Networking from the Bottom Up: IPv6

George Neville-Neil

gnn@neville-neil.com

May 8, 2010
What We Will Cover

- A bit of the History and Goals of IPv6
- IPv6 Protocol Code
- Neighbor Discovery
- Router Discovery
- ICMPv6
- IPSec
What We Will *Not* Cover

- Routing
- TCPv6
- UDPv6
- SCTP (See Randall Stewart’s excellent tutorial.)
What Problem Are You Trying To Solve?

- Running out of addresses
- Efficiency
- Manageability
- Security
Protocol Historical Context

- Early 90s move to classless inter domain routing (CIDR)
- 1990: RFC 1287 Future Internet Architecture
- 1992: RFC 1335 Discusses exhaustion issue
- 1995: First IPv6 RFCs
- 1998: First acceptable IPv6 RFCs (2460 et al)
Code History

- Originally three open source implementations of IPv6
  - Naval Research Lab (US)
  - INRIA (France)
  - Kame (Japan)

- Kame Project wins out over the other two

- All work originally done in the BSD community

- Largest and riskiest kernel sub system developed outside of the BSD projects

- Kame Project ends active development in 2007

- Code fully taken over by the relevant OS projects
IPv6 Differences

- Addresses
- MTU
- Header Processing
- Scoping
- Multicast
- Autoconfiguration
Addresses

- The most obvious and talked about change
- 128 bits for the host address
- IPv4 didn’t have enough for everyone alive
- IPv6 has enough for every atom in the universe
MTU

- Maximum Transfer Unit
- IP is a hop by hop, packet switched protocol
- Fragmentation was seen as a problem
- Having an end to end MTU improves performance
The IPv4 header is messy

- Two 4 bit fields
- One 3 bit field
- One 13 bit field
- Options

Make the header as simple as possible

Have the packet look like a linked list
Scoping

- A novel way of asking the local/remote question
- An attempt to replace subnetting within organizations
- Too complicated for many uses
Multicast

- More efficient than broadcast
- Available in most common data-link protocols
- Used heaving in auto configuration
Autoconfiguration

- Trying to solve the Dentist’s Office
- Does anyone still create isolated networks?
- Introduces new problems
Neighbor Discovery

- Replacement for ARP
- Partial replacement for DHCP
- Removal of a layering violation
Router Discovery

- Lessens the burden of administrators
- Partial replacement for DHCP
Sockets API

- A by-product of some of the changes
- Overcome problems with socket addressing
Directories and Files

- Majority of the code resides in sys/netinet6
- Two files present in sys/netinet
  - icmp6.h
  - ip6.h
Memory for Packets

- Packets need to be stored for reception and transmission
- The basic packet memory structures are the mbuf and cluster
- mbuf structures have several types and purposes
- Clusters hold only data
- History dictates that mbufs are named m
- In the kernel we will see many pointers to mbufs
Types of mbufs

- Wholly contained
- Packet Header
- Using a cluster
Welcome to SMP

- FreeBSD is a multi-threaded, re-entrant kernel
- Only way to scale on multicore and multi-processor systems
- Kernel is full of cooperating tasks
- Inter process synchronization is *required*
Kernel Synchronization Primitives

- Spin Locks
- Mutexes
- Reader/Writer Locks
- Shared/Exclusive Locks
- Drivers use mostly spin locks or mutexes
  - See locking(9) for more information
IPv6 Specific Data Structures

