Migrating Code to IPv6
the good, the bad and the ugly

Christian Grothoff
christian@grothoff.org
http://grothoff.org/christian/

Colorado Research Institute for Security and Privacy

"One of the chief factors that has prevented this transformation, though objectively it has been on the agenda for years, is the absence or the repression of the need for transformation, which has to be present as the qualitatively differentiating factor among the social groups that are to make the transformation." – Herbert Marcuse

Overview

- A minimal example: TCP server and client
- Migration of the minimal example
- DNS, URLs and other migration concerns
- Hard problems
- Checking application IPv6 readiness

Example: minimal IPv4 TCP server

Functionality:

- Listen to port 5002
- Write incoming TCP stream to disk
- Support multiple clients in parallel using `pthreads`

Use of `select` or `epoll` instead of `pthreads` to handle multiple clients never changes anything for IPv6.

Keeping it short...

- No declarations of variables unrelated to IPv4/6
- No error handling code
- Minor details ignored

$\Rightarrow$ Read man-pages to easily fill the gaps

CRISP

Server Example: processing

```
static void * process (struct T * t) {
  int n;
  char buf[4092];

  int f = creat (filename, S_IRUSR | S_IWUSR);
  while ( (-1 != (n=read (t->a, buf, sizeof (buf)))) &&
          (n != 0) )
    write (f, buf, n);
  close (f);
  close (t->a);
  return NULL;
}
```

IPv4 Server Example: accepting

```
struct sockaddr addr;
int s = socket (PF_INET,  SOCK_STREAM, 0);
memset (&addr, 0, sizeof (addr));
struct sockaddr_in * ia = (struct sockaddr_in*) &addr;
ia->sin_family = AF_INET; ia->sin_port = htons (5002);
bind (s, &addr, sizeof (struct sockaddr_in));
listen (s, 5);
while (1) {
  memset (&addr, 0, sizeof (addr));
  socklen_t alen = sizeof (struct sockaddr);
  t->a = accept (s, &addr, &alen);
  pthread_create (&pt, NULL, &process, t);          }
```

IPv6 Server Example: accepting

```
struct sockaddr_in6 addr;
int s = socket (PF_INET6, SOCK_STREAM, 0);
memset (&addr, 0, sizeof (addr));
struct sockaddr_in* ia = (struct sockaddr_in*)&addr;
ia->sin6_family=AF_INET6; ia->sin6_port= htons (5002);
bind (s, &addr, sizeof (struct sockaddr_in6));
listen (s, 5);
while (1) {
  memset (&addr, 0, sizeof (addr));
  socklen_t alen = sizeof (struct sockaddr);
  t->a = accept (s, &addr, &alen);
  pthread_create (&pt, NULL, &process, t);           }
```

Client Example: processing

```
static void process (int s) {
  char buf[4092];
  int f = open (FILENAME, O_RDONLY);
  while ( (-1 != (n = read (f, buf, sizeof (buf))) &&
          (n != 0) ) {
    pos = 0;
    while (pos < n) {
      ssize_t got = write (s, &buf[pos], n - pos);
      if (got <= 0) goto END;
      pos += got;                                   } }
END:
  close (f);                                        }
```

IPv4 Client Example

```
struct sockaddr_in addr;
struct sockaddr_in *ia;

int s = socket (PF_INET,  SOCK_STREAM, 0);
memset (&addr, 0, sizeof (addr));
addr.sin_family = AF_INET;
addr.sin_port = htons (5002);
addr.sin_addr.s_addr = htonl (INADDR_LOOPBACK);
ia = (struct sockaddr_in *) &addr;
connect (s, ia, sizeof (addr));
process(s);
close (s);
```

IPv6 Client Example

```
struct sockaddr_in6 addr;
struct sockaddr *ia;

int s = socket (PF_INET6, SOCK_STREAM, 0);
memset (&addr, 0, sizeof (addr));
addr.sin6_family= AF_INET6;
addr.sin6_port= htons (5002);
addr.sin6_addr = in6addr_loopback;
ia = (struct sockaddr_in*) &addr;
connect (s, ia, sizeof (addr));
process(s);
close (s);
```

What are we missing?

What about...

- ... running on an OS that does not support IPv6?
- ... parsing user-specified addresses?
- ... IP-based access control?
- ... DNS resolution?
- ... URL support?

