La semántica de bind en la RFC 2553 bloquea el camino hacia la independencia de protocolos

Horacio J. Peña *

La RFC 2553 fuerza el modelo de IPv4 mapeado en IPv6 para bind(2). Esto ha tenido algunos resultados a corto plazo muy buenos, pero hace un gran daño a la transición hacia la independencia de protocolos, un objetivo que a nuestro juicio deberíamos perseguir.

1- Condición fundamental

Este trabajo parte de la premisa que dice que son mejores los programas independientes de la familia de protocolos usada, que los que funcionan exclusivamente con IPv6.

El fundamento para pensar esto es que no creemos que IPv6 sea el protocolo que vaya a solucionar todos los problemas de las redes y que en algún momento del futuro (probablemente un futuro lejano) habría una nueva transición de IPv6 a algún otro protocolo. Y deberíamos hoy hacer lo que sea necesario para que cuando eso pase los que daban hacer esa transición puedan hacerlo de la manera mas fácil posible.

Estamos viviendo la transición de IPv4 a IPv6 y es muy costosa. Hay mucho trabajo que hacer, y la modificación de las aplicaciones es el mayor responsable de ese costo.

Modificar una aplicación no es tan difícil. Pero cuando hay tantas, se convierte en un problema. ¿Cuántas veces hemos escuchado que la adopción de IPv6 es tan lenta porque no hay soporte de parte de los clientes? ¿Cuánto más fácil podría haber sido si no fuera necesario modificar las aplicaciones para conseguir ese soporte? Creemos que la respuesta es “mucho más fácil”. La modificación de las aplicaciones para que sean independientes de la familia de protocolos usada hará las próximas transiciones mucho mas fáciles.

2- Lo que dice la RFC 2553

Nota: Al hablar de la “RFC 2553” queremos decir “RFC 2553 y sus sucesores’’, por lo que citaremos aquí el internet-draft rfc2553bis-03, no la RFC.

Because of the importance of providing IPv4 compatibility in the API, these extensions are explicitly designed to operate on machines that provide complete support for both IPv4 and IPv6. A subset of this API could probably be designed for operation on systems that support only IPv6. However, this is not addressed in this memo.

(de “2. Design Considerations”)

---

* Alumno de la Facultad de Ingeniería. Universidad de Palermo.

O sea, la RFC 2553 se aplica a equipos con soporte dual IPv4-IPv6.

Applications may use PF_INET6 sockets to open TCP connections to IPv4 nodes, or send UDP packets to IPv4 nodes, by simply encoding the destination's IPv4 address as an IPv4-mapped IPv6 address, and passing that address, within a sockaddr_in6 structure, in the connect() or sendto() call. When applications use PF_INET6 sockets to accept TCP connections from IPv4 nodes, or receive UDP packets from IPv4 nodes, the system returns the peer's address to the application in the accept(), recvfrom(), or getpeername() call using a sockaddr_in6 structure encoded this way.

(de "3.7 Compatibility with IPv4 Nodes")

5.3 IPV6_V6ONLY option for AF_INET6 Sockets

This socket option restricts AF_INET6 sockets to IPv6 communications only. As stated in section <3.7 Compatibility with IPv4 Nodes>, AF_INET6 sockets may be used for both IPv4 and IPv6 communications. Some applications may want to restrict their use of an AF_INET6 socket to IPv6 communications only. For these applications the IPV6_V6ONLY socket option is defined. When this option is turned on, the socket can be used to send and receive IPv6 packets only. This is an IPPROTO_IPV6 level option. This option takes an int value. This is a boolean option. By default this option is turned off.

Esto implica que cuando ligamos un socket INET6 a un puerto (sin especificar una dirección particular a la que ligarlo) este escuchará también los pedidos IPv4 salvo que se active explícitamente la opción IPV6_V6ONLY.

Esto se hace usando las direcciones IPv4 mapeadas a IPv6.

3- Como programar un servidor

Apartémonos un poco del tema. Veremos como se programa un server que conozca solo IPv4, uno que conozco solo IPv6 y otro que sea independiente de la familia de protocolos usada, para poder entender el resto de este trabajo.

