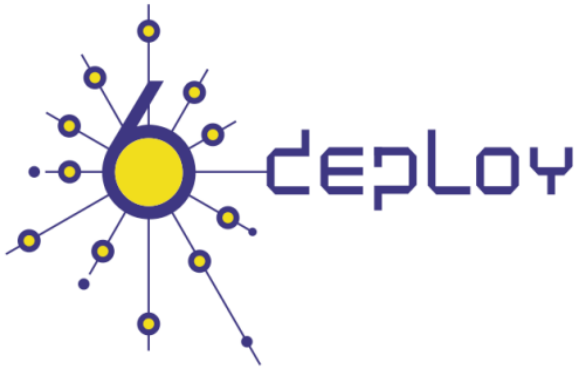
WALC2011
Track 2: Despliegue de IPv6
Día -5
Guayaquil - Ecuador
10-14 Octubre 2011



Alvaro Vives (alvaro.vives@consulintel.es)

Agenda

10. Calidad de Servicio (QoS)
11. IPv6 sobre MPLS
12. Movilidad IPv6
13. Multi-homing
14. Porting de Aplicaciones
15. Multicast

14. Porting de Aplicaciones

14.1 Introducción

14.2 Cambios con IPv6

14.3 Escenarios Transición

14.4 Lenguajes de programación

14.1 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

14.2 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 independienes 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 