IPv6 Network Processing White Paper

© Copyright 2001 IP Infusion Inc. All rights reserved. IP Infusion, ipinfusion, and ZebOS are trademarks of IP Infusion Inc. All other brands or product names are the property of their respective holders.

Table of Contents

TABLE OF CONTENTS **2**

INTRODUCTION **3**

IPV6 PACKET & ADDRESSING ISSUES **3**

- IPv6 HEADER PROCESSING 4
  - *Extension Headers* *5*
- ICMP V6 7
- ADDRESS RESOLUTION & NEIGHBOR DISCOVERY 8
  - *Neighbor Discovery* *8*
  - *Router Advertisements* *9*
- IPv6 ADDRESSING ISSUES 9
  - *Unicast Addressing* *9*
  - *Special Unicast Addressing* *9*
  - *Multicast Addressing* *10*
  - *AnyCast Addressing* *11*
- IPv6 TUNNELING AND TRANSLATION 11

IPV6 CONTROL PLANE ROUTING SOFTWARE **12**

- NETWORK SERVICES MODULE 14
- ROUTING PROTOCOL MODULES 15

IPV6 CONTROL AND DATA PLANE INTERFACE **16**

- UNICAST FORWARDING TABLE INTERFACE 17
- MULTICAST FORWARDING TABLE INTERFACE 18

DATA PLANE IPV6 PACKET HANDLING **21**

ROUTING OPTIMIZATION IN THE DATA PLANE **21**

© Copyright 2001 IP Infusion Inc. All rights reserved. IP Infusion, ipinfusion, and ZebOS are trademarks of IP Infusion Inc. All other brands or product names are the property of their respective holders.

Introduction

This paper describes some of the high-level issues involved in the integration and optimization of data and control plane components for the construction of IPv6 routing devices in a network environment. This has real world applications in a number of environments, including IPv6 routing and switches devices, storage area networking, devices requiring guaranteed quality of service, as well as a myriad of wireless devices.

The goal of this paper is to outline the issues necessary to construct an IPv6 device from high-speed data and control plane components. We also discuss supporting dual IPv4/IPv6 stacks, tunneling, and translation. We address the following issues in this paper:

- IPv6 Packet & Addressing Issues
- Control Plane Routing Software for IPv6
- Control and Data Plane Interface
- Data Plane IPv6 Packet Forwarding
- Routing Optimization in the Data Plane

IPv6 Packet & Addressing Issues

The structure of the IPv6 packet has been in some ways significantly simplified, but in others will be more difficult for network processing vendors. Specifically the size of the address has been expanded. An IPv6 address is 128 bits long. This means a 256 bit stream for the source and destination addresses alone. The good news is that a number of fields in the IPv4 header have been removed from the IPv6 header (primarily because they are unused or no longer needed today).

For review, an IPv6 address is written as eight 16-bit hexidecimal integers separated by colons. Hexidecimal is used because it is more efficient for such long character strings. For example, an IPv6 address is represented as:

FECE:BC23.0000:0000:0000:0000:CB34:200C

A specific notation may also shorten the addresses. Since the address above has many zeros in the address. We can shorten it to:

FECE:BC23::CB34:200C

The two colons in the middle represent the zeros in the center of the full IPv6 address above. Network prefixes can also be represented by including a slash and the number of bits:

FECE:BC23::/8

© Copyright 2001 IP Infusion Inc. All rights reserved. IP Infusion, ipinfusion, and ZebOS are trademarks of IP Infusion Inc. All other brands or product names are the property of their respective holders.

This might define one level of the aggregator (specifically, the first Network Level Aggregator or NLA). This is similar to the IPv4 subnet concept.

IPv6 Header Processing

The IPv6 header is a simplified header. The first 64 bits of the packet include only 6 parameters. They are:

- Version Field (4 bits)
- Traffic Class (8 bits)
- Flow Label (24 bits)
- Length of the Payload (16 bits)
- Type of Next Header (8 bits)
- Hop Limit (8 bits)

The *Version* field is always set to 6 for IPv6 packets. This is so that IPv4 and IPv6 packets can be distinguished. The IPv6 header is composed of a 64 bit header, followed by the source and destination IP addresses (each 128 bits long). The total length of an IPv6 packet is 40 bytes long. The packet is shown below:

<table>
<tr><td>Version</td><td colspan="2">Traffic Class</td><td colspan="2">Flow Label</td></tr>
<tr><td colspan="2">Payload Length (16 bits)</td><td colspan="2">Next Hdr Type</td><td>Hop Limit</td></tr>
<tr><td colspan="5">Source Address (128 bits)</td></tr>
<tr><td colspan="5">Destination Address (128 bits)</td></tr>
</table>

The *Version* field, as discussed above, is always set to the binary value of 6 to distinguish it from an IPv4 header. A data plane processor may need to distinguish between IPv6 packets and IPv4 packets on the media it is addressing. In an Ethernet MAC environment, IPv6 packets will be sent with an Ethernet content type of 86DD (hex) instead of IPv4's 8000. It is likely that systems nearer the access points of the network might be running IPv6 native, and would not need to worry about running both an IPv4 and IPv6 stack. This would simplify the logic; but for the near future, and especially for edge routing and switching systems, dual stack and/or translation or tunneling of IPv6 packets is probably recommended.

The *Traffic Class* field is the same as the IPv4 TOS (Type of Service), which is now being reviewed as part of a wider “differentiated services” initiative. This is beyond the scope of this paper, but could be important in providing Quality of Services (QOS) initiatives which are key to things like VoIP, VPN's, and wireless services. The DiffServ initiative is defined by RFC2474 and RFC 2475.

The *Flow Label* is used to distinguish packets that require the same treatment (which are sent by a specific source to a specific destination).

