IPv6 - Migrating Applications

RBG-Seminar
Universitaet Bielefeld
2010-11-23

Holger Kälberer

Agenda

01 **IPv4 vs. IPv6: Overview and Status Quo**

02 **IPv6 support (1): Content level**

03 **IPv6 support (2): Socket level**

01

IPv4 vs. IPv6:
Overview and Status Quo

IPv4 vs. IPv6

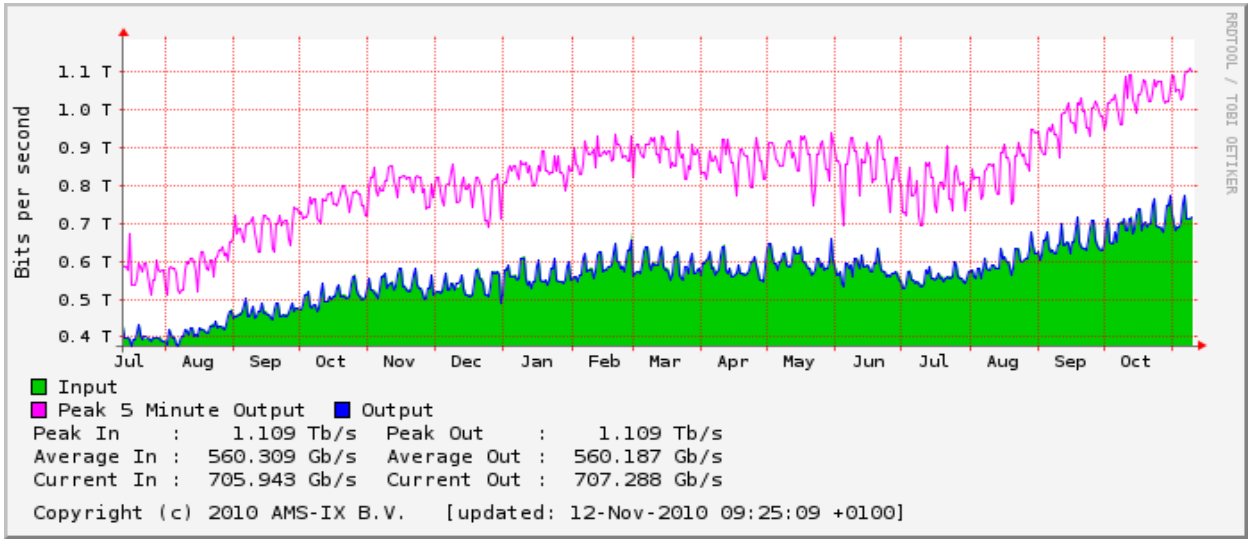
Context and Status quo

- IPv4 address depletion:
  - IPv4 Address Report http://www.potaroo.net/tools/ipv4/index.html: 06-Jun-2011 / 28-Jan-2012
- Transition: missing economic incentive for quick transistion
- Maturing Hardware + Software Support: Vista, Fritz! Box 7270 (02/2009), ...
- IPv4 and IPv6 are not interoperable --> interconnection & coexistence

