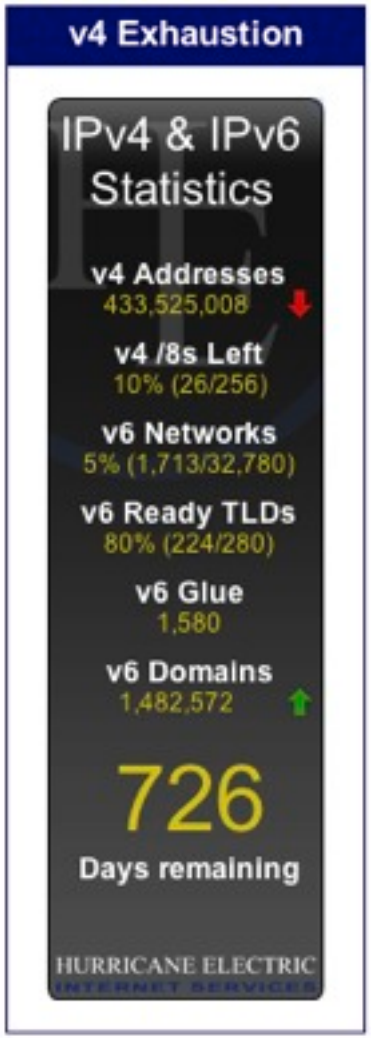
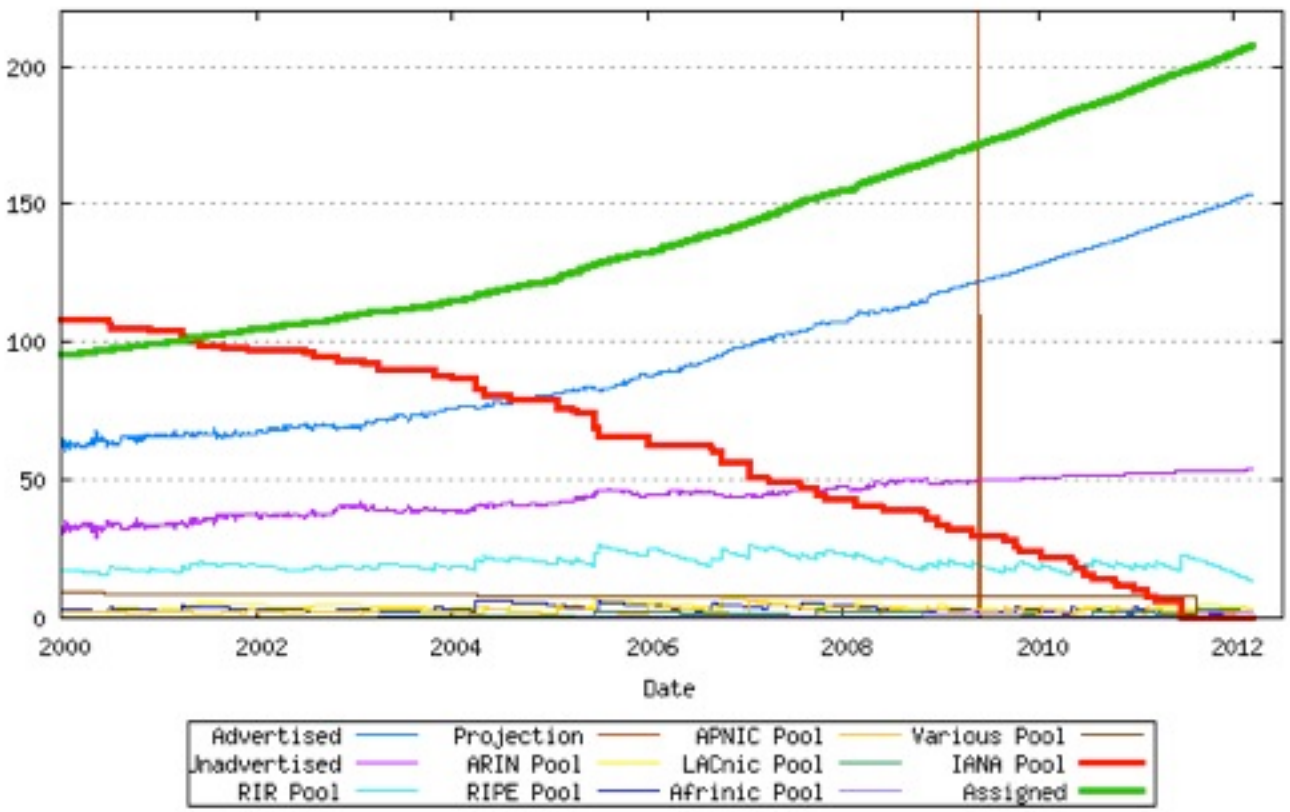
HURRICANE ELECTRIC
INTERNET SERVICES