© Copyright 2001 IP Infusion Inc. All rights reserved. IP Infusion, ipinfusion, and ZebOS are trademarks of IP Infusion Inc. All other brands or product names are the property of their respective holders.

Perhaps the key field is the *Payload Length* field in IPv6. This defines the length of the data immediately following the header. IPv4 headers have a total length field. This has been eliminated from IPv6. Essentially all of the option fields in IPv4 have also been eliminated from the IPv6 header. Since different headers are needed, the Next Header Type field in IPv6 allows daisy chains of headers to be defined. In a normal packet delivery case (with say a unicast packet), the *Next Heade*r Field would define a UDP or TCP packet. The TCP (6) or UDP (17) packet would immediately follow the standard IPv6 40-byte header.

The *Payload Length* field is a 16 bit field, which limits the packet size to 64K bytes. IPv6 does have a provision for larger packets, with its “jumbogram” option.

As discussed above, the Next Header Type field replaced the IPv4 Protocol *Type* field. In the case of a normal unicast packet, the *Next Header Type* field would be set to indicate a TCP/UDP packet. But IPv6 allows extension headers, which can be inserted between the IPv6 and TCP/UDP payload. The *extension headers* provide flexibility to define additional header types. For *extension headers*, the *Next Header Type* field will be set to the first extension header type. These, in turn can be daisy chained arbitrarily in length. Extension header types will be discussed in the Extension Header section below.

The *Hop Limit* field is basically the same as the IPv4 TTL parameter, but there is no time concept anymore. Now the hop limit is simply decremented per hop. Most routing systems used this concept for the TTL anyway.

The IPv6 header does not contain a checksum field. This simplifies header processing significantly. Most medias provide checksums anyway, so the loss of this field does not increase the risk of data loss, so this simplification comes at little or no cost. The IPv6 fragmentation procedure has been eliminated from IPv6 handling. With IPv6, hosts must learn the maximum segment size by using the path MTU discovery procedure. If an IPv6 packet is greater than the supported segment size, the packet will be rejected. IPv6 networks must support a minimum 1280 byte packet. Hosts that do not support the path MTU discovery option should send minimum sized packets.

Extension Headers

IPv6 specifies 6 *extension header* types. They are:

- Routing
- Fragment
- Security Payload (Encrypted)
- Authentication Destination Options
- Destination Options Header
- Hop-by-Hop Options

Handling extension headers is an important part of the IPv6 experience. These extensions will allow different services to be provided for different applications.

© Copyright 2001 IP Infusion Inc. All rights reserved. IP Infusion, ipinfusion, and ZebOS are trademarks of IP Infusion Inc. All other brands or product names are the property of their respective holders.

In cases where only headers are to be sent with no data payload, the no next header types should end the chain of headers (this is denoted by the value 59).

Routing Header

The Routing Header extension is similar to an IPv4 source address routing option. The Routing Header extension caries a list of either source or loose intermediate routing addresses. The Routing Header has the following format:

- Next Header (8 bits) – This indicates the type of the next header after this routing header in the chain of headers
- Length of extension header (8 bits) – The number of 64 bits words in the header
- Routing type (8 bits) – This defaults to 0. Other values may be used in the future.
- Number of segments not yet processed (8 bits)
- Reserved Field (32 bits)
- IPv6 Address 1 … IPv6 Address N

The routing extension header parameter fields are followed by up to N IPv6 addresses; these specify the source routes to be used on the path to the destination. Each router that receives a packet with routing extension headers only processes the packet if it determines that the destination address in the main header is an address the router recognizes. Routers that are not explicitly mentioned in the source route forward the packet without processing the routing extension header.

If a router determines that it should process the routing extension header, it looks at the Segments Left field to determine if it is the end station. If there are no segments left, then the packet is at its final destination and the router skips over the routing header and processes the Next Header field as it normally would.

If the router is to process the routing header and perform source routing, the router swaps the destination field and the next element in the address list, then decrements the Segments Left field and forwards the packet normally.

Fragment Header

Although IPv6 does not support normal fragmentation, as does IPv4, there is a method for handling packets that are greater in size than the next routers MTU. This method allows IPv6 packets to be fragmented before they are sent on an IPv6 network that would normally reject the packet. When these larger packets are required (for instance by UDP), the large data packet can be sent in a set of IPv6 packets using the Fragment extension header.

The initial station performs the fragmentation, not the network, so that routers do not have to normally process this extension header. The final destination station performs reassembly.

© Copyright 2001 IP Infusion Inc. All rights reserved. IP Infusion, ipinfusion, and ZebOS are trademarks of IP Infusion Inc. All other brands or product names are the property of their respective holders.

Security Headers

There are two types of security headers in an IPv6 header. They are encrypted security payload (ESP) and authentication (AH) headers. ESP headers provide authentication and in addition encrypt the payload. An AH header simply provides a method to validate that the IPv6 packets indeed comes from the specific station, but the payload is in plain text.

The authentication header is composed of a 64 bit header which contains the number of the next header in the chain, the length of the header in 32 bit units, a reserved 16 bit field which should be set and left to zero, and a 32 bit Security Index. This 64 bit fixed header should be followed by the authentication data, coded as a number of 32 bit words.

The ESP header begins with a security parameter index (32 bits) and then contains the encrypted data. The syntax of the data depends on the encryption algorithm.

Destination Options Header

IPv6 contains a single header type denoted by the hex value 60 that defines a destination options header. This defines a generic header type. The destination address node will only examine this header. This keeps router header processing simpler and faster (which is the same reason why the fragment header is not to be processed by routers).

Hop-by-Hop Options

A header type of 0 specifies a hop-by-hop extension header. This is a header that is meant to be used to send additional information (such as management or debugging functions) to routers. If the next header is null, then the router is to process this header even if it does not recognize the destination address as one of its local addresses.

