く言葉ですが、ある名前を名乗る人が本当にその人だと確認することを認証といいます。インターネットのセキュリティでは、名前という文字列は単なる識別子なので、識別子自体が間違っていても問題ではありません。そう主張し続けている名前と、その名前が指している実態が一致していることが重要になるわけです。ある名前とその名前を使っている人がバインディングされている、対応関係がついていることが重要で、その名前本体の方は偽名であっても、ずっと偽名を使ってくれさえすれば問題ないわけです。

方法としては、その人だけが知っている秘密を確認することで認証を行います。

まず電子署名では公開鍵と秘密鍵そして公開鍵暗号というものが出てきますが、公開鍵から秘密鍵を検証できます。この場合、その本人だけが知っている秘密というのは、電子署名を使う場合は秘密鍵になります。また共有している秘密を確認するという方法もありますが、この二つは違います。

逆にあらかじめ共有している秘密を確認するという方法もあります。これはわかりやすいと思いますが、最初に会っておいて何かパスワードを決めておくわけです。そして、そのパスワードがちゃんと言えるならば、その人だということがわかるわけです。これを実現する方法が先ほどの MAC になります。

この二つの方法があるというところが少しミソです。IPsec では、IPsec の基本的な部分は電子署名を使いませんので、MAC のみを使うことになります。

(4) 否認防止

・通信内容をあとから否定できないこと

電子商取引には必ず必要

・電子署名で実現する

認証

本当にその人が書いた

その人のみが書ける(MAC ではダメ)

完全性

内容は完全である

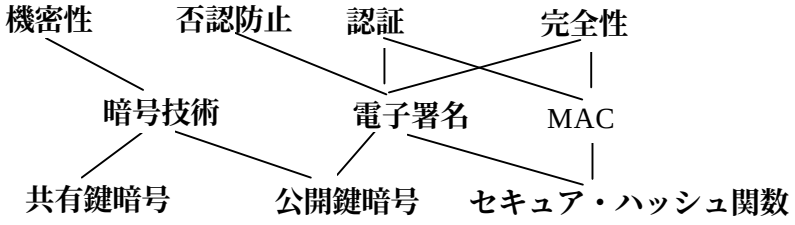
・秘密が漏洩する場合を想定しなければならないサービスもある

必ず第三者を通して通信する

最後の 4 番目が否認防止になります。これは結構知らない方もいるのではないかと思いますが、通信の内容を後から否定できないことを否認防止といいます。例えばインターネットで商売をしようとすると、発注をもらった後でそういう発注をかけた覚えはありませんと言われても困ります。ですから、内容を否定させないという仕組みが必要になります。
先ほどの完全性や認証には電子署名と MAC という二つの方法がありましたが、否認防止は MAC では実現できません。電子署名のみでできます。否認防止は、認証をして本当にその人が書いたことを確かめること、そして内容が完全であるという完全性を検証することが必要です。認証と完全性というのは電子署名でもできますから、否認防止というのは電子署名でできることになります。
しかし MAC でも両方できるということになっていましたが、MAC では通信相手と秘密を共有していることが問題になります。自分もその内容を書けるけれども相手も書けてしまうので、どちらが書いたかはわからないということになり、否認防止というのはできません。例えば発注を受けたときに、発注先も書けますが、受注元の方も同じ内容を書けてしまうので、MAC という方法ではだめなわけです。ですから IPsec というのは、電子署名を使うという拡張を用いない限り否認防止は実現できないことになります。

SSL を使って Netscape などで、クレジットカード番号を暗号化して発注を受けるということをやると思います。実は SSL ではサーバ側は電子署名を使っていますが、クライアント側は電子署名は多くの場合は使いませんので否認防止は実現できません。実現できないというとうそになりますが、普通は否認防止のないモードで使っています。
ですからクレジットカードの番号を暗号化して安心していますが、嫌がらせでたくさん発注をされると否認防止できませんから裁判をやっても勝てません。今のインターネットでは、このようなことはあまり考えずに使っていると思いますが、本当にそこまで突き詰めてやろうと思うとクライアントまで認証するようなシステムを作らないといけません。どこかで妥協しなければいけないので、今の SSL はいい妥協をしていると思いますが、このような機能が欠けていることをどれぐらいの人が気づいて使っているかというのは少し疑問です。
否認防止について付け加えますと、秘密の漏洩を想定する必要のあるサービスもありまして、例えば銀行ネットワーク間でお金の取り引きをするとして、1 回の取り引きが 1 億円などになると万が一、例えば秘密鍵のようなものが漏洩するという場合も考えなければ怖くてやっていられません。そのような場合の否認防止は、必ず第三者を通して通信することで、本当に 1 億円払い込みましたということを保存してもらって証明する必要があります。

(5) **セキュリティ保護に求められる機能と技術**



・インターネットでは保護できない項目
トラフィック解析の防止: 通信のパターンから有益な情報を得ること
いやがらせの防止

まとめますと、インターネットのセキュリティ技術で守れるものには、四つの機能があります。機密性というのは暗号技術を使用し、暗号技術には共有鍵暗号と公開鍵暗号という二つがあります。IPsec で、コアの IPsec が使うのは共有鍵暗号だけです。また否認防止というのは電子署名でしか実現できません。電子署名というのは公開鍵暗号とセキュア・ハッシュ関数というものに依存しています。それから認証というのは電子署名でも MAC でもできます。これはセキュア・ハッシュ関数というものを使います。完全性も電子署名と MAC で実現できるということです。それからインターネットでは保護できない項目というのは先ほど説明しました。これがインターネットとセキュリティの大枠です。

IPsec の構成

・**認証ヘッダ**: AH(Authentication Header)
完全性、認証、否認防止
暗号の輸出規制のため暗号ペイロードから切り離された
IP **ヘッダも保護できる**

・**暗号ペイロード**: ESP(Encapsulating Security Payload)
機密性、完全性、認証、否認防止

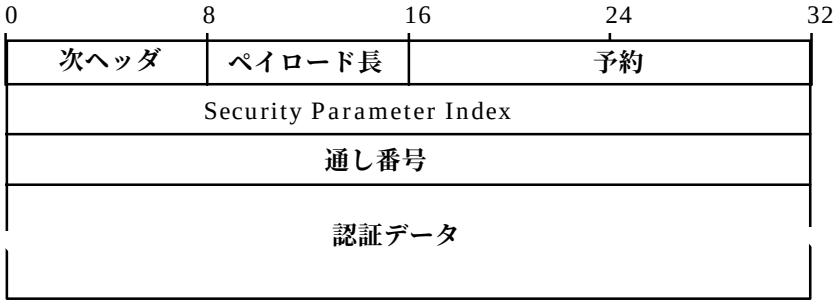
・**変換**(Transform)
認証ヘッダや暗号ペイロードは枠組のみ提供
具体的な方式は変換で定める

次はこれを IPsec **に当てはめるとどうなるかということを説明します。**IPsec **のコアの部分には、「認証ヘッダ」と「暗号ペイロード」という二つの技術があります。**
認証ヘッダでは完全性と認証が実現でき、オプショナルで否認防止も実現できますが、普段は否認防止を使わない、実現できないモードで使うと思います。暗号ペイロードは機密性、暗号化、完全性、認証も実現でき、オプショナルで否認防止ができます。
暗号ペイロードでは認証ヘッダが実現できるものは、全部実現できるので、暗号ペイロードがあれば認証ヘッダはいらないのではという疑問もあるかと思いますが、その疑問は正しくて、これは歴史的な経緯で二つに分かれました。その歴史的な経緯というのは、暗号の輸出規制や暗号の使用自体を禁止している国の存在です。例えばフランスでは市民が暗号を使うことは禁止されています。Netscape **を使うだけで、暗号というものを知らない間に使っていることになりますが、フランスでは** Netscape **社もフランス用の** Netscape **を作っていまして、暗号がまったく入っていません。ですから暗号ペイロードを使っていると、認証にしか使っていなくても、暗号化しているのではないかと政府が疑う可能性があったので、明示的に分けて暗号化には使えないというものを作ったわけです。**

話がそれますが、暗号の輸出規制という話があります。まず日本では憲法で通信の自由が保護されているので、日本国民が暗号を使うことは保証されています。これをくつがえそうとするような動きもあるようですが、今のところは自由です。しかし輸出規制というものがあり、アメリカでは昔は 40 **ビット以下でないと輸出できないという話を聞いたことがあると思います。この** 4 **月に日本はワッセナー条約というものにサインしました。ワッセナー条約というのは、ココムの後継者で、ココムというのは共産圏に危険な機材の流入を防止するというものでしたが、冷戦が終わったのでそれを少し和らげまして、北朝鮮、イラクやイランなどへの流入を防止しようとするものです。日本も調印しましたので、この** 4 **月から劇的に情勢が変わり、私の印象では日本というのは世界一輸出が規制されている、暗号技術の輸出が規制されているという国です。今やアメリカより厳しいので、暗号を使**

った製品をアメリカに輸出するということになると非常に大変です。
また、ベンチャーの方だと web や FTP サイトに置いて取っていってというようなシェアウエアを作ろうと思っている方もいると思いますが、通産省はそれもやめて下さいというので今は結構大変です。それを和らげようという試みもありますが、現在はそのような状況にあります。
認証ヘッダや暗号ペイロードが具体的にどういう形をしているかは後で説明しますが、この二つは大枠を決めているだけです。具体的な内容をどうやって決めるかは、柔軟に決定できるようになっており、それは「変換(transform)」と呼ばれています。

(1) 認証ヘッダ



・SPI(Security Parameter Index)
　方式は隠蔽されている
・通し番号によるリプレイ攻撃の防止
認証ヘッダですが、このような形をしています。これはどこにも、どういう鍵を使っているかとか、どういう認証方式を使っているかというものはありません。重要なのは SPI(Security Parameter Index)いうインデックスがあることで、これですべてを抽象化しているわけです。送信者と受信者はあらかじめ、どのような暗号、認証を使うかを合意しており、それに対する 4 バイトの数値でポインタを指しています。ですからポインタがこのヘッダに収まっているので、受け取った人は何を使うかわかりますが、途中でのぞいている人には何を使っているのかは全然わからないような仕組みになっています。このようにフレームしか与えていず、実際に何を使っているのかという具体的な値というのが transform でそれは SPI によって指されているということになっています。
IPsec の RFC が 3 年かかってリバイズされ、番号は 2410 ぐらいだったと思いますが、昔の RFC1825 から始まる IPsec と、新しく出版された IPsec には互換性がありません。ですから二つのモードを実装しなければいけませんが、具体的に増えたのは通し番号で、毎回通信するたびにカウンタを増やしていき、絶対に同じものが生成できないようになっています。あるパケットをのぞき込んで一時的にためて、そのパケットを送りつけることで攻撃するというのをリプレイアタックと呼んでおり、リプレイすることで攻撃するものですが、これを防止するための共通の枠組みが新しい IPsec では入りました。

(2) MAC

・Message Authentication Code

・認証と完全性の確認

・特殊なセキュア・ハッシュ関数を利用

メッセージとパスワードを連結しハッシュ値を計算する

パスワードが異なるとハッシュ値が異なる

H(Password+Message)→a719abh80

改竄の事実は検出できる

どこが改竄されたかは分からない

認証ヘッダでは認証と完全性が基本的に実現きるわけですが、それを実現するには MAC **というものを使います。**IPsec **で通信する場合に、どのようなハッシュ関数とパスワードを使うかを、あらかじめ合意しておきます。そして、その通信の内容とパスワードを文字列として連結して、合意したハッシュ関数を掛けてハッシュ値を出します。正確にはもう少し複雑なことをしていますが、このハッシュ値が** MAC **です。**

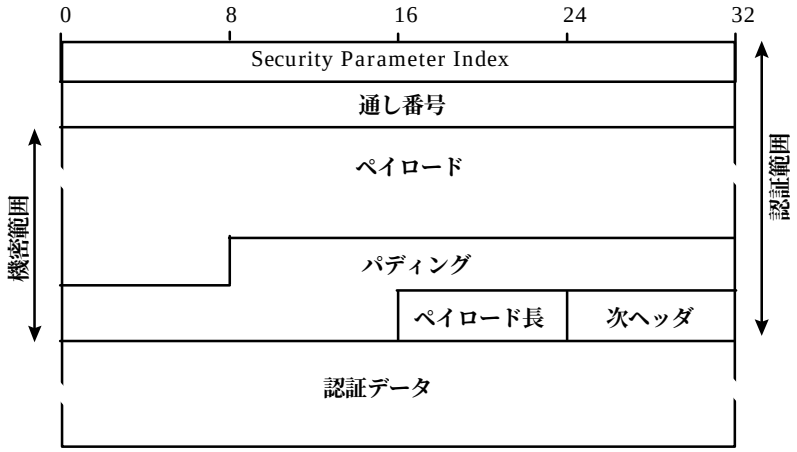
これはいったい何なのかということを説明します。まずパスワードはアリスとボブしかわかっていません。お互いがメッセージにパスワードを足してハッシュを取ったものというのは、このパスワードを知らない人には生成できません。ですからアリスがハッシュを取ってメッセージといっしょに送ったとして、それを途中で見ている人は、ハッシュ関数には二種類ぐらいしかありませんので、どちらかというのは予想できます。しかしパスワードがわからないので同じハッシュ値の生成、つまり偽造はできません。ボブはそれを確かめるときに、自分とアリスしか知らないパスワードを付けてハッシュを取って、アリスが送ってきたものと一致することを確認します。そのハッシュを生成できるのはパスワードを知っているアリスだけという認証ができるわけです。