esta 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 unas 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

```
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: Path MTU – (tamaño cabecera UDP + tamaño cabecera IP) necesitan saber que el tamaño de la cabecera IP ha cambiado y es 20 bytes mayor

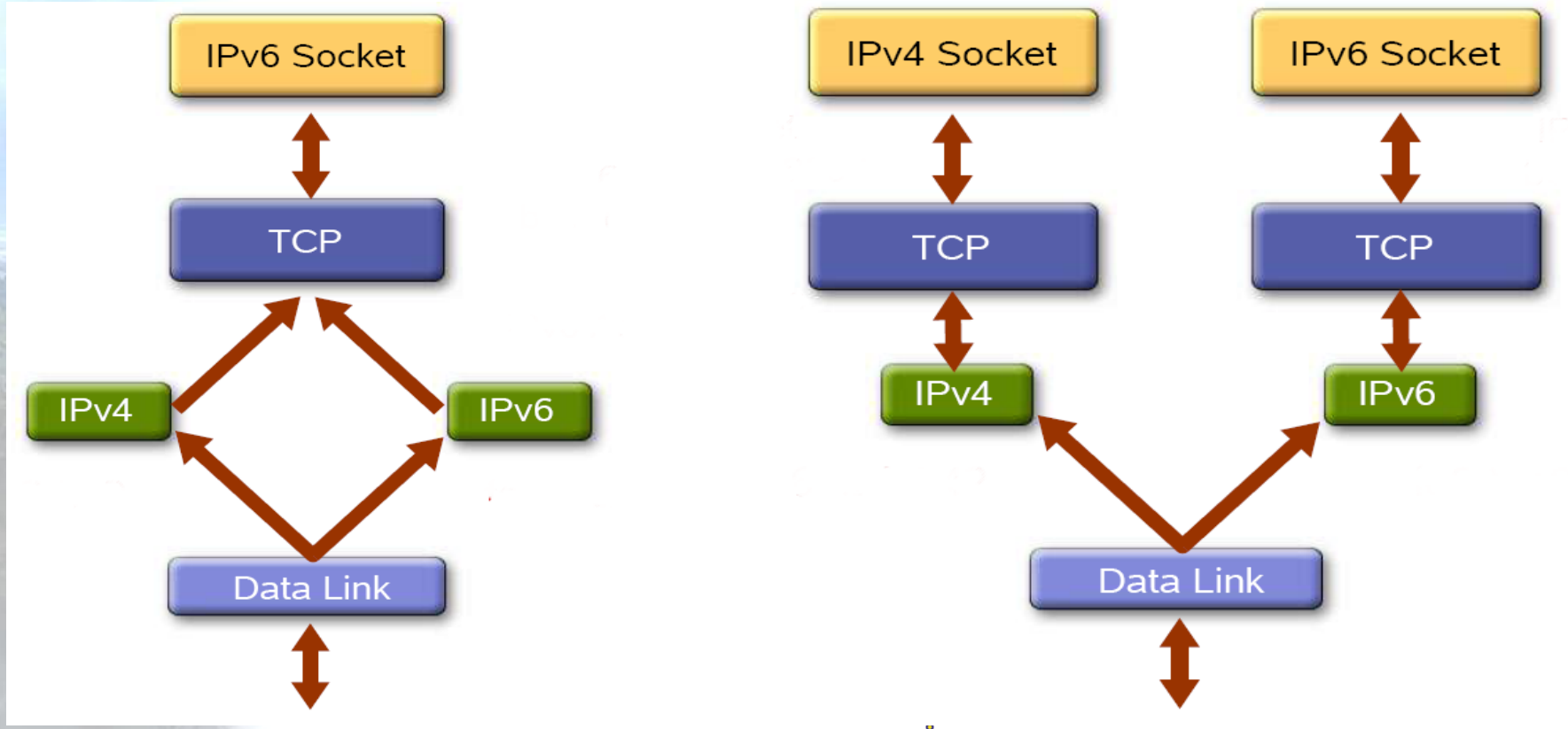
Dependencias con las Direcciones IPv4

- 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

14.3 Escenarios Transición

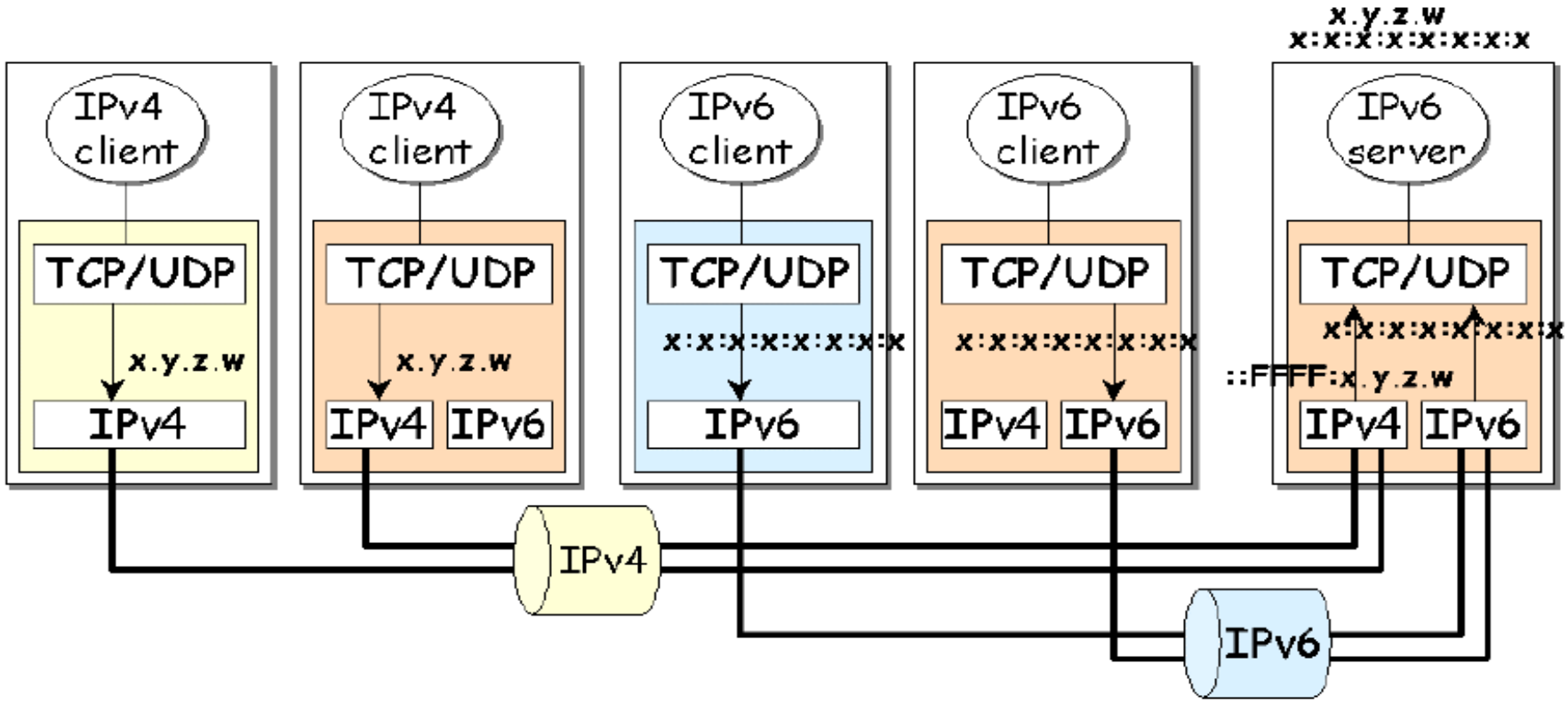
Escenarios (1)

- Las dos pilas (IPv4 e IPv6) estarán disponibles durante la transición. Dos posibles implementaciones de la pila de red