Levels of OS support

The OS could:

- Lack basic IPv6 definitions in the C libraries (i.e., no `PF_INET6` constant defined)
- Have support in the C libraries but lack kernel support (IPv6 operations fail)
- Have kernel support enabled but only use IPv4 addresses for networking (some IPv6 operations succeed)
- Use IPv4 and IPv6 for networking, possibly depending on the interface
- Only use IPv6

Handling lack of IPv6 OS support (1/2)

```
  int v6 = 0;
  int s = -1;
#if HAVE_INET6_DEFINES
  s = socket (PF_INET6, SOCK_STREAM, 0);
  if (s != -1)
    v6 = 1;
  else
#endif
    s = socket (PF_INET4, SOCK_STREAM, 0);
  memset (&addr, 0, sizeof (addr));
```

Handling lack of IPv6 OS support (2/2)

```
#if HAVE_INET6_DEFINES
  if (v6 == 1) {
    ia6 = ...
    ia6.sin_family = AF_INET6;
    socklen = sizeof(struct sockaddr_in6);
    addr = (struct sockaddr_in*) &ia6;
  } else
#endif
  { ia4.sin_family = AF_INET;
    socklen = sizeof(struct sockaddr_in);  // ...
    addr = (struct sockaddr_in*) &ia4;
  }
```

```
connect (s, &addr, socklen);
```

IP-based access control

- Bind socket to limited IP addresses
- Check that connection is from trusted network
- Check that IP matches certain DNS names

IPv4 Server Example: loopback only

```
struct sockaddr_in ia;
int s = socket (PF_INET,  SOCK_STREAM, 0);
memset (&ia, 0, sizeof (ia));
ia.sin_family = AF_INET;
ia.sin_addr.s_addr = htonl (INADDR_LOOPBACK);
ia.sin_port = htons (5002);
struct sockaddr_in * addr = (struct sockaddr_in *)&ia;
bind (s, addr, sizeof (struct sockaddr_in));
// ...
```

IPv6 Server Example: loopback only

```
struct sockaddr_in6 ia;
int s = socket (PF_INET6, SOCK_STREAM, 0);
memset (&ia, 0, sizeof (ia));
ia.sin6_family= AF_INET6;
ia.sin6_addr = inaddr6_loopback;
ia.sin6_port= htons (5002);
struct sockaddr_in* addr = (struct sockaddr_in*)&ia;
bind (s, addr, sizeof (struct sockaddr_in6));
// ...
```

Parsing IPv4 addresses

```
int parse_v4(const char * in,
             struct in_addr * out) {
  int ret = inet_pton(AF_INET,  in, out);
  if (ret < 0)
    fprintf(stderr, "AF_INET not supported!\n");
  else if (ret == 0)
    fprintf(stderr, "Syntax error!\n");
  else
    return 0;
  return -1;
}
```

Parsing IPv6 addresses

```
int parse_v6(const char * in,
             struct in6_addr * out) {
  struct in_addr v4;
  int ret = inet_pton(AF_INET6, in, out);
  if (ret > 0) return 0;
  ret = inet_pton(AF_INET, in, &v4);
  if (ret < 0) return -1; /* error */
  memset(out, 0, sizeof(struct in6_addr));
  ((unsigned int *) out)[2] = htonl (0xffff);
  memcpy (&((char *) out)[sizeof (struct in6_addr) -
                          sizeof (struct in_addr)],
        &v4, sizeof (struct in_addr)); return 0; }
```

IPv4 network check

```
int
test_in_network_v4 (const struct in_addr * network,
                    const struct in_addr * mask,
                    const struct in_addr * addr) {
  return ( (addr->s_addr & mask.s_addr)
         == network.s_addr & mask.s_addr)
}
```

IPv6 network check

```
int test_in_network_v6 (const struct in6_addr * network,
                        const struct in6_addr * mask,
                        const struct in6_addr * addr) {
  unsigned int i;
  for (i=0; i<sizeof(struct in6_addr)/sizeof (int); i++)
    if ( ((((int *) ip     )[i] & ((int *) mask)[i])) !=
          (((int *) network)[i] & ((int *) mask)[i]))
      return 0;
  return 1;
}
```