重要なのは MAC **では改竄された事実が検知できることです。例えば、アリスがメッセージとハッシュの値を送って、その途中でクリスがメッセージを書き換えたとします。そうすると受け取ったボブが計算したハッシュ値とアリスが生成したハッシュ値というのは必ず変わりますので改竄が起こったというのがわかるわけです。**

数学的にそのような特徴を持つ、ハッシュ関数というものを使用していますので、ハッシュ値とメッセージのどちらの方を改竄されても合わなくなるため、どこを改竄されてもわかるわけです。ただし、どこを改竄されたかというのはわかりません。しかしデータ量を減らしたために、どこか改竄されたのだろうというのはわかりますが、具体的にどこが改竄されたかはわからないというのが最近定義されている MAC **の特徴です。**

付け加えますと、電子署名も方式は違いますが同様に、改竄の事実は検知できますが、どこが改竄されたかはわからないというものになっています。

(3) **暗号ペイロード**

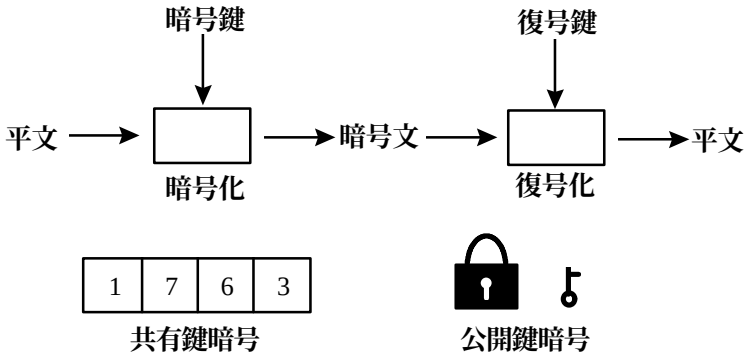


・SPI と通し番号は平文

・通し番号によるリプレイ攻撃の防止

認証ヘッダはその後に平文としてデータが付いていましたが、暗号ペイロードではデータの内容を暗号化しています。認証ヘッダと同じような Security Parameter Index **という** 4 **バイトのポインタと通し番号があり、そこから先が暗号化されたデータです。ヘッダに対する内容のことをペイロードというので暗号ペイロードという名前が付いています。オプションとして認証データを付けることもでき、認証ヘッダと同じようにチェックサムを取ってその値を付加します。認証の範囲と暗号化されている範囲が図に示されています。**

暗号



・共有鍵暗号
暗号鍵と復号鍵が同一
・公開鍵暗号
暗号鍵と復号鍵が異なる

暗号ペイロードを使うと機密性が達成できるわけですが、実際に暗号とは何かということを説明します。上に示すのは暗号の概念図です。ブラックボックスが二つありますが、まず一方は、平文と暗号鍵を入れると暗号文になって出てくるもので、これを暗号化といいます。もう一方のブラックボックスは復号化を行うもので、暗号文を入れて復号鍵を入れると元に戻って出てきます。これで二つの暗号がありますが、一つは暗号鍵と復号鍵が共通のもので「共有鍵暗号」と呼び、暗号鍵と復号鍵が異なるものを「公開鍵暗号」と呼びます。
実生活に照らし合わせた例でいいますと共有鍵暗号というのまさに自転車を止めるチェーンで、閉めるときも、開けるときも 1763 という同じ鍵を使います。それから公開鍵暗号は掛ける鍵と開ける鍵が違いますが、これは鍵と錠前の関係にそっくりなわけです。歴史的なことをいいますと、共有鍵暗号というのは 2000 年前ぐらいからあった暗号で、公開鍵暗号というのは本当にこの 20 年ぐらいの暗号です。

IPsec のモード

・トランスポート・モード

始点ホストが AH **や** ESP **を作る**

IPv6	AH	TCP

IPv6	ESP	TCP

・トンネル・モード

トンネル・ルータが AH **や** ESP **を作る**

VPN(Virtual Private Network)

IPv6	AH	IPv6	TCP

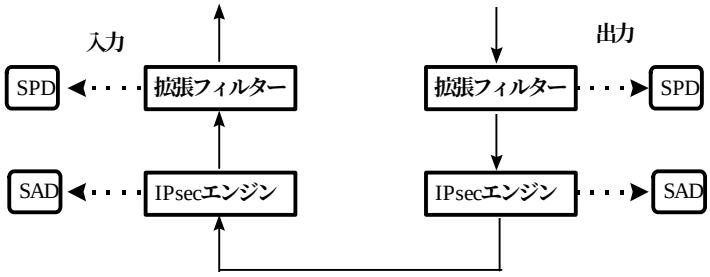
IPv6	ESP	IPv6	TCP

暗号ペイロードでは具体的には共有鍵暗号を使います。お互いに鍵を共有することにより内容を暗号化して送るというものが暗号ペイロードです。IPsec **の暗号ペイロードと認証ヘッダには二つのモードがあり、使い方によって二つのモードに分けられますが、一つがトランスポートモードで、もう一つはトンネル・モードといいます。**

トランスポートモードとは、始点ホストが認証ヘッダや暗号ペイロードを作るものをいいます。これは最初に IPsec **の用途で説明したホスト単位の暗号化というものです。**

逆に VPN **で使われている方はトンネル・モードといいまして、例えばサイトの出口に暗号化ルータを置くというようなものです。サイト内のホストというのは、いくつかパケットを投げるわけですが、それは平文です。それが暗号ルータに入りますと、そのルータが** AH **や** ESP **を作って、あたかも** IPv6/TCP **というものを内側のホストが作っているかのようにします。それに認証ヘッダをかけて新たなパケットを作り出すというのがトンネル・モードになります。こういうものを使うことで複数のホストが投げるパケットを一括して暗号化とか認証とかが取れるというのがトンネル・モードです。**

IPsec アーキテクチャ



・SPD

Security Policy Database

拡張フィルター

・SAD

Security Association Database

IPsec **エンジンが参照**

SA **は暗号や認証方式などの合意**

IPsec **のアーキテクチャはこのようになっています。ここで** IPsec **というのはフィルタのことです。パケットが出力のところにくると、その出力がネットワークの外へ出ていくのをフィルタにより禁止したり、マッチするものは** IPsec **にかけたりします。**
この IPsec **のエンジンで** SA(Security Association)**というデータベースを引いて、どのようなものを作り、どのような** AH**、認証、暗号化を施すかということを決定し、実際に** AH**、認証ヘッダ、暗号ペイロードなどを作ってネットワークに送出します。**
受け取った方はそこで SPI **というのものを決定して送出します。受け取った方はその** SPI **を頼りに** Security Association Database**、セキュリティでどういうものを作るか、鍵は何にするかというような合意のデータベースを引いて、認証の仕組みに何を使っているかを理解して、例えば暗号化されていたら復号化します。それから認証が施されたら、それを確認して平文に戻して上に上げます。例えば認証や暗号化をしていないパケットは受け付けない、というようなフィルタをかけて入力に上げるということをやります。**
IPsec **のアーキテクチャというものは、二つの要素で構成されています。一つはセキュリティのポリシー、どのようなパケットをどう扱うか、を決定する** SPD(Security PolicyDatabase)**です。もう一つは、セキュリティというものを具体的に決定する合意のデータベース、例えば暗号化は** DES**、認証は** H MAC MD5 **などという** SAD(Security Association Database)**です。**SPI(Security Parameter Index)**は** SAD **を引くために使うということがポイントです。**

AH での SPD と SAD の例

・SPD 出力
　アドレス + ポートで検索
　AH を付ける
・SAD 出力
　アドレス + ポートで検索
　AH、HMAC-MD5、認証鍵、SPI
・SAD 入力
　SPI で検索
　AH、HMAC-MD5、認証鍵
・SPD 入力
　アドレス + ポートで検索
　認証に失敗したら捨てる
　AH が付いていないと捨てる

具体的な例で説明しますが、SPD 出力と SAD 出力というのは先ほどの図の右側の部分、SAD 入力と SPD 入力は左側の部分です。SPD 出力では、パケットのアドレスとポートについてソースとディスティネーションの両方を検索します。検索した結果、そのパケットに認証ヘッダを付けなさいと書いていれば、認証ヘッダを付けるために下へいきます。SAD の方はまた先ほど引いたアドレスとポートで検索します。そうすると認証ヘッダを付けて、その認証方法が HMAC-MD5 であり、認証のための鍵の値がわかります。また、こういう 4 バイトのインデックスを付けなさいというのが出てくるわけです。
それで実際に暗号化、この場合は認証なので認証データを付けて、認証ヘッダを作って出力します。受け取る方では SPI が付いていますので、その SPI で SAD を検索します。そうすると同じもの、認証ヘッダがでてきます。認証の方式は HMAC-MD5 です。そして認証鍵、合意している鍵はこういうものですというのが出てきます。認証の検証をして検証した結果と共に上に上げます。そこで、これは認証だけですが、暗号化されているとここで平文に戻るわけです。
平文に戻るとアドレスとかポートなどは全部見えるようになりますので、今度はポリシーデータベースの方を検索します。もう SPI というのはなくなってしまいますので、逆に平文に戻って見えるアドレスやポートで検索します。そうすると認証に失敗したら捨てろなどと書いてあるわけです。
認証に失敗したものは捨て、成功していたら上に上げる。これでポリシーを実現できるということになります。
話がややこしくなってきましたが、IPsec というのはそのようにしてして使います。ポイントは、IPsec は知っているホストとしか通信しない、という大胆な仮定をしていることです。

セキュリティには二つのクラスがあり、一つは特定小数の人との通信、もう一つは不特定多数と通信するというクラスで、不特定多数のものの例は電子メールです。電子メールというのは知らない人とも通信するので、第三者認証のような難しいことをやらなければいけませんが、IPsec というのは大胆にも知っている人としか通信しないという仮定をしていますので、あらかじめ鍵を交換しておくということが可能になるわけです。
例えば Virtual Private Network を作るために暗号化ルータをサイトの外、例えば東京と九州に置きます。九州と東京で鍵を共有しなければいけませんが、1 回ならいいでしょうということで、エンジニアが出張して鍵をインストールするということをします。電子メールの場合、誰と通信するかは、わかりませんのでこのような鍵の共有はできません。IPsec だとそういう大胆な仮定を設けるので鍵交換という問題がそんなに難しい問題にならないわけです。

IPsec と NAT

・IPsec と NAT は相性が悪い
　IP ヘッダを書き換えられる: AH は困る
　サイト内ホストの IP アドレスをポートにマップする: ESP ではポートが見えない
　NAT は状態を持つ: リブートすると IP アドレスの対応関係が変わる
・NAT は IPv4 にとっての必要悪: IPv6 には NAT は不要

次に IPsec と NAT との関係を説明します。ファイアウォールを作る技術の一部に NAT というものがあります。NAT というのは内側から外には出られますが、外から内側には入って来られないという性質がありますので、ファイアウォールには好都合なわけです。また今では IPv4 のアドレスが足りませんので、内側でプライベートアドレスを使ってグローバルなインターネットと NAT を介して接続するということがよく行われています。
ではこの NAT と IPsec を両方使いたいという場合ですが、結論からいうと IPsec と NAT というのは非常に相性が悪いです。理由としてはまず、NAT が IP ヘッダを書き換えるということがあります。内側のプライベートアドレスを外のグローバルアドレスに書き換えるのですが、認証ヘッダというのは IP ヘッダまで認証していますので、書き換えられると、これは改竄ですから認証が失敗してしまいます。つまり NAT は、認証ヘッダにとっては天敵のようなものです。
また NAT には大きくわけて二つあり、一つはベーシック NAT いわれているもので、内側のアドレスを外側のアドレスにマップするという非常に簡単なものです。それから NAPT と呼ばれているものがあり、ポートまで変えます。例えばダイヤルアップ回線のように、アドレスは一個しかもらえないが内側には 4 台ぐらいホストがあるというとき、内側の 4 つのアドレスを外側の一つのアドレスにマップすることは不可能ですから、TCP や UDP のポートをいじって、内側のアドレスを外側のアドレスとポートにマップするということ