Escenarios (2)

- Clientes IPv6/IPv4 conectándose a un servidor IPv6 en un nodo doble pila -> 1 socket

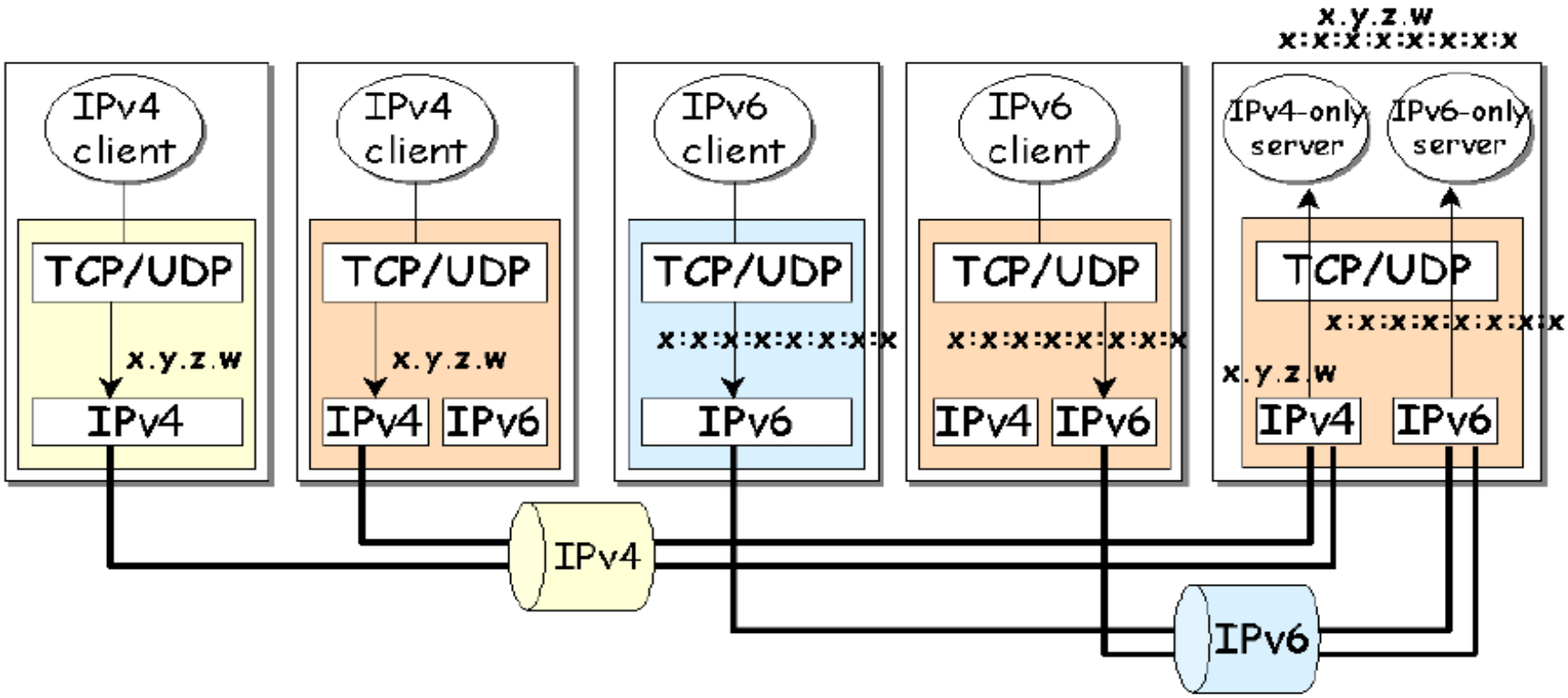


Dual Stack

Single IPv4 or IPv6 stacks

Escenarios (3)

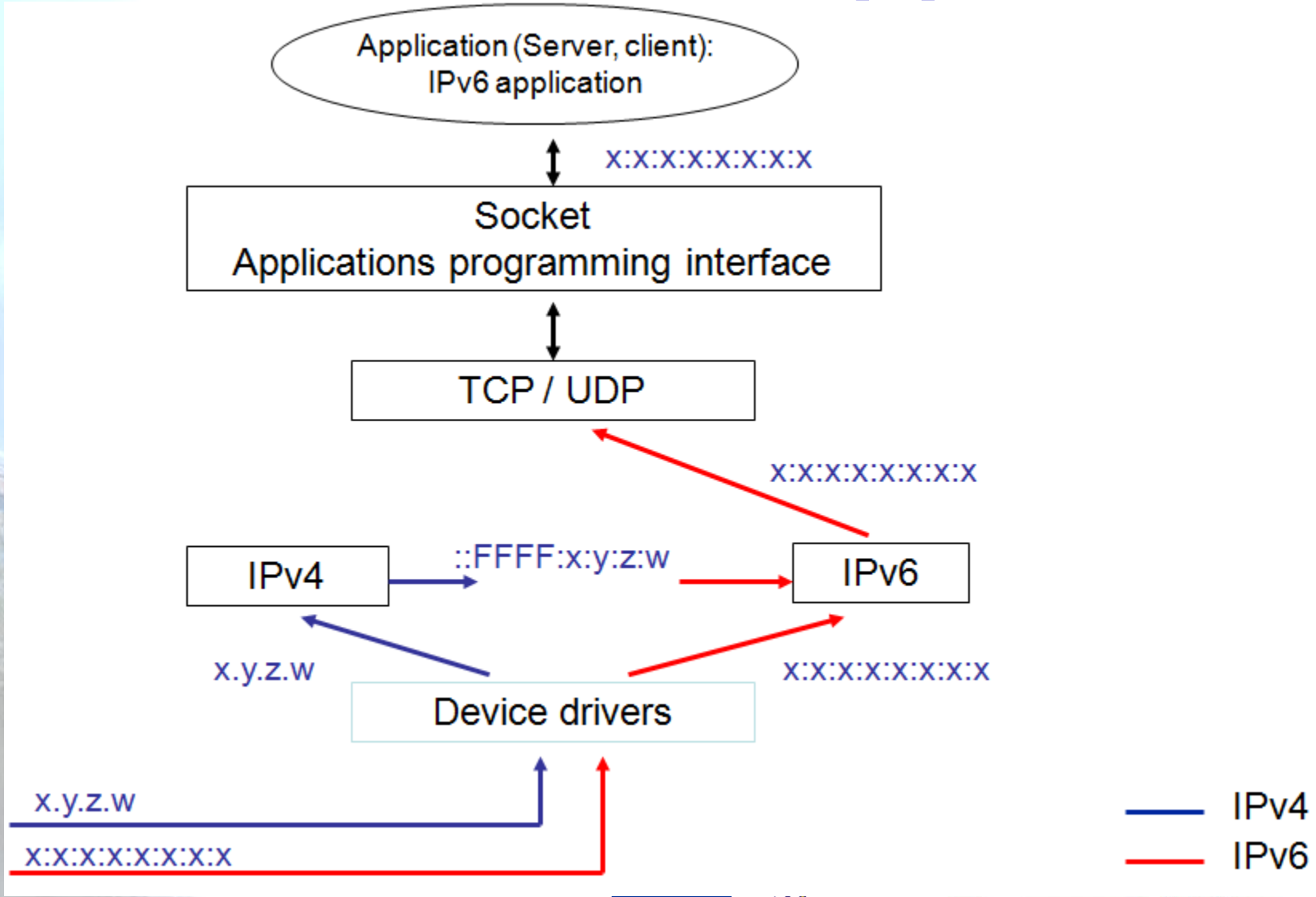
- Clientes IPv6/IPv4 conectándose a un servidor solo-IPv4 y a otro servidor solo-IPv6 en un nodo doble-pila -> 2 sockets



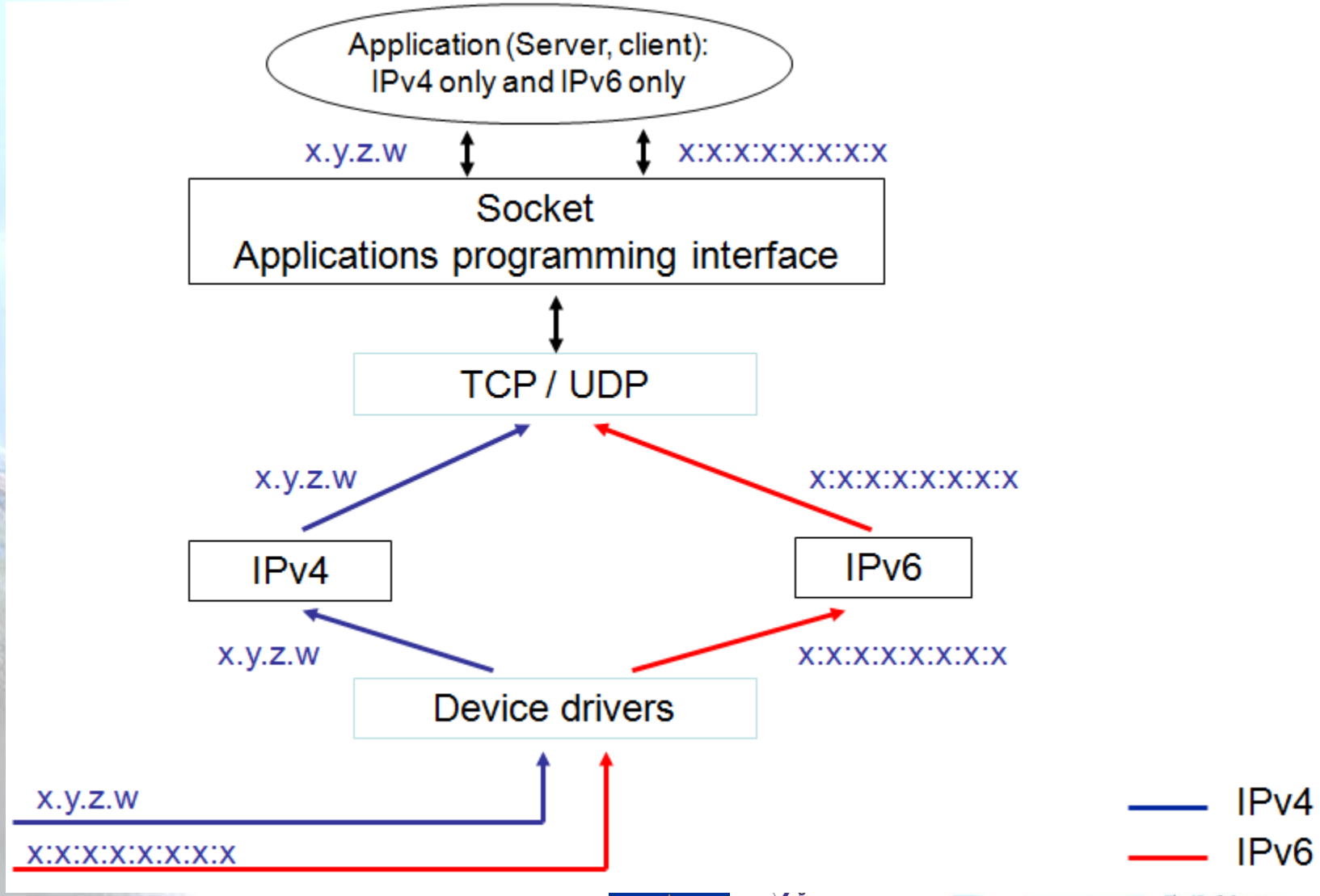
Dual Stack or separated stack

Single IPv4 or IPv6 stacks

Escenarios (4)



Escenarios (5)



14.4 Lenguajes de Programación

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