Porting IPv4 applications to IPv4/v6 dual stack

Owen DeLong
owend@he.net

Why is this important?





Apologies in advance for the Text-Fest

Text Text Text Text Text
Text
Text Text Text Text Text Text
Text Text Text Text
Text Text Text Text Text Text
Text Text

Summary of Porting Steps

- Sample code available at: http://owend.corp.he.net/ipv6/sample_application/
- Change variable names when changing types.(e.g. `dest_sin` -> `dest6_sin`)
- Look for old variable name(s) as markers for code to be updated.
- Compile->Repair->Recompile (iterative)
- Test->Debug->Retest (iterative)

General Changes (IPv4 to dual stack)

- `AF_INET` -> `AF_INET6`
- `sockaddr_in` -> `sockaddr_in6`, `sockaddr_storage` (Generic storage type)
- Same structure members, similar constants, mostly just the address size changes.
- If necessary, check address scoping (link local vs. global and interface scope for link locals)

Some possible gotchas not covered in the examples

- IP Addresses in logs
- IP Addresses stored in databases
- Parsing or other routines that need to deal with IP addresses (use library functions if at all possible)

C porting example

- Refer to the Source Code Examples
- v4_* are IPv4 only code
- v6_* are same applications ported to dual stack
- By renaming affected variables, most calls that need to be updated are automatically flagged (markers).

Migrating the server (C)

- The easy part:
  - Additional include `<netinet/in.h>`
  - Rename `sockfd` to `sockfd6` (optional)
  - Change `sockaddr_in` to `sockaddr_in6` (new struct) and rename as `dest_sin6` (marker)
  - update initializations of `dest_sin6` (new members)
  - change args in `socket()` call
  - socket related error messages (variable renaming)
  - update `setsockopt()`, `bind()`, `listen()` (variable renaming)

Migrating the server (C)

- The easy part (cont'd):
  - update preparation for `select()` (variable renaming)
  - update initialization of `socklen`
  - update call to accept (renaming)
  - Other miscellaneous variable renaming
  - `inet_ntoa()` -> `inet_ntop()`

Migrating the client (C)

- Similar to porting the server...
- The less easy parts
  - Need a helper function (`get_ip_str()`) to front `inet_ntop()` for different possible return structures from `getaddrinfo()`
  - replacing `gethostbyname()`/`getservbyname()` with `getaddrinfo()` requires some effort. The `getaddrinfo()` process is actually MUCH cleaner. (newer v4-only code may already use `getaddrinfo()`)
  - Remember to free memory allocated by `getaddrinfo()`

Warning: Eye Charts ahead

- A handout with side-by side diffs of the source code is available at http://owend.corp.he.net/ipv6/

I

FY

OUCAN

READT

HISYO

UREYE

SAREBET

TERTHANMINE

More detail on the hard parts (C)

