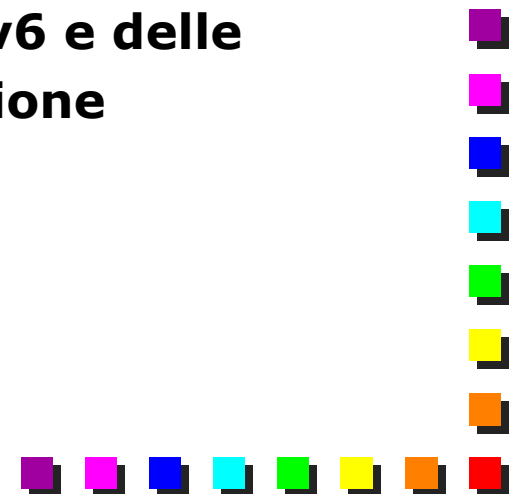




La transizione ad IPv6

Panoramica delle principali problematiche legate alla transizione da IPv4 ad IPv6 e delle tecniche per facilitare la migrazione



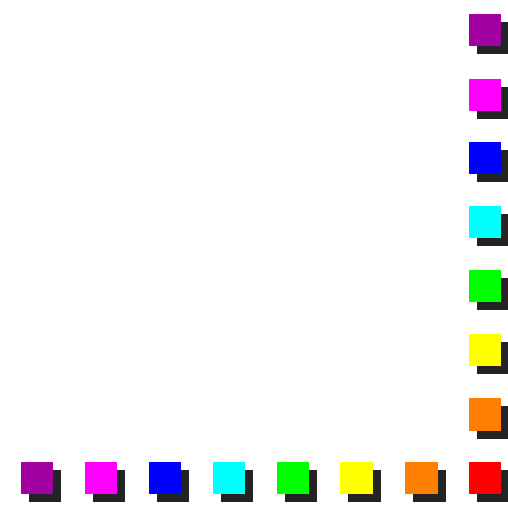


Le difficoltà della transizione

■ Transizione

- Necessaria principalmente per garantire l'interoperabilità con macchine IPv4-only
- Non un problema (al momento) l'interoperabilità con macchine IPv6-only
- IPv4 e IPv6 sono protocolli diversi e non interoperano tra loro

■ Migrazione di:

- Apparati di rete
 - Rete
 - Host
 - Applicazioni
- 

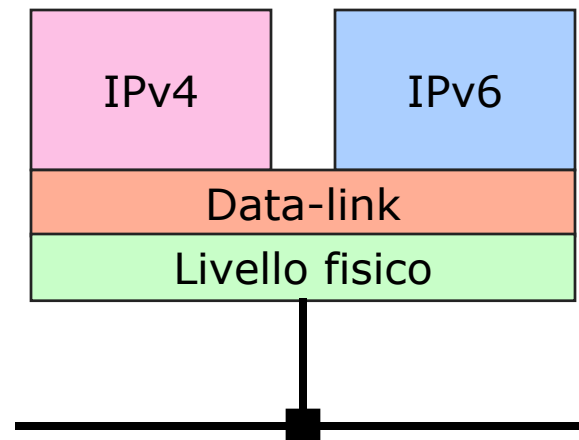
Migrazione degli apparati di rete

■ Problema solamente per apparati L3+

- Alcuni problemi possono insorgere per apparati L2 con funzioni particolari (es. IGMP snooping, ...), principalmente dovuto ad errori di implementazione

■ Dual stack, approccio "ships in the night"

- Protocolli di routing, routing tables, access lists





Migrazione degli host


■ Dual Stack

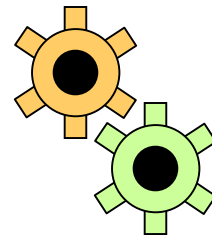
- Soluzione maggiormente gettonata
- Stack IPv6 e stack IPv4 su ogni host
- Supporto completo per entrambi i protocolli
- Attualmente più utilizzata nella variante "dual layer"

■ Limiti:

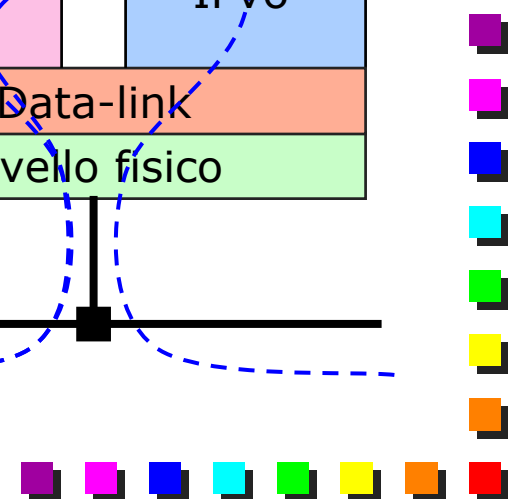
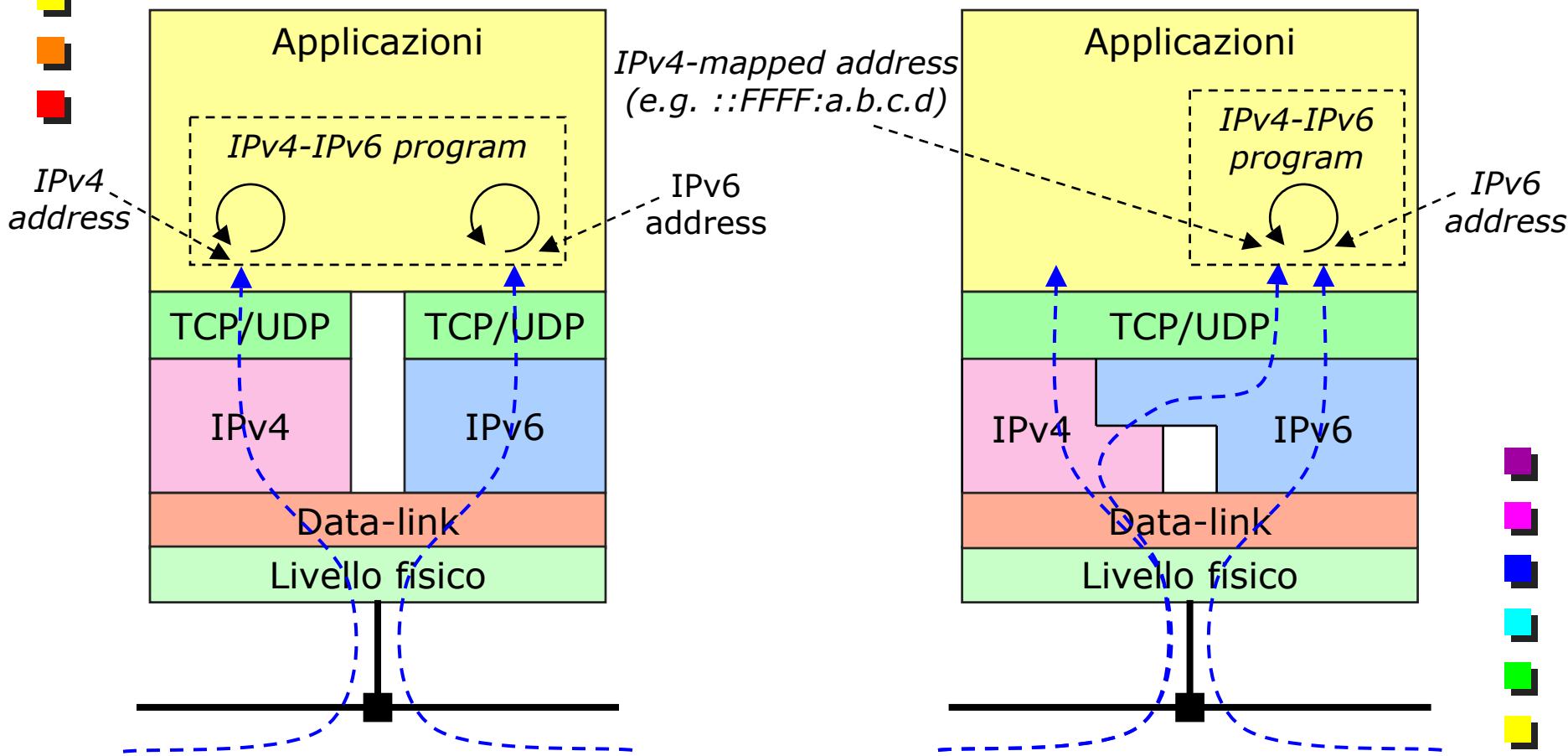
- non riduce fabbisogno di indirizzi IPv4
- aumenta complessità della rete