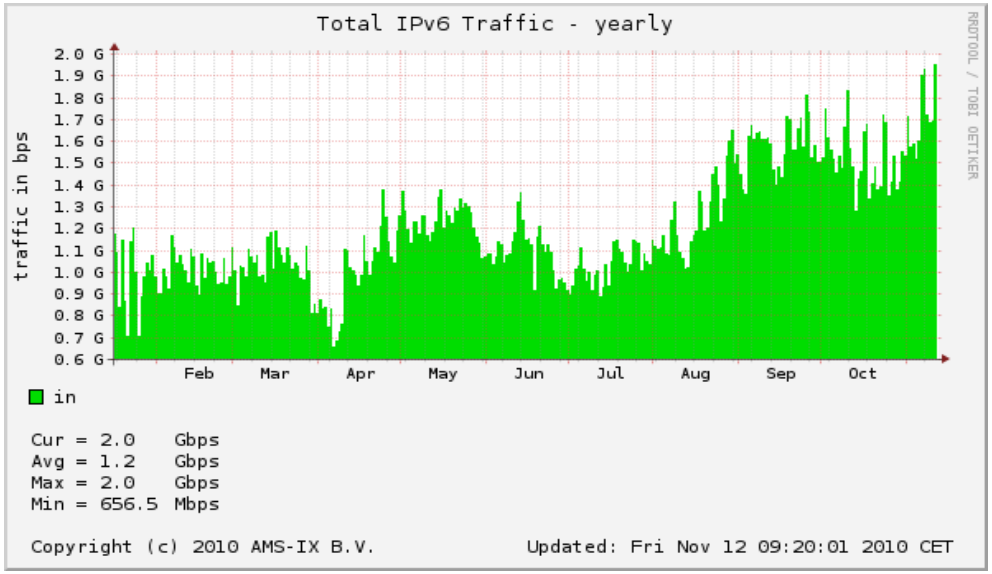
IPv4 vs. IPv6

AMSIX Traffic

IPv4



IPv6



IPv4 vs. IPv6

Major Changes/Benefits (RFC 2460)

- Expanded Addressing Capabilities:

  larger address space, auto-configuration, scope
- Header Format Simplification:

  fixed header length: 40 octets = 5 x 64 bit
- Improved Support for Extensions and Options
- Flow Labeling Capability

  labeling of packets allow defining services-classes (QoS, RT)
- Authentication and Privacy Capabilities

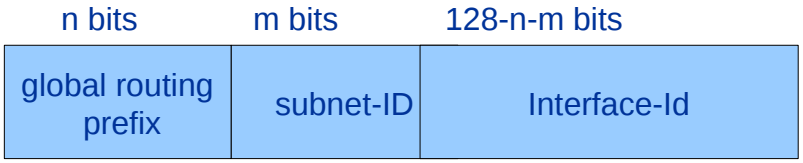
195.71.38.13 -- 2^{32}

3ffe:400:ad0:20d:290:27ff:fe1d:d0e9 -- 2^{128}

IPv6 Addressing

RFC 4291

- Address Types: Unicast, Multicast, Anycast
- Notation:
- General unicast format:




```
Addresses: x:x:x:x:x:x:x:x
2001:0470:1f0a:02a6:0000:0000:0000:0002
2001:470:1f0a:2a6:0:0:0:2
2001:470:1f0a:2a6::2

Prefixes: <v6-address>/<prefix-length>
2001:470:1f0a:2a6::/64
2001:470:1f0a:2a6::2/64
::1/128
```

- Scope: Link-Local; Global; (Site-Local, deprecated)
- Some prefixes/addresses:

Prefix/addr.	Use
2000::/3	global Unicast
FE80::/10	Link Local Unicast
FF00::/8	Multicast
FF02::1	Link-local all nodes mutlicast group

IPv4 vs. IPv6

Layers of IP(v6)-support

1. Network connectivity (incl. IP-address)
2. OS, TCP/IP-stack
3. Application
    1. content
    2. socket

Application Layer

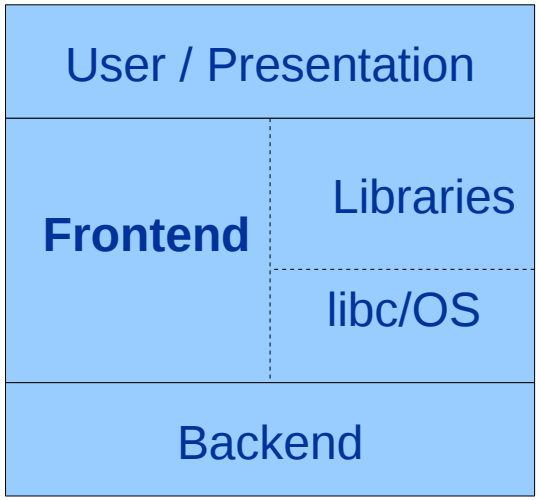
Content vs. Socket Level

- Processing of IPv6 content/addresses vs. IPv6 network communication
- Some Apps dealing with IPv6 (ISP/Carrier):
  - Inventory, Network Managment tools (SNMP-poller, Monitoring, ...), Access (AAA, Billing), DNS, Ticketing, Reporting, ...
- Services will be migrated first
- Network managment (monitoring, inventory, maintenance, ...) can/will use IPv4 for some time
- But: IPv6 content will need to be processed
- --> IPv6-content comes first