をやります。ポートが見えないと NAPT とは生きていけないわけですが、暗号ペイロードを使うと TCP や UDP のヘッダまで暗号化してしまうのでポートが見えなくなります。
例えば自宅で NAPT を使っている場合、内側のホストは ESP を使おうと思ったらけんかしてしまって使えないということになります。NAT というのはそのように変換する情報などを持っているわけです。また例えば FTP では、FTP のパケット中には IP アドレスが埋めこまれていますが、それを書き換えるのでバイトが少しずれてしまいます。例えば 133.5. というのを 10.0.と変えるとバイト数が減ります。つまり、どれだけ TCP のシーケンスがずれたかという情報も持っているわけです。そのようなたくさんの情報を持っていますが、NAT が何か事故が起こってリブートすると、IP の対応関係がリセットされて新しくなってしまいますので、今まで IPsec が仮定していたアドレスとは全然違うものが付けられるようになってしまい困るわけです。

アドレスの足りない IPv4 では、やむをえず NAT を使っていますが、NAT を使うと IPsec はほとんど役に立たなくなってしまいます。つまり IPv6 に移行すると NAT は不要で、IPv6 と IPsec というのは非常に相性が良く、NAT を使わなくてよいならばエンドツーエンドの通信が可能になります。つまり、NAT でパッチワークされたインターネットのままでよいか、一時的に混乱はあるけれども IPv6 に移行してもう少し楽しい世界を作るのかということが問題であるわけです。

鍵配送

・手動設定

SA のすべてのパラメータを設定するのは面倒

ホストごと、通信ごとに設定するのは面倒

定期的な更新が面倒

・IKE(Internet Key Exchange)

SA の自動設定プロトコル

IPsec **を使用しない**

UDP **を利用(デーモンとして実装可能)**

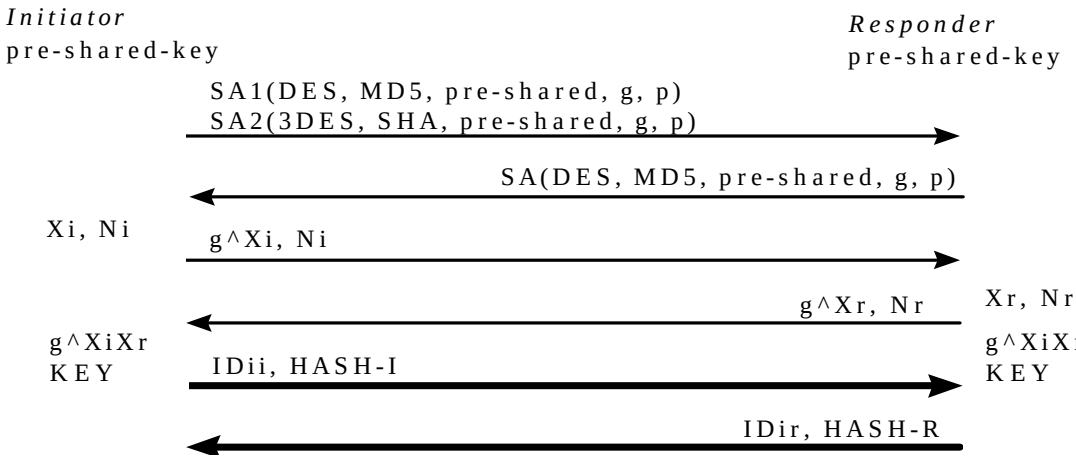
IPsec を使うときにはあらかじめ通信するホスト間で SA**、つまり合意を入れ込むということが重要になってきますが、簡単な方法としては手動設定というものがあります。**

最初は必ず、出張して手動でインストールして帰ってくるということをやらなくてはいけません。SA のパラメータはたくさんあるので、手で入力するというのは結構大変です。また、ホストごとに、あるいは通信先(ポート番号)ごとに設定するのも非常に大変です。同じ鍵を使い続けるのは非常に危険なので、定期的に鍵を変えないといけませんが、定期的な更新も全部やるとなるととても面倒です。

このような問題を解決するために、SA を動的に変える、SA を自動的に更新するプロトコルが IKE **と呼ばれるものです。上の方がルーティングテーブル、下の方がルーティングデーモンあるいはルーティングプロトコルだと思ってもらっても結構です。**

IKE **のプロトコルでは、**IPsec **を使おうとすると鶏と卵の問題になってしまうので、**IPsec **は使いません。**IKE **のプロトコル自体が暗号化の仕組みを持っていますので、その中で** IPsec **のための** SA **を交換して、カーネルの中にインストールします。**

(1) IKE Phase I



・IKE のための SA を確立

このプロトコルは UDP のベースで作られていますので、UNIX ではデーモンとして実装可能なように設計されています。IKE というプロトコルは二つのフェーズがあり、一つは IKE のための SA を確立するというものです。SA というのは大きな概念で、セキュリティのための合意というぐらいの意味しかありません。ですから具体的なことを指すときは、IPsec のための SA とか IKE のための SA というような表現をして区別します。

フェーズ 1 というのは、後のフェーズ 2 で IKE が使うセキュリティの合意を得るためのフェーズです。どのような暗号化、認証方法を使うのかということを今から IKE 同士が決定するというものです。先ほど述べたように IKE では IPsec は使いません。具体的には、そのホストの通信を IPsec で行うというようなポリシーが記述されていても、IKE のデーモンはバイパスというモードを使って穴を開けて、生でするというモードにしてから通信し始めますので IPsec の影響は受けません。

フェーズ 2 のための SA を確立しますが、これは Diffie-Hellman という公開鍵暗号を使います。Diffie-Hellman というのは、ある呪文を B さんに投げて、B さんが A さんにある呪文を投げると、途中のぞいている人はわからない魔法の数値を共有できるという、非常に不思議な公開鍵暗号です。通信をのぞき見ている人にはわからないように、通信者同士が合意を作ることが可能なわけです。

Initiator(通信を開始する方)は、このようなものならば自分が受け付けるという候補を投げます。ここでは二つの候補を投げており、一つめの候補 SA1 では暗号化は DES、MD5 というハッシュ関数を使って pre-shared 方式でというものです。

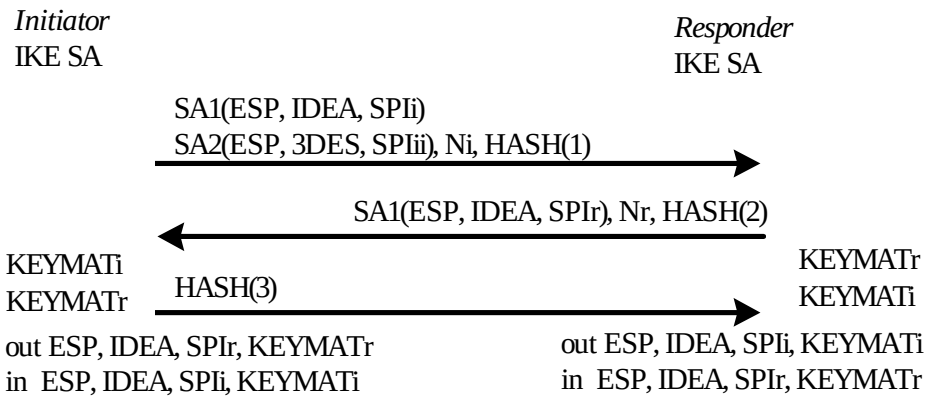
pre-shared というのは、pre-shared 鍵をコピーして送るということではなくて、方式を指しています。例えば認証局のような公平な第三者機関を使うという方式もありますが、pre-shared 方式は、簡略化して共有鍵で認証し合うというものです。

g と p というのは、Diffie-Hellman を作るための魔法の数値です。それを受け取った responder 側では、上の SA1 の方なら受け付けることができるというように返します。ここで暗号化方式やハッシュ関数の種類の合意などが交換できるわけで、これは見られてもかまいません。
Initiator 側では、二つの乱数を生成します。一つは本当に乱数で、もう一つは Diffie-Hellman のための乱数です。
Diffie-Hellman というのは先ほどもらった g という魔法の数字から g^Xi を計算します。実際は、g^Xi は g を Xi 乗して p で割った余りを取りますが、ここでは長くなるので g^Xi と書いています。
Initiator では g^Xi, Ni を投げます。responder でも同じように Xr と Nr を作り、それを投げます。
ここで、お互いにそれぞれの X を用いて g の XiXr 乗を計算します。ここをのぞいている人には、g^Xi や g^Xr はわかります。しかしこれをどう数学的に組み合わせても g^(XiXr)、g の Xi・Xr 乗というものは作れません。
このような手順で Initiator と Responder は、誰にもわからない文字列、数値列といってもよいのですが、g^XiXr を共有できます。これを共有鍵にしてここから先は、先ほど合意した DES で暗号化します。

実は Diffie-Hellman というのは、"Man-in-the-Middle"という攻撃の可能性があるので必ず認証が必要です。これは、例えば A と B の間に人が挟まって、一方では A のふりをし、もう一方では B のふりをするというものです。互いに認証情報を交換し、これで通信経路の途中に誰も人がいないということを保証して安心してフェーズ 2 に行くわけです。

(2) IKE Phase II

・IPsec のための SA を確立



フェーズ 2 は、IPsec のための SA を確立するものです。これまでに IKE のための SA という合意はできているので、フェーズ 2 が暗号化できているわけですが、ここでは IPsec が使う SA を交換し合います。

暗号化されているのでもう話は簡単で、また候補を出して出力と入力に何を使うかを決めます。responder では入力と出力は反転しますから、出力と入力はこういうものを使うというような合意が得られるわけです。鍵が共有できた後は、暗号化した安全な通信路で IPsec のための SA を交換します。

これはデーモンが勝手にカーネルに書き込みますので、人間は何らわずらわしいことをする必要がありません。具体的にいうと、まず pre-shared 鍵を共有しており、何か通信をし始めようとしてパケットが流れたとします。そうするとカーネルはポリシーデータベースを見て、そのパケットが暗号化の必要があるとわかれば、IPsec の SA がないので交換しろというシグナルを IKE デーモンに送ります。そこでフェーズ 1 になってお互いの合意を作って、またカーネルに戻ってきます。トリガとなったパケットは、IKE のフェーズ 1 と 2 が終わるまで待たされ、SA が確立した後にやっと暗号化されて出ていくということになります。

もう一つのきっかけとしては、ある程度の時間が過ぎたら IKE が勝手に SA を交換して IPsec のための SA を更新するという場合があります。

IPsec と IKE で利用可能な暗号

・DOI(Domain of Interpretation)

・ハッシュ関数

MD5, SHA-1(MUST)

DES

・共有鍵暗号

DES, NULL(MUST)

3DES, CAST, BLOWFISH

IDEA, 3IDEA, RC4, RC5

・公開鍵暗号

Diffie-Hellman

IPsec と IKE で利用可能な暗号というのが標準化されています。IANA(Internet Assigned Number Authority)という機関があり、インターネットで数字を使おうとするときに同じ数字を違う意味で使ってしまわないように、IANA にその数字が何を意味しているかを登録して、あらゆる数値に対して意味が一意になるようにしています。その IPsec 版を DOI といいます。IANA へいちいち登録すると手間がかかるので、IPsec は自分たちのワーキンググループで権利をもらっています。この DOI という RFC を発行して、3 というのは DES であり、2 というのは IDEA という意味である、というのを決めているわけです。

その RFC には、ハッシュでは MD5 と SHA-1 というハッシュ関数がありますが、それを必ず実装しなさい(MUST)というように書いてあります。DES のような暗号も実はハッシュ関数として使えますので、ハッシュで使ってもよいと書いてありますが、これは MUST ではありません。

共有鍵暗号では、DES と NULL(暗号化を何もしない)というものが MUST になっています。NULL というのは例えば相互接続性の実験をするときに使います。後は 3DES(トリプル DES)、CAST、BLOWFISH などたくさんありますが、これらも使ってよいというように書いてあります。これは IKE で SA を交換するときに、自分が実装しているものの候補を投げますので、MUST になっている二つ以外に、MUST 以外のものも候補として出して、向こうが受理してくれたら使うというようにします。

また、公開鍵暗号は Diffie-Hellman を使いなさいということが書いてあります。

IPsec と IKE の実装状況

・米国では少なくとも 60 社が IPsec を実装

・ルータでの実装が多い

VPN(トンネル・モード)

・国内では数社

住友電工、東芝、常陸
YAMAHA、KAME、ソリトン、etc.

