L-016

Linux 仮想ネットワークスタックを用いたプログラマブルルータの試作

Prototype of programmable router using Linux virtual network stack

馬場 隆章　Takaaki Baba
後藤 邦夫　Kunio Goto

1. はじめに

近年の急速なインターネットの進展に伴い, 各ネットワーク上のノードを仮想化し,ユーザがネットワークを自由にカスタマイズできる「アクティブネットワーク」(以下, AN)[1]という概念が提唱された.

ANは,「ルータがアプリケーション層までの処理を実行することを許すネットワーク」また,「ユーザがプログラムを注入することで, ネットワークをプログラムすることが可能である」という特徴を持っている.

ANの概念が提案された技術的背景として, プロトコル仕様変更への迅速な対応を可能とする技術が望まれていたこと, すでにネットワーク内でネットワーク層以上の簡単な処理を実行するものとして, ファイアウォール, Web Proxyなどの技術が先行して導入されていること, またJavaなどのセキュリティを保証しつつプログラムの可搬性を提供するソフトウェア技術の進展などが挙げられる.

しかし, ANを導入するためにはルータをAN対応のものに置き換える必要がある. だがAN機能が全ルータに導入されるには相当の期間を要する. そこで本稿ではLinuxカーネル2.6.29で使用可能となった, Network Namespace(以下, NETNS)[2]をホストの仮想化に, libpcap[3]をパケット横取りに利用し, 複数のプログラマブルルータを1台のホストでエミュレーションすることにより, プログラマブルルータの容易な実験環境を構築する. なおエミュレーションには南山大学後藤邦夫教授考案のGINE[4]を使用した.

2. アクティブネットワーク

ANは, それぞれのユーザ(エンドユーザやネットワーク管理者)がネットワークの中の通信機器(ルータやサーバ)で動作する通信処理ソフトウェアを, 自由に操作することができる. つまり, ユーザの思うがままにネットワーク自体を変化させることができるネットワークの事である.

ANを実現するための基本アーキテクチャとして, 2つの方式が提案されている. それらはプログラマブルスイッチ方式とカプセル方式である.

プログラマブルスイッチ方式はネットワーク内で実行すべき処理プログラムを事前にルータに注入しておく.

ネットワーク内で処理を必要とするパケットにはどのプログラムを実行すべきかを識別するための識別子を格納し, パケット到着時にこの識別子で示されたプログラムに従って処理を実行する方式である.

カプセル方式はネットワーク内で実行すべき処理プログラムを各パケットに内蔵して転送する. ルータにおいては共通処理系が用意されており, パケットに内蔵されてきたプログラムにしたがって処理を実行する.

カプセル方式では, 処理内容をすべてプログラムとしてパケットに格納する必要があるために用いられる言語も低レベルのものに限られ, さらに処理できる内容もその言語を用いてパケットに収容できるサイズのものに制約される. これに対してプログラマブルスイッチ方式では, 事前に処理内容を示すプログラムをルータに注入しておくのである程度の高機能言語の使用が可能であり, また処理内容もある程度複雑な内容のものが可能である.

本稿ではプログラマブルスイッチ方式を採用したルータをプログラマブルルータと呼び, 1台のホストでNETNSによりネットワークスタックの仮想化を行い, 複数のプログラマブルルータのエミュレーションする.

3. Network Namespace

NETNSはLinuxカーネル2.6.29以降標準搭載されたネットワークスタック(ネットワーク空間)のみを仮想化する機能である.

仮想ネットワークスタックでは生成時, ループバックを含むネットワークスタックインターフェースを共有することができない. そこで特殊な仮想ネットワークデバイスであるVirtual Ethernet Pair(以下veth)を用いる. このデバイスはペアデバイスになっており, 片方のデバイスでパケットを受信するともう片方のデバイスに転送する.

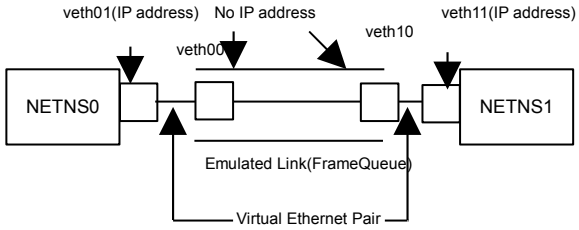

図1: NETNSによるホストでの通信方式

図1のようにveth01, veth11はNETNSに渡し, IPアドレスを付与しているが, GINEを起動しているホスト側のveth00, veth10はIPアドレスを付与していない. 付与してしまうと, ホストの外部あるいはホスト自体からのパケットがインタフェースへ直接転送されてしまう. したがって, パケットはエミュレートを回避し, エミュレーションできなくなってしまう.

そこで各デバイスをそれぞれNETNSに引き渡すことで, お互いのNETNS間で通信可能となる. また, 作成したNETNSはそれを機動したプロセスとその子プロセスで有効なので, 端末を機動すればコマンド操作が可能である.

さらにネットワークスタックのみの仮想化ではメモリを消費しないため, 大規模なネットワークトポロジを構築してもメモリの消費を抑えることができる. このNETNSの機能を用いネットワークスタックのみを仮想化し, そのホストを複数起動させることでネットワークを構築することができる. 構成したネットワークのうちいくつかのホ

ストをプログラマブルルータとすることでエミュレーションする.

4．プログラマブルルータの実現方法

プログラマブルルータには事前にプログラムを注入しておく必要があり, プログラムを外部からルータに注入する仕組みが必要となる. 方法としてはコンパイル済みのプログラムを SSL で送るなどの方法が考えられる.