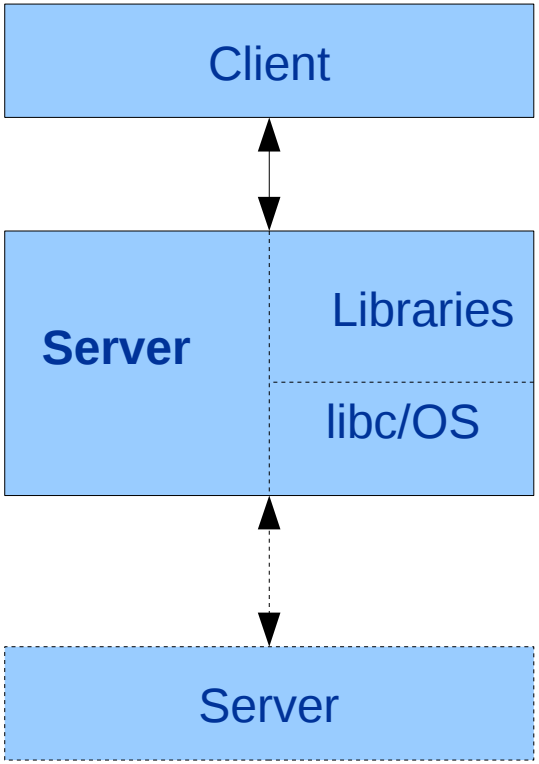
Application Layer

Architectures

- 3-Tier Application



- Client-Server



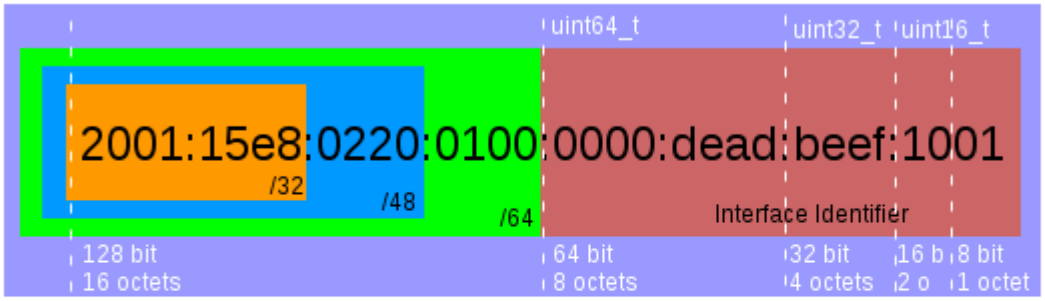
IPv6 Support (1): Content Level

IPv6 content

IPv6 addresses again

- 2^{128} = 2 * 64 bit = 4 * 32 bit = 16 bytes / octets
- 2^{128}-1 = 340282366920938463463374607431768211455 --> strlen(toString(2^{128}-1)) = 39
- strlen("2001:15e8:0220:0100:0000:dead:beef:1001/128") + 1 = 44

```
#define INET6_ADDRSTRLEN 46
```



Representing 128 bit

Numeric vs. String-type

- Numeric vs. String: depends on your application ...
  - memory/storage (2x64 bit vs. INET6_ADDRSTRLEN)
  - performance? (e.g. IN6_ARE_ADDR_EQUAL vs. strcmp)
  - use (do we perform many calculations?)
  - existing interfaces
- Talking about *migration* of applications a numeric representation might be the better way...