・IKE の相互接続性テストはこれから

IPsec と IKE の実装状況ですが、この前、ニューヨークで相互接続性の試験があり、そのときに参加した企業というのは 60 社ありました。4 日間では全然時間が足りないので、60 社すべてでフルメッシュで試験したということはもちろんありませんが、お互いにその一部同士では相互接続性を検証しています。
もう既に 60 社ぐらいが製品を出しており、実際に今でも使えます。やはりルータでの実装が多く、VPN を作るためにトンネル・モードだけをサポートして組み込んでいます。国内でも住友電工、東芝、日立、YAMAHA、KAME、ソリトンなど、一部はもう製品にしていますし、KAME はフリーなので、取ってきて自分で試すということができます。
それから肝心な IKE がないと IPsec の運用は非常に大変ですが、IKE の実装がいくつかあります。IKE については、相互接続性の実験をすると何かうまくいかないということが多く、これから相互接続性を上げていくという時期で、あと半年とか 1 年すれば落ち着くと思います。
ですからコアの部分はもう簡単に手に入りますし、鍵交換の方は 1 年ぐらいすれば安定するのではないかという状況です。

3. IPv6

次世代IP の必要性

・経路表増加の抑制
　1 組織 1 クラス B アドレスの割り当て
・クラス B アドレスの枯渇
　複数のクラス C アドレスの割り当て
・経路表の急増
　CIDR による経路表増加の抑制
・それでもインターネットは成長する
　IPv4 アドレス全体の枯渇
・アドレス空間の大きな IP が必要
　次世代 IP or IPng(IP next generation)

IPv4 のアドレスがもう足りないということはご存じだと思います。今ではアドレスをもらうときに JPNIC と交渉して、結局あまりもらえないので NAT を使うというのがお決まりのパターンです。NAT には多くの問題があります。一番の問題はエンドツーエンドで通信しなければいけないというインターネットの基本的なアーキテクチャを破壊したということです。今のインターネットというのはパッチワークと呼ばれていて、つぎはぎだらけのネットワークになってしまいましたが、本質的にそういうアドレスの問題を解決するにはアドレス空間の大きな IP が必要です。それが IPv6 です。

IPv6 は NAT のいらない世界

・インターネットの基本は双方向性
　NAT は IPv4 には必要悪
　一方向性はパケット・フィルタリングで実現できる
・双方向性の必要な環境
　ホーム・ネットワーク
　複数のコネクションを必要とする通信
・グローバル空間におく必要がある端末
　自動車
　携帯電話
・IPv6 + IPsec + パケット・フィルタリング

まずインターネットの基本は双方向性なので、NAT のような一方向のものではたくさんの通信が制約を受けるわけです。例えばストリーム系の通信というのはいったん外向きにコネクションを張りますが、向こうからもコネクションを張り直してきて通信するというも

のです。そういうものが多くありますが、NAT ではそれを利用できません。一方向性というのは、必要であればパケットフィルタリングで簡単に実現できてしまいますから、IPv6 に移行したら NAT は使わずにフィルタリングを使用します。IPsec というのは、フィルタリングそのものなので IPv6 と相性が良いのです。

また双方向性の必要な例としては、まず家庭のネットワークです。最近、ビデオ系の通信プロトコルが標準化されて、家庭の機器がネットワークにつながりそうな時代になっていますので、そういうものを外から制御するというときに双方性というのは必要です。それから本当に大きなグローバルな空間に置く必要のあるという端末がたくさんあり、端的な例は自動車や携帯電話です。携帯電話のメーカーでも、IPv6 の実装に非常に真剣に取り組んでいます。携帯電話の第 2 層のデータリンクというのは、非常に複雑になっており、その上のモバイル IP というのが頑張ってくれると下がすっきりして楽だということになりますが、そういうのも目指して頑張っている人たちがたくさんいます。携帯電話はプライベートなネットワークに置くと通信は全然できませんので、こういうインターネットの可能性を考えても非常に大きな空間が必要になる思います。

覚えておいて頂きたいのは、「**IPv6 + IPsec + パケット・フィルタリング**」ということが、インターネットのアーキテクチャとしてはやはり正しいということです。

IPv6 の特徴

・アドレスの拡張

2^32 = 43 億 → 2^128 = 3.4 × 10^38

・ヘッダの簡略化

ヘッダ長、TOS、断片オフセットなどの排除

・数珠つなぎヘッダ

利用頻度の低い機能を追い出す(断片ヘッダなど)

汎用的なオプションの定義

・プラグ&プレイ

デフォルト経路、プレフィックスなどの取得

・IPv4 と変わらない機能

セキュリティ: ただし IPv6 では IPsec が必須

マルチキャスト、モバイル

IPv4 のアドレス空間は 2 の 32 乗ですから大体 43 億です。これは世界人口よりも小さいので、足りないということが簡単にわかるわけですが、IPv6 では 128 ビットのアドレスですので、使いきれないぐらい大きな空間なわけです。

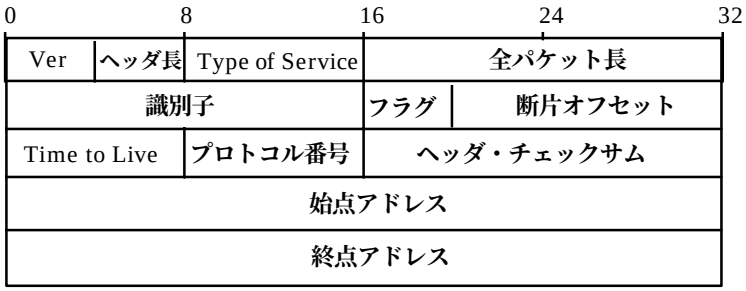
重要なのは、IPv6 が IPv4 に対して有利なのはアドレスが大きいということだけという点です。それ以外を変えないというのがバージョン 6 の設計精神なので、アドレスが拡張されたということだけが有利で、それにより派生するよいことが、たくさんあります。

いちばん大きな特徴は「プラグ&プレイ」です。これはネットワークに差したら、その場でつながってしまうというような機能ですが、これは DHCP があれば実現できるではないかと思われるかもしれません。しかし実際には、IPv4 のアドレスが少ないので、プールしているアドレスも数も少なく、なかなかアドレスがもらえないということが起きます。またアドレスをもらっても、途中のルータがキャッシュしている以前に割り当てていた別のホストでのイーサネットアドレスとの対応関係が消えるまで 5 分間使いものにならないというようなことも起きます。

IPv6 の神話というのがあり、マルチキャストもモバイルもセキュリティも解決できるというような話がありますが、それは嘘です。確かに IPv6 では、IPsec を使いますのでセキュリティがよくなりますが、IPv4 でも IPsec は使えますので比較して有利だとはいえません。またマルチキャストやモバイルというのは IPv4 でも IPv6 でも、しょせん難しいものです。

アドレスが拡張されることで、プラグ&プレイというのは非常に簡単になります。これを使ってリナンバリングという優れた機能も実現でき、空間が広がるのでインターネットの可能性として自動車、携帯電話、ホームネットワークなどをつなぐことができるという可能性があるわけです。

(1) IPv4 ヘッダ



・除去

ヘッダ長、すべてのオプション(ヘッダ長の固定化)

TOS ヘッダチェックサム

識別子、フラグ、断片オフセット(できるだけ断片化しない)

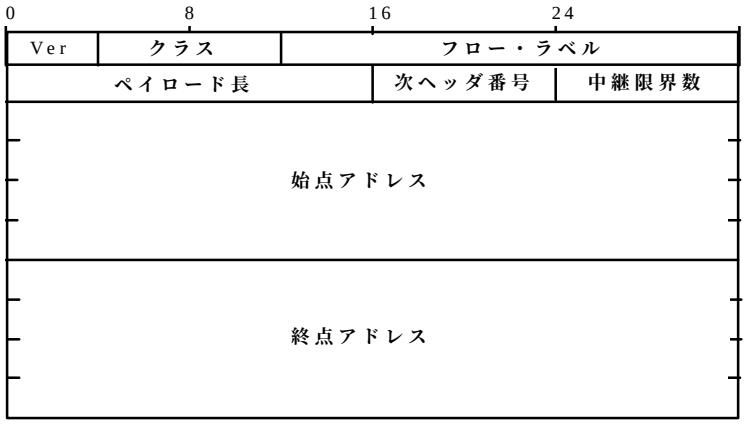
・名称変更

プロトコル番号→次ヘッダ番号

TTL→中継限界数(Hop Limit)

IPv4 のヘッダとから、経験的に使わないというフィールドを全部除去してすっきりさせたのが IPv6 です。

(2) IPv6 ヘッダ



・アドレス長は 4 倍、ヘッダ長は 2 倍

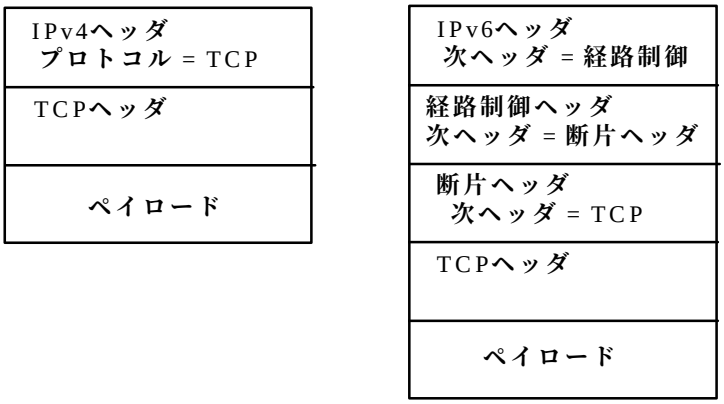
・オプションは拡張ヘッダで実現

・中継点限界数は最大 255

やはりすごいと思うのは、アドレス長は 4 倍になり、アドレスの数は何か無限大ぐらいに大きくなっているわけですが、ヘッダの大きさは IPv4 は 20 で IPv6 が 40 ですから、2 倍

にしかなっていないということです。いらないオプションというのは全部、拡張ヘッダに押し込んで外で実現するということになっています。
実はパケットにはブレーキの仕組みがあります。例えばぐるぐるとネットワークを回ってしまうというパケットが発生することがありますが、ルータを経由すると必ず 1 減らすという安全弁が付いており、それがゼロになったら迷子になったパケットは消去します。これは IPv4 でも IPv6 でも 255 です。つまりルータを 255 経由したネットワークとはつながりません。これはかなり議論になりましたが、IPv4 では 255 ホップなので、IPv6 では絶対にネットワークはもっと大きくなるのだから、限界数を大きくしようという意見もたくさんありました。しかし、限界数がそんなに大きくなったら、そもそもインターネットはだめだろうという思想があり、もっとネットワークの構成を簡単にして、できるだけ短いホップでつながるように意識させるためにも、これは変えないということになりました。

(3) **拡張ヘッダの数珠つなぎ**



・TCP や UDP を示すプロトコル番号を次ヘッダに抽象化
・汎用的なオプション
・TCP、UDP、認証ヘッダ、暗号ペイロードの IPv4 との共有

拡張ヘッダというのはあまり使わないと思いますが、念のため説明しておきます。IPv4 ではバージョンヘッダがあり、TCP ヘッダがあって、ペイロードというのがありました。また、内側にどういうトランスポートプロトコルを格納しているかを示すプロトコルフィールドというのがありました。このフィールドに、次のトランスポートプロトコルは TCP であるというように書いていましたが、このプロトコルというのを「次ヘッダ」というように拡張しました。次は絶対にトランスポートプロトコルであると決め打ちだったのを、次ヘッダというように考えて、ここにあらゆるヘッダを入れるようにしています。
拡張することにより、次ヘッダとして経路制御ヘッダやフラグメントのための断片ヘッダなどのいろいろなヘッダをつぎ込めるようにして、オプションを実現できるようにしたものが、「拡張ヘッダの数珠つなぎ」です。このアイディアを元に、次ヘッダは認証ヘッダであるとか、次ヘッダはセキュリティペイロードというふうにして、応用したのが IPsec なわけです。IPsec というのはもともと IPv6 のために作られていましたが、その技術は IPv4 にも使えるので、今では IPv4 のために IPsec を作っているという人が多いというような歴史があります。

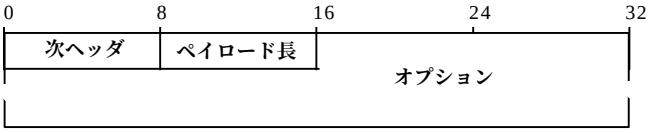
(4) **拡張ヘッダとヘッダ番号**

0	**中継点オプション・ヘッダ**(Hop-by-Hop Options Header)
1	ICPM
4	IPv4 **ヘッダ**
6	TCP **ヘッダ**
13	UDP **ヘッダ**
41	IPv6 **ヘッダ**
43	**経路制御ヘッダ**(Routing Header)
44	**断片ヘッダ**(Fragment Header)
50	**暗号ペイロード**<IPsec>
51	**認証ヘッダ**<IPsec>
58	ICMPv6
60	**終点オプション・ヘッダ**(Destination Options Header)

ICMP: Internet Control Message Protocol

たくさんヘッダが定義されていますが、UDP**、**TCP **というのは** 6 **番、**13 **番です。暗号ペイロードと認証ヘッダは** 50 **と** 51 **になっていますが、つまり** IPv4 **に次のヘッダが** 50 **であると書いていれば、その中身は暗号ペイロードであるということが、わかるようになっているわけです。**

(5) **オプション・ヘッダ**



・「オプション番号、長さ、値」形式
・8 ビットのオプション番号
　ICMP 動作ビット
　change en-route ビット(for IPsec)
・中継点オプション・ヘッダ
　巨大ペイロード・オプション
・終点オプション・ヘッダ