The hop-by-hop option is similar in format to the destination options header. One use of this option is for a jumbo payload to be sent. If the jumbo payload option (type 194) is used, then a packet could be sent whose length is greater than could be sent with the normal 16-bit payload field. When used, the normal length field is set to zero. The node processing this jumbo packet will determine the actual packet length as a 32-bit integer. To make it easier to process this length field aligning the 194 option field as 4n+2 should align it on a 32-bit word boundary.

ICMP v6

The Internet Control Message Protocol for IPv6 is a much more simplified protocol versus its IPv4 cousin. The ICMP v6 protocol now also has the IPv4 Group Membership Protocol (IGMP) included to allow for multicast features. The ICMP v6 protocol is not compatible with the ICMP v4 protocol.

© Copyright 2001 IP Infusion Inc. All rights reserved. IP Infusion, ipinfusion, and ZebOS are trademarks of IP Infusion Inc. All other brands or product names are the property of their respective holders.

ICMP v6 messages contain a 32-bit header followed by a ICMP body (variable length). The header contains the ICMP message type (58), a code field, and a checksum (16 bits). A pseudo header includes the IPv6 source and destination addresses. There are six ICMP message types:

- Destination Unreachable (1)
- Packet Too Big (2)
- Time Exceeded (3)
- Parameter Problem (4)
- Echo Request (128)
- Echo Reply (129)

There are other ICMP message types defined in the neighbor discovery and for multicast purposes.

Address Resolution & Neighbor Discovery

Address Resolution in IPv6 land is different than IPv4 because we now rely on a neighbor discovery and router discovery algorithm rather than the simple ARP of IPv4.

Neighbor Discovery

Neighbor Discovery is defined as part of ICMP v6. With neighbor discovery, there is no need to define a different ARP for each media type. When media addresses of the destinations are unknown, multicast is used to transmit the messages. The media-specific multicast address is used for these multicast addresses. For Ethernet, the media specific multicast address can be constructed by concatenating a fixed 16 bit prefix, 333 hex, and the last 32 bits of the IPv6 multicast address. Other media require different construction algorithms.

There are three types of caches one must employ:

- Destination Cache – This cache has an entry for each destination address where the host has recently sent packets towards. The IPv6 address of the destination is associated with the neighbor towards which the packets were sent.
- Neighbors Cache – This cache has an entry for the adjacent neighbors where the packets were sent (such as a router). The IPv6 address of each neighbor is associated with the media address used.
- Forwarding Cache – This cache contains entries learned from routing table updates

Neighbor entries will be flushed from the cache if they have not been updated recently. In this case an ICMP type 135, or neighbor solicitation message must be sent. This message contains the solicited address. This message is a multicast message and the host that receives this message (including routers) and recognizes the target address will respond back with a neighbor advertisement message (ICMP type 136). The soliciting host will update the neighbor cache with the responding hosts address.

© Copyright 2001 IP Infusion Inc. All rights reserved. IP Infusion, ipinfusion, and ZebOS are trademarks of IP Infusion Inc. All other brands or product names are the property of their respective holders.

Router Advertisements

Routers should periodically send router advertisement messages (message type 134). These messages provide the following information: max hop limit, router lifetime, reachable time, reachable transmission timer, source link layer option, and mtu option. Hosts know they can send messages "offnet" if they receive any router advertisements.

IPv6 Addressing Issues

This section discusses the architecture of an IPv6 address. There are a number of address types including: Unicast, Special Unicast, Multicast, and Anycast.

Unicast Addressing

IPv6 addresses use the same subnet concept as IPv4 addresses, but there is nearly no limit to the number of components of arbitrary size. This allows the highest layers of the hierarchy to be addresses by a wide "subnet" prefix and in theory should decrease the route table requirements that are currently a problem for the backbone and metropolitan area routers. If not dealt with, these problems will push further out to the edge of the network hierarchy of routing and switching devices.

Aggregatable global unicast addresses are composed of the 3-bit prefix 001, a Top Level Aggregator (TLA), some reserved space (8-bits), a Next Level Aggregator (NLA), a Site Level Aggregator (SLA), and the Interface-ID (kind of what used to be in IPv4 land the host portion of the address). The entire address is shown in the table below:

Bit Length	3	13	8	24	16	64
Address	001	TLA	reserved	NLA	SLA	Interface ID

The nice thing about this address specification is that the NLA can be further subdivided by long-haul and exchange carriers any way they desire.

Special Unicast Addressing

There are five special unicast addresses. They are:

- Unspecified
- Loopback
- IPv4
- Site Local
- Link Local

An unspecified address is one with 16 null bytes. This address should be used by a station that does not yet know its IPv6 address. Any packets it sends should have the unspecified value in the source field. Never use it as a destination address. The unspecified address could also be used in control messages that need something in the address field, but the address field will not be in fact analyzed.

© Copyright 2001 IP Infusion Inc. All rights reserved. IP Infusion, ipinfusion, and ZebOS are trademarks of IP Infusion Inc. All other brands or product names are the property of their respective holders.

The loopback address ::1 can be used by a station to send an IPv6 message to itself.

The IPv4 address type is simply a 96 bit null prefix appended with an IPv4 address. For instance the address:

::192.1.168.1

is a valid IPv6 address.

Site Local addresses are similar to net 10 IPv4 addresses, in that they are only understood within the site, and are not official internet IPv6 addresses. A Site Local prefix, 1111111011 has been reserved for these organizations. Obviously these addresses are not routable on the Internet.

A Link Local address is given to a station that is not yet configured with a Site Local or globally unique IPv6 unicast address. These addresses are configured with a reserved link local prefix 1111111010, a set of zeros, and the host portion (station ID), which must be unique within that local network.

When an interface first comes up, the host should define a link local address for this interface. To do this, the reserved link local prefix should be concatenated with a unique value (for instance the 48-bit Ethernet address of the interface). A check should be made to determine that this is in fact unique on the local network.

