Linux IPv6 HOWTO (pt_BR)

Peter Bieringer

pb at bieringer dot de

Linux IPv6 HOWTO (pt_BR)

by Peter Bieringer

A meta deste HOWTO de IPv6 em Linux é responder as questões básicas e avançadas sobre a versão 6 do protocolo IP em um sistema com Linux. Este HOWTO dará ao leitor informação suficiente para instalar, configurar e usar aplicações IPv6 em máquinas com o Linux. Versões intermediárias deste HOWTO estão disponíveis nos endereços mirrors.bieringer.de (http://mirrors.bieringer.de/Linux+IPv6-HOWTO/) ou mirrors.deepspace6.net (http://mirrors.deepspace6.net/Linux+IPv6-HOWTO/). Veja também revision history para saber das mudanças.

Revision History

Revision 0.66wip 2010-04-20 Revised by: PB

Revision 0.65 2009-12-13 Revised by: PB

Revision 0.64 2009-06-11 Revised by: PB

Revision 0.60 2007-05-31 Revised by: PB

Revision 0.51 2006-11-08 Revised by: PB

Table of Contents

Chapter 1. Geral

As informações sobre as traduções disponíveis estão na seçãoTranslations.

1.1. Copyright, licença e outros

1.1.1. Copyright

Escrito e com Copyright (C) 2001-2011 por Peter Bieringer

1.1.2. Licença

Este Linux IPv6 HOWTO está publicado sob a licença GNU GPL versão 2:

Linux IPv6 HOWTO, um guia para configurar e usar o IPv6 em sistemas Linux.

Copyright © 2001-2011 Peter Bieringer

Este documento é um software livre; voce pode redistribui-lo e/ou modifica-lo sob os termos da licença GNU GPL, tal como está publicado pela Free Software Foundation; seja pela versão 2 da Licença, ou (em sua opinião) qualquer versão posterior.

Este programa é distribuído na esperança de que seja útil, mas SEM QUALQUER GARANTIA; nem mesmo qualquer garantia de COMERCIALIZAÇÃO ou ADEQUAÇÃO PARA UM PROPÓSITO PARTICULAR. Veja a GNU GPL para mais detalhes.

Se voce quiser uma cópia da licença GNU GPL, solicite através de carta para o endereço Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110, USA.

1.1.3. Sobre o autor

1.1.3.1. História do autor com a Internet e IPv6

- 1993: Eu entrei em contato com a internet utilizando um cliente de email e news baseado em console há bastante tempo (procure por "e91abier" no grupo groups.google.com (http://groups.google.com/), sou eu).
- 1996: Foi solicitado que eu produzisse um curso de IPv6, incluindo um workshop com o Sistema Operacional Linux.
- 1997: Comecei escrevendo um guia sobre como instalar, configurar e utilizar o IPv6 em Linux, chamadoIPv6 & Linux - HowTo (http://www.bieringer.de/linux/IPv6/) (veja IPv6 & Linux - HowTo/History (http://www.bieringer.de/linux/IPv6/IPv6-HOWTO/IPv6-HOWTO-0.html#history) para mais informações).
- 2001: Comecei a escrever este novo Linux IPv6 HOWTO.

1.1.3.2. Contato

O autor pode ser contactado através do endereço de email <pb at bieringer dot de> e também através de suahomepage (http://www.bieringer.de/pb/).

Atualmente ele mora em Munique [parte nordeste de Schwabing] / Bavaria / Alemanha (sul) / Europa (centro) / Terra (superfície).

1.2. Categoria

Este HOWTO deve ser listado na categoria "Networking/Protocols".

1.3. Versão, Histórico e To-Do

1.3.1. Versão

A versão atual deste documento é mostrada no começo do documento.

Informação CVS:

```
CVS-ID: $Id: Linux+IPv6-HOWTO.lyx,v 1.119 2006/11/08 22:45:25 pbldp Exp $
```

Para outras versões ou traduções disponíveis, veja o sitehttp://www.bieringer.de/linux/IPv6/.

1.3.2. Histórico

1.3.2.1. Relevantes

2001-11-30: Começo do estilo do novo HOWTO.

2002-01-02: Maioria do conteúdo feito, primeira publicação do capítulo 1(versão 0.10).

2002-01-14: Mais conteúdo, algumas revisões, e publicação do documento completo (version 0.14).

2002-08-16: Tradução para o Polones em progresso

2002-10-31: Tradução para o Chines disponível (veja Translations para mais informações)

2002-11-10: Tradução para o Alemão em progresso

2003-02-10: Tradução para o Alemão disponível

2003-04-09: Tradução para o Frances em progresso

2003-05-09: Tradução para o Frances disponível

2003-10-16: Tradução para o Italiano em progresso

2004-03-12: Tradução para o Italiano disponível

2004-06-18: Tradução para o Grego em progresso

2005-07-25: Tradução para o Turco disponível

2007-03-28: Tradução para o Portugues-Brazil em progresso

2008-07-30: Tradução para o Espanhol disponível (mas ainda em progresso)

1.3.2.2. História completa

Veja revision history no final deste documento

1.3.3. To-Do

- Completar conteúdo que ainda falta
- Finalizar a checagem gramatical

1.4. Traduções

As traduções devem sempre conter a URL, número da versão c copyright do documento original (e o seu também). Por favor não traduza o changelog original, pois isto não terá muita importancia. Também não traduza a seção sobre traduções disponíveis, pois elas podem estar desatualizadas. Ao invés disso, adicione uma URL para a seção em Ingles deste HOWTO.

Aparentemente a frequencia de mudança deste documento ocorre menos de uma vez por mes. Desde a versão 0.2.7 parece que a maioria do conteúdo que eu fiz foi escrito. As traduções devem sempre ter a versão em ingles como original..

1.4.1. Linguagens

Nota: uma olhada nesta URL pode ajudarhttp://www.bieringer.de/linux/IPv6/.

1.4.1.1. Chines

A tradução para o Chines, feito por Burma Chen <expns at yahoo dot com> (informada a mim em 2002-10-31) e pode ser encontrada no TLDP:http://www.ibiblio.org/pub/Linux/docs/HOWTO/translations/zh/Linux-IPv6-HOWTO.txt.gz (g'zipped txt) (http://www.ibiblio.org/pub/Linux/docs/HOWTO/translations/zh/Linux-IPv6-HOWTO.txt.gz). É uma foto da tradução, eu não sei se está atualizada.

1.4.1.2. Polones

Desde 2002-08-16 a tradução para o Polones foi iniciada e ainda está em progresso por Lukasz Jokiel <Lukasz dot Jokiel at klonex dot com dot pl>. Versão usada: CVS-version 1.29 do arquivo LyX, a qual foi a origem para a versão 0.2.7. O status é de ainda em progresso (2004-08-30).

1.4.1.3. German

Em 2002-11-10 a versão Alemã foi iniciada por Georg Käfer <gkaefer at gmx dot at> iniciou a tradução para o alemão, e a primeira publicação foi feita em 2003-02-10. Ela está originalmenmte disponível na Deep Space 6http://mirrors.deepspace6.net/Linux+IPv6-HOWTO-de/ (e também em http://mirrors.bieringer.de/Linux+IPv6-HOWTO-de/). Esta versão se manterá atualizada tanto quanto for possível.

1.4.1.4. Frances

Em 2003-04-09 foi iniciada a tradução da versão em Frances, por Michel Boucey <mboucey at free dot fr> e a primeira publicação foi em 2003-05-09. Ela está originalmente disponível na Deep Space 6http://mirrors.deepspace6.net/Linux+IPv6-HOWTO-fr/ (e também em http://mirrors.bieringer.de/Linux+IPv6-HOWTO-fr/).

1.4.1.5. Espanhol

Um membro do projeto MontevideoLibre, localizado no Uruguai (América do Sul) iniciou a tradução para espanhol no formato wiki:http://www.montevideolibre.org./manuales:libros:ipv6

1.4.1.6. Italiano

Em 2003-10-16 a tradução para uma versão em Italiano foi iniciada Michele Ferritto <m dot ferritto at virgilio dot it> para oILDP (http://ildp.pluto.linux.it/) (Italian Linux Documentation Project) e a sua primeira publicação foi em 2004-03-12. Ela está originalmente disponível no ILDP em http://it.tldp.org/HOWTO/Linux+IPv6-HOWTO/.

1.4.1.7. Japones

Em 2003-05-14 Shino Taketani <shino_1305 at hotmail dot com> me enviou uma nota dizendo que planejava traduzir o HOWTO para o japones.

1.4.1.8. Grego

Em 2004-06-18 Nikolaos Tsarmpopoulos <ntsarb at uth dot gr> me enviou uma nota dizendo que planejava traduzir o HOWTO para o Grego.

1.4.1.9. Turco

Em 2005-07-18 Necdet Yucel <nyucel at comu dot edu dot tr> me enviou uma nota dizendo que a tradução em Turco estava disponível. E uma fotografia da tradução (versão 0.61) pode ser encontrada na URLhttp://docs.comu.edu.tr/howto/ipv6-howto.html.

1.4.1.10. Portuguese-Brazil

Em 2011-05-06 Gustavo Mendes de Carvalho <gmcarvalho at gmail dot com> iniciou a tradução deste HowTo para Portugues-Brazil. A primeira tentativa realizada em 2007 por Claudemir da Luz <claudemir dot daluz at virtuallink dot com dot br> nunca foi finalizada.

1.5. Técnico

1.5.1. Fonte original deste HOWTO

Este HOWTO foi escrito usando LyX versão 1.6.1 em um sistema Linux Fedora 10 com o template SGML/XML (DocBook). Ele está disponível emTLDP-CVS / users / Peter-Bieringer (http://cvs.tldp.org/go.to/LDP/LDP/users/Peter-Bieringer/) para contribuições.

1.5.1.1. Divisor de linha de código

Eu utilizei um utilitário divisor de linha de código (Code line wrapping - "lyxcodelinewrapper.pl") feito por mim mesmo, e ele está disponível para seu próprio uso emTLDP-CVS / users / Peter-Bieringer (http://cvs.tldp.org/go.to/LDP/LDP/users/Peter-Bieringer/)

1.5.1.2. Geração de SGML

O SGML/XML é gerado usando a função de exportar do LyX

1.5.2. Referencias On-line para a versão em HTML deste HOWTO

1.5.2.1. Página principal

Como boa prática, uma referencia à página principal deste HOWTO é recomendada.

1.5.2.2. Páginas dedicadas

Como as páginas em HTML são geradas a partir dos arquivos SGML, os nomes dos arquivos em HTML podem ser bastante diferentes (randomicos). Entretanto, algumas páginas são etiquetadas em LyX, resultando em nomes estáticos. Estas etiquetas são úteis para referencias e não deveriam ser alteradas no futuro. Se voce acredita que eu esqueci qualquer etiqueta, por favor me avise, e eu colocarei.

1.6. Prefácio

Algumas coisas antes:

1.6.1. Quantas versões deste HOWTO existem por aí ?

Incluindo esta, existem 3 (tres) documentos HOWTO disponíveis. Minhas desculpas se forem demais ;-)

1.6.1.1. Linux IPv6 FAQ/HOWTO (desatualizada)

O primeiro documento HOWTO relacionado a IPv6 foi escrito pelo Eric Osborne e era chamadoLinux IPv6 FAQ/HOWTO (http://www.linuxhq.com/IPv6/) (por favor use-o somente para fins históricos). Sua última versão foi a 3.2.1, lançada em 14 de Julho de 1997.

Por favor ajude: Se alguém souber a data de nascimento deste HOWTO, por favor me envie um e-mail (estas informações são necessárias para o "histórico").

1.6.1.2. IPv6 & Linux - HowTo (mantida)

Esta segunda versão existe e se chamaIPv6 & Linux - HowTo (http://www.bieringer.de/linux/IPv6/) escrita por mim (Peter Bieringer) em HTML puro. Ele nasceu em Abril de 1997 e sua primeira versão em ingles foi publicada em Junho de 1997. Eu vou continuar a manter esta versão, mas isto deve acontecer devagar e não será por completo, em favor da atualização da versão do Linux IPv6 HOWTO que voce está lendo agora.

1.6.1.3. Linux IPv6 HOWTO (este documento)

Já que esteIPv6 & Linux - HowTo (http://www.bieringer.de/linux/IPv6/) foi escrito em HTML puro, ele não é compatível com o The Linux Documentation Project (TLDP) (http://www.tldp.org/). Eu recebi então um pedido no final de Novembro de 2001 para reescrever este HowTo IPv6 & Linux - HowTo (http://www.bieringer.de/linux/IPv6/) em SGML. Entretanto, por causa da descontinuidade do HOWTO (Future of IPv6 & Linux - HowTo (http://www.bieringer.de/linux/IPv6/IPv6-HOWTO/IPv6-HOWTO-0.html#history)), e como o IPv6 estava ficando mais e mais padronizado, eu decidi escrever um novo documento cobrindo os pontos básicos e um pouco avançados que permaneceram importantes ao longo destes anos. Algum conteúdo mais dinamico e avançado ainda pode ser encontrado neste segundo HOWTO (IPv6 & Linux - HowTo (http://www.bieringer.de/linux/IPv6/)).

1.7. Termos usados, glossário e atalhos

1.7.1. Rede

Base 10

: Sistema numérico muito bem conhecido e representa qualquer valor com os dígitos 0 a 9.

Base 16

: Geralmente usado em linguagens de programação, também conhecido como sistema numérico hexadecimal, representa qualquer valor com os dígitos 0 a 9 e caracteres A a F (case insensitive).

Base 85

: Representação de um valor com 85 dígitos/caracteres diferentes, ele pode levar a strings menores, mas nunca vi ser usado em campo.

Bit

: A menor unidade de armazenamento, representa on/verdade/1 e off/falso/0.

Byte

Geralmente uma coleção de 8 bits (mas não é necessariamente uma verdade - veja outros sistemas computacionais)

Device

Aqui, o hardware para a conexão de rede, veja também NIC

Dual homed host

Um sistema dual homed é um nó com duas redes (física ou virtual) com interfaces em dois links diferentes, mas sem encaminhar qualquer pacote entre eles (não é um router).

Host

Geralmente um sistema single homed com somente uma interface de rede ativa, exemplo Ethernet ou (não e) PPP.

Interface

Quase sempre o mesmo que "device", veja também NIC

IP Header

Cabeçalho de um pacote IP (cada pacote de rede tem um cabeçalho e seu tipo depende do nível de rede)

Link

Um link é o nível 2 de rede, ou meio de transporte de pacotes. exemplos são Ethernet, Token Ring, PPP, SLIP, ATM, ISDN, Frame Relay,...

Node

Um nó é um host ou um router.

Octet

É uma coleção de 8 bits, hoje bem similar a "byte".

Port

Informação utilizada pelo TCP/UDP dispatcher (camada 4) para transportar informações para as camadas superiores

Protocolo

Cada camada de rede que contém a maioria dos campos e informações para tornar a vida mais fácil ao enviar a informação transportada para as camadas superiores, veja camada 2 (MAC) e 3 (IP)

Router

Um router é um nó com 2 ou mais redes interfaces de rede (física ou virtual), capaz de encaminhar os pacotes entre as suas interfaces.

Socket

Um socket IP é definido pelo endereço de origem e destino, e suas portas

Stack

Relacionado às várias camadas de rede

Subnetmask

Redes IP usam bits de máscara para separar redes locais de redes remotas

Tunnel

Um túnel é tipicamente uma conexão ponto-a-ponto sobre a qual pacotes são trocados, e que carregam dados de outro protocolo. Exemplo túnel IPv6-in-IPv4.

1.7.1.1. Atalhos

ACL

Access Control List - Lista de controle de acesso

API

Application Programming Interface - Interface de programação de aplicação

ASIC

Application Specified Integrated Circuit

BSD

Berkeley Software Distribution

CAN-Bus

Controller Area Network Bus (physical bus system)

ISP

Internet Service Provider - Provedor de serviços Internet

KAME

Projeto - um esforço conjunto de seis companhias no Japão para fornecer grátis uma pilha IPv6 e IPsec (para ambos IPv4 e IPv6) para as variantes BSD existentes no mundo www.kame.net (http://www.kame.net/)

LIR

Local Internet Registry - No Brasil, o registro.br

NIC

Network Interface Card

RFC

Request For Comments - conjunto de notas técnicas organizacionais sobre a Internet

USAGI

UniverSAl playGround for Ipv6 Project - trabalho para entregar uma pilha IPv6 de qualidade para os sistemas Linux.

1.7.2. Informações úteis

1.7.2.1. Sinal de divisão de linha longa de código

O caractere especial "¬" é usado para sinalizar que esta linha de código foi dividida para se obter uma melhor visualização em arquivos PFG e PS.

1.7.2.2. Marcadores

Nos exemplos genéricos voce encontrará as seguintes marcações:

```
<myipaddress>
```

Para o uso real em seu sistema de linha de comando ou em scripts, isto deve ser substituído pelo conteúdo correto (removendo os sinais < e >). Desta forma o resultado seria

```
1.2.3.4
```

1.7.2.3. Comandos no shell

Comandos executados no shell por usuários normais (não root) começam com $

```
$ whoami
```

Comandos executados pelo usuário root começam com #

```
# whoami
```

1.8. Necessidades para usar este HOWTO

1.8.1. Prerequisitos pessoais

1.8.1.1. Experiencia com ferramentas Unix

Voce deve estar familiarizado com a maioria das ferramentas Unix, como grep, awk, find, etc. , e saber sobre a maioria das opções mais usadas de cada um deles.

1.8.1.2. Experiencia com teoria de rede

Voce deve estar famliarizado com camadas, protocolos, endereços, cabos, conectores, etc. Se voce é novo nesta área, este é um bom local para voce iniciar seus estudoshttp://www.rigacci.org/docs/biblio/online/intro_to_networking/book1.htm

1.8.1.3. Experiencia com configuração IPv4

Voce deve definitivamente ter alguma experiencia em configuração de redes IPv4, caso contrário será difícil para voce entender o que realmente está acontecendo.

1.8.1.4. Experiencia com Domain Name System (DNS)

Permitirá a voce entender o que é um Domain Name System (DNS), que serviço ele fornece e como usa-lo.

1.8.1.5. Experiencia com estratégias de debug de rede

Voce deve pelo menos entender como usar o tcpdump e o que ele pode te mostrar. Caso contrário a depuração de problemas de rede será muito difícil para voce.

1.8.2. Hardware compatível com o sistema Linux

É claro que voce vai precisar usar algum hardware (pode ser uma maquina virtual), e não somente ler este HOWTO para dormir. ;-7)

Chapter 2. Básico

2.1. O que é IPv6?

O IPv6 é um novo protocolo de camada 3 que tem a função de substituir o IPv4 (também conhecido apenas por IP). O IPv4 foi projetado a muito tempo atrás(RFC 760 / Internet Protocol (http://www.faqs.org/rfcs/rfc760.html) de Janeiro de 1980) e desde o começo tem havido muitos pedidos de atender mais capacidades e funcionalidades. A última RFC é RFC 2460 / Internet Protocol Version 6 Specification (http://www.faqs.org/rfcs/rfc2460.html).As grandes mudanças no IPv6 foram o novo formato do cabeçalho, incluindo o tamanho da capacidade de endereços, de 32 para 128 bits. Já que a camada 3 é a responsável por transporte de pacotes fim a fim usando o roteamento baseado em endereços, ele deveria incluir os endereços IPv6 de origem e destino tal como o IPv4.

Para mais informações sobre a história do IPv6, de uma olhada nas RFC's mais antigas do IPv6 listadas aquiSWITCH IPv6 Pilot / References (http://www.switch.ch/lan/ipv6/references.html).

2.2. História do IPv6 no Linux

Os anos de 1992, 1993 e 1994 do IPv6 no Linux (linhas gerais) são cobertos pelo seguinte documento:IPv6 or IPng (IP next generation) (http://www.laynetworks.com/IPv6.htm#CH3).

To-do: melhorar a linha do tempo, adicionar conteúdo...

2.2.1. O começo

O primeiro trecho de código de rede relacionado com o IPv6 foi adicionado ao kernel 2.1.8 do Linux em novembro de 1996 por Pedro Roque. Ele foi baseado na API do BSD:

```
diff -u --recursive --new-file v2.1.7/linux/include/linux/in6.h
¬ linux/include/linux/in6.h
--- v2.1.7/linux/include/linux/in6.h Thu Jan 1 02:00:00 1970
+++ linux/include/linux/in6.h Sun Nov 3 11:04:42 1996
@@ -0,0 +1,99 @@
+/*
+ * Types and definitions for AF_INET6
+ * Linux INET6 implementation
+ * + * Authors:
+ * Pedro Roque <******>
+ *
+ * Source:
+ * IPv6 Program Interfaces for BSD Systems
+ * <draft-ietf-ipngwg-bsd-api-05.txt>
```

As linhas mostradas foram copiadas do patch-2.1.8 (o email foi limpo para evitar spam).

2.2.2. Enquanto isso

Por conta do buraco da manpower, a implementação do IPv6 no kernel foi incapaz de seguir os rascunhos propostos pelos novos RFC's publicados. Em outubro de 2000, um projeto foi iniciado no Japão, chamadoUSAGI (http://www.linux-ipv6.org/), cujo objetivo foi implantar todo o restante, ou desatualizado, suporte ao IPv6 para o

Linux. Ele utiliza a implementação IPv6 atual para o FreeBSD feita pelo KAME project (http://www.kame.net/). De tempos em tempos, eles criavam fotos da versão vanilla do código do kernel do Linux.

Até a implementação do desenvolvimento na série 2.5 do kernel ter sido iniciado, os patches doUSAGI (http://www.linux-ipv6.org/) eram tão grandes, que os mantenedores de rede do Linux eram incapazes de inclui-lo completamente no código final do kernel do Linux série 2.4.x.

Durante o desenvolvimento da serie 2.5, o tentou inserir todas as suas extensões usadas nesta série.

2.2.3. Atualmente

Muito do desenvolvimento feito para o IPv6 e patches doUSAGI (http://www.linux-ipv6.org/) e outros estão integrados na série vanilla do kernel 2.6.x.

2.2.4. O futuro

O USAGI (http://www.linux-ipv6.org/) e outros ainda mantém o trabalho na implementação de novas características e funcionalidades, como a mobilidade e outros. De tempos em tempos, novos patches com extensões são lançados e também integrados à série vanilla do kernel.

2.3. Como o endereço IPv6 se parece?

Como já mencionado antes, os endereços IPv6 possuem 128 bits de tamanho. Este número de bits gera um número decimal extremamente grande, com 39 dígitos de tamanho:

```
2^128-1: 340282366920938463463374607431768211455
```

Tais números não são endereços fáceis de serem memorizados. Os endereços IPv6 também tem um esquema orientado a bits (assim como o IPv4, mas não tão facilmente reconhecido). Assim a melhor notação de números tão grandes é em formato hexadecimal. Em hexadecimal, 4 bits (também conhecidos como "nibble") são representados por um dígito ou caractere, de 0-9 e A-F. Desta forma, o tamanho do endereço é reduzido para 32 caracteres.

```
2^128-1: 0xffffffffffffffffffffffffffffffff
```

Esta representação ainda não é muito conveniente (com a possível mistura ou perda de um único dígito hexadecimal), então os desenvolvedores do IPv6 escolheram um formato hexadecimal com um ":" separando cada bloco de 16 bits. Com isso, o sinal inicial 0x (um prefixo para valores hexadecimais em linguagens de programação) foi removido:

```
2^128-1: ffff:ffff:ffff:ffff:ffff:ffff:ffff:ffff
```

Um endereço utilizável seria:

```
2001:0db8:0100:f101:0210:a4ff:fee3:9566
```

Para simplificar, os zeros iniciais de cada bloco de 16 bits pode ser omitido:

```
2001:0db8:0100:f101:0210:a4ff:fee3:9566  ->
¬ 2001:db8:100:f101:210:a4ff:fee3:9566
```

Um bloco de 16 bits contendo somente zeros também pode ser omitida, sendo representada por "::", mas não mais de uma única vez no endereço. Caso contrário poderia haver duplicação de endereços.

```
2001:0db8:100:f101:0:0:0:1  ->  2001:db8:100:f101::1
```

A maior redução possível é vista do endereço IPv6 de localhost:

```
0000:0000:0000:0000:0000:0000:0000:0001  ->  ::1
```

Há também um outro representação em modo compacto (Código base 85) baseado naRFC 1924 / A Compact Representation of IPv6 Addresses (http://www.faqs.org/rfcs/rfc1924.html)(publicada em 1º Abril 1996), nunca vista em campo. Provavelmente é uma pegadinha ou mentirinha da data. Mas aqui está um exemplo:

```
# ipv6calc --addr_to_base85 2001:0db8:0100:f101:0210:a4ff:fee3:9566
9R}vSQZ1W=9A_Q74Lz&R
```

> Info: ipv6calc é uma calculadora de formato de endereços IPv6 que também faz conversões, e pode ser encontrada aqui: ipv6calc homepage (http://www.deepspace6.net/projects/ipv6calc.html) (Mirror (http://mirrors.bieringer.de/www.deepspace6.net/projects/ipv6calc.html))

2.4. FAQ (Básico)

2.4.1. Porque o nome do sucessor do IPv4 é IPv6 e não IPv5 ?

No cabeçalho IP, os primeiros 4 bits são reservados para a versão do protocolo. Então em teoria qualquer número entre 0 e 15 seria possível.

- 4: já em uso pelo IPv4
- 5: está reservado para o Stream Protocol (STP,RFC 1819 / Internet Stream Protocol Version 2 (http://www.faqs.org/rfcs/rfc1819.html)) (o qual nunca foi realmente feito para o público)

O próximo número disponível era 6. Portando, assim nasceu o IPv6!

2.4.2. Endereços IPv6: porque um número tão grande de bits ?

Durante o desenvolvimento do IPv4, as pessoas pensaram que 32 bits seriam suficientes para o mundo. Olhando de volta, realmente 32 bits foram suficientes por bastante tempo. entretanto 32 bits não foram suficientes para prover endereços globais para todos os dispositivos de rede no futuro (ou será já no presente ?). Pense em telefones celulares, tablets, computadores virtuais, carros, GPS's, geladeiras, TV's, etc.

Assim, os desenvolvedores escolheram 128 bits, 4 vezes maior (no campo do tamanho do endereço) do que o IPv4.

Mas o tamanho utilizável é menor do que parece. Isto se deve por causa do esquema utilizado na definição do endereçamento: 64 bits são usados para identificar a interface. Os outros 64 bits são usados para o roteamento. Assumindo os níveis de agregação (/48, /32, ...), é possível que os endereços também se esgotem, mas esperamos que não em futuro próximo.

Para mais informações veja tambémRFC 1715 / The H Ratio for Address Assignment Efficiency (http://www.faqs.org/rfcs/rfc1715.html) e RFC 3194 / The Host-Density Ratio for Address Assignment Efficiency (http://www.faqs.org/rfcs/rfc3194.html).

2.4.3. Endereços IPv6: porque um número tão pequeno de bits em um nova versão ?

Enquanto existam (possivelmente) algumas pessoas (só sei do Jim Fleming...) na internet que estejam pensando sobre o IPv8 ou IPv16, estes projetos estão muito longe de serem aceitos e implementados. Enquanto isso, 128 bits foi a melhor escolha levando em consideração o overhead do cabeçalho e o transporte de dados. Considere o MTU mínimo no IPv4 (576 octetos) e no IPv6 (1280 octetos), o tamanho do cabeçalho em iIPv4 é de 20 octetos (mínimo, e pode aumentar até 60 octetos com outras opções usadas) e no IPv6 é de 40 octetos (fixo). Isto representa 3,4% de overhead no IPv4 (com o tamanho mínimo) e 3,1 % do menor MTU em IPv6. O overhead é praticamente igual. Mais bits para endereço necessitariam cabeçalhos maiores e consequentemente mais overhead. Além disso, considere o tamanho máximo de uma MTU em links normais (como em Ethernet hoje): são 1500 octetos (em alguns casos especiais 9.000 octetos usando jumbo frames). Assim, não seria um projeto apropriado se 10% a 20% dos dados transportados para a camada 3 fosse usado para endereçamento e não para dados propriamente ditos.

Chapter 3. Tipos de endereço

Como no IPv4, os endereços em IPv6 também podem ser divididos em duas partes - host e rede - com a utilização de máscaras de rede.

O IPv4 tem mostrado que algumas vezes é bom, se mais de um endereço IP puder ser configurado em uma interface, cada um deles com um propósito bem diferente (aliases, multicast). Então para continuar extensível, o IPv6 também suporta esta característica e permite que mais de 1 endereço seja configurado na mesma interface. Atualmente não existe qualquer limitação definida pela RFC, a não ser na implementação da pilha IPv6 (para evitar ataques DoS).

Ao usar este grande número de bits para endereço, o IPv6 define tipos de endereços baseado nos bits iniciais, os quais são, espera-se, não sejam quebrados no futuro, como acontece hoje com o IPv4 e as suas classes A, B e C.

Estes números de bits são separados para endereçar redes (os primeiros 64 bits) e para endereços de host (os últimos 64 bits), para facilitar a auto-configuração.

3.1. Endereços sem um prefixo especial

3.1.1. Endereço localhost

Este é um endereço especial para a interface de loopback, similar ao 127.0.0.1 no IPv4. Em IPv6, este endereço de localhost é:

```
0000:0000:0000:0000:0000:0000:0000:0001
```

ou em sua forma comprimida:

```
::1
```

Os pacotes com este endereço como origem ou destino nunca devem sair ou entrar em um host.

3.1.2. Endereços não especificados

Este é um endereço especial, como "any" ou "0.0.0.0". Em IPv6 é representado assim:

```
0000:0000:0000:0000:0000:0000:0000:0000
```

ou:

```
::
```

Este endereço é geralmente usado para especificação de portas (qualquer IPv6) ou tabelas de roteamento.

Nota: este endereço nunca pode ser usado como um endereço de destino.

3.1.3. Endereços IPv6 vinculados a endereços IPv4

Existem dois endereços que contém endereços IPv4.

3.1.3.1. IPv4-mapeado para IPv6

Um endereço IPv4-único para compatibilidade IPv6 é às vezes usado ou mostrado para sockets criados por um daemon IPv6, mas que só recebe conexões de endereços IPv4.

Estes endereços são definidos dentro de um prefixo especial, com o tamanho /96 (a.b.c.d é o endereço IPv4):

```
0:0:0:0:0:ffff:a.b.c.d/96
```

ou em seu formato comprimido

```
::ffff:a.b.c.d/96
```

por exemplo, o endereço IP 1.2.3.4 seria assim:

```
::ffff:1.2.3.4
```

3.1.3.2. por exemplo, o endereço IP 1.2.3.4 seria assim:

Usado para tunelamento automático(RFC 2893 / Transition Mechanisms for IPv6 Hosts and Routers (http://www.faqs.org/rfcs/rfc2893.html)), o qual é substituído pelo 6to4 tunneling.

```
0:0:0:0:0:0:a.b.c.d/96
```

ou em seu formato comprimido

```
::a.b.c.d/96
```

3.2. Parte da rede, também conhecido como prefixo

Os designers definiram alguns tipos de endereços e deixaram muito disto para futuras definições quando novas necessidades surgirem. A RFC 4291 / IP Version 6 Addressing Architecture (http://www.faqs.org/rfcs/rfc4291.html) define o esquema utilizado no endereçamento atual.

Vamos agora dar uma olhada nos diferentes tipos de prefixos (e também em tipos de endereços):

3.2.1. Endereço tipo "link local"

Estes são endereços epeciais que são válidos somente no link de uma interface. Usando este endereço como destino, os pacotes nunca serão encaminhados a um router. Isto é usado para links de comunicação, tais como:

- Há alguém está aqui neste link ?
- Há alguém aqui com endereços especiais (ex. procurando por um router) ?

Eles começam com (onde "x" é qualquer caractere hexadecimal, normalmente "0")

```
fe8x:  <- atualmente é o único em uso
fe9x:
feax:
febx:
```

Um endereço com este prefixo é encontrado em cada interface com IPv6 habilitado após a auto-configuração stateless (a qual é normalmente sempre o caso).

3.2.2. Endereço tipo "site local"

Estes endereços são similares aos daRFC 1918 / Address Allocation for Private Internets (http://www.faqs.org/rfcs/rfc1918.html) em uso atualmente em IPv4, com a vantagem adicional de que qualquer pessoa que usar este tipo de endereço tem a capacidade de usar até 16 bits para a definição máxima de 65535 subredes. Comparável com o 10.0.0.0/8 do IPv4.

Outra vantagem: como é possível colocar mais de 1 endereço em uma interface com IPv6, voce pode configurar um endereço de "site local" junto com um endereço global.

Ele começa com:

```
fecx:  <- mais usado, mais comum
fedx:
feex:
fefx:
```

(onde o "x" é qualquer caractere hexadecimal, geralmente um "0")

Este tipo de endereço não deveria mais ser usado, RFC 3879 / Deprecating Site Local Addresses (http://www.faqs.org/rfcs/rfc3879.html), mas para um teste em laboratório, estes endereços ainda continuam sendo uma boa escolha (IMHO - em minha humilde opinião).

3.2.3. Endereços locais Unicast IPv6

Por causa da definição original de endereços de site local não serem únicos, pode haver algum problema se duas redes já configuradas forem se conectar em um futuro próximo (overlap de subredes). Este e outros problemas foram os motivos para um novo tipo de endereço definido naRFC 4193 / Unique Local IPv6 Unicast Addresses (http://www.faqs.org/rfcs/rfc4193.html).

Ele começa com:

```
fcxx:
fdxx:  <- atualmente o único em uso
```

Uma parte do prefixo (40 bits) é gerada usando um algoritmo pseudo-randomico e é improvável que dois resultados gerados por este algoritmo sejam iguais.

Exemplo de um prefixo gerado por este algoritmo (veja em: Goebel Consult / createLULA (http://www.goebel-consult.de/ipv6/createLULA)):

```
fd0f:8b72:ac90::/48
```

3.2.4. Endereço tipo Global "(Aggregatable) global unicast"

Atualmente, existe um tipo de endereço definido globalmente (o primeiro design, chamado "provider based") que foi jogado fora a alguns anos atrásRFC 1884 / IP Version 6 Addressing Architecture [obsolete] (http://www.faqs.org/rfcs/rfc1884.html), e voce consegue encontrar em algumas versões do kernel do Linux.

Ele começa com (os caracteres "x" são hexadecimais)

`2xxx:`
`3xxx:`

Nota: o prefixo "aggregatable" foi descartado nos atuais drafts. Há ainda alguns outros subtipos definidos. Veja abaixo:

3.2.4.1. Endereço de teste 6bone

Estes foram os primeiros endereços globais que foram definidos e usados. Eles começam com

```
3ffe:
```

Exemplo:

```
3ffe:ffff:100:f102::1
```

Um endereço de teste especial para o 6bone que nunca seria globalmente único começa com

```
3ffe:ffff:
```

e a maioria deles é mostrado em exemplos antigos. A razão para isso é, se endereços reais são mostrados, seria possível alguém copiar e colar estes endereços de arquivos de configuração antigos, o que inadvertidamente causaria um erro de duplicação de endereço de um endereço global único. Isto poderia causar sérios problemas para o host original (como nunca receber as respostas de requisições feitas).

Como o IPv6 agora já está em produção, este prefixo não é mais delegado e ele foi removido do processo de roteamento (vejaRFC 3701 / 6bone Phaseout (http://www.faqs.org/rfcs/rfc3701.html) para mais detalhes).

3.2.4.2. Endereços 6to4

Estes endereços, feitos para um mecanismo de túnel especial [RFC 3056 / Connection of IPv6 Domains via IPv4 Clouds (http://www.faqs.org/rfcs/rfc3056.html) e RFC 2893 / Transition Mechanisms for IPv6 Hosts and Routers (http://www.faqs.org/rfcs/rfc2893.html)],utilizam um endereço IPv4 já fornecido e a sua possível subnet, e começam com

```
2002:
```

Por exemplo, este endereço 192.168.1.1/5 ficaria:

```
2002:c0a8:0101:5::1
```

Um pequeno comando em shell poderia ajudar voce a gerar este endereço, baseado em um endereço IPv4 fornecido:

```
ipv4="1.2.3.4"; sla="5"; printf "2002:%02x%02x:%02x%02x:%04x::1" `echo $ipv4
¬ | tr "." " "` $sla
```

Veja também tunneling using 6to4 e information about 6to4 relay routers.

3.2.4.3. Designado pelo provedor para roteamento hierárquico

Este endereço é delegado pelo ISP e começa com

```
2001:
```

Prefixos de ISP's maiores (ou AS's) são delegados pelos local registries e tem atualmente um tamanho de prefixo /32.

Qualquer outro ISP/empresa pode solicitar um prefixo de tamanho /48, mas isto depende da política de distribuição de endereços dos registros locais de cada país ou região.

3.2.4.4. Endereços reservados para exemplos e documentação

Atualmente, dois blocos de endereço estão reservados para exemplos e documentação. Veja aRFC 3849 / IPv6 Address Prefix Reserved for Documentation (http://www.faqs.org/rfcs/rfc3849.html):

```
3fff:ffff::/32
2001:0DB8::/32   EXAMPLENET-WF
```

Estes endereços devem ser filtrados baseados no endereço de origem e NÃO devem ser roteados em roteadores de borda em direção à internet, se possível.

3.2.5. Endereços Multicast

Endereços Multicast são usados por serviços específicos.

Eles sempre começam com (xx é o valor de escopo)

`ffxy:`

Existem divisões entre escopo e tipo:

3.2.5.1. Escopo Multicast

O escopo Multicast é um parametro usado para especificar a distancia máxima que um pacote multicast pode "viajar" a partir de sua origem.

Atualmente, os seguintes escopos (ou regiões) estão definidos:

- ffx1: nó local, os pacotes nunca deixam o nó.
- ffx2: link-local, os pacotes nunca são encaminhados pelos routers, assim eles nunca deixam o link especificado.
- ffx5: site-local, os pacotes nunca deixam o site.
- ffx8: organization-local, os pacotes nunca deixam a organização (não é tão fácil de implementar, mas deve ser coberto pelo protocolo de roteamento).
- ffxe: escopo global.
- outros são reservados

3.2.5.2. Tipos Multicast

Já existem muitos tipos definidos/reservados (vejaRFC 4291 / IP Version 6 Addressing Architecture (http://www.faqs.org/rfcs/rfc4291.html) para mais detalhes). Alguns exemplos são:

- Endereço All Nodes: ID = 1h, endereça todos os host no nó local (ff01:0:0:0:0:0:0:1) ou no link conectado (ff02:0:0:0:0:0:0:1).

- Endereço All Routers: ID = 2h, endereça tosos os routers no nó local (ff01:0:0:0:0:0:0:2), no link conectado (ff02:0:0:0:0:0:0:2), ou no site local (ff05:0:0:0:0:0:0:2)

3.2.5.3. Endereço multicast solicitado nó link-local

Um endereço de multicast especial que é usado como endereço de destino para a descoberta da vizinhança , uma vez que no IPv6 não há ARP, como existe no IPv4.

Um exemplo deste endereço se parece com

```
ff02::1:ff00:1234
```

Os prefixos usados mostram que este é um endereço multicast link-local. O sufixo é gerado a partir do endereço de destino. Neste exemplo, um pacote deveria ser enviado ao endereço "fe80::1234", mas a parte de rede não conhece o MAC atual deste destino. Ele então substitui os 104 bits mais altos com "ff02:0:0:0:0:1:ff00::/104" e deixa os menores 24 bits intocados. Este endereço então é agora usado no link para achar o nó correspondente que tem que enviar uma resposta contendo o endereço MAC usado na camada 2.

3.2.6. Endereços Anycast

Endereços Anycast são endereços especiais e eles são usados para muitas coisas, como o servidor DNS ou DHCP mais próximo, e outras coisas. Estes endereços são obtidos do espaço de endereçamento Unicast (aggregatable global ou site-local). O mecanismo anycast (do ponto de vista do cliente) será tratado pelos protocolos de roteamento dinamico.

Nota: Endereços anycast não podem ser usados como endereços de origem, pois eles se aplicam somente a endereços de destino.

3.2.6.1. Endereços Anycast Subnet-router

Um exemplo simples para um endereço unicast é o anycast subnet-router. Assumindo que um nó tem os seguintes endereços globais IPv6 configurados:

```
2001:db8:100:f101:210:a4ff:fee3:9566/64  <- Node's address
```

O endereço unicast subnet-router será criado removendo o sufixo (os 64 bits menos significantes) completamente:

```
2001:db8:100:f101::/64  <- subnet-router anycast address
```

3.3. Tipos de endereço (parte de host)

Para a auto-configuração e questões de mobilidade, foi decidido usar os 64 bits de menor significado como a parte de host do endereço na maioria dos tipos de endereços atuais. Desta forma, cada subnet pode suportar uma grande quantidade de endereços.

A parte de host pode ser verificada de maneira distinta:

3.3.1. Computado automaticamente (também conhecido como stateless)

Com a auto-configuração, a parte host do endereço é feita através da conversão do endereço MAC da interface (se disponível), através do método EUI-64, para um único endereço IPv6. Se nenhum MAC estiver disponível para este dispositivo (isto acontece bastante em dispositivos virtuais), outra coisa (como o endereço IPv4 ou o MAC da interface física) é usada no lugar.