途中のルータが必ず解釈する必要のあるヘッダに中継点オプション・ヘッダというものがあります。また、終点だけで解釈する必要のあるオプション・ヘッダもあり、このオプションというのは柔軟に定義できるようになっています。ヘッダの数を増やすのではなく、軽いオプションを増やすことでいろいろな機能を持たせようとしています。
これらのオプションは共通の形をしており、終点オプション・ヘッダも中継点オプション・ヘッダでもこういう形をしています。オプションはオプション番号、長さ、値の形式をしており、いろいろな情報があります。
例えば中継点オプション・ヘッダでは、オプションの内容を書き換えてしまうようなものがありますが、それだと認証ヘッダの場合にはチェックサムの値が変わると困ってしまいます。そのために一つビットがあり、認証ヘッダを計算するときは、そこを無視して全部ゼロで埋めて計算するという指示をします。
ここで重要なのは、単にオプションの形式が決まっていて、そのオプションには中継ルータで処理されるものと終点で処理されるものがあり、そのオプションが好きに増やせるという点です。

中継点オプションの実例としては、巨大ペイロード・オプションというのがあります。これはジャンボクラムという名前でも知られています。
スーパーコンピュータは HIPI などというネットワークの上でパケットを交換しますが、これは IPv4 がペイロードで指定できる 64 キロバイトよりも大きなデータを相手に送りつけるということを好む計算機です。通信にはコストをかけたくなく割り込みを非常に嫌うということで、割り込みはなるべく少なく通信量はなるべく大きく、というのがスーパーコンピュータの世界です。そういうときに中継点オプションを使って、IPv6 でのペイロー

ド長は 2 バイトですが、その値を無視してオプションの中身を見ることを指示します。オプションの方では 4 バイトでペイロードの大きさを指定しています。
これはルータが解釈するのではなく、その一つのネットワークで閉じた話なので、全然中継点という感じがしませんが、中継点オプションの応用例としてそういう巨大な 64 キロを超えるパケットを送りつけるということも可能になっているということです。

アドレス

(1) **アドレスの表記**

・16 進数 4 桁ごとに":"で区切る

3ffe:0501:0008:0000:0060:97ff:fe40:efab

ff02:0000:0000:0000:0000:0000:0000:0001

・それぞれの頭の 0 は省略可

3ffe:501:8:0:60:97ff:fe40:efab

ff02:0:0:0:0:0:1

・連続する 0 は "::"で表現可

3ffe:501:8::60:97ff:fe40:efab

ff02::1

・プレフィックス長は"/"の後に 0〜128

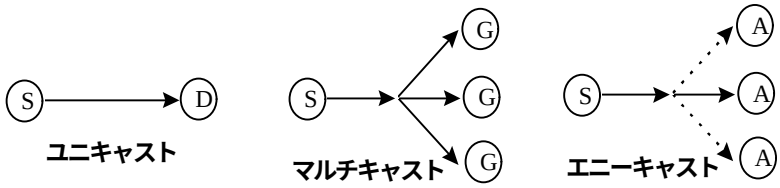
3ffe:100::/16

IPv4 では 4 バイトでしたから、バイトごとにドットで区切って、1 バイトを 10 進数で表わしていました。IPv6 では 16 バイトで、10 進数に直すのは大変なので 16 進数のまま表記しています。1 バイトごとに区切るのも大変なので、2 バイトごとにコロンで区切って書きます。
表記を簡単にする方法があり、それぞれのブロックの中のゼロは省略可能で、連続するゼロは::で表現可能ということになっています。また、連続するゼロが 2 個所にある場合に、両方::と書くとお互いの長さがわからなくなるため、::は一回しか使えないということになっています。
IPv4 ではマスクといっていたものは、IPv6 ではプレフィックスと呼びます。なぜマスクと呼ばないかといいますと、昔のインターネットでは、マスクは連続していなくてもよく、1 バイト目と 3 バイト目がネットワークであるいうような、かなりチャレンジングなこともできました。IPv4 でも CIDR(Class-less Inter-Domain Routing)がでて、必ずネットマスクは連続している必要があるということになりました。IPv6 では、上のプレフィックス側の方しかネットワーク部分でしかありえないので、マスクという表現ではなく、プレフィックス、前の方という表現を使います。
プレフィックス、つまりネッワーク部分を表わす場合はスラッシュを書いて、その後にゼロから 128 の数字を書きます。ですからデフォルトは::/0 になります。

実はたくさんの問題が起きており、致命的なのは区切りにコロンを使ってしまったことです。やはりもめているのは、URL はどうやって書くのだということです。"http://"の後にアドレスを埋め込むことができますが、Netscape などのパーサでは、アドレスを後ろからみていき最初の":"を取るというような思想になっています。このあたりは何となく HTTP と同じスキーマになってしまうとのかなみたいな話になっており、結局これをどうするか合意が取れていません。やはり IPv6 ではアドレスは直接埋め込んではいけないのかな、というのが今の状況です。また UNIX の世界では X-Window System を使いますが、そのときもウィンドウを指定するのにコロンを使います。それをどうするのかと質問すると、スティーブ・ディアリングは考え込んでしまいます。

(2) アドレスの種類

・ユニキャスト
　特定の 1 ホストと通信
・マルチキャスト
　ホストのグループと通信
・ブロードキャスト
　あるリンクに属す全ホストと通信
・エニーキャスト
　複数のホストが受け取れるアドレスに送信、受け取るのは 1 ホスト



アドレスの種類は概念的には四つあります。一つがユニキャストで、一対一の通信のためのアドレスです。それからグループ通信のためにマルチキャストというのがあります。マルチキャストというのはソース、つまり始点の人が一個パケットを投げると、途中でコピーなどが起こり、一つのグループに属しているすべてのメンバーにパケットが届くというような通信形態です。ブロードキャストというものがありますが、これは IPv6 ではマルチキャストの一部だと考えていますので、これはもうなくなりました。わかりにくいのはエニーキャストです。マルチキストは複数の人が受け取れて、届くのも複数の人ですが、エニーキャストでは複数の人が受け取れるけれども実際にパケットが届くのは一人です。これはサーバ探索のようなことに使います。使い方というのは誰もあまり考えていず概念としてあるのですが、KAME プロジェクトでは DNS サーバをエニーキャストで探すというようなことをしています。IPv6 のプラグ&プレイでは DNS サーバのアドレスは取れないので、何とかして DNS サーバを探すということをやらなければいけません。well-known な、エニーキャストアドレスを定義しておいて、サイト内にある DNS サーバのエニーキャストアドレスに投げます、そうするとどれかから DNS の答えが返ってくる、というようなことに使えると思っています。

(3) アドレスの大まかな分類

3 ビットのプレフィックス(2 進数)

000　特殊なアドレス

001　経路集約型アドレス

010　未割り当て(was プロバイダ型アドレス)

011　未割り当て(was 地域型アドレス)

100　未割り当て

101　未割り当て

110　未割り当て

111　リンクローカル、サイトローカル、マルチキャスト

IPv6 のアドレス空間というのは上の 3 ビットで 8 つに分割されています。000 で始まるのが特殊なアドレスで、001 で始まるのが経路集約型アドレス、いわゆる CIDR ベースのアドレスです。また、111 で始まるところにリンクローカルアドレス、サイトローカルアドレス、それからマルチキャストというものが格納されています。

(4) 特殊なアドレス(ユニキャスト)

・ループバック・アドレス

0000:0000:0000:0000:0000:0000:0000:0001 or ::1

・未指定アドレス

0000:0000:0000:0000:0000:0000:0000:0000 or ::

重複アドレス検知に利用

・IPv4 互換アドレス

0000:0000:0000:0000:0000:0000:xxxx:xxxx

(例) ::163.221.202.11

自動トンネルに利用

・IPv4 射影アドレス

0000:0000:0000:0000:0000:ffff:xxxx:xxxx

(例) ::ffff:163.221.202.11

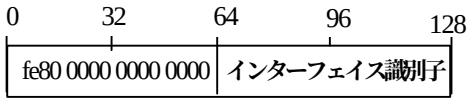
カーネルの実装に利用

第 1 ブロック目に特殊なアドレスが定義されていますが、一つはループバックアドレスです。IPv4 では 127.0.0.1 だと思っているかもしれませんが、実は IPv4 ではどれがループバックアドレスという決まりはありません。サイト内で勝手に定義することになっており、ループバックアドレスが固定でないと、UNIX のカーネルでは決め打ちできないので非常に大変ですが、IPv6 では::1、全部ゼロで 1 というのをループバックアドレスと決めていますので、カーネルがきれいに書けます。

全部ゼロというのは未指定アドレスで、これは重複アドレス検知、自分が作ったアドレスが他の人と重複していないかということを調べるときに使います。
IPv4 互換アドレスというのは、上が全部ゼロで IPv4 のアドレスを下の 4 バイトに埋め込むというものです。これは IPv4 の大海の中にぽつんと浮かんでいる IPv6 のホストがあったときに互換アドレスを使うと、自動的にトンネルを張って IPv6 の島にたどり着くというものですが、あまり使いませんので、こういうものもあるということだけです。
また、射影アドレスというのは上が全部ゼロでその次に ffff があり、その後にアドレスを埋め込みます。これはカーネルの中で、IPv4 のアドレスをこのように表記して、IPv6 のアドレスとして見てしまうというものです。TCP とか UDP に見せるのは IPv6 だけにしてしまい、IPv4 からパケットが上がってきても上には IPv6 に変換してしまって見せるというときのために使います。そうすると UDP と TCP では IPv6 のアドレスだけを考慮すればよくなり、カーネルがきれいに書けるわけです。
なぜこれが ffff なのかといいますと、TCP ヘッダや UDP ヘッダチェックサムがありますが、チェックサムは足していって何かで割るというような簡単なものなので、ゼロを足しても変わりません。TCP とか UDP で定義されているチェックサムでは ffff というのもチェックサムの値を変えません。ですからここに ffff を埋め込んでおけば TCP と UDP では何らコードを変える必要がない、というような工夫が見られるわけです。

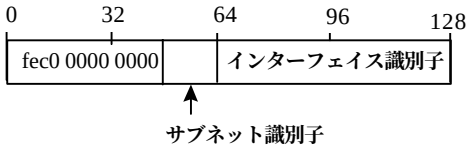
(5) ローカルアドレス

・リンクローカル



(例) fe80::260:97ff:fe40:efab

・サイトローカル



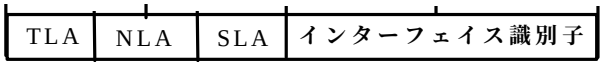
(例) fec0::1234;260:97ff:fe40:efab

アドレスブロックの下の方のローカルアドレスですが、IPv6 ではまずリンクローカルというアドレスがあり、そのリンクのみでユニーク、一意になっています。これを使ってプラグ&プレイをしますが、それは fe80 から始まって、ずっとゼロで、下 8 バイトに何かインターフェースの識別子、インデックスを埋め込みなさいというものです。
これはリンクだけで有効なので、リンク内で一意であればよいというものです。つまり、

あるホストがたくさん足を持っているとして、それぞれのリンクでユニークであれば、ホスト内でさえもユニークでなくてかまいません。ホストの中ではユニークではないけれども、あるリンクの中ではユニークであるというアドレスです。
こういうものは何のために必要かといいますと、IPv6 のエンジニアはプラグ&プレイで、デンティスト、歯医者さんのオフィスとネットワークという話をよく引き合いに出します。歯医者さんというのは非常に高学歴ですが、やはりネットワークのことは全然知らないので、IPv6 が入った計算機を買ってくるとハブにつなげれば通信できるものだと思っています。そういう人たちがハブを 1 個買ってきて、自分たちのカルテを処理するために三つのコンピュータを買ってつなげます。もしグローバルアドレスというものしかなかったら、インターネットに接続のないハブですのでグローバルアドレスをもらいようがありませんから、その三つのコンピュータは通信できません。それでは困るので、ハブに差したら使えるネットワークアドレスが必要になります。それがリンクローカルアドレスで、リンクでユニークであればよいというものです。

次がサイトローカルアドレスですが、下 8 バイトがインターフェースのインデックスということは決まっています。これはグローバルアドレスになっても共通で、その上にサブネットの識別子を 2 バイト置いています。サブネットが 2 バイトということは 65,000 個のサブネットを持てるわけで、これは IPv6 で必ず保証されています。2 バイトということで、クラス B ではないかと思うかもしれませんが、サブネットの数が 2 バイトなので、実はこれはクラス A 相当です。1 サブネットは 8 バイトもありますので、これだけで IPv4 の空間より大きいぐらいです。リンクローカルアドレスと区別するために fec0 で始まります。サイトローカルアドレスはまだ使われておらず、全然経験がないので、今後はこれは使われなくなるかもしれません。僕はこのサイトローカルアドレスというのに反対している方なので、こんなものはなくしましょうと私自身は思っています。