A Numeric Representation

<netinet/in.h>

```
/* Internet address. */
typedef uint32_t in_addr_t;
struct in_addr
  {
    in_addr_t s_addr;
  };
```

```
/* IPv6 address */
struct in6_addr
  {
    union
      {
        uint8_t __u6_addr8[16];
#if defined __USE_MISC || defined __USE_GNU
        uint16_t __u6_addr16[8];
        uint32_t __u6_addr32[4];
#endif
      } __in6_u;
#define s6_addr                 __in6_u.__u6_addr8
#if defined __USE_MISC || defined __USE_GNU
# define s6_addr16              __in6_u.__u6_addr16
# define s6_addr32              __in6_u.__u6_addr32
#endif
  };
```

```
#define IN6_ARE_ADDR_EQUAL(a,b) \
       ((((__const uint32_t *) (a))[0] == ((__const uint32_t *) (b))[0])     \
        && (((__const uint32_t *) (a))[1] == ((__const uint32_t *) (b))[1])  \
        && (((__const uint32_t *) (a))[2] == ((__const uint32_t *) (b))[2])  \
        && (((__const uint32_t *) (a))[3] == ((__const uint32_t *) (b))[3]))
```

Interfaces

Be generic! in ...

- ... data structures and

```
typedef struct gen_in_addr
{
  uint8_t family;
  union {
    struct in_addr  gi_iaddr;
    struct in6_addr gi_i6addr;
  } addr_u;
#define gi_addr4 addr_u.so_sin
#define gi_addr6 addr_u.so_sin4
} gen_in_addr_t;
```

```
class IP {
  public $ipArray; // uint32_t[4]
  public $ipVer;

  // ...
};
```

- ... interfaces

```
int cmp_gen_in_addr(gen_in_addr_t *a, gen_in_addr_t *b)
{
  switch(a->family) {
    case AF_INET:
      return(memcmp( &(a->gi_addr4), &(b->gi_addr4), sizeof(struct in_addr)));
      break;
    case AF_INET6:
      return(memcmp( &(a->gi_addr6), &(b->gi_addr6), sizeof(struct in6_addr)));
      break;
  }
  return -1;
}
```

- Example: Net::IP

IPv6 Arithmetic

- Arithmetic on IPv6 addresses
  - ... using existing libraries (Perl: Net::IP, NetAddr:IP; Python)
  - or do it yourself (PHP, C, ...):

quick-n-dirty

```
/** Multiply the first IPv6 by the passed factor
 * ...
 * @return ptr to the resulting IPv6 address
 *         or NULL on error
 */
static struct in6_addr*
in6_addr_mul(struct in6_addr *a,
             const uint16_t factor)
{
  uint32_t res = 0, rem = 0;
  assert(factor < (1 << 16));
  for (int i = 7; i >= 0; i--) {
    res = ntohs(a->s6_addr16[i]) * factor
          + rem;
    a->s6_addr16[i] = htons(res & 0xffff);
    rem = (res & 0xffff0000) >> 16;
  }
  if (rem != 0) {
    fprintf(stderr, „overflow\n“);
    return NULL;
  }
  return a;
}
```

slower-and-generic (gmp)

```
/**
 * Compute diff between IP2 and IP1
 * ...
 * @return the resulting int-value or false
 *         on error
 */
public function diff($lIP1, $lIP2, $ipVer = 4)
{
  if ($ipVer == 4) {
    /* speed up processing of v4-adresses */
    $result = $lIP1[0] - $lIP2[0];
  } else if ($ipVer == 6) {
    $gmpIP1 = self::IPToGMP($lIP1, $ipVer);
    $gmpIP2 = self::IPToGMP($lIP2, $ipVer);
    $diff = gmp_sub($gmpIP1, $gmpIP2);
    $result = gmp_intval($diff);
  } else
    return $result = false;
  return $result;
}
```

RDBM Backends

1. Use existing builtin datatypes (cidr/inet in PostgreSQL); might force you to use strings on app-layer.
2. Use string-representation/VARCHAR
3. Use a n-tupel of Number (Oracle) / Unsigned Int (MySQL):

```
ALTER TABLE ip
  ADD (
    ip1 NUMBER(10) NULL,
    ip2 NUMBER(10) NULL,
    ip3 NUMBER(10) NULL,
    ip_ver NUMBER(1) NULL
    );

DROP INDEX IDX_IP_IP;
CREATE UNIQUE INDEX IDX_IP_UNIQUE_V6IP ON ip (ip, ip1, ip2, ip3, ip_ver);
```

- For MySQL cf. article Blapp, Martin - „Datenbanken fuer IP-Adressen. Optimale Strategien fuer die Adreßspeicherung in Datenbanken“ (freeX 6'2010, pp. 48-57)

IPv6 Support (2): Socket Level

IPv6 Socket API Extension

RFC 3493 (Obsoletes: 2553)

- Designed to
  - support IPv6
  - be backward compatible
  - be generic
- Changed Components
  1. Core socket functions.
  2. Address data structures.
  3. Name-to-address translation functions.
  4. Address conversion functions.