Exemplo: uma placa de rede tem o seguinte endereço MAC (48 bit):

```
00:10:a4:01:23:45
```

Isto poderia ser expandido de acordo com o tutorial do IEEE -IEEE-Tutorial EUI-64 (http://standards.ieee.org/regauth/oui/tutorials/EUI64.html) resultando no endereço abaixo:

```
0210:a4ff:fe01:2345
```

Com um prefixo já fornecido, o resultado é o endereço IPv6 mostrado abaixo:

```
2001:0db8:0100:f101:0210:a4ff:fe01:2345
```

3.3.1.1. Problema de privacidade com os endereços automaticamente computados e uma solução

Por causa da parte host do endereço "automaticamente computado" ser único, (exceto quando um fabricante de placas de rede usa o mesmo MAC em mais de um NIC), o rastreamento de um cliente é possível quando não há um proxy de qualquer tipo.

Este é um problema já conhecido, e a sua solução foi definida através da extensão de privacidade, definida naRFC 3041 / Privacy Extensions for Stateless Address Autoconfiguration in IPv6 (http://www.faqs.org/rfcs/rfc3041.html) (já existe também um novo draft disponível: draft-ietf-ipv6-privacy-addrs-v2-* (http://www.ietf.org/ids.by.wg/ipv6.html)). Usando um valor estático e um valor randomico, um novo sufixo é gerado de tempos em tempos.

Nota: isto é somente interessante em conexões finais de clientes, e não é realmente útil para servidores já conhecidos.

3.3.2. Definido manualmente

Para servidores, provavelmente é mais fácil se lembrar de endereços mais simples, mas isto também pode ser resolvido. Em IPv6, é possível configurar um endereço adicional para uma interface. Veja o exemplo:

```
2001:0db8:100:f101::1
```

Para sufixos manuais como o ::1 mostrado acima, é necessário que o sétimo bit mais significativo seja definido como 0 (o bit universal/local do identificador gerado automaticamente). Existe também uma outra combinação de bits (não utilizada) que é reservada para endereços unicast.

3.4. Tamanho de prefixos para roteamento

Na fase inicial de projeto e design do IPv6, foi planejada a utilização hierárquica de roteamento para reduzir o tamanho das tabelas de roteamento ao menor valor possível. As razões por trás desta abordagem foram o grande número de rotas nos grandes roteadores de borda (cerca de 300.000 em janeiro de 2011), reduzindo a necessidade de memória nos routers e a capacidade de se utilizar chips ASIC (Application Specified Integrated Circuit) para manipular esta tabela, aumentando a velocidade (uma tabela menor aumenta a velocidade).

A visão de hoje é que o roteamento será mais hierárquico para redes com somente 1 ISP. Em redes com mais de 1 conexão, isto não é possível, e está sujeita a sistemas multi-homed (informações de multi-homing:drafts-ietf-multi6-* (http://www.ietf.org/ids.by.wg/multi6.html),IPv6 Multihoming Solutions (http://arneill-py.sacramento.ca.us/ipv6mh/)).

3.4.1. Tamanho de prefixo (também conhecido como "netmasks")

Semelhante ao IPv4, a rede roteável entra em cena. Por causa da notação do padrão de máscara de rede (128 bits) não parecer bom, os designers utilizaram o mesmo método utilizado no IPv4, chamado Classless Inter Domain Routing (CIDR,RFC 1519 / Classless Inter-Domain Routing (http://www.faqs.org/rfcs/rfc1519.html))o qual especifica o número de bits do endereço IP que será usado para o roteamento. E ele também é chamado notação "slash".

Exemplo:

```
2001:0db8:100:1:2:3:4:5/48
```

Esta notação pode ser expandida:

- Rede:

```
2001:0db8:0100:0000:0000:0000:0000:0000
```

- Máscara de rede:

```
ffff:ffff:ffff:0000:0000:0000:0000:0000
```

3.4.2. Encontrando uma rota

Em circunstancias normais, (sem QoS), a procura em uma tabela de roteamento resulta na rota mais adequada com o número mais significante de bits do endereço. Em outras palavras, a rota com o maior prefixo tem a preferencia.

Por exemplo, se uma tabela de rotas mostra as seguintes entradas (a lista é parcial):

```
2001:0db8:100::/48     ::             U  1 0 0 sit1
2000::/3               ::192.88.99.1 UG 1 0 0 tun6to4
```

Os endereços de destino mostrados dos pacotes IPv6 serão roteados através das interfaces mostradas

```
2001:0db8:100:1:2:3:4:5/48  ->  routed through device sit1
2001:0db8:200:1:2:3:4:5/48  ->  routed through device tun6to4
```

Chapter 4. Verificação do sistema para IPv6

Antes de voce começar a utilizar o IPv6 em uma máquina com Linux, é necessário testar para saber se seu sistema tem o suporte ao protocolo. Talvez voce tenha que fazer algum ajuste para prepara-lo antes de começar a usar.

4.1. Kernel com IPv6

As distribuições mais novas de Linux já tem o kernel com suporte ao IPv6, e este suporte geralmente acontece com a compilação em módulos, mas é possível que estes módulos sejam carregados no momento do boot.

Nota: voce não deve usar o kernel da série 2.2, porque ele já não é mais atualizado. A série 2.4 também já não tem todas as atualizações de acordo com as últimas RFC's, então recomendamos utilizar um kernel da série 2.6.

4.1.1. Verificação do suporte a IPv6 no kernel utilizado

Para verificar se o seu kernel já está com o suporte a IPv6 habilitado, de uma olhada nos arquivos do diretório /proc. A seguinte entrada deve existir:

```
/proc/net/if_inet6
```

Para quem gosta de scripts, é possível usar estes comandos:

```
# test -f /proc/net/if_inet6 && echo "Running kernel is IPv6 ready"
```

Se este teste falhar, provavelmente seu sistema não está com os módulos de IPv6 carregados.

4.1.2. Tentando carregar os módulos para o IPv6

Voce pode tentar carregar os módulos do IPv6 com o comando

```
# modprobe ipv6
```

Se a carga ocorreu sem problemas, verifique o status com estes comandos:

```
# lsmod |grep -w 'ipv6' && echo "IPv6 module successfully loaded"
```

Depois disso, rode os comandos novamente do item 4.1.1 para ter certeza de que está tudo certo.

Note: a remoção do módulo (rmmod) não é suportada, e recomendo não utilizar, pois pode haver alguma instabilidade no sistema.

4.1.2.1. Carga automática do módulo

É possível automatizar a carga do módulo IPv6 conforme seja necessário. Para isto, basta adicionar a seguinte entrada no arquivo de configuração (/etc/modules.conf ou /etc/conf.modules):

```
alias net-pf-10 ipv6  # automatically load IPv6 module on demand
```

Também é possível desabilitar a carga do módulo automaticamente usando a seguinte entrada:

```
alias net-pf-10 off   # disable automatically load of IPv6 module on demand
```

Nota: no kernel da série 2.6, o mecanismo carregador de módulos mudou, e o novo arquivo de configuração é o /etc/modprobe.conf.

4.1.3. Compilando o kernel 2.6 para suportar o IPv6

Se os dois resultados acima foram negativos, e o seu kernel não tem suporte para o IPv6, então voce tem algumas coisas a fazer:

- Atualizar a sua distribuição para uma que suporte o IPv6 (recomendado para os novatos)
- Compilar um novo kernel (fácil, se voce souber quais opções são necessárias)
- Recompilar os fontes do kernel dado pela sua distribuição (nem sempre tão fácil)
- Compilar um kernel com as extensões USAGI

Se voce decidir compilar um kernel, voce precisa ter alguma experiencia nisso e também ler oLinux Kernel HOWTO (http://www.tldp.org/HOWTO/Kernel-HOWTO.html).

Uma comparação entre o kernel vanilla e as extensões USAGI está disponível aquiIPv6+Linux-Status-Kernel (http://www.bieringer.de/linux/IPv6/status/IPv6+Linux-status-kernel.html).

4.1.3.1. Compilando um kernel vanilla

Mais detalhes e dicas sobre a compilação de um kernel com suporte a IPv6 pode ser encontrado emIPv6-HOWTO-2#kernel (http://www.bieringer.de/linux/IPv6/IPv6-HOWTO/IPv6-HOWTO-2.html#kernel).

Nota: voce deve usar, sempre que possível, um kernel da séria 2.6, uma vez que o suporte ao IPv6 na série 2.4 já não teve as últimas atualizações definidas nas RFC's, e a série 2.2 não tem mais o suporte atualizado ou mesmo mantido por alguém.

4.1.3.2. Compilando um kernel com as extensões USAGI

Para a família vanilla de kernel, recomendado somente para usuários avançados, os quais já estão familiarizados com o IPv6 e com compilação de kernel. Veja tambémUSAGI project / FAQ (http://www.linux-ipv6.org/faq.html) e Obtaining the best IPv6 support with Linux (Article) (http://www.deepspace6.net/docs/best_ipv6_support.html) (Mirror (http://mirrors.bieringer.de/www.deepspace6.net/docs/best_ipv6_support.html)).

4.1.4. Dispositivos de rede com suporte a IPv6

Nem todos os dispositivos de rede tem suporte (ou terão) para transportar pacotes IPv6. Um status atualizado pode ser encontrado emIPv6+Linux-status-kernel.html#transport (http://www.bieringer.de/linux/IPv6/status/IPv6+Linux-status-kernel.html#transport).

O maior problema disso é causado na implementação da camada de rede, já que o pacote IPv6 não é reconhecido pelo cabeçalho IP (6 ao invés de 4). Ele é reconhecido pelo protocolo da camada 2 (transporte). Da mesma maneira, qualquer protocolo da camada 2 que não usa numeração de protocolo não conseguirá encaminhar os pacotes IPv6.

Nota: mesmo assim o pacote ainda é transportado pelo link, mas no lado receptor, o encaminhamento não ocorrerá (voce pode verificar isso com a utilização do tcpdump).

4.1.4.1. Estes links nunca suportarão IPv6

- Serial Line IP (SLIP, RFC 1055 / SLIP (http://www.faqs.org/rfcs/rfc1055.html)), deveria ser chamado de SLIPv4, nome do dispositivo: slX
- Parallel Line IP (PLIP), que nem o SLIP, nome do dispositivo: plipX
- ISDN com encapsulamento rawip, nome do dispositivo: isdnX

4.1.4.2. Este link atualmente não suporta IPv6

- ISDN com encapsulamento syncppp, nome do dispositivo: ipppX (problema de projeto do ipppd, que deveria ter sido resolvido com um PPP mais generalista na série de kernel 2.5)

4.2. Ferramentas de configuração de rede que suportam IPv6

Voce não irá muito longe, se voce estiver rodando um kernel com suporte a IPv6, mas não tiver ferramentas que o ajudem a configurar o IPv6. Existem vários pacotes para ajudá-lo neste trabalho.

4.2.1. Pacote net-tools

O pacote net-tools inclui algumas ferramentas como ifconfig e route, que ajudam a configurar uma interface em IPv6. Veja a saída dos comandos ifconfig -? ou route -?, se eles mostrarem algo parecido com IPv6 ou inet6, então a ferramenta tem suporte a IPv6.

Novamente, para quem gosta de scripts:

```
# /sbin/ifconfig -? 2>& 1|grep -qw 'inet6' && echo "utility 'ifconfig' is
¬ IPv6-ready"
```

Verificando o route:

```
# /sbin/route -? 2>& 1|grep -qw 'inet6' && echo "utility 'route' is IPv6-ready"
```

4.2.2. Pacote iproute

Alexey N. Kuznetsov (atual mantenedor do código de rede no Linux) criou um grupo de ferramentas que configuram redes através do dispositivo netlink. O uso destas ferramentas dá mais funcionalidades do que as do pacote net-tools, mas elas não estão muito bem documentadas e não são para os fracos de coração.

```
# /sbin/ip 2>&1 |grep -qw 'inet6' && echo "utility 'ip' is IPv6-ready"
```

Se o programa /sbin/ip não for encontrado em seu sistema, então eu recomendo que voce instale o pacote iproute.

- Voce pode pega-lo através de sua distribuição (se houver)

- Voce pode procurar o pacote RPM emRPMfind/iproute (http://rpmfind.net/linux/rpm2html/search.php?query=iproute) (em alguns casos é recomendada a reconstrução do SRPMS)

4.3. Programas de teste e debug IPv6

Após a preparação do seu sistema para o IPv6, está na hora de usar este protocolo para a sua comunicação com outros sistemas. Primeiro voce deve aprender como analisar os pacotes através de um sniffer. Isto é altamente recomendável para que qualquer debug ou troubleshooting seja feito de maneira rápida.

4.3.1. Ping IPv6

Este programa está incluído no pacote iputils. Seu objetivo é enviar e testar o transporte de pacotes ICMPv6 echo-request packets e aguardar pelos pacotes ICMPv6 echo-reply.

Uso

```
# ping6 <hostwithipv6address>
# ping6 <ipv6address>
# ping6 [-I <device>] <link-local-ipv6address>
```

Exemplo

```
# ping6 -c 1 ::1
PING ::1(::1) from ::1 : 56 data bytes
64 bytes from ::1: icmp_seq=0 hops=64 time=292 usec

--- ::1 ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max/mdev = 0.292/0.292/0.292/0.000 ms
```

Dica: o comando ping6 precisa de acesso direto ao socket e por isso precisa de permissão de root. Então se usuários comuns (não-root) tentarem usar o ping6 e não obtiverem sucesso, podem ser um dos dois problemas:

1. ping6 não está na variável PATH deste usuário (provavelmente porque o ping6 é geralmente localizado em /usr/sbin, e adicionar este diretório ao path do usuário comum não é muito recomendado)
2. ping6 não executa corretamente, geralmente porque faltam permissões de root. A sugestão neste caso é executar o comando chmod u+s /usr/sbin/ping6 para permitir o uso do programa.

4.3.1.1. Especificando a interface para o ping em IPv6

Ao usar um endereço link-local para pingar alguém em IPv6 o kernel pode não reconhecer ou saber através de qual interface (física ou virtual) o pacote deve ser enviado. Por causa disso, a seguinte mensagem de erro deve aparecer:

```
# ping6 fe80::212:34ff:fe12:3456
connect: Invalid argument
```

Neste caso, voce precisa especificar qual interface deve ser usada para enviar o pacote, como mostrado abaixo:

```
# ping6 -I eth0 -c 1 fe80::2e0:18ff:fe90:9205
PING fe80::212:23ff:fe12:3456(fe80::212:23ff:fe12:3456) from
¬ fe80::212:34ff:fe12:3478 eth0: 56 data bytes
```

```
64 bytes from fe80::212:23ff:fe12:3456: icmp_seq=0 hops=64 time=445 usec

--- fe80::2e0:18ff:fe90:9205 ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss round-trip
¬ min/avg/max/mdev = 0.445/0.445/0.445/0.000 ms
```

4.3.1.2. Ping6 para endereços multicast

Um mecanismo interessante para detectar hosts com endereço IPv6 é pingar o endereço all-node multicast:

```
# ping6 -I eth0 ff02::1
PING ff02::1(ff02::1) from fe80:::2ab:cdff:feef:0123 eth0: 56 data bytes
64 bytes from ::1: icmp_seq=1 ttl=64 time=0.104 ms
64 bytes from fe80::212:34ff:fe12:3450: icmp_seq=1 ttl=64 time=0.549 ms (DUP!)
```

Diferente do IPv4, onde as respostas ao ping para endereços de broadcast podem ser desabilitadas, em IPv6 este comportamento não pode ser desabilitado, exceto pela utilização de um firewall IPv6 local.

4.3.2. Traceroute6 IPv6

Este programa geralmente está incluso no pacote iputils. É um programa conhecido, similar ao do IPv4. Veja o exemplo:

```
# traceroute6 www.6bone.net
traceroute to 6bone.net (3ffe:b00:c18:1::10) from 2001:0db8:0000:f101::2, 30
¬ hops max, 16 byte packets
 1 localipv6gateway (2001:0db8:0000:f101::1) 1.354 ms 1.566 ms 0.407 ms
 2 swi6T1-T0.ipv6.switch.ch (3ffe:2000:0:400::1) 90.431 ms 91.956 ms 92.377 ms
 3 3ffe:2000:0:1::132 (3ffe:2000:0:1::132) 118.945 ms 107.982 ms 114.557 ms
 4 3ffe:c00:8023:2b::2 (3ffe:c00:8023:2b::2) 968.468 ms 993.392 ms 973.441 ms
 5 3ffe:2e00:e:c::3 (3ffe:2e00:e:c::3) 507.784 ms 505.549 ms 508.928 ms
 6 www.6bone.net (3ffe:b00:c18:1::10) 1265.85 ms * 1304.74 ms
```

Nota: diferente das versões mais atuais co traceroute do IPv4, que usa pacotes ICMPv4 echo-request e pacotes UDP (default), o traceroute do IPv6 só é capaz de enviar pacotes UDP. Como voce já deve saber, pacotes ICMP echo-request são mais aceitos pelos firewalls e listas de acesso (ACL) de routers do que pacotes UDP.

4.3.3. Tracepath6 IPv6

Este programa costuma estar incluído no pacote iputils. É um programa similar ao traceroute6 e ele traça o caminho para um endereço dado, descobrindo o MTU ao longo deste caminho. Veja o exemplo abaixo:

```
# tracepath6 www.6bone.net
 1?: [LOCALHOST] pmtu 1480
 1: 3ffe:401::2c0:33ff:fe02:14 150.705ms
 2: 3ffe:b00:c18::5 267.864ms
 3: 3ffe:b00:c18::5 asymm 2 266.145ms pmtu 1280
 3: 3ffe:3900:5::2 asymm 4 346.632ms
 4: 3ffe:28ff:ffff:4::3 asymm 5 365.965ms
 5: 3ffe:1cff:0:ee::2 asymm 4 534.704ms
 6: 3ffe:3800::1:1 asymm 4 578.126ms !N
Resume: pmtu 1280
```

4.3.4. Tcpdump IPv6

No Linux, o tcpdump é a maior ferramenta para a captura de pacotes. Abaixo estão alguns exemplos. O suporte ao IPv6 já está adicionado nas versões 3.6 ou superiores deste programa.

O tcpdump usa diversas expressões e argumentos para realizar a filtragem de pacotes para minimizar o volume de informações apresentado:

- icmp6: filtra o tráfego ICMPv6 nativo
- ip6: filtra o tráfego nativo IPv6 (incluindo ICMPv6)
- proto ipv6: filtra o tráfego IPv6 tunelado em IPv4 (IPv6-in-IPv4)
- not port ssh: para evitar mostrar os pacotes se voce estiver usando uma conexão SSH

Além disso, algumas opções são muito úteis para obter mais informações de cada pacote, bem interessantes para pacotes ICMPv6:

- "-s 512": aumenta o tamanho do pacote capturado para 512 bytes. Se for usada a opção "-s 0" o pacote é capturado por inteiro
- "-s 512": aumenta o tamanho do pacote capturado para 512 bytes. Se for usada a opção "-s 0" o pacote é capturado por inteiro
- "-n": não resolve os endereços para nomes, muito útil quando o DNS reverso não está funcionando corretamente

4.3.4.1. Ping IPv6 para 2001:0db8:100:f101::1 nativo sobre um link local

```
# tcpdump -t -n -i eth0 -s 512 -vv ip6 or proto ipv6
tcpdump: listening on eth0
2001:0db8:100:f101:2e0:18ff:fe90:9205 > 2001:0db8:100:f101::1: icmp6: echo
¬ request (len 64, hlim 64)
2001:0db8:100:f101::1 > 2001:0db8:100:f101:2e0:18ff:fe90:9205: icmp6: echo
¬ reply (len 64, hlim 64)
```

4.3.4.2. Ping IPv6 para 2001:0db8:100::1 roteado através de um túnel IPv6-in-IPv4

Os endereços IPv4 1.2.3.4 e 5.6.7.8 são os tunnel endpoints (todos os endereços são exemplos)

```
# tcpdump -t -n -i ppp0 -s 512 -vv ip6 or proto ipv6
tcpdump: listening on ppp0
1.2.3.4 > 5.6.7.8: 2002:ffff:f5f8::1 > 2001:0db8:100::1: icmp6: echo request
¬ (len 64, hlim 64) (DF) (ttl 64, id 0, len 124)
5.6.7.8 > 1.2.3.4: 2001:0db8:100::1 > 2002:ffff:f5f8::1: icmp6: echo reply (len
¬ 64, hlim 61) (ttl 23, id 29887, len 124)
1.2.3.4 > 5.6.7.8: 2002:ffff:f5f8::1 > 2001:0db8:100::1: icmp6: echo request
¬ (len 64, hlim 64) (DF) (ttl 64, id 0, len 124)
5.6.7.8 > 1.2.3.4: 2001:0db8:100::1 > 2002:ffff:f5f8::1: icmp6: echo reply (len
¬ 64, hlim 61) (ttl 23, id 29919, len 124)
```

4.4. Programas com suporte a IPv6

As distribuições Linux atuais já contém a maioria dos serviços Cliente e Servidor em IPv6. Veja aqui emIPv6+Linux-Status-Distribution (http://www.bieringer.de/linux/IPv6/status/IPv6+Linux-status-distributions.html). Se ainda não estiver incluído, voce pode verificar em IPv6 & Linux - Current Status - Applications (http://www.bieringer.de/linux/IPv6/status/IPv6+Linux-status-apps.html) se o programa já está portado para o IPv6 e pronto para o Linux. Para os programas mais comuns existem dicas disponíveis emIPv6 & Linux - HowTo - Part 3 (http://www.bieringer.de/linux/IPv6/IPv6-HOWTO/IPv6-HOWTO-3.html) e IPv6 & Linux - HowTo - Part 4 (http://www.bieringer.de/linux/IPv6/IPv6-HOWTO/IPv6-HOWTO-4.html).

4.5. Programas cliente com suporte a IPv6

Para executar os testes abaixo, é necessário que seu sistema seja um host IPv6 e os exemplos mostrados podem ser feitos se voce tiver acesso ao 6bone.

4.5.1. Verificando a resolução DNS para endereços IPv6

Por causa dos updates de segurança aplicados nos últimos anos, o Servidor DNS que roda a versão mais atual já tem a capacidade de entender os endereços IPv6 tipo AAAA (o named A6 mais novo ainda não é usado porque só no BIND9 o suporte aos root domais ARPA IP6 está em uso). Um teste bem simples para ver o sistema resolver endereços IPv6 é:

```
# host -t AAAA www.join.uni-muenster.de
```

e a resposta deve ser alguma coisa parecida com isso:

```
www.join.uni-muenster.de. is an alias for tolot.join.uni-muenster.de.
tolot.join.uni-muenster.de. has AAAA address
¬ 2001:638:500:101:2e0:81ff:fe24:37c6
```

4.5.2. Cliente de Telnet com suporte a IPv6

Cliente de telnet com suporte a IPv6 estão disponíveis. Um teste simples pode ser feito com o comando:

```
$ telnet 3ffe:400:100::1 80
Trying 3ffe:400:100::1...
Connected to 3ffe:400:100::1.
Escape character is '^]'.
HEAD / HTTP/1.0

HTTP/1.1 200 OK
Date: Sun, 16 Dec 2001 16:07:21
GMT Server: Apache/2.0.28 (Unix)
Last-Modified: Wed, 01 Aug 2001 21:34:42 GMT
ETag: "3f02-a4d-b1b3e080"
Accept-Ranges: bytes
Content-Length: 2637
Connection: close
Content-Type: text/html; charset=ISO-8859-1

Connection closed by foreign host.
```

Se o cliente de telnet não entende o IPv6, a mensagem de erro será "cannot resolve hostname".

4.5.3. SSH com suporte a IPv6

4.5.3.1. openssh

As versões atuais do openssh já suportam IPv6. Dependendo da configuração utilizada, ele tem dois comportamentos:

- --without-ipv4-default: o cliente tenta se conectar primeiro em IPv6, e ele usa IPv4 se a conexão em IPv6 não for estabelecida
- --with-ipv4-default: a conexão é feita primeiro em IPv4 e para usar algum endereço IPv6, deve-se forçar a sua utilização. Veja o exemplo

```
$ ssh -6 ::1
user@::1's password: ******
[user@ipv6host user]$
```

Se seu cliente ssh não entende a opção "-6" então o suporte a IPv6 não está habilitado, como muitos pacotes de ssh na versão 1.

4.5.3.2. ssh.com

O software cliente e servidor SSH da SSH.com já suporta o IPv6 e agora ele é grátis para todos os Linux e FreeBSD, independente se o seu uso é pessoal ou comercial.

4.5.4. Browsers com suporte a IPv6

O status atual dos browsers com suporte a IPv6 pode ser encontrado aquiIPv6+Linux-status-apps.html#HTTP (http://www.bieringer.de/linux/IPv6/status/IPv6+Linux-status-apps.html#HTTP).

A maioria deles ainda tem problemas pendentes, tais como:

1. Se a configuração de proxy usa somente endereços IPv4, os pedidos em IPv6 também serão enviados a este proxy, emas como o proxy não saberá resolver o endereço, o pedido não vai funcionar. A única solução é verificar se o seu software de proxy tem alguma atualização para resolver este problema.
2. Configuração automática de proxy (*.pac) não pode ser utilizada para manipular pedidos em IPv6 de maneira diferenciada (exemplo: não usar o proxy para o IPv6) por causa da sua natureza (escritos em Javascript e muito encrustado no código fonte, como é visto no Mozilla).

As versões anteriores de browsers também não entenderiam uma URL com o endereço IPv6, como no exemplo http://[2001:a60:9002:1::186:6]/ (isto funciona somente se a URL for usada em um browser que suporte IPv6).

Um pequeno teste é tentar este endereço em um browser sem o proxy configurado.

4.5.4.1. URLs para teste

Um bom ponto de partida para browsers que usam IPv6 é o sitehttp://www.kame.net/.Se a tartaruga da página estiver animada, a sua conexão é em IPv6. caso contrário, a tartaruga ficará parada.

4.6. Programas servidores com suporte a IPv6

Nesta parte deste HowTo, outros softwares cliente IPv6 são mencionados, assim como dicas para servidores com suporte a IPv6, como sshd, httpd, telnetd, etc, assim como outras dicas emHints for IPv6-enabled daemons.

4.7. FAQ (checagem de sistema com suporte a IPv6)

4.7.1. Usando ferramentas

4.7.1.1. Q: Não consigo pingar (ping6) o endereço link-local

Mensagem de erro: "connect: Invalid argument"

O kernel não conhece qual interface física ou virtual voce quer utilizar para enviar o pacote ICMPv6. Assim, a solução poderia aparecer assim.

Solução:: Determine a interface, como: "ping6 -I eth0 fe80::2e0:18ff:fe90:9205", veja tambémprogram ping6 usage.

4.7.1.2. Q: Não consigo pingar (ping6) ou efetuar traceroute (traceroute6) como usuário normal

Menagem de erro: "icmp socket: Operation not permitted"

Estes utilitários criam pacotes especiais ICMPv6 e então os enviam. Isto é feito usando conexões brutas no kernel. Mas estas conexões somente podem ser usadas pelo usuário "root". Desta forma, esta mensagem vai aparecer para os usuários normais.

Solução: Se for realmente necessário que todos os usuários utilizem estas ferramentas, voce pode adicionar o "suid" bit usando o comando "chmod u+s /caminho/para/o/programa", e veja também este linkprogram ping6 usageSe nem todos os usuários necessitam usá-lo, voce pode mudar o grupo do programa, para "wheel" por exemplo, e todos os usuários pertencentes a este grupo poderão executar estes programas sem problema. Voce também pode configurar o "sudo" para isto também.

Chapter 5. Configurando interfaces

5.1. Dispositivos de rede diferentes

Em um nó, podem haver diferentes tipos de interfaces. Elas podem estar agrupadas em classes

- Físicas, como eth0, tr0
- Virtuais, como ppp0, tun0, tap0, sit0, isdn0, ippp0

5.1.1. Físicas

As interfaces físicas, como Ethernet ou Token Ring são exemplos de interfaces comuns que não precisam de qualquer tipo de tratamento especial.

5.1.2. Virtuais

As interfaces virtuais sempre precisam de algum tratamento especial

5.1.2.1. Interfaces Túnel IPv6-in-IPv4

Estas interfaces normalmente recebem o nome sit*x*. O nome sit é uma atalho para Simple Internet Transition. Esta interface tem a capacidade de encapsular os pacotes IPv6 em pacotes IPv4 e tunelar estes pacotes para um endpoint remoto.

A interface sit0 tem um papel especial e não pode ser usada para túneis dedicados.

5.1.2.2. Interfaces PPP

As interfaces PPP obteem sua capacidade IPv6 do daemon PPP para IPv6.

5.1.2.3. Interfaces ISDN HDLC

A capacidade IPv6 para HDLC com encapsulamento IP já está contida no kernel

5.1.2.4. Interfaces ISDN PPP

A interface ISDN PPP (ippp) não tem o suporte ao IPv6 no kernel. E também não há qualquer plano para suportar, porque o kernel da série 2.5 ele será substituído por uma camada de interface PPP mais genérica.

5.1.2.5. SLIP + PLIP

Como mencionado anteriormente neste documento, esta interface não suporta o IPv6 (no envio até que funciona, mas a recepção não funciona).

5.1.2.6. Dispositivo Ether-tap

Dispositivos Ether-tap já possuem o IPv6 habilitado e o stateless configurado. Para usá-lo, o módulo "ethertap" deve ter sido carregado antes.

5.1.2.7. Dispositivos tun

Atualmente não foi testado por mim

5.1.2.8. ATM

01/2002: Não são mais suportados pelo kernel vanilla, mas somente pelas extensões USAGI

5.1.2.9. Outras

Eu esqueci de alguma ?

5.2. Colocando as interfaces em up e down

Existem dois métodos usados para colocar as interfaces em up ou down..

5.2.1. Usando "ip"

Uso:

```
# ip link set dev <interface> up
# ip link set dev <interface> down
```

Exemplo:

```
# ip link set dev eth0 up
# ip link set dev eth0 down
```

5.2.2. Usando "ifconfig"

Uso:

```
# /sbin/ifconfig <interface> up
# /sbin/ifconfig <interface> down
```

Exemplo:

```
# /sbin/ifconfig eth0 up
# /sbin/ifconfig eth0 down
```

Chapter 6. Configurando endereços IPv6

Existem várias maneiras de configurar um endereço IPv6 em uma interface. As mais comuns são "ifconfig" e "ip".

6.1. Mostrando os endereços IPv6 existentes

Antes de mais nada, voce precisa checar se já existe algum endereço IPv6 configurado e qual é o seu tipo (talvez atribuído durante uma auto-configuração stateless).

6.1.1. Usando "ip"

Uso:

```
# /sbin/ip -6 addr show dev <interface>
```

Exemplo para uma configuração de host estático:

```
# /sbin/ip -6 addr show dev eth0
2: eth0: <BROADCAST,MULTICAST,UP&gt; mtu 1500 qdisc pfifo_ fast qlen 100
inet6 fe80::210:a4ff:fee3:9566/10 scope link
inet6 2001:0db8:0:f101::1/64 scope global
inet6 fec0:0:0:f101::1/64 scope site
```

Exemplo de um host auto-configurado

Aqui voce pode ver a configuração de IPv6 através do processo auto-magically, além do tempo de vida do endereço.

```
# /sbin/ip -6 addr show dev eth0
3: eth0: <BROADCAST,MULTICAST,PROMISC,UP&gt; mtu 1500 qdisc pfifo_fast qlen
¬ 100
inet6 2002:d950:f5f8:f101:2e0:18ff:fe90:9205/64 scope global dynamic
valid_lft 16sec preferred_lft 6sec
inet6 3ffe:400:100:f101:2e0:18ff:fe90:9205/64 scope global dynamic
valid_lft 2591997sec preferred_lft 604797sec inet6 fe80::2e0:18ff:fe90:9205/10
¬ scope link
```

6.1.2. Usando "ifconfig"

Uso:

```
# /sbin/ifconfig <interface>
```

Exemplo (a saída foi filtrada com o grep para mostrar somente os endereços IPv6). Aqui voce pode ver diferentes endereços IPv6 com diferentes escopos.

```
# /sbin/ifconfig eth0 |grep "inet6 addr:"
inet6 addr: fe80::210:a4ff:fee3:9566/10 Scope:Link
inet6 addr: 2001:0db8:0:f101::1/64 Scope:Global
inet6 addr: fec0:0:0:f101::1/64 Scope:Site
```

6.2. Adicionando um endereço IPv6

A adição de um endereço IPv6 é muito similar ao endereço "IP ALIAS" nas interfaces IPv4 no Linux.

6.2.1. Usando "ip"

Uso:

```
# /sbin/ip -6 addr add <ipv6address>/<prefixlength> dev <interface>
```

Exemplo:

```
# /sbin/ip -6 addr add 2001:0db8:0:f101::1/64 dev eth0
```

6.2.2. Usando "ifconfig"

Uso:

```
# /sbin/ifconfig <interface> inet6 add <ipv6address>/<prefixlength>
```

Exemplo:

```
# /sbin/ifconfig eth0 inet6 add 2001:0db8:0:f101::1/64
```

6.3. Removendo um endereço IPv6

Como esta ação não é tão necessária, tenha cuidado ao remover endereços IPv6 que não existem, pois ao realizar esta ação em kernels mais antigos, voce pode causar um grande estrago no sistema.

6.3.1. Usando "ip"

Uso:

```
# /sbin/ip -6 addr del <ipv6address>/<prefixlength> dev <interface>
```

Exemplo:

```
# /sbin/ip -6 addr del 2001:0db8:0:f101::1/64 dev eth0
```

6.3.2. Usando "ifconfig"

Uso:

```
# /sbin/ifconfig <interface> inet6 del <ipv6address>/<prefixlength>
```

Exemplo:

```
# /sbin/ifconfig eth0 inet6 del 2001:0db8:0:f101::1/64
```

Chapter 7. Configurando rotas IPv6

Se voce quer deixar seu link e quer enviar pacotes a todo o mundo que está só aguardando a sua conexão em IPv6, voce vai precisar de uma rota. Se já houver um router com IPv6 habilitado (e uma rota para ele), estes passos abaixo vão te ensinar como adicionar mais rotas em IPv6.

7.1. Mostrando as rotas IPv6 existentes

Antes de mais nada, é interessante verificar quais são as rotas IPv6 já configuradas (talvez atribuído durante uma auto-configuração).

7.1.1. Usando "ip"

Uso:

```
# /sbin/ip -6 route show [dev <device>]
```

Exemplo:

```
# /sbin/ip -6 route show dev eth0
2001:0db8:0:f101::/64 proto kernel metric 256 mtu 1500 advmss 1440
fe80::/10             proto kernel metric 256 mtu 1500 advmss 1440
ff00::/8              proto kernel metric 256 mtu 1500 advmss 1440
default               proto kernel metric 256 mtu 1500 advmss 1440
```

7.1.2. Usando "route"

Uso:

```
# /sbin/route -A inet6
```

Exemplo (a saída foi filtrada para a interface eth0). Aqui voce pode ver rotas IPv6 para diferentes endereços em uma única interface.

```
# /sbin/route -A inet6 |grep -w "eth0"
2001:0db8:0:f101 ::/64 :: UA  256 0 0 eth0 <- Interface route for global
¬ address
fe80::/10        ::       UA  256 0 0 eth0 <- Interface route for link-local
¬ address
ff00::/8         ::       UA  256 0 0 eth0 <- Interface route for all multicast
¬ addresses
::/0             ::       UDA 256 0 0 eth0 <- Automatic default route
```

7.2. Adicionando uma rota IPv6 através de um gateway

Bastante necessário quando se quer acessar outras redes com IPv6 usando um router IPv6-enabled em seu link.

7.2.1. Usando "ip"

Uso:

```
# /sbin/ip -6 route add <ipv6network>/<prefixlength> via <ipv6address>
¬ [dev <device>]
```

Exemplo:

```
# /sbin/ip -6 route add 2000::/3 via 2001:0db8:0:f101::1
```

7.2.2. Usando "route"

Uso:

```
# /sbin/route -A inet6 add <ipv6network>/<prefixlength> gw
¬ <ipv6address> [dev <device>]
```

Um dispositivo pode ser necessário se o dispositivo do endereço IPv6 do gateway for um dispositivo link local.

Veja o exemplo abaixo, como adicionar uma rota para todos os endereços globais (2000::/3) através do gateway 2001:0db8:0:f101::1

```
# /sbin/route -A inet6 add 2000::/3 gw 2001:0db8:0:f101::1
```

7.3. Removendo uma rota IPv6 através de um gateway

Não é geralmente usada no modo manual, pois sugerimos a utilização de scripts ou shutdown nas interfaces (todas ou por interface)

7.3.1. Usando "ip"

Uso:

```
# /sbin/ip -6 route del <ipv6network>/<prefixlength> via <ipv6address>
¬ [dev <device>]
```

Exemplo:

```
# /sbin/ip -6 route del 2000::/3 via 2001:0db8:0:f101::1
```

7.3.2. Usando "route"

Uso:

```
# /sbin/route -A inet6 del <network>/<prefixlength> gw <ipv6address> [dev
¬ <device>]
```

Exemplo para remover a rota adicionada anteriormente (acima):

```
# /sbin/route -A inet6 del 2000::/3 gw 2001:0db8:0:f101::1
```

7.4. Adicionando uma rota IPv6 através de uma interface

Nem sempre usado, mas quando usado é em links ponto a ponto.

7.4.1. Usando "ip"

Uso:

```
# /sbin/ip -6 route add <ipv6network>/<prefixlength> dev <device>
¬ metric 1
```

Exemplo:

```
# /sbin/ip -6 route add 2000::/3 dev eth0 metric 1
```

A métrica "1" é usada aqui para se manter compatível com as métricas usadas pelo comando route, já que a métrica default ao se usar "ip" é "1024".

7.4.2. Usando "route"

Uso:

```
# /sbin/route -A inet6 add <ipv6network>/<prefixlength> dev <device>
```

Exemplo:

```
# /sbin/route -A inet6 add 2000::/3 dev eth0
```

7.5. Removendo uma rota IPv6 através de uma interface

Nem sempre utilizado manualmente, pois recomenda-se a utilização de scripts.

7.5.1. Usando "ip"

Uso:

```
# /sbin/ip -6 route del <ipv6network>/<prefixlength> dev <device>
```

Exemplo:

```
# /sbin/ip -6 route del 2000::/3 dev eth0
```

7.5.2. Usando "route"

Uso:

```
# /sbin/route -A inet6 del <network>/<prefixlength> dev <device>
```

Exemplo:

```
# /sbin/route -A inet6 del 2000::/3 dev eth0
```

7.6. FAQ para rotas em IPv6

7.6.1. Suporte de uma rota default IPv6

Uma boa idéia do IPv6 foi o roteamento hierárquico, o que proporcionaria a necessidade de menos rotas nos roteadores.

Aqui estão alguns problemas no kernel atual do Linux:

7.6.1.1. Clientes (não roteando qualquer pacote!)

Um cliente pode configurar uma rota default prefixo "::/0", mas eles também aprendem uma rota no processo de auto configuração, ex.: uso do radvd no link abaixo mostra:

```
# ip -6 route show | grep ^default
default via fe80::212:34ff:fe12:3450 dev eth0 proto kernel metric 1024 expires
¬ 29sec mtu 1500 advmss 1440
```

7.6.1.2. Roteadores em caso de packet forwarding

Os linux mais velhos (pelo menos inferiores ao kernel 2.4.17) não suportam rotas default. Voce pode configura-las, mas a pesquisa na tabela vai retornar uma falha quando deveria ser encaminhado (intenção normal do roteador). Se voce ainda está usando uma versão antiga de kernel, as "rotas default" podem ser configuradas usando o prefixo de endereço global "2000::/3".

Nota: tome cuidado com as rotas default sem filtragem de endereços em roteadores de borda, pois o tráfego multicast ou site-local pode sair para o mundo.

Chapter 8. Descoberta de vizinhos

A descoberta de vizinhos funciona como um sucessor do ARP (Address Resolution Protocol) em IPv4, no mundo IPv6. Voce pode obter estas informações sobre os vizinhos atuais, e adicionalmente voce pode configurar e excluir entradas. O kernel mantém o rastreamento das descobertas bem sucedidas (como no ARP em IPv4). Voce pode pesquisar as entradas nesta tabela usando o comando "ip".

8.1. Mostrando os vizinhos usando "ip"

Com o comando abaixo voce pode verificar a tabela de vizinhos aprendida ou configurada

```
# ip -6 neigh show [dev <device>]
```

O exemplo a seguir mostra um vizinho, o qual é um router acessível

```
# ip -6 neigh show
fe80::201:23ff:fe45:6789 dev eth0 lladdr 00:01:23:45:67:89 router nud reachable
```

8.2. Manipulando a tabela de vizinhos usando "ip"

8.2.1. Adicionando uma entrada manualmente

Com o comando abaixo, voce consegue adicionar uma entrada manualmente

```
# ip -6 neigh add <IPv6 address> lladdr <link-layer address> dev <device>
```

Exemplo:

```
# ip -6 neigh add fec0::1 lladdr 02:01:02:03:04:05 dev eth0
```

8.2.2. Excluindo uma entrada manualmente

Similar à adição de uma entrada, uma entrada pode ser excluída

```
# ip -6 neigh del <IPv6 address> lladdr <link-layer address> dev <device>
```

Exemplo:

```
# ip -6 neigh del fec0::1 lladdr 02:01:02:03:04:05 dev eth0
```

8.2.3. Opções mais avançadas

A ferramenta "ip" não é tão documentada, mas é bem útil e forte. Veja o seu help online para mais informações:

```
# ip -6 neigh help
Usage: ip neigh { add | del | change | replace } { ADDR [ lladdr LLADDR ]
          [ nud { permanent | noarp | stale | reachable } ]
          | proxy ADDR } [ dev DEV ]
```

```
ip neigh {show|flush} [ to PREFIX ] [ dev DEV ] [ nud STATE ]
```

Aparentemente algumas opções são somente para o IPv4... e se voce quiser contribuir com mais informações sobre outras opções da ferramenta e usos avançados, por favor, me envie.

