平成23年度

論文題目

nチャネルメッセージ伝送方式のための jailによる経路制御

神奈川大学 工学部 電子情報フロンティア学科

学籍番号 200802991

末田　雄己

指導担当者　木下　宏揚 教授

目 次

図 目 次

表 目 次

第1章

序論

1.1　序論

従来の暗号通信は公開鍵暗号方式というものが使用されている。これは鍵と第三者機関を必要とする通信である。一方 n チャネルメッセージ伝送方式は鍵や第三者機関を必要としない通信である。これは n 本経路を使用して通信を行いそのうち何本かに盗聴や改ざんを行う敵が潜んでいても、残りの経路で文書を復号することができ、敵には文書の中身がわからないという安全な通信を行うことができるというものである。

現在のインターネットの経路制御の仕組みは、一つの経路で通信を行う仕組みになっている。これに対して本研究で用いる n チャネルメッセージ伝送方式の経路制御は複数の経路を使用して通信を行う仕組みになっている。これは、現在のインターネットの仕組は n チャネルメッセージ伝送方式を行うにあたって適していない。そこで本研究では n チャネル通信の経路制御について考え、複数の経路を用意するために、FreeBSD という OS の jail と Vimage という機能を用いる。

jail を用いて経路を複数使うための仮想ルータを作成する。jail を用いる理由は、複数の経路を確保するための仮想ルータを一台のパソコンで簡単に大量作成できるためである。また、Vimage を用いることで仮想ルータどうしをつなげ、仮想環境内で通信を行えるようにする。これらを用いることによって n チャネルメッセージ伝送方式を検証できる仮想環境を構築することを目的とする。

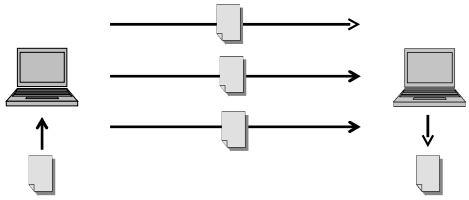


図 1.1 n チャネルメッセージ伝送方式

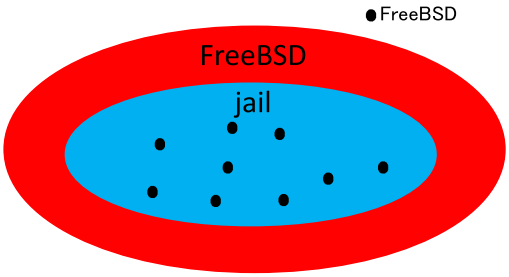


図 1.2 jail

第2章

nチャネルメッセージ伝送方式

従来の公開鍵暗号方式では公開鍵の正当性を証明するために認証局のような信頼できる第三者機関が必要であった。それに対してn チャネルメッセージ伝送方式では事前の鍵が不要なため第三者機関も必要ない。そこで本論文ではn チャネルメッセージ伝送方式に着目している。

n チャネルメッセージ伝送方式は文書をn 本の通信路を使用してファイルを分散させて通信を行う方式である。分散されたファイルが、すべて違う経路を通り相手に届くことが理想である。しかし、現在のネットワークシステムでは、送信先しか指定できないので、n 本の経路を用意するのは不可能である。n 本の経路を用意するために、本研究では、経路制御（ソースルーティング）に着目した。n チャネルメッセージ伝送方式にはPSMT とASMT という方式がある。

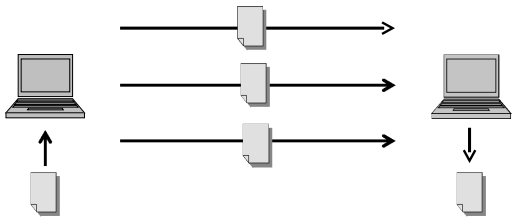


図 2.1　n チャネルメッセージ伝送方式

2.1 PSMT(Perfectly Secure Message Transmission)

2.1.1 PSMTにおける安全性の定義

n チャネルメッセージ伝送方式において次の 2 つの条件を満たしたものを PSMT と呼ぶ。

1. 敵は送信メッセージに関する情報を何も得られない。(盗聴耐性)
2. 受信者がメッセージを正しく受信できる確率が 100 %である。(改竄耐性)

また、送信者が受信者に 1 回送信するだけで済む方式を 1-round 方式、送信者と受信者が相互に r 回やり取りを行う方式を r-round 方式と呼ぶ (図 4.2)。
このとき敵が n 本の通信路のうち t 本に潜んでいるとしたときに PSMT プロトコルが存在するための必要十分条件は、1-round 方式では $n \geqq 3t+1$、2-round 方式では $n \geqq 2t+1$ であることが証明されている。