Java ならプラットフォーム互換性があるため比較的容易にプログラムの差し替えなどが出来き, オブジェクト指向言語ではルータのプログラムをオブジェクトとして保存し, ネットワークで送ることができる.

我々は C++でプログラムを作成し, GNU commoncpp ライブラリに実装されている STL Engine をオブジェクトの保存, 読み出しを用いる.

これらのプログラマブルな処理を実行するためには通常のパケットの処理を行う前に, 一度パケットを横取りして処理を施す必要がある. パケットの横取り方法としては divert socket, NFQUEUE などが考えられる.

IPv4/IPv6 で一般的に使えるのは netfilter NFQUEUE だが, 残念ながら NETNS に対応していない.

Divert socket は FreeBSD が IPv4 だけ対応しており, それ以外は我々が開発したカスタムカーネルが必要になり使いづらい.

そこで NETNS 上でのエミュレーションには IPv6 にも対応している Unix 系 OS でのパケットキャプチャ用ライブラリの libcap を使用した.

必要なパケットをキャプチャすることにより, netfilter (iptables/ip6tables) でキャプチャしたものを捨てることで横取り機能を実現させた.

5．エミュレーションの実験例

ここでのプログラマブルルータに行わせるプログラマブルな処理は以下の通りである.

- パケットのアドレス変換
- サーバの負荷情報の取得

本実験では AN をサーバ負荷分散に応用し, ルータに最適サーバ選択処理を行わせるアクティブエニーキャスト[5]を IPv6 環境で実験した.

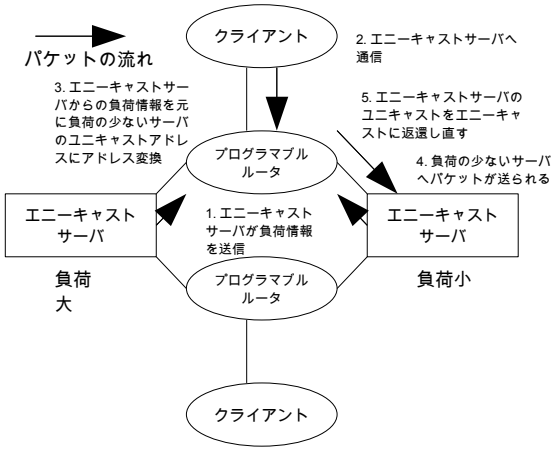

図 2:ネットワーク構成

エニーキャストとは IPv6 に移行するにあたり新しく実装されたアドレスのことで, 一部 IPv4 にも使われている.

エニーキャストはインターネット上の複数のノードに同一の IP アドレスを割り当てたもので, 普通対応するサーバ群の中から最近隣サーバがルータにおいて選択され, 該当パケットは最近隣サーバへの出方路へと送出される. これに対してアクティブエニーキャストにおいては, 図 2 の 1 ようにエニーキャストサーバがプログラマブルルータへサーバの負荷情報を送り, それを元にプログラマブルルータは最適サーバを選択する.

エニーキャストアドレスを受信したプログラマブルルータは 2 のようにエニーキャスト宛ての通信が来た場合, サーバの負荷情報をもとに最適サーバを選択する.

そして 3 のように最適サーバ(負荷が少ないサーバ)へのアドレスに変換してパケットを送信し, 4 でエニーキャストサーバがユニキャストに返還されたパケットを受け取る.

最後の 5 でエニーキャストサーバのユニキャストのパケットを, プログラマブルルータがエニーキャストに戻すことにより, クライアントは負荷の少ないサーバと通信することができる.

サーバの負荷情報を各エンドユーザが独自に把握しようとすると, 全エンドユーザが複数のサーバに対して負荷情報を獲得する必要があるため大量の負荷情報がネットワーク内を流れることとなる. これに対してアクティブエニーキャストではネットワーク内のルータが負荷情報を獲得するので, ネットワークを流れる負荷情報は全体として少なくて済む.

6．おわりに

本研究では NETNS の機能を用いネットワークスタックのみを仮想化し, そのホストを複数起動させることでネットワークを構築することができた. さらに構成したネットワークのホストをルータとすることでプログラマブルルータのエミュレータによる実験環境を整える事が可能となった. これにより AN の実験が容易なものとなり, AN のアーキテクチャの開発がスムーズに行えるようになった.

7．参考文献

[1] Konstantinos Psounis : “Active Networks: Applications, Security, Safety and Architectures”, IEEE Communications Surveys, First Quarter 1999.

[2] Network Namespace: Linux Containers, http://lxc.sourceforge.net/network.php (accessed Apr.2009).

[3] Van Jacobson, Craig Leres and Steven McCanne, all of the Lawrence Berkeley National Laboratory, University of California, Berkeley, CA : tcpdump/libpcap. http://www.tcpdump.org/ (accessed Apr. 2009).

[4] Sugiyama Yusuke , Goto Kunio :Design and Implementation of a Network Emulator using Virtual Network Stack, Proc. Of the Seventh International Symposium on Operations Research and Its Applications (ISORA2008), Lecture Note in Operations Research, Vol8, pp.351-358 (2008).

[5] 三浦 浩一, 西村 健治, 山本 幹, 池田博昌 : “Active Network 技術を用いたサーバ負荷分散方式”, 1999 年テレコミュニケーションマネージメントワークショップ TMWS 99-9, March 1999.