Chapter 9. Configurando um túnel IPv6-in-IPv4

Se voce quer deixar seu link e ter uma rede IPv6 à sua volta, voce vai precisar de um túnel IPv6-in-IPv4 para acessar a web em modo IPv6.

Existem alguns tipos de mecanismo e também algumas possibilidades da configuração deste túnel.

9.1. Tipos de túneis

Existe mais de uma possibilidade de transportar pacotes IPvb6 em links IPv4.

9.1.1. Túnel estático ponto a ponto: 6bone

Um túnel ponto a ponto é um túnel dedicado em direção a um ponto final., o qual sabe informações sobre uma rede IPv6A (para rotas de volta) e os endereços IPv4 deste túnel estão definidos na RFC 2893 / Transition Mechanisms for IPv6 Hosts and Routers (http://www.faqs.org/rfcs/rfc2893.html). Necessidades:

- O endereço IPv4 local do túnel precisa ser estático, global, único e acessível a partir da outra ponta
- Um prefixo global IPv6 deve ser designado a voce (veja o registro 6bone)
- Um ponto final remoto do túnel deve ser capaz de rotear seu prefixo IPv6para seu ponto final local (uma configuração manual pode ser necessária)

9.1.2. Túnel automático

Um túnel automático acontece quando um nó diretamente conectado a outro nó obtém um IPv4 do outro nó anterior.

9.1.3. 6to4-Tunneling

O túnel 6to4 (RFC 3056 / Connection of IPv6 Domains via IPv4 Clouds (http://www.faqs.org/rfcs/rfc3056.html)) utiliza um mecanismo simples para criar túneis automáticos. Cada nó com um endereço global único é capaz de ser uma ponta final de um túnel 6to4 (se nenhum firewall no meio do caminho bloquear este tipo de tráfego). Túneis 6to4 não costumam ser túneis um a um. Este tipo de túnel pode ser dividido em Upstream e Downstream. Além disso, um endereço especial IPv6 indica que este nó vai usar o tunelamento 6to4 para se conectar a redes IPv6 mundiais.

9.1.3.1. Geração de prefixo 6to4

O endereço 6to4 está definido abaixo (o esquema foi pego da RFC 3056 / Connection of IPv6 Domains via IPv4 Clouds (http://www.faqs.org/rfcs/rfc3056.html)):

```
|   3+13   |    32     |    16  |            64 bits             |
+---+------+-----------+--------+--------------------------------+
|  FP+TLA  |  V4ADDR   | SLA ID |           Interface ID         |
|  0x2002  |           |        |                                |
```

```
+---+------+-----------+--------+-------------------------------+
```

FP e TLA juntos (16 bits) tem o valor 0x2002. V4ADDR é o endereço único IPv4 (em notação hexadecimal). SLA é o identificador de rede (65536 redes locais possíveis) e são usados para representar a sua estrutura de rede local.

Para os gateways, tal prefixo é gerado normalmente usando o SLA “0000” e o sufixo "::1" (não é uma exigencia, pode ser arbitrário com um escopo local) e então assinalado a uma interface de túnel 6to4. Veja que a Microsoft também utiliza V4ADDR para o sufixo.

9.1.3.2. Tunelamento Upstream 6to4

O nó tem que saber para qual ponta remota de túnel os pacotes IPv4 com pacotes IPv6 devem ser encaminhados. No início dos tempos de tunelamento 6to4, upstreams dedicados aceitavam que os routers fizessem isso. Veja NSayer’s 6to4 information (http://www.kfu.com/~nsayer/6to4/) para uma lista de routers.

Hoje em dia, os routers upstream 6to4 podem ser encontrados automaticamente, usando o endereço de unicast 192.88.99.1. Os protocolos de roteamento se incumbem desta função, veja RFC 3068 / An Anycast Prefix for 6to4 Relay Routers (http://www.faqs.org/rfcs/rfc3068.html) para mais detalhes.

9.1.3.3. Tunelamento Downstream 6to4

O downstream (6bone -> seu nó com 6to4 habilitado) não é realmente correto e pode variar de um host remoto estranho que originou os pacotes que foram enviados a voce. Existem duas possibilidades:

- Estes hosts remotos usam endereços 6to4 e enviam os pacotes de volta diretamente a seu nó (veja abaixo)
- Estes hosts remotos enviam os pacotes de volta à rede IPv6 e dependendo do roteamento, um router no meio do caminho cria um túnel automaticamente em direção ao seu nó.

9.1.3.4. Tráfego 6to4 possível

- 6to4 para 6to4: normalmente é um túnel direto entre as duas pontas, ambos habilitados em 6to4
- 6to4 para não-6to4: é enviado via um túnel upstream
- não-6to4 para 6to4: é enviado via um túnel downstream

9.2. Mostrando os túneis existentes

9.2.1. Usando "ip"

Uso:

```
# /sbin/ip -6 tunnel show [<device>]
```

Exemplo:

```
# /sbin/ip -6 tunnel show
```

```
sit0: ipv6/ip remote any local any ttl 64 nopmtudisc
sit1: ipv6/ip remote 195.226.187.50 local any ttl 64
```

9.2.2. Usando "route"

Uso:

```
# /sbin/route -A inet6
```

Exemplo (a saída está filtrada para mostrar somente os túneis através da interface virtual sit0):

```
# /sbin/route -A inet6 | grep "\Wsit0\W*$"
::/96      ::               U   256  2  0  sit0
2002::/16  ::               UA  256  0  0  sit0
2000::/3   ::193.113.58.75  UG    1  0  0  sit0
fe80::/10  ::               UA  256  0  0  sit0
ff00::/8   ::               UA  256  0  0  sit0
```

9.3. Configuração de um túnel ponto a ponto

Existem possibilidades para adicionar ou remover um túnel ponto a ponto.

Uma boa informação adicional sobre a configuração de túneis fornecida através do comando “ip” está aqui Configuring tunnels with iproute2 (article) (http://www.deepspace6.net/docs/iproute2tunnel-en.html) (Mirror (http://mirrors.bieringer.de/www.deepspace6.net/docs/iproute2tunnel-en.html)).

9.3.1. Adicionando túneis ponto a ponto

9.3.1.1. Usando "ip"

Um método comum para a criação de uma quantidade pequena de túneis.

Use-o para criar um dispositivo túnel (mas não depois, o TTL também deve ser especificado, porque seu valor default é 0).

```
# /sbin/ip tunnel add <device> mode sit ttl <ttldefault> remote
¬ <ipv4addressofforeigntunnel> local <ipv4addresslocal>
```

Uso (exemplo genérico para 3 túneis):

```
# /sbin/ip tunnel add sit1 mode sit ttl <ttldefault> remote
¬ <ipv4addressofforeigntunnel1> local <ipv4addresslocal>
# /sbin/ip link set dev sit1 up
# /sbin/ip -6 route add <prefixtoroute1> dev sit1 metric 1

# /sbin/ip tunnel add sit2 mode sit ttl <ttldefault>
¬ <ipv4addressofforeigntunnel2> local <ipv4addresslocal>
# /sbin/ip link set dev sit2 up
# /sbin/ip -6 route add <prefixtoroute2> dev sit2 metric 1

# /sbin/ip tunnel add sit3 mode sit ttl <ttldefault>
```

```
¬ <ipv4addressofforeigntunnel3> local <ipv4addresslocal>
# /sbin/ip link set dev sit3 up
# /sbin/ip -6 route add <prefixtoroute3> dev sit3 metric 1
```

9.3.1.2. Usando "ifconfig" e "route" (obsoleto)

Esta não é uma maneira muito recomendada de se criar um túnel porque ele é um pouco estranho. Não há qualquer problema se voce adicionar somente um, mas se voce for configurar mais de um, voce não poderá desfazer os primeiros túneis criados e deixar os últimos em funcionamento.

Uso (exemplo genérico para 3 túneis):

```
# /sbin/ifconfig sit0 up

# /sbin/ifconfig sit0 tunnel <ipv4addressofforeigntunnel1>
# /sbin/ifconfig sit1 up
# /sbin/route -A inet6 add <prefixtoroute1> dev sit1

# /sbin/ifconfig sit0 tunnel <ipv4addressofforeigntunnel2>
# /sbin/ifconfig sit2 up
# /sbin/route -A inet6 add <prefixtoroute2> dev sit2

# /sbin/ifconfig sit0 tunnel <ipv4addressofforeigntunnel3>
# /sbin/ifconfig sit3 up
# /sbin/route -A inet6 add <prefixtoroute3> dev sit3
```

Importante: NÃO USE ISSO, porque esta configuração habilita implicitamente um “tunelamento automático” de qualquer lugar na internet, e isto é um risco, então não o utilize.

9.3.1.3. Usando somente "route"

Também é possível configurar túneis no modo Non Broadcast Multiple Access (NBMA), pois é uma maneira fácil de adicionar muitos túneis de uma vez. Mas nenhum dos túneis pode ser numerado (o que não é uma característica).

Uso (exemplo genérico para 3 túneis):

```
# /sbin/ifconfig sit0 up

# /sbin/route -A inet6 add <prefixtoroute1> gw
¬ ::<ipv4addressofforeigntunnel1> dev sit0
# /sbin/route -A inet6 add <prefixtoroute2> gw
¬ ::<ipv4addressofforeigntunnel2> dev sit0
# /sbin/route -A inet6 add <prefixtoroute3> gw
¬ ::<ipv4addressofforeigntunnel3> dev sit0
```

Importante: NÃO USE ISSO, porque esta configuração habilita implicitamente um “tunelamento automático” de qualquer lugar na internet, e isto é um risco, então não o utilize.

9.3.2. Removendo os túneis ponto a ponto

Nem sempre é necessário fazer isso manualmente, mas pode ser usado em scripts para uma limpeza ou restart de uma configuração IPv6.

9.3.2.1. Usando "ip"

Uso para remover um dispositivo túnel:

```
# /sbin/ip tunnel del <device>
```

Uso (exemplo genérico para 3 túneis):

```
# /sbin/ip -6 route del <prefixtoroute1> dev sit1
# /sbin/ip link set sit1 down
# /sbin/ip tunnel del sit1

# /sbin/ip -6 route del <prefixtoroute2> dev sit2
# /sbin/ip link set sit2 down
# /sbin/ip tunnel del sit2

# /sbin/ip -6 route del <prefixtoroute3> dev sit3
# /sbin/ip link set sit3 down
# /sbin/ip tunnel del sit3
```

9.3.2.2. Usando "ifconfig" e "route" (não é mais usado por ser estranho)

Não somente a criação é estranha, como também o shutdown... voce tem que remover os tuneis na ordem inversa em que eles foram criados.

Uso (exemplo genérico para 3 túneis):

```
# /sbin/route -A inet6 del <prefixtoroute3> dev sit3
# /sbin/ifconfig sit3 down

# /sbin/route -A inet6 del <prefixtoroute2> dev sit2
# /sbin/ifconfig sit2 down

# /sbin/route -A inet6 add <prefixtoroute1> dev sit1
# /sbin/ifconfig sit1 down

# /sbin/ifconfig sit0 down
```

9.3.2.3. Usando "route"

Isto é como remover rotas normais em IPv6.

Uso (exemplo genérico para 3 túneis):

```
# /sbin/route -A inet6 del <prefixtoroute1> gw
¬ ::<ipv4addressofforeigntunnel1> dev sit0
# /sbin/route -A inet6 del <prefixtoroute2> gw
¬ ::<ipv4addressofforeigntunnel2> dev sit0
# /sbin/route -A inet6 del <prefixtoroute3> gw
¬ ::<ipv4addressofforeigntunnel3> dev sit0
```

```
# /sbin/ifconfig sit0 down
```

9.3.3. Tuneis ponto a ponto numerados

Às vezes é necessário configurar túneis ponto a ponto com endereços IPv6 como em IPv4. Isto somente é possível utilizando o primeiro (ifconfig+route - obsoleto) e o terceiro (ip+route) modo de configuração. Em tais casos, voce pode adicionar os endereços IPv6 na interface de túnel conforme é mostrado na configuração de uma interface.

9.4. Configuração de túneis 6to4

Preste atenção pois o suporte aos túneis 6to4 atualmente não está implementado completamente no kernel vanilla da série 2.2.x (veja systemcheck/kernel para mais detalhes). Veja também que o tamanho do prefixo para um endereço 6to4 é 16, uma vez que do ponto de vista da rede, todos os outros hosts habilitados para 6to4 estão na mesma camada 2.

9.4.1. Adição de um túnel 6to4

Antes de tudo, voce precisa calcular o seu prefixo 6to4 usando o seu endereço público IPv4 (se seu host não tem este endereço, é possível utilizar um NAT no router de borda em alguns casos especiais):

Assumindo que o seu endereço IPv4 seja este

```
1.2.3.4
```

o prefixo 6to4 gerado será este

```
2002:0102:0304::
```

Gateways locais 6to4 deveriam (mas não é uma regra fixa, pois voce pode escolher um sufixo arbitrário, se voce preferir) sempre assinalar o sufixo "::1", desta forma seu endereço local 6to4 será este

```
2002:0102:0304::1
```

Por exemplo, use a seguinte automação:

```
ipv4="1.2.3.4"; printf "2002:%02x%02x:%02x%02x::1" `echo $ipv4 | tr "." " "`
```

Atualmente existem duas maneiras possíveis de configurar um túnel 6to4.

9.4.1.1. Usando "ip" e um dispositivo de túnel dedicado

Esta é a maneira recomendada (um TTL deve ser especificado, pois o valor default é 0).

Criando um dispositivo de túnel

```
# /sbin/ip tunnel add tun6to4 mode sit ttl <ttldefault> remote any local
¬ <localipv4address>
```

Ativando a interface

```
# /sbin/ip link set dev tun6to4 up
```

Adicionando o endereço local 6to4 na interface (nota: o tamanho do prefixo - /16 - é importante!)

```
# /sbin/ip -6 addr add <local6to4address>/16 dev tun6to4
```

Adicionando uma rota default para a rede global IPv6 usando o endereço anycast IPv4 todos-6to4-router

```
# /sbin/ip -6 route add 2000::/3 via ::192.88.99.1 dev tun6to4 metric 1
```

É sabido que algumas versões do comando "ip" (exemplo SuSE Linux 9.0) não suportam endereços IPv6 no formato compatível IPv4 para seus gateways, e neste caso o endereço IPv6 relativo a ele deve ser usado:

```
# /sbin/ip -6 route add 2000::/3 via 2002:c058:6301::1 dev tun6to4 metric 1
```

9.4.1.2. Usando "ifconfig" e "route" e um dispositivo de túnel genérico "sit0" (obsoleto)

Este método está obsoleto porque o uso de um túnel genérico sit0 não permite especificar filtros pelo dispositivo.

Ativando a interface genérica sit0

```
# /sbin/ifconfig sit0 up
```

Adicionando um endereço 6to4 na interface

```
# /sbin/ifconfig sit0 add <local6to4address>/16
```

Adicionando uma rota default para a rede global IPv6 usando o endereço anycast IPv4 todos-6to4-router

```
# /sbin/route -A inet6 add 2000::/3 gw ::192.88.99.1 dev sit0
```

9.4.2. Removendo um túnel 6to4

9.4.2.1. Usando "ip" e um dispositivo de túnel dedicado

Remova todas as rotas que utilizam este dispositivo

```
# /sbin/ip -6 route flush dev tun6to4
```

Desligue a interface

```
# /sbin/ip link set dev tun6to4 down
```

Remova o dispositivo criado

```
# /sbin/ip tunnel del tun6to4
```

9.4.2.2. Usando "ifconfig" e "route" e o dispositivo genérico de túnel "sit0" (obsoleto)

Remova as rotas default que usam esta interface

```
# /sbin/route -A inet6 del 2000::/3 gw ::192.88.99.1 dev sit0
```

Remova o endereço local 6to4 desta interface

```
# /sbin/ifconfig sit0 del <local6to4address>/16
```

Desligue o dispositivo genérico de túnel (cuidado com isto, pois ela ainda pode estar em uso...)

```
# /sbin/ifconfig sit0 down
```

Chapter 10. Configurando túneis IPv4-in-IPv6

A RFC 2473 / Generic Packet Tunneling in IPv6 Specification (http://www.faqs.org/rfcs/rfc2473.html) especifica o mecanismo para tunelar diferentes tipos de pacotes em IPv6 incluindo IPv4.

NOTAE: O suporte para túneis IPv4-in-IPv6 está disponível somente a partir da versão de kernel 2.6.22.

10.1. Mostrando os túneis existentes

Uso:

```
# /sbin/ip -6 tunnel show [<device>]
```

Exemplo:

```
# /sbin/ip -6 tunnel show mode any
ip6tnl0: ipv6/ipv6 remote :: local :: encaplimit 0 hoplimit 0 tclass 0x00
¬ flowlabel 0x00000 (flowinfo 0x00000000)
ip6tnl1: ip/ipv6 remote fd00:0:0:2::a local fd00:0:0:2::1 dev eth1 encaplimit 4
¬ hoplimit 64 tclass 0x00 flowlabel 0x00000 (flowinfo 0x00000000)
```

NOTA: Se voce não incluir "mode any", somente os túneis IPv6-in-IPv6 serão mostrados.

10.2. Configuração de túnel ponto a ponto

Uso para criar um dispositivo de túnel 4over6

```
# /sbin/ip tunnel add <device> mode ip4ip6 remote <ipv6addressofforeigntunnel>
¬ local <ipv6addresslocal>
```

Uso (exemplo genérico para 3 túneis):

```
# /sbin/ip -6 tunnel add ip6tnl1 mode ip4ip6 remote
¬ <ipv6addressofforeigntunnel1> local <ipv6addresslocal>
# /sbin/ip link set dev ip6tnl1 up
# /sbin/ip -6 route add <prefixtoroute1> dev ip6tnl1 metric 1

# /sbin/ip -6 tunnel add ip6tnl2 mode ip4ip6 remote
¬ <ipv6addressofforeigntunnel2> local <ipv6addresslocal>
# /sbin/ip link set dev ip6tnl2 up
# /sbin/ip -6 route add <prefixtoroute2> dev ip6tnl2 metric 1

# /sbin/ip -6 tunnel add ip6tnl3 mode ip4ip6 remote
¬ <ipv6addressofforeigntunnel3> local <ipv6addresslocal>
# /sbin/ip link set dev ip6tnl3 up
# /sbin/ip -6 route add <prefixtoroute3> dev ip6tnl3 metric 1
```

10.3. Removendo túneis ponto a ponto

Uso ara remover um dispositivo de túnel:

```
# /sbin/ip -6 tunnel del <device>
```

Uso (exemplo genérico para 3 túneis):

```
# /sbin/ip -6 route del <prefixtoroute1> dev ip6tnl1
# /sbin/ip link set ip6tnl1 down
# /sbin/ip -6 tunnel del ip6tnl1

# /sbin/ip -6 route del <prefixtoroute2> dev ip6tnl2
# /sbin/ip link set ip6tnl2 down
# /sbin/ip -6 tunnel del ip6tnl2

# /sbin/ip -6 route del <prefixtoroute3> dev ip6tnl3
# /sbin/ip link set ip6tnl3 down
# /sbin/ip -6 tunnel del ip6tnl3
```

Chapter 11. Configuraçoes de Kernel nos arquivos do /proc

Nota: a fonte desta seção é em sua maioria o arquivo "ip-sysctl.txt", o qual está incluído no diretório "Documentation/networking" do código fonte do kernel usado. Crédito para Pekka Savola pela manutençao da parte IPv6 neste arquivo. Além disso, muito do que está escrito abaixo é um Copy & Paste deste arquivo mencionado.

11.1. Como acessar os arquivos do /proc

11.1.1. Usando "cat" e "echo"

Usar os comandos "cat" e "echo" é a maneira mais simples de acessar os arquivos deste diretório, mas alguns pontos devem ser observados:

- O sistema de arquivos /proc deve estar habilitado no kernel , ou seja, a seguinte chave deve estar configurada

```
CONFIG_PROC_FS=y
```

- O sistema de arquivos já deve estar montado, o que pode ser testado como comando

```
# mount | grep "type proc"
none on /proc type proc (rw)
```

- Voce pode ler e também escrever (geralmente como root) nos arquivos contidos aqui (/proc)

Normalmente, somente as entradas localizadas em /proc/sys/* podem ser alteradas, as demais são somentepara leitura e para obtenção de informações.

11.1.1.1. Obtendo um valor

O valor de uma entrada pode ser obtido com o comando "cat":

```
# cat /proc/sys/net/ipv6/conf/all/forwarding
0
```

11.1.1.2. Definindo um valor

Um novo valor pode ser definido (se a entrada aceitar a escrita) através do comando "echo":

```
# echo "1" >/proc/sys/net/ipv6/conf/all/forwarding
```

11.1.2. Usando "sysctl"

O uso do programa "syysctl" para acessar as chaves do kernel é uma maneira moderna utilizada hoje em dia. Voce também pode usar se o sistema de arquivos /proc não estiver montado. Mas voce só terá acesso às entradas /proc/sys/*!

O programa "sysctl" está incluído no pacote "procps" (em sistemas Red Hat Linux).

- A interface do sysctl deve estar habilitada no kernel, então a seguinte chave deve estar habilitada

```
CONFIG_SYSCTL=y
```

11.1.2.1. Obtendo um valor

O valor de uma entrada pode ser obtida da seguinte maneira:

```
# sysctl net.ipv6.conf.all.forwarding
net.ipv6.conf.all.forwarding = 0
```

11.1.2.2. Definindo um valor

Um novo valor pode ser definido (se a entrada aceitar a escrita):

```
# sysctl -w net.ipv6.conf.all.forwarding=1
net.ipv6.conf.all.forwarding = 1
```

Nota: Não use espaços entre o sinal = para definir os valores. Se forem possíveis diversos valores, coloque-os entre aspas

```
# sysctl -w net.ipv4.ip_local_port_range="32768 61000"
net.ipv4.ip_local_port_range = 32768 61000
```

11.1.2.3. Adicionais

Nota: Existem versões em campo que mostram "/" ao invés de "."

Para mais detalhes, de uma olhada na manpage do sysctl.

Dica: para achar mais rapidamente as definiçoes, use a opçao "-a" (mostra todas as entradas) junto com o comando "grep".

11.1.3. Valores encontrados nas entradas /proc

Existem vários formatos vistos no sistema de arquivos /proc:

- BOOLEANO: simplesmente um "0" (falso) ou um "1" (verdadeiro)
- INTEIRO: um valor inteiro, também pode ser sem sinal
- linhas mais sofisticadas com muitos valores: às vezes uma linha de cabeçalho também é mostrada, senão, uma olhada no código fonte do kernel pode ser necessário para entender o significado dos valores apresentados...

11.2. Entradas em /proc/sys/net/ipv6/

11.2.1. conf/default/*

Muda as configurações específicas da interface.

11.2.2. conf/all/*

Muda todas as configurações esoecíficas da interface.

Exceção: "conf/all/forwarding" tem um significado diferente aqui

11.2.2.1. conf/all/forwarding

- Tipo: BOOLEANO

Isto habilita o encaminhamento de pacotes IPv6 entre todas as interfaces.

Em IPv6 voce não pode controlar o envio de pacotes por dispositivo. este encaminhamento deve ser feito através de regras do IPv6-netfilter (controlado com ip6tables) e especificando os dispositivos de entrada e saída (veja Firewalling/Netfilter6 para mais detalhes). Isto é diferente no IPv4, onde é possível controlar o encaminhamento por dispositivo (a decisão é feita na interface onde o pacote entra).

Isto também define todas as interfaces do host e/ou router para o valor especificado. Veja os detalhes abaixo. Esta definição refere-se ao encaminhamento global.

Se este valor é 0, nenhum pacote IPv6 é encaminhado, os pacotes nunca deixarão outra interface, seja ela física ou lógica, como os túneis.

11.2.3. conf/interface/*

Muda configurações especiais por interface.

O comportamento funcional de certos ajustes é diferente, dependendo se o encaminhamento local está habilitado ou não.

11.2.3.1. accept_ra

- Tipo: BOOLEAN
- Ajuste default: habilitado se o encaminhamento local está desabilitado, e desabilitado se o encaminhamento local está habilitado.

Aceita Router Advertisements, e configura automaticamente esta interface com os dados recebidos.

11.2.3.2. accept_redirects

- Tipi: BOOLEAN

- Ajuste default: habilitado se o encaminhamento local está desabilitado, e desabilitado se o encaminhamento local está habilitado.

Aceita os Redirects enviados por um router IPv6.

11.2.3.3. autoconf

- Tipo: BOOLEAN
- Ajuste default: habilitado se o accept_ra_pinfo está babilitado, e desabilitado se o accept_ra_pinfo estiver desabilitado.

Os endereços e prefixos usados na configuração automática proveem de anúncios dos routers.

11.2.3.4. dad_transmits

- Tipo: INTEGER
- Default: 1

Quantidade de probes de detecção de endereços duplicados enviados.

11.2.3.5. forwarding

- Tipo: BOOLEAN
- Default: FALSE se o encaminhamento global estiver desabilitado (default), caso contrário TRUE

Comportamento Host/Router específico.

Nota: É recomendado ter a mesma configuração em todas as interfaces; cenários diferentes são bem incomuns.

- Valor FALSE: Por default, o comportamento do Host é assumido. Isto significa:

  1. O flag IsRouter não está definido em aúncios de vizinhança.
  2. Solicitaçoes Router são enviados quando necessário.
  3. Se accept_ra é TRUE (default), aceita anúncios router (e fazem auti configuração).
  4. Se accept_redirects é TRUE (default), aceita Redirects.

- Valor TRUE: se o encaminhamento local está habilitado, o comportamento Router é assumido. Isto significa que o contrário da lista acima pode acontecer:

  1. O flag IsRouter é definido nos anúncios de vizinhança.
  2. Solicitaçoes Router não são enviadas.
  3. Anúncios Router são ignorados.
  4. Redirects são ignorados.

11.2.3.6. hop_limit

- Tipo: INTEGER
- Default: 64

Hop Limit default para definir.

11.2.3.7. mtu

- Tipo: INTEGER
- Default: 1280 (mínimo necessário no IPv6)

Default Maximum Transfer Unit

11.2.3.8. router_solicitation_delay

- Tipo: INTEGER
- Default: 1

Número de segundos a esperar após a interface ser ativada antes de enviar solicitaçoes Router.

11.2.3.9. router_solicitation_interval

- Tipo: INTEGER
- Default: 4

Número de segundos a esperar entre solicitaçoes Router.

11.2.3.10. router_solicitations

- Tipo: INTEGER
- Default: 3

Número de solicitaçoes Router a enviar até assumir que não há um router presente.

11.2.4. neigh/default/*

Muda as definiçoes default para detecçao de vizinhos e alguns valores globais e de limites:

11.2.4.1. gc_thresh1

- Tipo: INTEGER
- Default: 128

Precisa ser preenchido.

11.2.4.2. gc_thresh2

- Tipo: INTEGER
- Default: 512

Precisa ser preenchido.

11.2.4.3. gc_thresh3

- Tipo: INTEGER
- Default: 1024

Parametro para o tamanho da tabela de vizinhança.

Aumente este valor se voce tem muitas interfaces e os routers começam a apresentar problemas misteriosos de funcionamento e falhas. Ou se uma mensagem dessas aparecer Zebra (routing daemon) (http://www.zebra.org/):

```
ZEBRA: netlink-listen error: No buffer space available, type=RTM_NEWROUTE(24),
¬ seq=426, pid=0
```

11.2.4.4. gc_interval

- Tipo: INTEGER
- Default: 30

Precisa ser preenchido.

11.2.5. neigh/interface/*

Muda ajustes especiais por interface para detecção de vizinhos.

11.2.5.1. anycast_delay

- Tipo: INTEGER
- Default: 100

Precisa ser preenchido.

11.2.5.2. gc_stale_time

- Tipo: INTEGER
- Default: 60

Precisa ser preenchido.

11.2.5.3. proxy_qlen

- Tipo: INTEGER
- Default: 64

Precisa ser preenchido.

11.2.5.4. unres_qlen

- Tipo: INTEGER
- Default: 3

Precisa ser preenchido.

11.2.5.5. app_solicit

- Tipo: INTEGER
- Default: 0

Precisa ser preenchido.

11.2.5.6. locktime

- Tipo: INTEGER
- Default: 0

Precisa ser preenchido.

11.2.5.7. retrans_time

- Tipo: INTEGER
- Default: 100

Precisa ser preenchido.

11.2.5.8. base_reachable_time

- Tipo: INTEGER
- Default: 30

Precisa ser preenchido.

11.2.5.9. mcast_solicit

- Tipo: INTEGER
- Default: 3

Precisa ser preenchido.

11.2.5.10. ucast_solicit

- Tipo: INTEGER
- Default: 3

Precisa ser preenchido

11.2.5.11. delay_first_probe_time

- Tipo: INTEGER
- Default: 5

Precisa ser preenchido.

11.2.5.12. proxy_delay

- Tipo: INTEGER
- Default: 80

Precisa ser preenchido.

11.2.6. route/*

Ajustes globais para roteamento.

11.2.6.1. flush

Removido nas novas versões de kernel - Precisa ser preenchido.

11.2.6.2. gc_interval

- Tipo: INTEGER
- Default: 30

Precisa ser preenchido.

11.2.6.3. gc_thresh

- Tipo: INTEGER
- Default: 1024

Precisa ser preenchido.

11.2.6.4. mtu_expires

- Tipo: INTEGER
- Default: 600

Precisa ser preenchido.

11.2.6.5. gc_elasticity

- Tipo: INTEGER
- Default: 0

Precisa ser preenchido.

11.2.6.6. gc_min_interval

- Tipo: INTEGER
- Default: 5

Precisa ser preenchido.

11.2.6.7. gc_timeout

- Tipo: INTEGER
- Default: 60

Precisa ser preenchido.

11.2.6.8. min_adv_mss

- Tipo: INTEGER
- Default: 12

Precisa ser preenchido.

11.2.6.9. max_size

- Tipo: INTEGER
- Default: 4096

Precisa ser preenchido.

11.3. Entradas relacionadas a IPv6 em /proc/sys/net/ipv4/

Neste momento (e será até que o IPv6 seja completamente convertido para um módulo independente do kernel) algumas chaves para IPv6 são usadas aqui.

11.3.1. ip_*

11.3.1.1. ip_local_port_range

Esta definição também é usada para o IPv6.

11.3.2. tcp_*

Esta definição também é usada para o IPv6.

11.3.3. icmp_*

Esta definição não é usada para o IPv6. Para habilitar o limite ICMPv6 (o que é muito recomendado) regras netfilter-v6 devem ser usadas.

11.3.4. others

Desconhecido, mas provavelmente não usado pelo IPv6.

11.4. Entradas em /proc/net relacionadas com IPv6

No /proc/net existem diversas variáveis disponíveis, somente para leitura, Não é possível obter informaçoes através do "sysctl", então utilize "cat".

11.4.1. if_inet6

- Tipo: Uma linha por endereço contendo vários valores

Aqui todos os endereços IPv6 cpnfigurados são mostrados em um formato especial. O exemplo mostra somente a interface de loopback. O significado é mostrado abaixo (veja "net/ipv6/addrconf.c" para mais informações).

```
# cat /proc/net/if_inet6
00000000000000000000000000000001 01 80 10 80 lo
+--------------------------------+ ++ ++ ++ ++ ++
|                                  |  |  |  |  |
1                                  2  3  4  5  6
```

1. Endereço IPv6 mostrado em hexadecimal (32 caracteres) sem os dois pontos ":" como separadores
2. Número do dispositivo netlink (índice da interface) em hexadecimal (veja “ip addr” , também)
3. tamanho do prefixo, em hexadecimal
4. Valor do escopo (veja o fonte do kernel " include/net/ipv6.h" e "net/ipv6/addrconf.c" para mais informações)
5. Flags da interface (veja "include/linux/rtnetlink.h" e"net/ipv6/addrconf.c" para mais informações)
6. Nome do disppositivo

11.4.2. ipv6_route

- Tipo: Uma linha por rota contém várias valores

Aqui toda a configuração de rotas em IPv6 é mostrada em um formato especialt. O exemplo mostra informações somente para a interface de loopback O significado é mostrado abaixo (veja "net/ipv6/route.c" para mais informações).

```
# cat /proc/net/ipv6_route
00000000000000000000000000000000 00 00000000000000000000000000000000 00
+--------------------------------+ ++ +--------------------------------+ ++
|                                  |  |                                  |
1                                  2  3                                  4

¬ 00000000000000000000000000000000 ffffffff 00000001 00000001 00200200 lo
¬ +--------------------------------+ +------+ +------+ +------+ +------+ ++
¬ |                                  |        |        |        |        |
¬ 5                                  6        7        8        9        10
```

1. Rede de destino IPv6 mostrada em hexadecimal (32 caracteres) som dois pontos ":" como separador
2. Tamanho do prefixo de destino, em hexadecimal
3. Rede de origem IPv6 mostrada em hexadecimal (32 caracteres) som dois pontos ":" como separador
4. Tamanho do prefixo de origem, em hexadecimal
5. Próximo salto IPv6 mostrado em hexadecimal (32 caracteres) som dois pontos ":" como separador
6. Métrica em hexadecimal

7. Contador de referencia
8. Contadoer de uso
9. Flags
10. Nome do dispositivo

11.4.3. sockstat6

- Tipo: Uma linha por protocolo, com descrição e valor

Estatísticas sobre o uso de sockets IPv6. Exemplo:

```
# cat /proc/net/sockstat6
TCP6: inuse 7
UDP6: inuse 2
RAW6: inuse 1
FRAG6: inuse 0 memory 0
```

11.4.4. tcp6

Precisa ser preenchido.

11.4.5. udp6

Precisa ser preenchido.

11.4.6. igmp6

Precisa ser preenchido.

11.4.7. raw6

Precisa ser preenchido.

11.4.8. ip6_flowlabel

Precisa ser preenchido.

11.4.9. rt6_stats

Precisa ser preenchido.

11.4.10. snmp6

- Tipo: Uma linha por descrição SNMP e valor

Estatísticas SNMP podem ser obtidas via um servidor SNMP e suas MIB's relacionadas, através um software de gerencia de rede.

11.4.11. ip6_tables_names

Tabelas netfilter6 disponíveis

Chapter 12. Netlink-Interface to kernel

To be filled...I have no experience with that...

Chapter 13. Address Resolver

Name to IPv4 or IPv6 address resolving is usually done using a libc resolver library. There are some issues known using the function *getaddrinfo*.

More info can be found at Linux & IPv6: getaddrinfo and search domains - Research (http://www.bieringer.de/linux/IPv6/getaddrinfo/) and RFC 3484 on Linux (http://people.redhat.com/drepper/linux-rfc3484.html).

More to be filled later...

Chapter 14. Network debugging

14.1. Server socket binding

14.1.1. Using “netstat” for server socket binding check

It’s always interesting which server sockets are currently active on a node. Using “netstat” is a short way to get such information:

Used options: -nlptu

Example:

```
# netstat -nlptu
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
¬ PID/Program name
tcp        0      0 0.0.0.0:32768           0.0.0.0:*               LISTEN
¬ 1258/rpc.statd
tcp        0      0 0.0.0.0:32769           0.0.0.0:*               LISTEN
¬ 1502/rpc.mountd
tcp        0      0 0.0.0.0:515             0.0.0.0:*               LISTEN
¬ 22433/lpd Waiting
tcp        0      0 1.2.3.1:139             0.0.0.0:*               LISTEN
¬ 1746/smbd
tcp        0      0 0.0.0.0:111             0.0.0.0:*               LISTEN
¬ 1230/portmap
tcp        0      0 0.0.0.0:6000            0.0.0.0:*               LISTEN
¬ 3551/X
tcp        0      0 1.2.3.1:8081            0.0.0.0:*               LISTEN
¬ 18735/junkbuster
tcp        0      0 1.2.3.1:3128            0.0.0.0:*               LISTEN
¬ 18822/(squid)
tcp        0      0 127.0.0.1:953           0.0.0.0:*               LISTEN
¬ 30734/named
tcp        0      0 ::ffff:1.2.3.1:993      :::*                    LISTEN
¬ 6742/xinetd-ipv6
tcp        0      0 :::13                   :::*                    LISTEN
¬ 6742/xinetd-ipv6
tcp        0      0 ::ffff:1.2.3.1:143      :::*                    LISTEN
¬ 6742/xinetd-ipv6
tcp        0      0 :::53                   :::*                    LISTEN
¬ 30734/named
tcp        0      0 :::22                   :::*                    LISTEN
¬ 1410/sshd
tcp        0      0 :::6010                 :::*                    LISTEN
¬ 13237/sshd
udp        0      0 0.0.0.0:32768           0.0.0.0:*
¬ 1258/rpc.statd
udp        0      0 0.0.0.0:2049            0.0.0.0:*
¬ -
udp        0      0 0.0.0.0:32770           0.0.0.0:*
¬ 1502/rpc.mountd
udp        0      0 0.0.0.0:32771           0.0.0.0:*
```

```
¬ -
udp        0      0 1.2.3.1:137             0.0.0.0:*
¬ 1751/nmbd
udp        0      0 0.0.0.0:137             0.0.0.0:*
¬ 1751/nmbd
udp        0      0 1.2.3.1:138             0.0.0.0:*
¬ 1751/nmbd
udp        0      0 0.0.0.0:138             0.0.0.0:*
¬ 1751/nmbd
udp        0      0 0.0.0.0:33044           0.0.0.0:*
¬ 30734/named
udp        0      0 1.2.3.1:53              0.0.0.0:*
¬ 30734/named
udp        0      0 127.0.0.1:53            0.0.0.0:*
¬ 30734/named
udp        0      0 0.0.0.0:67              0.0.0.0:*
¬ 1530/dhcpd
udp        0      0 0.0.0.0:67              0.0.0.0:*
¬ 1530/dhcpd
udp        0      0 0.0.0.0:32858           0.0.0.0:*
¬ 18822/(squid)
udp        0      0 0.0.0.0:4827            0.0.0.0:*
¬ 18822/(squid)
udp        0      0 0.0.0.0:111             0.0.0.0:*
¬ 1230/portmap
udp        0      0 :::53                   :::*
¬ 30734/named
```

14.2. Examples for tcpdump packet dumps

Here some examples of captured packets are shown, perhaps useful for your own debugging...

...more coming next...

14.2.1. Router discovery

14.2.1.1. Router advertisement

```
15:43:49.484751 fe80::212:34ff:fe12:3450 > ff02::1: icmp6: router
¬ advertisement(chlim=64, router_ltime=30, reachable_time=0,
¬ retrans_time=0)(prefix info: AR valid_ltime=30, preffered_ltime=20,
¬ prefix=2002:0102:0304:1::/64)(prefix info: LAR valid_ltime=2592000,
¬ preffered_ltime=604800, prefix=2001:0db8:0:1::/64)(src lladdr:
¬ 0:12:34:12:34:50) (len 88, hlim 255)
```

Router with link-local address “fe80::212:34ff:fe12:3450” send an advertisement to the all-node-on-link multicast address “ff02::1” containing two prefixes “2002:0102:0304:1::/64” (lifetime 30 s) and “2001:0db8:0:1::/64” (lifetime 2592000 s) including its own layer 2 MAC address “0:12:34:12:34:50”.

14.2.1.2. Router solicitation

```
15:44:21.152646 fe80::212:34ff:fe12:3456 > ff02::2: icmp6: router solicitation
¬ (src lladdr: 0:12:34:12:34:56) (len 16, hlim 255)
```

Node with link-local address “fe80::212:34ff:fe12:3456” and layer 2 MAC address “0:12:34:12:34:56” is looking for a router on-link, therefore sending this solicitation to the all-router-on-link multicast address “ff02::2”.

14.2.2. Neighbor discovery

14.2.2.1. Neighbor discovery solicitation for duplicate address detection

Following packets are sent by a node with layer 2 MAC address “0:12:34:12:34:56” during autoconfiguration to check whether a potential address is already used by another node on the link sending this to the solicited-node link-local multicast address.