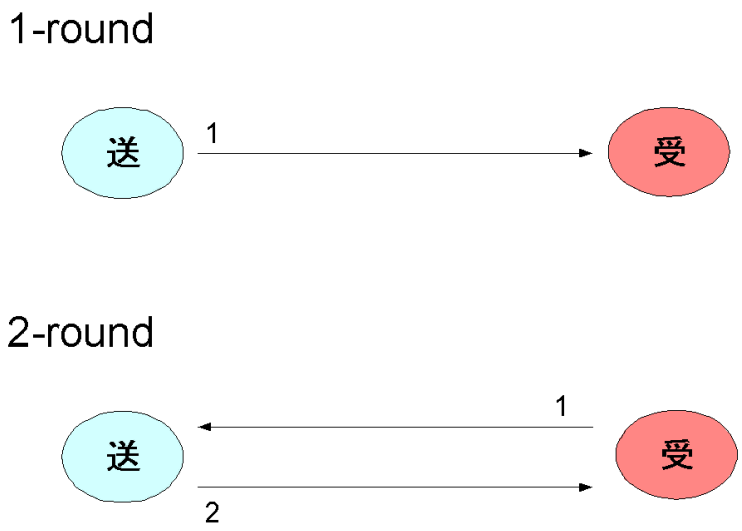


図 2.2 1-round 方式と 2-round 方式

2.1.2 PSMTの歴史

PSMTは1993年にDolevら[13]によって提案された。彼らは、敵がn本の通信路のうちt本に潜んでいるとしたときにPSMTプロトコルが存在するための必要十分条件は、1-round方式では$n \geqq 3t+1$、2-round方式では$n \geqq 2t+1$であることが証明し、また、それぞれの通信量が$O(n)$、$O(2^n)$のプロトコルを提案した。その後2-round方式は1996年にSayeedら[24]が通信量$O(n^3)$のプロトコルを提案し、2006年にはAgarwalら[25]が大量の文書を送ることを前提に通信量$O(n)$で計算量が指数関数的であるプロトコルを提案した。そして2008年にはKurosawaら[26]が通信量$O(n)$、計算量$O(n^3)$となるプロトコルを提案した。このように2-round方式の通信量、計算量は改善されていった。一方、1-round方式は、通信量$O(n)$、計算量が多項式時間となるプロトコルをDolveらが最初に提案しており、既にそれが$n \geqq 3t+1$における最善のプロトコルであった。PSMTにおいては、$n \geqq 3t+1$でなければ使えない1-round方式よりも、$n \geqq 2t+1$で使える2-round方式のほうが優れている。そこで、1-round方式における必要十分条件を$n \geqq 2t+1$に改善するために生まれたのが次に述べるASMTである。

	PSMT	
	2-round	1-round
Dolev 1993	$n \geqq 2t+1$が必要十分条件 通信量：$O(2^n)$	$n \geqq 3t+1$が必要十分条件 通信量：$O(n)$
Sayeed 1996	通信量：$O(n^3)$	
Agarwal 2006	通信量：$O(n)$ ただし大量の文書を送ることが前提 計算量：指数的	
Kurosawa 2008	通信量：$O(n)$ 計算量：$O(n^3)$	

表 2.1 PSMTの歴史

2.2 ASMT(Almost Secure Message Transmission)

2.2.1 ASMT における安全性の定義

ASMT における安全性の定義は以下のとおりである。

1. 敵は送信メッセージに関する情報を何も得られない。(盗聴耐性)
2. 受信者がメッセージを正しく受信できる確率が $1-\delta$ 以上である。(改竄耐性)
3. 受信者が正しく受信できない確率が δ 以下であり、そのとき受信者は failure を出力できる。(失敗検知能力)
4. 敵が t 本の通信路を遮断しても受信者は残りの通信路で得た情報だけからメッセージを受信できる。(遮断耐性)

PSMT と比較して主に異なる点は定義 2 においてメッセージを正しく受信できる確率が $1-\delta$ となっており、失敗した時はそれを検知できるという定義 3 が加わっている点である。

2.2.2 ASMT の歴史

ASMT は 2004 年に Srinathan ら [14] によって提案されたが、そのプロトコルには間違いがあった。その後 2007 年に Kurosawa ら [15] によって厳密に定義された。そのなかで $n = 2t + 1$ での通信効率の限界が以下のように示された。

$$|X_i| \geqq (|S| - 1)/\delta + 1 \tag{2.1}$$

X_i：channel(i) を流れる情報の集合
S：秘密情報の集合
δ：失敗確率

また、Kurosawa らは通信効率の限界に近い通信量で通信できるプロトコルも提案した。そのプロトコルの通信効率は失敗確率を ϵ とすると、

$$|X_i| = \frac{|S|-1}{\delta} + 1 > \frac{|S|-1}{\epsilon} + 1$$

$$\text{ただし } \epsilon = \left\{ \begin{pmatrix} n \\ t+1 \end{pmatrix} - \begin{pmatrix} n-t \\ t+1 \end{pmatrix} \right\} \delta$$

となっている。

2.2.3 Basic プロトコル

Kurosawa らが提案したプロトコルは計算量が指数関数的であるという問題があった。木下研究室がこれを改善して多項式時間となる Basic プロトコルを提案した。

2.2.4 Basic プロトコルの通信量

Basic プロトコルの特徴はハッシュ関数 H を用いる点であった。
敵の Second Preimage Attack が成功したときがこのプロトコルの失敗となる。敵は t 個のハッシュ値に Second Preimage Attack をするので、このプロトコルの失敗確率 ϵ は、

$$\epsilon \leqq [\text{ハッシュ関数 } H \text{ への } \mathit{SecondPreimageAttack} \text{ が成功する確率}] \times t \quad (2.2)$$