■ Altre soluzioni dual-stack :

- DSTM (Dual Stack Transition Mechanism)
 - ALG (Application Level Gateway)
- 
- 
- 
- 
- 
- 
- 
- 



Dual stack vs. Dual Layer






Migrazione della rete

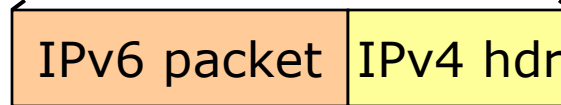
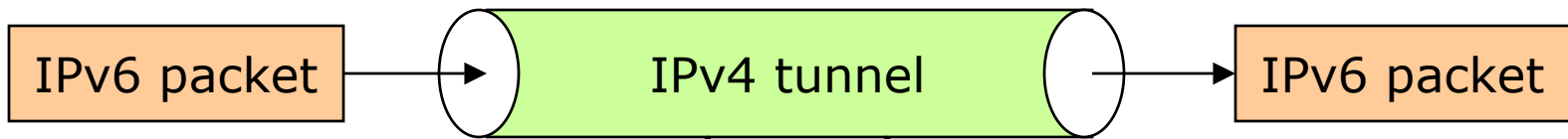
■ Soluzione principe: Tunnelling

- Permette di collegare reti IPv6 tra loro, anche se interconnesse da nuvole IPv4

■ Meccanismi di tunneling in uso :

- Configured Tunnelling
 - tunneling manuale
 - Automatic Tunnelling
 - Indirizzi "IPv4-compatible"
 - 6over4 (RFC 2529)
 - 6to4
 - Tunnel Broker (RFC 3053)
 - ISATAP
 - Teredo
- 

Tunnelling



- IPv6 in IPv4 (protocol type= 41)
- GRE
- ...

Migrazione della rete

- **Problema da risolvere:**

Garantire il recapito di pacchetti IPv6 attraverso nuvole IPv4

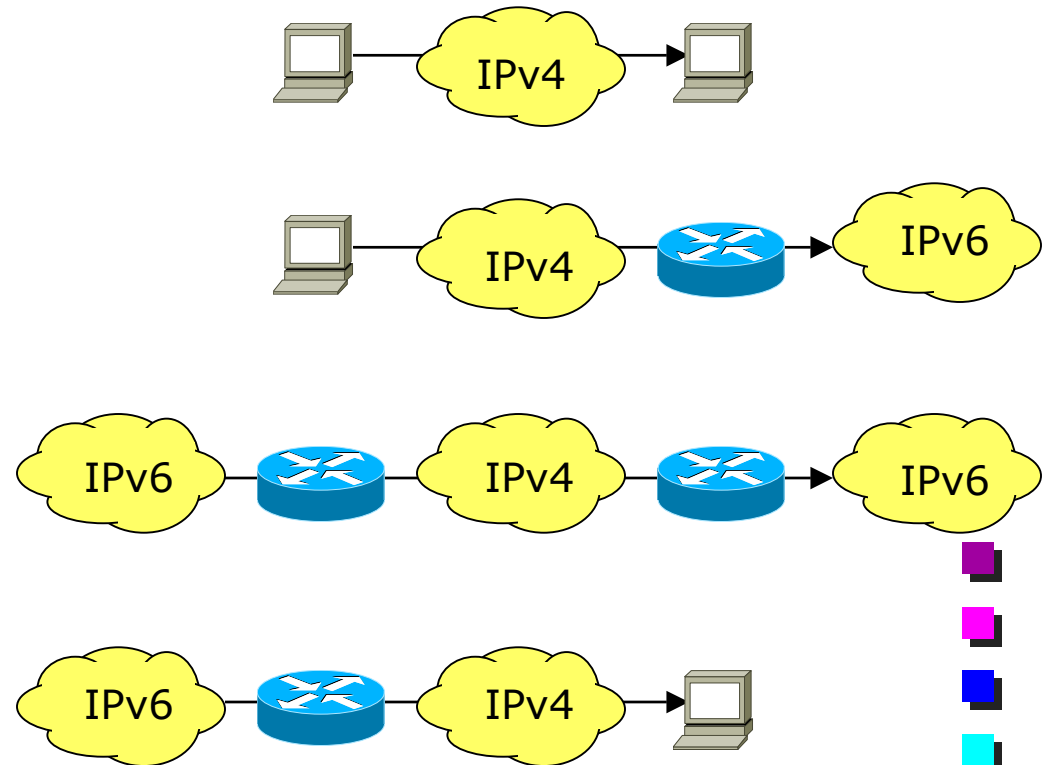
- **Falso problema**

Garantire il recapito di pacchetti IPv4 all'interno di nuvole IPv6

- **Ipotesi**

- Tutti i nodi della rete "attivi" (relativamente alla) transizione sono dual-stack

Scenari





Indirizzi "IPv4-compatible"

- **Spesso indicato impropriamente come "automatic tunnelling"**
 - Viene definita una pseudo-interfaccia "Automatic Tunneling Pseudo-Interface"
 - Tutti i pacchetti uscenti da essa vengono "tunnellati" in base all'indirizzo IPv6 di destinazione
 - L'indirizzo IPv6 deve essere un "IPv4-compatible"
 - Es. `::130.192.226.140`
 - È necessario definire una route per instradare in questo modo solamente i pacchetti indirizzati verso `::/96`
 - Teoricamente possibile annunciare anche route più specifiche di `::/96` sulla rete IPv6, ma questo porterebbe ad un'esplosione delle tabelle di routing

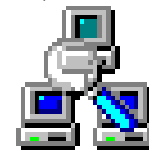




Indirizzi "IPv4-compatible": catture

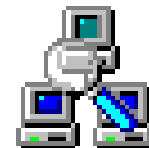
Traffico con route diretta attraverso la rete IPv4 (scenario 1)

```
C:\> netsh interface ipv6 add route ::/96 "Automatic Tunneling Pseudo-Interface"  
C:\> ipv6 rt  
::/96 -> 2 pref lif+0=1 life infinite, publish, no aging (manual)  
::/0 -> 3/2002:c058:6301::c058:6301 pref lif+2147483647=2147483648 life 2h/30m, publish, no aging (manual)  
::/0 -> 3/2002:836b:213c:1:e0:8f08:f020:8 pref lif+1180=1181 life 2h/30m, publish, no aging (manual)  
2002::/16 -> 3 pref lif+1000=1001 life 2h/30m, publish, no aging (manual)  
C:\> ping ::130.192.225.135
```