- Node wants to configure its link-local address “fe80::212:34ff:fe12:3456”, checks for duplicate now

```
15:44:17.712338 :: > ff02::1:ff12:3456: icmp6: neighbor sol: who has
¬ fe80::212:34ff:fe12:3456(src lladdr: 0:12:34:12:34:56) (len 32, hlim 255)
```

- Node wants to configure its global address “2002:0102:0304:1:212:34ff:fe12:3456” (after receiving advertisement shown above), checks for duplicate now

```
15:44:21.905596 :: > ff02::1:ff12:3456: icmp6: neighbor sol: who has
¬ 2002:0102:0304:1:212:34ff:fe12:3456(src lladdr: 0:12:34:12:34:56) (len 32,
¬ hlim 255)
```

- Node wants to configure its global address “2001:0db8:0:1:212:34ff:fe12:3456” (after receiving advertisement shown above), checks for duplicate now

```
15:44:22.304028 :: > ff02::1:ff12:3456: icmp6: neighbor sol: who has
¬ 2001:0db8:0:1:212:34ff:fe12:3456(src lladdr: 0:12:34:12:34:56) (len 32, hlim
¬ 255)
```

14.2.2.2. Neighbor discovery solicitation for looking for host or gateway

- Node wants to send packages to “2001:0db8:0:1::10” but has no layer 2 MAC address to send packet, so send solicitation now

```
13:07:47.664538 2002:0102:0304:1:2e0:18ff:fe90:9205 > ff02::1:ff00:10: icmp6:
¬ neighbor sol: who has 2001:0db8:0:1::10(src lladdr: 0:e0:18:90:92:5) (len 32,
¬ hlim 255)
```

- Node looks for “fe80::10” now

```
13:11:20.870070 fe80::2e0:18ff:fe90:9205 > ff02::1:ff00:10: icmp6: neighbor
¬ sol: who has fe80::10(src lladdr: 0:e0:18:90:92:5) (len 32, hlim 255)
```

Chapter 15. Support for persistent IPv6 configuration in Linux distributions

Some Linux distribution contain already support of a persistent IPv6 configuration using existing or new configuration and script files and some hook in the IPv4 script files.

15.1. Red Hat Linux and “clones”

Since starting writing the IPv6 & Linux - HowTo (http://www.bieringer.de/linux/IPv6/) it was my intention to enable a persistent IPv6 configuration which catch most of the wished cases like host-only, router-only, dual-homed-host, router with second stub network, normal tunnels, 6to4 tunnels, and so on. Nowadays there exists a set of configuration and script files which do the job very well (never heard about real problems, but I don’t know how many use the set). Because this configuration and script files are extended from time to time, they got their own homepage: initscripts-ipv6 homepage (http://www.deepspace6.net/projects/initscripts-ipv6.html) (Mirror (http://mirrors.bieringer.de/www.deepspace6.net/projects/initscripts-ipv6.html)). Because I began my IPv6 experience using a Red Hat Linux 5.0 clone, my IPv6 development systems are mostly Red Hat Linux based now, it’s kind a logic that the scripts are developed for this kind of distribution (so called *historic issue*). Also it was very easy to extend some configuration files, create new ones and create some simple hook for calling IPv6 setup during IPv4 setup.

Fortunately, in Red Hat Linux since 7.1 a snapshot of my IPv6 scripts is included, this was and is still further on assisted by Pekka Savola.

Mandrake since version 8.0 also includes an IPv6-enabled initscript package, but a minor bug still prevents usage (“ifconfig” misses “inet6” before “add”).

15.1.1. Test for IPv6 support of network configuration scripts

You can test, whether your Linux distribution contain support for persistent IPv6 configuration using my set. Following script library should exist:

```
/etc/sysconfig/network-scripts/network-functions-ipv6
```

Auto-magically test:

```
# test -f /etc/sysconfig/network-scripts/network-functions-ipv6 && echo "Main
¬ IPv6 script library exists"
```

The version of the library is important if you miss some features. You can get it executing following (or easier look at the top of the file):

```
# source /etc/sysconfig/network-scripts/network-functions-ipv6 &&
¬ getversion_ipv6_functions
20011124
```

In shown example, the used version is 20011124. Check this against latest information on initscripts-ipv6 homepage (http://www.deepspace6.net/projects/initscripts-ipv6.html) (Mirror (http://mirrors.bieringer.de/www.deepspace6.net/projects/initscripts-ipv6.html)) to see what has been changed. You will find there also a change-log.

15.1.2. Short hint for enabling IPv6 on current RHL 7.1, 7.2, 7.3, ...

- Check whether running system has already IPv6 module loaded

```
# modprobe -c | grep net-pf-10
alias net-pf-10 off
```

- If result is “off”, then enable IPv6 networking by editing /etc/sysconfig/network, add following new line

```
NETWORKING_IPV6=yes
```

- Reboot or restart networking using

```
# service network restart
```

- Now IPv6 module should be loaded

```
# modprobe -c | grep ipv6
alias net-pf-10 ipv6
```

If your system is on a link which provides router advertisement, autoconfiguration will be done automatically. For more information which settings are supported see /usr/share/doc/initscripts-$version/sysconfig.txt.

15.2. SuSE Linux

In newer 7.x versions there is a really rudimentary support available, see /etc/rc.config for details.

Because of the really different configuration and script file structure it is hard (or impossible) to use the set for Red Hat Linux and clones with this distribution. In versions 8.x they completly change their configuration setup.

15.2.1. SuSE Linux 7.3

- How to setup 6to4 IPv6 with SuSE 7.3 (http://www.feyrer.de/IPv6/SuSE73-IPv6+6to4-setup.html)

15.2.2. SuSE Linux 8.0

15.2.2.1. IPv6 address configuration

Edit file /etc/sysconfig/network/ifcfg-<Interface-Name> and setup following value

```
IP6ADDR="<ipv6-address>/<prefix>"
```

15.2.2.2. Additional information

See file /usr/share/doc/packages/sysconfig/README

15.2.3. SuSE Linux 8.1

15.2.3.1. IPv6 address configuration

Edit file /etc/sysconfig/network/ifcfg-<Interface-Name> and setup following value

```
IPADDR="<ipv6-address>/<prefix>"
```

15.2.3.2. Additional information

See file /usr/share/doc/packages/sysconfig/Network

15.3. Debian Linux

Following information was contributed by Stephane Bortzmeyer <bortzmeyer at nic dot fr>

1. Be sure that IPv6 is loaded, either because it is compiled into the kernel or because the module is loaded. For the latest, three solutions, adding it to /etc/modules, using the pre-up trick shown later or using kmod (not detailed here).
2. Configure your interface. Here we assume eth0 and address (2001:0db8:1234:5::1:1). Edit /etc/network/interfaces:

```
iface eth0 inet6 static
        pre-up modprobe ipv6
        address 2001:0db8:1234:5::1:1
        # To suppress completely autoconfiguration:
        # up echo 0 > /proc/sys/net/ipv6/conf/all/autoconf
        netmask 64
        # The router is autoconfigured and has no fixed address.
        # It is magically
        # found. (/proc/sys/net/ipv6/conf/all/accept_ra). Otherwise:
        #gateway 2001:0db8:1234:5::1
```

And you reboot or you just

```
# ifup --force eth0
```

and you have your static address.

15.3.1. Further information

- IPv6 with Debian Linux (http://ipv6.debian.net/)

- Jean-Marc V. Liotier's HOWTO for Freenet6 & Debian Users (http://www.ruwenzori.net/ipv6/Jims_LAN_IPv6_global_connectivity_howto.html) (announced 24.12.2002 on mailinglist users@ipv6.org)

Chapter 16. Auto-configuration

16.1. Stateless auto-configuration

Is supported and seen on the assigned link-local address after an IPv6-enabled interface is up.

Example:

```
# ip -6 addr show dev eth0 scope link
2: eth0: <BROADCAST,MULTICAST,UP> mtu 1500 qlen1000
    inet6 fe80::211:d8ff:fe6b:f0f5/64 scope link
       valid_lft forever preferred_lft forever
```

16.2. Stateful auto-configuration using Router Advertisement Daemon (radvd)

to be filled. See radvd daemon autoconfiguration below.

16.3. Dynamic Host Configuration Protocol v6 (DHCPv6)

After a long time discussing issues, finally RFC 3315 / Dynamic Host Configuration Protocol for IPv6 (DHCPv6) (http://www.faqs.org/rfcs/rfc3315.html) was finished. At time updating this part (10/2005) currently two implementations are available:

- Dibbler (http://klub.com.pl/dhcpv6/) by Tomasz Mrugalski <thomson at klub dot com dot pl> (Hints for configuration)
- DHCPv6 on Sourceforge (http://dhcpv6.sourceforge.net/) (Hints for configuration)
- ISC DHCP (http://www.isc.org/software/dhcp) (Hints for configuration)

Chapter 17. Mobility

17.1. Common information

17.1.1. Node Mobility

Support for IPv6 mobility can be enabled in Linux by installing the MIPL2 implementation found at: http://www.mobile-ipv6.org/

This implementation is compliant with RFC 3775. It is composed of a kernel patch and a mobility daemon called mip6d. Version 2.0.1 applies on Linux kernel 2.6.15.

Installation and setup are described in the Linux Mobile IPv6 HOWTO (http://tldp.org/HOWTO/Mobile-IPv6-HOWTO/).

17.1.2. Network Mobility

There also exists an implementation of network mobility for Linux, it is called NEPL and is based on MIPL. It can also be downloaded from: http://www.mobile-ipv6.org/.

The HOWTO document describing setup and configuration is available at: http://www.nautilus6.org/doc/nepl-howto/.

17.1.3. Links

- Mobile IPv6 for Linux (MIPL) project: http://www.mobile-ipv6.org/
- Nautilus6 working group: http://nautilus6.org/
- Fast Handovers for Mobile IPv6 for Linux project: http://www.fmipv6.org/
- USAGI-patched Mobile IPv6 for Linux (UMIP):http://umip.linux-ipv6.org/
- Deploying IPsec/IKE-protected MIPv6 under Linux:http://natisbad.org/MIPv6/
- RFC 3775 / Mobility Support in IPv6 (http://www.faqs.org/rfcs/rfc3775.html)
- RFC 3776 / Using IPsec to Protect Mobile IPv6 Signaling Between Mobile Nodes and Home Agents (http://www.faqs.org/rfcs/rfc3776.html)
- RFC 3963 / Network Mobility (NEMO) (http://www.faqs.org/rfcs/rfc3963.html)
- RFC 4068 / Fast Handovers for Mobile IPv6 (http://www.faqs.org/rfcs/rfc4068.html)
- RFC 4423 / Host Identity Protocol (HIP) Architecture (http://www.faqs.org/rfcs/rfc4423.html)
- RFC 5201 / Host Identity Protocol (http://www.faqs.org/rfcs/rfc5201.html)
- HIP implementations: http://infrahip.hiit.fi/, http://hip4inter.net/, http://www.openhip.org/

Chapter 18. Firewalling

IPv6 firewalling is important, especially if using IPv6 on internal networks with global IPv6 addresses. Because unlike at IPv4 networks where in common internal hosts are protected automatically using private IPv4 addresses like RFC 1918 / Address Allocation for Private Internets (http://www.faqs.org/rfcs/rfc1918.html) or Automatic Private IP Addressing (APIPA)Google search for Microsoft + APIPA (http://www.google.com/search?q=apipa+microsoft), in IPv6 normally global addresses are used and someone with IPv6 connectivity can reach all internal IPv6 enabled nodes.

18.1. Firewalling using netfilter6

Native IPv6 firewalling is only supported in kernel versions 2.4+. In older 2.2- you can only filter IPv6-in-IPv4 by protocol 41.

Attention: no warranty that described rules or examples can really protect your system!

Audit your ruleset after installation, see Section 19.3 for more.

Since kernel version 2.6.20 IPv6 connection tracking is fully working (and does not break IPv4 NAT anymore like versions before)

18.1.1. More information

- Netfilter project (http://www.netfilter.org/)
- maillist archive of netfilter users (https://lists.netfilter.org/mailman/listinfo/netfilter)
- maillist archive of netfilter developers (https://lists.netfilter.org/mailman/listinfo/netfilter-devel)
- Unofficial status informations (http://www.bieringer.de/linux/IPv6/status/IPv6+Linux-status-kernel.html#netfilter6)

18.2. Preparation

This step is only needed if distributed kernel and netfilter doesn't fit your requirements and new features are available but still not built-in.

18.2.1. Get sources

Get the latest kernel source: http://www.kernel.org/

Get the latest iptables package:

- Source tarball (for kernel patches): http://www.netfilter.org/

18.2.2. Extract sources

Change to source directory:

```
# cd /path/to/src
```

Unpack and rename kernel sources

```
# tar z|jxf kernel-version.tar.gz|bz2
# mv linux linux-version-iptables-version+IPv6
```

Unpack iptables sources

```
# tar z|jxf iptables-version.tar.gz|bz2
```

18.2.3. Apply latest iptables/IPv6-related patches to kernel source

Change to iptables directory

```
# cd iptables-version
```

Apply pending patches

```
# make pending-patches KERNEL_DIR=/path/to/src/linux-version-iptables-version/
```

Apply additional IPv6 related patches (still not in the vanilla kernel included)

```
# make patch-o-matic KERNEL_DIR=/path/to/src/linux-version-iptables-version/
```

Say yes at following options (iptables-1.2.2)

- ah-esp.patch
- masq-dynaddr.patch (only needed for systems with dynamic IP assigned WAN connections like PPP or PPPoE)
- ipv6-agr.patch.ipv6
- ipv6-ports.patch.ipv6
- LOG.patch.ipv6
- REJECT.patch.ipv6

Check IPv6 extensions

```
# make print-extensions
Extensions found: IPv6:owner IPv6:limit IPv6:mac IPv6:multiport
```

18.2.4. Configure, build and install new kernel

Change to kernel sources

```
# cd /path/to/src/linux-version-iptables-version/
```

Edit Makefile

```
- EXTRAVERSION =
+ EXTRAVERSION = -iptables-version+IPv6-try
```

Run configure, enable IPv6 related

```
Code maturity level options
      Prompt for development and/or incomplete code/drivers : yes
Networking options
      Network packet filtering: yes
      The IPv6 protocol: module
           IPv6: Netfilter Configuration
                 IP6 tables support: module
                 All new options like following:
                       limit match support: module
                       MAC address match support: module
                       Multiple port match support: module
                       Owner match support: module
                       netfilter MARK match support: module
                       Aggregated address check: module
                       Packet filtering: module
                            REJECT target support: module
                            LOG target support: module
                       Packet mangling: module
                       MARK target support: module
```

Configure other related to your system, too

Compilation and installing: see the kernel section here and other HOWTOs

18.2.5. Rebuild and install binaries of iptables

Make sure, that upper kernel source tree is also available at /usr/src/linux/

Rename older directory

```
# mv /usr/src/linux /usr/src/linux.old
```

Create a new softlink

```
# ln -s /path/to/src/linux-version-iptables-version /usr/src/linux
```

Rebuild SRPMS

```
# rpm --rebuild /path/to/SRPMS/iptables-version-release.src.rpm
```

Install new iptables packages (iptables + iptables-ipv6)

- On RH 7.1 systems, normally, already an older version is installed, therefore use "freshen"

```
# rpm -Fhv /path/to/RPMS/cpu/iptables*-version-release.cpu.rpm
```

- If not already installed, use "install"

```
# rpm -ihv /path/to/RPMS/cpu/iptables*-version-release.cpu.rpm
```

- On RH 6.2 systems, normally, no kernel 2.4.x is installed, therefore the requirements don't fit. Use "--nodeps" to install it

```
# rpm -ihv --nodeps /path/to/RPMS/cpu/iptables*-version-release.cpu.rpm
```

Perhaps it's necessary to create a softlink for iptables libraries where iptables looks for them

```
# ln -s /lib/iptables/ /usr/lib/iptables
```

18.3. Usage

18.3.1. Check for support

Load module, if so compiled

```
# modprobe ip6_tables
```

Check for capability

```
# [ ! -f /proc/net/ip6_tables_names ] && echo "Current kernel doesn't support
¬ 'ip6tables' firewalling (IPv6)!"
```

18.3.2. Learn how to use ip6tables

18.3.2.1. List all IPv6 netfilter entries

- Short

```
# ip6tables -L
```

- Extended

```
# ip6tables -n -v --line-numbers -L
```

18.3.2.2. List specified filter

```
# ip6tables -n -v --line-numbers -L INPUT
```

18.3.2.3. Insert a log rule at the input filter with options

```
# ip6tables --table filter --append INPUT  -j LOG --log-prefix "INPUT:"
¬ --log-level 7
```

18.3.2.4. Insert a drop rule at the input filter

```
# ip6tables --table filter --append INPUT  -j DROP
```

18.3.2.5. Delete a rule by number

```
# ip6tables --table filter --delete INPUT 1
```

18.3.2.6. Enable connection tracking

Since kernel version 2.6.20 IPv6 connection tracking is well supported and should be used instead of using stateless filter rules.

```
# ip6tables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
```

18.3.2.7. Allow ICMPv6

Using older kernels (unpatched kernel 2.4.5 and iptables-1.2.2) no type can be specified

- Accept incoming ICMPv6 through tunnels

```
# ip6tables -A INPUT -i sit+ -p icmpv6 -j ACCEPT
```

- Allow outgoing ICMPv6 through tunnels

```
# ip6tables -A OUTPUT -o sit+ -p icmpv6 -j ACCEPT
```

Newer kernels allow specifying of ICMPv6 types:

```
# ip6tables -A INPUT -p icmpv6 --icmpv6-type echo-request -j ACCEPT
```

18.3.2.8. Rate-limiting

Because it can happen (author already saw it to times) that an ICMPv6 storm will raise up, you should use available rate limiting for at least ICMPv6 ruleset. In addition logging rules should also get rate limiting to prevent DoS attacks against syslog and storage of log file partition. An example for a rate limited ICMPv6 looks like:

```
# ip6tables -A INPUT --protocol icmpv6 --icmpv6-type echo-request
¬ -j ACCEPT --match limit --limit 30/minute
```

18.3.2.9. Allow incoming SSH

Here an example is shown for a ruleset which allows incoming SSH connection from a specified IPv6 address

- Allow incoming SSH from 2001:0db8:100::1/128

```
# ip6tables -A INPUT -i sit+ -p tcp -s 2001:0db8:100::1/128 --sport 512:65535
¬ --dport 22 -j ACCEPT
```

- Allow response packets (no longer needed if connection tracking is used!)

```
# ip6tables -A OUTPUT -o sit+ -p tcp -d 2001:0db8:100::1/128 --dport 512:65535
```

```
¬ --sport 22 ! --syn -j ACCEPT
```

18.3.2.10. Enable tunneled IPv6-in-IPv4

To accept tunneled IPv6-in-IPv4 packets, you have to insert rules in your IPv4 firewall setup relating to such packets, for example

- Accept incoming IPv6-in-IPv4 on interface ppp0

```
# iptables -A INPUT -i ppp0 -p ipv6 -j ACCEPT
```

- Allow outgoing IPv6-in-IPv4 to interface ppp0

```
# iptables -A OUTPUT -o ppp0 -p ipv6 -j ACCEPT
```

If you have only a static tunnel, you can specify the IPv4 addresses, too, like

- Accept incoming IPv6-in-IPv4 on interface ppp0 from tunnel endpoint 192.0.2.2

```
# iptables -A INPUT -i ppp0 -p ipv6 -s 192.0.2.2 -j ACCEPT
```

- Allow outgoing IPv6-in-IPv4 to interface ppp0 to tunnel endpoint 1.2.3.4

```
# iptables -A OUTPUT -o ppp0 -p ipv6 -d 192.0.2.2 -j ACCEPT
```

18.3.2.11. Protection against incoming TCP connection requests

VERY RECOMMENDED! For security issues you should really insert a rule which blocks incoming TCP connection requests. Adapt "-i" option, if other interface names are in use!

- Block incoming TCP connection requests to this host

```
# ip6tables -I INPUT -i sit+ -p tcp --syn -j DROP
```

- Block incoming TCP connection requests to hosts behind this router

```
# ip6tables -I FORWARD -i sit+ -p tcp --syn -j DROP
```

Perhaps the rules have to be placed below others, but that is work you have to think about it. Best way is to create a script and execute rules in a specified way.

18.3.2.12. Protection against incoming UDP connection requests

ALSO RECOMMENDED! Like mentioned on my firewall information it's possible to control the ports on outgoing UDP/TCP sessions. So if all of your local IPv6 systems are using local ports e.g. from 32768 to 60999 you are able to filter UDP connections also (until connection tracking works) like:

- Block incoming UDP packets which cannot be responses of outgoing requests of this host

```
# ip6tables -I INPUT -i sit+ -p udp ! --dport 32768:60999 -j DROP
```

- Block incoming UDP packets which cannot be responses of forwarded requests of hosts behind this router

```
# ip6tables -I FORWARD -i sit+ -p udp ! --dport 32768:60999 -j DROP
```

18.3.3. Examples

18.3.3.1. Simple example for Fedora

Following lines show a simple firewall configuration for Fedora 6 (since kernel version 2.6.20). It was modfied from the default one (generated by system-config-firewall) for supporting connection tracking and return the proper ICMPv6 code for rejects. Incoming SSH (port 22) connections are allowed.

File: /etc/sysconfig/ip6tables

```
*filter :INPUT ACCEPT [0:0]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
:RH-Firewall-1-INPUT - [0:0]
-A INPUT -j RH-Firewall-1-INPUT
-A FORWARD -j RH-Firewall-1-INPUT
-A RH-Firewall-1-INPUT -i lo -j ACCEPT
-A RH-Firewall-1-INPUT -p icmpv6 -j ACCEPT
-A RH-Firewall-1-INPUT -p 50 -j ACCEPT
-A RH-Firewall-1-INPUT -p 51 -j ACCEPT
-A RH-Firewall-1-INPUT -p udp --dport 5353 -d ff02::fb -j ACCEPT
-A RH-Firewall-1-INPUT -p udp -m udp --dport 631 -j ACCEPT
-A RH-Firewall-1-INPUT -p tcp -m tcp --dport 631 -j ACCEPT
-A RH-Firewall-1-INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
-A RH-Firewall-1-INPUT -m state --state NEW -p tcp --dport 22 -j ACCEPT
-A RH-Firewall-1-INPUT -j REJECT --reject-with icmp6-adm-prohibited
COMMIT
```

For completeness also the IPv4 configuration is shown here:

File: /etc/sysconfig/iptables

```
*filter :INPUT ACCEPT [0:0]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
:RH-Firewall-1-INPUT - [0:0]
-A INPUT -j RH-Firewall-1-INPUT
-A FORWARD -j RH-Firewall-1-INPUT
-A RH-Firewall-1-INPUT -i lo -j ACCEPT
-A RH-Firewall-1-INPUT -p icmp --icmp-type any -j ACCEPT
-A RH-Firewall-1-INPUT -p 50 -j ACCEPT
-A RH-Firewall-1-INPUT -p 51 -j ACCEPT
-A RH-Firewall-1-INPUT -p udp --dport 5353 -d 224.0.0.251 -j ACCEPT
-A RH-Firewall-1-INPUT -p udp -m udp --dport 631 -j ACCEPT
```

```
-A RH-Firewall-1-INPUT -p tcp -m tcp --dport 631 -j ACCEPT
-A RH-Firewall-1-INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
-A RH-Firewall-1-INPUT -m state --state NEW -m tcp -p tcp --dport 22 -j ACCEPT
-A RH-Firewall-1-INPUT -j REJECT --reject-with icmp-host-prohibited
COMMIT
```

Usage:

- Create/modify the configuration files
- Activate IPv4 & IPv6 firewalling

```
# service iptables start
# service ip6tables start
```

- Enable automatic start after reboot

```
# chkconfig iptables on
# chkconfig ip6tables on
```

18.3.3.2. Sophisticated example

Following lines show a more sophisticated but still stateless filter setup as an example. Happy netfilter6 ruleset creation....

```
# ip6tables -n -v -L
Chain INPUT (policy DROP 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
    0     0 extIN      all      sit+   *       ::/0                 ::/0
    4   384 intIN      all      eth0   *       ::/0                 ::/0
    0     0 ACCEPT     all      *      *       ::1/128              ::1/128
    0     0 ACCEPT     all      lo     *       ::/0                 ::/0
    0     0 LOG        all      *      *       ::/0                 ::/0
¬        LOG flags 0 level 7 prefix 'INPUT-default:'
    0     0 DROP       all      *      *       ::/0                 ::/0

Chain FORWARD (policy DROP 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
¬
    0     0 int2ext    all      eth0   sit+    ::/0                 ::/0
    0     0 ext2int    all      sit+   eth0    ::/0                 ::/0
    0     0 LOG        all      *      *       ::/0                 ::/0
¬        LOG flags 0 level 7 prefix 'FORWARD-default:'
    0     0 DROP       all      *      *       ::/0                 ::/0

Chain OUTPUT (policy DROP 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
¬
    0     0 extOUT     all      *      sit+    ::/0                 ::/0
    4   384 intOUT     all      *      eth0    ::/0                 ::/0
    0     0 ACCEPT     all      *      *       ::1/128              ::1/128
    0     0 ACCEPT     all      *      lo      ::/0                 ::/0
    0     0 LOG        all      *      *       ::/0                 ::/0
¬        LOG flags 0 level 7 prefix 'OUTPUT-default:'
```

```
    0     0 DROP       all      *      *       ::/0                ::/0

Chain ext2int (1 references)
 pkts bytes target     prot opt in     out     source               destination
¬
    0     0 ACCEPT     icmpv6    *      *       ::/0                 ::/0
    0     0 ACCEPT     tcp       *      *       ::/0                ::/0
¬        tcp spts:1:65535 dpts:1024:65535 flags:!0x16/0x02
    0     0 LOG        all       *      *       ::/0                ::/0
¬        LOG flags 0 level 7 prefix 'ext2int-default:'
    0     0 DROP       tcp       *      *       ::/0                ::/0
    0     0 DROP       udp       *      *       ::/0                ::/0
    0     0 DROP       all       *      *       ::/0                ::/0

Chain extIN (1 references)
 pkts bytes target     prot opt in     out     source               destination
¬
    0     0 ACCEPT     tcp       *      *       3ffe:400:100::1/128  ::/0
¬        tcp spts:512:65535 dpt:22
    0     0 ACCEPT     tcp       *      *       3ffe:400:100::2/128  ::/0
¬        tcp spts:512:65535 dpt:22
    0     0 ACCEPT     icmpv6    *      *       ::/0                 ::/0
    0     0 ACCEPT     tcp       *      *       ::/0                ::/0
¬        tcp spts:1:65535 dpts:1024:65535 flags:!0x16/0x02
    0     0 ACCEPT     udp       *      *       ::/0                ::/0
¬        udp spts:1:65535 dpts:1024:65535
    0     0 LOG        all       *      *       ::/0                ::/0
¬        limit: avg 5/min burst 5 LOG flags 0 level 7 prefix 'extIN-default:'
    0     0 DROP       all       *      *       ::/0                ::/0

Chain extOUT (1 references)
 pkts bytes target     prot opt in     out     source               destination
¬
    0     0 ACCEPT     tcp       *      *       ::/0
¬ 2001:0db8:100::1/128tcp spt:22 dpts:512:65535 flags:!0x16/0x02
    0     0 ACCEPT     tcp       *      *       ::/0
¬ 2001:0db8:100::2/128tcp spt:22 dpts:512:65535 flags:!0x16/0x02
    0     0 ACCEPT     icmpv6    *      *       ::/0                 ::/0
    0     0 ACCEPT     tcp       *      *       ::/0                ::/0
¬        tcp spts:1024:65535 dpts:1:65535
    0     0 ACCEPT     udp       *      *       ::/0                ::/0
¬        udp spts:1024:65535 dpts:1:65535
    0     0 LOG        all       *      *       ::/0                ::/0
¬        LOG flags 0 level 7 prefix 'extOUT-default:'
    0     0 DROP       all       *      *       ::/0                ::/0

Chain int2ext (1 references)
 pkts bytes target     prot opt in     out     source               destination
¬
    0     0 ACCEPT     icmpv6    *      *       ::/0                 ::/0
    0     0 ACCEPT     tcp       *      *       ::/0                ::/0
¬        tcp spts:1024:65535 dpts:1:65535
    0     0 LOG        all       *      *       ::/0                ::/0
¬        LOG flags 0 level 7 prefix 'int2ext:'
    0     0 DROP       all       *      *       ::/0                ::/0
    0     0 LOG        all       *      *       ::/0                ::/0
¬        LOG flags 0 level 7 prefix 'int2ext-default:'
```

```
    0     0 DROP       tcp      *      *       ::/0                ::/0
    0     0 DROP       udp      *      *       ::/0                ::/0
    0     0 DROP       all      *      *       ::/0                ::/0

Chain intIN (1 references)
 pkts bytes target     prot opt in     out     source              destination
¬
    0     0 ACCEPT     all      *      *       ::/0
¬ fe80::/ffc0::
    4   384 ACCEPT     all      *      *       ::/0                ff02::/16

Chain intOUT (1 references)
 pkts bytes target     prot opt in     out     source              destination
¬
    0     0 ACCEPT     all      *      *       ::/0
¬ fe80::/ffc0::
    4   384 ACCEPT     all      *      *       ::/0                ff02::/16
    0     0 LOG        all      *      *       ::/0                ::/0
¬        LOG flags 0 level 7 prefix 'intOUT-default:'
    0     0 DROP       all      *      *       ::/0                ::/0
```

Chapter 19. Security

19.1. Node security

It's very recommended to apply all available patches and disable all not necessary services. Also bind services to the needed IPv4/IPv6 addresses only and install local firewalling.

More to be filled...

19.2. Access limitations

Many services uses the tcp_wrapper library for access control. Below is described the use of tcp_wrapper.

More to be filled...

19.3. IPv6 security auditing

Currently there are no comfortable tools out which are able to check a system over network for IPv6 security issues. Neither Nessus (http://www.nessus.org/) nor any commercial security scanner is as far as I know able to scan IPv6 addresses.

19.3.1. Legal issues

ATTENTION: always take care that you only scan your own systems or after receiving a written order, otherwise legal issues are able to come up to you. CHECK destination IPv6 addresses TWICE before starting a scan.

19.3.2. Security auditing using IPv6-enabled netcat

With the IPv6-enabled netcat (see IPv6+Linux-status-apps/security-auditing (http://www.bieringer.de/linux/IPv6/status/IPv6+Linux-status-apps.html#security-auditing) for more) you can run a portscan by wrapping a script around which run through a port range, grab banners and so on. Usage example:

```
# nc6 ::1 daytime
13 JUL 2002 11:22:22 CEST
```

19.3.3. Security auditing using IPv6-enabled nmap

NMap (http://www.insecure.org/nmap/), one of the best portscaner around the world, supports IPv6 since version 3.10ALPHA1. Usage example:

```
# nmap -6 -sT ::1
Starting nmap V. 3.10ALPHA3 ( www.insecure.org/nmap/ )
Interesting ports on localhost6 (::1):
(The 1600 ports scanned but not shown below are in state: closed)
Port       State       Service
```

```
22/tcp     open        ssh
53/tcp     open        domain
515/tcp    open        printer
2401/tcp   open        cvspserver
Nmap run completed -- 1 IP address (1 host up) scanned in 0.525 seconds
```

19.3.4. Security auditing using IPv6-enabled strobe

Strobe is a (compared to NMap) more a low budget portscanner, but there is an IPv6-enabling patch available (see IPv6+Linux-status-apps/security-auditing (http://www.bieringer.de/linux/IPv6/status/IPv6+Linux-status-apps.html#security-auditing) for more). Usage example:

```
# ./strobe ::1 strobe 1.05 (c) 1995-1999 Julian Assange <proff@iq.org>.
::1 2401 unassigned unknown
::1 22 ssh Secure Shell - RSA encrypted rsh
::1 515 printer spooler (lpd)
::1 6010 unassigned unknown
::1 53 domain Domain Name Server
```

Note: strobe isn't really developed further on, the shown version number isn't the right one.

19.3.5. Audit results

If the result of an audit mismatch your IPv6 security policy, use IPv6 firewalling to close the holes, e.g. using netfilter6 (see Firewalling/Netfilter6 for more).

Info: More detailed information concerning IPv6 Security can be found here:

- IETF drafts - IPv6 Operations (v6ops) (http://www.ietf.org/ids.by.wg/v6ops.html)
- RFC 3964 / Security Considerations for 6to4 (http://www.faqs.org/rfcs/rfc3964.html)

Chapter 20. Encryption and Authentication

Unlike in IPv4, encryption and authentication is a mandatory feature of IPv6. Those features are normally implemented using IPsec (which can be also used by IPv4).

20.1. Modes of using encryption and authentication

Two modes of encryption and authentication of a connection are possible:

20.1.1. Transport mode

Transport mode is a real end-to-end connection mode. Here, only the payload (usually ICMP, TCP or UDP) is encrypted with their particular header, while the IP header is not encrypted (but usually included in authentication).

Using AES-128 for encryption and SHA1 for authentication, this mode decreases the MTU by 42 octets.

20.1.2. Tunnel mode

Tunnel mode can be used either for end-to-end or for gateway-to-gateway connection modes. Here, the complete IP packet is being encrypted and gets a new IP header prepended, all together constituing a new IP packet (this mechanism is also known as "encapsulation")

This mode usually decreases the MTU by 40 octets from the MTU of transport mode. I.e. using AES-128 for encryption and SHA1 for authentication 82 octets less than the normal MTU.

20.2. Support in kernel (ESP and AH)

20.2.1. Support in vanilla Linux kernel 2.4.x

At the time of writing missing in vanilla up to 2.4.28. There was an issue about keeping the Linux kernel source free of export/import-control-laws regarding encryption code. This is also one case why FreeS/WAN project (http://www.freeswan.org/) wasn't included in vanilla source. Perhaps a backport from 2.6.x will be done in the future.

20.2.2. Support in vanilla Linux kernel 2.6.x

Current versions (as time of writing 2.6.9 and upper) support native IPsec for IPv4 and IPv6.

Implementation was helped by the USAGI project.

20.3. Automatic key exchange (IKE)

IPsec requires a key exchange of a secret. This is mostly done automatically by so called IKE daemons. They also handle the authentication of the peers, either by a common known secret (so called “pre-shared secret”) or by RSA keys (which can also be used from X.509 certificates).

Currently, two different IKE daemons are available for Linux, which totally differ in configuration and usage.

I prefer “pluto” from the *S/WAN implementation because of the easier and one-config-only setup.

20.3.1. IKE daemon “racoon”

The IKE daemon “racoon” is taken from the KAME project and ported to Linux. Modern Linux distributions contain this daemon in the package “ipsec-tools”. Two executables are required for a proper IPsec setup. Take a look on Linux Advanced Routing & Traffic Control HOWTO / IPSEC (http://lartc.org/howto/lartc.ipsec.html), too.

20.3.1.1. Manipulation of the IPsec SA/SP database with the tool “setkey”

“setkey” is important to define the security policy (SP) for the kernel.

File: /etc/racoon/setkey.sh

- Example for an end-to-end encrypted connection in transport mode

```
#!/sbin/setkey -f
flush;
spdflush;
spdadd 2001:db8:1:1::1 2001:db8:2:2::2 any -P out ipsec esp/transport//require;
spdadd 2001:db8:2:2::2 2001:db8:1:1::1 any -P in  ipsec esp/transport//require;
```

- Example for a end-to-end encrypted connection in tunnel mode

```
#!/sbin/setkey -f
flush;
spdflush;
spdadd 2001:db8:1:1::1 2001:db8:2:2::2 any -P out ipsec
¬ esp/tunnel/2001:db8:1:1::1-2001:db8:2:2::2/require;
spdadd 2001:db8:2:2::2 2001:db8:1:1::1 any -P in  ipsec
¬ esp/tunnel/2001:db8:2:2::2-2001:db8:1:1::1/require;
```

For the other peer, you have to replace “in” with “out”.

20.3.1.2. Configuration of the IKE daemon “racoon”

“racoon” requires a configuration file for proper execution. It includes the related settings to the security policy, which should be set up previously using “setkey”.

File: /etc/racoon/racoon.conf

```
# Racoon IKE daemon configuration file.
# See 'man racoon.conf' for a description of the format and entries.
path include "/etc/racoon";
path pre_shared_key "/etc/racoon/psk.txt";
```

```
listen
{
        isakmp 2001:db8:1:1::1;
}

remote 2001:db8:2:2::2
{
        exchange_mode main;
        lifetime time 24 hour;
        proposal
        {
                encryption_algorithm 3des;
                hash_algorithm md5;
                authentication_method pre_shared_key;
                dh_group 2;
        }
}

# gateway-to-gateway
sainfo address 2001:db8:1:1::1 any address 2001:db8:2:2::2 any
{
        lifetime time 1 hour;
        encryption_algorithm 3des;
        authentication_algorithm hmac_md5;
        compression_algorithm deflate;
}

sainfo address 2001:db8:2:2::2 any address 2001:db8:1:1::1 any
{
        lifetime time 1 hour;
        encryption_algorithm 3des;
        authentication_algorithm hmac_md5;
        compression_algorithm deflate;
}
```

Also set up the pre-shared secret:

File: /etc/racoon/psk.txt

```
# file for pre-shared keys used for IKE authentication
# format is: 'identifier' 'key'

2001:db8:2:2::2 verysecret
```

20.3.1.3. Running IPsec with IKE daemon "racoon"

At least the daemon needs to be started. For the first time, use debug and foreground mode. The following example shows a successful IKE phase 1 (ISAKMP-SA) and 2 (IPsec-SA) negotiation:

```
# racoon -F -v -f /etc/racoon/racoon.conf
Foreground mode.
2005-01-01 20:30:15: INFO: @(#)ipsec-tools 0.3.3
¬ (http://ipsec-tools.sourceforge.net)
2005-01-01 20:30:15: INFO: @(#)This product linked
¬ OpenSSL 0.9.7a Feb 19 2003 (http://www.openssl.org/)
```

```
2005-01-01 20:30:15: INFO: 2001:db8:1:1::1[500] used as isakmp port (fd=7)
2005-01-01 20:31:06: INFO: IPsec-SA request for 2001:db8:2:2::2
¬ queued due to no phase1 found.
2005-01-01 20:31:06: INFO: initiate new phase 1 negotiation:
¬ 2001:db8:1:1::1[500]<=>2001:db8:2:2::2[500]
2005-01-01 20:31:06: INFO: begin Identity Protection mode.
2005-01-01 20:31:09: INFO: ISAKMP-SA established
¬ 2001:db8:1:1::1[500]-2001:db8:2:2::2[500] spi:da3d3693289c9698:ac039a402b2db401
2005-01-01 20:31:09: INFO: initiate new phase 2 negotiation:
¬ 2001:6f8:900:94::2[0]<=>2001:db8:2:2::2[0]
2005-01-01 20:31:10: INFO: IPsec-SA established:
¬ ESP/Tunnel 2001:db8:2:2::2->2001:db8:1:1::1 spi=253935531(0xf22bfab)
2005-01-01 20:31:10: INFO: IPsec-SA established:
¬ ESP/Tunnel 2001:db8:1:1::1->2001:db8:2:2::2 spi=175002564(0xa6e53c4)
```

Each direction got its own IPsec-SA (like defined in the IPsec standard). With “tcpdump” on the related interface, you will see as result of an IPv6 ping:

```
20:35:55.305707 2001:db8:1:1::1 > 2001:db8:2:2::2: ESP(spi=0x0a6e53c4,seq=0x3)
20:35:55.537522 2001:db8:2:2::2 > 2001:db8:1:1::1: ESP(spi=0x0f22bfab,seq=0x3)
```

As expected, the negotiated SPIs are being used here.

And using “setkey”, current active parameters are shown:

```
# setkey -D
2001:db8:1:1::1 2001:db8:2:2::2
        esp mode=tunnel spi=175002564(0x0a6e53c4) reqid=0(0x00000000)
        E: 3des-cbc  bd26bc45 aea0d249 ef9c6b89 7056080f 5d9fa49c 924e2edd
        A: hmac-md5  60c2c505 517dd8b7 c9609128 a5efc2db
        seq=0x00000000 replay=4 flags=0x00000000 state=mature
        created: Jan  1 20:31:10 2005   current: Jan  1 20:40:47 2005
        diff: 577(s)    hard: 3600(s)   soft: 2880(s)
        last: Jan  1 20:35:05 2005      hard: 0(s)      soft: 0(s)
        current: 540(bytes)     hard: 0(bytes)  soft: 0(bytes)
        allocated: 3    hard: 0 soft: 0
        sadb_seq=1 pid=22358 refcnt=0
2001:db8:2:2::2 2001:db8:1:1::1
        esp mode=tunnel spi=253935531(0x0f22bfab) reqid=0(0x00000000)
        E: 3des-cbc  c1ddba65 83debd62 3f6683c1 20e747ac 933d203f 4777a7ce
        A: hmac-md5  3f957db9 9adddc8c 44e5739d 3f53ca0e
        seq=0x00000000 replay=4 flags=0x00000000 state=mature
        created: Jan  1 20:31:10 2005   current: Jan  1 20:40:47 2005
        diff: 577(s)    hard: 3600(s)   soft: 2880(s)
        last: Jan  1 20:35:05 2005      hard: 0(s)      soft: 0(s)
        current: 312(bytes)     hard: 0(bytes)  soft: 0(bytes)
        allocated: 3    hard: 0 soft: 0
        sadb_seq=0 pid=22358 refcnt=0
```

20.3.2. IKE daemon “pluto”

The IKE daemon “pluto” is included in distributions of the *S/WAN projects. *S/WAN project starts at the beginning as FreeS/WAN (http://www.freeswan.org/). Unfortunately, the FreeS/WAN project stopped further development in 2004. Because of the slow pace of development in the past, two spin-offs started: strongSwan

(http://www.strongswan.org/) and Openswan (http://www.openswan.org/). Today, readily installable packages are available for at least Openswan (included in Fedora Core 3).