API Extensions

Data Structures

- Address/Protocol Families

```
<sys/socket.h>
#define PF_INET   AF_INET
```

```
<sys/socket.h>
#define PF_INET6   AF_INET6
```

- IP-Addresses

```
<netinet/in.h>
struct in_addr;
```

```
<netinet/in.h>
struct in6_addr;
```

- Socket Addresses

```
<sys/socket.h>
struct sockaddr_in {
  sa_family_t     sin_family;   /* AF_INET */
  in_port_t       sin_port;     /* port number */
  struct in_addr  sin_addr;     /* IPv4 address */
  char            sin_zero[8];  /* unused */
};

struct sockaddr {
  sa_family_t  sa_family;   /* address family */
  char         sa_data[14]; /* prot-specific address */
};
```

```
<netinet/in.h>
struct sockaddr_in6 {
  sa_family_t     sin6_family;    /* AF_INET6 */
  in_port_t       sin6_port;      /* port # */
  uint32_t        sin6_flowinfo;  /* flow info */
  struct in6_addr sin6_addr;      /* IPv6 address */
  uint32_t        sin6_scope_id;  /* scope */
};

struct sockaddr_storage {
  sa_family_t    __ss_family;      /* address family */
  __ss_aligntype __ss_align;       /* alignment */
  char __ss_padding[_SS_PADSIZE];  /* padding */
};
```

API Extensions

Functions

- Address Conversion

```
<arpa/inet.h>
int inet_aton(const char *cp, struct in_addr *inp);
char *inet_ntoa(struct in_addr in);
in_addr_t inet_addr(const char *cp);
… INET(3)
```

```
<arpa/inet.h>
int inet_pton(int af, const char *src, void *dst);
const char *inet_ntop(int af, const void *src,
                      char *dst, socklen_t size);
```

- Nodename/Servicename Translation

```
<netdb.h>
struct hostent *gethostbyname(const char *name);
struct hostent *gethostbyname2(const char *name,
                               int af);
struct servent *getservbyname(const char *name,
                              const char *proto);

struct hostent *gethostbyaddr(const void *addr,
                              socklen_t len, int type);
struct servent *getservbyport(int port,
                              const char *proto);


… cf. GETHOSTBYNAME(3) GETSERVENT(3)
```

```
<netdb.h>

int getnameinfo(const struct sockaddr *sa,
                socklen_t salen, char *host,
                size_t hostlen, char *serv,
                size_t servlen, int flags);

int getaddrinfo(const char *node, const char *service,
                const struct addrinfo *hints,
                struct addrinfo **res);
```

Translation

getaddrinfo(3) (POSIX 1.g)

- {DEMO ...}

```
                          <netdb.h>
struct addrinfo {
  int     ai_flags;     /* AI_PASSIVE, AI_CANONNAME,
                           AI_NUMERICHOST, .. */
  int     ai_family;    /* AF_xxx */
  int     ai_socktype;  /* SOCK_xxx */
  int     ai_protocol;  /* 0 or IPPROTO_xxx for IPv4 and IPv6 */
  socklen_t  ai_addrlen;   /* length of ai_addr */
  char   *ai_canonname; /* canonical name for nodename */
  struct sockaddr  *ai_addr; /* binary address */
  struct addrinfo  *ai_next; /* next structure in linked list */
};
```