Generic network check

```
int test (struct in_addr * n4, struct in_addr * m4,
          struct in6_addr* n6, struct in6_addr* m6,
          struct in6_addr * addr) {
  struct in_addr ip4;
  if (test_in_network_v6(n6, m6, addr)) return 1;
  memcpy (&ip4, &((char *) &ip6)
     [sizeof(struct in6_addr)-sizeof(struct in_addr)],
     sizeof (struct in_addr));
  if (IN6_IS_ADDR_V4MAPPED (&a6->sin6_addr))
    return test_in_network_v4(n4, m4, addr);
  return 0; }
```

IPv4 DNS request

```
int
resolve_v4 (const char * hostname,
            struct in_addr * addr) {
  struct hostent * he;
  struct sockaddr_in *addr;
  he = gethostbyname(hostname);
  assert (he->h_addrtype == AF_INET);
  assert (hp->h_length == sizeof (struct in_addr));
  memcpy (addr, hp->h_addr_list[0], hp->h_length);
  return OK;
}
```

`gethostbyname` issues

- Synchronous
- IPv4 only

⇒ `gethostbyname2`

`gethostbyname` issues

- Synchronous
- IPv4 only

⇒ `gethostbyname2`

- Not reentrant

⇒ both are obsolete!

IPv4 DNS request with getaddrinfo

```
int resolve_v4 (const char * hostname,
                struct sockaddr_in * addr) {
  struct addrinfo hints;
  struct addrinfo *result;
  memset (&hints, 0, sizeof (struct addrinfo));
  hints.ai_family = AF_INET;
  getaddrinfo (hostname, NULL, &hints, &result);
  assert(sizeof(struct sockaddr_in) ==result->ai_addrlen);
  memcpy (addr, result->ai_addr, result->ai_addrlen);
  freeaddrinfo (result);
  return OK;
}
```

IPv6 DNS request with getaddrinfo

```
int resolve_v6 (const char * hostname,
                struct sockaddr_in6* addr) {
  struct addrinfo hints;
  struct addrinfo *result;
  memset (&hints, 0, sizeof (struct addrinfo));
  hints.ai_family = AF_INET6;
  getaddrinfo (hostname, NULL, &hints, &result);
  assert(sizeof(struct sockaddr_in6)==result->ai_addrlen);
  memcpy (addr, result->ai_addr, result->ai_addrlen);
  freeaddrinfo (result);
  return OK;
}
```

Generic DNS request with getaddrinfo

```
int resolve (const char * hostname,
             struct sockaddr** addr,
             socklen_t * addr_len, int * af) {
  struct addrinfo hints;
  struct addrinfo *result;
  memset (&hints, 0, sizeof (struct addrinfo));
  hints.ai_family = AF_UNSPEC;
  getaddrinfo (hostname, NULL, &hints, &result);
  *addr = malloc(*addr_len = sizeof(result->ai_addrlen));
  memcpy (*addr, result->ai_addr, result->ai_addrlen);
  *af = result->ai_family;
  freeaddrinfo (result);  return OK;  }
```

Generic Client Example

```
struct sockaddr * addr;
resolve(HOSTNAME, &addr, &alen, &af);
s = socket (af == AF_INET ? PF_INET : PF_INET6,
            SOCK_STREAM, 0);
if (af == AF_INET)
  ((struct sockaddr_in*)addr)->sin_port=htons (5002);
else
  ((struct sockaddr_in6*)addr)->sin6_port=htons (5002);
connect (s, addr, alen);
process(s);
free(addr); close (s);
```

URL support

- IPv4: `http://127.0.0.1:8080/`

URL support

- IPv4: `http://127.0.0.1:8080/`
- IPv6: `http://::1:8080/` – does not work!

URL support

- IPv4: `http://127.0.0.1:8080/`
- IPv6: `http://::1:8080/` – does not work!
- Solution: `http://[::1]:8080/`

Other considerations

- Use `getnameinfo` instead of `gethostbyaddr` for reverse lookup
- Check if your system uses IPv4 binary addresses embedded in network protocols
- You must specify the interface if you use IPv6 link local addresses (or do not use them!)
- Check IPv6 support in libraries (GNU ADNS does not support IPv6!)

IPv6 and Infrastructure

- IPv6 clients talking to IPv4-only server
- IPv4 clients talking to IPv6-only server
- Improved security / new IPv6 options:
  - Some new options require using raw sockets
  - Compatibility and migration nightmare
  - Applications already use SSL/IPsec

$\Rightarrow$ Rarely supported (nicely) by OS

Are we done yet?