IPv6 en Perl (1)

- IPv6 API para Perl 5
- Dos módulos ofrecen la API IPv6:

1. Módulo **Socket6**. Disponible en CPAN. Funciones:
   - getaddrinfo()
   - gethostbyname2 HOSTNAME, FAMILY - family specific gethostbyname
   - getnameinfo NAME, [FLAGS] - see usage later
   - getipnodebyname HOST, [FAMILY, FLAGS] - list of five elements - usage not recommended
   - getipnodebyaddr FAMILY, ADDRESS - list of five elements - usage not recommended
   - gai_strerror ERROR_NUMBER - returns a string of the error number
   - inet_pton FAMILY, TEXT_ADRESS - text->binary conversion
   - inet_ntop FAMILY, BINARY_ADDRESS - binary-> text conversion

IPv6 en Perl (2)

- pack_sockaddr_in6 PORT, ADDR - creating sockaddr_in6 structure
- pack_sockaddr_in6_all PORT, FLOWINFO, ADDR, SCOPEID - complete implementation of the above
- unpack_sockaddr_in6 NAME - unpacking sockaddr_in6 to a 2 element list
- unpack_sockaddr_in6_all NAME - unpacking sockaddr_in6 to a 4 element list
- in6addr_any - 16-octet wildcard address.
- in6addr_loopback - 16-octet loopback address

IPv6 en Perl (3)

2. **IO::Socket::INET6** facilita la creación de sockets.

Hereda todas las funciones de IO::Socket + IO::Handle. Es una generalización de IO:Socket:INET para ser independiente del protocolo. Disponible en CPAN. Nuevos métodos:

- sockdomain() - Returns the domain of the socket - AF_INET or AF_INET6 or else
- sockflow () - Return the flow information part of the sockaddr structure
- sockscope () - Return the scope identification part of the sockaddr structure
- peerflow () - Return the flow information part of the sockaddr structure for the socket on the peer host
- peerscope () - Return the scope identification part of the sockaddr structure for the socket on the peer host.

Gracias !!

Contacto:

- Alvaro Vives (Consulintel): alvaro.vives@consulintel.es