3.1- IPv4 server

```
int listenfd, connfd;
struct sockaddr_in cliaddr, servaddr;
socklen_t clilen;
```

```
listenfd = socket(AF_INET, SOCK_STREAM, 0);

if(listenfd < 0)
  die();

memset(&servaddr, 0, sizeof(servaddr));
servaddr.sin_family = AF_INET;
servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
servaddr.sin_port = htons(1000);

if(bind(listenfd, &servaddr, sizeof(servaddr)) != 0)
  die();

if(listen(listenfd, 10) != 0)
  die();

connfd = accept(listenfd, (struct sockaddr *) &cliaddr, &clilen);

if(connfd < 0)
  die();

/* do something with connfd */
```

Esto trabaja sólo con conexiones IPv4, si se intenta una conexión al puerto 1000 de TCP por IPv6 no funcionara.

3.2- IPv6 centric server

```
int listenfd, connfd;
struct sockaddr_in6 cliaddr, servaddr;
socklen_t clilen;

listenfd = socket(AF_INET6, SOCK_STREAM, 0);

if(listenfd < 0)
  die();

memset(&servaddr, 0, sizeof(servaddr));
servaddr.sin6_family = AF_INET6;
servaddr.sin6_addr = in6addr_any;
servaddr.sin6_port = htons(1000);
```

```
if(bind(listenfd, &servaddr, sizeof(servaddr)) != 0)
  die();

if(listen(listenfd, 10) != 0)
  die();

connfd = accept(listenfd, (struct sockaddr *) &cliaddr, &clilen);

if(connfd < 0)
  die();

/* do something with connfd */
```

Casi sin cambios desde el servidor IPv4. Acepta conexiones IPv4 e IPv6. No funciona en SO sin soporte de IPv6 activo (ni siquiera funciona para conexiones IPv4)

3.3- AF independent server

```
int listenfds[MAX_AF], connfd;
struct addrinfo hints, *res, *ressave;
struct sockaddr_storage ss;
socklen_t sslen;
int n, i, m;
fd_set fdset;

memset(&hints, 0, sizeof(hints));
hints.ai_flags = AI_PASSIVE;
hints.ai_family = AF_UNSPEC;
hints.ai_socktype = SOCK_STREAM;

if(getaddrinfo(NULL, “1000”, &hints, &res) != 0)
  die();

ressave = res;

for(n = 0; (n < MAX_AF) && res ; res = res->ai_next) {
     listenfds[n] = socket(res->ai_family, res->ai_socktype,
              res->ai_protocol);
     if(listenfds[n] < 0)
        continue; /* libc supports protocols that kernel don’t */

     if(bind(listenfds[n], res->ai_addr, res->ai_addrlen) != 0)
       die();
```

```
        if(listen(listenfds[n], 10) != 0)
       die();
         n++;
         }

freeaddrinfo(ressave);

m = 0;
FD_ZERO(&fdset);
for(i = 0; i < n; i++) {
        FD_SET(listenfds[i], &fdset);
        m = MAX(listenfds[i]+1,m);
         }

if(select(m, &fdset, NULL, NULL, NULL) < 0)
  die();

for(i = 0; i < n; i++) {
        if(FD_ISSET(listenfds[i], &fdset)) {
        sslen = sizeof(ss);
                connfd = accept(listenfds[i], (struct sockaddr*) &ss, &sslen);
               break;
               }
         }

if(connfd < 0)
  die();

/* do something with connfd */
```

Mucho mas complicado.

3.4- Comentario

Pero esto no es todo.

Ni el método de IPv6 exclusivo, ni el independiente de familia de protocolos funcionan tan simplemente como los hemos presentado. El programa IPv6 puro no funciona cuando el SO no tiene activado IPv6, por lo tanto cuando se programa de esta manera se debería verificar la respuesta de la llamada a socket y si esta falla caer a trabajar como un server IPv4 (con la duplicación de código que esto conlleva)

Y acerca del modelo independiente de protocolos... Hablaremos acerca de los problemas que tiene en el resto de este trabajo.

Pero, incluso si ambos métodos funcionaran tan bien como la sección anterior daba a entender, aunque el modelo independiente de protocolos es mucho más complicado, es un cambio definitivo, mientras que con el modelo de IPv6 puro se deberá volver a modificar el programa cuando se quiera manejar algún protocolo que no pueda ser mapeado a IPv6.

Volvamos a la RFC 2553.

4- Implementaciones de RFC 2553

Clasificamos las implementaciones de la RFC 2553 según como implementan la semántica de bind.

4.1- Las que no cumplen

Estos sistemas consideran IPv4 e IPv6 como protocolos diferentes, por lo que no existe el mapeo de IPv4 a IPv6.

El modelo de programación independiente de protocolo funciona bien. El modelo de IPv6 puro funciona sólo para conexiones IPv6, dado que los sockets IPv6 nunca toman conexiones IPv4.