(6) **グローバル・アドレス**



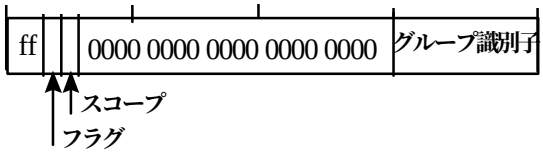
・経路集約型アドレス
位置情報と識別子の分離
パブリックとサイトの分離
・TLA(Top Level Aggregator)
8192 **個**
・NLA(Next Level Aggregator)
NLA1, NLA2,...
・SLA(Site Level Aggregator)
サイトローカルとサブネット番号を共有
3ff3:501:8:1234:260:97ff:fe40:efab

グローバルアドレスは今では経路集約型アドレスと呼ばれており、位置情報と識別子を分離しています。つまり上 8 バイトがホストの位置で、ここを見てルーティングします。識別子としては下 8 バイトを使うというようになっています。
それからルーティングで使う上 8 バイトも 6 バイトと 2 バイトに分かれており、上の 6 バイトがパブリックです。つまり JPNIC からは上 6 バイトをもらうことになります。下の 2 バイトはサイトで自由に使えます。つまり 65,000 のサブネットを使えることが保証されているのが経路集約型アドレスです。
パブリックの方も 2 バイトと 4 バイトに分かれていまして、上の 2 バイトを TLA といいます。これは 2 バイトではなくて、上いくつかが取られていますので約 8,000 個あるということです。

つまりこれは大きなプロバイダのようなもので、IPv6 の世界では大きなプロバイダを 8200 個ぐらい提供できるというモデルにしているわけです。これ以上プロバイダが増えたらインターネットは通信できないので、トポロジーを簡単にするためにこのような足かせを設けるということにしています。上のプロバイダがもらったアドレスは、その下の NLA というものにどんどん渡していくということになります。
重要なのは、IPv4 では CIDR が始まる前にアドレスを割り当ててしまったので集約できませんが、IPv6 では仕切り直しなので、必ず集約するようにしかアドレスを与えませんということです。
バックボーンルータの TLA を持つ、巨大なインターネットエクスチェンジみたいなものがあって、そこに足を突っ込んだら自分のネットワークのネットワーク側はまたたくさん情

報を持たなければいけませんが、そこが外と通信する場合は経路情報は 8,200 個ぐらいに抑えられるということを保証しているわけです。
きちんと集約できるようにアドレスを下の子供、その子供というように渡していくと、バックボーンルータに負荷をかけない形、つまりルーティング情報を集約できる形でテーブルを小さくできるというような工夫がされているのがこの経路集約型のアドレスです。

(7) マルチキャスト



・4 ビットのスコープ
1　ノードローカル・スコープ
2　リンクローカル・スコープ
5　サイトローカル・スコープ
8　組織ローカル・スコープ
e　グローバル・スコープ

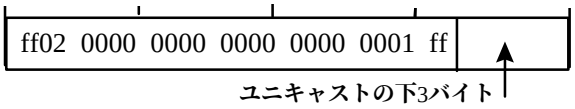
・32 ビットのグループ識別子
ff01::1 (ノードローカル全ノード)
ff02::1 (リンクローカル全ノード)
ff02::2 (リンクローカル全ルータ)

マルチキャストですが、上が ff で始まっているとマルチキャストで、フラグとスコープというのがあります。フラグは今のところ未定義です。
スコープですが、1 はノードローカル、つまりホスト内です。2 はリンクローカル、8 は組織ローカルと言っていますが、これは多分何となく作っておきましょうという感じで作ったと思うので、ほとんど 8 番は無意味です。それから e のグローバルというのがあります。
下 4 バイトがグループの識別子で、グループ番号 1 番はすべてのノード、ホストという意味になります。また 2 番は全ルータという意味です。
ff01::1 というとノードローカルの全ノード、なぜノードの中に全ノードなのだという感じもしますが、ノードローカルの全ノードが指定できます。つまりこのアドレスを使うとそのノードがどういうユニキャストアドレスを持っているかわからなくても ping の ff01::1 というと、そのホストはそのホスト内で反応してくれます。
それからよく使うのは ff02::1 と::2 です。ff02::1 というのに ping を送ると、そのハブにい

る全ホストが反応してくれます。それから::2 に ping を送ると、そのリンクにいる全ルータが反応してくれます。

もし KAME をインストールしているなら、ターミナルルームに行って ping6 の ff02::1 などとするとたくさんの IPv6 のホスト、少なくとも 10 個ぐらいが反応してくれると思います。ちなみに IETF でやると 10 台ぐらいありました。

(8) 要請マルチキャスト・アドレス



・アドレス解決(いわゆる ARP)に使う
　ブロードキャストはない
　リンクローカル・全ノード・アドレスは大きすぎる
　もう少し小さいマルチキャスト・アドレスが必要
　　(例) fe80::2056:01ff:fe12:3456 → ff02::1:ff12:3456
　通常のマルチキャストでは ff00:0000〜ffff:ffff を使わない

マルチキャストにはもう一個、要請マルチキャストアドレスというのがあります。ARP を解決するときに全ノードに投げているのでは迷惑なので、そのホストが確実に反応するような、もう少し小さいアドレスを用意しています。昔のブロードキャストよりももっと小さい安全なアドレスだと思って下さい。これはプラグ&プレイをやるときに役に立ちます。

プラグ&プレイ

プラグ&プレイがどのように実現されるかということを説明します。

(1) アドレスの自動生成

・イーサネット(IEEE802 アドレス)
　00:60:97:40:ef:ab
・インターフェイス識別子(EUI64 アドレス)の生成
　260:97ff:fe40:efab
・リンクローカルを仮に割り当てる
　fe80::260:97ff:fe40:efab
・マルチキャスト・アドレスへの参加
　リンクローカル・全ノード・マルチキャスト・アドレス
　　ff02::1
　要請マルチキャスト・アドレス

fe80:260:97ff:fe40:efab→ff02::1:ff40:efab

イーサネットの MAC アドレスというのは 6 バイトです。例えば PCMCIA を差した瞬間にこのアドレスが取れます。次にインターフェースの識別子を作ります。インターフェース識別子は 8 バイトでしたので、この 6 バイトから 8 バイトのものを作るという方式は決まっています。3 つに分けて間に fffe を埋め込んで上の 1 ビットを反転させるという方法で、8 バイトの識別子を作ります。
fe80 を前に付けると、これでリンクローカルアドレスです。このアドレスを使うと、そのリンク内で通信ができ、これでもう歯医者さんの問題は解決するわけです。イーサネットのアドレスは重ならないことが世界的に保証されていますが、万が一重なるかもしれないので、これは今のところ仮アドレスになります。これはホストなので、リンクローカルの全ノードマルチキャストアドレスにも受け取りなさいということを書き込みます。それから ARP に使う、要請マルチキャストアドレスにも受け取りなさいということを書き込みます。

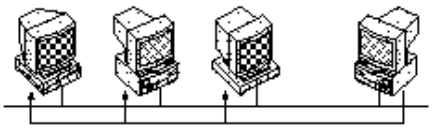
(2) 重複アドレス検知

・近隣要請を出す

　終点アドレスは、要請マルチキャスト・アドレス

　始点アドレスは、未指定アドレス(::)

　対象アドレスは、自分の仮のアドレス



・対象アドレスが重複した場合

　近隣通知で重複を知らせる

　終点アドレスはリンクローカル全ノード・マルチキャスト

リンクローカルアドレスの重複の検知に近隣要請を使いますが、これは実は ICMP バージョン 6 というものにカプセル化されています。このアドレスが存在するかということを問いかけるわけですが、そのときに始点のアドレスは未定義にしておきます。自分の今作っているリンクローカルアドレスはあくまで仮なので、ソース、始点にはその値を入れることはできません。未指定のアドレスを入れるということで、全部ゼロで埋めて出します。それで何の反応もなければユニークですから、それを使ってよいことになります。もし同じアドレスがいる場合は、近隣通知で重複していることを知らせます。重複していた場合は IPv6 のスペックでは未定義なので、マニュアルで、手で設定し直すということになると思いますが、ほぼ重ならないということはわかっているので、念のため重複アドレスを検知して反応がなければそれを使い始めるということになると思います。

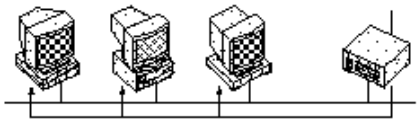
(3) デフォルト経路とプレフィックスの取得

・ルータ通知

　定期的なアナウンス(to ff02::1)

　ルータ要請(to ff02::2)への反応

　デフォルト経路、プレフィックス、etc.



・グローバル・アドレスの生成

　プレフィックス + インターフェイス識別子

　(例) 3ffe:0501:0808::260:97ff:fe40:efab

ルータ経由でインターネットに出ていくには、グローバルなアドレスが必要ですが、定期的に全ノードアドレスにルータ通知というものをアナウンスしています。そこにはデフォルト経路やプレフィックス、つまり自分のこのネットワークのグローバルの部分の上 8 バイトがありますので、自分が作った下のインターフェースの 8 バイトの識別子と上のプレフィックスを重ね合わせてグローバルアドレスを作ります。ルータ通知を行っている人をデフォルトルータだと思って、そちらに経路制御を向けると、インターネットに接続できることになります。

注意して頂きたいのは、IPv6 ではネームサーバをどのようにして解決するかは、まだ決まっていないということです。これは別途やらなければいけません。

(4) アドレス解決

・ARP の抽象化

IPv4 と違いデータリンクごとに ARP を定める必要はない

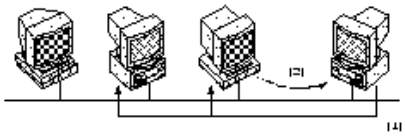
・近隣要請

近隣要請マルチキャスト・アドレスに近隣要請

・近隣通知

受信ホストは対象アドレスを検査

対象アドレスが一致すると MAC アドレスを応答



これでもう通信できますが、イーサネットのレベルでは、イーサネットのアドレスがわからないと通信できませんので、IPv6 のアドレスをイーサネット・アドレスに変換しなくてはいけません。

これはいわゆる ARP というものですが、IPv4 の ARP というのはイーサネットや FDDI などで個別に定義されていましたが、IPv6 では ICMP の上に乗っていますので全部抽象化されています。ですからメディアが増えるたびにたくさんのコードを書いて対応するのではなくて、ほんの少しだけ書き換えることでいくつものメディアに対応できるということになっています。

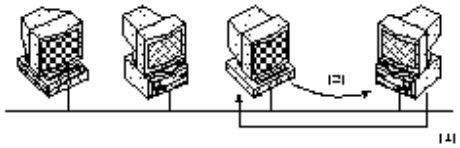
ここで何をやるかといいますと、全ノードアドレスに対して、この IP アドレスに対応するイーサネットを返して下さいというのではなく、近隣要請マルチキャスト・アドレスで、全ノードではなく、それに自分がほしいと思っているアドレスに登録していないホスト、いくつもあるかもしれませんが、たいてい一つです、それに対してパケットを送ることで、他の人には迷惑をかけないということになっています。細かい話をすると IPv6 というのは他のホストには迷惑をかけないという点で IPv4 よりも有利ですが、ネットワークが速くなると気にしなくてもよくなるので有利とはいえないかもしれません。

到達不能検知

・近隣キャッシュ(ARP 表)は状態を持つ

・ユニキャストの近隣要請

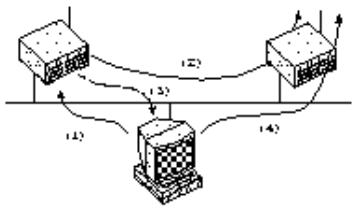
時間がたって「あやしく」なったとき



ノートブック PC などでは、接続した後でネットワークから切り離されてしまい、到達が不能になるかもしれません。ですから解決した ARP では、実は状態を持たせ、タイマーを持たせて定期的に検索しています。怪しくなると、本当にまだいますかというのをおこなって、到達可能かどうかを定期的に測るということをやります。

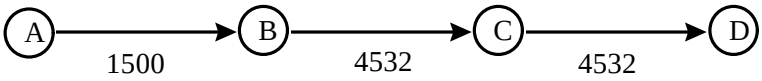
向け直し

・ホストは経路制御に参加しない
・ホストははじめデフォルト経路のみを持つ
・間違っている経路は向け直してもらい、学習する
・次からは正しいルータに転送できる



IPv4 **でも向け直しという機能がありますが、使い方を知らない方が結構いると思いますので紹介しておきます。出口のルータが複数あるとき、**IPv4 **ではデフォルトルータは一個しか指定できないので、どちらかに向けています。この場合は左の方に向けていますが、ある終点に対するパケットが、本当は右のルータに送るのが正しいという状態だったとします。このときパケットを投げると、デフォルトである左側のルータに送られますが、左側のルータは実は右に送らなければいけないというのを知っていますので、右に転送して、そのパケットは終点に届きます。ルータは経路が間違っているということを発送元のホストに教えて、そのホストは学習しますので次からのパケットは右側のルータに送られます。これが向け直しで、ホストは始めデフォルトルートだけを切っておき、徐々に学習していくというために、この機能があります。繰り返しになりますが、**IPv4 **でもこういう機能はあり、それを** IPv6 **のために作り直したというだけです。**