- IPv4 Only (gethostbyname):

```
/* Try as host name */
if (host_ent = gethostbyname(argv[1])) {
  dest_sin.sin_family = host_ent->h_addrtype;
  if (host_ent->h_length > sizeof( dest_sin.sin_addr)) {
    fprintf(stderr, "%s: address length wrong.\n", argv[0]);
    exit(2);
  }
  memcpy(&dest_sin.sin_addr, host_ent->h_addr_list[0], host_ent->h_length);
/* Try as IP address */
} else {
  if(dest_sin.sin_addr.s_addr = inet_addr(argv[1])) {
    fprintf(stderr, "%s: cannot find address for '%s'.\n", argv[0], argv[1]);
    exit(2);
}
```


More detail on the hard parts (C) (cont'd)

- IPv4 Only (getservbyname):

```
/* Get service information */
if ((srvp = getservbyname("demo", "tcp")) == 0) {
  fprintf(stderr, "%s: cannot find port number for demo service.\n", argv[0]);
  exit(3);
} else {
  dest_sin.sin_port = srvp->s_port;
}
```


More detail on the hard parts (C) (cont'd)

- IPv4/v6 Dual Stack (`getaddrinfo()`) does both:
  - Gets both Service and Host information at once.
  - Returns a dynamically allocated linked list
  - Don't forget to free the list when you no longer need it.

```
/* Get address info for specified host and demo service */
memset(&addrinfo, 0, sizeof(addrinfo));
addrinfo.ai_family=PF_UNSPEC;
addrinfo.ai_socktype=SOCK_STREAM;
addrinfo.ai_protocol=IPPROTO_TCP;
if (rval = getaddrinfo(argv[1], "demo", &addrinfo, &res) != 0) {
  fprintf(stderr, "%s: Failed to resolve address information.\n", argv[0]);
  exit(2);
}
```


Trying to connect -- Differences (C)

- IPv4 Only (see example source code):

```
for(addrlist = host_ent->h_addr_list; *addrlist != NULL; addrlist++)
{
  memcpy((caddr_t)&dest_sin.sin_addr, (caddr_t)*addrlist, sizeof(dest_sin.sin_addr));
  if ((sockfd = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP)) < 0)
  {
    fprintf(stderr, "%s: Could not create socket.\n", argv[0]);
    exit(4);
  }
  if (connect(sockfd, (struct sockaddr *)&dest_sin, sizeof(dest_sin)) < 0)
  {
    e_save = errno;
    (void) close(sockfd);
    errno = e_save;
    fprintf(stderr, "%s: Failed attempt to %s.\n", argv[0],
                              inet_ntoa(dest_sin.sin_addr));
    perror("Socket error");
  } else {
    snprintf(s, BUFLEN, "%s: Succeeded to %s (%d).", argv[0],
                              inet_ntoa(dest_sin.sin_addr), dest_sin.sin_addr);
    debug(5, argv[0], s);
    success++;
    break;
  }
}
if (success == 0)
{
  fprintf(stderr, "%s: Failed to connect to %s.\n", argv[0], argv[1]);
  exit(5);
}
```


Trying to connect -- Differences (C)