Multicast Addressing

Multicasting is critical to the Internet and has been designed into the IPv6 protocol. Multicast addresses have the following format:

1 1 1 1 1 1 1 1	Flags	Scope	Group ID

The prefix 11111111 (8 bits) signifies a multicast IPv6 address. The Flags and Scope fields are both 4 bits each, and the Group ID is 112 bits. The fields have the following meanings:

- Prefix 11111111 signifies a multicast address
- Flags field is defined as a 4-bit field, with value 000T, where the T is the transient indicator. The Transient indicator implies that the Group ID is a transient multicast group that will end when the multicast session is terminated.
- The Scope field is a 4-bit field used to define the scope of the multicast group. This limits the extent of the multicast group. Most of the values of the scope field are unassigned, with the following exceptions:
  - 0 is reserved
  - 1 defines a local node multicast group
  - 2 defines a Link local multicast group
  - 5 defines a Site Local multicast group
  - 8 defines a local organization multicast group
  - E defines a Global multicast group

© Copyright 2001 IP Infusion Inc. All rights reserved. IP Infusion, ipinfusion, and ZebOS are trademarks of IP Infusion Inc. All other brands or product names are the property of their respective holders.

Routers must maintain knowledge of which link is associated with which site and organization in order that the site and organization scopes be correctly identified.

There are fixed multicast groups that can be used with the scope field to define groups of like servers. For example, let's say we want to send a message to all servers in Site B running Service A. From within Site B we would send a message with destination FF05::(group ID for Service A). For NTP the group ID would be 43 hex; so the destination address would be FF05::43. This would send an NTP message to all NTP servers in the same site.

IPv6 allows for four permanent group identifiers. They are:

1. Group ID 0 is reserved
2. Group ID 1 is for all the IPv6 addresses on the current node. Using FF01::1 identifies all nodes on this node. Using FF01::2 identifies all nodes on this link.
3. Group ID 2 is for all IPv6 routers addresses. Again as above, FF01::1 implies all addresses on this router. FF01::2 means all routers in this link.
4. Group ID 10000 (hex) means all DHCP servers.

The multicast address FF02::1:0:0 to FF02::1:FFFF:FFFF is reserved for ARP.

AnyCast Addressing

An Anycast address is one in which a packet is sent to one address and each server of a specific type recognizes and processes that packet. The routing infrastructure must deliver these packets to the closest of these servers. Anycast addresses are ideal for finding services on the network. Anycast addresses have the following format:

Subnet (64 bits) composed of an n bit subnet prefix and 64-n bits of subnet pad.
Unique Identifier with a 57 bit reserved prefix (11111101….1111) and 7 bits of Anycast ID.

The main issue for routers is that they have to maintain a unique route for each anycast address that is active in a subnet.

IPv6 Tunneling and Translation

Interoperability between IPv4 and IPv6 protocols can be complex. A number of approaches have emerged to deal with the problem of interoperability and translation between IPv4 and IPv6. These mechanisms include:

- A Dual Stack Approach – Dual IP Stacks are employed for interoperability
- Configured and Automatic Tunnels – Tunnels are manually or automatically configured to allow exchange of IPv6 hosts over a network
- Tunnel Brokers – Tools that allow the set up of IPv6 tunnels
- 6to4 - IPv6 domains are connected over an IPv4 network using a tunnel with IPv4 endpoints

© Copyright 2001 IP Infusion Inc. All rights reserved. IP Infusion, ipinfusion, and ZebOS are trademarks of IP Infusion Inc. All other brands or product names are the property of their respective holders.

- Socks64 - A system accepts SOCKS connections from IPv4 hosts and relays it to IPv4 or IPv6 hosts
- Stateless IP/ICMP Translation - A method of the translation of the IP packet header between IPv4 and IPv6
- NAT-PT - Translation is done between IPv4 and IPv6 using a method similar to IPv4 NAT
- Bump-In-the-Stack – Modules added to the IPv4 stack allow for IPv4 applications to communicate with IPv6 hosts.
- Dual Stack Transition Mechanism (DSTM)– A IPv4 address is requested of the DSTM server and communication occurs on top of IPv6
- TCP-UDP Relay – IPv4 and IPv6 hosts communicate at the transport layer

Many of these approaches are still in development and have not been fully standardized as of yet.

There will both be native IPv6 implementations (IPv6 end-to-end) and IPv6 implementations that use existing IPv4 infrastructure.

Although the state of the art of IPv6 tunneling is still being developed, we suggest using an approach such as 6to4 (RFC3056) to allow IPv6 to be tunneled over an IPv4 backbone. Essentially an edge router accepts and processes an IPv6 packet and encapsulates it into an IPv4 packet that is destined for another edge router that strips off the IPv4 header and reinserts the IPv6 packet and its associated data back onto the network headed for the IPv6 destination.

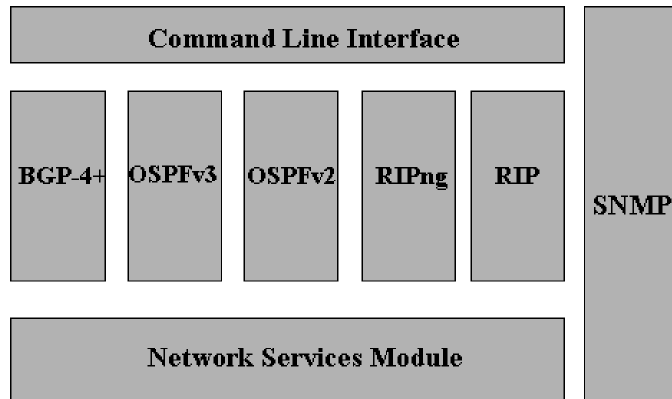
The SIIT (Stateless IP/ICMP Translation Algorithm, RFC 2765) defines a way to translate between IPv4 and IPv6 packet headers that lets IPv6 hosts communicate over an IPv4 based router network.