ここで H の出力値のビット数を h とすると、出力値は 2^h 通りとなり、H がランダムオラクル (ランダムに値を出力する) と仮定すると H への Second Preimage Attack が成功する確率は $1/2^h$ となる。したがって式 (4.2) は

$$\epsilon \leqq \frac{t}{2^h} \quad (2.3)$$

となる。さらに秘密 s の長さを q ビットとおくと秘密の集合 S との関係は

$$|S| = 2^q \tag{2.4}$$

通信効率の限界式 (式 (4.1)) と式 (4.3)、式 (4.4) より

$$|X_i| \geqq \frac{|S|-1}{\epsilon} + 1 \geqq \frac{2^q - 1}{t/2^h} + 1 = \frac{2^h(2^q-1)}{t} + 1 \tag{2.5}$$

となるので、Basic プロトコルの実際の $|X_i|$ が式 (4.5) の右辺に近ければ通信効率が限界に近いと言える。次にビット数で評価するために両辺の $\log_2$ をとる。

$$\begin{aligned} \log_2 |X_i| &\geqq \log_2 \left\{ \frac{2^h(2^q-1)}{t} + 1 \right\} \approx \log_2 \left\{ \frac{2^h(2^q-1)}{t} \right\} \\ &= \log_2 2^h + \log_2(2^q-1) - \log_2 t \approx h + q - \log_2 t \end{aligned} \tag{2.6}$$

一方、このプロトコルの X_i とはチャンネル ch-i を使って送信者が送る $f(i)$ と $H(f(1))$〜$H(f(n))$ である。秘密 s の長さを q ビットとしているので $f(i)$ も q ビットであり、ハッシュ値は h ビットであるので $|X_i|$ は $q+hn$ ビットとなる。よって式 (5.5) は

$$q + hn = \log_2 |X_i| \geqq h + q - \log_2 t$$

となる。両辺を比較すると左辺の hn と右辺の h との差が大きいことがわかる。したがって Basic プロトコルは通信効率の限界に近いとは言えない。そこで本研究では Basic プロトコルの通信効率を改善したプロトコルを提案する。[6]

第3章

FreeBSD

3.1　FreeBSD

FreeBSD は UNIX 互換の無償で提供されている OS である。FreeBSD はカリフォルニア大学バークリー校の 4.3BSD から派生した Net/2 という OS をベースに開発された、しかし著作権の問題で、4.4BSD-Lite をベースとしたものに移行された。FreeBSD は、セキュリティが優れており高い安定性をなどから、広く使われている。FreeBSD は BSD ライセンスというライセンスで公開されており、フリーソフトウェアとしてソースコードが無償で公開されている。現在では FreeBSD は 9.0-RELEASE まで公開されており本研究でもこの FreeBSD9.0-RELEASE を使用している。[1] [2] [3]

3.2 jail

jail は FreeBSD の機能の一つである。jail は OS レベル仮想化で、jail を使うことによってが FreeBSD ベースのシステムを jail と呼ばれる独立したシステムとして分割することができる。jail は作成したときに割り当てられたファイルシステムおよびプロセス空間しか使用することができない。

jail の利点

1. 仮想化:各 jail は FreeBSD 上で動作する仮想マシンであり、独自のファイルシステム、プロセス空間、ユーザーアカウントを持つ。jail の中のプロセスからは実際のシステムなのか jail の中なのかを区別するのが難しい。

2. 安全性：jail からは他の jail にアクセスできないようになっている。これにより複数のユーザーを持つときのアクセス等の安全性が高まっている。

3. 権限委譲の簡素化:管理者権限のスコープが jail 内に制限されているため、システムの管理者は管理者の権限が必要になる操作をすべての権限を渡すことなく実行させることができる。

[4] 図 3.1 は jail のイメージである。FreeBSD の中に大量の FreeBSD を作成するといったものである。

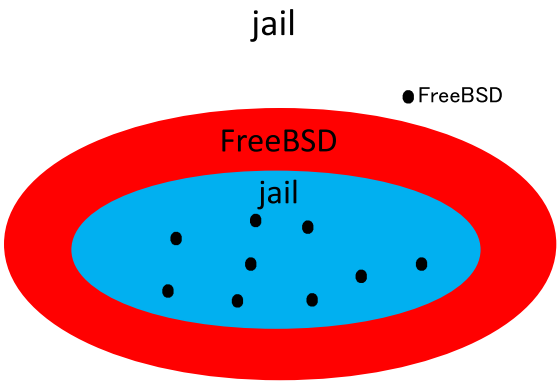


図 3.1 jail

3.3 Vimage

Vimage は FreeBSD の機能の一つである。Vimage は FreeBSD8.0-RELEASE から導入された機能で、これは FreeBSD のネットワークスタックを複数使うためのものである。

ひとつの FreeBSD カーネル上で複数のネットワークスタックを動かすことができ、それらは全て独立して動作する。IP アドレスや、ルーティングテーブルも独立したものになる。1 つの FreeBSD カーネルで複数のノードを動かすことができる。動作中のプロセスとネットワークスタックが複数になる。

