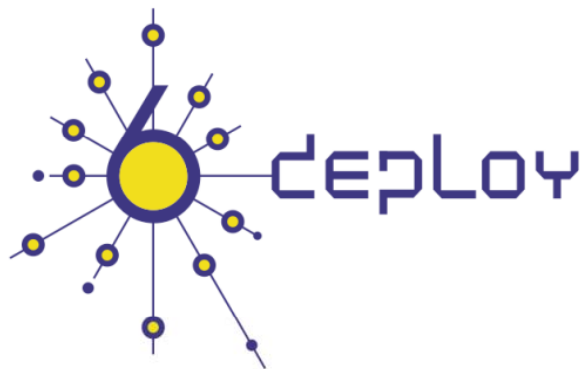


Curso IPv6

WALC 2009

Bogotá – Colombia

21 al 25 Septiembre 2009



César Olvera (cesar.olvera@consulintel.es)

Alvaro Vives (alvaro.vives@consulintel.es)



Contenido del curso (1)

- **Bloque 1. Tutorial IPv6**

1. Introducción a IPv6
2. Formatos de cabeceras y tamaño de paquetes
3. Direccionamiento IPv6
4. ICMPv6, Neighbor Discovery y DHCPv6
5. Seguridad IPv6
6. Encaminamiento con IPv6
7. Mecanismos de Transición
8. Movilidad IPv6



Contenido del curso (2)

- **Bloque 2. Otros Aspectos Avanzados**
 9. Calidad de Servicio (QoS)
 10. Multicast
 11. Multi-homing
 12. Porting de aplicaciones
 13. Gestión SNMP sobre IPv6
 14. IPv6 sobre MPLS
 15. DNS IPv6





Bloque 2

Otros Aspectos Avanzados





12. Porting de Aplicaciones

12.1 Introducción

12.2 Cambios con IPv6





12.1 Introducción



Centrando el problema del porting a IPv6

- El cambio en la capa de Red no es transparente.
 - Las aplicaciones IPv4 necesitan modificarse para ser usadas con IPv6
- La mejor aproximación es convertir las aplicaciones IPv4 en aplicaciones independientes del protocolo de Red usado
- Usualmente no es una tarea demasiado difícil
 - Aplicaciones sencillas como por ejemplo telnet tan solo requiere de una pocas horas para realizar el porting a IPv6
- Debe tenerse cuidado con la manera de elegir entre IPv6 e IPv4 cuando ambos están disponibles
 - Las aplicaciones pueden necesitar reintentar las conexiones debido a varias direcciones IPv6 disponibles (o IPv6 e IPv4)
- Información en RFC3493, RFC3542, RFC4038, RFC5014





12.2 Cambios con IPv6



Cambios principales desde IPv4

- Tamaño de direcciones.
 - 32 bits (IPv4) a 128 bits (IPv6)
- Implicaciones de la nueva representación de direcciones
 - Se pueden necesitar hasta 39 caracteres (45 para IPv4-mapped)
- Cambios necesarios en la API
 - Tamaño de direcciones
 - Independencia del protocolo
- Dependencias del tamaño de la cabecera IP
- Dependencias con direcciones particulares
- **RECOMENDACIÓN:** Usar FQDN en vez de direcciones IP
 - También a la hora de almacenarlos



No todas las aplicaciones necesitan ser cambiadas

- Muchas aplicaciones no se comunican directamente con la Red sino que emplean funciones de librería que se encargan de esas tareas. En estos casos solo se necesita modificar dichas librerías
- Ejemplos:
 - RPC
 - DirectPlay



Problemas con el almacenamiento de direcciones

- Problema: no se puede almacenar un valor de 128 bits en un lugar reservado a 32 bits.
- La mayoría de las aplicaciones actualmente almacenan y referencian las direcciones como:
 - sockaddrs (bueno)
 - in_addrs (correcto)
 - ints (malo)
- Almacenamiento versus referencia



Anatomía de la estructura sockaddr

```
struct sockaddr {  
    u_short sa_family; // Address family  
    char sa_data[14]; // Address data  
};
```