IPv6 Control Plane Routing Software

The data plane handles IPv4 and IPv6 forwarding when the destination addresses are already in the Forwarding Information Base (FIB). If the destination address is not in the FIB, routing protocols must be used to determine the next hop route so that the packet can eventually be delivered to the correct station. This section discusses IPv6 routing protocols and how they operate in the control plane using the ZebOS Advanced Routing Suite set of IPv6 protocols. Since it is likely that in tunneling/translating edge devices will need to handle both IPv6 and IPv4, the IPv4 routing protocols will also be discussed in this section.

The diagram below shows the architecture of the ZebOS Advanced Routing Suite.

© Copyright 2001 IP Infusion Inc. All rights reserved. IP Infusion, ipinfusion, and ZebOS are trademarks of IP Infusion Inc. All other brands or product names are the property of their respective holders.



ZebOS Advanced Routing Suite Architecture

There are three distinct pieces to the control plane routing architecture. The first is the ZebOS Network Services Module (NSM). The NSM is a standard-compliant routing software package that interfaces to an OS or to a packet forwarding data plane, handles route redistribution, provides management functionality for the entire routing suite, and manages the route table. The second part of the ZebOS Advanced Routing Suite is the routing protocols themselves. IP Infusion provides five modular routing suites, including:

- IPv6 routing modules: OSPFv3, BGP-4+, and RIPng
- IPv4 routing modules: OSPFv2, BGP-4, and RIPv1/v2

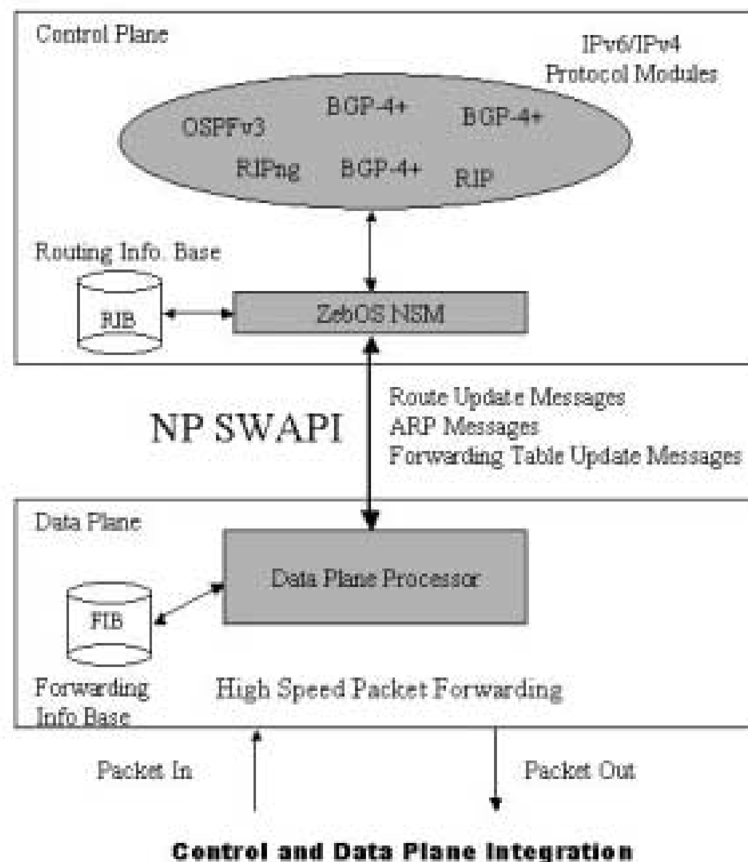
These protocol modules are implemented as separate processes and therefore can be added or removed from any specific control plane design. For core or edge routers, the entire suite may be needed. For some wireless or access devices, only a portion of the suite may be needed. The BGP-4 module includes BGP-4+ extensions for IPv6. This is the only routing module that has both IPv4 and IPv6 processing capabilities. Both OSPF and RIP have separate modules for IPv4 and IPv6 processing.

The third piece is the command line interface and SNMP management facility. ZebOS has a Cisco-like command line interface that is easy and intuitive for those that understand the Cisco command line interface structure. This makes it easy for IT personnel to configure ZebOS since many of them already know the Cisco command line interface structure. The SNMP management feature allows SNMP data to be retrieved for management purposes from all of the protocol modules.

© Copyright 2001 IP Infusion Inc. All rights reserved. IP Infusion, ipinfusion, and ZebOS are trademarks of IP Infusion Inc. All other brands or product names are the property of their respective holders.

Network Services Module

The Network Services Module (NSM) is the base module that binds the protocol modules for IPv4/IPv6 in the control plane with the data plane through a network processing (NP) API. This API will be discussed in the next section. The NSM also provides Route Redistribution, Route Conversion, and Route Table Management. The NSM manages the RIB (route information base) that contains all of the routes available in this router. The NSM makes calls through the Network Processor (NP) API to the forwarding/data plane to fill the Forwarding Information Base with next hop information. The diagram below shows this interaction.



Control and Data Plane Integration

Both IPv6 and IPv4 packets would come in to the Data Plane and be processed accordingly. Although this application note concerns IPv6, it applies equally to both IPv4 and IPv6 packet handling in the data plane. In fact, many IPv6 applications require that dual stacks be implemented. When a normal unicast packet arrives into the data plane engine, the IPv6 header is examined and if the destination address is in the Forwarding Information Base (FIB), then the packet is sent out the correct router interface. In case the address in unknown but local, an ARP message is generated back on the media to find the correct MAC address for the station. ARP may be processed in the control plane, but could also optimally be processed in the data plane itself. If the address is unknown but not on the local subnet, then a message is passed

© Copyright 2001 IP Infusion Inc. All rights reserved. IP Infusion, ipinfusion, and ZebOS are trademarks of IP Infusion Inc. All other brands or product names are the property of their respective holders.

to the control plane software via the NP API, to check the Routing Information Base (RIB) to determine if the route is available. If it is, ZebOS sends a message to the data plane via the NP API to provide the updated route information. The data plane software updates the FIB and sends the packet to the correct local interface for processing. ARP processing may be required to determine the station address of the next hop router.