一台の FreeBSD で複数のノードを動かすことができるためネットワークの実験など様々なことを試すことができる。[5]

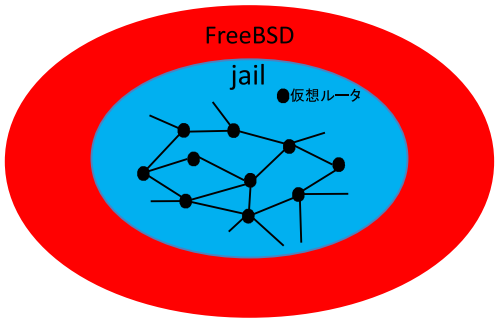


図 3.2 vimage

第4章

経路制御（ソースルーティング）

この章では、経路制御、ヘッダーと実験環境について説明する

4.1　ルーティング

インターネットのように大規模なネットワークでは、あて先ネットワークまでデータを送信したい場合、複数の経路があることが大半である。そこで、パケットの中継地点となるルータが、適切だと思われる経路にパケットを送り出し、あて先ネットワークまでデータを届けるのである。このように、通信経路から最適な経路を選択・制御する仕組みのことをルーティングと呼ぶ。インターネットは、小規模なネットワークが細かい網の目のようになることで世界中を繋いでおり、ルーター同士が、接続されることでつながっている。データパケットをあて先のホストにきちんと届けるためには、各ルーターが正しい経路にデータを転送する必要がある。各ルーターは、自分の持つ経路制御表 (ルーティングテーブル) を参照して、データを転送する。ルーティングテーブルが異なっていれば、ルーターはデータを正しい方向に転送できず、データは目的地まで届かないことになる。

4.1.1 経路制御表 (ルーティングテーブル)

ルーティングテーブルとは IP パケットの中継装置（ルータ）や TCP/IP の通信ができる端末、コンピュータが持つ「次に IP パケットを送るべき相手」を指定したテーブルである。中継装置や端末は IP パケットを送る時は必ずこのテーブルを見て、IP パケットを送るべき相手を決定する。もし、IP パケットを送るべき最終目的地が同じイーサネットに接続されている場合は直接相手に IP パケットを送りますが、相手が異なるネットワークに接続されている時はルーティングテーブルを参照し相手に最も近いと思われる中継装置（ルータ）に IP パケットを送る。
ルーティングテーブルの作成方法にはスタティクルーティングとダイナミックルーティングと言う方法がある。

- 1. スタティクルーティング
  この方法では予め固定的にルートを設定する。ルートが固定的に設定されるので指定されたルートで障害が起きた場合は代替ルートに切り替わらず通信が途絶えてしまう。

- 2. ダイナミックルーティング
  この方法は他の中継装置（ルータ）から送られてくる情報（ルーティング情報）を使って定期的にルーティングテーブルを更新する方法である。この方法では予めルート情報を設定しておく必要は無いし、通信ルートに障害が起きた場合でも、その通信ルートを使って送られてくるルーティング情報そのものが途絶えるため、別のルートに自動的に切り換えられる。従って、幹線系ネットワークにおいて特別な理由が無い限り、ルーティングはダイナミックルーティングを使用するのが一般的となっている。

ダイナミックルーティングが一般的となっているため、複数回通信を行っていると、途中で経路が変わる場合もあるが、複数本の経路を意図的に用いて

通信することはできない。そこで、経路制御を利用して、複数本の経路を用意する。

4.2 経路制御（ソースルーティング）

ネットワーク・プロトコルは一般的に階層（レイヤ）という概念をもとに開発されていて、各階層ごとに役割が異なる。今回の目的である経路制御はネットワーク層で行われる。これは、ネットワーク上でのパケットの移動を制御する。先に説明したように、TCP/IP のルーティング経路は途中のルータが自動的に指定するようになっているが、ネットワーク層、TCP/IP ネットワークのルーティングテーブルに、パケットの通過経路を送信者が指定するのが、ソースルーティングである。途中のすべての経路を指定する「ストリクトソースルーティング」(strict source routing) と、いくつかの経路を通過することを指定して、それ以外は途中のルータに任せる「ルーズソースルーティング」(loose source routing) の 2 種類がある。図 6.1 のように、最短経路ではなく迂回させた経路を辿らせることによって、n 本の経路を確保する。赤の線が最適化された経路であり、緑や青は迂回させたルートとなっている。