Traffico con route attraverso un gateway IPv4/IPv6 (scenario 2)

```
C:\> netsh interface ipv6 add route ::/96 "Automatic Tunneling Pseudo-Interface" ::163.162.170.177  
C:\> ipv6 rt  
::/96 -> 2/::163.162.170.177 pref lif+0=1 life infinite (manual)  
::/0 -> 3/2002:c058:6301::c058:6301 pref lif+2147483647=2147483648 life 2h/30m, publish, no aging (manual)  
::/0 -> 3/2002:836b:213c:1:e0:8f08:f020:8 pref lif+1180=1181 life 2h/30m, publish, no aging (manual)  
2002::/16 -> 3 pref lif+1000=1001 life 2h/30m, publish, no aging (manual)  
C:\> ping ::130.192.225.135
```





6over4

- **“Virtual Ethernet”**

- Basato sul multicast
- Richiede la configurazione di una interfaccia di tipo “6over4”

- **Indirizzi**








- NetID:InterfaceID/64
 - InterfaceID è derivato dall'indirizzo IPv4

- **Attualmente in disuso**

- Infrastruttura multicast spesso non disponibile



6over4 e multicast IPv4

- **Indirizzi 239.192.[penultimo byte IPv6].[ultimo byte IPv6]**
 - Es: FF02::1 → 239.192.0.1
 - Es: FF02::1:FF28:9C5A (indirizzo mcast del nodo richiesto) → 239.192.156.90
 - L'indirizzo multicast IPv4 può a sua volta essere mappato su un indirizzo multicast Ethernet
 - **Limitato alla rete aziendale ("site") e non all'intera Internet**
 - **Va configurato opportunamente lo scope del multicast IPv4**
 - Indirizzi IPv6 con prefisso diverso possono far uso dello stesso multicast-v4
 - Indirizzi link-local (FE80::a.b.c.d) potrebbero avere uno scope globale
 - **IGMP per segnalare l'appartenenza ad un gruppo multicast**
 - **Neighbor discovery: esteso per supportare "link-layer" di tipo IPv4**
 - **Router advertisement/solicitation: utilizzato per configurare indirizzi globali**
- 
- 
- 
- 
- 
- 
- 



6to4

Your IPv4 address:

62.157.9.98

Decimal:

62 157 9 98

Hex:

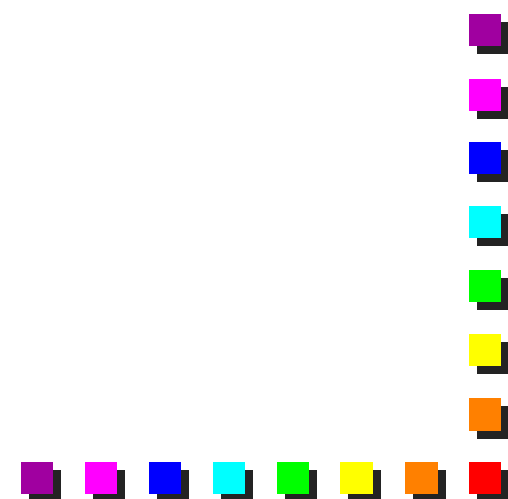
3e 9d 09 62

Your IPv6 address:

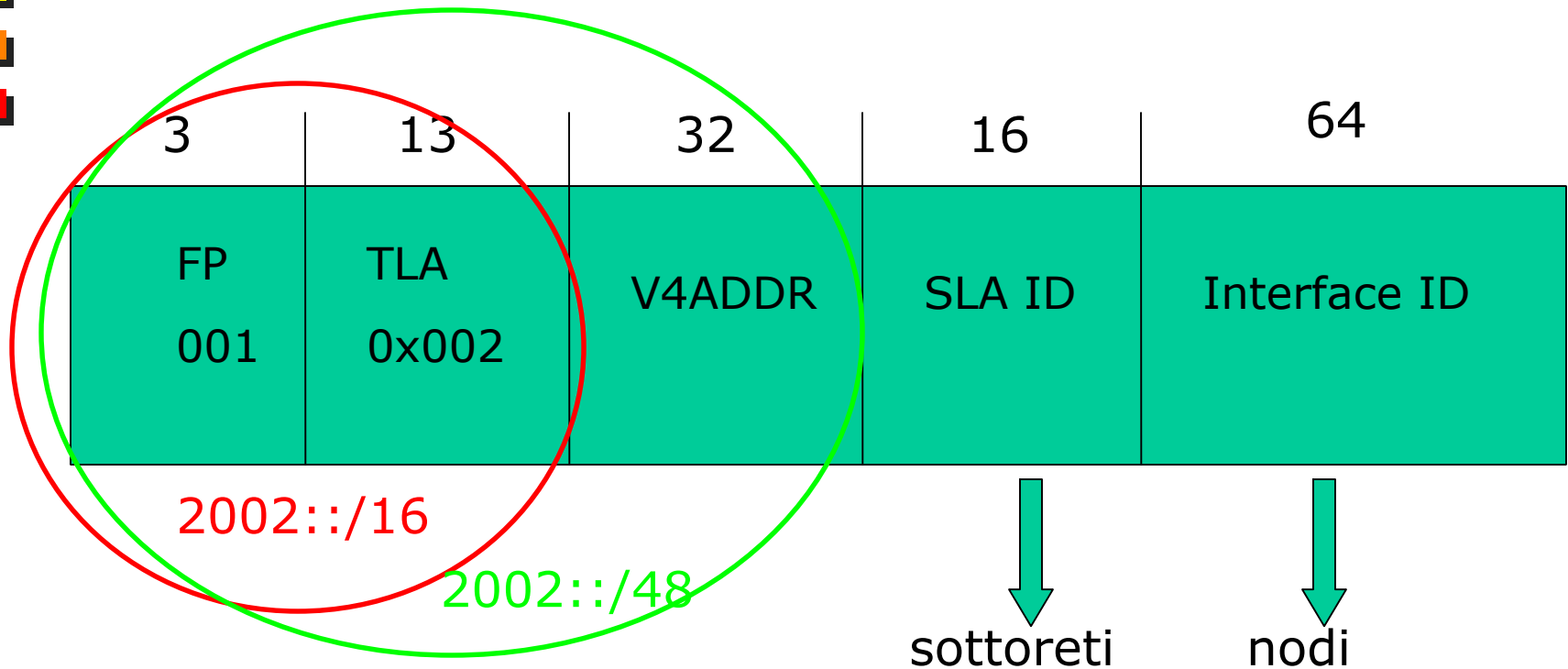
2002:3e9d:0962:0001::1

6to4
prefix

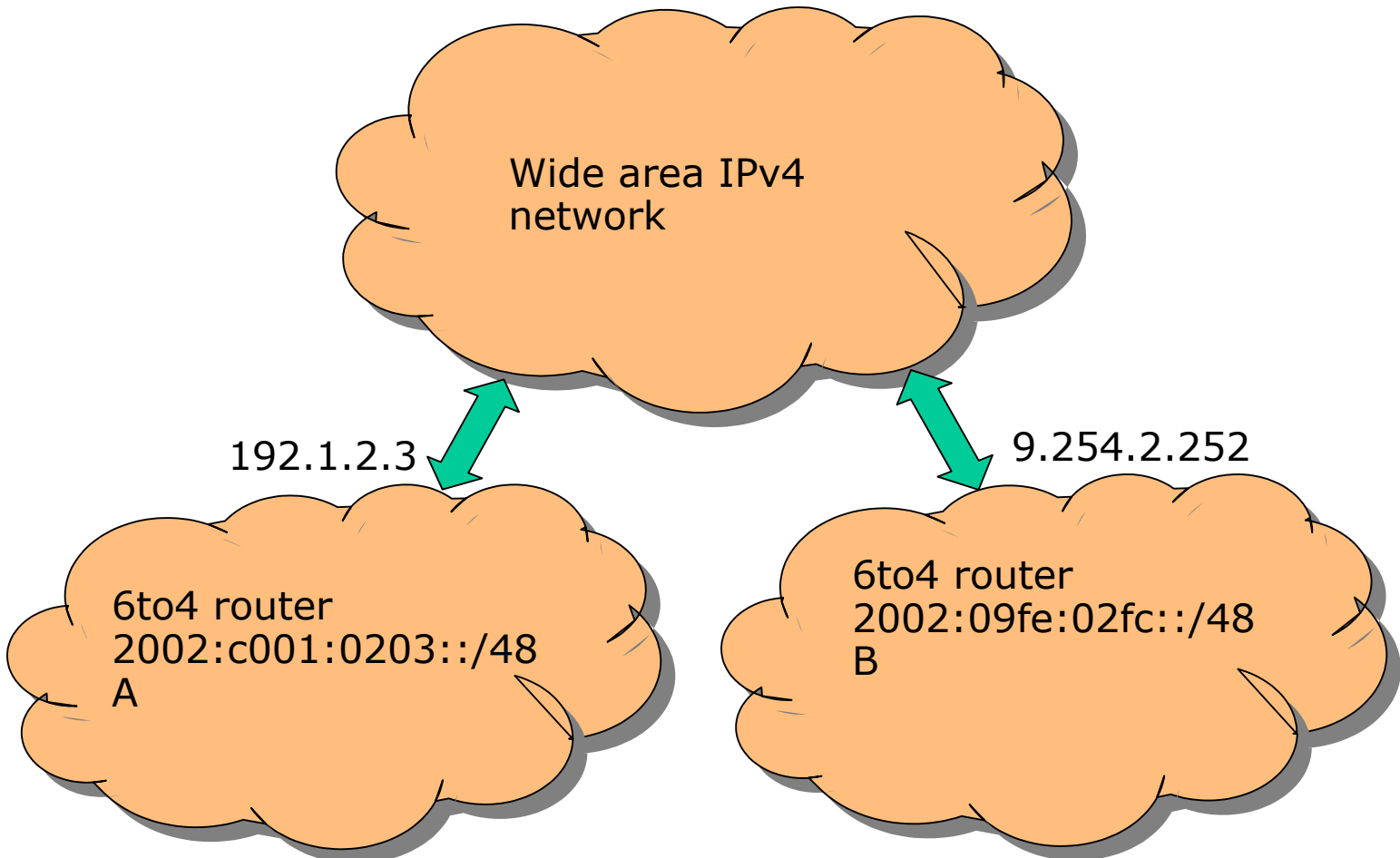
80 bit
address space



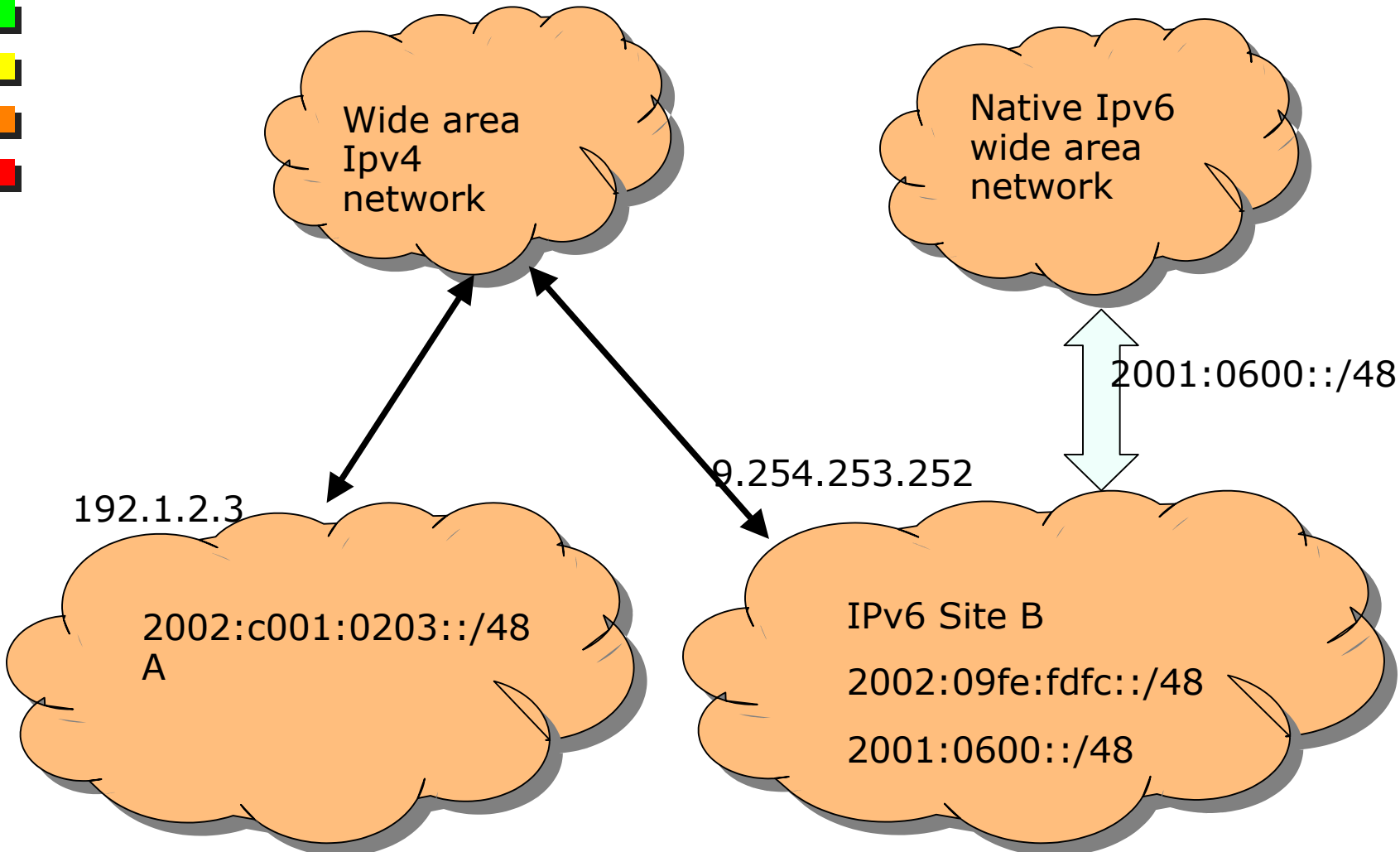
Struttura del prefisso 6to4



6to4: Scenario elementare



6to4: Scenario misto





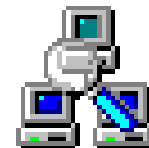
6to4

■ Utilizzo

- Normalmente in modalità host → 6to4 relay
- Definiti indirizzi IPv4 anycast
 - 131.107.33.60
 - 192.88.99.1
 - Necessaria la route di ritorno