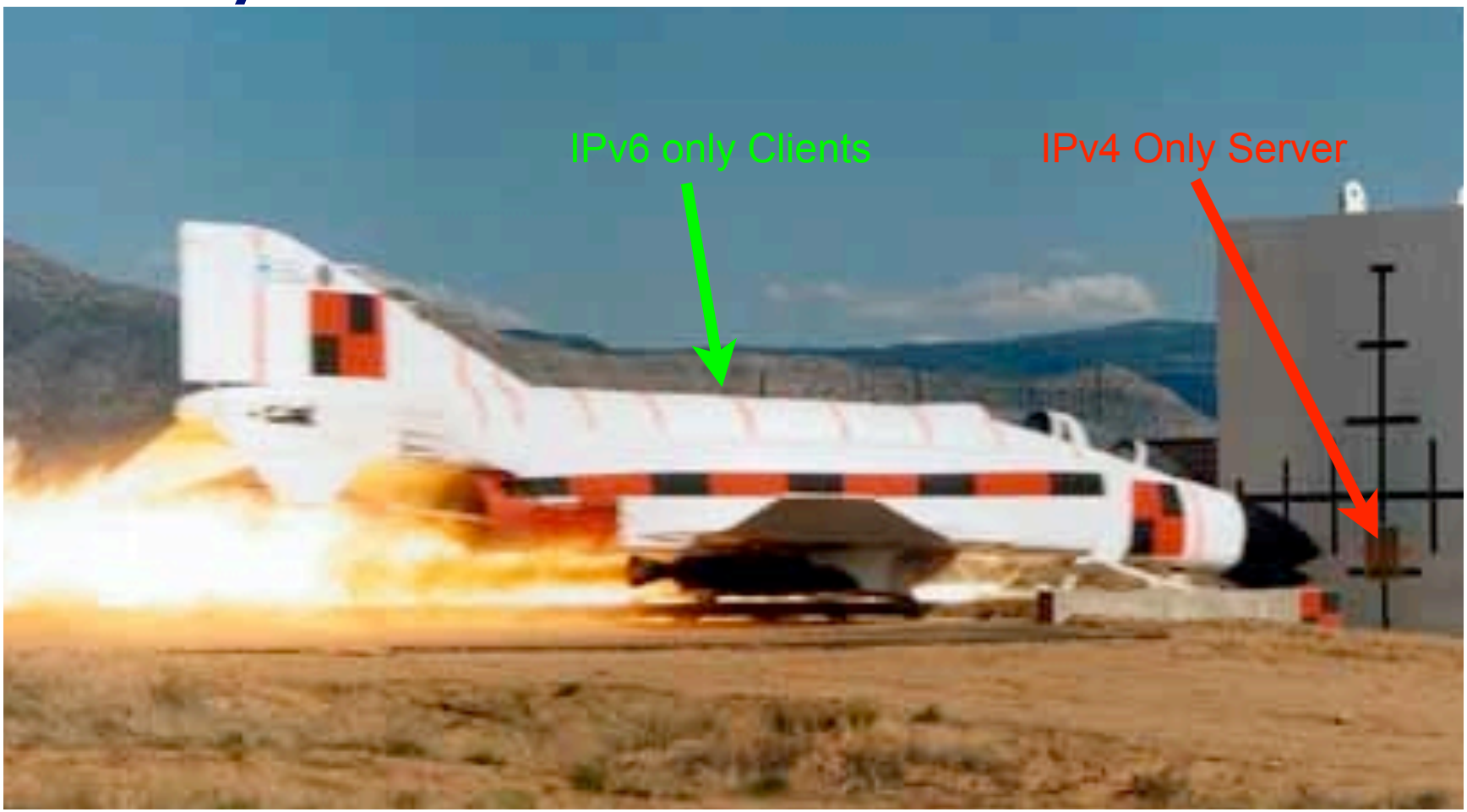
- The new way (a bit easier) (see example code):

```
for (r=res; r; r = r->ai_next) {
  sockfd6 = socket(r->ai_family, r->ai_socktype, r->ai_protocol);
  if (connect(sockfd6, r->ai_addr, r->ai_addrlen) < 0)
  {
    e_save = errno;
    (void) close(sockfd6);
    errno = e_save;
    fprintf(stderr, "%s: Failed attempt to %s.\n", argv[0],
                              get_ip_str((struct sockaddr *)r->ai_addr, buf, BUFLEN));
    perror("Socket error");
  } else {
    snprintf(s, BUFLEN, "%s: Succeeded to %s.", argv[0],
                              get_ip_str((struct sockaddr *)r->ai_addr, buf, BUFLEN));
    debug(5, argv[0], s);
    success++;
    break;
  }
}
if (success == 0)
{
  fprintf(stderr, "%s: Failed to connect to %s.\n", argv[0], argv[1]);
  freeaddrinfo(res);
  exit(5);
}
printf("%s: Successfully connected to %s at %s on FD %d.\n", argv[0], argv[1],
              get_ip_str((struct sockaddr *)r->ai_addr, buf, BUFLEN),
              sockfd6);
```