OpenBSD, NetBSD (en su configuración por defecto) y Windows son algunos de los sistemas con implementaciones que no cumplen la especificación.

4.2- Las falladas

Nota: hablamos acerca de "falladas" sólo acerca del tema en discusión, la calidad de los sistemas que calificamos de fallados es muy buena, y en cierta manera preferimos este fallo que la corrección que hace el trabajo independiente de protocolo algo muy difícil.

Estos sistemas permiten que se ligue un puerto a un socket IPv4 cuando el mismo puerto esta ya ligado a un socket IPv6. Esto permite "robar" conexiones a otro proceso, lo que debería ser considerado un fallo.

Este comportamiento hace que funcionen bien los modelos IPv6 puro e independiente de protocolos. Pero tiene un fallo grave.

FreeBSD, NetBSD (opcionalmente) y BSDI tienen esta conducta fallida, así como la mayor parte de los sistemas propietarios.

4.3- La que cumple y es correcta, pero intratable

Esta es la de Linux. Linux cumple con la RFC permitiendo que los sockets IPv6 tomen conexiones IPv4, no tiene el bug mencionado antes, pero es imposible trabajar según el modelo independiente de protocolo.

Nota: No estamos diciendo que Linux no tenga bugs. Los tiene y muchos. Tratamos sólo de las semánticas de bind en este trabajo.

Durante el ciclo socket/bind/listen, la llamada a bind para el socket IPv4 falla porque hay un socket IPv6 ligado al puerto requerido. Esto implica que han de ignorarse los errores de bind, o hacerles caso sólo si ninguna de las llamadas a bind funciona. Pero esto sería lo mismo que ignorar los errores de bind si existe un nuevo protocolo que no pueda mapearse a alguno de los otros. Ignorar los errores del sistema operativo puede traer grandes problemas.

4.4- Conclusión: no hay ninguna buena implementación de la RFC 2553

Es nuestra creencia que no hay ninguna buena implementación de la RFC 2553 porque la especificación no es buena en este punto y que se debería trabajar más al respecto.

Hasta que la especificación se cambie, sólo podemos sugerir algunas formas de solucionar este problema.

5- Soluciones posibles

Cualquiera de las posibles soluciones presentadas es suficientemente buena para nosotros, siendo nuestro objetivo poder programar aplicaciones portables entre plataformas y entre protocolos.

Creemos que las direcciones IPv4 mapeadas a IPv6 serán abandonadas tarde o temprano porque el mismo IPv4 será abandonado. Y creemos que tal vez es el momento de empezar a abandonarlas, no prohibiéndolas, sino aceptando la existencia de sistemas en que los protocolos IPv4 e IPv6 estén aislados.

5.1- Abandonar las direcciones IPv4 mapeadas

Quizá el tiempo de las direcciones IPv4 mapeadas haya pasado, tal vez han sido un buen mecanismo para que las cosas empezaran a funcionar pero ya es tiempo de crecer y dejarlo a un lado.

Itojun ha mencionado en su ipv6-transition-abuse internet-draft varios otros problemas que tienen estas direcciones.

Pero hay demasiado trabajo hecho ya según el modelo de IPv6 puro, por lo que quizá no es prudente abandonarlas de un golpe.

5.2- Hacer que IPV6_V6ONLY este activa por default

En este caso los sockets IPv6 deberán ser explícitamente habilitados para aceptar conexiones IPv4. Los programas que usen el modelo IPv6 puro deberán ser modificados levemente para seguir funcionando, pero las implementaciones falladas y la intratable podrían ser corregidas sin perder capacidades.

5.3- Más magia a getaddrinfo

Tomemos el modelo de linux como bueno (es el único que cumple y no tiene fallos en bind) agreguémosle un poco más (todavía más!) de magia a getaddrinfo para que devuelva el socket de INET sólo cuando el kernel no tiene soporte para IPv6, y entonces los sistemas fallados podrían ser corregidos sin perdida de capacidades y el intratable se volvería tratable.

5.4- Agréguese la provisión para permitir implementación dobles

La RFC 2553 apunta a los sistemas de protocolo dual, donde IPv4 e IPv6 son tratados como un mismo protocolo.

Si la especificación tuviera un comentario diciendo que está permitida la existencia de sistemas con protocolos independientes, y que en esos sistemas el mapeo de IPv4 a IPv6 puede no existir, las implementaciones que no cumplen pasarían a cumplir, y tendríamos automáticamente implementaciones adecuadas, sin fallos y en las que sea fácil trabajar independientemente del protocolo usado.