図 6.2 のような情報をヘッダーにつけることで、経路制御は実現できる。

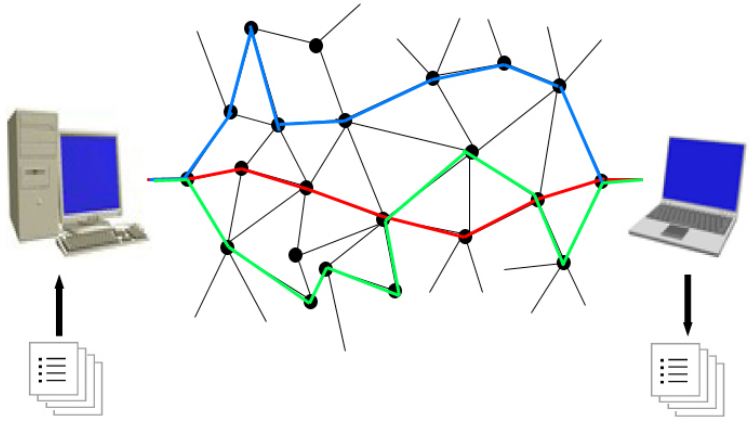


図 4.1　経路制御（ソースルーティングのイメージ）

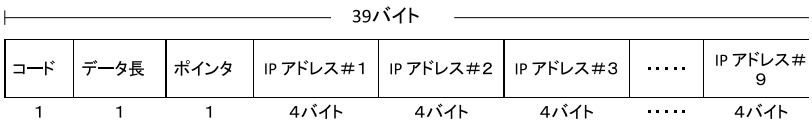
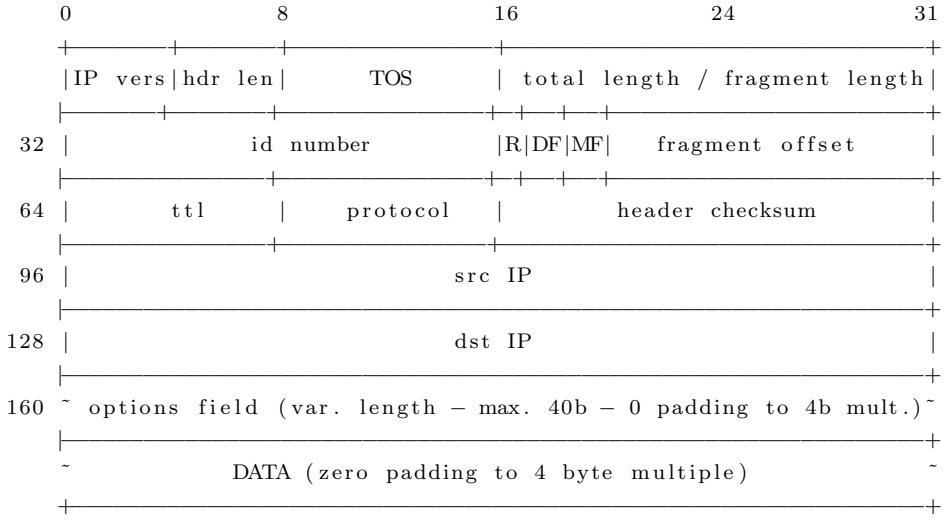
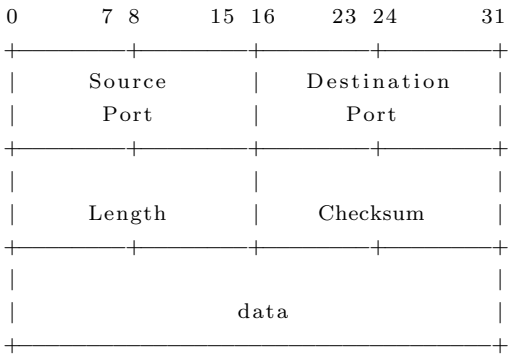


図 4.2　始点経路制御指定の形式

4.3 ヘッダー



IP ヘッダー



UDP ヘッダー

ヘッダーはこのようになっており、IP ヘッダのデータの所に、UDP ヘッダが入り、そのデータ部分に図 6.2 のような、経路制御表を入れることで、経路制御が実現できる。[6]

第5章

提案方法

5.1　FreeBSD を用いた仮想環境の構築

n チャネル通信を行うためには複数の経路を用意する必要がある。そこで、FreeBSD の jail と Vimage を用いて n チャネル通信を行うための仮想環境を構築する。

jail と Vimage を用いる理由は、jail を使うことにより簡単に FreeBSD つまり仮想ルータとなるものを大量に用意することができるためである。しかし jail だけでは各々の jail に独立したネットワークをもてないので Vimage という機能も合わせて使う。これを使うことによって、各々の jail に独立したネットワークを持つことができるようになる。