■ Non compatibile con NAT (a meno di supporto specifico)

```
C:\> ipv6 rt
::/0 -> 3/2002:c058:6301::c058:6301 pref 1if+2147483647=2147483648 life 2h/30m, publish, no aging (manual)
::/0 -> 3/2002:836b:213c:1:e0:8f08:f020:8 pref 1if+1171=1172 life 2h/30m, publish, no aging (manual)
2002::/16 -> 3 pref 1if+1000=1001 life 2h/30m, publish, no aging (manual)
C:\> ping 2001:0610:0148:DEAD:0210:18FF:FE02:0E38 (www.6net.org)
```





ISATAP (intra-site automatic tunnel addressing protocol)

- **In 6to4, è necessaria la connettività nativa IPv6 intra-site**
 - Non prevede il concetto di router advertisement sul tunnel
- **ISATAP rimuove questo limite**
 - Il router può essere identificato attraverso una query al DNS
 - Record A, nome `_isatap.dominio.com`
 - È possibile interagire con il router attraverso router advertisement/solicitation conoscendone il suo indirizzo IPv4
- **Indirizzi**
 - `::0:5EFE:a.b.c.d`
 - 00-00-5E: OUI assegnato all'IANA
 - FE: tipo che identifica gli indirizzi "incorporati"
 - Ammette un prefisso /64 (come altre tipologie di tunnelling)

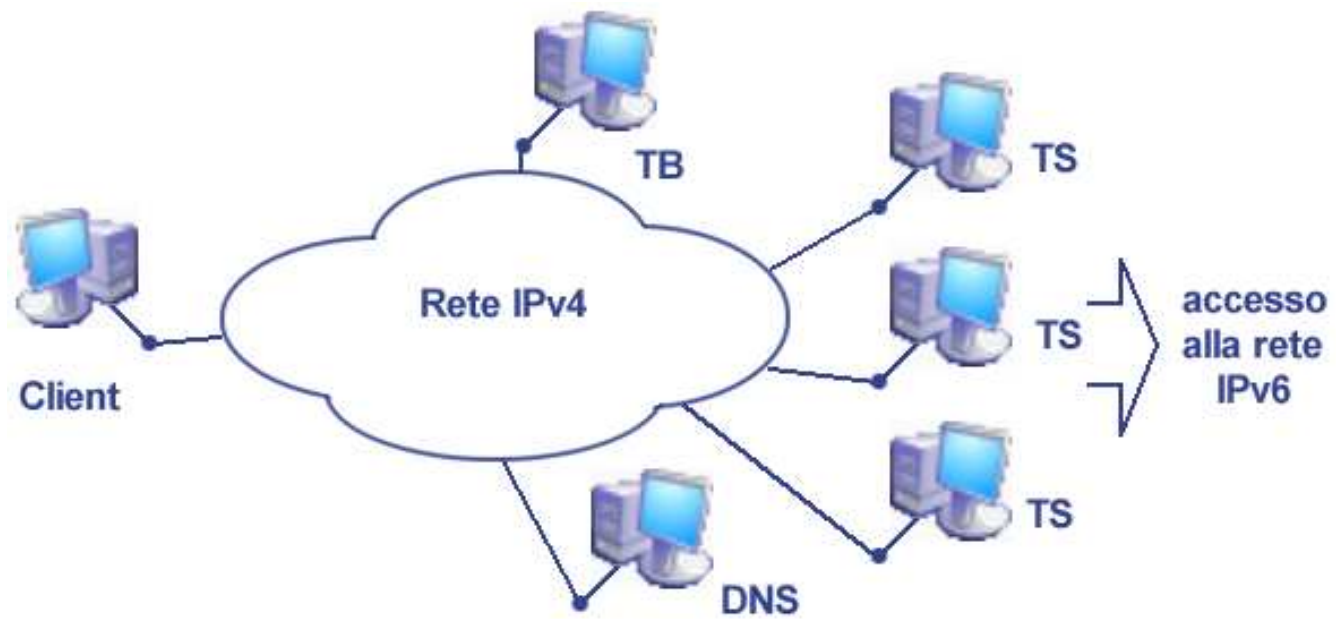


Teredo

- **Imbustamento di IPv6 in IP/UDP per superare il problema dei NAT**
- **Richiede un server esterno alla rete privata per la configurazione dell'indirizzo**



Tunnel broker



Tunnel broker: esempio

The screenshot shows a web browser window displaying the Hurricane Electric Internet Services website. The browser's address bar shows the URL `http://ipv6tb.he.net/conf.php`. The website has a logo for Hurricane Electric Internet Services and a navigation menu with the following items:

- Account Menu
 - +Main Page
 - Update Info
 - Logout
- Administration
 - +Tunnel Details
 - IPv4 End
 - /64 Allocation
 - BGP4+
 - Connectivity
 - Example Configs

The 'Example Configurations' section contains a dropdown menu with 'Linux-net-tools' selected and a 'Show Config' button. Below this, the following configurations are listed:

```
ifconfig sbl0 up
ifconfig sbl0 inet6 tunnel : 64.71.128.82
ifconfig sbl1 up
ifconfig sbl1 inet6 add 2001:470:1f:00:ffff:109:127
route -A inet6 add :0 dev sbl1
```

A note below the configurations states: "These are only example configurations and may be different depending on the version OS or tools you are using."

At the bottom of the page, there is contact information for Hurricane Electric, including their address (700 Mission Court, Fremont, CA 94539), phone number (Voice 510 580-4100), fax number (Fax 510 880-4151), and website (Comment? ip6@he.net). Copyright information for HE - Broker version 1.0 is also present.



Writing IPv6 applications

Fulvio Riso

NetGroup, Dipartimento di Automatica e Informatica

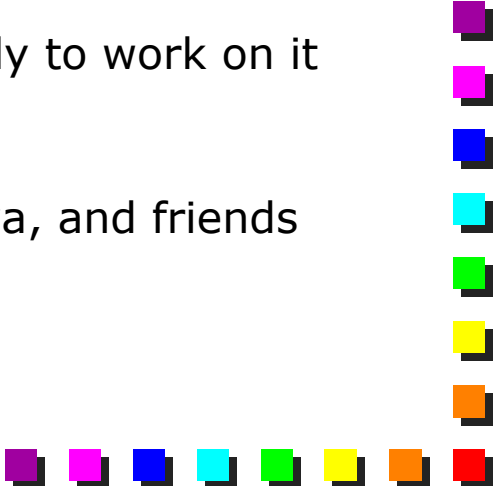
Politecnico di Torino

<http://staff.polito.it/risso/>





Why writing applications?

- **Most of the work, so far, focused on network-related issues**
 - Definition of the IPv6 protocol and related stuff
 - Network infrastructure
 - Operating System support
 - **A few people are currently working on the application side**
 - **We must avoid the problem of ATM**
 - Excellent technology, but no applications ready to work on it
 - **The socket interface**
 - We're network people, we don't like .NET, Java, and friends
 - C/C++ only if you want flexibility and speed
- 



The old programming code (BSD-style API)

```
#define PORT 2000                /* This definition is a number */

void server ()
{
    int Sock;                    /* Descriptor for the network socket */
    struct sockaddr_in SockAddr; /* Address of the server socket descr */

    if ( ( Sock = socket(AF_INET, SOCK_STREAM, 0)) < 0 ) {
        error("Server: cannot open socket.");
        return;
    }

    memset(& SockAddr, 0, sizeof(SockAddr));
    SockAddr.sin_family = AF_INET;
    SockAddr.sin_addr.s_addr= htonl(INADDR_ANY); /* all local addresses */
    SockAddr.sin_port = htons(PORT),           /* Convert to network byte order */

    if (bind(Sock, (struct sockaddr *) &SockAddr, sizeof(SockAddr)) < 0) {
        error("Server: bind failure");
        return;
    }

    /* ... */
}
```