Migrating the client (C)

- The easy parts
  - Use the same variable name flagging method as with server.
  - Mostly update the same structure names and calls, flagged the same way.
  - `getaddrinfo()` will automatically return the AAAA and A records, so, v6/v4 is automatic with one codebase.
  - `inet_ntop()` needs a helper function (see `get_ip_str()` in the example code)

What happens if we aren't ready?



PERL Porting Example

- Refer to the Source Code Examples
- v4_* are IPv4 only code
- v6_* are same applications ported to dual stack
- Did not rename most variables in this example. (Small codebase, not as important)

Server Differences (PERL)

- Add `Socket6` to the modules "`used`" (you still need `Socket`, too). PERL documentation for `Socket6` is minimal and examples limited.
- Gut and replace `get*byname` calls (more on this next slide)
- Change protocol and address families in `socket()` and `bind()` calls.
- Minor changes to processing incoming connections (mostly related to name/address display).

Server Differences (PERL) (cont.)

- Biggest change is conversion from `get*byname()` to `getaddrinfo()`
- Similar changes to C port (same underlying library changes)
- C `getaddrinfo()` returns linked list. PERL `getaddrinfo()` returns straight list (multiple of 5 elements, each 5 elements is a list entry).
- Gotcha on `getaddrinfo()` -- passing in `in6addr_any` does not return `in6addr_any`.

Code Changes (PERL)

- Old way (getservbyname()):

```
my $tcp = getprotobyname('tcp');
my $tcpport = getservbyname('demo', 'tcp');
```

- New way (getaddrinfo()):

```
my ($fam, $stype, $tcp, $saddr, $cname);
my @res = getaddrinfo(in6addr_any(), 'demo', AF_UNSPEC, SOCK_STREAM);
my ($tcpport, $addr);
die "$0: Could not get protocol information" unless @res;
# THis is ugly, but, seems to be necessary to bind to IPv6.
$fam = 0;
($fam, $stype, $tcp, $saddr, $cname, @res) = @res while $fam != AF_INET6;
die "$0: IPv6 unsupported on this system.\n" unless ($fam == AF_INET6);
($tcpport, $addr) = unpack_sockaddr_in6($saddr);
$addr = in6addr_any();
$saddr = pack_sockaddr_in6($tcpport, $addr);
```


Code Changes (PERL) (Cont.)

- IPv4 only:

```
socket(TCPServer, PF_INET, SOCK_STREAM, $tcp) ||
                    die "$0: Could not create socket: $!";
bind(TCPServer, sockaddr_in($tcpport, INADDR_ANY)) ||
                    die "$0: Bind failed: $!";
```

- IPv4/v6 Dual Stack:

```
socket(TCPServer, PF_INET6, SOCK_STREAM, $tcp) ||
                    die "$0: Could not create socket: $!";
bind(TCPServer, $saddr) || die "$0: Bind failed: $!";
```


Code Changes (PERL) (Cont.)

- IPv4 only:

```
my ($port, $iaddr) = sockaddr_in($paddr);
my $name = gethostbyaddr($iaddr, AF_INET);
debug(5, "TCP Connection from $name [".inet_ntoa($iaddr)."] at port $port.\n");
$CLIENTS{$CLIENT} = inet_ntoa($iaddr)."/".$port;
```

- IPv4/v6 Dual Stack:

```
my ($port, $iaddr) = unpack_sockaddr_in6($paddr);
my ($name, $svc)  = getnameinfo($paddr);
debug(5, "TCP Connection from $name [".inet_ntop(AF_INET6, $iaddr).
                        "] at port $port.\n");
$CLIENTS{$CLIENT} = inet_ntop(AF_INET6, $iaddr)."/".$port;
```


PERL Client Migration

- Similar changes to C client
- Add module Socket6 (just like the server)
- Rearrange the address resolution stuff for `getaddrinfo()`
- Add some handling for `AF_INET6` to the connection loop
- Convert `inet_ntoa()` to `inet_ntop()` calls.
- Handle Protocol Family for `socket()` call

Code Changes (PERL) (Cont.)