A major difference to "racoon", only one configuration file is required. Also, an initscript exists for automatic setup after booting.

20.3.2.1. Configuration of the IKE daemon "pluto"

The configuration is very similar to the IPv4 one, only one important option is necessary.

File: /etc/ipsec.conf

```
# /etc/ipsec.conf - Openswan IPsec configuration file
#
# Manual:     ipsec.conf.5
version 2.0     # conforms to second version of ipsec.conf specification

# basic configuration
config setup
        # Debug-logging controls:  "none" for (almost) none, "all" for lots.
        # klipsdebug=none
        # plutodebug="control parsing"

#Disable Opportunistic Encryption
include /etc/ipsec.d/examples/no_oe.conf

conn ipv6-p1-p2
        connaddrfamily=ipv6       # Important for IPv6!
        left=2001:db8:1:1::1
        right=2001:db8:2:2::2
        authby=secret
        esp=aes128-sha1
        ike=aes128-sha-modp1024
        type=transport
        #type=tunnel
        compress=no
        #compress=yes
        auto=add
        #auto=start
```

Don't forget to define the pre-shared secret here also.

File: /etc/ipsec.secrets

```
2001:db8:1:1::1 2001:db8:2:2::2 : PSK      "verysecret"
```

20.3.2.2. Running IPsec with IKE daemon "pluto"

If installation of Openswan was successfully, an initscript should exist for starting IPsec, simply run (on each peer):

```
# /etc/rc.d/init.d/ipsec start
```

Afterwards, start this connection on one peer. If you saw the line "IPsec SA established", all worked fine.

```
# ipsec auto --up ipv6-peer1-peer2
104 "ipv6-p1-p2" #1: STATE_MAIN_I1: initiate
```

```
106 "ipv6-p1-p2" #1: STATE_MAIN_I2: sent MI2, expecting MR2
108 "ipv6-p1-p2" #1: STATE_MAIN_I3: sent MI3, expecting MR3
004 "ipv6-p1-p2" #1: STATE_MAIN_I4: ISAKMP SA established
112 "ipv6-p1-p2" #2: STATE_QUICK_I1: initiate
004 "ipv6-p1-p2" #2: STATE_QUICK_I2: sent QI2,
¬ IPsec SA established {ESP=>0xa98b7710 <0xa51e1f22}
```

Because *S/WAN and setkey/racoon do use the same IPsec implementation in Linux 2.6.x kernel, “setkey” can be used here too to show current active parameters:

```
# setkey -D
2001:db8:1:1::1 2001:db8:2:2::2
        esp mode=transport spi=2844489488(0xa98b7710) reqid=16385(0x00004001)
        E: aes-cbc  082ee274 2744bae5 7451da37 1162b483
        A: hmac-sha1  b7803753 757417da 477b1c1a 64070455 ab79082c
        seq=0x00000000 replay=64 flags=0x00000000 state=mature
        created: Jan  1 21:16:32 2005   current: Jan  1 21:22:20 2005
        diff: 348(s)    hard: 0(s)      soft: 0(s)
        last:                           hard: 0(s)      soft: 0(s)
        current: 0(bytes)       hard: 0(bytes)  soft: 0(bytes)
        allocated: 0    hard: 0 soft: 0
        sadb_seq=1 pid=23825 refcnt=0
2001:db8:2:2::2 2001:db8:1:1::1
        esp mode=transport spi=2770214690(0xa51e1f22) reqid=16385(0x00004001)
        E: aes-cbc  6f59cc30 8d856056 65e07b76 552cac18
        A: hmac-sha1  c7c7d82b abfca8b1 5440021f e0c3b335 975b508b
        seq=0x00000000 replay=64 flags=0x00000000 state=mature
        created: Jan  1 21:16:31 2005   current: Jan  1 21:22:20 2005
        diff: 349(s)    hard: 0(s)      soft: 0(s)
        last:                           hard: 0(s)      soft: 0(s)
        current: 0(bytes)       hard: 0(bytes)  soft: 0(bytes)
        allocated: 0    hard: 0 soft: 0
        sadb_seq=0 pid=23825 refcnt=0
```

20.4. Additional informations:

On Linux Kernel 2.6.x you can get the policy and status of IPsec also using “ip”:

```
# ip xfrm policy
...

# ip xfrm state
...
```

Chapter 21. Quality of Service (QoS)

IPv6 supports QoS with use of Flow Labels and Traffic Classes.

Additional infos:

- RFC 3697 / IPv6 Flow Label Specification (http://www.faqs.org/rfcs/rfc3697.html)

21.1. General

Proper working QoS is only possible on the outgoing interface of a router or host, where the bottleneck begins. Everything else is a hickup and not subject to work as expected or has a successful result.

```
-------------->-------
           Queue 1    \
           --->---     ---->--------->--------->---------------
Big pipe   Queue 2      Queue 1 / Queue 2 / Queue 3   Thin Pipe
           --->----    ---->--------->--------->---------------
           Queue 3    /
--------------->-------
```

21.2. Linux QoS using “tc”

Linux is using “tc” from the “iproute2” package to configure traffic shaping, generally described in the Linux Advanced Routing & Traffic Control HOWTO (http://lartc.org/).

21.2.1. Example for a constant bitrate queuing

With the “cbq” scheduler, pipes with constant bit rates can be defined.

21.2.1.1. Root qdisc definition

Define root qdisc with a bandwidth of 1000 MBit/s on eth1

```
# tc qdisc add dev eth1 root handle 1: cbq avpkt 1000 bandwidth 1000Mbit
```

21.2.1.2. QoS class definition

Define a class 1:1 with 1 MBit/s

```
# tc class add dev eth1 parent 1: classid 1:1 cbq rate   1Mbit allot 1500 bounded
```

Define a class 1:2 with 50 MBit/s

```
# tc class add dev eth1 parent 1: classid 1:2 cbq rate  50Mbit allot 1500 bounded
```

Define a class 1:3 with 10 MBit/s

```
# tc class add dev eth1 parent 1: classid 1:3 cbq rate  10Mbit allot 1500 bounded
```

Define a class 1:4 with 200 kBit/s

```
# tc class add dev eth1 parent 1: classid 1:4 cbq rate 200kbit allot 1500 bounded
```

21.2.1.3. QoS filter definition

Define a filter for IPv4 (*protocol ip*), TCP (*match ip protocol 6 0xff*) destination port 5001 (*match ip dport 5001 0xffff*) using class 1:2 from above

```
# tc filter add dev eth1 parent 1: protocol ip   u32 match ip  protocol 6 0xff match ip dport 500
```

Define a filter for IPv6 (*protocol ip*), TCP (*match ip6 protocol 6 0xff*) destination port 5001 using class 1:2 from above

```
# tc filter add dev eth1 parent 1: protocol ipv6 u32 match ip6 protocol 6 0xff match ip6 dport 50
```

Define a filter for IPv6 for packets having flow label 12345 (*match ip6 flowlabel 12345 0x3ffff*) using class 1:3 from above

```
# tc filter add dev eth1 parent 1: protocol ipv6 u32 match ip6 flowlabel 12345 0x3ffff flowid 1:3
```

Define a filter for IPv6 for packets having Linux iptables mark 32 (*handle 32 fw*) specified using class 1:4 from above

```
# tc filter add dev eth1 parent 1: protocol ipv6 handle 32 fw flowid 1:4
```

The last filter definition requires an entry in the ip6tables to mark a packet

```
# ip6tables -A POSTROUTING -t mangle -p tcp --dport 5003 -j MARK --set-mark 32
```

21.2.1.4. Testing filter definitions using iperf

Start on server side each one one separate console:

```
# iperf -V -s -p 5001
# iperf -V -s -p 5002
# iperf -V -s -p 5003
```

Start on client side and compare results:

```
# iperf -V -c SERVER-IPv4 -p 5001    (expected:      1 MBit/s)
# iperf -V -c SERVER-IPv6 -p 5001    (expected:     50 MBit/s)
# iperf -V -c SERVER-IPv4 -p 5002    (expected:  >> 50 MBit/s && <= 1000 MBit/s)
# iperf -V -c SERVER-IPv6 -p 5002    (expected:  >> 50 MBit/s && <= 1000 MBit/s)
# iperf -V -c SERVER-IPv4 -p 5003    (expected:  >> 50 MBit/s && <= 1000 MBit/s)
# iperf -V -c SERVER-IPv6 -p 5003    (expected:    200 kBit/s)
```

The rate result should be as defined in the classes (see above), the results on port 5002 should be very similar independend from used IP version.

Chapter 22. Hints for IPv6-enabled daemons

Here some hints are shown for IPv6-enabled daemons.

22.1. Berkeley Internet Name Domain (BIND) daemon “named”

IPv6 is supported since version 9. Always use newest available version. At least version 9.1.3 must be used, older versions can contain remote exploitable security holes.

22.1.1. Listening on IPv6 addresses

Note: unlike in IPv4 current versions doesn’t allow to bind a server socket to dedicated IPv6 addresses, so only *any* or *none* are valid. Because this can be a security issue, check the Access Control List (ACL) section below, too!

22.1.1.1. Enable BIND named for listening on IPv6 address

To enable IPv6 for listening, following options are requested to change

```
options {
        # sure other options here, too
        listen-on-v6 { any; };
};
```

This should result after restart in e.g.

```
# netstat -lnptu |grep "named\W*$"
tcp 0 0 :::53         :::*      LISTEN 1234/named
¬ # incoming TCP requests
udp 0 0 1.2.3.4:53    0.0.0.0:*        1234/named
¬ # incoming UDP requests to IPv4 1.2.3.4
udp 0 0 127.0.0.1:53  0.0.0.0:*        1234/named
¬ # incoming UDP requests to IPv4 localhost
udp 0 0 0.0.0.0:32868 0.0.0.0:*        1234/named
¬ # dynamic chosen port for outgoing queries
udp 0 0 :::53         :::*             1234/named
¬ # incoming UDP request to any IPv6
```

And a simple test looks like

```
# dig localhost @::1
```

and should show you a result.

22.1.1.2. Disable BIND named for listening on IPv6 address

To disable IPv6 for listening, following options are requested to change

```
options {
        # sure other options here, too
```

```
        listen-on-v6 { none; };
};
```

22.1.2. IPv6 enabled Access Control Lists (ACL)

IPv6 enabled ACLs are possible and should be used whenever it's possible. An example looks like following:

```
acl internal-net {
        127.0.0.1;
        1.2.3.0/24;
        2001:0db8:100::/56;
        ::1/128;
        ::ffff:1.2.3.4/128;
};
acl ns-internal-net {
        1.2.3.4;
        1.2.3.5;
        2001:0db8:100::4/128;
        2001:0db8:100::5/128;
};
```

This ACLs can be used e.g. for queries of clients and transfer zones to secondary name-servers. This prevents also your caching name-server to be used from outside using IPv6.

```
options {
        # sure other options here, too
        listen-on-v6 { none; };
        allow-query { internal-net; };
        allow-transfer { ns-internal-net; };
};
```

It's also possible to set the *allow-query* and *allow-transfer* option for most of single zone definitions, too.

22.1.3. Sending queries with dedicated IPv6 address

This option is not required, but perhaps needed:

```
query-source-v6 address <ipv6address|*> port <port|*>;
```

22.1.4. Per zone defined dedicated IPv6 addresses

It's also possible to define per zone some IPv6 addresses.

22.1.4.1. Transfer source address

Transfer source address is used for outgoing zone transfers:

```
transfer-source-v6 <ipv6addr|*> [port port];
```

22.1.4.2. Notify source address

Notify source address is used for outgoing notify messages:

```
notify-source-v6 <ipv6addr|*> [port port];
```

22.1.5. IPv6 DNS zone files examples

Some information can be also found at IPv6 DNS Setup Information (article) (http://www.isi.edu/~bmanning/v6DNS.html). Perhaps also helpful is the IPv6 Reverse DNS zone builder for BIND 8/9 (webtool) (http://tools.fpsn.net/ipv6-inaddr/).

22.1.6. Serving IPv6 related DNS data

For IPv6 new types and root zones for reverse lookups are defined:

- AAAA and reverse IP6.INT: specified in RFC 1886 / DNS Extensions to support IP version 6 (http://www.faqs.org/rfcs/rfc1886.html), usable since BIND version 4.9.6
- A6, DNAME (DEPRECATED NOW!) and reverse IP6.ARPA: specified in RFC 2874 / DNS Extensions to Support IPv6 Address Aggregation and Renumbering (http://www.faqs.org/rfcs/rfc2874.html), usable since BIND 9, but see also an information about the current state at Domain Name System Extension (dnsext) (http://www.ietf.org/ids.by.wg/dnsext.html)

Perhaps filled later more content, for the meantime take a look at given RFCs and

- AAAA and reverse IP6.INT: IPv6 DNS Setup Information (http://www.isi.edu/~bmanning/v6DNS.html)
- A6, DNAME (DEPRECATED NOW!) and reverse IP6.ARPA: take a look into chapter 4 and 6 of the BIND 9 Administrator Reference Manual (ARM) distributed with the bind-package or get this here: BIND manual version 9.3 (http://www.isc.org/sw/bind/arm93/)

Because IP6.INT is deprecated (but still in use), a DNS server which will support IPv6 information has to serve both reverse zones.

22.1.6.1. Current best practice

Because there are some troubles around using the new formats, current best practice is:

Forward lookup support:

- AAAA

Reverse lookup support:

- Reverse nibble format for zone ip6.int (FOR BACKWARD COMPATIBILITY)
- Reverse nibble format for zone ip6.arpa (RECOMMENDED)

22.1.7. Checking IPv6-enabled connect

To check, whether BIND named is listening on an IPv6 socket and serving data see following examples.

22.1.7.1. IPv6 connect, but denied by ACL

Specifying a dedicated server for the query, an IPv6 connect can be forced:

```
$ host -t aaaa www.6bone.net 2001:0db8:200:f101::1
Using domain server:
Name: 2001:0db8:200:f101::1
Address: 2001:0db8:200:f101::1#53
Aliases:

Host www.6bone.net. not found: 5(REFUSED)
```

Related log entry looks like following:

```
Jan 3 12:43:32 gate named[12347]: client
¬ 2001:0db8:200:f101:212:34ff:fe12:3456#32770:
 query denied
```

If you see such entries in the log, check whether requests from this client should be allowed and perhaps review your ACL configuration.

22.1.7.2. Successful IPv6 connect

A successful IPv6 connect looks like following:

```
$ host -t aaaa www.6bone.net 2001:0db8:200:f101::1
Using domain server:
Name: 2001:0db8:200:f101::1
Address: 2001:0db8:200:f101::1#53
Aliases:

www.6bone.net. is an alias for 6bone.net.
6bone.net. has AAAA address 3ffe:b00:c18:1::10
```

22.2. Internet super daemon (xinetd)

IPv6 is supported since xinetd (http://www.xinetd.org/) version around 1.8.9. Always use newest available version. At least version 2.3.3 must be used, older versions can contain remote exploitable security holes.

Some Linux distribution contain an extra package for the IPv6 enabled xinetd, some others start the IPv6-enabled xinetd if following variable is set: NETWORKING_IPV6="yes", mostly done by /etc/sysconfig/network (only valid for Red Hat like distributions). In newer releases, one binary supports IPv4 and IPv6.

If you enable a built-in service like e.g. daytime by modifying the configuration file in /etc/xinetd.d/daytime like

```
# diff -u /etc/xinetd.d/daytime.orig /etc/xinetd.d/daytime
--- /etc/xinetd.d/daytime.orig Sun Dec 16 19:00:14 2001
+++ /etc/xinetd.d/daytime Sun Dec 16 19:00:22 2001
@@ -10,5 +10,5 @@
```

```
        protocol = tcp
        user = root
        wait = no
-       disable = yes
+       disable = no
 }
```

After restarting the xinetd you should get a positive result like:

```
# netstat -lnptu -A inet6 |grep "xinetd*"
tcp 0 0 ::ffff:192.168.1.1:993  :::*  LISTEN  12345/xinetd-ipv6
tcp 0 0 :::13                   :::*  LISTEN  12345/xinetd-ipv6 <- service
¬ daytime/tcp
tcp 0 0 ::ffff:192.168.1.1:143  :::*  LISTEN  12345/xinetd-ipv6
```

Shown example also displays an IMAP and IMAP-SSL IPv4-only listening xinetd.

Note: earlier versions had a problem that an IPv4-only xinetd won't start on an IPv6-enabled node and also the IPv6-enabled xinetd won't start on an IPv4-only node. This is known to be fixed in later versions, at least version 2.3.11.

22.3. Webserver Apache2 (httpd2)

Apache web server supports IPv6 native by maintainers since 2.0.14. Available patches for the older 1.3.x series are not current and shouldn't be used in public environment, but available at KAME / Misc (ftp://ftp.kame.net/pub/kame/misc/).

22.3.1. Listening on IPv6 addresses

Note: virtual hosts on IPv6 addresses are broken in versions until 2.0.28 (a patch is available for 2.0.28). But always try latest available version first because earlier versions had some security issues.

22.3.1.1. Virtual host listen on an IPv6 address only

```
Listen [2001:0db8:100::1]:80
<VirtualHost [2001:0db8:100::1]:80>
        ServerName ipv6only.yourdomain.yourtopleveldomain
        # ...sure more config lines
</VirtualHost>
```

22.3.1.2. Virtual host listen on an IPv6 and on an IPv4 address

```
Listen [2001:0db8:100::2]:80
Listen 1.2.3.4:80
<VirtualHost [2001:0db8:100::2]:80 1.2.3.4:80>
        ServerName ipv6andipv4.yourdomain.yourtopleveldomain
        # ...sure more config lines
</VirtualHost>
```

This should result after restart in e.g.

```
# netstat -lnptu |grep "httpd2\W*$"
```

```
tcp 0 0 1.2.3.4:80          0.0.0.0:* LISTEN 12345/httpd2
tcp 0 0 2001:0db8:100::1:80 :::*      LISTEN 12345/httpd2
tcp 0 0 2001:0db8:100::2:80 :::*      LISTEN 12345/httpd2
```

For simple tests use the telnet example already shown.

22.3.1.3. Additional notes

- Apache2 supports a method called “sendfile” to speedup serving data. Some NIC drivers also support offline checksumming. In some cases, this can lead to connection problems and invalid TCP checksums. In this cases, disable “sendfile” either by recompiling using configure option “--without-sendfile” or by using the "EnableSendfile off" directive in configuration file.

22.4. Router Advertisement Daemon (radvd)

The router advertisement daemon is very useful on a LAN, if clients should be auto-configured. The daemon itself should run on the Linux default IPv6 gateway router (it’s not required that this is also the default IPv4 gateway, so pay attention who on your LAN is sending router advertisements).

You can specify some information and flags which should be contained in the advertisement. Common used are

- Prefix (needed)
- Lifetime of the prefix
- Frequency of sending advertisements (optional)

After a proper configuration, the daemon sends advertisements through specified interfaces and clients are hopefully receive them and auto-magically configure addresses with received prefix and the default route.

22.4.1. Configuring radvd

22.4.1.1. Simple configuration

Radvd’s config file is normally /etc/radvd.conf. An simple example looks like following:

```
interface eth0 {
        AdvSendAdvert on;
        MinRtrAdvInterval 3;
        MaxRtrAdvInterval 10;
        prefix 2001:0db8:0100:f101::/64 {
                AdvOnLink on;
                AdvAutonomous on;
                AdvRouterAddr on;
        };
};
```

This results on client side in

```
# ip -6 addr show eth0
```

```
3: eth0: <BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast qlen 100
    inet6 2001:0db8:100:f101:2e0:12ff:fe34:1234/64 scope global dynamic
       valid_lft 2591992sec preferred_lft 604792sec
    inet6 fe80::2e0:12ff:fe34:1234/10 scope link
```

Because no lifetime was defined, a very high value was used.

22.4.1.2. Special 6to4 configuration

Version since 0.6.2pl3 support the automatic (re)-generation of the prefix depending on an IPv4 address of a specified interface. This can be used to distribute advertisements in a LAN after the 6to4 tunneling has changed. Mostly used behind a dynamic dial-on-demand Linux router. Because of the sure shorter lifetime of such prefix (after each dial-up, another prefix is valid), the lifetime configured to minimal values:

```
interface eth0 {
        AdvSendAdvert on;
        MinRtrAdvInterval 3;
        MaxRtrAdvInterval 10;
        prefix 0:0:0:f101::/64 {
                AdvOnLink off;
                AdvAutonomous on;
                AdvRouterAddr on;
                Base6to4Interface ppp0;
                AdvPreferredLifetime 20;
                AdvValidLifetime 30;
        };
};
```

This results on client side in (assuming, ppp0 has currently 1.2.3.4 as local IPv4 address):

```
# /sbin/ip -6 addr show eth0
3: eth0: <BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast qlen 100
   inet6 2002:0102:0304:f101:2e0:12ff:fe34:1234/64 scope global dynamic
      valid_lft 22sec preferred_lft 12sec
   inet6 fe80::2e0:12ff:fe34:1234/10 scope link
```

Because a small lifetime was defined, such prefix will be thrown away quickly, if no related advertisement was received.

Additional note: if you do not used special 6to4 support in initscripts, you have to setup a special route on the internal interface on the router, otherwise you will get some backrouting problems. for the example showh here:

```
# /sbin/ip -6 route add 2002:0102:0304:f101::/64 dev eth0 metric 1
```

This route needs to be replaced every time the prefix changes, which is the case after a new IPv4 address was assigned to the dial-up interface.

22.4.2. Debugging

A program called “radvdump” can help you looking into sent or received advertisements. Simple to use:

```
# radvdump
Router advertisement from fe80::280:c8ff:feb9:cef9 (hoplimit 255)
        AdvCurHopLimit: 64
```

```
        AdvManagedFlag: off
        AdvOtherConfigFlag: off
        AdvHomeAgentFlag: off
        AdvReachableTime: 0
        AdvRetransTimer: 0
        Prefix 2002:0102:0304:f101::/64
                AdvValidLifetime: 30
                AdvPreferredLifetime: 20
                AdvOnLink: off
                AdvAutonomous: on
                AdvRouterAddr: on
        Prefix 2001:0db8:100:f101::/64
                AdvValidLifetime: 2592000
                AdvPreferredLifetime: 604800
                AdvOnLink: on
                AdvAutonomous: on
                AdvRouterAddr: on
        AdvSourceLLAddress: 00 80 12 34 56 78
```

Output shows you each advertisement package in readable format. You should see your configured values here again, if not, perhaps it's not your radvd which sends the advertisement...look for another router on the link (and take the LLAddress, which is the MAC address for tracing).

22.5. Dynamic Host Configuration v6 Server (dhcp6s)

DHCPv6 can be used for stateful configurations. The daemon itself need not necessary run on the Linux default IPv6 gateway router.

You can specify more information than by using radvd. The are most similar to IPv4 DHCP server.

After a proper configuration, the daemon reacts on received ICMPv6 multicast packets sent by a client to address ff02::1:2

22.5.1. Configuration of the DHCPv6 server (dhcp6s)

22.5.1.1. Simple configuration

dhcp6s's config file is normally /etc/dhcp6s.conf. An simple example looks like following:

```
interface eth0 {
        server-preference 255;
        renew-time 60;
        rebind-time 90;
        prefer-life-time 130;
        valid-life-time 200;
        allow rapid-commit;
        option dns_servers 2001:db8:0:f101::1 sub.domain.example;
        link AAA {
                range 2001:db8:0:f101::1000 to 2001:db8:0:f101::ffff/64;
                prefix 2001:db8:0:f101::/64;
        };
};
```

22.5.2. Configuration of the DHCPv6 client (dhcp6c)

22.5.2.1. Simple configuration

dhcp6c's config file is normally /etc/dhcp6c.conf. An simple example looks like following:

```
interface eth0 {
        send rapid-commit;
        request domain-name-servers;
};
```

22.5.3. Usage

22.5.3.1. dhcpv6_server

Start server, e.g.

```
# service dhcp6s start
```

22.5.3.2. dhcpv6_client

Start client in foreground, e.g.

```
# dhcp6c -f eth0
```

22.5.4. Debugging

22.5.4.1. dhcpv6_server

The server has one foreground and two debug toggles (both should be used for debugging), here is an example:

```
# dhcp6s -d -D -f eth0
```

22.5.4.2. dhcpv6_client

As general debugging for test whether the IPv6 DHCP server is reable on the link use an IPv6 ping to the DHCP multicast address:

```
# ping6 -I eth0 ff02::1:2
```

The client has one foreground and two debug toggles, here is an example:

```
# dhcp6c -d -f eth0
Oct/03/2005 17:18:16 dhcpv6 doesn't support hardware type 776
Oct/03/2005 17:18:16 doesn't support sit0 address family 0
Oct/03/2005 17:18:16 netlink_recv_rtgenmsg error
Oct/03/2005 17:18:16 netlink_recv_rtgenmsg error
```

```
Oct/03/2005 17:18:17 status code for this address is: success
Oct/03/2005 17:18:17 status code: success
Oct/03/2005 17:18:17 netlink_recv_rtgenmsg error
Oct/03/2005 17:18:17 netlink_recv_rtgenmsg error
Oct/03/2005 17:18:17 assigned address 2001:db8:0:f101::1002 prefix len is not
¬ in any RAs prefix length using 64 bit instead
Oct/03/2005 17:18:17 renew time 60, rebind time 9
```

Note that the netlink error messages have no impact.

22.6. ISC Dynamic Host Configuration Server (dhcpd)

ISC DHCP supports IPv6 since version 4.x.

22.6.1. Configuration of the ISC DHCP server for IPv6 (dhcpd)

Note that currently, the ISC DHCP server can only serve IPv4 or IPv6, means you have to start the daemon twice (for IPv6 with option "-6") to support both protocols.

22.6.1.1. Simple configuration

Create a dedicated configuration file /etc/dhcp/dhcpd6.conf for the IPv6 part of the dhcpd. Note, that the router requires to have a interface configured with an IPv6 address out of the defined subnet.

```
default-lease-time 600;
max-lease-time 7200;
log-facility local7;
subnet6 2001:db8:0:1::/64 {
        # Range for clients
        range6 2001:db8:0:1::129 2001:db8:0:1::254;
        # Additional options
        option dhcp6.name-servers fec0:0:0:1::1;
        option dhcp6.domain-search "domain.example";
        # Prefix range for delegation to sub-routers
        prefix6 2001:db8:0:100:: 2001:db8:0:f00:: /56;
        # Example for a fixed host address
        host specialclient {
      host-identifier option dhcp6.client-id 00:01:00:01:4a:1f:ba:e3:60:b9:1f:01:23:45;
         fixed-address6 2001:db8:0:1::127;
     }
}
```

Note that the "dhcp.client-id" no longer belongs to a MAC address, an unique ID is used instead! "dhcp6c" (see above) uses the file /var/lib/dhcpv6/dhcp6c_duid (would be created during first start, if not existing) as unique identity. It's a 14 byte long identifier, starting with a 2 byte length information (usually "0x000e"):

```
# hexdump -e '"%07.7_ax " 1/2 "%04x" " " 14/1 "%02x:" "\n"' /var/lib/dhcpv6/dhcp6c_duid 0000000 0
```

22.6.2. Usage

22.6.2.1. dhcpd

Start server in foreground:

```
# /usr/sbin/dhcpd -6 -f -cf /etc/dhcp/dhcpd.conf eth1
Internet Systems Consortium DHCP Server 4.1.0
Copyright 2004-2008 Internet Systems Consortium.
All rights reserved.
For info, please visit http://www.isc.org/sw/dhcp/
Not searching LDAP since ldap-server, ldap-port and ldap-base-dn were not specified in the config
Wrote 0 leases to leases file.
Bound to *:547
Listening on Socket/5/eth1/2001:db8:0:1::/64
Sending on   Socket/5/eth1/2001:db8:0:1::/64
```

22.7. DHCP Server Dibbler

Dibbler is also a DHCP server

22.7.1. Configuration of the Dibbler DHCP server for IPv6

22.7.1.1. Simple configuration

Create a dedicated configuration file /etc/dibbler/server.conf . Note, that the router requires to have a interface configured with an IPv6 address out of the defined subnet.

```
log-level 8
log-mode short
preference 0
iface "eth1" {
 // also ranges can be defines, instead of exact values  t1 1800-2000  t2 2700-3000
  prefered-lifetime 3600
  valid-lifetime 7200
  class {
    pool 2001:6f8:12d8:1::/64
  }
  option dns-server fec0:0:0:1::1
  option domain domain.example
}
```

22.7.2. Usage

22.7.2.1. dibbler-server

Start server in foreground:

```
# dibbler-server run
| Dibbler - a portable DHCPv6, version 0.7.3 (SERVER, Linux port)
| Authors : Tomasz Mrugalski<thomson(at)klub.com.pl>,Marek Senderski<msend(at)o2.pl>
| Licence : GNU GPL v2 only. Developed at Gdansk University of Technology.
| Homepage: http://klub.com.pl/dhcpv6/
2009.05.28 10:18:48 Server Notice    My pid (1789) is stored in /var/lib/dibbler/server.pid
2009.05.28 10:18:48 Server Notice    Detected iface eth0/3, MAC=54:52:00:01:23:45.
2009.05.28 10:18:48 Server Notice    Detected iface eth1/2, MAC=54:52:00:67:89:ab.
2009.05.28 10:18:48 Server Notice    Detected iface lo/1, MAC=00:00:00:00:00:00.
2009.05.28 10:18:48 Server Debug     Skipping database loading.
2009.05.28 10:18:48 Server Debug     Cache:server-cache.xml file: parsing started, expecting 0 en
2009.05.28 10:18:48 Server Notice    Parsing /etc/dibbler/server.conf config file...
18:48 Server Debug     Setting 0 generic option(s).
18:48 Server Debug     0 per-client configurations (exceptions) added.
18:48 Server Debug     Parsing /etc/dibbler/server.conf done.
18:48 Server Info      0 client class(es) defined.
18:48 Server Debug     1 interface(s) specified in /etc/dibbler/server.conf
18:48 Server Info      Mapping allow, deny list to class 0:0 allow/deny entries in total.
18:48 Server Info      Interface eth1/2 configuration has been loaded.
18:48 Server Notice    Running in stateful mode.
18:48 Server Info      My DUID is 00:01:00:01:11:aa:6d:a7:54:52:00:67:89:ab.
18:48 Server Notice    Creating multicast (ff02::1:2) socket on eth1/2 (eth1/2) interface.
18:48 Server Debug     Cache: size set to 1048576 bytes, 1 cache entry size is 87 bytes, so maxim
18:48 Server Notice    Accepting connections. Next event in 4294967295 second(s).
```

22.8. tcp_wrapper

tcp_wrapper is a library which can help you to protect service against misuse.

22.8.1. Filtering capabilities

You can use tcp_wrapper for

- Filtering against source addresses (IPv4 or IPv6)
- Filtering against users (requires a running ident daemon on the client)

22.8.2. Which program uses tcp_wrapper

Following are known:

- Each service which is called by xinetd (if xinetd is compiled using tcp_wrapper library)
- sshd (if compiled using tcp_wrapper)

22.8.3. Usage

tcp_wrapper is controlled by two files name /etc/hosts.allow and /etc/hosts.deny. For more information see

```
$ man hosts.allow
```

22.8.3.1. Example for /etc/hosts.allow

In this file, each service which should be positive filtered (means connects are accepted) need a line.

```
sshd:           1.2.3. [2001:0db8:100:200::]/64
daytime-stream: 1.2.3. [2001:0db8:100:200::]/64
```

Note: there are broken implementations around, which uses following broken IPv6 network description: [2001:0db8:100:200::/64]. Hopefully, such versions will be fixed soon.

22.8.3.2. Example for /etc/hosts.deny

This file contains all negative filter entries and should normally deny the rest using

```
ALL: ALL
```

If this node is a more sensible one you can replace the standard line above with this one, but this can cause a DoS attack (load of mailer and spool directory), if too many connects were made in short time. Perhaps a logwatch is better for such issues.

```
ALL: ALL: spawn (echo "Attempt from %h %a to %d at `date`"
 | tee -a /var/log/tcp.deny.log | mail root@localhost)
```

22.8.4. Logging

Depending on the entry in the syslog daemon configuration file /etc/syslog.conf the tcp_wrapper logs normally into /var/log/secure.

22.8.4.1. Refused connection

A refused connection via IPv4 to an xinetd covered daytime service produces a line like following example

```
Jan 2 20:40:44 gate xinetd-ipv6[12346]: FAIL: daytime-stream libwrap
¬ from=::ffff:1.2.3.4
Jan 2 20:32:06 gate xinetd-ipv6[12346]: FAIL: daytime-stream libwrap
 from=2001:0db8:100:200::212:34ff:fe12:3456
```

A refused connection via IPv4 to an dual-listen sshd produces a line like following example

```
Jan 2 20:24:17 gate sshd[12345]: refused connect from ::ffff:1.2.3.4
¬ (::ffff:1.2.3.4)
Jan 2 20:39:33 gate sshd[12345]: refused connect
 from 2001:0db8:100:200::212:34ff:fe12:3456
¬ (2001:0db8:100:200::212:34ff:fe12:3456)
```

22.8.4.2. Permitted connection

A permitted connection via IPv4 to an xinetd covered daytime service produces a line like following example

```
Jan 2 20:37:50 gate xinetd-ipv6[12346]: START: daytime-stream pid=0
¬ from=::ffff:1.2.3.4
Jan 2 20:37:56 gate xinetd-ipv6[12346]: START: daytime-stream pid=0
 from=2001:0db8:100:200::212:34ff:fe12:3456
```

A permitted connection via IPv4 to an dual-listen sshd produces a line like following example

```
Jan 2 20:43:10 gate sshd[21975]: Accepted password for user from ::ffff:1.2.3.4
¬ port 33381 ssh2
Jan 2 20:42:19 gate sshd[12345]: Accepted password for user
 from 2001:0db8:100:200::212:34ff:fe12:3456 port 33380 ssh2
```

22.9. vsftpd

22.9.1. Listening on IPv6 addresses

Edit the configuration file, ususally /etc/vsftpd/vsftpd.conf, and adjust the listen option like

```
listen_ipv6=yes
```

That's all.

22.10. proftpd

22.10.1. Listening on IPv6 addresses

Edit the configuration file, ususally /etc/proftpd.conf, but take care, not 100% logical in virtual host setup

```
<VirtualHost 192.0.2.1>
        ...
        Bind 2001:0DB8::1
        ...
</VirtualHost>
```

That's all.

22.11. Other daemons

Nowadays it's mostly simple, look for either a command line option or a configuration value to enable IPv6 listening. See manual page of the daemon or check related FAQs. It can happen that you can bind a daemon only

to the IPv6-“any”-address (::) and not to bind to a dedicated IPv6 address, because the lack of support (depends on that what the programmer has implemented so far...).

Chapter 23. Programming

23.1. Programming using C-API

Related RFCs:

- RFC 3493 / Basic Socket Interface Extensions for IPv6 (http://www.faqs.org/rfcs/rfc3493.html)
- RFC 3542 / Advanced Sockets Application Program Interface (API) for IPv6 (http://www.faqs.org/rfcs/rfc3542.html)

Following contents of this section is contributed by John Wenker, Sr. Software Engineer Performance Technologies San Diego, CA USA http://www.pt.com/.

This section describes how to write IPv6 client-server applications under the Linux operating system. First thing's first, and credit must be given where it is due. The information contained in this section is derived from Chapters 2 through 4 of IPv6 Network Programming by Jun-ichiro itojun Hagino (ISBN 1-55558-318-0). The reader is encouraged to consult that book for more detailed information. It describes how to convert IPv4 applications to be IPv6 compatible in a protocol-independent way, and describes some of the common problems encountered during the conversion along with suggested solutions. At the time of this writing, this is the only book of which the author is aware that specifically addresses how to program IPv6 applications [since writing this section, the author has also become aware of the Porting applications to IPv6 HowTo by Eva M. Castro at http://jungla.dit.upm.es/~ecastro/IPv6-web/ipv6.html]. Unfortunately, of the almost 360 pages in the book, maybe 60 are actually useful (the chapters mentioned). Nevertheless, without the guidance of that book, the author would have been unable to perform his job duties or compose this HowTo. While most (but certainly not all) of the information in the Hagino book is available via the Linux 'man' pages, application programmers will save a significant amount of time and frustration by reading the indicated chapters of the book rather than searching through the 'man' pages and online documentation.

Other than the Hagino book, any other information presented in this HowTo was obtained through trial and error. Some items or explanations may not be entirely "correct" in the grand IPv6 scheme, but seem to work in practical application.

The discussion that follows assumes the reader is already experienced with the traditional TCP/IP socket API. For more information on traditional socket programming, the Internetworking with TCP/IP series of textbooks by Comer & Stevens is hard to beat, specifically Volume III: Client-Server Programming and Applications, Linux/POSIX Sockets Version (ISBN 0-13-032071-4). This HowTo also assumes that the reader has had at least a bare basic introduction to IPv6 and in particular the addressing scheme for network addresses (see Section 2.3).

23.1.1. Address Structures

This section provides a brief overview of the structures provided in the socket API to represent network addresses (or more specifically transport endpoints) when using the Internet protocols in a client-server application.

23.1.1.1. IPv4 sockaddr_in

In IPv4, network addresses are 32 bits long and define a network node. Addresses are written in dotted decimal notation, such as 192.0.2.1, where each number represents eight bits of the address. Such an IPv4 address is represented by the struct sockaddr_in data type, which is defined in <netinet/in.h>.

```
struct sockaddr_in
{
```

```
   sa_family_t    sin_family;
   in_port_t      sin_port;
   struct in_addr sin_addr;
   /* Plus some padding for alignment */
};
```

The sin_family component indicates the address family. For IPv4 addresses, this is always set to AF_INET. The sin_addr field contains the 32-bit network address (in network byte order). Finally, the sin_port component represents the transport layer port number (in network byte order). Readers should already be familiar with this structure, as this is the standard IPv4 address structure.

23.1.1.2. IPv6 sockaddr_in6

The biggest feature of IPv6 is its increased address space. Instead of 32-bit network addresses, IPv6 allots 128 bits to an address. Addresses are written in colon-hex notation of the form fe80::2c0:8cff:fe01:2345, where each hex number separated by colons represents 16 bits of the address. Two consecutive colons indicate a string of consecutive zeros for brevity, and at most only one double-colon may appear in the address. IPv6 addresses are represented by the struct sockaddr_in6 data type, also defined in <netinet/in.h>.

```
struct sockaddr_in6
{
   sa_family_t     sin6_family;
   in_port_t       sin6_port;
   uint32_t        sin6_flowinfo;
   struct in6_addr sin6_addr;
   uint32_t        sin6_scope_id;
};
```

The sin6_family, sin6_port, and sin6_addr components of the structure have the same meaning as the corresponding fields in the sockaddr_in structure. However, the sin6_family member is set to AF_INET6 for IPv6 addresses, and the sin6_addr field holds a 128-bit address instead of only 32 bits.

The sin6_flowinfo field is used for flow control, but is not yet standardized and can be ignored.

The sin6_scope_id field has an odd use, and it seems (at least to this naïve author) that the IPv6 designers took a huge step backwards when devising this. Apparently, 128-bit IPv6 network addresses are not unique. For example, it is possible to have two hosts, on separate networks, with the same link-local address (see Figure 1). In order to pass information to a specific host, more than just the network address is required; the scope identifier must also be specified. In Linux, the network interface name is used for the scope identifier (e.g. “eth0”) [be warned that the scope identifier is implementation dependent!]. Use the ifconfig(1M) command to display a list of active network interfaces.

A colon-hex network address can be augmented with the scope identifier to produce a "scoped address”. The percent sign (’%’) is used to delimit the network address from the scope identifier. For example, fe80::1%eth0 is a scoped IPv6 address where fe80::1 represents the 128-bit network address and eth0 is the network interface (i.e. the scope identifier). Thus, if a host resides on two networks, such as Host B in example below, the user now has to know which path to take in order to get to a particular host. In Figure 1, Host B addresses Host A using the scoped address fe80::1%eth0, while Host C is addressed with fe80::1%eth1.

```
Host A (fe80::1) ---- eth0 ---- Host B ---- eth1 ---- Host C (fe80::1)
```

Getting back to the sockaddr_in6 structure, its sin6_scope_id field contains the index of the network interface on which a host may be found. Server applications will have this field set automatically by the socket API when they accept a connection or receive a datagram. For client applications, if a scoped address is passed as the node parameter to getaddrinfo(3) (described later in this HowTo), then the sin6_scope_id field will be filled in

correctly by the system upon return from the function; if a scoped address is not supplied, then the sin6_scope_id field must be explicitly set by the client software prior to attempting to communicate with the remote server. The if_nametoindex(3) function is used to translate a network interface name into its corresponding index. It is declared in <net/if.h>.

23.1.1.3. Generic Addresses

As any programmer familiar with the traditional TCP/IP socket API knows, several socket functions deal with "generic" pointers. For example, a pointer to a generic struct sockaddr data type is passed as a parameter to some socket functions (such as connect(2) or bind(2)) rather than a pointer to a specific address type. Be careful... the sockaddr_in6 structure is larger than the generic sockaddr structure! Thus, if your program receives a generic address whose actual type is unknown (e.g. it could be an IPv4 address structure or an IPv6 address structure), you must supply sufficient storage to hold the entire address. The struct sockaddr_storage data type is defined in <bits/socket.h> for this purpose [do not #include this file directly within an application; use <sys/socket.h> as usual, and <bits/socket.h> will be implicitly included].

