Best practices in IPv6-enabled networking software development

www.deepspace6.net

The IPv6 Protocol - 1

- New version of the Internet Protocol
- Devised by IETF to replace IPv4
- It solves all the problems of IPv4
  - Address space exhaustion
  - Explosion of routing tables
  - Mobility
  - Performance and scalability
- IPv6 is ready for mainstream adoption

www.deepspace6.net

The IPv6 Protocol - 2

- Enormous address space:
  - 340,282,366,920,938,463,463,374,607,431,768,211,456 (2^128) possible addresses, ~79*10^27 greater than IPv4
  - 665,570,793,348,866,943,898,599 addresses per square meter on Earth
- Aggregatable address space
- Mandatory IPSEC support
- Stateless address autoconfiguration
- Improved mobile networking
- Performance and scalability

www.deepspace6.net

The transition to IPv6

- All nodes (hosts, routers, firewalls, L3 switches, etc...) must be upgraded to support IPv6
- IPv6 connectivity must be provided to LANs and WANs
- All applications must be ported to IPv6
- IPv6 nodes and applications should preserve compatibility with IPv4
- Very difficult task!!!

www.deepspace6.net

The transition scenario - 1

- During the transition phase we'll have mixed IPv4 and IPv6 environments
- Many networks won't have native IPv6 connectivity
- Transition tools and mechanisms will be deployed to provide IPv6 connectivity to hosts and LANs (6TO4, NAT-PT, etc...)
- The network scenarios will be very complex
- Applications must be designed to work in all possible environments

www.deepspace6.net

The transition scenario - 2

- During the transition we'll have:
  - nodes with IPv4 connectivity but no IPv6 connectivity (or support)
  - nodes with IPv6 connectivity but no IPv4 connectivity (or support)
  - nodes with both IPv4 and IPv6 connectivity
- IPv4 connectivity may be preferred to IPv6 connectivity or viceversa (cost, reliability, etc...)
- There may be problems with DNS resolution

www.deepspace6.net

When will IPv4 die?

www.deepspace6.net

- Always too late ;-)
- There are areas in which the shortage of IPv4 addresses is really dramatic (especially Asia)
- However, IPv4 is not going to disappear soon:
  - http://potaroo.net/2003-08/ale.html
- NAT, private networks and Realm IP will extend lifetime of IPv4
- The transition to IPv6 will be probably very long

Applications design guidelines - 1

- Applications should support both IPv4 and IPv6
- Having two different applications, one for IPv4 and the other for IPv6, to handle the same service is annoying:
  - confusing for the users on client side
  - possible inconsistencies on server side
- Applications must work even if IPv6 (or IPv4) support is disabled

www.deepspace6.net

Applications design guidelines - 2

- There are cases in which an application must work only with a given protocol version (IPv4 or IPv6) or with both of them:
  - if a user is connecting to a service handled by two different server applications (one for IPv4 and one for IPv6)
  - if a developer wants to test IPv6 compliance of his application
  - if a developer wants to test an IPv6 transition tool or mechanism
  - a user knows that the service he is trying to connect to is available only via a specific protocol version and wants to speed up the procedure to establish connections to the server node

www.deepspace6.net

Applications design guidelines - 3

- IPv6-enabled applications should allow the expert user to choose if he wants to use IPv4, IPv6 or both:
  - -4 and -6 options for command line tools
  - entries in configuration files
- IPv6-enabled client applications should handle connectivity problems in a robust way (connection failover)
- Applications should handle scoped addresses if the system supports them

www.deepspace6.net

Applications design guidelines - 4

- Applications that perform DNS caching must:
  - cache both A (IPv4) and AAAA (IPv6) DNS records
  - discard cached records as soon as their lifetime expires
- Since many problems can arise from the interaction of the DNS caching pratice and the use of dynamic DNS, the expert user should be allowed to disable application-level DNS caching
- Applications should handle potential problems related to the malicious use of IPv4-mapped IPv6 addresses on the wire

www.deepspace6.net

Applications design guidelines - 5

- It may be desirable to keep also the old IPv4-only source code and choose at build time if the application must use the old IPv4-only code or the new IPv6-enabled code:
  - portability towards older systems which do not support IPv6 yet
  - to release an IPv6-enabled development version of the application while retaining production quality IPv4 support code

www.deepspace6.net

Applications design guidelines - 6

```
#ifdef ENABLE_IPv6
/* new IPv6-enabled code */
#else
/* old IPv4-only code */
#endif /* ENABLE_IPv6 */
```