- IPv4 only:

```
my $tcp = getprotobyname('tcp');
my $tcpport = getservbyname($port, 'tcp');
...
my ($name, $aliases, $addrtype, $length, @addrs) = gethostbyname($server);
die("$0: gethostbyname error: $!\n") if ($?);
die("Invalid server specified.\n") unless(@addrs);
socket(SOCKFD, PF_INET, SOCK_STREAM, $tcp) || die "Couldn't create socket: $!\n";
SOCKFD->autoflush(1);
```

- IPv4/v6 Dual Stack:

```
my @res = getaddrinfo($server, 'demo', AF_UNSPEC, SOCK_STREAM, 'tcp');
die("Could not resolve $server or service demo: ".$res[0].".\n")
                        unless(scalar(@res) >= 5);
```

- Note: In IPv4, socket can be recycled for multiple connects. IPv4/v6 Dual Stack, not so due to possible family change (PF_INET/PF_INET6)

Code Changes (PERL) (Cont.)

- IPv4 only:

```
while (@addrs)
{
  $a = shift(@addrs);
  print "Trying host ", inet_ntoa($a),".\n";
  $dest_sin = sockaddr_in($tcpport, $a);
  last if(connect(SOCKFD, $dest_sin));
  print "Failed to connect to ", inet_ntoa($a), ".\n";
  $dest_sin = -1;
}
```

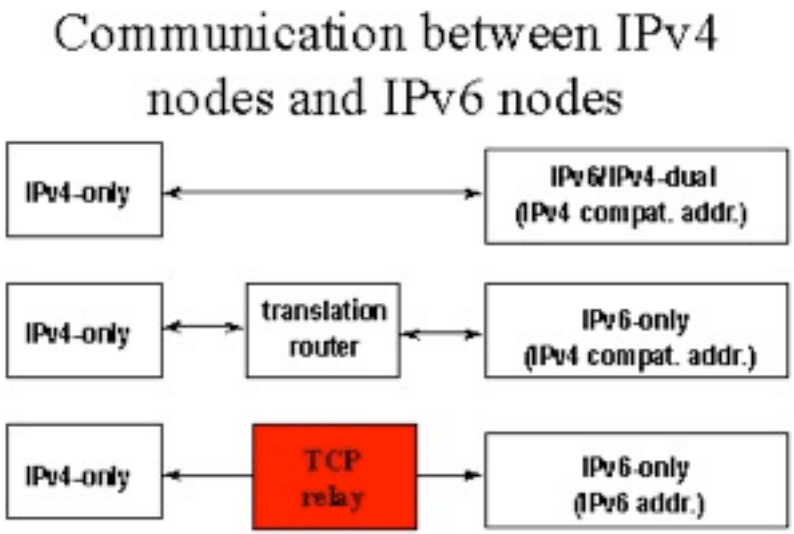

Code Changes (PERL) (Cont.)