The code must be duplicated for each address family

The new programming style (RFC 3493 API)

```
#define PORT "2000" /* This definition is a string */

void server ()
{
  int Sock; /* Descriptor for the network socket */
  struct addrinfo Hints, *AddrInfo; /* Helper structures */

  memset(&Hints, 0, sizeof(Hints));
  Hints.ai_family = AF_UNSPEC; /* or AF_INET / AF_INET6 */
  Hints.ai_socktype = SOCK_STREAM;
  Hints.ai_flags = AI_PASSIVE; /* ready to a bind() socket */

  if (getaddrinfo(NULL /* all local addr */, PORT, Hints, AddrInfo) != 0) {
    error("Server: cannot resolve Address / Port ");
    return;
  }

  // Open a socket with the correct address family for this address.
  if ((Sock=socket(AddrInfo->ai_family, AddrInfo->ai_socktype, AddrInfo->ai_protocol))<0){
    error("Server: cannot open socket.");
    return;
  }

  if (bind(Sock, AddrInfo->ai_addr, AddrInfo->ai_addrlen) < 0) {
    error("Server: bind failure");
    return;
  }
  /* ... */
}
```

Family-independent code

Fills some internal structures with family-independent data using literal / numeric host and port

Data returned by getaddrinfo() is used in a family-independent way



Modification to the system calls (1)

Parameter changes	<code>socket()</code> <code>bind()</code> , <code>connect()</code> , <code>accept()</code> <code>sendto()</code> , <code>recvfrom()</code> <code>setsockopt()</code> , <code>getsockopt()</code>	Example: <code>socket(AF_INET, ...</code> → <code>socket(AddrInfo->ai_family,...</code>
Unchanged	<code>recv()</code> , <code>send()</code> <code>listen()</code> , <code>select()</code> <code>shutdown()</code> <code>htonl</code> , <code>htons()</code> , <code>ntohl()</code> , <code>ntohs()</code>	
Replaced functions	<code>gethostbyaddr()</code> , <code>gethostbyname()</code> <code>gethostbyaddr()</code> , <code>getservbyport()</code>	→ <code>getaddrinfo()</code> , <code>freeaddrinfo()</code> → <code>getnameinfo()</code>
Obsolete functions	<code>inet_addr()</code> <code>inet_ntoa()</code>	→ <code>getaddrinfo()</code> → <code>getnameinfo()</code>



■ Other helper functions are usually unchanged

- `gethostname()`, `getsockname()`, `getpeername()`
- `getprotobyname()`, `getprotobynumber()`, `getservbyname()`, `getservbyport()`






Modification to the system calls (2)

■ Client side: fallback mechanism

- If a server does not respond to an IPv6 connection and it does have an IPv4 address, let's try to connect in IPv4
 - Problems due to the timeout
- This leads to a duplication of code and some additional control logic, which must be done by hand

■ Server side: dual-server capabilities


- If a server has both IPv4 and IPv6 addresses, it should accept both types of connections
 - Most of the OS requires this to be coded by hand
 - Notably exception: FreeBSD
 - Leads to duplicated code and some additional efforts in synchronization (we may have two waiting threads)
- 



Capabilities detection at compile time

- **Some application are distributed as a source code instead as a binaries**
- **Are IPv6 system calls and structures supported by the OS in which the compilation has to be done?**
 - Autoconf – automake try to automate the building process
 - This is not always possible with automatic tools
 - The programmer may have to add several alternatives for the code in order to be able to compile on older platforms
 - The building process may activate portions of code by means of some `#define`






Adding IPv6 support to old IPv4 applications (1)




■ We need to locate the code that needs to be changed

- “string search” to locate the system calls related to the `socket` interface
 - This is simple
- “visual inspection” for other parts of the code
 - This is not

■ System calls related to the `socket` interface

- Convert part of the code to become protocol independent
 - The most part of socket functions
 - Add special code for IPv6
 - Some functions (`getsockopt()`, `setsockopt()`) which behave differently in IPv4 and IPv6
- 






Adding IPv6 support to old IPv4 applications (2)



■ Other code

- Custom control used as input for an IPv4 address
 - Parsing of URLs
 - Several allowed strings
 - `http://130.192.16.81`
 - `http://truciolo.polito.it`
 - `http://2001:760:400:1:20b:dbff:fe14:50bb`
 - The ":" symbol is a "port delimiter" in IPv4, while it is the "address separator" in IPv6
 - `http://truciolo.polito.it:80`
 - `http://[2001:760:400:1:20b:dbff:fe14:50bb]:80`
 - Application-layer protocol
 - Is this protocol defining a field that carries IPv4 addresses (e.g. peer-to-peer applications)?
 - Difficult to locate
- 



Adding IPv6 support to old IPv4 applications: experimental results



Application	Lines of code of the <u>socket interface</u> that need to be changed	Other lines of code that need to be changed
FreeAMP Free MP3 player	88 (→ 59%) The code is not well organized	40 URL parsing
GNUcucleus Free peer-to-peer application	30 (→ 83%) The code related to the network is well defined into a C++ class	Undefined Protocol change

Far more than 50% of the code related to the socket interface must be changed.

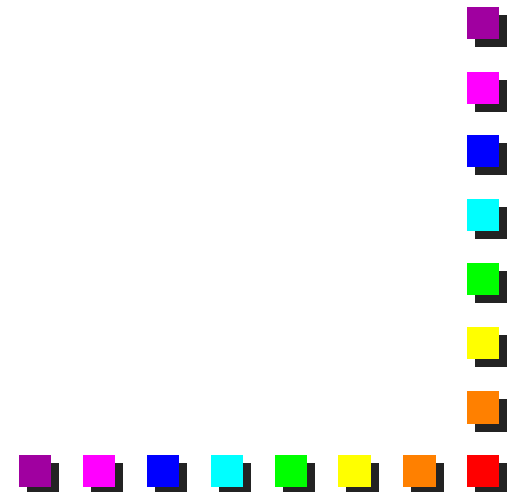
And, for the rest, who knows?





Creating IPv6-only application

- **The effort is slightly less than adding IPv6 support**
 - We can adopt some system calls which are “deprecated” but allow converting the code easier
 - We are not forced to add some code to support both the IPv4 and IPv6 case
- **It does not make sense**
 - Who does have an IPv6-only network?