- ENABLE_IPv6 is defined in one of the application header files at compile time by the user or the autoconfiguration process

www.deepspace6.net

The Extended BSD Socket API - 1

- IETF has developed an Extended BSD socket API to introduce support for the IPv6 protocol
- The old BSD socket API was incompatible with IPv6:
  - sockaddr_in and in_addr structs are inadequate to store IPv6 addresses
  - inet_ntoa(3) and inet_aton(3) are inadequate for the conversion of IPv6 addresses from network to ASCII string format and viceversa
  - gethostbyname(3) and gethostbyaddr(3) cannot handle scoped IPv6 addresses

www.deepspace6.net

The Extended BSD Socket API - 2

- The Extended BSD socket API defines the new address family AF_INET6 and the related protocol family PF_INET6
- It also introduces new data structures to store IPv6 addresses: in6_addr and sockaddr_in6
- To preserve backward compatibility PF_INET6 sockets do not support only IPv6 but also IPv4
- Connection to an IPv4 server application via a PF_INET6 socket is supported by means of IPv4-mapped addresses

www.deepspace6.net

The Extended BSD Socket API - 3

- IPv6-enabled server applications which bind to :: will also bind to 0.0.0.0 and will accept incoming connections via both IPv4 and IPv6
- Developers can change this default behaviour by setting the IPV6_V6ONLY socket option for PF_INET6 sockets
- IPV6_V6ONLY turns off IPv4 compatibility and makes the PF_INET6 socket support only IPv6

www.deepspace6.net

The Extended BSD Socket API - 4

- Two new functions for conversion of IP address formats:
  - inet_ntop(3) converts IP addresses from network to presentation (ASCII string) format
  - inet_pton(3) converts IP addresses from presentation (ASCII string) to network format
- inet_ntop and inet_pton support both IPv6 and IPv4 addresses and (unlike the old inet_ntoa and inet_aton) are also reentrant

www.deepspace6.net

The Extended BSD Socket API - 5

- Two new functions for DNS name resolution:
  - getaddrinfo translates a location and/or a service name and returns a set of socket addresses that can be used to connect or bind to the specified service
  - getnameinfo translates a socket address structure to a node and/or service name
- The results returned by getaddrinfo and getnameinfo are highly configurable

www.deepspace6.net

AF-independent apps - 1

- Porting applications to IPv6 by simply changing all the occurrences of AF_INET and sockaddr_in to AF_INET6 and sockaddr_in6 in most cases is not the best approach
- Hardcoding AF_INET6 and sockaddr_in6 in the sources:
  - undermines the portability of the code
  - prevents the application from working properly on dual stack systems where the IPv6 support is disabled
  - is complex and __VERY__ bug prone
- A preferrable solution is the adoption of an AF-independent development style
- The code becomes totally independent from the address family and we have a complete separation of IPv4 and IPv6 sockets

www.deepspace6.net

AF-independent apps - 2

- getaddrinfo and getnameinfo have been designed to be AF-independent
- They can provide name-to-address and address-to-name resolution for all the communication protocols supported by the system (even not based on DNS)
- If called with AF_UNSPEC, getaddrinfo performs translation for ALL protocols supported by the system
- Applications that use getaddrinfo and getnameinfo correctly will automatically take advantage of other protocol families and communication protocols supported by the target host

www.deepspace6.net

AF-independent apps - 3

- For generic name-to-address resolution, applications should call getaddrinfo and try __EACH__ returned socket addresses for connecting to the remote service
- Server applications should call getaddrinfo with the AI_PASSIVE flag and bind to __ALL__ the returned socket addresses
- Generic address-to-name resolution is straightforward
- To ease the development of AF-independent applications, the Extended BSD Socket API defines the sockaddr_storage structure

www.deepspace6.net

AF-independent apps - 4

- Writing AF-independent code is usually very easy (often easier than writing non AF-independent code!!!)
- There may be problems with IPV6_V6ONLY
- Not all the systems support that option in the same way:
  - some systems (NetBSD, OpenBSD, FreeBSD >= 5.0) turn IPV6_V6ONLY on by default
  - other systems turn IPV6_V6ONLY off by default, but let sysops to choose the default behaviour at run time (Linux >= 2.4.21 has the sysctl configuration option /proc/sys/net/ipv6/bindv6only)
  - older systems (Linux < 2.4.21) do __NOT__ support IPV6_V6ONLY