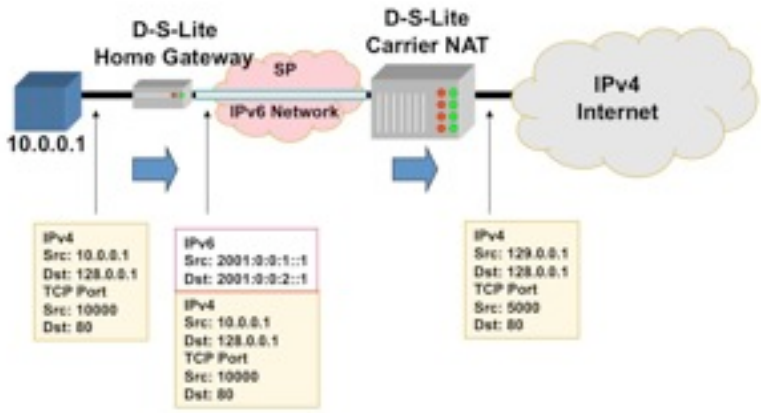
- IPv4/v6 Dual Stack:

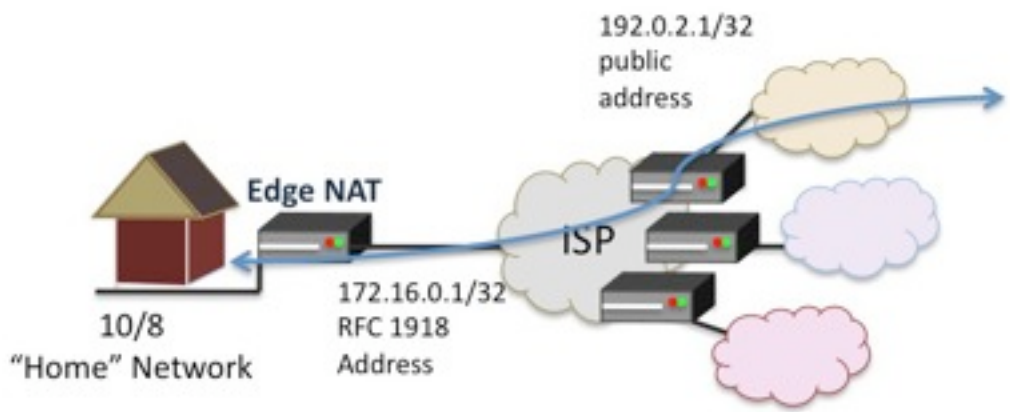
```
my ($fam, $stype, $proto, $saddr, $cname);
my ($port, $addr);
while (scalar(@res) >= 5)
{
  ($fam, $stype, $proto, $saddr, $cname, @res) = @res;
  next unless($saddr);
  $cname = $server unless $cname;
  print "Unpacking $cname...";
  ($port, $addr) = ($fam == AF_INET6) ?
                unpack_sockaddr_in6($saddr) : sockaddr_in($saddr);
  $addr = inet_ntop($fam, $addr);
  print "Trying host $cname ($addr) port $port.\n";
  my $PF = ($fam == AF_INET6) ? PF_INET6 : PF_INET;
  socket(SOCKFD, $PF, SOCK_STREAM, $proto) || die "Couldn't create socket: $!\n";
  SOCKFD->autoflush(1);
  last if(connect(SOCKFD, $saddr));
  close SOCKFD;
  print "Failed to connect to $cname ($addr): $!.\n";
  $saddr = -1;
}
```

- This isn't as bad as it looks. Need better libraries?

No, really, what happens?







Python Porting Example

- Refer to the Source Code Examples
- v4_* are IPv4 only code
- v6_* are same applications ported to dual stack
- Did not rename most variables in this example. (Small codebase, not as important)

Server Differences (Python)

- Gut and replace `get*byname()` calls (more on this next slide)
- Replace default fatal error for single attempt at binding with iterative loop to handle multiple address families
- Minor changes to processing incoming connections (4-tuple instead of 2).