In addition, routing packets for OSPF, BGP, and RIP are also passed by the data plane back to the control plane for processing. Routing updates from the control plane are sent to the data plane so that they can be sent out the appropriate media.

The NSM and ZebOS in general supports or will support a number of Operating Systems, including:

- VxWorks
- Linux
- BSD
- OS-9
- QNX
- OSE

A real-time OS porting feature is available to port ZebOS to a variety of real-time platforms. Since ZebOS has a Posix compliant interface, it is easily compiled on most Unix systems.

Routing Protocol Modules

A number of Routing Protocol Modules are available that provide full IPv6 (as well as IPv4 routing) support for the control plane. Protocol Modules are available for BGP, OSPF, and RIP for both IPv4 and IPv6. This section discusses each of the IPv6 routing protocols.

BGP-4 Routing Module

The BGP-4+ protocol module provides complete IPv6 and IPv4 routing support in a single module. The BGP-4 module supports IPv4 and the BGP-4+ extensions are provided for IPv6 support. The BGP-4+ module communicates with the NSM to provide full BGP routing support. A number of RFC’s are supported, including:

- RFC 1771 Border Gateway protocol version 4
- RFC 1772 Applications of BGP-4 in the Internet
- RFC 2439 Route Flap Damping
- RFC 1965 Autonomous System Conferderations for BGP
- RFC 1997 BGP Communities Attribute
- RFC 2545 BGP Multiprotocol Extensions for IPv6 InterDomain Routing
- RFC 2547 MPLS BGP VPN Support
- RFC 2796 BGP Route Reflection – An Alternative to Full Mesh IBGP
- RFC 2842 Capabilities Advertisement with BGP-4
- RFC 2858 Multiprotocol Extensions for BGP-4

© Copyright 2001 IP Infusion Inc. All rights reserved. IP Infusion, ipinfusion, and ZebOS are trademarks of IP Infusion Inc. All other brands or product names are the property of their respective holders.

This module also supports route reflection, MBGP, route aggregation, soft reconfiguration, update source configuration, route refresh, capability negotiation, AS path access-lists, peer groups, and full SNMP MIB get/set support.

OSPF Routing Module

Unlike the BGP-4+ module, the IPv6 version of OSPF is a separate module than the IPv4 version. The IPv4 version of OSPF is known as OSPFv2. The IPv6 version of OSPF is known as OSPFv3. The table below outlines this info:

IPv4 OSPF module name	IPv6 OSPF module name
OSPFv2	OSPFv3

IP Infusion's ZebOS supports both modules. The OSPFv3 IPv6 module supports the main IETF RFC 2740 (Open Shortest Path First version 3 for IPv6 support). In addition the OSPFv3 module supports route redistribution with type, the OSPF Hello parameter configuration, and the OSPF interface configuration feature. The OSPFv2 for IPv4 supports the main RFC 2328 (Open Shortest Path First version 2 for IPv4 support).

RIP Routing Module

The RIPng protocol module delivers IPv6 RIP support. The RIPng module fully supports RFC 2080 (Routing Information Protocol Next Generation). This includes support for static routes and basic timer configuration. The RIP protocol module delivers IPv4 support, but like OSPF the RIP and RIPng protocol modules are separate. Both the RIP and RIPng protocol modules are ideal for simple access devices that don't need sophisticated routing protocol support.

IPv6 Control and Data Plane Interface

While the data plane is responsible for processing and forwarding IPv6 packets, it is up to the control plane to provide support for the interior and exterior routing protocols. The ZebOS Network Services Module (NSM) manages the Routing Information Base (RIB) and provides services to the Data Plane to insure that as many packets as possible are forwarded directly by the Data Plane. In order to forward packets directly, the data plane contains a Forwarding Information Base (FIB). One can think of the FIB as cached routes. The control plane routing protocols and the NSM are responsible for filling the data plane FIB, as well as providing other services. This section outlines a Network Processor API that has been defined by IP Infusion. Our intention is to merge this API with the API being developed in the software working group of the Network Processing Forum (www.npforum.org).

Note that the functions defined below are examples of how a network processing interface for forwarding table updates might be designed. This specification has not been complete, and will require some work to be fully specified. But we believe the outline below is a good example of the kinds of features/functions that will be needed.

© Copyright 2001 IP Infusion Inc. All rights reserved. IP Infusion, ipinfusion, and ZebOS are trademarks of IP Infusion Inc. All other brands or product names are the property of their respective holders.

Unicast Forwarding Table Interface

We outline three functions that need to be defined for the control plane to data plane interface to update the forwarding table for unicast calls. These calls have been generalized. Separate calls would need to be provided for both IPv6 and IPv4 unicast forwarding updates. The three NP (Network Processing) API calls are shown below:

npfIPUCForwardTableUpdate(ucForwardTableAction theAction, ucForwardTableHandle theForwardingTable, ipUCForwardTableEntry theEntry);