www.deepspace6.net

IPv4-only broken client code - 1

```
int connect_wrapper(const char *location, const char *service)
{
  int fd;
  struct sockaddr_in sin;
  socklen_t salen;
  unsigned short servnum = get_serv_num(service);

  fd = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);

  memset(&sin, 0, sizeof(sin));
  salen = sizeof(struct sockaddr_in);
  sin.sin_family = AF_INET;
  sin.sin_port = servnum;
```

www.deepspace6.net

IPv4-only broken client code - 2

```
 if (inet_aton(location, &sin.sin_addr) != 0) {
   if (connect(fd, (struct sockaddr *)&sin, salen) == 0) return fd;
 } else {
   int i;
   struct hostent *hp;

   hp = gethostbyname(hostname);
   memcpy(&sin.sin_addr, hp->h_addr, sizeof(struct in_addr));
   if (connect(fd, (struct sockaddr *)&sin, salen) == 0)
     return fd;
 }
 return -1;
}
```

www.deepspace6.net

IPv4-only broken client code - 3

```
unsigned short get_serv_num(const char *service)
{
 long int num; char *tail;
 unsigned short servnum;

 num = strtol(service, &tail, 10);
 if (*tail == 0) {
  servnum = htons((unsigned short)num);
 } else {
  sp = getservbyname(argv[1], NULL);
  servnum = (unsigned short)sp->s_port;
 }
 return servnum;
}
```

www.deepspace6.net

IPv4-only client code - 1

```
int connect_wrapper(const char *location, const char *service)
{
  int fd;
  struct sockaddr_in sin;
  socklen_t salen;
  unsigned short servnum = get_serv_num(service);

  fd = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);

  memset(&sin, 0, sizeof(sin));
  salen = sizeof(struct sockaddr_in);
  sin.sin_family = AF_INET;
  sin.sin_port = servnum;
```

www.deepspace6.net

IPv4-only client code - 2

```
 if (inet_aton(location, &sin.sin_addr) != 0) {
   if (connect(fd, (struct sockaddr *)&sin, salen) == 0) return fd;
 } else {
   int i; struct hostent *hp;

   hp = gethostbyname(hostname);

   for (i = 0; hp->h_addr_list[i] != NULL; ++i) {
     memcpy(&sin.sin_addr, hp->h_addr_list[i], sizeof(struct in_addr));
     if (connect(fd, (struct sockaddr *)&sin, salen) == 0)
       return fd;
   }
 }
 return -1;
}
```

www.deepspace6.net

IPv6-enabled client code - 1

```
int connect_wrapper(const char *location, const char *service)
{
  struct addrinfo hints, *res, *ptr;
  int fd, connected = 0;

  fd = socket(AF_INET6, SOCK_STREAM, IPPROTO_TCP);

  memset(&hints, 0, sizeof(hints));
  hints.ai_family = AF_INET6;
  hints.ai_socktype = SOCK_STREAM;
  hints.ai_protocol = IPPROTO_TCP;
  hints.ai_flags = AI_V4MAPPED | AI_ALL;

  getaddrinfo(location, service, &hints, &res);
```

www.deepspace6.net

IPv6-enabled client code - 1

```
 for (ptr = res; ptr != NULL; ptr = ptr->ai_next) {
   if (connect(fd, ptr->ai_addr, ptr->ai_addrlen) == 0) {
     connected = 1;
     break;
   }
 }
 freeaddrinfo(res);

 return (connected ? fd : -1);
}
```

www.deepspace6.net

AF-independent client code - 1

```
int connect_wrapper(const char *location, const char *service)
{
  struct addrinfo hints, *res, *ptr;
  socklen_t salen;
  int fd, connected = 0;

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

  getaddrinfo(location, service, &hints, &res);
```

www.deepspace6.net

AF-independent client code - 2

```
for (ptr = res; ptr != NULL; ptr = ptr->ai_next) {
  int fd = socket(ptr->ai_family, ptr->ai_socktype, ptr->ai_protocol);
  if (fd < 0) {
    if (unsupported_sock_error(errno)) continue;
    return -1; /* this is a fatal error */
  }
  if (ptr->ai_family == AF_INET6)
    setsockopt(fd, IPPROTO_IPV6, IPV6_V6ONLY, &on, sizeof(on));
  if (connect(fd, ptr->ai_addr, ptr->ai_addrlen) == 0) {
    connected = 1; break;
  } else close(fd);
}
freeaddrinfo(res);
return (connected ? fd : -1);
}
```