- Name to Socket Address (getaddrinfo):

```
                                <netdb.h>
/* Possible values for `ai_flags' field in `addrinfo' structure.  */
# define AI_PASSIVE     0x0001      /* Socket address is intended for `bind'.  */
# define AI_CANONNAME   0x0002      /* Request for canonical name.  */
# define AI_NUMERICHOST 0x0004      /* Don't use name resolution.  */
# define AI_V4MAPPED    0x0008      /* IPv4 mapped addresses are acceptable.  */
# define AI_ALL         0x0010      /* Return IPv4 mapped and IPv6 addresses.  */
# define AI_ADDRCONFIG  0x0020      /* Use configuration of this host to choose
                                       returned address type..  */
```

Translation

getnameinfo(3) POSIX 1.g

- Socket Address to Name (getnameinfo)

```
                                        <netdb.h>
# define NI_NUMERICHOST 1          /* Don't try to look up hostname.  */
# define NI_NUMERICSERV 2          /* Don't convert port number to name.  */
# define NI_NOFQDN      4          /* Only return nodename portion.  */
# define NI_NAMEREQD    8          /* Don't return numeric addresses.  */
# define NI_DGRAM       16         /* Look up UDP service rather than TCP.  */
```

- Note IPv6 preference over IPv4 for destination address selection and beware of IPv6 blackholes:

```
 RFC 3484 - Default Address Selection for Internet Protocol version 6 (IPv6)

[...]
10.3. Configuring Preference for IPv6 or IPv4

   The default policy table gives IPv6 addresses higher precedence than
   IPv4 addresses.  This means that applications will use IPv6 in
   preference to IPv4 when the two are equally suitable.
[...]
```

Resume

(1) Content

- Representation: Use numeric representation if ...
  - you perform calculations / have to be fast
  - size matters
- Use strings otherwise
- Storage: Use the same representation as on app-tier
- Interface: Handle IP-version transparently
- Transformation: Use IPv6-aware functions (inet_ntop)

Resume

(2) Socket

1. Identify non-portable code:

```
$ egrep '(gethostby|getservby|inet_(aton|ntoa)|sockaddr_in|in_addr') `find . -iname '*[hc]'
```

2. Replace by portable code:

```
struct sockaddr_in                          → struct sockaddr (passing)
                                            → struct sockaddr_storage (storing)
struct in_addr                              → struct in6_addr OR struct sockaddr_storage
                                              OR struct addrinfo OR self-defined struct

gethostbyname                               → getaddrinfo
gethostbyname2
getservbyname

gethostbyaddr                               → getnameinfo
getservbyport

FIXME...
```

Some IPv6 RFCs

- RFC 2292, "Advanced Sockets API for IPv6“ (Feb. 1998)
- RFC 2460, "Internet Protocol, Version 6 (IPv6) Specification" (December, 1998)
- RFC 3484, "Default Address Selection for Internet Protocol version 6 (IPv6)“ (Feb. 2003)
- RFC 3493, "Basic Socket Interface Extensions for IPv6“ (Feb. 2003; Obsoletes: 2553, Mar. 1999)
- RFC 3596, "DNS Extensions to Support IP Version 6“ (Oct. 2003)
- RFC 4291, "IP Version 6 Addressing Architecture" (Februray, 2006)
- RFC 4443, "Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification" (March, 2006)
- RFC 4861, "Neighbor Discovery for IP version 6 (IPv6)“ (Sept. 2007)
- RFC 4862, "IPv6 Stateless Address Autoconfiguration“ (Sept. 2007)

References

- Understanding IP-Addressing: http://www.ripe.net/info/info-services/addressing.html
- Jun-ichiro itojun Hagino; IPv6 Network Programming; Elsevier Digital Press; 2004
- R. Stevens et al., UNIX® Network Programming Volume 1, Third Edition: The Sockets Networking API, Addison Wesley, 2003
- Silvia Hagen; IPv6 Essentials; O'Reilly Media; 2002
- TR-187. IPv6 for PPP Broadband Access (Broadband Forum): http://www.broadband-forum.org/technical/download/TR-187.pdf
- Geoff Huston; IPv6 Transition at IETF 72; in: IETF Journal,Volume 4 Issue 2 (October 2008); pp. 25-30: http://isoc.org/wp/ietfjournal/
- IPv6 Deployment Strategies (Cisco Systems) ?

Thanks!
Questions?