経路 MTU **探索**



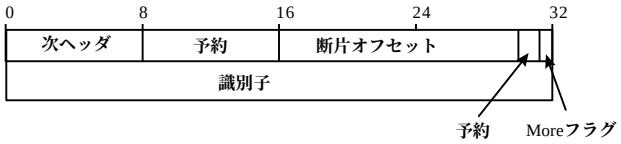
・ローカル MTU **が最小の場合が多い**
TCP **は** MSS(Maximum Segment Size)**を交換**
1500 **が大多数**
・経路上で MTU **が小さくなるのはせいぜい** 1 **回**
経路 MTU **は一方向**
・経路 MTU **探索**
ローカル MTU **で送信**
ルータからのパケット過大メッセージにより補正
・IPv6 **最小** MTU
1280= 1024 + 256
(was 576)

IPv4 **ではパケットを途中のルータが分割してくれます。例えば** IPv4 **で通信するとき、途**

中に FDDI やイーサがあると、ルータがパケットを分割して、3 つのパケットに分割して送るということをやっていたわけです。分割することで、TCP の通信の単位と IP の通信の単位の区切りが違ってきますので、これは罪悪だといわれています。一つ分割した後にすぐ届いているならばよいのですが、経路の途中で分割されてその一つが失われてしまうと、結局 TCP 全体の発送をやり直さなければいけません。
分割は途中のルータではなく、始点のホストがやるというのが IPv6 の考え方です。IPv6 の場合はバケットを送ると、途中のルータが分割するのではなく分割できませんというメッセージを返します。そうすると次からは、では 1,500 バイトですねということで、ちゃんと TCP と IP の通信単位がいっしょになって出ていくということになるわけです。
実は TCP が通信を開始するときには、TCP の通信単位である MSS というのを TCP のオプションで交換しています。パケットがあり、例えばイーサでは 1,500 バイトで、IP ヘッダと TCP ヘッダを引いた数を MSS といいますが、実はこれをあらかじめ交換しているので、先ほど分割が起こるような話をしましたが、それはうそで、TCP を使う限りは、非常に小さいパケットですから絶対分割されません。しょせん世の中の MTU というのは 1,500 バイトなので、相手が 1,500 バイトだとわかれば、ほとんど途中でのパケットの分割というのが起きません。

重要なのは、MTU が大きいところから小さいところへ送る場合には、分割が起きますが、逆の場合は起きませんので、片方向なわけです。経路上で MTU が小さくなっても、せいぜい 1 回だろうということがあります。ほとんど起きないし、起きてもせいぜい 1 回なので、途中のルータで分割しなければいけないときは、そのパケットを破棄してしまってメッセージを送り返すことでもよいというのが IPv6 のアイディアなわけです。
IPv6 では必ず最少のパケットは 1280 以上で、これは 1024 プラス 256 ということですが、これ以下の MTU しかないデータリンクは使ってはいけないということが決まっています。

分割&再構成



・分割は回避した方がよい
TCP の通信単位(セグメント)にあわせる
UDP や IP では分割は避けられない
・ルータでは分割しない
パケットは破棄
パケット過大メッセージを返す
・始点ホストのみで分割
・断片ヘッダ

TCP ではほとんど分割は起きませんが、UDP や IP ではホストが分割しないとどうしようもないということがあります。つまり TCP の MSS のようなものを交換しないので、UDP や IP では、むりやり大きなパケットを送りつけることが可能なので、そういう場合はホストで分割しなければいけません。
それが IPv4 では IP ヘッダの中にあったフラグメント、断片化して再構成するための情報でしたが、IPv6 ではほとんど使わないだろうということでオプションの拡張ヘッダとして、断片ヘッダというものが定義されています。
ping で 1 万バイトを送信しろということをユーザが指定できますが、そのようなときはフラグメントヘッダを付けて分割して送るということになります。例えば FDDI では 3,000 パケットというものを送れるので、IP の場合は 3,000 パケットを出します。そうすると途中のルータが、向こうがイーサなので 1,500 にして下さいということを送り返してきます。ホストは次からは 1,500 バイトに分割して、このフラグメントヘッダを付けて出すということになるわけです。

IPv6 の現状

・IETF の IPng 分科会

http://playground.sun.com/pub/ipng/html/ipng-main.html

・ルータ

Bay Networks, Cisco, 3Com

・ホスト&ルータ

日立、東芝、富士通

WIDE プロジェクト、INRIA

NRL, UNH, etc.

実は先週 IETF でオーランドに行ってきましたが、ここに書いていることは、もうほとんど情報量がないのではないかというぐらい、たくさんのことが起こりました。

BSD 系の実装では、僕たちがやっている KAME、そして NRL、INRIA というものがありますが、マージしましょうということが IETF で決まりました。FreeBSD、OpenBSD、NetBSD、BSD/OS、すべてにマージしたものを入れるということになり、来年の夏をめどにマージしますので、どれを使おうかということで迷わなくてすみます。

もう WIDE プロジェクトでは IPv6 は当たり前のように使っていますが、そういうことをきちんと仕掛けていこうということで、IPv4 でのネットワーク間の調整をする機関に NANOG、日本では JANOG というのがありますが、それに相当する 6REN というのを作るということになりました。どこに web ページが上がるかわかりませんが、www.kame.net からはリンクを張るようにします。世界的に教育機関や研究機関でまず v6 を使いだして、徐々にビジネスサイドに移行させようというような少し政治的な機関が作られることになりました。

それともう一つビジネスサイドの BOF がありまして、IETF のような堅苦しいメーリングリストではなく、どういうのが今使えて、どういう設定をしたらいいのかというようなユーザグループのメーリングリストが上がることになりました。これも KAME のホームページからリンクを張っておきますので興味のある方は見て下さい。

現状は、ほとんどできていますが、世の中がまだ移行を迷っていて移行できないという状況だと思います。

4. IPv6 への移行

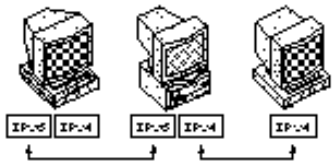
移行のストーリー

・初期
　IPv4 が多数、IPv6 が少数
　デュアル・スタック
　IPv6 in IPv4 トンネル
　トランスレータ
・後期
　IPv6 が多数、IPv4 が少数
　IPv4 in IPv6 トンネル
　トランスレータ

だいぶ移行のストーリーが明確になってきまして、移行には初期と後期があるということがわかってきました。
初期というのは今ですが、IPv4 が多くて IPv6 が少ないときです。このときにはデュアル・スタックとか IPv6 in IPv4 トンネルとかトランスレータというものが役に立ちます。それから後期では逆に v6 が多くなって v4 が少なくなります。そのときはこの逆の、IPv4 in IPv6 トンネルとかトランスレータとかが役に立ちます。

デュアル・スタック

・デュアル・スタック・ホスト
　初期の IPv6 ホストは IPv4 もしゃべる必要がある
　IPv4 ホストとは IPv4 で
　デュアル・スタック・ホストとはどちらでも可
　DNS は IPv4 で検索
・BITS(Bump In The Stack)
　OS の入れ換えなしにデュアル・スタック機能を持たせる
　　IPv4 ホストのドライバを入れ換える

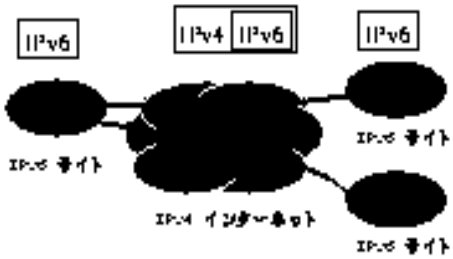


初期の IPv6 というのは必ずデュアル・スタックにしなさいということが決まっていまして、デュアル・スタックというのは IPv4 と IPv6、両方を話すバイリンガルです。こうしておくとデュアル・スタック同士は IPv6 で話せるし、IPv4 のホストとは IPv4 のプロトコルで話せます。
実はよく考えると落とし穴があり、IPv4 のアドレスはしょせん少ないので、いくらデュアル・スタックにしたところで IPv4 のアドレスを付けられませんから、やはりトランスレータが必要なのだということが最近わかってきました。

このデュアル・スタックにするためには OS の再インストールなどが必要と思われるかもしれませんが、最近、IPsec で作られた BITS(Bump In The Stack)という新しいテクニックがでて来ました。
例えば、IPsec ですが、Windows2000 には IPsec は入っていますが、Windows98 には入っていません。Windows98 に IPsec を入れるとき、OS 入れ替えではなくドライバを入れ替えることで IPsec のスタックにしてしまうという技術があります。これが BITS というものですが、その技術を応用してドライバを入れ替えることで、例えば 3COM のドライバを変えることでデュアル・スタックにしてしまうことができます。
これは日立がフリーで出していますので、興味がある人は見て下さい。KAME の web ページからたどれると思います。

IPv6 in IPv4 トンネル

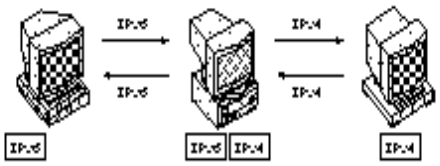
・IPv6 サイトは IPv4 の海に浮かぶ島
・IPv6 島を接続
　IPv4 インターネットをデータリンクだと思う
　IPv6 パケットを IPv4 パケットにカプセル化



今 IPv4 のインターネットがあって、IPv6 の島があるわけですが、トンネルというもので IPv6 同士が通信することができます。IPsec にトンネル・モードというのがあったように、あるパケットを単にデータだと思って他のパケットにカプセル化するというのがトンネルです。IPv6 のパケットを単なるデータだと思って、IPv4 に埋め込んで IPv4 の海を越えていき、向こう側で戻してもらって通信します。今の初期の状況では、IPv6 in IPv4 というトンネルを使うことで島どうしを接続できる。この例が 6bone です。

トランスレータ

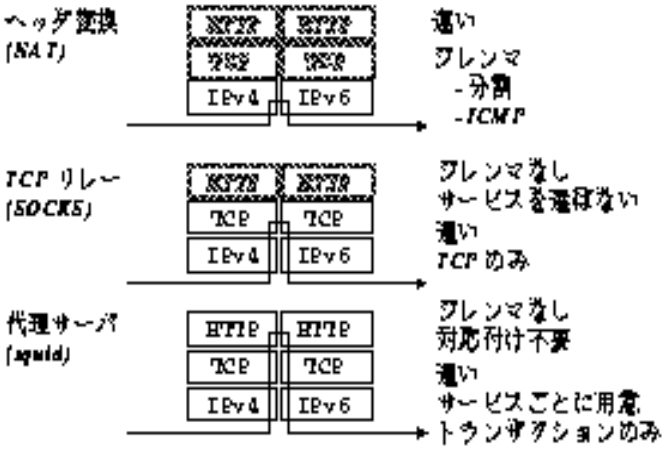
・移行初期から IPv6 ホストが出現
・移行後期も IPv4 ホストは残る
　IPv4 アドレスはいずれ枯渇する
・IPv4 ホストと IPv6 ホストが混在する
　通訳が必要
・トランスレータ
　プロトコル変換
　アドレスの対応付け



今までデュアル・スタックにしておけばよいといわれてきましたが、しょせん IPv4 のアド

レスは少ないのでデュアル・スタックを持ってきたところでそれは IPv6 のホストだということになりました。IPv4 のホストもあるということで、本当に少ししかない IPv4 のアドレスを奪い取ったバイリンガルのホストがトランスレーションしないと IPv6 と IPv4 のホストはつながらないということがだいぶわかってきました。昔は、移行の後期に IPv4 を捨てられないホストを救うためにトランスレータを作るということでしたが、移行の初期からもういきなりトランスレータがないと話にならないということがわかってきましたので、このトランスレータが重要になってきているわけです。