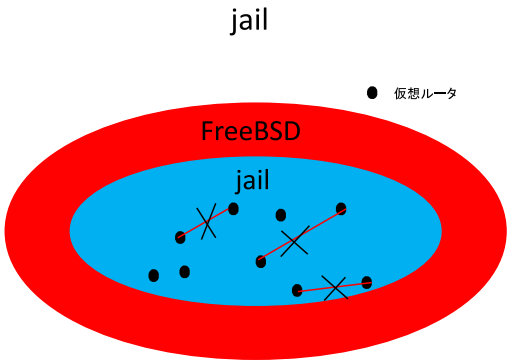


図 5.1　jail

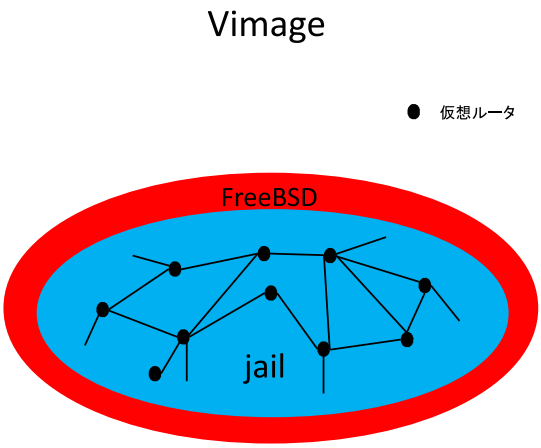


図 5.2 vimage

図 5.1 に示したように jail ではお互いの仮想ルータ間の通信ができない。そこで Vimage を用いることによって図 5.2 のように仮想ルータ間で通信ができるようにしている。この jail と Vimage を用いて作成した仮想ルータを用いて n チャネル通信を行うための仮想環境の作成を行う。この仮想環境の構築には FreeBSD9.0-RELEASE を用いている。

5.1.1 Vimage の設定

FreeBSD の jail で Vimage の機能を使うには設定がいる。設定方法は図 5.3 に示してあるようにまず usr/src/sys/amd64/conf/GENERIC ファイルを編集する。図 5.4 に示している緑で囲ってある部分を GENERIC ファイルに書き加える。この作業が終わったら,

make KERNCONF=GENERIC buildkernel

make KERNCONF=GENERIC installkernel

を行いカーネルのコンパイル、インストールを行う。

これで Vimage を使うことができる。

[5]

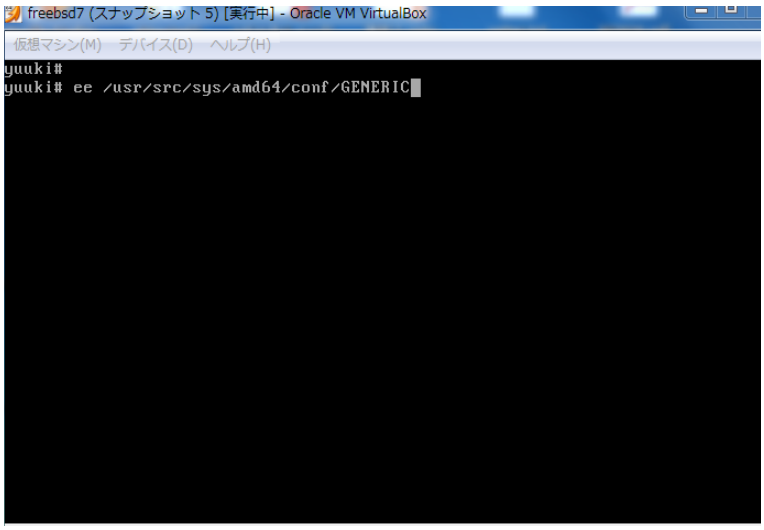


図 5.3 vimage の設定 1

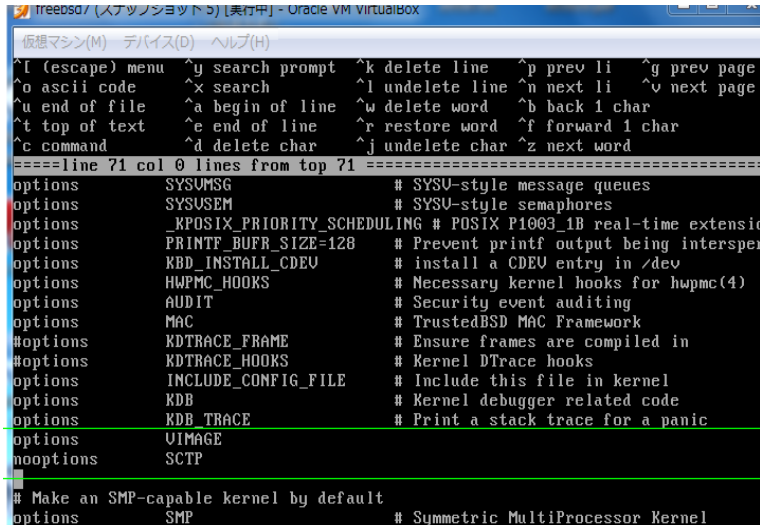


図 5.4 vimage の設定 2

この設定をした後に、jail を作成するために/usr/src 上で

mkdir /usr/local/jails

make buildworld

make installworld DESTDIR=/usr/local/jails/test1

make distribution DESTDIR=/usr/local/jails/test1

make installworld DESTDIR=/usr/local/jails/test2

make distribution DESTDIR=/usr/local/jails/test2

make installworld DESTDIR=/usr/local/jails/test3

make distribution DESTDIR=/usr/local/jails/test3

make installworld DESTDIR=/usr/local/jails/test4

make distribution DESTDIR=/usr/local/jails/test4 [12]

このコマンドを実行して test1〜4 までの jail を作成する。これにより jail の test 環境ができたので個々の設定を行う。そのためのシェルスクリプトを書く。