Writing new applications with IPv4 and IPv6 support

- **For the most part, this is much easier than writing IPv4-only applications with the older BSD programming style**
 - `getaddrinfo()` and `getnameinfo()` are very handy
 - Code is smaller and easier to understand than the one written according to the old socket interface
- **Some code may be duplicated**
 - `getsockopt()`, `setsockopt()`
 - URL parsing



Platform compatibility

- **A network application may run on any network device**
 - Important to write portable network code in order to be able to run everything on other platforms
- **Most of the `socket` implementation is portable among platforms**
 - Some minor issues still remain



Platform differences (1)

Variables	Win32 socket() returns an unsigned integer	UNIX socket() returns an integer
Error functions	Win32 Default choice is WSAGetLastError() or GetLastError(); the gai_strerror() exists, but it is not thread safe and the previous functions should be used instead	UNIX gai_strerror() or the errno variable depending on the function
Error messages	Win32 Error codes are different from UNIX (although most of them have the same name)	UNIX Standard error codes
Interface management functions	Win32 Missing; there are some proprietary functions: GetNumberOfInterfaces(), GetInterfaceInfo() Additionally, there are the following: GetAdaptersInfo(), GetAdaptersAddresses(), GetAdapterIndex()	UNIX Standard if_nameindex() if_nametoindex() if_indextoname()
Address management	Win32 getaddrinfo() with some specific flags	UNIX inet_ntop(), inet_pton()

Platform differences (2)

Socket and files	Win32 Sockets are not 'standard' files read() and write() do not work ReadFile() and WriteFile() are working	UNIX The the same functions that are used to read and write files can be used with sockets
Initialization / cleanup	Win32 Required: WSStartup() and WSACleanup()	UNIX Not needed
Closing a socket	Win32 closesocket()	UNIX close()
Libraries	Win32 Include "winsock.h" and "ws2_32.lib"	UNIX Several files in addition to "sys/socket.h"; socketlib
Server Socket	Win32, Linux A server socket binds to a single address family; a server that has to accept IPv4/6 connections must open two server sockets	FreeBSD A single server socket can be created for both IPv4 and IPv6
Signals	Linux Generates a SIGPIPE signal whenever an error occurs when sending data on stream-oriented sockets	Win32, FreeBSD A "write" error on connected sockets generates only an error message



Platform differences (3)

- **Several differences**
 - Often “minor issues”, although “several minor issues” may become “a major problem”
- **A C wrapper or C++ class that implements a socket interface may be an excellent solution**
 - Not too much code to write
 - 1000 lines of code are enough
 - Just to hide some details
- **Win32: things are getting slightly worse when migrating from old sockets to new ones**





Applications that use a network abstraction

- **Several applications do not use the socket interface directly**
 - .NET or Java applications
 - MFC, QT or wxWindows-based applications
- **The environment (either the run-time or the library) will manage the network connections**
 - Usually, applications do not care about network protocols
 - These application should be able to run in IPv6 seamlessly
 - Obviously, the environment must support IPv6
 - Some problems (application-level protocols, GUI custom controls) may remain





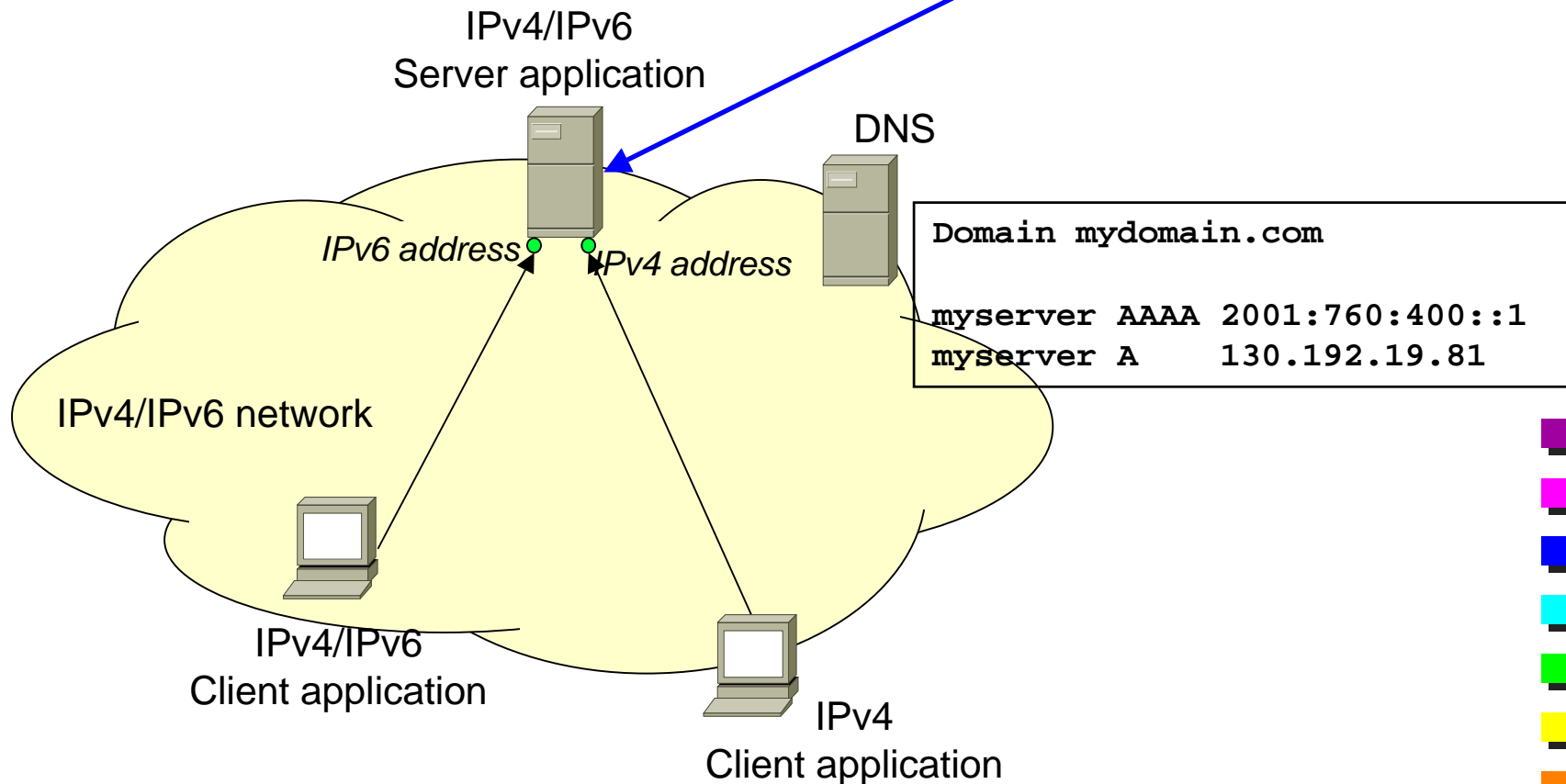
What about if the application already exists and the source code is not available?

- **Most of the applications do not make the source code available**
- **If we want to use IPv6 networks, we must have IPv6 applications**
 - Several projects around the world are still missing the point, creating IPv6 networks with no traffic on them
- **Some applications are ready**
 - Some examples on Win32:
 - Internet Explorer, Mozilla Firebird and Thunderbird
 - Internet Information Server (Windows Server 2003), Apache
 - ...