www.deepspace6.net

AF-independent client code - 3

```
int unsupported_sock_error(int err)
{
  return (err == EPFNOSUPPORT ||
          err == EAFNOSUPPORT ||
          err == EPROTONOSUPPORT ||
          err == ESOCKTNOSUPPORT ||
          err == ENOPROTOOPT) ?
          1: 0;
}
```

www.deepspace6.net

IPv4-only server code - 1

```
int bind_wrapper(const char *service, callback_t fn)
{
  int fd, ns;
  struct sockaddr_in sin;
  socklen_t salen;
  unsigned short servnum = get_serv_num(service);

  memset(&sin, 0, sizeof(sin));
  sin.sin_family = AF_INET;
  sin.sin_addr.s_addr = INADDR_ANY;
  sin.sin_port = servnum;

  fd = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
```

www.deepspace6.net

IPv4-only server code - 2

```
 salen = sizeof(struct sockaddr_in);
 bind(fd, (struct sockaddr *)&sin, salen);
 listen(fd, SOMAXCONN);

 for(;;) {
   if ((ns = accept(fd, NULL, NULL)) >= 0) fn(ns);
 }

 return 0;
}
```

www.deepspace6.net

IPv6-enabled server code - 1

```
int bind_wrapper(const char *service, callback_t fn)
{
  int fd, ns;
  struct sockaddr_in6 sin6;
  socklen_t salen;
  unsigned short servnum = get_serv_num(service);

  memset(&sin6, 0, sizeof(sin6));
  sin6.sin6_family = AF_INET6;
  sin6.sin6_addr = in6addr_any;
  sin6.sin6_port = servnum;

  fd = socket(AF_INET6, SOCK_STREAM, IPPROTO_TCP);
```

www.deepspace6.net

IPv6-enabled server code - 2

```
 salen = sizeof(struct sockaddr_in6);
 bind(fd, (struct sockaddr *)&sin6, salen);
 listen(fd, SOMAXCONN);

 for(;;) {
   if ((ns = accept(fd, NULL, NULL)) >= 0) fn(ns);
 }

 return 0;
}
```

www.deepspace6.net

AF-independent server code - 1

```
int bind_wrapper(const char *service, callback_t fn)
{
  struct addrinfo hints, *res, *ptr;
  socklen_t salen;
  int fd, on = 1;
  fd_set bound_sockets;

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

  getaddrinfo(NULL, service, &hints, &res);

  FD_ZERO(bound_sockets);  maxfd = -1;
```

www.deepspace6.net

AF-independent server code - 2

```
for (ptr = res; ptr != NULL; ptr = ptr->ai_next) {
  int fd = socket(ptr->ai_family, ptr->ai_socktype, ptr->ai_protocol);
  if (fd < 0) {
    if (unsupported_sock_error(errno)) continue;
    return -1; /* this is a fatal error */
  }
  if (ptr->ai_family == AF_INET6)
    setsockopt(fd, IPPROTO_IPV6, IPV6_V6ONLY, &on, sizeof(on));
  if (bind(fd, ptr->ai_addr, ptr->ai_addrlen) == 0) {
    FD_SET(bound_sockets); if (fd > max_fd) max_fd = fd;
  } else {
    close(fd); continue;
  }
  listen(fd, SOMAXCONN);
}
freeaddrinfo(res);
```

www.deepspace6.net

<mauro@deepspace6.net>

AF-independent server code - 3

```
for (;;) {
  int ns, tmpfd, tmpfd2 = max_fd;
  fd_set read_fds = bound_sockets;

  select(max_fd + 1, read_fdset, NULL, NULL, NULL);
  while(tmpfd2 >= 0) {
    tmpfd = tmpfd2;
    do { --tmpfd; } while (tmpfd >= 0 && !FD_ISSET(tmpfd, &read_fds));
    if (tmpfd >= 0 &&
         ((ns = accept(tmpfd, NULL, NULL)) >= 0)) fn(ns);
    tmpfd2 = tmpfd;
  }
 }
 return 0;
}
```

www.deepspace6.net

Autoconfiguration - 1

- Autoconfiguration systems split the building process in two steps: a configuration step and a build step
- The autoconfiguration script discovers IPv6 compliance of the system
- The code is compiled according to the information provided by the autoconfiguration system
- GNU autoconf is an invaluable help when writing portable IPv6-enabled software
- The autoconfiguration process for IPv6-enabled code is very complex