- El campo sa_family contiene un valor que indica de qué tipo de dirección se trata (IPv4, IPv6, etc)



sockaddr_in

```
struct sockaddr_in {  
    short sin_family;  
    u_short sin_port;  
    struct in_addr sin_addr;  
    char sin_zero[8];  
};
```

- Cuando se programa con IPv4, la estructura `sockaddr` se rellena utilizando la estructura `sockaddr_in` que es específica para IPv4



sockaddr_in6

```
struct sockaddr_in6 {  
    short sin6_family; // AF_INET6  
    u_short sin6_port;  
    u_long sin6_flowinfo;  
    struct in_addr6 sin6_addr;  
    u_long sin6_scope_id;  
};
```

- Cuando se programa con IPv6, la estructura `sockaddr` se rellena utilizando la estructura `sockaddr_in6` que es específica para IPv6
- Para portar código hay que cambiar la estructura `sockaddr_in` por `sockaddr_in6`



Anatomía de la estructura sockaddr_storage

```
struct sockaddr_storage
{
    sa_family_t  ss_family;      /* Address family  */
    __ss_aligntype __ss_align;  /* Force alignment.*/
    char  __ss_padding[_SS_PADSIZE];
};
```

- Cuando se escribe código nuevo y se desea independencia del tipo de protocolo de red, es mejor usar la estructura `sockaddr_storage` en vez de `sockaddr`
 - Tamaño suficiente para albergar cualquier dirección



Cambios en la API

- La mayoría de las APIs de sockets no necesitan ser cambiadas puesto que fueron originalmente diseñadas para ser independientes del protocolo de Red, de manera que toman punteros a sockaddrs como entradas o salidas
 - bind, connect, getsockname, getpeername, etc.
- Las APIs de resolución de nombres son las grandes perjudicadas que necesitan ser cambiadas
 - gethostbyname, gethostbyaddr



APIs de Resolución de Nombres Nuevas

- **getaddrinfo**: Para encontrar la dirección y/o el número de puerto que se corresponde con un nombre de nodo y servicio dado
- **getnameinfo**: Para encontrar el nombre del nodo y/o servicio que se corresponde respectivamente con una dirección o número de puerto dados
- Ambas APIs son ya independientes del protocolo, de manera que funcionan correctamente tanto con IPv4 como con IPv6



Getaddrinfo

```
int  
getaddrinfo(  
    IN const char FAR * nodename,  
    IN const char FAR * servicename,  
    IN const struct addrinfo FAR * hints,  
    OUT struct addrinfo FAR * FAR * res  
);
```



Anatomía de addrinfo

```
typedef struct addrinfo {
    int ai_flags;
    int ai_family; // PF_xxx.
    int ai_socktype; // SOCK_xxx.
    int ai_protocol; // IPPROTO_xxx.
    size_t ai_addrlen;
    char *ai_canonname;
    struct sockaddr *ai_addr;
    struct addrinfo *ai_next;
    int ai_eflags; // Extended flags for special usage
} ADDRINFO, FAR * LPADDRINFO;
```

- `getaddrinfo` devuelve la información en forma de lista de estructuras `addrinfo`



Getnameinfo

```
int  
getnameinfo(  
    IN  const struct sockaddr FAR * sa,  
    IN  socklen_t salen,  
    OUT char FAR * host,  
    IN  DWORD hostlen,  
    OUT char FAR * service,  
    IN  DWORD servlen,  
    IN  int flags  
);
```



Selección de la dirección origen en sockets IPv6 (1)

- Problema: Los nodos IPv6 pueden tener diversas direcciones IPv6 globales y válidas
 - Formadas mediante RA
 - Permanentes
 - Temporales
 - CGAs
 - Asignadas mediante DHCPv6
 - HoA y CoA cuando se emplea MIPv6
- En estas condiciones el nodo IPv6 debe poder elegir una dirección IPv6 concreta para establecer comunicación con otro nodo IPv6
 - El RFC3484 define un algoritmo por defecto para la preferencia en la selección de la dirección más adecuada. En resumen:
 - Direcciones permanentes sobre direcciones temporales
 - Dirección HoA sobre dirección CoA
 - Dirección de ámbito mayor sobre otra de ámbito menor
 - Dirección CGA en entorno SEND sobre dirección no-CGA
- En algunas ocasiones, las aplicaciones tienen la necesidad de no seguir el criterio por defecto. Algunos ejemplos:
 - Si se está empleando MIPv6, el MN podría usar la CoA para resoluciones DNS, puesto que el servidor DNS usado estará más cerca de la red visitada
 - Un navegador web podría preferir el uso de una dirección temporal en vez de la permanente