- Addresses
- Packet Header
- Extension Headers
  - Examined at the endpoint.
- Hop by Hop Options
  - Examined at each intermediate hop
Address Structures

```c
struct sockaddr_in6 {
    uint8_t    sin6_len;    /* length of this struct */
    sa_family_t sin6_family; /* AF_INET6 */
    in_port_t   sin6_port;  /* Transport layer port # */
    uint32_t    sin6_flowinfo; /* IP6 flow information */
    struct in6_addr sin6_addr; /* IP6 address */
    uint32_t    sin6_scope_id; /* scope zone index */
};

struct in6_addr {
    union {
        uint8_t    __u6_addr8[16];
        uint16_t   __u6_addr16[8];
        uint32_t   __u6_addr32[4];
    } __u6_addr;   /* 128-bit IP6 address */
};
```
**IPv4 Header**

```c
struct ip {
  #if BYTE_ORDER == LITTLE_ENDIAN
    u_int ip_hl:4, /* header length */
    ip_v:4; /* version */
  #endif
  #if BYTE_ORDER == BIG_ENDIAN
    u_int ip_v:4, /* version */
    ip_hl:4; /* header length */
  #endif
  u_char ip_tos; /* type of service */
  u_short ip_len; /* total length */
  u_short ip_id; /* identification */
  u_short ip_off; /* fragment offset field */
  #define IP_RF 0x8000 /* reserved fragment flag */
  #define IP_DF 0x4000 /* dont fragment flag */
  #define IP_MF 0x2000 /* more fragments flag */
  #define IP_OFFMASK 0xffff /* mask for fragmenting bits */
  u_char ip_ttl; /* time to live */
  u_char ip_p; /* protocol */
  u_short ip_sum; /* checksum */
  struct in_addr ip_src, ip_dst; /* source and dest address */
} __packed __aligned(4);
```
IPv6 Header

```c
struct ip6_hdr {
    union {
        struct ip6_hdrctl {
            u_int32_t ip6_un1_flow; /* 20 bits of flow-ID */
            u_int16_t ip6_un1_plen; /* payload length */
            u_int8_t  ip6_un1_nxt;  /* next header */
            u_int8_t  ip6_un1_hlim; /* hop limit */
        } ip6_un1;
        u_int8_t  ip6_un2_vfc;  /* 4 bits version, top 4 bits class */
    } __packed;

    struct in6_addr ip6_src;  /* source address */
    struct in6_addr ip6_dst;  /* destination address */
};
```
Extension Header Structure

```c
struct ip6_ext {
    u_int8_t ip6e_nxt;
    u_int8_t ip6e_len;
} __packed;
```

```c
/* Fragment header */
struct ip6_frag {
    u_int8_t ip6f_nxt;         /* next header */
    u_int8_t ip6f_reserved;    /* reserved field */
    u_int16_t ip6f_offlg;      /* offset, reserved, and flag */
    u_int32_t ip6f_ident;      /* identification */
} __packed;
```
## Hop by Hop Options

<table>
<thead>
<tr>
<th>Next: 0</th>
<th>Next: 51</th>
<th>Next: 50</th>
<th>Next: 6</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPv6</td>
<td>Hop by Hop</td>
<td>AH</td>
<td>ESP</td>
<td>TCP</td>
</tr>
</tbody>
</table>

Length: 1
Length: 4
Length: 6

IPv6 Hop by Hop AH ESP TCP
Hop by Hop Options Structure

```c
struct ip6_hbh {
    u_int8_t ip6h_nxt;       /* next header */
    u_int8_t ip6h_len;       /* length in units of 8 octets */
    /* followed by options */
} __packed;

/* Jumbo Payload Option */
struct ip6_opt_jumbo {
    u_int8_t ip6oj_type;
    u_int8_t ip6oj_len;
    u_int8_t ip6oj_jumbo_len[4];
} __packed;
```
The Four Paths

- Packets traverse four possible paths in the network code
- Inbound (for this host)
- Outbound (from this host)
- Forwarding (between two interfaces on this host)
- Error
Four Paths Through The Stack

IPv4, IPv6, ...

igb0
inbound
outbound

igb1
error
forwarding

Network Layer
Interfaces
Packet Reception (Outline)

- Initial Reception
- Handle Scoping
- Hop by Hop Options
- Forwarding Decision
- More Packet Unwrapping
IPv6 Routing and Forwarding

- IPv6
- Neighbor Discovery
- Router Discovery
- ICMPv6
- RIB
- FIB
Packet Transmission

- Extension Headers
- IPSec Handling
- Length Calculation
- Jumbo Payload
- Pick a Source Address
- Routing Lookup
Packet Transmission (Con’t)

- Traffic Class
- Hop Limit
- IPSec Re-injection
- Select a Route
- Outbound Scope Check
- Multicast Handling
- Path MTU
Packet Transmission (Con’t)

- Hop by Hop
- Checksumming
- Fragmentation
- Transmit
- Cleanup
ICMPv6

- Now used for more than errors
- An integral part of auto-configuration
- Handles Neighbor and Router Discovery (see next slides)
ICMPv6 Packet Reception

- `icmp6_input`
- Preamble
- Get a usable structure
- Calculate the Checksum
- The Massive Switch
- Echo Request (ping)
- Neighbor and Router
Autoconfiguration

The ARP Replacement

- Translate an IPv6 address into a hardware address
- Piggy backed on top of ICMPv6
- Can take the place of DHCP
IPv6 Module Relationships

IPv6

ICMPv6

ND6

Network Interface

if_output()

ip6_input()

icmp6_input

ip6_output()

nd6_output()

nd6_ra_input()

nd6_rs_input()
Neighbor Lookup

- nd6_output
- Initial error checks
- Find a cached entry
- Create a new entry
- Send a neighbor solicitation
- Queue the packet
- Transmit the packet
Neighbor Solicitation Transmission