www.deepspace6.net

Autoconfiguration - 2

- The autoconfiguration script should check:
  - if the system supports the IPV6_V6ONLY option and if that option is set by default
  - if the getaddrinfo function supports all the flags defined by RFC3493
  - if the system (sockaddr_in6, getaddrinfo, getnameinfo) supports scoped IPv6 addresses
  - if the systems supports the Extended BSD socket API via a system library that the application must explicitly link

www.deepspace6.net

Autoconfiguration - 3

```
TYPE_STRUCT_SOCKADDR_STORAGE(,[AC_MSG_ERROR([...])])

ipv6=
AC_ARG_ENABLE(ipv6,
  AC_HELP_STRING([--disable-ipv6],[disable IPv6 support]),
  [case "${enable_ipv6}" in
  no)
    AC_MSG_NOTICE([Disabling IPv6 at user request])
    ipv6=no
    ;;
  *)
    ipv6=yes
    ;;
  esac],
  [ipv6=yes])
```

www.deepspace6.net

Autoconfiguration - 4

```
if test "X$ipv6" = "Xyes"; then
  TYPE_STRUCT_SOCKADDR_IN6(,[AC_MSG_NOTICE([...])
    ipv6=no
  ])
  MEMBER_SIN6_SCOPE_ID
fi

if test "X$ipv6" = "Xyes"; then
  PROTO_INET6(,[AC_MSG_NOTICE([...])
    ipv6=no
  ])
fi
if test "X$ipv6" = "Xyes"; then
  AC_DEFINE([ENABLE_IPV6], 1, [Define if IPv6 support is enabled.])
fi
```


www.deepspace6.net

Autoconfiguration - 5

```
AC_ARG_ENABLE(stack-guess,
  AC_HELP_STRING([--disable-stack-guess],[disable stack guess]),
  [case "${enable_stack_guess}" in
  yes)
    stack_guess=yes
    ;;
  no)
    stack_guess=no
    ;;
  *)
    AC_MSG_ERROR([...])
    ;;
  esac],
  [stack_guess=yes]
)
```

www.deepspace6.net

Autoconfiguration - 6

```
if test "X$stack_guess" != "Xno"; then
  IN6_GUESS_STACK
  NC6_CFLAGS="${NC6_CFLAGS} ${INET6_CFLAGS}"
  LIBS="${INET6_LIBS} ${LIBS}"
fi

AC_CHECK_FUNCS(
  [getaddrinfo freeaddrinfo gai_strerror getnameinfo],,
  AC_MSG_ERROR([...])
)
```