#!/bin/bash

sysctl -w security.jail.socket_unixiproute_only=0

sysctl -w security.jail.allow_raw_sockets=1

for i in 1 2 3 4; do

jail -c vnet host.hostname=test$i name=vm$i path=/usr/jails/test$i persist

mount -t devfs devfs /usr/local/jails/test$i/dev

mount -t procfs proc /usr/local/jails/test$i/proc

jexec test$i sysctl -w net.inet.ip.forwarding=1

jexec test$i ifconfig lo0 127.0.0.1/24 up

jexec test$i /etc/rc.d/sshd onestart

jexec test$i routed -s -m

done [5]

実行権限を与えるために。chmod +x で実行権限を与える。これにより jail の設定が完了する。

次に各々の jail の通信のための設定をする。

ifconfig epair create

ifconfig epair0a vnet test1

ifconfig epair0b vnet test2

jexec test1 ifconfig epair0a 10.0.10.1/24 up

jecec test2 ifconfig epair0b 10.0.10.2/24 up

ifconfig epair create

ifconfig epair1a vnet test1

ifconfig epair1b vnet test3

jexec test1 ifconfig epair1a 10.0.20.1/24 up

jecec test3 ifconfig epair1b 10.0.20.2/24 up

ifconfig epair create

ifconfig epair2a vnet test2

ifconfig epair2b vnet test4

jexec test2 ifconfig epair2a 10.0.30.1/24 up

jecec test4 ifconfig epair2b 10.0.30.2/24 up

ifconfig epair create

ifconfig epair3a vnet test3

ifconfig epair3b vnet test4

jexec test3 ifconfig epair3a 10.0.40.1/24 up

jecec test4 ifconfig epair3b 10.0.40.2/24 up #ここはうまくいかない。

これにより各 jail をつないでいる。

[5]

次に通信の確認を行う。

jexec test1 ping -c 3 10.0.10.2 #これにより test1-test2 の通信

jexec test1 ping -c 3 10.0.20.2 #これにより test1-test3 の通信

jexec test2 ping -c 3 10.0.30.2 #これにより test2-test4 の通信

これにより通信を確認できた。

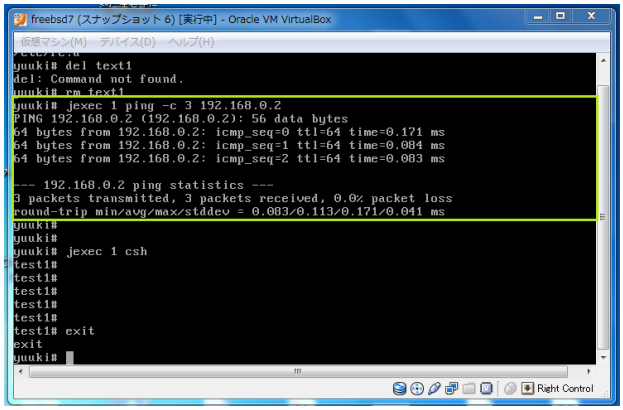


図 5.5 構築した環境のイメージ

5.1.2 仮想環境の構築

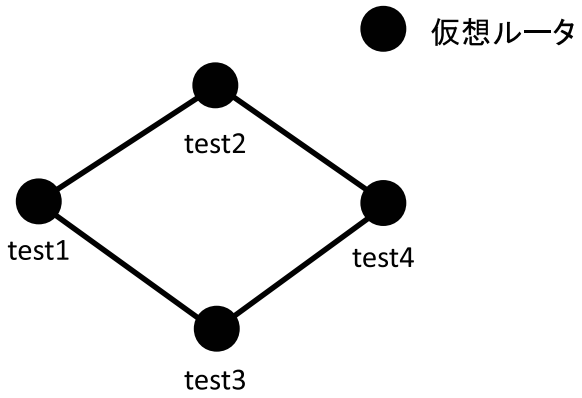


図 5.6 構築した環境のイメージ

jail を用いることによって今回は図 5.5 に示したように test1,test2,test3,test4 を作成した。今回はこの 4 つを用いて仮想環境を構築した。

第6章

結論

今回の研究でjailを用いることによって仮想環境の構築のための仮想ルータを作成することができた。
今回は４つという少ない数で仮想環境を構築したが、jailの機能を用いればもっと多くの仮想ルータを作成することができる。それにより大量の仮想ルータを作成することができるので、nチャネル通信を行うための検証環境として使用できると考えられる。
今後の課題としてはネットワークを拡大してより大規模な環境を構築して検証を行えるようにする。今回の研究では少ない仮想ルータしか作成していないがjailの機能を使えば、メモリにもよるが1000個ぐらいの仮想ルータを作成できるので今後はより大規模な仮想環境の構築が課題となってくると考えられる。

謝辞

本研究を行なうにあたり，終始熱心に御指導していただいた木下宏揚教授、南出 和宏氏に心から感謝致します。良き研究生活を送らせていただいた木下研究室の方々に深く感謝致します.