プロトコル変換

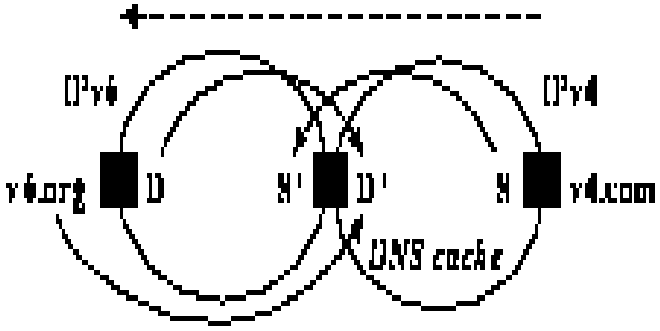


トランスレータにはいくつか種類があり、ここにあげているもの全て作られています。
まずヘッダ変換ですが、ほとんど NAT のようなことをします。IPv4 ヘッダを IPv6 のヘッダに書き換えることで通訳してくれるというのがヘッダ変換方式です。これはかなり速いですが、ジレンマがたくさんあります。例えば IPv4 のヘッダは 20 バイト、IPv6 のヘッダは 40 バイトなので、どうしてもぎりぎりで来たパケットを IPv6 に変換しようとすると分割しないといけません。ここが始点になりますので分割は許されますが、何となく IPv6 は分割をやめたのに分割しなければいけないということになります。IPv4 では ICMP に "source quench"というようなコマンドがありますが、IPv6 にはないので、ICMP バージョン 4 を ICMP バージョン 6 には変換できないとか、いろいろ問題があるわけです。
TCP リレーという方式は SOCKS のようなものです。TCP over version4 などというのは面倒なので、TCP4 などと呼びますが、TCP4 と TCP6 をデーモンで実装し、TCP4 のコネクションが来るとむりやり奪い取って、あたかも自分が終点であるようにふるまって、もう一個、TCP6 という新しいコネクションを張ってデータを転送するというものがあります。これは TCP のコネクションが 6 と 4 で閉じていますので、上であげているようなジレンマはありませんが、そんなに速くありません。カーネルでは作れないのでハードウエア化できないなどの問題があり、コネクションの終了がわかる TCP のみにしか使えないとい

う問題もあります。
代理サーバですが、例えば HTTP でバイリンガルのサーバにプロキシを向けておきます。そうすると実際取りにいくときは、実は IPv6 を使うということにできます。これは非常に素直なアプローチで何の問題も起きません。
あえて問題をあげるなら、いろいろなサーバがあるわけで、HTTP サーバやメールサーバなどサービスごとに用意しなければいけません。また、インタラクティブもの例えば telnet というのはプロキシとは相性が悪いですから、トランザクション指向のものしか扱えないとかというような、一長一短がそれぞれあるわけです。
これらのプロトコル変換は全部実装があり、使うことができます。

アドレスの対応付け

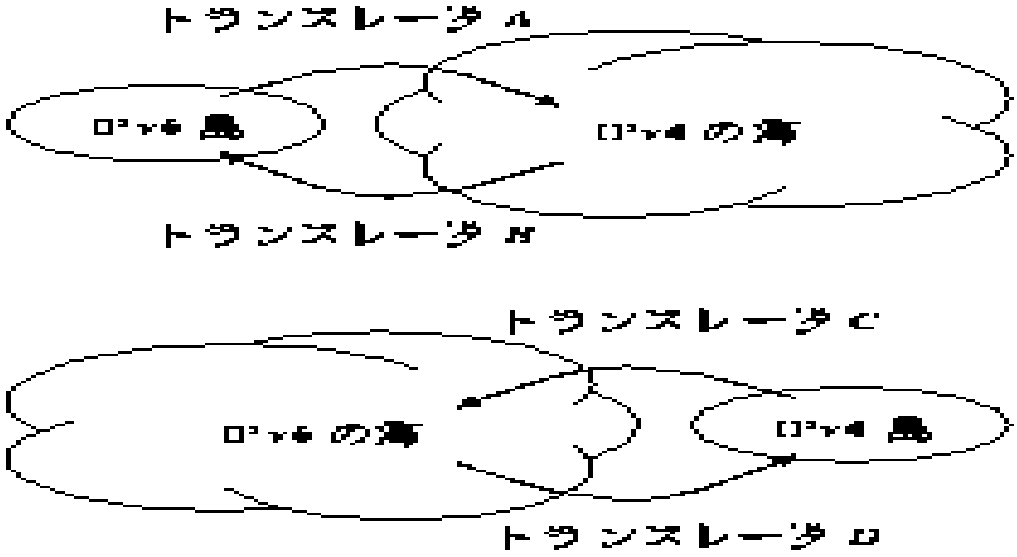
・v4.com が v6.org に通信を始める
・トランスレータは v6.org に IPv4 アドレスを割り当てる
・終点アドレスの対応付けが本質
　始点アドレスはトランスレータのアドレスで代用可
・IPv4 から IPv6 への対応付けは容易(静的)
・IPv6 から IPv4 への対応付けは困難(動的)
　IPv4 アドレス・プールは小さい
　DNS のキャッシュ問題



先ほどの代理サーバ方式では全然問題が出ませんが、ヘッダ変換方式や TCP リレーでは、必ず IPv4 のアドレスを IPv6 に対応付けなければいけません。逆も対応付けなければいけませんが、小さいものを大きいものに埋め込むのは簡単ですが、その逆はできないのでなかなか難しくなってくるわけです。また、割り当てたアドレスは、DNS のキャッシュ問題を起こしてはいけないという制約があります。
DNS のキャッシュ問題について説明しておきます。トランスレータの両側に IPv6 と IPv4 のネットワークがあり、IPv4 の v4.com というのが v6.org に通信を開始したとします。ネームサーバで名前を引くと、v6.org は IPv6 ですがトランスレータが IPv4 を返しますので、相手は IPv4 だと思って通信するわけです。この対応付けたアドレスは、お互いにリーチャブルである必要があります。また、v6.org には IPv6 のアドレスしかないのに IPv4 のアドレスを動的に対応付けて返しますので、その情報は IPv4 のネットワークに DNS のキャッシュとして広がります。
これは NAT の問題と同じですが、IPv4 のアドレスは IPv6 のサイトには、少ししか割り当ててありませんので、どんどん再利用して IPv4 に割り当てていく必要がありますが、その

アドレスがキャッシュで外に出てしまうとそのアドレスプールというのは非常に小さいですから、これがキャッシュの問題を起こすと困るということがあります。

トランスレータの分類



トランスレータというものは、アドレスマップの視点から見ると四つに分かれます。初期と後期があり、初期の方の IPv6 の島から外へ出ていくときを A と呼んで、逆方向を B と呼びます。ここで方向とは、コネクションを開始する方向のことです。後期の方を C と D と呼びます。

初期における、IPv6 のホストが IPv4 に通信を開始するためのトランスレータは簡単です。これはグローバルの IPv4 をグローバルの IPv6 に静的に割り当てればよいわけで、静的ですから DNS のキャッシュ問題というのは起きません。あらかじめ登録しておくということもできますし、何をやっても問題は起きないので実装は非常に簡単です。

(1) トランスレータ A と B

・トランスレータ A

グローバル IPv4→グローバル IPv6(静的)

実装は容易

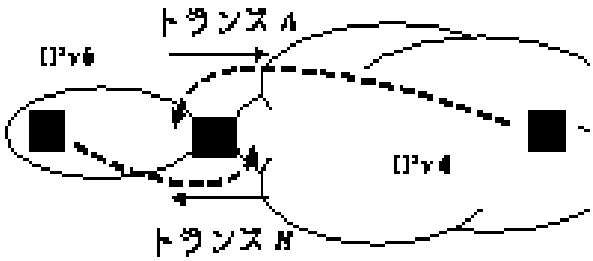
・トランスレータ B

グローバル IPv6→グローバル IPv4(動的)

グローバル IPv4 アドレス・プールは小さい

DNS キャッシュが IPv4 の海に拡散する

実装不可能に近い



KAME には TCP リレー方式の FAITH というコマンドが付いています。この会場は IPv6 のアドレス、コネクティビティが来ていますから、IPv6 のホストを持ってくるとプラグ&プレイでアドレスが取れます。KAME のネットワークにはトランスレータを一つ立ち上げていますので、そのアドレスを指定して通信すると外に出ていきます。実はその IPv6 のパケットは下 4 バイトのアドレスに通信して下さいというようなことを書いて出すわけです。ネームサーバには SSH だと ssh.iij.ad.jp というようにすると、ローカルのリゾルバが KAME ネットワーク行きのプレフィックスを付けて上に戻してくるので、SSH は IPv6 で通信を開始します。トランスレータは下の 4 バイトを取り出して、それをリレーして IIJ に接続しているということです。実際にこれは今でも使えています。

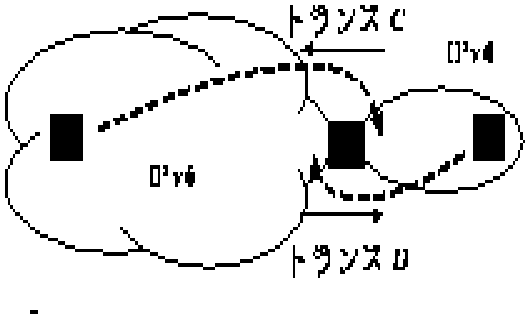
NAT とまったく同じで逆向きは非常に困ります。トランスレータ A の IPv4 のアドレスを IPv6 に組み込むということは非常に簡単ですが、IPv6 のアドレスを IPv4 にマップするというのは動的にしかできませんし、その対応関係というのはインターネット全体に広がってしまうので、多分トランスレータ B というのは作れません。

方法としては、IPv4 から IPv6 のネットワークにアクセスしたい場合は、IPv4 のネットワークに見せたい IPv6 のサーバがあるわけなので、その IPv6 のサーバをデュアル・スタックにして IPv4 からもアクセスするというものがあります。トランスレータというのはあきらめてサービスを提供したいサーバだけをデュアル・スタックにして当面しのぐということです。

(2) トランスレータ C と D

・トランスレータ C
　グローバル IPv6→プライベート IPv4(動的)
　プライベート IPv4 アドレスを利用可能
　DNS キャッシュは IPv4 島に閉じる
　実装可能

・トランスレータ D
　グローバル IPv4→グローバル IPv6(静的)
　実装は容易



当分先ですが、移行後期には IPv6 の方が大きくなります。そのときにはトランスレータ C というものと外向きと内向きのトランスレータ D というのがあります。トランスレータ D というのは IPv4 から IPv6 への変換で、これは静的ですし実装は容易です。SOCKS64 というのが富士通からでていますが、それはトランスレータ A でも使えるし、D としても実際に使えます。

それからトランスレータ C を考えると、アドレス・プールというのはプライベート・アドレスを利用できるので、 B に比べると容易です。

B というのは少ない IPv4 のグローバルなアドレスをアドレスプールとしなければいけませんでしたが、C の場合はプライベート・アドレスをアドレス・プールに持てるので、たくさんマップする先はいっぱいあります。それから DNS のキャッシュが広がってしまうのはサイト内のみなので、面倒だがやってやれないことはありません。

まとめますと、今すぐ使い始めたいトランスレータ A というのはもう簡単に作れますし、実際に使っている人もたくさんいます。B というのは恐らく作れないので、公開したいサーバはデュアル・スタックにするという方法をとるということです。後期は多分すごく時間がかかると思いますが、トランスレータ D というのは簡単ですし、C というのはやってやれないことはないですが、非常に面倒くさいということが最近わかりました。

5. Q&A

・ルーティングプロトコルの現状とInternet2X との関係

一点めは、まずルーティングプロトコルの現状です。まず昔は RIPNG ですべてを解決していましたが、歴史は繰り返すものでやはりそれは破綻しまして、今は BGP4 プラスという BGP を拡張したもので AS 間では経路制御を交換しています。AS 内ではいまだに RIPNG ですが、OSPF をとにかく作らなければ負けだろうということで、この前の IETF でも OSPF の IPv6 対応をしなければ全然使ってもらえないという危機感を持っています。KAME プロジェクトでは、ある優秀なルーティングエンジニアを採用することになっており、その人が作るということになっています。どれぐらいでできるかわかりませんが、やろうとしています。まとめますと、AS 間では BGP4 プラスを使っています。それから RIPNG は使いますが、OSPF はないという状況です。

それから Internet2 みたいなものとの関係ですが、Internet2 につながるためにアジア・パシフィックの連合である APAN というものがあります。かなりポリシーが強いのでアカデミックとか研究とかしかできませんが、IPv6 を重要なテーマとしてとらえおり、IPv6 に限ってはコマーシャルでもよいということになっていますので、関連は非常にあります。

・IPv6 の需要

これはもう鶏と卵の問題ですが、NAT で接続されたインターネットで永遠に我慢できると思うと需要はありません。NAT にはたくさん問題がありますが、それをなくすためには IPv6 しかあり得ません。技術的な問題を考えて、やはり IPv6 でないといけないと思うなら移って行かなければいけないし、NAT でよい IPsec もいらないと思えば需要はゼロです。それぞれ人は考え方が違うのでまとまらないとは思いますが、IPv6 の方がよいということを理解してもらって、移行はたいへんだけれどもやっていきましょうということで、先ほどの 6REN や v6 ユーザ会などを作って皆をできるだけ説得しようと思っているところです。

企業ではやはりファイアウォールがあるので、プライベートアドレスでいいと思う人が多いです。しかしこれはインターネットの単なる一瞬の姿であって、これから実際には家で使いたいとか自動車で使いたいとか、たくさん出てきます。会社レベルで使っている限りは、パッチされたインターネットでいいのですが、実際に自分たちの生活に必要なものとしてインターネットを使っていくという場合、皆が想像できているかどうかという話であって、それを想像したら今のインターネットではおかしいので、絶対に必要だとは思っていますが、それを皆さんがコンビンスしてくれるかどうかはかなり怪しいという感じです。

以上