Migrando aplicaciones a IPv6

Fundamentos técnico: DualStack y Socket.h

Alonso Sandoval A. Hernán Vargas L.

Universidad Técnica Federico Santa María

asandova@alumnos.inf.utfsm.cl, hvargas@alumnos.inf.utfsm.cl

3 de octubre de 2014

Índice

Índice

Índice

Paquetes IPv4 bajo IPv6

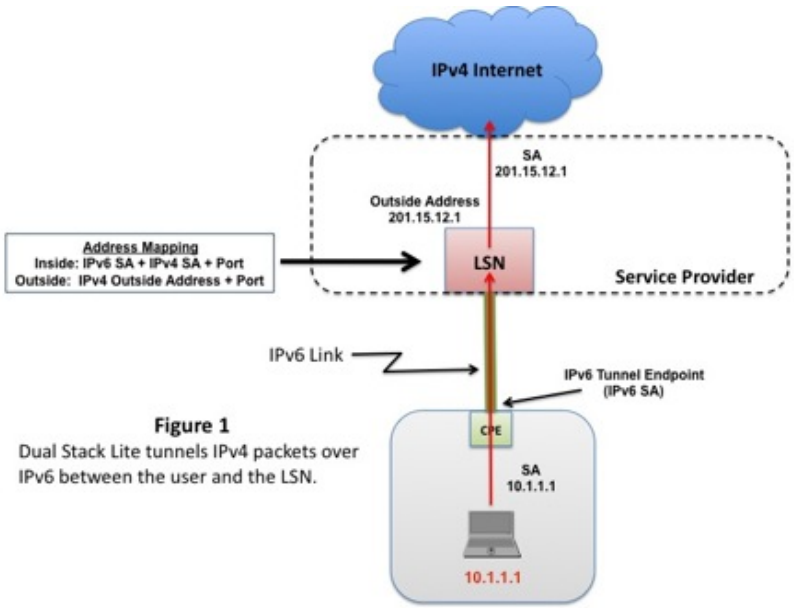

Figure 1
Dual Stack Lite tunnels IPv4 packets over IPv6 between the user and the LSN.

- Cuando el cliente envía mensajes IPv4 se encapsulan en paquetes IPv6.
- En el LSN(Large Scale Nat) el paquete de des-encapsula y NAT44 (Network address translate for IPv4) actúa a continuación.
- La funcionalidad se implementa en el LSN.
- Para identificar cada equipo se realiza un mapeo que se guarda en una tabla con la siguiente combinación: IPv6 address + IPv4 address + Port

Paquetes IPv4 bajo IPv6 - Generalización

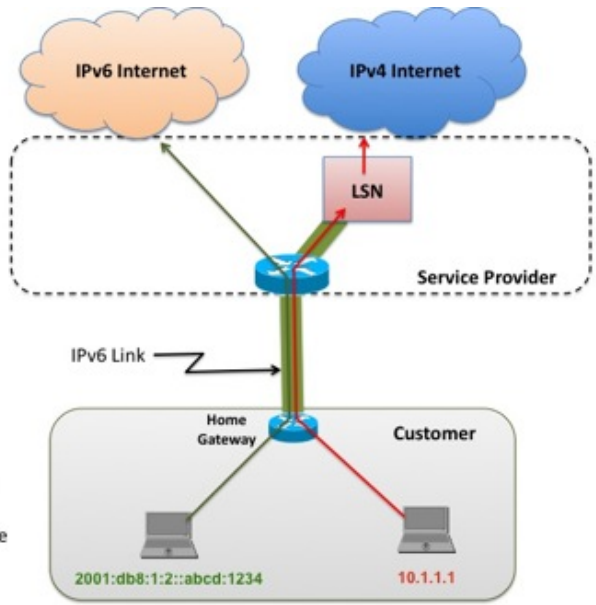

Figure 2
IPv6 packets are routed normally, while IPv4 packets are routed to the LSN.

- Un mensaje IPv6 se envía normalmente.
- Si el mensaje va en IPv4 se utiliza la técnica mencionada anteriormente. Es decir, el paquete IPv4 encapsulado en uno IPv6 y enviado a un LSN disponible.

Problemas

- La funcionalidad que permite el paso de mensajes en IPv4 a través de IPv6 debe ser implementada en los equipos locales.
- En general las ISP son reacias a molestar a los usuarios, por lo que la estrategia consistiría en implementar el método en los nuevos clientes y a medida que se renueven los equipos.

Esquema ideal

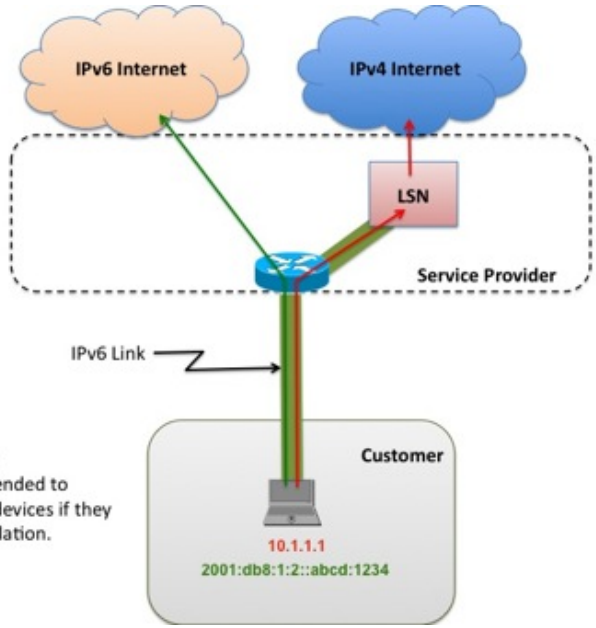

Figure 3
The IPv6 link can be extended to individual dual stacked devices if they support DS Lite encapsulation.