2012 年 2 月

末田　雄己

参考文献

[1] ”The FreeBSD Project” http://www.freebsd.org/ja/

[2] ”FreeBSD とは” http://e-words.jp/w/FreeBSD.html

[3] ”FreeBBSD wikipedia” http://ja.wikipedia.org/wiki/FreeBSD

[4] ”FreeBSD jail wikipedia” http://ja.wikipedia.org/wiki/FreeBSD_jail

[5] ”hasegaw blog” http://d.ballade.jp/blog/2011/01/freebsd-vimage.html

[6] 卒業研究　小川真人　”n チャネルメッセージ伝送方式”

[7] ”FreeBSD サーバ上に仮想サーバを構築する（jail）” http://www.kishiro.com/FreeBSD/jail.html

[8] ”マイナビニュース FreeBSD 新機能” http://news.mynavi.jp/articles/2009/11/24/freebsd80/index.html

[9] ”massi’s easy laboratory” http://www.massi.mydns.jp/massis_easy_laboratory/2009/10/jail.h

[10] ”jail manual” http://www.nxmnpg.com/ja/8/jail

[11] ”仮想化 wikipedia” http://ja.wikipedia.org/wiki/

[12] ”jail と vimage について” http://www.nisoc.or.jp/ ishimoto/s-tudy/20091205/jail_vimage.pdf

[13] DANNY DOLEV、CYNTHIA DWORK、ORLI WAARTS、MOTI YUNG ”Perfectly Secure Message Transmission”
Journal of the Association for Computing Machinery、Vol.40,No.1、
pp.17-47(1993)

[14] K. Srinathan、Arvind Narayanan、C. Pandu Rangan ”Optimal Perfectly Secure Message Transmission”
CRYPTO 2004、LNCS 3152、pp.545-561(2004)

[15] Kaoru KUROSAWA、Kazuhiro SUZUKI、Members ”Almost Secure (1-Round,n-Channel) Message Transmission Scheme”
IEICE TRANS. FUNDAMENTALS、VOL.E92-A,NO.1(2009)

[16] 笠原正雄、佐竹賢治：”誤り訂正符号と暗号の基礎数理”、コロナ社 (2004)

[17] 澤田秀樹：”暗号理論と代数学”、海文堂 (1997)

[18] ”整数の合同” http://www2.cc.niigata-u.ac.jp/ takeuchi/tbasic/BackGround/-Cong.html

[19] 三谷政昭、佐藤伸一、ひのきいでろう：”マンガでわかる暗号”、オーム社 (2007)

[20] 黒澤馨、尾形わかは：”現代暗号の基礎数理”、コロナ社 (2004)

[21] 今井秀樹：”情報・符号・暗号の理論”、コロナ社 (2004)

[22] Douglas R. Stinson、櫻井幸一：”暗号理論の基礎”、共立出版株式会社 (1996)

[23] DOUGLAS R. STINSON：”CRYPTOGRAPHY THEORY AND PRACTICE THIRD EDITION”、Chapman & Hall/CRC(2006)

[24] HASAN MD. SAYEED、HOSAME ABU-AMARA ”Efficient Perfectly Secure Message Transmission in Synchronous Networks”

INFORMATION AND COMPUTATION 126、pp.53-61(1996)、ARTICLE NO.0033

[25] Saurabh Agarwal、Ronald Cramer、Robbert de Haan ”Asymptotically Optimal Two-Round Perfectly Secure Message Transmission”
CRYPTO 2006、LNCS 4117、pp.394-408(2006)

[26] Kaoru Kurosawa、Kazuhiro Suzuki ”Truly Efficient 2-Round Perfectly Secure Message Transmission Scheme”
Advances in Cryptology、EUROCRYPT 2008 LNCS 4965、pp.324-340(2008)

[27] ”as3crypto” http://code.google.com/p/as3crypto/

[28] W. リチャード・スティーヴンス、篠田陽一：
”UNIX ネットワークプログラミング　第 2 版　Vol.1　ネットワーク API: ソケットと XTI”
株式会社ピアソン・エデュケーション（1999）

[29] . リチャード・スティーヴンス、橘康雄、井上尚司：”[新装版] 詳解 TCP/IP　Vol.1 プロトコル”
株式会社ピアソン・エデュケーション（2000）

[30] . リチャード・スティーヴンス、徳田　英幸、戸辺義人：”[新装版] 詳解 TCP/IP　Vol.2 プロトコル”
株式会社ピアソン・エデュケーション（2002）

[31] きみち： ”基礎と実践 Linux ネットワークプログラミング”
ソフトバンククリエイティブ株式会社（2010）

[32] ichael J.Donahoo、Kenneth L. Calvert、小高知宏：”TCO/IP ソケットプログラミング　C 言語編”
オーム社（2003）

[33] 松公保：”基礎からわかるTCP/IPネットワーク実験プログラミング　第2版”
オーム社（2004）

[34] 井康孝：”猫でもわかるネットワークプログラミング　第2版”
ソフトバンククリエイティブ株式会社（2006）

質疑応答

質問：jailで4つでやるっているが、700個作成しようとしたときの障害は何か？
答え：経路制御を4つでやった理由は、3年のころの実験でもあったように経路制御の挙動を見るのに4つあれば足りるから手始めに4つ作成した。
700個作る時の障害は作成するだけならメモリの容量ぐらいしか障害にならないと考えられる。しかし、本実験では設定を自動化しておらずすべて手動で行っているので、700個手動で行うことは現実的ではないのでそこを自動化することが必要になってくると考えられる。