For example, consider the recvfrom(2) system call, which is used to receive a message from a remote peer. Its function prototype is:

```
ssize_t recvfrom( int              s,
                  void            *buf,
                  size_t           len,
                  int              flags,
                  struct sockaddr *from,
                  socklen_t       *fromlen );
```

The from parameter points to a generic sockaddr structure. If data can be received from an IPv6 peer on the socket referenced by s, then from should point to a data type of struct sockaddr_storage, as in the following dummy example:

```
/*
** Read a message from a remote peer, and return a buffer pointer to
** the caller.
**
** 's' is the file descriptor for the socket.
*/
char *rcvMsg( int s )
{
   static char             bfr[ 1025 ];  /* Where the msg is stored. */
   ssize_t                 count;
   struct sockaddr_storage ss;           /* Where the peer adr goes. */
   socklen_t               sslen;
   sslen = sizeof( ss );
   count = recvfrom( s,
                     bfr,
                     sizeof( bfr ) - 1,
                     0,
                     (struct sockaddr*) &ss,
                     &sslen );
   bfr[ count ] = '\0';   /* Null-terminates the message. */
   return bfr;
}  /* End rcvMsg() */
```

As seen in the above example, ss (a struct sockaddr_storage data object) is used to receive the peer address information, but it's address is typecast to a generic struct sockaddr* pointer in the call to recvfrom(2).

23.1.2. Lookup Functions

Traditionally, hostname and service name resolution were performed by functions such as gethostbyname(3) and getservbyname(3). These traditional lookup functions are still available, but they are not forward compatible to IPv6. Instead, the IPv6 socket API provides new lookup functions that consolidate the functionality of several traditional functions. These new lookup functions are also backward compatible with IPv4, so a programmer can use the same translation algorithm in an application for both the IPv4 and IPv6 protocols. This is an important feature, because obviously a global IPv6 infrastructure isn't going to be put in place overnight. Thus, during the transition period from IPv4 to IPv6, client-server applications should be designed with the flexibility to handle both protocols simultaneously. The example programs at the end of this chapter do just that.

The primary lookup function in the new socket API is getaddrinfo(3). Its prototype is as follows.

```
int getaddrinfo( const char             *node,
                 const char             *service,
                 const struct addrinfo  *hints,
                 struct addrinfo       **res );
```

The node parameter is a pointer to the hostname or IP address being translated. The referenced string can be a hostname, IPv4 dotted decimal address, or IPv6 colon-hex address (possibly scoped). The service parameter is a pointer to the transport layer's service name or port number. It can be specified as a name found in /etc/services or a decimal number. getaddrinfo(3) resolves the host/service combination and returns a list of address records; a pointer to the list is placed in the location pointed at by res. For example, suppose a host can be identified by both an IPv4 and IPv6 address, and that the indicated service has both a TCP entry and UDP entry in /etc/services. In such a scenario, it is not inconceivable that four address records are returned; one for TCP/IPv6, one for UDP/IPv6, one for TCP/IPv4, and one for UDP/IPv4.

The definition for struct addrinfo is found in <netdb.h> (as is the declaration for getaddrinfo(3) and the other functions described in this section). The structure has the following format:

```
struct addrinfo
{
   int              ai_flags;
   int              ai_family;
   int              ai_socktype;
   int              ai_protocol;
   socklen_t        ai_addrlen;
   struct sockaddr *ai_addr;
   char            *ai_canonname;
   struct addrinfo *ai_next;
};
```

Consult the 'man' page for getaddrinfo(3) for detailed information about the various fields; this HowTo only describes a subset of them, and only to the extent necessary for normal IPv6 programming.

The ai_family, ai_socktype, and ai_protocol fields have the exact same meaning as the parameters to the socket(2) system call. The ai_family field indicates the protocol family (not the address family) associated with the record, and will be PF_INET6 for IPv6 or PF_INET for IPv4. The ai_socktype parameter indicates the type of socket to which the record corresponds; SOCK_STREAM for a reliable connection-oriented byte-stream or SOCK_DGRAM for connectionless communication. The ai_protocol field specifies the underlying transport protocol for the record.

The ai_addr field points to a generic struct sockaddr object. Depending on the value in the ai_family field, it will point to either a struct sockaddr_in (PF_INET) or a struct sockaddr_in6 (PF_INET6). The ai_addrlen field contains the size of the object pointed at by the ai_addr field.

As mentioned, getaddrinfo(3) returns a list of address records. The ai_next field points to the next record in the list.

The hints parameter to getaddrinfo(3) is also of type struct addrinfo and acts as a filter for the address records returned in res. If hints is NULL, all matching records are returned; but if hints is non-NULL, the referenced structure gives "hints" to getaddrinfo(3) about which records to return. Only the ai_flags, ai_family, ai_socktype, and ai_protocol fields are significant in the hints structure, and all other fields should be set to zero.

Programs can use hints->ai_family to specify the protocol family. For example, if it is set to PF_INET6, then only IPv6 address records are returned. Likewise, setting hints->ai_family to PF_INET results in only IPv4 address records being returned. If an application wants both IPv4 and IPv6 records, the field should be set to PF_UNSPEC.

The hints->socktype field can be set to SOCK_STREAM to return only records that correspond to connection-oriented byte streams, SOCK_DGRAM to return only records corresponding to connectionless communication, or 0 to return both.

For the Internet protocols, there is only one protocol associated with connection-oriented sockets (TCP) and one protocol associated with connectionless sockets (UDP), so setting hints->ai_socktype to SOCK_STREAM or SOCK_DGRAM is the same as saying, "Give me only TCP records," or "Give me only UDP records," respectively. With that in mind, the hints->ai_protocol field isn't really that important with the Internet protocols, and pretty much mirrors the hints->ai_socktype field. Nevertheless, hints->ai_protocol can be set to IPPROTO_TCP to return only TCP records, IPPROTO_UDP to return only UDP records, or 0 for both.

The node or service parameter to gethostbyname(3) can be NULL, but not both. If node is NULL, then the ai_flags field of the hints parameter specifies how the network address in a returned record is set (i.e. the sin_addr or sin6_addr field of the object pointed at by the ai_addr component in a returned record). If the AI_PASSIVE flag is set in hints, then the returned network addresses are left unresolved (all zeros). This is how server applications would use getaddrinfo(3). If the flag is not set, then the address is set to the local loopback address (::1 for IPv6 or 127.0.0.1 for IPv4). This is one way a client application can specify that the target server is running on the same machine as the client. If the service parameter is NULL, the port number in the returned address records remains unresolved.

The getaddrinfo(3) function returns zero on success, or an error code. In the case of an error, the gai_strerror(3) function is used to obtain a character pointer to an error message corresponding to the error code, just like strerror(3) does in the standard 'C' library.

Once the address list is no longer needed, it must be freed by the application. This is done with the freeaddrinfo(3) function.

The last function that will be mentioned in this section is getnameinfo(3). This function is the inverse of getaddrinfo(3); it is used to create a string representation of the hostname and service from a generic struct sockaddr data object. It has the following prototype.

```
int getnameinfo( const struct sockaddr *sa,
                 socklen_t              salen,
                 char                  *host,
                 size_t                 hostlen,
                 char                  *serv,
                 size_t                 servlen,
                 int                    flags );
```

The sa parameter points to the address structure in question, and salen contains its size. The host parameter points to a buffer where the null-terminated hostname string is placed, and the hostlen parameter is the size of that buffer. If there is no hostname that corresponds to the address, then the network address (dotted decimal or colon-hex) is placed in host. Likewise, the serv parameter points to a buffer where the null-terminated service name string (or port number) is placed, and the servlen parameter is the size of that buffer. The flags parameter modifies the function's behavior; in particular, the NI_NUMERICHOST flag indicates that the converted hostname should always

be formatted in numeric form (i.e. dotted decimal or colon-hex), and the NI_NUMERICSERV flag indicates that the converted service should always be in numeric form (i.e. the port number).

The symbols NI_MAXHOST and NI_MAXSERV are available to applications and represent the maximum size of any converted hostname or service name, respectively. Use these when declaring output buffers for getnameinfo(3).

23.1.3. Quirks Encountered

Before jumping into the programming examples, there are several quirks in IPv6 of which the reader should be aware. The more significant ones (in addition to the non-uniqueness of IPv6 network addresses already discussed) are described in the paragraphs below.

23.1.3.1. IPv4 Mapped Addresses

For security reasons that this author won't pretend to understand, "IPv4 mapped addresses" should not be allowed in IPv6-capable server applications. To put it in terms that everyone can understand, this simply means that a server should not accept IPv4 traffic on an IPv6 socket (an otherwise legal operation). An IPv4 mapped address is a mixed-format address of the form:

```
::ffff:192.0.2.1
```

where the first portion is in IPv6 colon-hex format and the last portion is in IPv4 dotted decimal notation. The dotted decimal IPv4 address is the actual network address, but it is being mapped into an IPv6 compatible format.

To prevent IPv4 mapped addresses from being accepted on an IPv6 socket, server applications must explicitly set the IPV6_V6ONLY socket option on all IPv6 sockets created [the Hagino book implies that this is only a concern with server applications. However, it has been observed during testing that if a client application uses an IPv4 mapped address to specify the target server, and the target server has IPv4 mapped addresses disabled, the connection still completes regardless. On the server side, the connection endpoint is an IPv4 socket as desired; but on the client side, the connection endpoint is an IPv6 socket. Setting the IPV6_V6ONLY socket option on the client side as well as the server side prevents any connection from being established at all.]. There's only one problem. Apparently, IPV6_V6ONLY isn't defined on all systems [or at least it wasn't in 2005 when the Hagino book was written]. The server example at the end of this chapter provides a method for handling this problem.

If IPv4 traffic cannot be handled on IPv6 sockets, then that implies that server applications must open both an IPv4 and IPv6 socket for a particular network service if it wants to handle requests from either protocol. This goes back to the flexibility issue mentioned earlier. If getaddrinfo(3) returns multiple address records, then server applications should traverse the list and open a passive socket for each address provided.

23.1.3.2. Cannot Specify the Scope Identifier in /etc/hosts

It is possible to assign a hostname to an IPv6 network address in /etc/hosts. For example, the following is an excerpt from the /etc/hosts file on the author's development system.

```
::1                     localhost
127.0.0.1               localhost
fe80::2c0:8cff:fe01:2345  pt141
192.0.2.1               pt141
```

The "localhost" and "pt141" hostnames can be translated to either an IPv4 or IPv6 network address. So, for example, if "pt141" is passed as the node parameter to getaddrinfo(3), the function returns both an IPv4 and IPv6

address record for the host (assuming the behavior hasn't been modified by the hints parameter). Unfortunately, a scoped address cannot be used in /etc/hosts. Doing so results in getaddrinfo(3) returning only the IPv4 record.

23.1.3.3. Client & Server Residing on the Same Machine

Suppose a machine has the IPv4 address 192.0.2.1. A client application running on that machine can connect to a server application on the same machine by using either the local loopback address (127.0.0.1) or the network address (192.0.2.1) as the target server. Much to this author's surprise (and dismay), it turns out that an IPv6 client application cannot connect to a server application on the same machine if it uses the network address of that machine as the target; it must use the local loopback address (::1).

23.1.4. Putting It All Together (A Client-Server Programming Example)

Now it's time to put everything discussed thus far together into a sample client-server application. The remainder of this section is devoted to a remote time-of-day application (the 'daytime' Internet service) [I noticed that Ms. Castro used a 'daytime' example in her *Porting applications to IPv6 HowTo*. For the record, the source code presented here is original, developed from scratch, and any similarity between it and any other publicly available 'daytime' example is purely coincidental.]. The source code presented in this section was developed and tested on a RedHat Linux release using the 2.6 kernel (2.6.9 to be specific). Readers may use the source code freely, so long as proper credit is attributed; but of course the standard disclaimer must be given first:

> Although the sample source code is believed to be free of errors, the author makes no guarantees as to its reliability, especially considering that some error paths were intentionally omitted for brevity. Use it at your own risk!

When you get right down to it, there really aren't that many differences between IPv4 and IPv6 applications. The trick is to code IPv6 applications in a protocol-independent manner, such that they can handle both IPv4 and IPv6 simultaneously and transparently. This sample application does just that. The only protocol-dependent code in the example occurs when printing network addresses in verbose mode; but only after the ai_family field in the addrinfo structure has been checked, so the programs know exactly what type of address they're handling at the time.

23.1.4.1. 'Daytime' Server Code

The server code is found in file tod6d.c (time-of-day IPv6 daemon). Once built, the server may be started using the following command syntax (assuming tod6d is the executable file):

```
tod6d [-v] [service]
```

ARGUMENTS:

service

The service (or well-known port) on which to listen. Default is "daytime".

OPTIONS:

-v

Turn on verbose mode.

The server handles both TCP and UDP requests on the network. The server source code contained in tod6d.c follows:

```
/*************************************************************************
* File: tod6d.c
* Description: Contains source code for an IPv6-capable 'daytime' server.
* Author: John Wenker, Sr. Software Engineer,
*         Performance Technologies, San Diego, USA
*************************************************************************/
/*
** System header files.
*/
#include <errno.h>        /* errno declaration & error codes.            */
#include <netdb.h>        /* getaddrinfo(3) et al.                       */
#include <netinet/in.h>   /* sockaddr_in & sockaddr_in6 definition.      */
#include <stdio.h>        /* printf(3) et al.                            */
#include <stdlib.h>       /* exit(2).                                    */
#include <string.h>       /* String manipulation & memory functions.     */
#include <sys/poll.h>     /* poll(2) and related definitions.            */
#include <sys/socket.h>   /* Socket functions (socket(2), bind(2), etc). */
#include <time.h>         /* time(2) & ctime(3).                         */
#include <unistd.h>       /* getopt(3), read(2), etc.                    */
/*
** Constants.
*/
#define DFLT_SERVICE "daytime"   /* Default service name.                  */
#define INVALID_DESC -1          /* Invalid file descriptor.               */
#define MAXCONNQLEN  3           /* Max nbr of connection requests to queue. */
#define MAXTCPSCKTS  2           /* One TCP socket for IPv4 & one for IPv6.  */
#define MAXUDPSCKTS  2           /* One UDP socket for IPv4 & one for IPv6.  */
#define VALIDOPTS    "v"         /* Valid command options.                 */
/*
** Simple boolean type definition.
*/
typedef enum { false = 0, true } boolean;
/*
** Prototypes for internal helper functions.
*/
static int  openSckt( const char *service,
                      const char *protocol,
                      int         desc[ ],
                      size_t     *descSize );
static void tod( int    tSckt[ ],
                 size_t tScktSize,
                 int    uSckt[ ],
                 size_t uScktSize );
/*
** Global (within this file only) data objects.
*/
static char        hostBfr[ NI_MAXHOST ];   /* For use w/getnameinfo(3).    */
static const char *pgmName;                 /* Program name w/o dir prefix. */
static char        servBfr[ NI_MAXSERV ];   /* For use w/getnameinfo(3).    */
static boolean     verbose = false;         /* Verbose mode indication.     */
/*
** Usage macro for command syntax violations.
*/
#define USAGE                                       \
```

```
        {                                           \
           fprintf( stderr,                         \
                    "Usage: %s [-v] [service]\n",   \
                    pgmName );                      \
           exit( 127 );                             \
        }  /* End USAGE macro. */
/*
** Macro to terminate the program if a system call error occurs.  The system
** call must be one of the usual type that returns -1 on error.  This macro is
** a modified version of a macro authored by Dr. V. Vinge, SDSU Dept. of
** Computer Science (retired)... best professor I ever had.  I hear he writes
** great science fiction in addition to robust code, too.
*/
#define CHK(expr)                                                       \
        do                                                              \
        {                                                               \
           if ( (expr) == -1 )                                          \
           {                                                            \
              fprintf( stderr,                                          \
                       "%s (line %d): System call ERROR - %s.\n",       \
                       pgmName,                                         \
                       __LINE__,                                        \
                       strerror( errno ) );                             \
              exit( 1 );                                                \
           }   /* End IF system call failed. */                         \
        } while ( false )
/*****************************************************************************
* Function: main
*
* Description:
*    Set up a time-of-day server and handle network requests.  This server
*    handles both TCP and UDP requests.
*
* Parameters:
*    The usual argc and argv parameters to a main() function.
*
* Return Value:
*    This is a daemon program and never returns.  However, in the degenerate
*    case where no sockets are created, the function returns zero.
*****************************************************************************/
int main( int   argc,
          char *argv[ ] )
{
   int         opt;
   const char *service   = DFLT_SERVICE;
   int         tSckt[ MAXTCPSCKTS ];     /* Array of TCP socket descriptors. */
   size_t      tScktSize = MAXTCPSCKTS;  /* Size of uSckt (# of elements).   */
   int         uSckt[ MAXUDPSCKTS ];     /* Array of UDP socket descriptors. */
   size_t      uScktSize = MAXUDPSCKTS;  /* Size of uSckt (# of elements).   */
   /*
   ** Set the program name (w/o directory prefix).
   */
   pgmName = strrchr( argv[ 0 ], '/' );
   pgmName = pgmName == NULL  ?  argv[ 0 ]  :  pgmName + 1;
   /*
   ** Process command options.
   */
```

```
   opterr = 0;   /* Turns off "invalid option" error messages. */
   while ( ( opt = getopt( argc, argv, VALIDOPTS ) ) >= 0 )
   {
      switch ( opt )
      {
         case 'v':   /* Verbose mode. */
         {
            verbose = true;
            break;
         }
         default:
         {
            USAGE;
         }
      }  /* End SWITCH on command option. */
   }  /* End WHILE processing options. */
   /*
   ** Process command line arguments.
   */
   switch ( argc - optind )
   {
      case 0:  break;
      case 1:  service = argv[ optind ]; break;
      default: USAGE;
   }  /* End SWITCH on number of command line arguments. */
   /*
   ** Open both a TCP and UDP socket, for both IPv4 & IPv6, on which to receive
   ** service requests.
   */
   if ( ( openSckt( service, "tcp", tSckt, &tScktSize ) < 0 ) ||
        ( openSckt( service, "udp", uSckt, &uScktSize ) < 0 ) )
   {
      exit( 1 );
   }
   /*
   ** Run the time-of-day server.
   */
   if ( ( tScktSize > 0 ) || ( uScktSize > 0 ) )
   {
      tod( tSckt,         /* tod() never returns. */
           tScktSize,
           uSckt,
           uScktSize );
   }
   /*
   ** Since tod() never returns, execution only gets here if no sockets were
   ** created.
   */
   if ( verbose )
   {
      fprintf( stderr,
               "%s: No sockets opened... terminating.\n",
               pgmName );
   }
   return 0;
}  /* End main() */
/*****************************************************************************
```

```
* Function: openSckt
*
* Description:
*    Open passive (server) sockets for the indicated inet service & protocol.
*    Notice in the last sentence that "sockets" is plural.  During the interim
*    transition period while everyone is switching over to IPv6, the server
*    application has to open two sockets on which to listen for connections...
*    one for IPv4 traffic and one for IPv6 traffic.
*
* Parameters:
*    service  - Pointer to a character string representing the well-known port
*               on which to listen (can be a service name or a decimal number).
*    protocol - Pointer to a character string representing the transport layer
*               protocol (only "tcp" or "udp" are valid).
*    desc     - Pointer to an array into which the socket descriptors are
*               placed when opened.
*    descSize - This is a value-result parameter.  On input, it contains the
*               max number of descriptors that can be put into 'desc' (i.e. the
*               number of elements in the array).  Upon return, it will contain
*               the number of descriptors actually opened.  Any unused slots in
*               'desc' are set to INVALID_DESC.
*
* Return Value:
*    0 on success, -1 on error.
**************************************************************************/
static int openSckt( const char *service,
                     const char *protocol,
                     int         desc[ ],
                     size_t     *descSize )
{
   struct addrinfo *ai;
   int              aiErr;
   struct addrinfo *aiHead;
   struct addrinfo  hints    = { .ai_flags  = AI_PASSIVE,    /* Server mode.
¬ */
                                 .ai_family = PF_UNSPEC };   /* IPv4 or IPv6.
¬ */
   size_t           maxDescs = *descSize;
   /*
   ** Initialize output parameters.  When the loop completes, *descSize is 0.
   */
   while ( *descSize > 0 )
   {
      desc[ --( *descSize ) ] = INVALID_DESC;
   }
   /*
   ** Check which protocol is selected (only TCP and UDP are valid).
   */
   if ( strcmp( protocol, "tcp" ) == 0 )        /* TCP protocol.     */
   {
      hints.ai_socktype = SOCK_STREAM;
      hints.ai_protocol = IPPROTO_TCP;
   }
   else if ( strcmp( protocol, "udp" ) == 0 )   /* UDP protocol.     */
   {
      hints.ai_socktype = SOCK_DGRAM;
      hints.ai_protocol = IPPROTO_UDP;
```

```
}
else                                        /* Invalid protocol. */
{
   fprintf( stderr,
            "%s (line %d): ERROR - Unknown transport "
            "layer protocol \"%s\".\n",
            pgmName,
            __LINE__,
            protocol );
   return -1;
}
/*
** Look up the service's well-known port number.  Notice that NULL is being
** passed for the 'node' parameter, and that the AI_PASSIVE flag is set in
** 'hints'.  Thus, the program is requesting passive address information.
** The network address is initialized to :: (all zeros) for IPv6 records, or
** 0.0.0.0 for IPv4 records.
*/
if ( ( aiErr = getaddrinfo( NULL,
                            service,
                            &hints,
                            &aiHead ) ) != 0 )
{
   fprintf( stderr,
            "%s (line %d): ERROR - %s.\n",
            pgmName,
            __LINE__,
            gai_strerror( aiErr ) );
   return -1;
}
/*
** For each of the address records returned, attempt to set up a passive
** socket.
*/
for ( ai = aiHead;
      ( ai != NULL ) && ( *descSize < maxDescs );
      ai = ai->ai_next )
{
   if ( verbose )
   {
      /*
      ** Display the current address info.   Start with the protocol-
      ** independent fields first.
      */
      fprintf( stderr,
               "Setting up a passive socket based on the "
               "following address info:\n"
               "   ai_flags     = 0x%02X\n"
               "   ai_family    = %d (PF_INET = %d, PF_INET6 = %d)\n"
               "   ai_socktype  = %d (SOCK_STREAM = %d, SOCK_DGRAM = %d)\n"
               "   ai_protocol  = %d (IPPROTO_TCP = %d, IPPROTO_UDP = %d)\n"
               "   ai_addrlen   = %d (sockaddr_in = %d, "
               "sockaddr_in6 = %d)\n",
               ai->ai_flags,
               ai->ai_family,
               PF_INET,
               PF_INET6,
```

```
         ai->ai_socktype,
         SOCK_STREAM,
         SOCK_DGRAM,
         ai->ai_protocol,
         IPPROTO_TCP,
         IPPROTO_UDP,
         ai->ai_addrlen,
         sizeof( struct sockaddr_in ),
         sizeof( struct sockaddr_in6 ) );
/*
** Now display the protocol-specific formatted socket address.  Note
** that the program is requesting that getnameinfo(3) convert the
** host & service into numeric strings.
*/
getnameinfo( ai->ai_addr,
             ai->ai_addrlen,
             hostBfr,
             sizeof( hostBfr ),
             servBfr,
             sizeof( servBfr ),
             NI_NUMERICHOST | NI_NUMERICSERV );
switch ( ai->ai_family )
{
   case PF_INET:   /* IPv4 address record. */
   {
      struct sockaddr_in *p = (struct sockaddr_in*) ai->ai_addr;
      fprintf( stderr,
               "   ai_addr      = sin_family:   %d (AF_INET = %d, "
               "AF_INET6 = %d)\n"
               "                  sin_addr:     %s\n"
               "                  sin_port:     %s\n",
               p->sin_family,
               AF_INET,
               AF_INET6,
               hostBfr,
               servBfr );
      break;
   }  /* End CASE of IPv4. */
   case PF_INET6:   /* IPv6 address record. */
   {
      struct sockaddr_in6 *p = (struct sockaddr_in6*) ai->ai_addr;
      fprintf( stderr,
               "   ai_addr      = sin6_family:   %d (AF_INET = %d, "
               "AF_INET6 = %d)\n"
               "                  sin6_addr:     %s\n"
               "                  sin6_port:     %s\n"
               "                  sin6_flowinfo: %d\n"
               "                  sin6_scope_id: %d\n",
               p->sin6_family,
               AF_INET,
               AF_INET6,
               hostBfr,
               servBfr,
               p->sin6_flowinfo,
               p->sin6_scope_id );
      break;
   }  /* End CASE of IPv6. */
```

```
            default:   /* Can never get here, but just for completeness. */
            {
               fprintf( stderr,
                        "%s (line %d): ERROR - Unknown protocol family (%d).\n",
                        pgmName,
                        __LINE__,
                        ai->ai_family );
               freeaddrinfo( aiHead );
               return -1;
            }  /* End DEFAULT case (unknown protocol family). */
         }  /* End SWITCH on protocol family. */
      }  /* End IF verbose mode. */
      /*
      ** Create a socket using the info in the addrinfo structure.
      */
      CHK( desc[ *descSize ] = socket( ai->ai_family,
                                       ai->ai_socktype,
                                       ai->ai_protocol ) );
      /*
      ** Here is the code that prevents "IPv4 mapped addresses", as discussed
      ** in Section 22.1.3.1.  If an IPv6 socket was just created, then set the
      ** IPV6_V6ONLY socket option.
      */
      if ( ai->ai_family == PF_INET6 )
      {
#if defined( IPV6_V6ONLY )
         /*
         ** Disable IPv4 mapped addresses.
         */
         int v6Only = 1;
         CHK( setsockopt( desc[ *descSize ],
                          IPPROTO_IPV6,
                          IPV6_V6ONLY,
                          &v6Only,
                          sizeof( v6Only ) ) );
#else
         /*
         ** IPV6_V6ONLY is not defined, so the socket option can't be set and
         ** thus IPv4 mapped addresses can't be disabled.  Print a warning
         ** message and close the socket.  Design note: If the
         ** #if...#else...#endif construct were removed, then this program
         ** would not compile (because IPV6_V6ONLY isn't defined).  That's an
         ** acceptable approach; IPv4 mapped addresses are certainly disabled
         ** if the program can't build!  However, since this program is also
         ** designed to work for IPv4 sockets as well as IPv6, I decided to
         ** allow the program to compile when IPV6_V6ONLY is not defined, and
         ** turn it into a run-time warning rather than a compile-time error.
         ** IPv4 mapped addresses are still disabled because _all_ IPv6 traffic
         ** is disabled (all IPv6 sockets are closed here), but at least this
         ** way the server can still service IPv4 network traffic.
         */
         fprintf( stderr,
                  "%s (line %d): WARNING - Cannot set IPV6_V6ONLY socket "
                  "option.  Closing IPv6 %s socket.\n",
                  pgmName,
                  __LINE__,
                  ai->ai_protocol == IPPROTO_TCP  ?  "TCP"  :  "UDP" );
```

```
         CHK( close( desc[ *descSize ] ) );
         continue;   /* Go to top of FOR loop w/o updating *descSize! */
#endif /* IPV6_V6ONLY */
      }  /* End IF this is an IPv6 socket. */
      /*
      ** Bind the socket.  Again, the info from the addrinfo structure is used.
      */
      CHK( bind( desc[ *descSize ],
                 ai->ai_addr,
                 ai->ai_addrlen ) );
      /*
      ** If this is a TCP socket, put the socket into passive listening mode
      ** (listen is only valid on connection-oriented sockets).
      */
      if ( ai->ai_socktype == SOCK_STREAM )
      {
         CHK( listen( desc[ *descSize ],
                      MAXCONNQLEN ) );
      }
      /*
      ** Socket set up okay.  Bump index to next descriptor array element.
      */
      *descSize += 1;
   }  /* End FOR each address info structure returned. */
   /*
   ** Dummy check for unused address records.
   */
   if ( verbose && ( ai != NULL ) )
   {
      fprintf( stderr,
               "%s (line %d): WARNING - Some address records were "
               "not processed due to insufficient array space.\n",
               pgmName,
               __LINE__ );
   }  /* End IF verbose and some address records remain unprocessed. */
   /*
   ** Clean up.
   */
   freeaddrinfo( aiHead );
   return 0;
}  /* End openSckt() */
/*****************************************************************************
* Function: tod
*
* Description:
*    Listen on a set of sockets and send the current time-of-day to any
*    clients.  This function never returns.
*
* Parameters:
*    tSckt     - Array of TCP socket descriptors on which to listen.
*    tScktSize - Size of the tSckt array (nbr of elements).
*    uSckt     - Array of UDP socket descriptors on which to listen.
*    uScktSize - Size of the uSckt array (nbr of elements).
*
* Return Value: None.
*****************************************************************************/
static void tod( int    tSckt[ ],
```

```
                size_t tScktSize,
                int    uSckt[ ],
                size_t uScktSize )
{
   char                     bfr[ 256 ];
   ssize_t                  count;
   struct pollfd           *desc;
   size_t                   descSize = tScktSize + uScktSize;
   int                      idx;
   int                      newSckt;
   struct sockaddr         *sadr;
   socklen_t                sadrLen;
   struct sockaddr_storage  sockStor;
   int                      status;
   size_t                   timeLen;
   char                    *timeStr;
   time_t                   timeVal;
   ssize_t                  wBytes;
   /*
   ** Allocate memory for the poll(2) array.
   */
   desc = malloc( descSize * sizeof( struct pollfd ) );
   if ( desc == NULL )
   {
      fprintf( stderr,
               "%s (line %d): ERROR - %s.\n",
               pgmName,
               __LINE__,
               strerror( ENOMEM ) );
      exit( 1 );
   }
   /*
   ** Initialize the poll(2) array.
   */
   for ( idx = 0;     idx < descSize;     idx++ )
   {
      desc[ idx ].fd      = idx < tScktSize  ?  tSckt[ idx ]
                                             :  uSckt[ idx - tScktSize ];
      desc[ idx ].events  = POLLIN;
      desc[ idx ].revents = 0;
   }
   /*
   ** Main time-of-day server loop.  Handles both TCP & UDP requests.  This is
   ** an interative server, and all requests are handled directly within the
   ** main loop.
   */
   while ( true )   /* Do forever. */
   {
      /*
      ** Wait for activity on one of the sockets.  The DO..WHILE construct is
      ** used to restart the system call in the event the process is
      ** interrupted by a signal.
      */
      do
      {
         status = poll( desc,
                        descSize,
```

```
                  -1 /* Wait indefinitely for input. */ );
} while ( ( status < 0 ) && ( errno == EINTR ) );
CHK( status );   /* Check for a bona fide system call error. */
/*
** Get the current time.
*/
timeVal = time( NULL );
timeStr = ctime( &timeVal );
timeLen = strlen( timeStr );
/*
** Indicate that there is new network activity.
*/
if ( verbose )
{
   char *s = malloc( timeLen+1 );
   strcpy( s, timeStr );
   s[ timeLen-1 ] = '\0';   /* Overwrite '\n' in date string. */
   fprintf( stderr,
            "%s: New network activity on %s.\n",
            pgmName,
            s );
   free( s );
}  /* End IF verbose. */
/*
** Process sockets with input available.
*/
for ( idx = 0;     idx < descSize;     idx++ )
{
   switch ( desc[ idx ].revents )
   {
      case 0:        /* No activity on this socket; try the next. */
         continue;
      case POLLIN:   /* Network activity.  Go process it.         */
         break;
      default:       /* Invalid poll events.                      */
      {
         fprintf( stderr,
                  "%s (line %d): ERROR - Invalid poll event (0x%02X).\n",
                  pgmName,
                  __LINE__,
                  desc[ idx ].revents );
         exit( 1 );
      }
   }  /* End SWITCH on returned poll events. */
   /*
   ** Determine if this is a TCP request or UDP request.
   */
   if ( idx < tScktSize )
   {
      /*
      ** TCP connection requested.  Accept it.  Notice the use of
      ** the sockaddr_storage data type.
      */
      sadrLen = sizeof( sockStor );
      sadr    = (struct sockaddr*) &sockStor;
      CHK( newSckt = accept( desc[ idx ].fd,
                             sadr,
```

```
                        &sadrLen ) );
CHK( shutdown( newSckt,       /* Server never recv's anything. */
               SHUT_RD ) );
if ( verbose )
{
   /*
   ** Display the socket address of the remote client.  Begin with
   ** the address-independent fields.
   */
   fprintf( stderr,
            "Sockaddr info for new TCP client:\n"
            "   sa_family = %d (AF_INET = %d, AF_INET6 = %d)\n"
            "   addr len  = %d (sockaddr_in = %d, "
            "sockaddr_in6 = %d)\n",
            sadr->sa_family,
            AF_INET,
            AF_INET6,
            sadrLen,
            sizeof( struct sockaddr_in ),
            sizeof( struct sockaddr_in6 ) );
   /*
   ** Display the address-specific fields.
   */
   getnameinfo( sadr,
                sadrLen,
                hostBfr,
                sizeof( hostBfr ),
                servBfr,
                sizeof( servBfr ),
                NI_NUMERICHOST | NI_NUMERICSERV );
   /*
   ** Notice that we're switching on an address family now, not a
   ** protocol family.
   */
   switch ( sadr->sa_family )
   {
      case AF_INET:   /* IPv4 address. */
      {
         struct sockaddr_in *p = (struct sockaddr_in*) sadr;
         fprintf( stderr,
                  "   sin_addr  = sin_family: %d\n"
                  "               sin_addr:   %s\n"
                  "               sin_port:   %s\n",
                  p->sin_family,
                  hostBfr,
                  servBfr );
         break;
      }  /* End CASE of IPv4. */
      case AF_INET6:   /* IPv6 address. */
      {
         struct sockaddr_in6 *p = (struct sockaddr_in6*) sadr;
         fprintf( stderr,
                  "   sin6_addr = sin6_family:   %d\n"
                  "               sin6_addr:     %s\n"
                  "               sin6_port:     %s\n"
                  "               sin6_flowinfo: %d\n"
                  "               sin6_scope_id: %d\n",
```

```
                        p->sin6_family,
                        hostBfr,
                        servBfr,
                        p->sin6_flowinfo,
                        p->sin6_scope_id );
               break;
            }  /* End CASE of IPv6. */
            default:   /* Can never get here, but for completeness. */
            {
               fprintf( stderr,
                        "%s (line %d): ERROR - Unknown address "
                        "family (%d).\n",
                        pgmName,
                        __LINE__,
                        sadr->sa_family );
               break;
            }  /* End DEFAULT case (unknown address family). */
         }  /* End SWITCH on address family. */
      }  /* End IF verbose mode. */
      /*
      ** Send the TOD to the client.
      */
      wBytes = timeLen;
      while ( wBytes > 0 )
      {
         do
         {
            count = write( newSckt,
                           timeStr,
                           wBytes );
         } while ( ( count < 0 ) && ( errno == EINTR ) );
         CHK( count );   /* Check for a bona fide error. */
         wBytes -= count;
      }  /* End WHILE there is data to send. */
      CHK( close( newSckt ) );
   }  /* End IF this was a TCP connection request. */
   else
   {
      /*
      ** This is a UDP socket, and a datagram is available.  The funny
      ** thing about UDP requests is that this server doesn't require any
      ** client input; but it can't send the TOD unless it knows a client
      ** wants the data, and the only way that can occur with UDP is if
      ** the server receives a datagram from the client.  Thus, the
      ** server must receive _something_, but the content of the datagram
      ** is irrelevant.  Read in the datagram.  Again note the use of
      ** sockaddr_storage to receive the address.
      */
      sadrLen = sizeof( sockStor );
      sadr    = (struct sockaddr*) &sockStor;
      CHK( count = recvfrom( desc[ idx ].fd,
                             bfr,
                             sizeof( bfr ),
                             0,
                             sadr,
                             &sadrLen ) );
      /*
```

```
** Display whatever was received on stdout.
*/
if ( verbose )
{
   ssize_t rBytes = count;
   fprintf( stderr,
            "%s: UDP datagram received (%d bytes).\n",
            pgmName,
            count );
   while ( count > 0 )
   {
      fputc( bfr[ rBytes - count-- ],
             stdout );
   }
   if ( bfr[ rBytes-1 ] != '\n' )
      fputc( '\n', stdout );   /* Newline also flushes stdout. */
   /*
   ** Display the socket address of the remote client.  Address-
   ** independent fields first.
   */
   fprintf( stderr,
            "Remote client's sockaddr info:\n"
            "   sa_family = %d (AF_INET = %d, AF_INET6 = %d)\n"
            "   addr len  = %d (sockaddr_in = %d, "
            "sockaddr_in6 = %d)\n",
            sadr->sa_family,
            AF_INET,
            AF_INET6,
            sadrLen,
            sizeof( struct sockaddr_in ),
            sizeof( struct sockaddr_in6 ) );
   /*
   ** Display the address-specific information.
   */
   getnameinfo( sadr,
                sadrLen,
                hostBfr,
                sizeof( hostBfr ),
                servBfr,
                sizeof( servBfr ),
                NI_NUMERICHOST | NI_NUMERICSERV );
   switch ( sadr->sa_family )
   {
      case AF_INET:   /* IPv4 address. */
      {
         struct sockaddr_in *p = (struct sockaddr_in*) sadr;
         fprintf( stderr,
                  "   sin_addr  = sin_family: %d\n"
                  "               sin_addr:   %s\n"
                  "               sin_port:   %s\n",
                  p->sin_family,
                  hostBfr,
                  servBfr );
         break;
      }  /* End CASE of IPv4 address. */
      case AF_INET6:   /* IPv6 address. */
      {
```

```
                  struct sockaddr_in6 *p = (struct sockaddr_in6*) sadr;
                  fprintf( stderr,
                           "   sin6_addr = sin6_family:   %d\n"
                           "               sin6_addr:     %s\n"
                           "               sin6_port:     %s\n"
                           "               sin6_flowinfo: %d\n"
                           "               sin6_scope_id: %d\n",
                           p->sin6_family,
                           hostBfr,
                           servBfr,
                           p->sin6_flowinfo,
                           p->sin6_scope_id );
                  break;
               }  /* End CASE of IPv6 address. */
               default:   /* Can never get here, but for completeness. */
               {
                  fprintf( stderr,
                           "%s (line %d): ERROR - Unknown address "
                           "family (%d).\n",
                           pgmName,
                           __LINE__,
                           sadr->sa_family );
                  break;
               }  /* End DEFAULT case (unknown address family). */
            }  /* End SWITCH on address family. */
         }  /* End IF verbose mode. */
         /*
         ** Send the time-of-day to the client.
         */
         wBytes = timeLen;
         while ( wBytes > 0 )
         {
            do
            {
               count = sendto( desc[ idx ].fd,
                               timeStr,
                               wBytes,
                               0,
                               sadr,        /* Address & address length   */
                               sadrLen );   /*    received in recvfrom(). */
            } while ( ( count < 0 ) && ( errno == EINTR ) );
            CHK( count );   /* Check for a bona fide error. */
            wBytes -= count;
         }  /* End WHILE there is data to send. */
      }  /* End ELSE a UDP datagram is available. */
      desc[ idx ].revents = 0;   /* Clear the returned poll events. */
   }  /* End FOR each socket descriptor. */
}  /* End WHILE forever. */
}  /* End tod() */
```

23.1.4.2. 'Daytime' TCP Client Code

The TCP client code is found in file tod6tc.c (time-of-day IPv6 TCP client). Once built, the TCP client may be started using the following command syntax (assuming tod6tc is the executable file):

```
tod6tc [-v] [-s scope_id] [host [service]]
```

ARGUMENTS:

host

The hostname or IP address (dotted decimal or colon-hex) of the remote host providing the service. Default is "localhost".

service

The TCP service (or well-known port number) to which a connection attempt is made. Default is "daytime".

OPTIONS:

-s

This option is only meaningful for IPv6 addresses, and is used to set the scope identifier (i.e. the network interface on which to establish the connection). Default is "eth0". If host is a scoped address, this option is ignored.

-v

Turn on verbose mode.