On a GNU/Linux system, run:

- `$ netstat -nl`

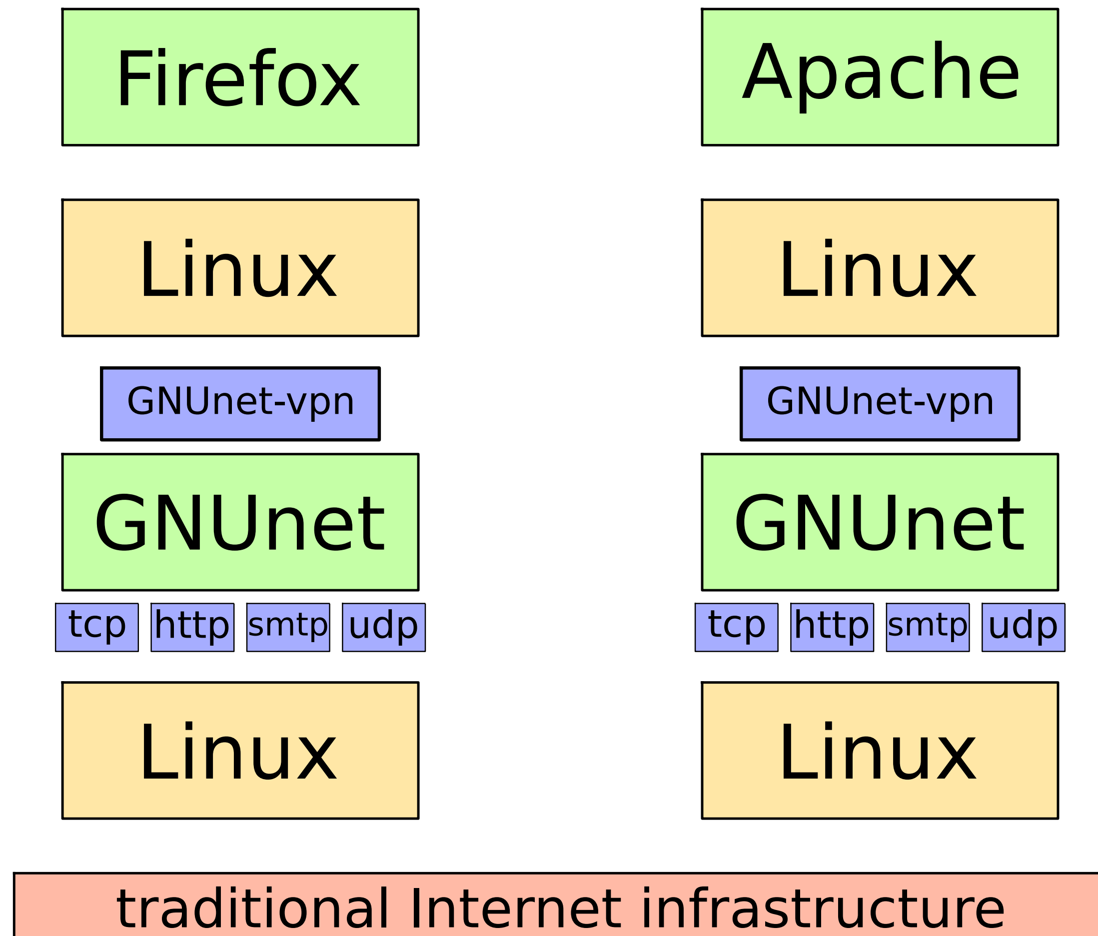
"Misery motivates, not utopia." – Karl Marx

Questions

IPv6 over GNUnet

- GNUnet is GNU's secure peer-to-peer framework
- Peers communicate using http, smtp, tcp or udp
- Both IPv4 and IPv6 supported
- If direct communications do not work, messages can be routed via other peers
- GNUnet uses link-encryption
- VPN application provides IPv6 interface using Linux VTUN interface

IPv6 over GNUnet



Copyright

Copyright (C) 2008 Christian Grothoff

Verbatim copying and distribution of this entire article is permitted in any medium, provided this notice is preserved.

GNUnet is part of the GNU project.