- Dual Stack implementado en los equipos.
- Los equipos son capaces de interactuar con ambos protocolos.

Índice

Índice

Recuerdo.

- Provee funciones básicas para el traspaso de paquetes vía socket.
- Soporta los protocolos de transporte `UDP` y `TCP`.
- Soporta los protocolos de red IPv4 y IPv6.
- Nos enfocaremos en el traspaso de mensajes `TCP/IP` tanto versión 4 como 6.

Índice

Creación del socket

```
#include <sys/socket.h>
/* int = socket(int domain, int type, int protocol); */
int mi_socket;
mi_socket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
```

- `domain`: dominios de transmisión, puede ser `AF_INET` en el caso de IPv4, `AF_UNIX` para comunicación interna, etc.
- `type`: Tipo de conexión, `SOCK_STREAM` para `TCP` o `SOCK_DGRAM` para `UDP`.
- `protocol`: Protocolo utilizado. Si el tipo (`type`) solo tiene un protocolo se puede poner un 0 (ej: `TCP`).

domain	**Descripción**
AF_UNIX	Comunicación interna Unix.
AF_INET	Comunicación IPv4.
AF_INET6	Comunicación IPv6.
AF_APPLETALK	Comunicación para AppleTalk.

type	**Descripción**
SOCK_STREAM	Socket de flujos, orientado a la conexión.
SOCK_DGRAM	Socket de datagramas, envía los paquetes individualmente.
SOCK_SEQPACKET	Envia los paquetes secuencialmente.
SOCK_RDM	Socket de datagramas, confiable pero no ordenado.

Una vez elegido el tipo de socket que queremos utilizar debemos ligarlo a nuestra dirección y puerto. Las estructuras de IPv4 y IPv6 son diferentes.

Comparación es estructuras.

Estructura IPv4:

```
struct in_addr {
    unsigned long s_addr;  // Dirección IPv4
};

struct sockaddr_in {
    short            sin_family;   // Dominio AF_INET.
    unsigned short   sin_port;     // Puerto
    struct in_addr   sin_addr;     // Dirección IPv4.
    char             sin_zero[8];  // Ceros.
};
```

Comparación es estructuras.

Estructura IPv6:

```
struct in6_addr {
    u_int8_t  s6_addr[16];  // Dirección IPv6
}

struct sockaddr_in6 {
    u_char           sin6_len;      // Largo de esta estructura.
    u_char           sin6_family;   // Dominio AF_INET6.
    u_int16m_t       sin6_port;     // Puerto.
    u_int32m_t       sin6_flowinfo; // Cero.
    struct in6_addr  sin6_addr;     // Dirección IPv6.
};
```

Convenciones.

En las estructuras vistas, los campos para manejar el puerto y la dirección utilizan un ordenamiento de bytes especial (formato network) para guardar la información, por ello se hacen necesarias funciones de conversión.

```
// Para la dirección IP:
int inet_pton(int af, const char *cp, void *buf);
/* af: AF_INET o AF_INET6,
 * cp: dirección IPv4 o IPv6,
 * buf: buffer para el resultado. */

// Para el puerto:
uint16_t htons(uint16_t hostshort);
// hostshort: Puerto.
```

Una vez configurado el socket y la estructura de dirección, debemos crear la relación entre ellos:

```
int bind(int fd, struct sockaddr *addr, int addrlen);
/* fd: Descriptor del socket,
 * addr: Estructura de dirección,
 * addrlen: Largo de la estructura de dirección. */
```

Con nuestro sockey y dirección configurados podemos intentar establecer una conexión (`connect();`) con un servidor que debe estar en "modo escucha" (`listen();`). Este ultimo debe aceptar la conexión (`accept();`) antes de poder enviar y recibir mensajes (`send();` y `recv();`).

Funciones importantes.

```
int listen(int fd, int backlog);
/* fd: descriptor del socket,
 * backlog: numero de conexiones permitidas. */

int connect(int fd, struct sockaddr *server, int addrlen);
/* server: estructura de dirección del servidor */

int accept(int fd, void *addr, int *addrlen);
/* addr: puntero para la dirección de quien nos contacta. */

int send(int fd, const void *msg, int len, int flags);
/* msg: puntero a los datos que queremos enviar */

int recv(int fd, void *buf, int len, unsigned int flags);
/* buf: buffer para guardar los datos */
```

Terminando la conexión.

Una vez enviados los paquetes debemos cerrar los sockets.

```
int shutdown(int fd, int how);
/* fd: descriptor del socket,
 * how: 0 para prohibir la recepción,
        1 para prohibir el envío,
        2 para prohibir ambos. */

/* Utilizar how=2  es equivalente a close() */
close(fd);
```