Code Changes (Python)

- Old way (getservbyname()):

```
tcp = socket.getprotobyname('tcp')
tcpport = socket.getservbyname(port, 'tcp')
```

- New way (getaddrinfo()):

```
try:
    res = socket.getaddrinfo(None, "demo", socket.AF_UNSPEC, \
      socket.SOCK_STREAM, 0, socket.AI_PASSIVE)
except socket.gaierror, (errno, msg):
    print >> sys.stderr, "%s: failed with error %s." \
      % (prog, msg)
    sys.exit(1)
```


Listening (Python)

- Old way:

```
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.setblocking(0)
s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
s.bind(('', tcpport))
s.listen(socket.SOMAXCONN)
```

- New way:

```
for (fam, stype, proto, cname, saddr) in res:
    if (fam is not socket.AF_INET6): continue
    (addr, tcpport, flow, scope) = saddr
    try:
        s = socket.socket(fam, stype, proto)
    except socket.error, (errno, msg):
        s = None
        continue
    try:
        s.setblocking(0)
        s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        s.bind(('', tcpport))
        s.listen(socket.SOMAXCONN)
    except socket.error, (errno, msg):
        s.close()
        s = None
        continue
    break
```


Code Changes (Python) (Cont.)

- Old way:

```
(host, port) = addr
```

- New way:

```
(host, port, flow, scope) = addr
```

- Clarification: this is parsing the output from the `accept()` call which returns `(conn, addr)`. As you can see, the IPv6 compatible change is the additional elements in the returned "`addr`" tuple.
- Used to make the address presentable in debugging output and user messages.

Python Client Migration

- Similar changes to C client
- Rearrange the address resolution stuff for `getaddrinfo()`
- Add some handling for `AF_INET6` to the connection loop
- Convert `inet_ntoa()` to `inet_ntop()` calls.
- Handle Protocol Family for `socket()` call

Code Changes (Python)

- Old Way:

```
for i in addrlist:
    print "Trying host %s." % i
    try:
        s.connect((i,tcpport))
    except socket.error, (errno, msg):
        print "Failed to connect to %s: %s." % (i, msg)
        continue
    break
else:
    print >>sys.stderr, "Connect failed."
    sys.exit(1)
```


Code Changes (Python) (Cont.)

- New Way:

```
for (fam, stype, proto, cname, saddr) in res:
    if (fam is socket.AF_INET6):
        (host, port, flow, scope) = saddr
    elif (fam is socket.AF_INET):
        (host, port) = saddr
    else:
        debug(3, "Skipping unknown address family:", fam)
        continue
    print "Trying host %s (%s) port %d." % (cname, host, port)
    try:
        s = socket.socket(fam, stype, proto)
    except socket.error, (errno, msg):
        s = None
        continue
    try:
        s.connect(saddr)
    except socket.error, (errno, msg):
        s.close()
        s=None
        print "Failed to connect to %s (%s): %s." % (cname, host, msg)
        continue
    if s: break
if s is None:
    print >> sys.stderr, "%s: No successful connection." % prog
    sys.exit(1)
```


Connecting (Python) (Cont.)

- In addition, there are minor modifications required in the successful connection message (variable names in print arguments).
- No other code changes needed in Python.

Function Replacement Guide (all languages)

Old Function	Current Function
get*by*()	getaddrinfo(), getnameinfo()
socket()	socket()†
bind()	bind()†
listen()	listen()
connect()	connect()†
recv*()	recv*()†
send*()	send*()†
accept()	accept()
read()/write()	read()/write()
inet_ntoa()/inet_aton()	inet_ntop()/net_pton() or getnameinfo()†
† parameters change for IPv6 suport	

Structure Replacement Guide

Old Structure	Current Structure
sockaddr_in,sockaddr_storage†	sockaddr_in6,sockaddr_storage†
in_addr, int (Don't do this, even in v4 only)	
hostent	addrinfo
servent	
†sockaddr_storage is a pointer type only can point to either actual type.	

Q&A

Copy of slides available at:
http://owend.corp.he.net/ipv6/PortMeth.pdf

Contact:

Owen DeLong
IPv6 Evangelist
Hurricane Electric
760 Mission Court
Fremont, CA 94539, USA
http://he.net/

owend at he dot net
+1 (408) 890 7992