Selección de la dirección origen en sockets IPv6 (2)

- Para permitir excepciones al comportamiento por defecto en la selección de la dirección origen se ha estandarizado una extensión de la API IPv6
 - RFC5014
- Se define una nueva opción de socket para el nivel IPPROTO_IPV6
 - IPV6_ADDR_PREFERENCES
 - La nueva opción se puede emplear con las funciones `setsockopt()` y `getsockopt()` para configurar y obtener las preferencias en la selección de la dirección origen que afectará a todos los paquetes que se envíen por un socket
- Se han definido los siguientes FLAGS nuevos definidos en `<netinet/in.h>`:
 - `IPV6_PREFER_SRC_HOME /* Prefer Home address as source */`
 - `IPV6_PREFER_SRC_COA /* Prefer Care-of address as source */`
 - `IPV6_PREFER_SRC_TMP /* Prefer Temporary address as source */`
 - `IPV6_PREFER_SRC_PUBLIC /* Prefer Public address as source */`
 - `IPV6_PREFER_SRC_CGA /* Prefer CGA address as source */`
 - `IPV6_PREFER_SRC_NONCGA /* Prefer a non-CGA address as source */`



Selección de la dirección origen en sockets IPv6 (3)

- Además se debe utilizar la función `getaddrinfo()`, utilizando la variable `ai_eflags` de la estructura `addrinfo` para establecer la preferencia.
- El orden en la programación sería:
 - Configuración del flag para establecer la preferencia
 - Llamada a `getaddrinfo()`
 - Llamada a `setsockopt()`



Ejemplo de selección de la dirección origen en sockets IPv6 (4)

```
struct addrinfo hints, *ai, *ai0;
uint32_t preferences;

preferences = IPV6_PREFER_SRC_TMP;

hints.ai_flags |= AI_EXTFLAGS;
hints.ai_eflags = preferences; /* Chosen address preference flag */
/* Fill in other hints fields */

getaddrinfo(.....,&hints,. &ai0..);

/* Loop over all returned addresses and do connect */
for (ai = ai0; ai; ai = ai->ai_next) {
    s = socket(ai->ai_family, ...);
    setsockopt(s, IPV6_ADDR_PREFERENCES, (void *) &preferences,
               sizeof (preferences));
    if (connect(s, ai->ai_addr, ai->ai_addrlen) == -1){
        close (s);
        s = -1;
        continue;
    }
    break;
}

freeaddrinfo(ai0);
```



Dependencias con el Tamaño de la Cabecera

- Problema: La cabecera IPv6 mínima es 20 bytes mayor que la cabecera IPv4
- Los programas que calculan el tamaño reservado para los datos de un datagrama de esta manera: $\text{Path MTU} - (\text{tamaño cabecera UDP} + \text{tamaño cabecera IP})$ necesitan saber que el tamaño de la cabecera IP ha cambiado y es 20 bytes mayor



Dependencias con las Direcciones IPv4

- Algunos programas conocen ciertas direcciones (por ejemplo loopback = dirección IPv4 127.0.0.1)
- Los programas que se encargan de manipular direcciones (por ejemplo NATs) obviamente tienen un conocimiento innato de las direcciones IPv4
- Este tipo de problema solo se da en programas de este tipo
- NATs son indeseados de cualquier forma, de manera que no es necesario preocuparse por su solución ☺



IPv6 en Java

- Java oculta casi completamente los detalles acerca de la doble pila
 - Simplificación en la tarea de programación
- Existe soporte de IPv6 en Java desde la versión 1.5



Gracias !!

Contacto:

- Cesar Olvera (Consulintel): cesar.olvera@consulintel.es
- Alvaro Vives (Consulintel): alvaro.vives@consulintel.es

6DEPLOY Project: <http://www.6deploy.org>

The IPv6 Portal: <http://www.ipv6tf.org>