- ·Arguments
  - ·ucForwardTableAction theAction = the action to take for the update : add (or replace), delete
  - ·ipvUCForwardTableHandle theForwardingTable = the handle for the forwarding table to update.
  - ·ipv4UCForwardTableEntry theEntry = the entry for the update
- ·Returns
  - ·(if request submitted) int = nonnegative
  - ·(if request not submitted) int = negative (indicates what error occurred on submission)
- ·Description (add/replace)
  - ·If there is a matching entry in the forwarding table, this will update that entry, completely replacing the previous entry with the new data, including all nextHop entries, even if the old entry has a different number of nextHop entries than the new entry.
  - ·If there is no matching entry in the forwarding table, this adds an entry to the forwarding table, including all given nextHop entries.
- ·Description (delete)
  - ·If there is a matching entry in the forwarding table, that entry is completely deleted.
  - ·If there is no matching entry in the forwarding table, this will do nothing.
- ·Notes (general)
  - ·This call is made by the protocol or routing/forwarding table manager.
  - ·Matching is done only on the address and mask; nextHop entries are not considered when matching an entry in this function.
  - ·Error returns after request submission are asynchronous as described by the NPF SwAPI Foundations specification.
  - ·The type of forwarding entry must match the type of forwarding table.
  - ·If operating in a table-based architecture, this should be used to pre-download any known data on initialization, and to update data as they are learned.
  - ·If operating on a cache-based architecture, this should be used as part of the response to a forwarding table miss callout, described below.

npfIPUCForwardTableRequest(ucForwardTableReqAction theAction, ipUCForwardTableHandle theForwardingTable, ipUCForwardTableContext theContext, byte *theFrame, unsigned int length);

- ·Arguments
  - ·ucForwardTableRequest theAction = indicates what action is being requested by the forwarding plane : table/cache miss is the only defined item at this time.
  - ·ipUCForwardTableHandle theForwardingTable = the handle for the forwarding table which was used when the miss occcurred
  - ·ipUCForwardTableContext theContext = context established when the forwarding table was initially set up
  - ·byte *theFrame = pointer to the frame which caused the miss, NULL if not applicable
  - ·unsigned int length = length of the frame which caused the miss, in bytes, zero if not applicable

© Copyright 2001 IP Infusion Inc. All rights reserved. IP Infusion, ipinfusion, and ZebOS are trademarks of IP Infusion Inc. All other brands or product names are the property of their respective holders.

·Returns
·(Not defined at this time)
·Description (table/cache miss)
·In a cache based architecture, this will scan the appropriate (according to the context and forwarding table handle) control plane forwarding or routing table for the correct entry for the frame, update that entry to the forwarding plane, and then either resubmit the frame to the forwarding plane for forwarding or forward the frame directly. If there is no appropriate entry, the control plane is expected to emit the proper response for the type of frame (for example, ICMP unreachable).
·In a table-based architecture, this indicates a forwarding failure (no route). The control plane may behave as for cache architectures (recommended), or it may elect to simply emit the proper response for the type of frame.
·This request includes a frame and frame length.
·Notes (table/cache miss)
·This call is made when the forwarding plane has received a frame which it cannot forward. The forwarding plane has already tried to forward the frame using all forwarding tables or caches which apply to the given context before this is dispatched. However, cache based systems may not have the entire table available to the forwarding plane and may require the control plane to scan the full table, just to be sure.
·A sophisticated forwarding plane may be capable of emitting these responses itself, so it is possible that a complex table-based forwarding plane may never make this request against the control plane.
·Notes (general)
·Error returns after request submission are asynchronous as described by the NPF SwAPI Foundations specification.

npfIPUCForwardTableFlush(ipUCForwardTableHandle theForwardingTable);
·Arguments
·ipUCForwardTableHandle theForwardingTable = the handle for the forwarding table to flush
·Returns
·(if request submitted) int = nonnegative
·(if request not submitted) int = negative (indicates what error occurred on submission)
·Description
·This will purge all entries from a forwarding table, resulting in any frame which goes to that table being considered a miss, which will be handled using npfForwardTableMiss, above.
·Notes
·This call is made by the protocol or routing/forwarding table manager.
·This may, on some forwarding hardware, require explicit addition of a rule to default forward frames to the control plane for processing. This must be handled within this call.
·A forwarding table will be initialized to this state on creation.
·Error returns after request submission are asynchronous as described by the NPF SwAPI Foundations specification.

An IPv6 and IPv4 unicast forwarding table entry and Next Hop address entry data structure would need to be defined.

Multicast Forwarding Table Interface

We also need to define a set of multicast forwarding table functions. As in the Unicast case we outline three functions that need to be defined for the control plane to data plane interface to update the multicast forwarding table. These calls have been

© Copyright 2001 IP Infusion Inc. All rights reserved. IP Infusion, ipinfusion, and ZebOS are trademarks of IP Infusion Inc. All other brands or product names are the property of their respective holders.

generalized. Separate calls would need to be provided for both IPv6 and IPv4 multicast forwarding updates. The three NP (Network Processing) API calls are shown below:

npfIPMCForwardTableUpdate(mcForwardTableUpdAction theAction, ipMCForwardTableHandle theForwardingTable, ipMCForwardTableEntry theEntry);

- ·Arguments
  - ·mcForwardTableUpdAction theAction = the action for the update : add (or replace), delete, conditional delete
  - ·ipMCForwardTableHandle theForwardingTable = the handle for the forwarding table to update
  - ·ipv4MCForwardTableEntry theEntry = the entry for the update
- ·Returns
  - ·(if request submitted) int = nonnegative
  - ·(if request not submitted) int = negative (indicates which error)
- ·Description (Add/Replace)
  - ·If there is a matching entry in the forwarding table, this will update that entry, completely replacing the previous entry with the new data, including all nextHop entries, even if the old entry has a different number of nextHop entries than the new entry.
  - ·If there is no matching entry in the forwarding table, this will insert the specified entry.
- ·Description (Delete)
  - ·If there is a matching entry in the forwarding table, this will remove it, including all nextHop entries.
  - ·If there is no matching entry in the forwarding table, this will do nothing.
- ·Description (Conditional Delete)
  - ·If there is a matching entry in the forwarding table that **has not** been used since it was last updated (or since the last conditional delete request against it), that entry will be deleted. The forwarding plane will be sent a message (using the method specified in the NPF SwAPI Foundations document) indicating that the delete was successful.
  - ·If there is a matching entry in the forwarding table that **has** been used since it was last updated (or since the last conditional delete request against it), that entry will not be deleted. The forwarding plane *may* be sent a message (using the method specified in the NPF SwAPI Foundations document) indicating that the delete was not successful.
  - ·If there is no matching entry, this will behave as if a successful conditional delete was performed.
- ·Notes (general)
  - ·This call is made by the protocol or routing table manager.
  - ·Matching is done only on MC source and group; nextHop entries are not considered when matching an entry in this function.
  - ·Error returns after request submission are asynchronous as described by the NPF SwAPI Foundations document.
  - ·If operating on a table-based architecture, this should be used to pre-download any known data on initialization and to update data as they are learned.
  - ·If operating on a cache-based architecture, this should be used as a part of the response to a forwarding table miss callout, descirbed below.