The TCP client source code contained in tod6tc.c follows:

```
/*************************************************************************
* File: tod6tc.c
* Description: Contains source code for an IPv6-capable 'daytime' TCP client.
* Author: John Wenker, Sr. Software Engineer
*         Performance Technologies, San Diego, USA
*************************************************************************/
/*
** System header files.
*/
#include <errno.h>        /* errno declaration and error codes.             */
#include <net/if.h>       /* if_nametoindex(3).                             */
#include <netdb.h>        /* getaddrinfo(3) and associated definitions.     */
#include <netinet/in.h>   /* sockaddr_in and sockaddr_in6 definitions.      */
#include <stdio.h>        /* printf(3) et al.                               */
#include <stdlib.h>       /* exit(2).                                       */
#include <string.h>       /* String manipulation and memory functions.      */
#include <sys/socket.h>   /* Socket functions (socket(2), connect(2), etc). */
#include <unistd.h>       /* getopt(3), read(2), etc.                       */
/*
** Constants & macros.
*/
#define DFLT_HOST      "localhost"   /* Default server name.              */
#define DFLT_SCOPE_ID  "eth0"        /* Default scope identifier.         */
#define DFLT_SERVICE   "daytime"     /* Default service name.             */
#define INVALID_DESC   -1            /* Invalid file (socket) descriptor. */
#define MAXBFRSIZE     256           /* Max bfr sz to read remote TOD.    */
#define VALIDOPTS      "s:v"         /* Valid command options.            */
/*
```

```
** Type definitions (for convenience).
*/
typedef enum { false = 0, true } boolean;
typedef struct sockaddr_in       sockaddr_in_t;
typedef struct sockaddr_in6      sockaddr_in6_t;
/*
** Prototypes for internal helper functions.
*/
static int  openSckt( const char   *host,
                      const char   *service,
                      unsigned int  scopeId );
static void tod( int sckt );
/*
** Global (within this file only) data objects.
*/
static const char *pgmName;           /* Program name (w/o directory). */
static boolean     verbose = false;   /* Verbose mode.                 */
/*
** Usage macro.
*/
#define USAGE                                                           \
        {                                                               \
           fprintf( stderr,                                             \
                    "Usage: %s [-v] [-s scope_id] [host [service]]\n",  \
                    pgmName );                                          \
           exit( 127 );                                                 \
        }  /* End USAGE macro. */
/*
** This "macro" (even though it's really a function) is loosely based on the
** CHK() macro by Dr. V. Vinge (see server code).  The status parameter is
** a boolean expression indicating the return code from one of the usual system
** calls that returns -1 on error.  If a system call error occurred, an alert
** is written to stderr.  It returns a boolean value indicating success/failure
** of the system call.
**
** Example: if ( !SYSCALL( "write",
**                         count = write( fd, bfr, size ) ) )
**          {
**             // Error processing... but SYSCALL() will have already taken
**             // care of dumping an error alert to stderr.
**          }
*/
static __inline boolean SYSCALL( const char *syscallName,
                                 int         lineNbr,
                                 int         status )
{
   if ( ( status == -1 ) && verbose )
   {
      fprintf( stderr,
               "%s (line %d): System call failed ('%s') - %s.\n",
               pgmName,
               lineNbr,
               syscallName,
               strerror( errno ) );
   }
   return status != -1;   /* True if the system call was successful. */
}  /* End SYSCALL() */
```

```
/*****************************************************************************
* Function: main
*
* Description:
*    Connect to a remote time-of-day service and write the remote host's TOD to
*    stdout.
*
* Parameters:
*    The usual argc & argv parameters to a main() program.
*
* Return Value:
*    This function always returns zero.
*****************************************************************************/
int main( int   argc,
          char *argv[ ] )
{
   const char   *host     = DFLT_HOST;
   int           opt;
   int           sckt;
   unsigned int  scopeId  = if_nametoindex( DFLT_SCOPE_ID );
   const char   *service  = DFLT_SERVICE;
   /*
   ** Determine the program name (w/o directory prefix).
   */
   pgmName = (const char*) strrchr( argv[ 0 ], '/' );
   pgmName = pgmName == NULL  ?  argv[ 0 ]  :  pgmName+1;
   /*
   ** Process command line options.
   */
   opterr = 0;   /* Turns off "invalid option" error messages. */
   while ( ( opt = getopt( argc, argv, VALIDOPTS ) ) != -1 )
   {
      switch ( opt )
      {
         case 's':   /* Scope identifier (IPv6 kluge). */
         {
            scopeId = if_nametoindex( optarg );
            if ( scopeId == 0 )
            {
               fprintf( stderr,
                        "%s: Unknown network interface (%s).\n",
                        pgmName,
                        optarg );
               USAGE;
            }
            break;
         }
         case 'v':   /* Verbose mode. */
         {
            verbose = true;
            break;
         }
         default:
         {
            USAGE;
         }
      }  /* End SWITCH on command option. */
```

```
} /* End WHILE processing command options. */
/*
** Process command arguments.  At the end of the above loop, optind is the
** index of the first NON-option argv element.
*/
switch ( argc - optind )
{
   case 2:   /* Both host & service are specified on the command line. */
   {
       service = argv[ optind + 1 ];
       /***** Fall through *****/
   }
   case 1:   /* Host is specified on the command line. */
   {
       host = argv[ optind ];
       /***** Fall through *****/
   }
   case 0:   /* Use default host & service. */
   {
       break;
   }
   default:
   {
      USAGE;
   }
}  /* End SWITCH on number of command arguments. */
/*
** Open a connection to the indicated host/service.
**
** Note that if all three of the following conditions are met, then the
** scope identifier remains unresolved at this point.
**    1) The default network interface is unknown for some reason.
**    2) The -s option was not used on the command line.
**    3) An IPv6 "scoped address" was not specified for the hostname on the
**       command line.
** If the above three conditions are met, then only an IPv4 socket can be
** opened (connect(2) fails without the scope ID properly set for IPv6
** sockets).
*/
if ( ( sckt = openSckt( host,
                        service,
                        scopeId ) ) == INVALID_DESC )
{
   fprintf( stderr,
            "%s: Sorry... a connection could not be established.\n",
            pgmName );
   exit( 1 );
}
/*
** Get the remote time-of-day.
*/
tod( sckt );
/*
** Close the connection and terminate.
*/
(void) SYSCALL( "close",
                __LINE__,
```

```
                  close( sckt ) );
   return 0;
}  /* End main() */
/*****************************************************************************
* Function: openSckt
*
* Description:
*    Sets up a TCP connection to a remote server.  Getaddrinfo(3) is used to
*    perform lookup functions and can return multiple address records (i.e. a
*    list of 'struct addrinfo' records).  This function traverses the list and
*    tries to establish a connection to the remote server.  The function ends
*    when either a connection has been established or all records in the list
*    have been processed.
*
* Parameters:
*    host    - A pointer to a character string representing the hostname or IP
*              address (IPv4 or IPv6) of the remote server.
*    service - A pointer to a character string representing the service name or
*              well-known port number.
*    scopeId - For IPv6 sockets only.  This is the index corresponding to the
*              network interface on which to set up the connection.  This
*              parameter is ignored for IPv4 sockets or when an IPv6 "scoped
*              address" is specified in 'host' (i.e. where the colon-hex
*              network address is augmented with the scope ID).
*
* Return Value:
*    Returns the socket descriptor for the connection, or INVALID_DESC if all
*    address records have been processed and a connection could not be
*    established.
*****************************************************************************/
static int openSckt( const char   *host,
                     const char   *service,
                     unsigned int  scopeId )
{
   struct addrinfo *ai;
   int              aiErr;
   struct addrinfo *aiHead;
   struct addrinfo  hints;
   sockaddr_in6_t  *pSadrIn6;
   int              sckt;
   /*
   ** Initialize the 'hints' structure for getaddrinfo(3).
   **
   ** Notice that the 'ai_family' field is set to PF_UNSPEC, indicating to
   ** return both IPv4 and IPv6 address records for the host/service.  Most of
   ** the time, the user isn't going to care whether an IPv4 connection or an
   ** IPv6 connection is established; the user simply wants to exchange data
   ** with the remote host and doesn't care how it's done.  Sometimes, however,
   ** the user might want to explicitly specify the type of underlying socket.
   ** It is left as an exercise for the motivated reader to add a command line
   ** option allowing the user to specify the IP protocol, and then process the
   ** list of addresses accordingly (it's not that difficult).
   */
   memset( &hints, 0, sizeof( hints ) );
   hints.ai_family   = PF_UNSPEC;     /* IPv4 or IPv6 records (don't care). */
   hints.ai_socktype = SOCK_STREAM;   /* Connection-oriented byte stream.   */
   hints.ai_protocol = IPPROTO_TCP;   /* TCP transport layer protocol only. */
```

```
/*
** Look up the host/service information.
*/
if ( ( aiErr = getaddrinfo( host,
                            service,
                            &hints,
                            &aiHead ) ) != 0 )
{
   fprintf( stderr,
            "%s (line %d): ERROR - %s.\n",
            pgmName,
            __LINE__,
            gai_strerror( aiErr ) );
   return INVALID_DESC;
}
/*
** Go through the list and try to open a connection.  Continue until either
** a connection is established or the entire list is exhausted.
*/
for ( ai = aiHead,   sckt = INVALID_DESC;
      ( ai != NULL ) && ( sckt == INVALID_DESC );
      ai = ai->ai_next )
{
   /*
   ** IPv6 kluge.  Make sure the scope ID is set.
   */
   if ( ai->ai_family == PF_INET6 )
   {
      pSadrIn6 = (sockaddr_in6_t*) ai->ai_addr;
      if ( pSadrIn6->sin6_scope_id == 0 )
      {
         pSadrIn6->sin6_scope_id = scopeId;
      }  /* End IF the scope ID wasn't set. */
   }  /* End IPv6 kluge. */
   /*
   ** Display the address info for the remote host.
   */
   if ( verbose )
   {
      /*
      ** Temporary character string buffers for host & service.
      */
      char hostBfr[ NI_MAXHOST ];
      char servBfr[ NI_MAXSERV ];
      /*
      ** Display the address information just fetched.  Start with the
      ** common (protocol-independent) stuff first.
      */
      fprintf( stderr,
               "Address info:\n"
               "   ai_flags     = 0x%02X\n"
               "   ai_family    = %d (PF_INET = %d, PF_INET6 = %d)\n"
               "   ai_socktype  = %d (SOCK_STREAM = %d, SOCK_DGRAM = %d)\n"
               "   ai_protocol  = %d (IPPROTO_TCP = %d, IPPROTO_UDP = %d)\n"
               "   ai_addrlen   = %d (sockaddr_in = %d, "
               "sockaddr_in6 = %d)\n",
               ai->ai_flags,
```

```
         ai->ai_family,
         PF_INET,
         PF_INET6,
         ai->ai_socktype,
         SOCK_STREAM,
         SOCK_DGRAM,
         ai->ai_protocol,
         IPPROTO_TCP,
         IPPROTO_UDP,
         ai->ai_addrlen,
         sizeof( struct sockaddr_in ),
         sizeof( struct sockaddr_in6 ) );
/*
** Display the protocol-specific formatted address.
*/
getnameinfo( ai->ai_addr,
             ai->ai_addrlen,
             hostBfr,
             sizeof( hostBfr ),
             servBfr,
             sizeof( servBfr ),
             NI_NUMERICHOST | NI_NUMERICSERV );
switch ( ai->ai_family )
{
   case PF_INET:   /* IPv4 address record. */
   {
      sockaddr_in_t *pSadrIn = (sockaddr_in_t*) ai->ai_addr;
      fprintf( stderr,
               "   ai_addr      = sin_family: %d (AF_INET = %d, "
               "AF_INET6 = %d)\n"
               "                  sin_addr:   %s\n"
               "                  sin_port:   %s\n",
               pSadrIn->sin_family,
               AF_INET,
               AF_INET6,
               hostBfr,
               servBfr );
      break;
   }  /* End CASE of IPv4 record. */
   case PF_INET6:   /* IPv6 address record. */
   {
      pSadrIn6 = (sockaddr_in6_t*) ai->ai_addr;
      fprintf( stderr,
               "   ai_addr      = sin6_family:   %d (AF_INET = %d, "
               "AF_INET6 = %d)\n"
               "                  sin6_addr:     %s\n"
               "                  sin6_port:     %s\n"
               "                  sin6_flowinfo: %d\n"
               "                  sin6_scope_id: %d\n",
               pSadrIn6->sin6_family,
               AF_INET,
               AF_INET6,
               hostBfr,
               servBfr,
               pSadrIn6->sin6_flowinfo,
               pSadrIn6->sin6_scope_id );
      break;
```

```
            }  /* End CASE of IPv6 record. */
            default:   /* Can never get here, but just for completeness. */
            {
               fprintf( stderr,
                        "%s (line %d): ERROR - Unknown protocol family (%d).\n",
                        pgmName,
                        __LINE__,
                        ai->ai_family );
               break;
            }  /* End DEFAULT case (unknown protocol family). */
         }  /* End SWITCH on protocol family. */
      }  /* End IF verbose mode. */
      /*
      ** Create a socket.
      */
      if ( !SYSCALL( "socket",
                     __LINE__,
                     sckt = socket( ai->ai_family,
                                    ai->ai_socktype,
                                    ai->ai_protocol ) ) )
      {
         sckt = INVALID_DESC;
         continue;   /* Try the next address record in the list. */
      }
      /*
      ** Connect to the remote host.
      */
      if ( !SYSCALL( "connect",
                     __LINE__,
                     connect( sckt,
                              ai->ai_addr,
                              ai->ai_addrlen ) ) )
      {
         (void) close( sckt );   /* Could use SYSCALL() again here, but why? */
         sckt = INVALID_DESC;
         continue;   /* Try the next address record in the list. */
      }
   }  /* End FOR each address record returned by getaddrinfo(3). */
   /*
   ** Clean up & return.
   */
   freeaddrinfo( aiHead );
   return sckt;
}  /* End openSckt() */
/*****************************************************************************
* Function: tod
*
* Description:
*    Receive the time-of-day from the remote server and write it to stdout.
*
* Parameters:
*    sckt - The socket descriptor for the connection.
*
* Return Value: None.
*****************************************************************************/
static void tod( int sckt )
{
```

```
   char bfr[ MAXBFRSIZE+1 ];
   int  inBytes;
   /*
   ** The client never sends anything, so shut down the write side of the
   ** connection.
   */
   if ( !SYSCALL( "shutdown",
                  __LINE__,
                  shutdown( sckt, SHUT_WR ) ) )
   {
      return;
   }
   /*
   ** Read the time-of-day from the remote host.
   */
   do
   {
      if ( !SYSCALL( "read",
                     __LINE__,
                     inBytes = read( sckt,
                                     bfr,
                                     MAXBFRSIZE ) ) )
      {
         return;
      }
      bfr[ inBytes ] = '\0';   /* Null-terminate the received string. */
      fputs( bfr, stdout );    /* Null string if EOF (inBytes == 0).  */
   } while ( inBytes > 0 );
   fflush( stdout );
}  /* End tod() */
```

23.1.4.3. 'Daytime' UDP Client Code

The UDP client code is found in file tod6uc.c (time-of-day IPv6 UDP client). It is almost an exact duplicate of the TCP client (and in fact was derived from it), but is included in this HowTo for completeness. Once built, the UDP client may be started using the following command syntax (assuming tod6uc is the executable file):

```
tod6uc [-v] [-s scope_id] [host [service]]
```

ARGUMENTS:

host

The hostname or IP address (dotted decimal or colon-hex) of the remote host providing the service. Default is "localhost".

service

The UDP service (or well-known port number) to which datagrams are sent. Default is "daytime".

OPTIONS:

-s

This option is only meaningful for IPv6 addresses, and is used to set the scope identifier (i.e. the network interface on which to exchange datagrams). Default is "eth0". If host is a scoped address, this option is ignored.

-v

Turn on verbose mode.

The UDP client source code contained in tod6uc.c follows:

```
/*****************************************************************************
* File: tod6uc.c
* Description: Contains source code for an IPv6-capable 'daytime' UDP client.
* Author: John Wenker, Sr. Software Engineer
*         Performance Technologies, San Diego, USA
*****************************************************************************/
/*
** System header files.
*/
#include <errno.h>        /* errno declaration and error codes.             */
#include <net/if.h>       /* if_nametoindex(3).                             */
#include <netdb.h>        /* getaddrinfo(3) and associated definitions.     */
#include <netinet/in.h>   /* sockaddr_in and sockaddr_in6 definitions.      */
#include <stdio.h>        /* printf(3) et al.                               */
#include <stdlib.h>       /* exit(2).                                       */
#include <string.h>       /* String manipulation and memory functions.      */
#include <sys/socket.h>   /* Socket functions (socket(2), connect(2), etc). */
#include <unistd.h>       /* getopt(3), recvfrom(2), sendto(2), etc.        */
/*
** Constants & macros.
*/
#define DFLT_HOST      "localhost"   /* Default server name.              */
#define DFLT_SCOPE_ID  "eth0"        /* Default scope identifier.         */
#define DFLT_SERVICE   "daytime"     /* Default service name.             */
#define INVALID_DESC   -1            /* Invalid file (socket) descriptor. */
#define MAXBFRSIZE     256           /* Max bfr sz to read remote TOD.    */
#define VALIDOPTS      "s:v"         /* Valid command options.            */
/*
** Type definitions (for convenience).
*/
typedef enum { false = 0, true } boolean;
typedef struct sockaddr_in       sockaddr_in_t;
typedef struct sockaddr_in6      sockaddr_in6_t;
/*
** Prototypes for internal helper functions.
*/
static int  openSckt( const char   *host,
                      const char   *service,
                      unsigned int  scopeId );
static void tod( int sckt );
/*
** Global (within this file only) data objects.
*/
static const char *pgmName;           /* Program name (w/o directory). */
static boolean     verbose = false;   /* Verbose mode.                 */
/*
** Usage macro.
```

```
*/
#define USAGE                                                                   \
        {                                                                       \
           fprintf( stderr,                                                     \
                    "Usage: %s [-v] [-s scope_id] [host [service]]\n",          \
                    pgmName );                                                  \
           exit( 127 );                                                         \
        }  /* End USAGE macro. */
/*
** This "macro" (even though it's really a function) is loosely based on the
** CHK() macro by Dr. V. Vinge (see server code).  The status parameter is
** a boolean expression indicating the return code from one of the usual system
** calls that returns -1 on error.  If a system call error occurred, an alert
** is written to stderr.  It returns a boolean value indicating success/failure
** of the system call.
**
** Example: if ( !SYSCALL( "write",
**                         count = write( fd, bfr, size ) ) )
**          {
**             // Error processing... but SYSCALL() will have already taken
**             // care of dumping an error alert to stderr.
**          }
*/
static __inline boolean SYSCALL( const char *syscallName,
                                 int         lineNbr,
                                 int         status )
{
   if ( ( status == -1 ) && verbose )
   {
      fprintf( stderr,
               "%s (line %d): System call failed ('%s') - %s.\n",
               pgmName,
               lineNbr,
               syscallName,
               strerror( errno ) );
   }
   return status != -1;   /* True if the system call was successful. */
}  /* End SYSCALL() */
/*****************************************************************************
* Function: main
*
* Description:
*    Connect to a remote time-of-day service and write the remote host's TOD to
*    stdout.
*
* Parameters:
*    The usual argc & argv parameters to a main() program.
*
* Return Value:
*    This function always returns zero.
*****************************************************************************/
int main( int   argc,
          char *argv[ ] )
{
   const char   *host     = DFLT_HOST;
   int           opt;
   int           sckt;
```

```
unsigned int  scopeId  = if_nametoindex( DFLT_SCOPE_ID );
const char   *service  = DFLT_SERVICE;
/*
** Determine the program name (w/o directory prefix).
*/
pgmName = (const char*) strrchr( argv[ 0 ], '/' );
pgmName = pgmName == NULL  ?  argv[ 0 ]  :  pgmName+1;
/*
** Process command line options.
*/
opterr = 0;   /* Turns off "invalid option" error messages. */
while ( ( opt = getopt( argc, argv, VALIDOPTS ) ) != -1 )
{
   switch ( opt )
   {
      case 's':   /* Scope identifier (IPv6 kluge). */
      {
         scopeId = if_nametoindex( optarg );
         if ( scopeId == 0 )
         {
            fprintf( stderr,
                     "%s: Unknown network interface (%s).\n",
                     pgmName,
                     optarg );
            USAGE;
         }
         break;
      }
      case 'v':   /* Verbose mode. */
      {
         verbose = true;
         break;
      }
      default:
      {
         USAGE;
      }
   }  /* End SWITCH on command option. */
} /* End WHILE processing command options. */
/*
** Process command arguments.  At the end of the above loop, optind is the
** index of the first NON-option argv element.
*/
switch ( argc - optind )
{
   case 2:   /* Both host & service are specified on the command line. */
   {
       service = argv[ optind + 1 ];
       /***** Fall through *****/
   }
   case 1:   /* Host is specified on the command line. */
   {
       host = argv[ optind ];
       /***** Fall through *****/
   }
   case 0:   /* Use default host & service. */
   {
```

```
            break;
         }
         default:
         {
            USAGE;
         }
      }  /* End SWITCH on number of command arguments. */
      /*
      ** Open a connection to the indicated host/service.
      **
      ** Note that if all three of the following conditions are met, then the
      ** scope identifier remains unresolved at this point.
      **    1) The default network interface is unknown for some reason.
      **    2) The -s option was not used on the command line.
      **    3) An IPv6 "scoped address" was not specified for the hostname on the
      **       command line.
      ** If the above three conditions are met, then only an IPv4 socket can be
      ** opened (connect(2) fails without the scope ID properly set for IPv6
      ** sockets).
      */
      if ( ( sckt = openSckt( host,
                              service,
                              scopeId ) ) == INVALID_DESC )
      {
         fprintf( stderr,
                  "%s: Sorry... a connectionless socket could "
                  "not be set up.\n",
                  pgmName );
         exit( 1 );
      }
      /*
      ** Get the remote time-of-day.
      */
      tod( sckt );
      /*
      ** Close the connection and terminate.
      */
      (void) SYSCALL( "close",
                      __LINE__,
                      close( sckt ) );
      return 0;
   }  /* End main() */
   /************************************************************************
   * Function: openSckt
   *
   * Description:
   *    Sets up a UDP socket to a remote server.  Getaddrinfo(3) is used to
   *    perform lookup functions and can return multiple address records (i.e. a
   *    list of 'struct addrinfo' records).  This function traverses the list and
   *    tries to establish a connection to the remote server.  The function ends
   *    when either a connection has been established or all records in the list
   *    have been processed.
   *
   * Parameters:
   *    host    - A pointer to a character string representing the hostname or IP
   *              address (IPv4 or IPv6) of the remote server.
   *    service - A pointer to a character string representing the service name or
```

```
*              well-known port number.
*    scopeId - For IPv6 sockets only.  This is the index corresponding to the
*              network interface on which to exchange datagrams.  This
*              parameter is ignored for IPv4 sockets or when an IPv6 "scoped
*              address" is specified in 'host' (i.e. where the colon-hex
*              network address is augmented with the scope ID).
*
* Return Value:
*    Returns the socket descriptor for the connection, or INVALID_DESC if all
*    address records have been processed and a socket could not be initialized.
*****************************************************************************/
static int openSckt( const char   *host,
                     const char   *service,
                     unsigned int  scopeId )
{
   struct addrinfo *ai;
   int              aiErr;
   struct addrinfo *aiHead;
   struct addrinfo  hints;
   sockaddr_in6_t  *pSadrIn6;
   int              sckt;
   /*
   ** Initialize the 'hints' structure for getaddrinfo(3).
   **
   ** Notice that the 'ai_family' field is set to PF_UNSPEC, indicating to
   ** return both IPv4 and IPv6 address records for the host/service.  Most of
   ** the time, the user isn't going to care whether an IPv4 connection or an
   ** IPv6 connection is established; the user simply wants to exchange data
   ** with the remote host and doesn't care how it's done.  Sometimes, however,
   ** the user might want to explicitly specify the type of underlying socket.
   ** It is left as an exercise for the motivated reader to add a command line
   ** option allowing the user to specify the IP protocol, and then process the
   ** list of addresses accordingly (it's not that difficult).
   */
   memset( &hints, 0, sizeof( hints ) );
   hints.ai_family   = PF_UNSPEC;     /* IPv4 or IPv6 records (don't care). */
   hints.ai_socktype = SOCK_DGRAM;    /* Connectionless communication.      */
   hints.ai_protocol = IPPROTO_UDP;   /* UDP transport layer protocol only. */
   /*
   ** Look up the host/service information.
   */
   if ( ( aiErr = getaddrinfo( host,
                               service,
                               &hints,
                               &aiHead ) ) != 0 )
   {
      fprintf( stderr,
               "%s (line %d): ERROR - %s.\n",
               pgmName,
               __LINE__,
               gai_strerror( aiErr ) );
      return INVALID_DESC;
   }
   /*
   ** Go through the list and try to open a connection.  Continue until either
   ** a connection is established or the entire list is exhausted.
   */
```

```
for ( ai = aiHead,   sckt = INVALID_DESC;
      ( ai != NULL ) && ( sckt == INVALID_DESC );
      ai = ai->ai_next )
{
   /*
   ** IPv6 kluge.  Make sure the scope ID is set.
   */
   if ( ai->ai_family == PF_INET6 )
   {
      pSadrIn6 = (sockaddr_in6_t*) ai->ai_addr;
      if ( pSadrIn6->sin6_scope_id == 0 )
      {
         pSadrIn6->sin6_scope_id = scopeId;
      }  /* End IF the scope ID wasn't set. */
   }  /* End IPv6 kluge. */
   /*
   ** Display the address info for the remote host.
   */
   if ( verbose )
   {
      /*
      ** Temporary character string buffers for host & service.
      */
      char hostBfr[ NI_MAXHOST ];
      char servBfr[ NI_MAXSERV ];
      /*
      ** Display the address information just fetched.  Start with the
      ** common (protocol-independent) stuff first.
      */
      fprintf( stderr,
               "Address info:\n"
               "   ai_flags     = 0x%02X\n"
               "   ai_family    = %d (PF_INET = %d, PF_INET6 = %d)\n"
               "   ai_socktype  = %d (SOCK_STREAM = %d, SOCK_DGRAM = %d)\n"
               "   ai_protocol  = %d (IPPROTO_TCP = %d, IPPROTO_UDP = %d)\n"
               "   ai_addrlen   = %d (sockaddr_in = %d, "
               "sockaddr_in6 = %d)\n",
               ai->ai_flags,
               ai->ai_family,
               PF_INET,
               PF_INET6,
               ai->ai_socktype,
               SOCK_STREAM,
               SOCK_DGRAM,
               ai->ai_protocol,
               IPPROTO_TCP,
               IPPROTO_UDP,
               ai->ai_addrlen,
               sizeof( struct sockaddr_in ),
               sizeof( struct sockaddr_in6 ) );
      /*
      ** Display the protocol-specific formatted address.
      */
      getnameinfo( ai->ai_addr,
                   ai->ai_addrlen,
                   hostBfr,
                   sizeof( hostBfr ),
```

```
                servBfr,
                sizeof( servBfr ),
                NI_NUMERICHOST | NI_NUMERICSERV );
   switch ( ai->ai_family )
   {
      case PF_INET:   /* IPv4 address record. */
      {
         sockaddr_in_t *pSadrIn = (sockaddr_in_t*) ai->ai_addr;
         fprintf( stderr,
                  "   ai_addr      = sin_family: %d (AF_INET = %d, "
                  "AF_INET6 = %d)\n"
                  "                  sin_addr:   %s\n"
                  "                  sin_port:   %s\n",
                  pSadrIn->sin_family,
                  AF_INET,
                  AF_INET6,
                  hostBfr,
                  servBfr );
         break;
      }  /* End CASE of IPv4 record. */
      case PF_INET6:   /* IPv6 address record. */
      {
         pSadrIn6 = (sockaddr_in6_t*) ai->ai_addr;
         fprintf( stderr,
                  "   ai_addr      = sin6_family:   %d (AF_INET = %d, "
                  "AF_INET6 = %d)\n"
                  "                  sin6_addr:     %s\n"
                  "                  sin6_port:     %s\n"
                  "                  sin6_flowinfo: %d\n"
                  "                  sin6_scope_id: %d\n",
                  pSadrIn6->sin6_family,
                  AF_INET,
                  AF_INET6,
                  hostBfr,
                  servBfr,
                  pSadrIn6->sin6_flowinfo,
                  pSadrIn6->sin6_scope_id );
         break;
      }  /* End CASE of IPv6 record. */
      default:   /* Can never get here, but just for completeness. */
      {
         fprintf( stderr,
                  "%s (line %d): ERROR - Unknown protocol family (%d).\n",
                  pgmName,
                  __LINE__,
                  ai->ai_family );
         break;
      }  /* End DEFAULT case (unknown protocol family). */
   }  /* End SWITCH on protocol family. */
}  /* End IF verbose mode. */
/*
** Create a socket.
*/
if ( !SYSCALL( "socket",
               __LINE__,
               sckt = socket( ai->ai_family,
                              ai->ai_socktype,
```

```
                                    ai->ai_protocol ) ) )
      {
         sckt = INVALID_DESC;
         continue;   /* Try the next address record in the list. */
      }
      /*
      ** Set the target destination for the remote host on this socket.  That
      ** is, this socket only communicates with the specified host.
      */
      if ( !SYSCALL( "connect",
                     __LINE__,
                     connect( sckt,
                              ai->ai_addr,
                              ai->ai_addrlen ) ) )
      {
         (void) close( sckt );   /* Could use SYSCALL() again here, but why? */
         sckt = INVALID_DESC;
         continue;   /* Try the next address record in the list. */
      }
   }  /* End FOR each address record returned by getaddrinfo(3). */
   /*
   ** Clean up & return.
   */
   freeaddrinfo( aiHead );
   return sckt;
}  /* End openSckt() */
/*****************************************************************************
* Function: tod
*
* Description:
*    Receive the time-of-day from the remote server and write it to stdout.
*
* Parameters:
*    sckt - The socket descriptor for the connection.
*
* Return Value: None.
*****************************************************************************/
static void tod( int sckt )
{
   char bfr[ MAXBFRSIZE+1 ];
   int  inBytes;
   /*
   ** Send a datagram to the server to wake it up.  The content isn't
   ** important, but something must be sent to let it know we want the TOD.
   */
   if ( !SYSCALL( "write",
                  __LINE__,
                  write( sckt, "Are you there?", 14 ) ) )
   {
      return;
   }
   /*
   ** Read the time-of-day from the remote host.
   */
   if ( !SYSCALL( "read",
                  __LINE__,
                  inBytes = read( sckt,
```

```
                                    bfr,
                                    MAXBFRSIZE ) ) )
   {
      return;
   }
   bfr[ inBytes ] = '\0';    /* Null-terminate the received string. */
   fputs( bfr, stdout );     /* Null string if EOF (inBytes == 0).  */
   fflush( stdout );
}  /* End tod() */
```

23.2. Other programming languages

23.2.1. JAVA

Sun Java versions since 1.4 are IPv6 enabled, see e.g. Inet6Address (1.5/5.0) (http://java.sun.com/j2se/1.5.0/docs/api/java/net/Inet6Address.html) class. Hints are available in the *Networking IPv6 User Guide for JDK/JRE* 1.4 (http://java.sun.com/j2se/1.4.2/docs/guide/net/ipv6_guide/index.html) and 1.5 (5.0) (http://java.sun.com/j2se/1.5.0/docs/guide/net/ipv6_guide/index.html).

23.2.2. Perl

As of May 2007 it's not known that the Perl core itself already supports IPv6. It can be added by using following modules:

- Socket6 (http://search.cpan.org/~umemoto/Socket6/)

Anyway, some other modules exist for/with IPv6 support (e.g. Net::IP), search for "IPv6" on http://search.cpan.org/.

Chapter 24. Interoperability

The TAHI Project (http://www.tahi.org/) checks the interoperability of different operating systems regarding the implementation of IPv6 features. Linux kernel already got the IPv6 Ready Logo Phase 1 (http://www.linux-ipv6.org/v6ready/).

Chapter 25. Further information and URLs

25.1. Paper printed books, articles, online reviews (mixed)

25.1.1. Printed Books (English)

25.1.1.1. Cisco

- Cisco Self-Study: Implementing IPv6 Networks (IPV6) by Regis Desmeules. Cisco Press; ISBN 1587050862; 500 pages; 1st edition (April 11, 2003). Note: This item will be published on April 11, 2003.
- Configuring IPv6 with Cisco IOS by Sam Brown, Sam Browne, Neal Chen, Robbie Harrell, Edgar, Jr. Parenti (Editor), Eric Knipp (Editor), Paul Fong (Editor)362 pages; Syngress Media Inc; ISBN 1928994849; (July 12, 2002).

25.1.1.2. General

- IPv6 in Practice: A Unixer's Guide to the Next Generation Internet (http://www.benedikt-stockebrand.de/books_e.html#ipv6-in-practice) von Benedikt Stockebrand, November 2006; ISBN 3-540-24524-3
- IPv6 Essentials (http://www.sunny.ch/publications/f_ipv6.htm) by Silvia Hagen, 2nd Edition, May 2006; ISBN 0-5961-0058-2 ToC, Index, Sample Chapter etc. (http://www.oreilly.com/catalog/ipv6ess/); O'Reilly Pressrelease (http://press.oreilly.com/ipv6ess.html)
- IPv6: The New Internet Protocol. By Christian Huitema; Published by Prentice-Hall; ISBN 0138505055. Description: This book, written by Christian Huitema - a member of the InternetArchitecture Board, gives an excellent description of IPv6, how it differs from IPv4, and the hows and whys of it's development. Source: http://www.cs.uu.nl/wais/html/na-dir/internet/tcp-ip/resource-list.html
- IPv6 Networks (http://www.epinions.com/book_mu-3402412/display_~full_specs) by Niles, Kitty; (ISBN 0070248079); 550 pages; Date Published 05/01/1998.
- Implementing IPV6. Supporting the Next Generation Internet Protocols by P. E. Miller, Mark A. Miller; Publisher: John Wiley & Sons; ISBN 0764545892; 2nd edition (March 15, 2000); 402 pages.
- Big Book of Ipv6 Addressing Rfcs by Peter H. Salus (Compiler), Morgan Kaufmann Publishers, April 2000, 450 pages ISBN 0126167702.
- Understanding IPV6 (http://www.epinions.com/book_mu-3922588/display_~full_specs) by Davies, Joseph; ISBN 0735612455; Date Published 05/01/2001; Number of Pages: 350.
- Migrating to IPv6 - IPv6 in Practice by Marc Blanchet Publisher: John Wiley & Sons; ISBN 0471498920; 1st edition (November 2002); 368 pages.
- Ipv6 Network Programming by Jun-ichiro Hagino; ISBN 1555583180
- Wireless boosting IPv6 (http://www.nwfusion.com/news/2000/1023ipv6.html) by Carolyn Duffy Marsan, 10/23/2000.

- O'reilly Network search for keyword IPv6 (http://www.oreillynet.com/search/index.ncsp?sp-q=IPv6) results in 29 hits (28. January 2002)

25.1.2. Articles, eBooks, Online Reviews (mixed)

- Getting Connected with 6to4 (http://www.onlamp.com/pub/a/onlamp/2001/06/01/ipv6_tutorial.html) by Huber Feyrer, 06/01/2001
- Transient Addressing for Related Processes: Improved Firewalling by Using IPv6 and Multiple Addresses per Host; written by Peter M. Gleiz, Steven M. Bellovin (PC-PDF-Version (http://www.securiteinfo.com/ebooks/pdf/tarp.pdf); Palm-PDF-Version (http://www.securiteinfo.com/ebooks/palm/tarp.pdf); PDB-Version (http://www.securiteinfo.com/ebooks/pdb/tarp.pdb))
- Internetworking IPv6 with Cisco Routers (http://www.ip6.com/index.html) by Silvano Gai, McGrawHill Italia, 1997. The 13 chapters and appendix A-D are downloadable as PDF-documents.
- Migration and Co-existence of IPv4 and IPv6 in Residential Networks (http://www.csc.fi/~psavola/residential.html) by Pekka Savola, CSC/FUNET, 2002

25.1.3. Science Publications (abstracts, bibliographies, online resources)

See also: liinwww.ira.uka.de/ipv6 (http://liinwww.ira.uka.de/mpsbib?query=ipv6&maxnum=200) or Google / Scholar / IPv6 (http://www.google.com/scholar?q=ipv6)

- GEANT IPv6 Workplan (http://www.ipv6.ac.uk/gtpv6/workplan.html)
- IPv6 Trials on UK Academic Networks: Bermuda Project Aug.2002 (http://www.ipv6.ac.uk/bermuda2/): Participants - Getting connected - Project deliverables - Network topology - Address assignments - Wireless IPv6 access - IPv6 migration - Project presentations - Internet 2 - Other IPv6 projects - IPv6 fora and standards Bermuda 2...
- http://www.ipv6.ac.uk/
- IPv6 at the University of Southampton (http://www.ipv6.ecs.soton.ac.uk/)
- Microsoft Research IPv6 Implementation (MSRIPv6): MSRIPv6 Configuring 6to4 - Connectivity with MSR IPv6 - Our 6Bone Node... (http://www.research.microsoft.com/msripv6/)

25.1.4. Others

See following URL for more: SWITCH IPv6 Pilot / References (http://www.switch.ch/lan/ipv6/references.html)

25.2. Conferences, Meetings, Summits

Something missing? Suggestions are welcome!

25.2.1. 2004

- 1st Global IPv6 Summit in Sao Paul, Brazil

25.3. Online information

25.3.1. Join the IPv6 backbone

More to be filled later...suggestions are welcome!

25.3.1.1. Global registries

See regional registries.

25.3.1.2. Major regional registries

- America: ARIN (http://www.arin.net/), ARIN / registration page (http://www.arin.net/registration/ipv6/index.html), ARIN / IPv6 guidelines (http://www.arin.net/registration/ipv6/index.html)
- EMEA: Ripe NCC (http://www.ripe.net/), Ripe NCC / registration page (http://www.ripe.net/ripencc/mem-services/registration/), Ripe NCC / IPv6 registration (http://www.ripe.net/ripencc/mem-services/registration/ipv6/)
- Asia/Pacific: APNIC (http://www.apnic.net/), APNIC / IPv6 ressource guide (http://www.apnic.net/services/ipv6_guide.html)
- Latin America and Caribbea: LACNIC (http://lacnic.org/), IPv6 Registration Services (http://lacnic.net/en/bt-IPv6.html), IPv6 Allocation Policy (http://lacnic.net/en/chapter-4-en.pdf)
- Africa: AfriNIC (http://www.afrinic.org/)

Also a list of major (prefix length 32) allocations per local registry is available here: Ripe NCC / IPv6 allocations (http://www.ripe.net/ripencc/mem-services/registration/ipv6/ipv6allocs.html).

25.3.1.3. Tunnel brokers

Note: A list of available Tunnel broker can be found in the section Tunnel broker below.

- Former IPng. Tunnelbroker and IPv6 resources, now migrated to the SixXs System (http://www.sixxs.net/main/).
- Eckes' IPv6-with-Linux (http://sites.inka.de/lina/linux/ipv6.html) Page.
- tunnelc - a perl based tunnel client script: freshmeat.net: Project details for tunnel client (http://freshmeat.net/projects/tunnelc) SourceForge: Project Info - tunnelc (http://sourceforge.net/projects/tunnelc) (also here (http://tunnelc.sourceforge.net/))

- Linux Advanced Routing & Traffic Control HOWTO, Chapter 6: IPv6 tunneling with Cisco and/or 6bone (http://howtos.linuxbroker.com/howtoreader.shtml?file=Adv-Routing-HOWTO.html#LARTC.TUNNEL-IPV6.ADDRESSING).

25.3.1.4. 6to4

- NSayer's 6to4 information (http://www.kfu.com/~nsayer/6to4/)
- RFC 3068 / An Anycast Prefix for 6to4 Relay Routers (http://www.faqs.org/rfcs/rfc3068.html)

25.3.1.5. ISATAP

- ISATAP (Intra-Site Automatic Tunnel Access Protocol) Information (http://www.join.uni-muenster.de/Dokumente/Howtos/Howto_ISATAP.php?lang=en) by JOIN (http://www.join.uni-muenster.de/)

25.3.2. Latest news and URLs to other documents

- Lot of URLs to others documents (http://www.estoile.com/links/ipv6) by Anil Edathara
- go6 - The IPv6 Portal (http://www.go6.net/): an IPv6 online portal with a wiki-based IPv6 knowledge center, an IPv6 discussion forum, an up-to-date collection of IPv6 Events and News, free IPv6 access and services, IPv6 software applications, and much more

25.3.3. Protocol references

25.3.3.1. IPv6-related Request For Comments (RFCs)

Publishing the list of IPv6-related RFCs is beyond the scope of this document, but given URLs will lead you to such lists:

- List sorted by IPng Standardization Status (http://playground.sun.com/pub/ipng/html/specs/standards.html) or IPng Current Specifications (http://playground.sun.com/pub/ipng/html/specs/specifications.html) by Robert Hinden
- IPv6 Related Specifications (http://www.ipv6.org/specs.html) on IPv6.org

25.3.3.2. Current drafts of working groups

Current (also) IPv6-related drafts can be found here:

- IP Version 6 (ipv6) (http://www.ietf.org/ids.by.wg/ipv6.html)
- Next Generation Transition (ngtrans) (http://www.ietf.org/ids.by.wg/ngtrans.html)
- Dynamic Host Configuration (dhc) (http://www.ietf.org/ids.by.wg/dhc.html)
- Domain Name System Extension (dnsext) (http://www.ietf.org/ids.by.wg/dnsext.html)

- IPv6 Operations (v6ops) (http://www.ietf.org/ids.by.wg/v6ops.html)
- Mobile IP (mobileip) (http://www.ietf.org/ids.by.wg/mobileip.html)
- Get any information about IPv6, from overviews, through RFCs & drafts, to implementations (http://playground.sun.com/pub/ipng/html/ipng-main.html) (including availability of stacks on various platforms & source code for IPv6 stacks)

25.3.3.3. Others

- SWITCH IPv6 Pilot / References (http://www.switch.ch/lan/ipv6/references.html), big list of IPv6 references maintained by Simon Leinen

25.3.4. More information

DeepSpace6 / more interesting links (http://www.deepspace6.net/sections/links.html)

25.3.4.1. Linux related

- DeepSpace6 / (Not only) Linux IPv6 Portal (http://www.deepspace6.net/) - Italy (Mirror (http://mirrors.bieringer.de/www.deepspace6.net/))
- IPv6-HowTo for Linux by Peter Bieringer (http://www.bieringer.de/linux/IPv6/) - Germany, and his Bieringer / IPv6 - software archive (ftp://ftp.bieringer.de/pub/linux/IPv6/)
- Linux+IPv6 status by Peter Bieringer (http://www.bieringer.de/linux/IPv6/status/IPv6+Linux-status.html) - Germany (going obsolete)
- DeepSpace6 / IPv6 Status Page (http://www.deepspace6.net/docs/ipv6_status_page_apps.html) - Italy (Mirror (http://mirrors.bieringer.de/www.deepspace6.net/docs/ipv6_status_page_apps.html)) (will superseed upper one)
- USAGI project (http://www.linux-ipv6.org/) - Japan, and their USAGI project - software archive (ftp://ftp.linux-ipv6.org/pub/)
- Linux Optimized Link State Routing Protocol (OLSR) IPv6 HOWTO (http://www.tldp.org/HOWTO/OLSR-IPv6-HOWTO/)
- LinShim6 (http://inl.info.ucl.ac.be/LinShim6/)

25.3.4.2. Linux related per distribution

PLD

PLD Linux Distribution (http://www.pld-linux.org/) (“market leader” in containing IPv6 enabled packages)

Red Hat

Red Hat Enterprise Linux (http://www.redhat.com/), Pekka Savola’s IPv6 packages (http://www.netcore.fi/pekkas/linux/ipv6/)

Fedora

Fedora Core Linux (http://www.fedora.redhat.com/)

Debian

Debian Linux (http://www.debian.org/), IPv6 with Debian Linux (http://ipv6.debian.net/)

Novell/SuSE

Novell/SuSE Linux (http://www.novell.com/linux/suse/)

Mandriva

Mandriva (http://www.mandriva.com/)

For more see the IPv6+Linux Status Distributions (http://www.bieringer.de/linux/IPv6/status/IPv6+Linux-status-distributions.html) page.