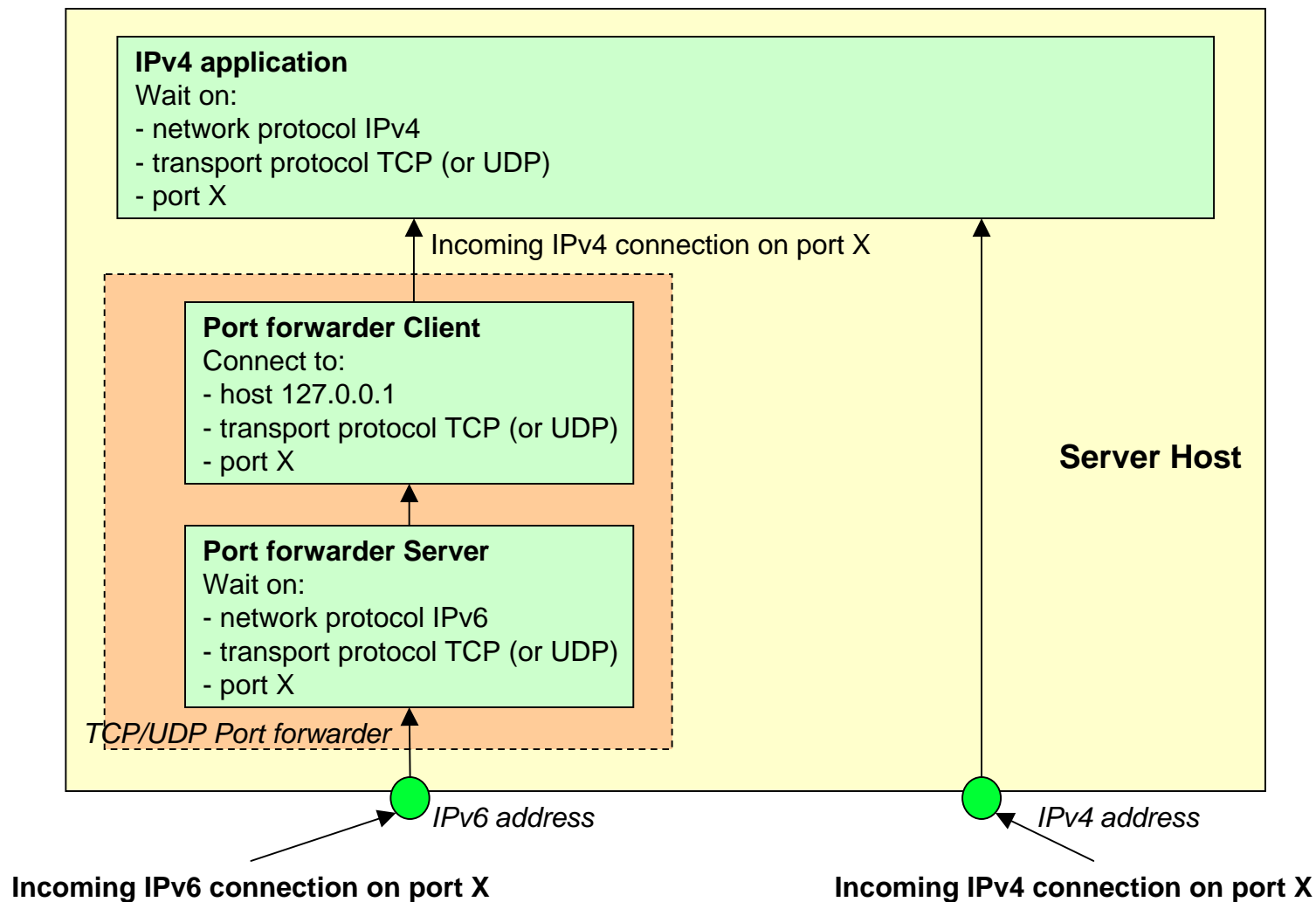
The most common deployment scenario

We must upgrade servers first

Avoids the timeout due to the IPv4 fallback

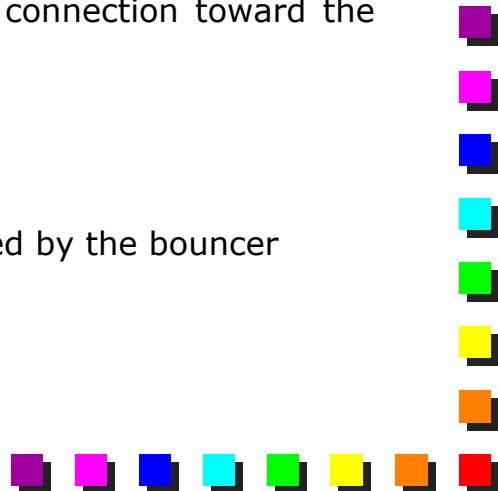


The TCP/UDP port forwarder (bouncer) (1)












The TCP/UDP port forwarder (bouncer) (2)

- ☺ **Very simple**
 - ☺ **Does not requires any modification to the legacy IPv4 application**
 - ☺ **Compatible with almost all TCP/UDP applications**
 - It does not undestand the application data (it is not a proxy)
 - ☺ **Hides the connecting network address**
 - Needed for statistics, logging, filtering
 - E.g. SMTP server which accepts only connections from local hosts
 - ☹ **Does not work with applications that do not follow the pure client-server model**
 - E.g. FTP in “active mode” because the server opens the data connection toward the client
 - ☹ **Fragmented packets**
 - IPv6 headers are bigger than IPv4 ones
 - Packets generated by the IPv4 server may need to be fragmented by the bouncer
- 



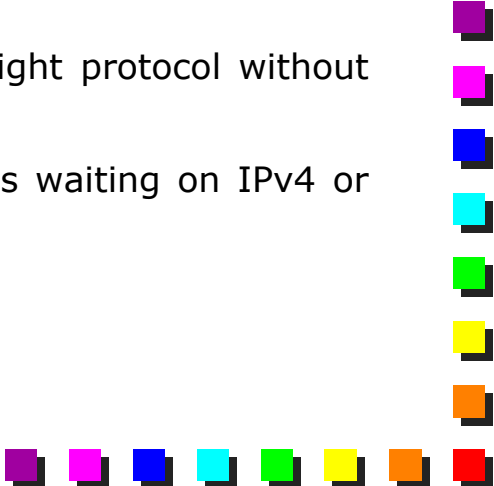
Bouncer and clients

- **In general, the deployment of a 'bouncer' is not limited to a server**
 - **However, it requires an explicit (and fixed) configuration**
 - of the address of the bouncer
 - of the server we want to connect to
 - **Using a bouncer to translate client connections**
 - A bouncer can be used to translate IPv4 requests coming from a legacy client into IPv6 streams
 - This approach works only if the client tries to connect always to the same server
 - E.g. DNS, POP, SMTP, IMAP, web browsing through a proxy
 - It does not work if the client can contact several servers
 - E.g. Standard web browsing, FTP, ...
 - **In any case, the bouncer is one of the best methods due to its simplicity**
- 
- 
- 
- 
- 
- 
- 



Other methods (1)

■ Most important approaches

- Bump In the Stack (BIS)
 - Packet-level translator embedded in the stack
 - IPv6 packets are translated into IPv4 ones and delivered to the application
 - It allows also initiating the connection
 - In case the contacted machine is IPv6, the DNS resolver returns a fake IPv4 address to the application; IPv4 packets directed to this host are intercepted and transformed in IPv6
 - Bump In the API (BIA)
 - More efficient approach: the socket library is modified in order to recognize if the host we want to contact is IPv4 or IPv6
 - The library generates the packets according to the right protocol without any need of translation
 - In case of a server, the library checks if the server is waiting on IPv4 or IPv6 socket and delivers the data accordingly
- 










Other methods (2)

■ Other approaches

- TCP/UDP Relay
 - Similar to a bouncer, but the address of the relay is returned by the DNS on an application-dependent way
- Network Address Translator – Protocol Translator
- SOCKS
- Application-level gateways







■ Problems

- Intrusiveness
 - content inspection
 - explicit client capabilities (e.g. proxy)
 - Not targeted to seamlessly migrating server applications
- 
- 
- 
- 
- 
- 
- 





Conclusions

- **Effort required to add IPv6 support to and old IPv4-oly application is not negligible**
 - Far more than 50% of the lines of code need to be changed
 - Hidden costs (input forms, application-dependent protocols, etc.)
 - **Creation of new IPv4 and IPv6 applications from scratch**
 - The socket interface is simpler than before
 - Some common issues:
 - Fallback: for clients
 - Dual-socket bind: for servers
 - **If we want to add IPv6 support to a closed source application**
 - The bouncer mechanism may be the best choice
- 
- 
- 
- 
- 
- 
- 