npfIPMCForwardTableRequest(mcForwardTableReqAction theAction, mcForwardTableHandle theForwardingTable, mcForwardTableContext theContext, byte *theFrame, quadByte length);

- ·Arguments
  - ·mcForwardTableReqAction theAction = indicates what is being requested by the forwarding plane : table/cache miss, entry expired
  - ·mcForwardTableHandle theForwardingTable = the handle assigned to the forwarding table being referenced
  - ·mcForwardTableContext theContext = the context established when the forwarding table was set up

© Copyright 2001 IP Infusion Inc. All rights reserved. IP Infusion, ipinfusion, and ZebOS are trademarks of IP Infusion Inc. All other brands or product names are the property of their respective holders.

·byte *theFrame = pointer to the frame which caused the request if it is applicable (NULL if not)
·quadByte length = length of the included frame if applicable (zero if not)

·Description (table/cache miss)

·In a cache based architecture, this will scan the appropriate (according to the context and forwarding table handle) control plane MC forwrding or routing table for the correct entry for the frame, update that entry to the forwarding plane, then either resubmit that frame to the forwarding plane for forwarding or directly forward the frame. If there is no appropriate entry, the control plane is expected to emit the correct response to the sender.
·In a table-based architecture, this indicates a forwarding failure (no entry). The control plane may behave as for cache architecture (recommended), or it may simply elect to emit the proper response.
·This will include the frame that caused the miss.

·Notes (table/cache miss)

·This call is made when the forwarding plane receives a frame which it can forward, but for which it has no matching entry. The forwarding plane has already searched appropriate tables or caches for this MC source/group and was unable to find it. A cache-based system may require the control plane to search its tables as well before considering this a forwarding failure.

·Description (entry expired)

·This is an informational message to the control plane indicating that the forwarding plane has not seen a frame matching this entry in the appropriate period of time, and that the forwarding plane is deleting the entry automatically.
·This does not include an initiating frame, since it is exactly the lack of frames using the entry that triggered this request.

·Notes (entry expired)

·This call is made by a forwarding plane capable of managing such timers as needed, to indicate that an entry has not been used for its TTL and was therefore purged.

·Notes (general)

·Error returns after request submission are asynchronous as described by the NPF SwAPI Foundations specification.

npfIPMCForwardTableFlush(mcForwardTableHandle theForwardingTable);

·Arguments

·mcForwardTableHandle theForwardingTable = the handle for the MC forwarding table to flush

·Description

·This will purge all entries from the specified MC forwarding table, resulting in any frame which goes to that table being considered a miss and handled using npfMCForwardTableRequest(miss,...) above.

·Notes

·This call is made by the protocol or routing table manager.
·This may, on some forwarding hardware, require addition of a rule to default forward frames to the control plane for processing. If this is the case, it must be done by this function.
·An MC forwarding table will be in this state on creation.
·Error returns after request submission are asynchronous as described by the NPF SwAPI Foundations specification.

An IPv6 and IPv4 multicast forwarding table entry and next hop address entry data structure would need to be defined.

Anycast forwarding table calls and entries would also need to be defined, but these would be similar to the unicast/multicast interface calls.

A set of calls would also need to be defined to allow the control plane routing software to send and receive normal IPv6 packets (route updates and other informational

© Copyright 2001 IP Infusion Inc. All rights reserved. IP Infusion, ipinfusion, and ZebOS are trademarks of IP Infusion Inc. All other brands or product names are the property of their respective holders.

messages). These would essentially be normal unicast/multicast IPv6 packets that the data plane would forward appropriately.

Data Plane IPv6 Packet Handling

In order for network processors to handle IPv6 packets, a number of issues we have discussed in the sections above must be dealt with. This includes:

- IPv6 header processing
- IPv6 address processing (unicast, multicast, anycast)
- ICMP v6 processing
- Neighbor and Router Discovery and Advertisements
- IPv6 tunneling and translation functions
- Processing of the NP API control plane to data plane interface calls

Routing Optimization in the Data Plane

As routing updates and unknown addresses are processed by the data plane hardware, these packets must be moved to the control processor for route information to be ultimately passed back to the forwarding plane. This slows down packet processing. There are a number of optimizations, which could occur, that would eliminate the need to pass all of these packets to the control processor. Although beyond the scope of this paper, these optimizations include: Moving high profile routing components to the data plane (things like OSPF hello processing) as well as moving ARP, router discovery, ICMP, and other high activity control message processing to the data plane. This may significantly improve normal routing and forwarding performance.

IP Infusion Inc.
111 W. St. John Street
Suite 910
San Jose, CA 95113
Tel: 408.794.1500
Fax: 408.278.0521
info@ipinfusion.com
www.ipinfusion.com

© Copyright 2001 IP Infusion Inc. All rights reserved. IP Infusion, ipinfusion, and ZebOS are trademarks of IP Infusion Inc. All other brands or product names are the property of their respective holders.