www.deepspace6.net

Autoconfiguration - 7

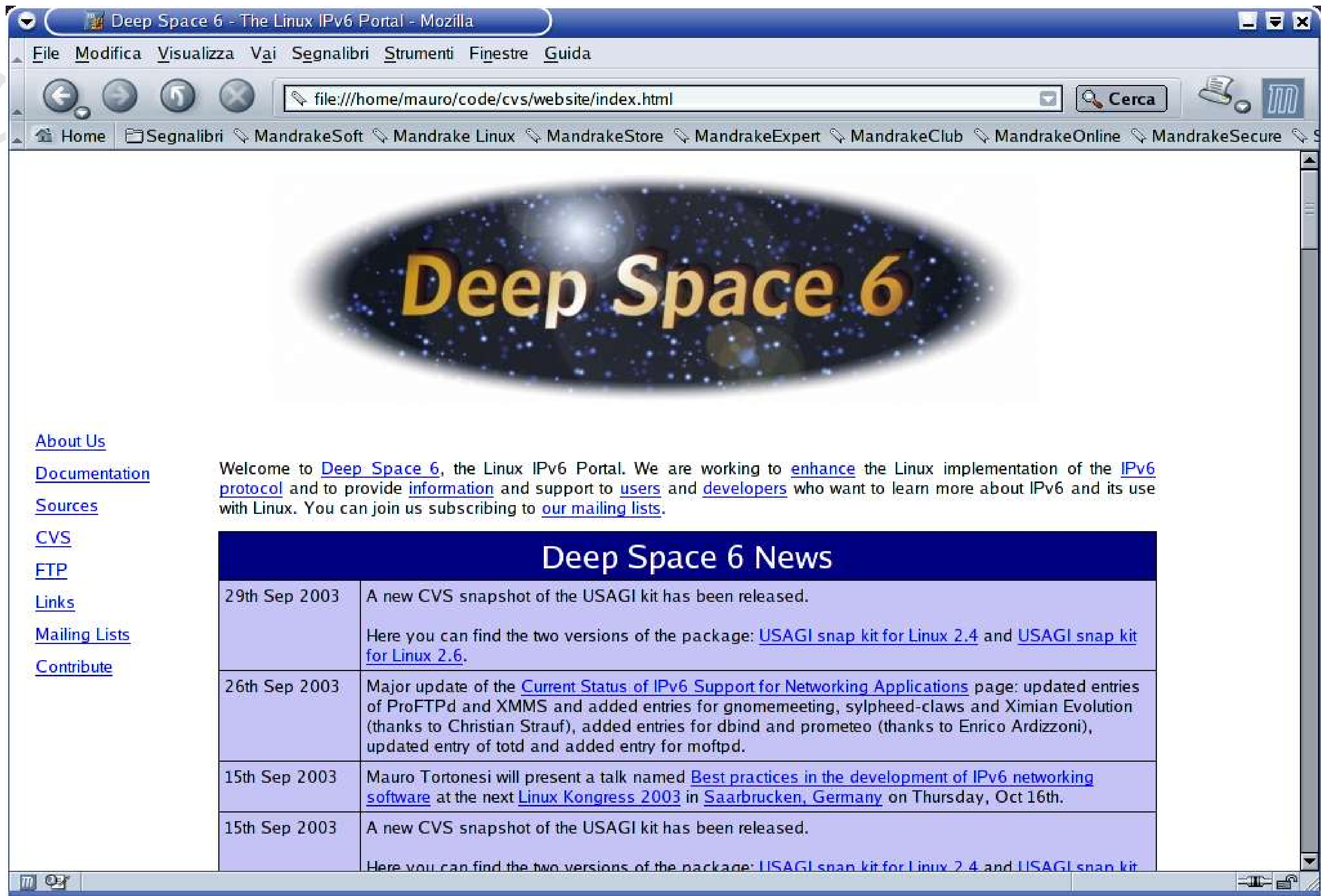
```
GETADDRINFO_AI_ADDRCONFIG(
  AC_DEFINE([HAVE_GETADDRINFO_AI_ADDRCONFIG], 1,
    [Define if the system headers support the AI_ADDRCONFIG flag.]))

GETADDRINFO_AI_V4MAPPED(
  AC_DEFINE([HAVE_GETADDRINFO_AI_V4MAPPED], 1,
    [Define if the system headers support the AI_V4MAPPED flag.]))

GETADDRINFO_AI_ALL(
  AC_DEFINE([HAVE_GETADDRINFO_AI_ALL], 1,
    [Define if the system headers support the AI_ALL flag.]))
```

www.deepspace6.net

Testing IPv6-enabled software

- Since IPv6 support is very different from one platform to the other, extensive testing of IPv6-enabled networking code is of great importance
- Developers should also use tools to verify the conformance of the Extended BSD socket API implementation of the target systems to the latest IETF standards
- libds6
  - testgetaddrinfo, testgetnameinfo, getaddrinfo
  - dumpsockaddr, dumpaddrinfo
- nc6

www.deepspace6.net



http://www.deepspace6.net

Conclusions

- Writing IPv6-enabled applications is very difficult as it requires more in-depth knowledge of the IP networking protocol
- If properly written, IPv6-enabled applications can easily support other communication protocols
- IPv6 is going to mainstream, so begin porting your networking applications IMMEDIATELY!!!

www.deepspace6.net

Suggestions

- Visit http://www.deepspace6.net, read the documentation and subscribe to the ds6 and ds6-devel mailing lists
- Upgrade to a Linux kernel release >= 2.4.21
- Install libinet6 from USAGI project (http://www.linux-ipv6.org) on your host
- Take a look at example-ipv6-package
- Test and debug your applications with nc6 and libds6

www.deepspace6.net

Acknowledgements

- Dr. Peter Bieringer and Simone Piunno (co-founders of Deep Space 6)
- Chris Leisham and Filippo Natali (co-authors of nc6)
- Jun-ichiro “itojun” Hagino
- Prof. Cesare Stefanelli and Dr. Michele Balestra

www.deepspace6.net

Questions?

www.deepspace6.net