- `nd6_ns_output`
- Preamble
- Multicast Check
- Fill in the solicitation packet
- Duplicate Address Detection
- Call `ip6_output` (again)
Solicitation Input

- nd6_ns_input
- On link check
- Option Processing
- Proxy check
- Tentative
Autoconfiguration

Advertisement Output

- nd6_na_output
- Setup
- Create Packet
- Scope Selection
- Target Link Layer Address
- Checksums
- Call ip6_output
Autoconfiguration

Advertisement Input

- nd6_na_input
- Preamble
- Flag extraction
- Multicast checks
- Options processing
- Cache lookup
- Entry update
- Address change
- Transmit held packets
Discovering Routers

- Separate from Neighbor Discovery
- Replaces manual configuration as well as DHCP
- Supposed to ease large deployments
- Has security and other implications
Autoconfiguration

Router Advertisement

- nd6_ra_input
- Security Check
- Options processing
- Setup default router structure
- Handle prefix information
- MTU advertisement
- Source Link Layer Address
Autoconfiguration

Router Solicitation

- Accepted by *only* by routers
- Used by hosts to find routers
- Handled in user-space by rtsold(8)
- Uses ICMPv6 messages to find nearby routers
Autoconfiguration Wrap Up

- Neighbor Discovery replaces ARP
- Router Discovery replaces configuration files
- ICMPv6 Used Throughout
- Most messages are multicast to known groups
IPSec

- The collection of protocols for IP Security
- Exist for IPv6 and IPv4
- Tunnel Architecture
- Authenticate and Encrypt Packets
- Keyeing is non-trivial
- Can be computationally expensive
- Code is in sys/netipsec
The Protocol Switch

- A table of protocols and functions
- One for each protocol
- Not specific to IPv6
- inet6sw
Initial Reception

- ipsec6_common_input
- Pull up the whole packet
- Check the packet
- ipsec_common_input
Transformations

- Another set of structures with functions
- One per protocol
  - xform_ah.[ch], xform_esp.[ch]
- Implement a callback API
- Can easily offload to specialized hardware
struct xformsw {
    u_short xf_type;     /* xform ID */
#define XF_IP4 1        /* IP inside IP */
#define XF_AH 2         /* AH */
#define XF_ESP 3        /* ESP */
#define XF_TCPPOLICY 5 /* TCP MD5 Signature option, RFC 2358 */
#define XF_IPCOMP 6     /* IPCOMP */
    u_short xf_flags;
#define XFT_AUTH 0x0001
#define XFT_CONF 0x0100
#define XFT_COMP 0x1000
    char *xf_name;        /* human-readable name */
    int (*xf_init)(struct secasvar*, struct xformsw*); /* setup */
    int (*xf_zeroize)(struct secasvar*);                  /* cleanup */
    int (*xf_input)(struct mbuf*, struct secasvar*,
        int, int);    /* input */
    int (*xf_output)(struct mbuf*,
        struct ipsec_request *, struct mbuf **, int, int); /* output */
    struct xformsw *xf_next;                                /* list of registered xforms */
};

static struct xformsw ah_xformsw = {
    XF_AH,  XFT_AUTH, "IPsec_AH",
    ah_init, ah_zeroize, ah_input, ah_output,
};
Authentication

- Verifies that the sender is who they say they are
- Required before adding or using Encryption
AH Reception

- ah_input
- Packet Verification
- Allocate Cryptographic Structures
- Check the mtag
- Setup the cryptographic structure
- Callback
AH Callback

- `ah_input_cb`
- Cleanup after the crypto operation
- Preamble
- Re-acquire embedded state
- Error checks
- Authentication approved
- Send to correct upper layer
IPv6 IPSec Processing Continued

- ipsec6_common_input_cb
- Called once all IPsec work is done
- Preamble
- Various Sanity Checks
- Header Fixup
- enc0 processing
- Protocol Handoff
Encryption

- Hides the contents of the packet from all but the key holder
- Required for a secure tunnel
- Should always be used with authentication
- Shares much boiler plate with AH
ESP Reception

- `esp_input`
- Preamble
- Pull the packet up
- Sequence check
- Crypto Check
- Get Cryptographic Descriptor
- Fill in descriptors
- Dispatch the operation
ESP Callback

- esp_input_cb
- Preamble
- Retrieving data from the descriptor
- Error checks
- Flag mbuf and update the replay sequence
- Strip the header
- See IPv6 IPSec Processing Continued (above)
Security Protocols

Security Section Wrap Up

- IPSec works with both IPv4 and IPv6
- AH for Authentication
- ESP for Encryption
- Heavy use of mtags
- Callbacks used to interact with hardware
- Transformations contain the protocol functions
Questions?