25.3.4.3. General

- IPv6.org (http://www.ipv6.org/)
- 6bone (http://www.6bone.net/)
- WIDE project (http://www.v6.wide.ad.jp/) - Japan
- SWITCH IPv6 Pilot (http://www.switch.ch/lan/ipv6/) - Switzerland
- IPv6 Corner of Hubert Feyrer (http://www.feyrer.de/IPv6/) - Germany
- IPv6 Forum (http://www.ipv6forum.com/) - a world-wide consortium of leading Internet vendors, Research & Education Networks...
- Playground.sun.com / IPv6 Info Page (http://playground.sun.com/pub/ipng/html/ipng-main.html) - maintained by Robert Hinden, Nokia. Get any information about IPv6, from overviews, through RFCs & drafts, to implementations (including availability of stacks on various platforms & source code for IPv6 stacks).
- 6INIT (http://www.6init.com/) - IPv6 Internet Initiative - an EU Fifth Framework Project under the IST Programme.
- IPv6 Task Force (European Union) (http://www.ipv6-taskforce.org/)
- 6init (http://www.6init.org/) - IPv6 INternet IniTiative
- IPv6: The New Version of the Internet Protocol (http://www.usenix.org/publications/library/proceedings/ana97/summaries/deering by Steve Deering.
- IPv6: The Next Generation Internet Protocol (http://www.garykessler.net/library/ipv6_exp.html), by Gary C. Kessler.
- IPv6: Next Generation Internet Protocol (http://www.3com.com/nsc/ipv6.html) - 3Com
- internet || site (http://www.internet2.org/) and internet2 Working Group (http://ipv6.internet2.edu/)
- NetworkWorldFusion: Search / Doc Finder: searched for IPv6 (http://search.nwfusion.com/query.html?qt=IPv6&qp=&ch=cn&) (102 documents found 22.12.2002)
- The Register (http://www.theregister.co.uk/) (Search for IPv6 will result in 30 documents, 22.12.2002)
- ZDNet Search for IPv6 (http://zdnet.search.com/search?cat=279&q=IPv6)
- TechTarget Search for IPv6 (http://whatis.techtarget.com/wsearchResults/1,290214,sid9,00.html?query=IPv6)

- IPv6 & TCP Resources List (http://www.faqs.org/faqs/internet/tcp-ip/resource-list/index.html)

Something missing? Suggestions are welcome!

25.3.4.4. Market Research

- A Tale of Two Wireless Technology Trends: Processor Development Outsourcing and IPv6 (http://www.seminarinformation.com/wconnect/wc.dll?sis~details0~307~TSN)Yankee Group - 4/1/2002 - 12 Pages - ID: YANL768881
- The World Atlas of the Internet: Americas (http://www.marketresearch.com/product/display.asp?SID=88602378-241489274-186851952&ProductID=803907); IDATE - 2/1/2002 - 242 PAges - ID: IDT803907. Countries covered: Central America, North America, South America; List: Price: $ 3,500.00; excerpt: Panorama of Internet access markets across the globe. Market assessment and forecasts up to 2006 for 34 countries: market structure: main ISPs and market shares; number of subscribers, of ISPs.
- Early Interest Rising for IPv6 by IDC (Author); List Price: $1,500.00; Edition: e-book (Acrobat Reader); Publisher: IDC; ISBN B000065T8E; (March 1, 2002)

25.3.4.5. Patents

- Delphion Research: Patent Search Page (http://www.delphion.com/research/). Basic (free) registration needed. Examples found 21.12.2002 searching for IPv6: Communicating method between IPv4 terminal and IPv6 terminal and IPv4-IPv6 converting apparatus (http://www.delphion.com/details?pn=US06118784__) Translator for IP networks, network system using the translator, and IP network coupling method therefor (http://www.delphion.com/details?pn=US06038233__)

25.3.5. By countries

25.3.5.1. Europe

- www.ist-ipv6.org (http://www.ist-ipv6.org/): IST IPv6 Cluster, European IPv6 Research and Development Projects
- Euro6IX (http://www.euro6ix.org/): European IPv6 Internet Exchanges Backbone

25.3.5.2. Austria

- IPv6@IKNnet and MIPv6 Research Group (http://www.ikn.tuwien.ac.at/~ipv6/): TU Vienna, Austria (IPv6: project, publications, diploma / doctor thesis, Conference Proceedings etc.)

25.3.5.3. Australia

- Carl's Australian IPv6 Pages (http://oversteer.bl.echidna.id.au/IPv6/) (old content)

25.3.5.4. Belgium

Suggestions are welcome!

25.3.5.5. Brasil

- IPv6 do Brasil (http://www.ipv6dobrasil.com.br/)

25.3.5.6. China

Suggestions are welcome!

25.3.5.7. Czech

Suggestions are welcome!

25.3.5.8. Germany

- Xing / IPv6 (https://www.xing.com/net/ipv6/)

25.3.5.9. France

- Renater (http://www.renater.fr/Projets/IPv6/index.htm): Renater IPv6 Project Page
- IPv6 - RSVP - ATM at INRIA (http://www.inria.fr/recherche/equipes/ipv6.fr.html)
- NetBSD IPv6 Documentation (http://www.netbsd.org/fr/Documentation/network/ipv6/)

25.3.5.10. Italy

- Project6 (http://project6.ferrara.linux.it/): IPv6 networking with Linux

25.3.5.11. Japan

- Yamaha IPv6 (http://www.rtpro.yamaha.co.jp/RT/ipv6/) (sorry, all in japanese native ...)

25.3.5.12. Korea

- ETRI (http://www.krv6.net/): Electronics and Telecommunications Research Institut
- IPv6 Forum Korea (http://www.ipv6.or.kr/english/index.new.htm): Korean IPv6 Deployment Project

25.3.5.13. Mexico

- IPv6 Mexico (http://www.ipv6.unam.mx/) (spain & english version): IPv6 Project Hompeage of The National Autonomous University of Mexico (UNAM)

25.3.5.14. Netherland

- SURFnet (http://www.ipv6.surfnet.nl/): SURFnet IPv6 Backbone
- STACK (http://www.stack.nl/), STACK (IPv6) (http://www.stack.nl/ipv6/): Students' computer association of the Eindhoven University of Technology, Netherland
- IPng.nl (http://www.ipng.nl/): collaboration between WiseGuys and Intouch

25.3.5.15. Portugal

Suggestions are welcome!

25.3.5.16. Russia

- IPv6 Forum for Russia (http://www.ipv6.ru/): Yaroslavl State University Internet Center

25.3.5.17. Switzerland

Suggestions are welcome!

25.3.5.18. United Kingdom

- British Telecom IPv6 Home (http://www.bt.com/ipv6/): BT's ISP IPv6 Trial, UK's first IPv6 Internet Exchange etc.

25.3.6. By operating systems

25.3.6.1. *BSD

- KAME project (http://www.kame.net/) (*BSD)
- NetBSD's IPv6 Networking FAQ (http://www.netbsd.org/Documentation/network/ipv6/)
- FreeBSD Ports: Ipv6 (http://www.freebsd.org/ports/ipv6.html)

25.3.6.2. Cisco IOS

- Cisco IOS IPv6 Entry Page (http://www.cisco.com/warp/public/732/Tech/ipv6/)
- IPv6 for Cisco IOS Software (http://www.cisco.com/univercd/cc/td/doc/product/software/ios122/122newft/122t/122t2/ipv6/ftipv
  File 2 of 3: Aug 2002 -- Table of Contents: IPv6 for Cisco IOS Software; Configuring Documentation Specifics; Enabling IPv6 Routing and Configuring; IPv6 Addressing; Enabling IPv6 Processing Globally.
- Cisco Internet Networking Handbook, Chapter IPv6 (http://www.cisco.com/univercd/cc/td/doc/cisintwk/ito_doc/ipv6.htm)

25.3.6.3. HPUX

- comp.sys.hp.hpux FAQ (http://www.faqs.org/faqs/hp/hpux-faq/index.html)

25.3.6.4. IBM

- Now that IBM's announced the availability of z/OS V1.4, what's new in this release? (http://search390.techtarget.com/ateQuestionNResponse/0,289625,sid10_cid486367_tax292523,00.html) This question was posed on 15 August 2002

25.3.6.5. Microsoft

- Microsoft Windows 2000 IPv6 (http://www.microsoft.com/windows2000/technologies/communications/ipv6/default.asp)
- MSRIPv6 (http://www.research.microsoft.com/msripv6) - Microsoft Research Network - IPv6 Homepage
- Internet Connection Firewall Does Not Block Internet Protocol Version 6 Traffic (http://support.microsoft.com/default.aspx?scid=kb;en-us;306203) (6.11.2001)
- Internet Protocol Numbers (http://support.microsoft.com/default.aspx?scid=kb;en-us;289892) (8.10.2002)
- IPv6 Technology Preview Refresh (http://support.microsoft.com/default.aspx?scid=kb;en-us;273826) (16.10.2002)
- HOW TO: Install and Configure IP Version 6 in Windows .NET Enterprise Server (http://support.microsoft.com/default.aspx?scid=kb;en-us;325449) (26.10.2002)
- Windows .NET Server 6to4 Router Service Quits When You Advertise a 2002 Address on the Public Interface (http://support.microsoft.com/default.aspx?scid=kb;en-us;329984) (28.10.2002)
- msdn - Microsoft Windows CE .NET - IPv6 commands (http://msdn.microsoft.com/library/default.asp?url=/library/en-us/wcetcpip/htm/cmconIPv6exe.asp)

25.3.6.6. Solaris

- Sun Microsystems Solaris (http://www.sun.com/software/solaris/)
- Solaris 2 Frequently Asked Questions (FAQ) 1.73 (http://www.cs.uu.nl/wais/html/na-dir/Solaris2/FAQ.html)

25.3.6.7. Sumitoma

- Sumitomo Electric has implemented IPv6 on Suminet 3700 family routers (http://playground.sun.com/pub/ipng/html/ipng-implementations.html#Sumitomo)

25.3.6.8. ZebOS

- IpInfusion's ZebOS Server Routing Software (http://www.ipinfusion.com/products/server/products_server.html)

25.3.7. IPv6 Security

- Internet Security Systems: Security Center, X-Force Database Search (http://www.iss.net/security_center/search.php?type=3&type=3&pattern=IPv6) (21.12.2002 - 6 topics found relating to IPv6)
- NIST IPsec Project (http://csrc.nist.gov/ipsec/) (National Institute of Standards and Technology, NIST)
- Information Security (http://www.infosecuritymag.com/index.shtml)
- NewOrder.box.sk (search for IPv6) (http://neworder.box.sk/search.php3?srch=IPv6) (Articles, exploits, files database etc.)

25.3.8. Application lists

- DeepSpace6 / IPv6 Status Page (http://www.deepspace6.net/docs/ipv6_status_page_apps.html) (Mirror (http://mirrors.bieringer.de/www.deepspace6.net/docs/ipv6_status_page_apps.html))
- IPv6.org / IPv6 enabled applications (http://www.ipv6.org/v6-apps.html)
- Freshmeat / IPv6 search (http://freshmeat.net/search/?q=IPv6), currently (14 Dec 2002) 62 projects
- IPv6 Forum / Web Links (http://www.ipv6forum.com/modules.php?op=modload&name=Web_Links&file=index)

25.3.8.1. Analyzer tools

- Wireshark (http://www.wireshark.org/) (former known as *Ethereal*) is a free network protocol analyzer for Unix and Windows
- Radcom RC100-WL (http://www.ip6.com/us/analyzer.htm) - Download Radcom RC100-WL protocol analyzer version 3.20

25.3.8.2. IPv6 Products

- 6wind (http://www.6wind.com/) - solutions for IPv4/IPv6 Router, QoS, Multicast, Mobility, Security/VPN/Firewall.

- Fefe's patches for IPv6 with djbdns (http://www.fefe.de/dns/)Aug 2002 -- What is djbdns and why does it need IPv6? djbdns is a full blown DNS server which outperforms BIND in nearly all respects.
- ZebOS Server Routing Suite (http://www.ipinfusion.com/products/server/products_server.html)
- SPA Mail Server 2.21 (http://download.com.com/3000-2165-10153543.html?tag=lst-0-21)
- Inframail (Advantage Server Edition) 6.0 (http://download.com.com/3000-2165-8202652.html?tag=lst-0-2)
- HTTrack Website Copier (http://download.com.com/3000-2377-10149393.html?tag=lst-0-1)
- CommView 5.0 (http://download.com.com/3000-2085-10132748.html?tag=lst-0-1)
- Posadis 0.50.6 (http://download.com.com/3000-2104-10149750.html?tag=lst-0-1)

25.3.8.3. SNMP

- comp.protocpols.snmp SNMP FAQ Part 1 of 2 (http://www.cs.uu.nl/wais/html/na-dir/snmp-faq/part1.html)

25.4. IPv6 Infrastructure

25.4.1. Statistics

- IPv6 routing table history (http://www.space.net/~gert/RIPE/) created by Gert Döring, Space.Net (http://www.space.net/)
- Official 6bone Webserver list Statisic (http://6bone.informatik.uni-leipzig.de/ipv6/stats/stats.php3)

25.4.2. Internet Exchanges

Another list of IPv6 Internet Exchanges can be found here: IPv6 status of IXPs in Europe (http://www.euro-ix.net/isp/choosing/search/matrix.php)

25.4.2.1. Estonia

- TIX (http://tix.estpak.ee/) (tallinn interneti exchange with ipv6 support)

25.4.2.2. Europe

- Euro6IX (http://www.euro6ix.net/), European IPv6 Internet Exchange Backbone

25.4.2.3. France

- French National Internet Exchange IPv6 (http://www.fnix6.net/) (since 1.11.2002 active). FNIX6 provides a free and reliable high speed FastEthernet interconnection between ISP located in TeleCity Paris.

25.4.2.4. Germany

- INXS (http://www.inxs.de/): (Cable & Wireless) Munich and Hamburg

25.4.2.5. Japan

- NSPIXP-6 (http://www.wide.ad.jp/nspixp6/): IPv6-based Internet Exchange in Tokyo
- JPIX (http://www.jpix.co.jp/), Tokyo

25.4.2.6. Korea

- 6NGIX (http://www.ngix.ne.kr/)

25.4.2.7. Netherlands

- AMS-IX (http://www.ams-ix.net/): Amsterdam Internet Exchange

25.4.2.8. UK

- UK6X (http://www.uk6x.com/): London
- XchangePoint (http://www.xchangepoint.net/): London

25.4.2.9. USA

- 6TAP (http://www.6tap.net/): Chicago. Supports peerings around the globe.
- PAIX (http://www.paix.net/): Palo Alto

25.4.3. Tunnel broker

See also: http://www.deepspace6.net/docs/tunnelbrokers.html

25.4.3.1. Belgium

Something missing? Suggestions are welcome!

25.4.3.2. Canada

- Freenet6 (http://www.freenet6.net/) - /48 Delegation, Canada Getting IPv6 Using Freenet6 on Debian (http://www.linuxjournal.com/article.php?sid=5963&mode=thread&order=0) Freenet6 creater (http://www.viagenie.qc.ca/en/index.shtml)

25.4.3.3. China

Something missing? Suggestions are welcome!

25.4.3.4. Estonia

- Estpak (http://tunnelbroker.ipv6.estpak.ee/?tunnel&PHPSESSID=aa2184190cc2cc6d3a6f6ddd01ae3635)

25.4.3.5. Germany

- 6bone Knoten Leipzig (http://6bone.informatik.uni-leipzig.de/) Info bez. Hackangriff (2001) (http://www.mail-archive.com/ipv6@uni-muenster.de/msg00056.html)

25.4.3.6. Italy

- Comv6 (http://www.comv6.com/)
- Bersafe (http://www.bersafe.it/) (Italian language)

25.4.3.7. Japan

Something missing? Suggestions are welcome!

25.4.3.8. Malaysia

Something missing? Suggestions are welcome!

25.4.3.9. Netherlands

- IPng Netherland (http://www.ipng.nl/) - Intouch, SurfNet, AMS-IX, UUNet, Cistron, RIPE NCC and AT&T are connected at the AMS-IX. It is possible (there are requirements...) to get an static tunnel.
- SURFnet Customers (http://www.ipv6.surfnet.nl/)

25.4.3.10. Norway

- UNINETT (http://www.uninett.no/testnett/index.en.html) - Pilot IPv6 Service (for Customers): tunnelbroker & address allocation Uninett-Autoupdate-HOWTO (http://www.guruz.de/Uninett-Autoupdate-HOWTO)

25.4.3.11. Spain

- Consulintel (http://tb.consulintel.euro6ix.org/)

25.4.3.12. Switzerland

Something missing? Suggestions are welcome!

25.4.3.13. UK

- NTT (http://www.nttv6.net/), United Kingdom - IPv6 Trial. IPv4 Tunnel and native IPv6 leased Line connections. POPs are located in London, UK Dusseldorf, Germany New Jersey, USA (East Coast) Cupertino, USA (West Coast) Tokyo, Japan

25.4.3.14. USA

- ESnet (http://www.es.net/hypertext/welcome/pr/ipv6.html), USA - Energy Sciences Network: Tunnel Registry & Address Delegation for directly connected ESnet sites and ESnet collaborators.
- Hurricane Electric (http://ipv6tb.he.net/), US backbone; Hurrican Electric Tunnelbroker (http://tunnelbroker.net/) (also available under http://tunnelbroker.com/) Press Release: Hurricane Electric Upgrades IPv6 Tunnel Broker (http://www.he.net/releases/release6.html) Tunnel Broker Endpoint Autoupdate (http://ipv6.he.net/tunnelbroker-update.php), Perl Script

25.4.3.15. Singapore

Something missing? Suggestions are welcome!

25.4.3.16. More Tunnel brokers...

- Public 6to4 relay routers (http://www.kfu.com/~nsayer/6to4/) (MS IIE boycott!)

25.4.4. Native IPv6 Services

Note: These services are mostly only available with a valid IPv6 connection!

25.4.4.1. Net News (NNTP)

Something missing? Suggestions are welcome!

25.4.4.2. Game Server

- Quake2 (http://www.viagenie.qc.ca/en/ipv6/quake2/ipv6-quake2.shtml) over IPv6

25.4.4.3. IRC Server

Something missing? Suggestions are welcome!

25.4.4.4. Radio Stations, Music Streams

Something missing? Suggestions are welcome!

25.4.4.5. Webserver

- Peter Bieringer's Home of Linux IPv6 HOWTO (http://www.ipv6.bieringer.de/)

Something missing? Suggestions are welcome!

25.5. Maillists

Lists of maillists are available at:

- DeepSpace6 / Mailling Lists (http://www.deepspace6.net/sections/lists.html)

Major Mailinglists are listed in following table:

Focus	Request e-mail address	What to subscribe	Maillist e-mail address	Language	Access through WWW
Linux kernel networking including IPv6	majordomo (at) vger.kernel.org	netdev	netdev (at) vger.kernel.org	English	Info (http://vger.kernel.org/vger-lists.html#netdev), Archive (http://www.spinics.net/lists/netdev/

Mobile IP(v6) for Linux	Web-based, see URL	mipl	mipl (at) mobile-ipv6.org	English	Info (http://www.mobile-ipv6.org/cgi-bin/mailman/listinfo), Archive (http://www.mobile-ipv6.org/pipermail/mipl/)
Linux IPv6 users using USAGI extension	usagi-users-ctl (at) linux-ipv6.org		usagi-users (at) linux-ipv6.org	English	Info / Search (http://www.linux-ipv6.org/ml/index.html#usagi-users), Archive (http://www.linux-ipv6.org/ml/usagi-users/)
IPv6 on Debian Linux			debian-ipv6 (at) lists.debian.org	English	Info/Subscription/Archive (http://lists.debian.org/debian-ipv6/)
6bone	majordomo (at) isi.edu	6bone	6bone (at) isi.edu	English	Info (http://www.6bone.net/6bone_emai Archive (http://mailman.isi.edu/pipermail/6b
IPv6 users in general	majordomo (at) ipv6.org	users	users (at) ipv6.org	English	Info (http://www.ipv6.org/mailing-lists.html), Archive (http://www.mail-archive.com/users@ipv6.org/)
Bugtracking of Internet applications (1)	bugtraq-subscribe (at) securityfo-cus.com		bugtraq (at) se-curityfocus.com (2)	English	Info (http://online.securityfocus.com/po Archive (http://online.securityfocus.com/arc

(1) very recommended if you provide server applications.

(2) list is moderated.

Something missing? Suggestions are welcome!

Following other maillinglists & newsgroups are available via web:

- student-ipv6 (India) (http://groups.yahoo.com/group/student-ipv6) Description: This is the group for the Student Awareness group of IPv6 in India
- sun-ipv6-users (http://groups.yahoo.com/group/sun-ipv6-users) Description: Please report problems/suggestions regarding SUN Microsystems IPng implementation

- IPv6-BITS (http://groups.yahoo.com/group/IPv6-BITS) Description: This List will co-ordinate the working of Project Vertebrae.
- linux-bangalore-ipv6 (http://groups.yahoo.com/group/linux-bangalore-ipv6) Description: The IPv6 deployment list of the Bangalore Linux User Group
- packet-switching (http://groups.yahoo.com/group/packet-switching) Description: This mailing list provides a forum for discussion of packet switching theory, technology, implementation and application in any relevant aspect including without limitation LAPB, X.25, SDLC, P802.1d, LLC, IP, IPv6, IPX, DECNET, APPLETALK, FR, PPP, IP Telephony, LAN PBX systems, management protocols like SNMP, e-mail, network transparent window systems, protocol implementation, protocol verification, conformance testing and tools used in maintaining or developing packet switching systems.
- de.comm.protocols.tcp-ip Description: Umstellung auf IPv6 Source: Chartas der Newsgruppen in de.* (http://www.faqs.org/faqs/de-newsgroups/chartas/index.html)
- Google Group: comp.protocols.tcp-ip (http://groups.google.com/groups?hl=en&lr=&ie=UTF-8&oe=UTF8&safe=off&group=comp.protocols.tcp-ip)
- Google Group: linux.debian.maint.ipv6 (http://groups.google.com/groups?hl=en&lr=&ie=UTF-8&oe=UTF8&safe=off&group=linux.debian.maint.ipv6)
- Google Group: microsoft.public.platformsdk.networking.ipv6 (http://groups.google.com/groups?hl=en&lr=&ie=UTF-8&oe=UTF8&safe=off&group=microsoft.public.platformsdk.networking.ipv6)
- Google Group: fa.openbsd.ipv6 (http://groups.google.com/groups?hl=en&lr=&ie=UTF-8&oe=UTF8&safe=off&group=fa.openbsd.ipv6)

25.6. Online tools

25.6.1. Testing tools

- ping, traceroute, tracepath, 6bone registry, DNS: JOIN / Testtools (http://www.join.uni-muenster.de/lab/testtools.html) (German language only, but should be no problem for non German speakers)
- traceroute6, whois: IPng.nl (http://www.ipng.nl/)
- AAAA Lookup Checker http://www.cnri.dit.ie/cgi-bin/check_aaaa.pl

25.6.2. Information retrievement

- List of worldwide all IPv6-aggregated IP-Blocks (http://www.ripe.net/ripencc/mem-services/registration/ipv6/ipv6allocs.html)

25.6.3. IPv6 Looking Glasses

- DRENv6 Looking Glass (http://www.v6.dren.net/lg/)

25.6.4. Helper applications

- IPv6 Prefix Calculator (http://www.tdoi.org/prefcalc.php) by TDOI (http://www.tdoi.org/)
- DNS record checker (http://www.maths.tcd.ie/cgi-bin/check_dns.pl)

25.7. Trainings, Seminars

- CIW Internetworking Professional Training CBT CD (http://www.e-trainonline.com/html/ciw_internetworking_profession.html#IPv6)
- Training Pages (http://www.trainingpages.com/x/category,kw-1628,.html), U.K. - Search for IPv6 (13 Courses, 2006-08-21)
- Erion IPv6 Training (http://www.erion.co.uk/ipv6.html), UK

Something missing? Suggestions are welcome!

25.8. 'The Online Discovery' ...

IPv6: Addressing The Needs Of the Future by Yankee Group (Author) List Price: $595.00 Edition: e-book (Acrobat Reader) Pages: 3 (three) Publisher: MarketResearch.com; ISBN B00006334Y; (November 1, 2001)

;-) The number of copies would be interesting...

Chapter 26. Revision history / Credits / The End

26.1. Revision history

Versions x.y are published on the Internet.

Versions x.y.z are work-in-progress and published as LyX and SGML file on CVS. Because Deep Space 6 mirrors these SGML files and generate independend from TLDP public versions, this versions will show up there and also on its mirrors.

26.1.1. Releases 0.x

0.65

2010-04-20/PB: extend QoS section with examples

0.65

2009-12-13/PB: minor fixes

0.64

2009-06-11/PB: extend DHCP server examples (ISC DHCP, Dibbler)

0.63

2009-02-14/PB: Fix FSF address, major update on 4in6 tunnels, add new section for address resolving, add some URLs, remove broken URLs

0.62

2008-11-09/PB: Adjust URL to Turkish howto, add some HIP related URLs, remove broken URLs

0.61.1

2007-11-11/PB: fix broken description of shortcut BIND

0.61

2007-10-06/PB: fix broken URLs to TLDP-CVS, minor URL update.

0.60.2

2007-10-03/PB: fix description of sysctl/autoconf (credits to Francois-Xavier Le Bail)

0.60.1

2007-06-16/PB: speling fixes (credits to Larry W. Burton)

0.60

2007-05-29/PB: import major contribution to Programming using C-API written by John Wenker, minor fixes

0.52

2007-05-23/PB: update firewalling chapter, improve document for proper SGML validation, minor bugfixes

0.51

2006-11-08/PB: remove broken URLs, add a new book (credits to Bryan Vukich)

0.50.2

2006-10-25/PB: fix typo in dhcp6 section (credits to Michele Ferritto)

0.50.1

2006-09-23/PB: add some URLs

0.50

2006-08-24/PB: check RFC URLs, fix URL to Chinese translation, finalize for publishing

0.49.5

2006-08-23/PB: fix/remove broken URLs

0.49.4

2006-08-21/PB: some review, update and enhancement of the content, replace old 6bone example addresses with the current defined ones.

0.49.3

2006-08-20/PB: fix bug in maillist entries, 'mobility' is now a separate chapter

0.49.2

2006-08-20/PB: update and cleanup of maillist entries

0.49.1

2006-06-13/PB: major update of mobility section (contributed by Benjamin Thery)

0.49

2005-10-03/PB: add configuration hints for DHCPv6, major broken URL cleanup (credits to Necdet Yucel)

0.48.1

2005-01-15/PB: minor fixes

0.48

2005-01-11/PB: grammar check and minor review of IPv6 IPsec section

0.47.1

2005-01-01/PB: add information and examples about IPv6 IPsec, add some URLs

0.47

2004-08-30/PB: add some notes about proftpd, vsftpd and other daemons, add some URLs, minor fixes, update status of Spanish translation

0.46.4

2004-07-19/PB: minor fixes

0.46.3

2004-06-23/PB: add note about started Greek translation, replace Taiwanese with Chinese for related translation

0.46.2

2004-05-22/PB: minor fixes

0.46.1

2004-04-18/PB: minor fixes

0.46

2004-03-04/PB: announce Italian translation, add information about DHCPv6, minor updates

0.45.1

2004-01-12/PB: add note about the official example address space

0.45

2004-01-11/PB: minor fixes, add/fix some URLs, some extensions

0.44.2

2003-10-30/PB: fix some copy&paste text bugs

0.44.1

2003-10-19/PB: add note about start of Italian translation

0.44

2003-08-15/PB: fix URLs, add hint on tcp_wrappers (about broken notation in some versions) and Apache2

0.43.4

2003-07-26/PB: fix URL, add archive URL for maillist users at ipv6.org, add some ds6 URLs

0.43.3

2003-06-19/PB: fix typos

0.43.2

2003-06-11/PB: fix URL

0.43.1

2003-06-07/PB: fix some URLs, fix credits, add some notes at IPsec

0.43

2003-06-05/PB: add some notes about configuration in SuSE Linux, add URL of French translation

0.42

2003-05-09/PB: minor fixes, announce French translation

0.41.4

2003-05-02/PB: Remove a broken URL, update some others.

0.41.3

2003-04-23/PB: Minor fixes, remove a broken URL, fix URL to Taiwanese translation

0.41.2

2003-04-13/PB: Fix some typos, add a note about a French translation is in progress

0.41.1

2003-03-31/PB: Remove a broken URL, fix another

0.41

2003-03-22/PB: Add URL of German translation

0.40.2

2003-02-27/PB: Fix a misaddressed URL

0.40.1

2003-02-12/PB: Add Debian-Linux-Configuration, add a minor note on translations

0.40

2003-02-10/PB: Announcing available German version

0.39.2

2003-02-10/GK: Minor syntax and spelling fixes

0.39.1

2003-01-09/PB: fix an URL (draft adopted to an RFC)

0.39

2003-01-13/PB: fix a bug (forgotten 'link" on "ip link set" (credits to Yaniv Kaul)

0.38.1

2003-01-09/PB: a minor fix

0.38

2003-01-06/PB: minor fixes

0.37.1

2003-01-05/PB: minor updates

0.37

2002-12-31/GK: 270 new links added (searched in 1232 SearchEngines) in existing and 53 new (sub)sections

0.36.1

2002-12-20/PB: Minor fixes

0.36

2002-12-16/PB: Check of and fix broken links (credits to Georg Käfer), some spelling fixes

0.35

2002-12-11/PB: Some fixes and extensions

0.34.1

2002-11-25/PB: Some fixes (e.g. broken linuxdoc URLs)

0.34

2002-11-19/PB: Add information about German translation (work in progress), some fixes, create a small shortcut explanation list, extend “used terms” and add two German books

0.33

2002-11-18/PB: Fix broken RFC-URLs, add parameter ttl on 6to4 tunnel setup example

0.32

2002-11-03/PB: Add information about Taiwanese translation

0.31.1

2002-10-06/PB: Add another maillist

0.31

2002-09-29/PB: Extend information in proc-filesystem entries

0.30

2002-09-27/PB: Add some maillists

0.29

2002-09-18/PB: Update statement about nmap (triggered by Fyodor)

0.28.1

2002-09-16/PB: Add note about ping6 to multicast addresses, add some labels

0.28

2002-08-17/PB: Fix broken LDP/CVS links, add info about Polish translation, add URL of the IPv6 Address Oracle

0.27

2002-08-10/PB: Some minor updates

0.26.2

2002-07-15/PB: Add information neighbor discovery, split of firewalling (got some updates) and security into extra chapters

0.26.1

2002-07-13/PB: Update nmap/IPv6 information

0.26

2002-07-13/PB: Fill /proc-filesystem chapter, update DNS information about depricated A6/DNAME, change P-t-P tunnel setup to use of “ip” only

0.25.2

2002-07-11/PB: Minor spelling fixes

0.25.1

2002-06-23/PB: Minor spelling and other fixes

0.25

2002-05-16/PB: Cosmetic fix for 2^128, thanks to José Abílio Oliveira Matos for help with LyX

0.24

2002-05-02/PB: Add entries in URL list, minor spelling fixes

0.23

2002-03-27/PB: Add entries in URL list and at maillists, add a label and minor information about IPv6 on RHL

0.22

2002-03-04/PB: Add info about 6to4 support in kernel series 2.2.x and add an entry in URL list and at maillists

0.21

2002-02-26/PB: Migrate next grammar checks submitted by John Ronan

0.20.4

2002-02-21/PB: Migrate more grammar checks submitted by John Ronan, add some additional hints at DNS section

0.20.3

2002-02-12/PB: Migrate a minor grammar check patch submitted by John Ronan

0.20.2

2002-02-05/PB: Add mipl to maillist table

0.20.1

2002-01-31/PB: Add a hint how to generate 6to4 addresses

0.20

2002-01-30/PB: Add a hint about default route problem, some minor updates

0.19.2

2002-01-29/PB: Add many new URLs

0.19.1

2002-01-27/PB: Add some forgotten URLs

0.19

2002-01-25/PB: Add two German books, fix quote entinities in exported SGML code

0.18.2

2002-01-23/PB: Add a FAQ on the program chapter

0.18.1

2002-01-23/PB: Move “the end” to the end, add USAGI to maillists

0.18

2002-01-22/PB: Fix bugs in explanation of multicast address types

0.17.2

2002-01-22/PB: Cosmetic fix double existing text in history (at 0.16), move all credits to the end of the document

0.17.1

2002-01-20/PB: Add a reference, fix URL text in online-test-tools

0.17

2002-01-19/PB: Add some forgotten information and URLs about global IPv6 addresses

0.16

2002-01-19/PB: Minor fixes, remove “bold” and “emphasize” formats on code lines, fix “too long unwrapped code lines” using selfmade utility, extend list of URLs.

0.15

2002-01-15/PB: Fix bug in addresstype/anycast, move content related credits to end of document

0.14

2002-01-14/PB: Minor review at all, new chapter “debugging”, review “addresses”, spell checking, grammar checking (from beginning to 3.4.1) by Martin Krafft, add tcpdump examples, copy firewalling/netfilter6 from IPv6+Linux-HowTo, minor enhancements

0.13

2002-01-05/PB: Add example BIND9/host, move revision history to end of document, minor extensions

0.12

2002-01-03/PB: Merge review of David Ranch

0.11

2002-01-02/PB: Spell checking and merge review of Pekka Savola

0.10

2002-01-02/PB: First public release of chapter 1

26.2. Credits

The quickest way to be added to this nice list is to send bug fixes, corrections, and/or updates to me ;-).

If you want to do a major review, you can use the native LyX file (see original source) and send diffs against it, because diffs against SGML don't help too much.

26.2.1. Major credits

- David Ranch <dranch at trinnet dot net>: For encouraging me to write this HOWTO, his editorial comments on the first few revisions, and his contributions to various IPv6 testing results on my IPv6 web site. Also for his major reviews and suggestions.
- Pekka Savola <pekkas at netcore dot fi>: For major reviews, input and suggestions.
- Martin F. Krafft <madduck at madduck dot net>: For grammar checks and general reviewing of the document.
- John Ronan <j0n at tssg dot wit dot ie>: For grammar checks.
- Georg Käfer <gkaefer at gmx dot at>: For detection of no proper PDF creation (fixed now by LDP maintainer Greg Ferguson), input for German books, big list of URLs, checking all URLs, many more suggestions, corrections and contributions, and the German translation
- Michel Boucey <mboucey at free dot fr>: Finding typos and some broken URLs, contribute some suggestions and URLs, and the French translation
- Michele Ferritto <m dot ferritto at virgilio dot it>: Finding bugs and the Italian translation
- Daniel Roesen <dr at cluenet dot de>: For grammar checks
- Benjamin Thery <benjamin dot thery at bull dot net>: For contribution of updated mobility section
- John Wenker <jjw at pt dot com>: major contribution to Programming using C-API
- Srivats P. <Srivats dot P at conexant dot com>: major contribution for 4in6 tunnels

26.2.2. Other credits

26.2.2.1. Document technique related

Writing a LDP HOWTO as a newbie (in LyX and exporting this to DocBook to conform to SGML) isn't as easy as some people say. There are some strange pitfalls... Nevertheless, thanks to:

- Authors of the LDP Author Guide (http://www.tldp.org/LDP/LDP-Author-Guide/)
- B. Guillon: For his DocBook with LyX HOWTO (http://perso.libertysurf.fr/bgu/doc/db4lyx/)

26.2.2.2. Content related credits

Credits for fixes and hints are listed here, will grow sure in the future

- S .P. Meenakshi <meena at cs dot iitm dot ernet dot in>: For a hint using a "send mail" shell program on tcp_wrapper/hosts.deny
- Frank Dinies <FrankDinies at web dot de>: For a bugfix on IPv6 address explanation
- John Freed <jfreed at linux-mag dot com>: For finding a bug in IPv6 multicast address explanation
- Craig Rodrigues <crodrigu at bbn dot com>: For suggestion about RHL IPv6 setup
- Fyodor <fyodor at insecure dot org>: Note me about outdated nmap information

- Mauro Tortonesi <mauro at deepspace6 dot net>: For some suggestions
- Tom Goodale <goodale at aei-potsdam dot mpg dot de>: For some suggestions
- Martin Luemkemann <mluemkem at techfak dot uni-bielefeld dot de>: For a suggestion
- Jean-Marc V. Liotier <jim at jipo dot com>: Finding a bug
- Yaniv Kaul <ykaul at checkpoint dot com>: Finding a bug
- Arnout Engelen <arnouten at bzzt dot net>: For sending note about a draft was adopted to RFC now
- Stephane Bortzmeyer <bortzmeyer at nic dot fr>: Contributing persistent configuration on Debian
- lithis von saturnsys <lithis at saturnsys dot com>: Reporting a misaddressed URL
- Guy Hulbert <gwhulbert at rogers dot com>: Send a note that RFC1924 is probably an April fool's joke
- Tero Pelander <tpeland at tkukoulu dot fi>: Reporting a broken URL
- Walter Jontofsohn <wjontof at gmx dot de>: Hints for SuSE Linux 8.0/8.1
- Benjamin Hofstetter <benjamin dot hofstetter at netlabs dot org>: Reporting a mispointing URL
- J.P. Larocque <piranha at ely dot ath dot cx>: Reporting archive URL for maillist users at ipv6 dot org
- Jorrit Kronjee <jorrit at wafel dot org>: Reporting broken URLs
- Colm MacCarthaigh <colm dot maccarthaigh at heanet dot ie>: Hint for sendfile issue on Apache2
- Tiago Camilo <tandre at ipg dot pt>: Contribute some URLs about Mobile IPv6
- Harald Geiger: Reporting a bug in how described the bit counting of the universal/global bit
- Bjoern Jacke <bjoern at j3e dot de>: Triggered me to fix some outdated information on xinetd
- Christoph Egger <cegger at chrrr dot com>: Sending note about "ip" has problems with IPv4-compatible addresses on SuSE Linux 9.0 and trigger to add a hint on 6to4-radvd example
- David Lee Haw Ling <hawling at singnet dot com dot sg>: Sending information about a tunnel broker
- Michael H. Warfield <mhw at iss dot net>: Sending note about suffix for 6to4 routers
- Tomasz Mrugalski <thomson at klub dot com dot pl>: Sending updates for DHCPv6 section
- Jan Minar <jjminar at fastmail dot fm>: Reporting minor bugs
- Kalin KOZHUHAROV <kalin at tar dot bz>: Fixing a not so well explanation
- Roel van Dijk <rdvdijk at planet dot nl>: Reporting broken URLs
- Catalin Muresan <catalin dot muresan at astral dot ro>: Reporting minor bugs
- Dennis van Dok <dvandok at quicknet dot nl>: Reporting minor bugs
- Necdet Yucel <nyucel at comu dot edu dot tr>: Reporting broken URLs
- Bryan Vukich: Reporting a broken URL
- Daniele Masini: reporting a broken iptables example
- Yao Zhao: reporting a bug in IPv6 route remove description
- Aaron Kunde: reporting a broken URL and a content related bug
- Larry W. Burton: speling fixes
- Justin Pryzby: reporting broken shortcut description of BIND

26.3. The End

Thanks for reading. Hope it helps!

If you have any questions, subscribe to proper maillist and describe your problem providing as much as information as possible.