Building Dual Stack IPv4 / IPv6 Router On Linux

LinuxCon Japan, Yokohama, June 6-8, 2012

Mario Smarduch
mario.smarduch@huawei.com
Chief Architect, Embedded Virtualization

Timo Jokiaho
timo.jokiaho@huawei.com
CTO, System Software

Huawei Central R&D
European Research Center
Munich, Germany
This Session Will Talk about -

- Current IPv4 Residential/SOHO Router Capabilities
  - Key Features ISPs require
- Well known IPv4 Limitations
- Brief IPv6 Background – relevant topics
- Linux Connection tracking, NAT, ALGs in IPv4 and IPv6
- IPv6 Integration of Connection tracking
- CPU Network Offload Linux Integration
- Brief 6rd overview
- Cable eRouter Dual-Stack Implementation
IPv4 Residential/SOHO Router Basic Features

- **Gateway – ISPs view as focal hub for**
  - Cable/IPTV streaming to consumer devices
  - Media Server, NAS/Print server, Home Automation, Guest Access
  - ISP – eventually IPv6 Support required

- **Provisioning - DHCPv4/DNS Client, Server (PPPoE, PPTP, L2TP going away) – Impacts IPv6**
  - IPv6 Provisioning - WAN dhclient6 – IP address; LAN Prefix delegation + host OUI Unique IP and DHCP info request
  - DNS – proxy server/or direct recursive – can handle resolving/caching ‘AAAA’ records

- **Performance**
  - Lowest 116Mbps/DS – 104Mbps/US – both DOCSIS/GPON will push up to 1Gbps
  - Packet routing Zero impact to application Processor – ISPs require Offload engine for predictable growth

- **Network Features**
  - NAT – IPv6 N/A - Whole purpose of IPv6 – global IP space
  - Bridging (i.e. Wireless/LAN, others) – IPv6 no impact
    - Bridged LAN+SSID and/or VLANs – only L2 header matters
  - WDS – depending on topology - at L2 no impact at L3 extra configuration – rare configuration.
  - VLANs (port based/Tagged) – Some IPv6 Impact
    - For routed vlan interfaces – Typical IPv6 Provisioning – LL, Prefix+OUI, DAD, IPv6 Routing
IPV4 RESIDENTIAL/ SOHO BASIC ROUTER FEATURES

- Basic Wireless Networking (user/guest) SSID – Some IPv6 Impact
  - Similar to VLANs – SSID an interface – guest SSIDs, hot-spots – IPv6 FW rules

- Advanced Wireless Networking (WPA/WPA2 Enterprise) – IPv6 Impact - Maybe
  - For EAP-TLS, EAP-PEAP, EAP-TTLS - Authenticator – maybe use IPv4 Radius – depends on ISP reqs.

- Routing – Heavy IPv6 impact
  - Different L2 → L3 hooks, routing table, cache, policy based routing, configuration differs

- Port Forwarding, DMZ – IPv6 N/A
  - Not needed in IPv6 private network dests can be reached directly, in IPv4 Dest is always GW Public IP

- Multicast routing
  - In IPv4 Gateway manages hosts queries/reports and forwards them on WAN and sets up multi-cast routes
  - Same is needed in IPv6
IPv4 Residential/SOHO Router

Features

- **Security**
  - IP/Port/Protocol Filtering - Same considerations as IPv4, require IPv6 FW rules
  - ALGs
    - Same considerations as IPv4 (with exception of packet rewrite), require IPv6 equivalent ALGs for port opening
  - SPI
    - Same considerations as IPv4 (tracking original/reply directions), require IPv6 equivalent FW rules
  - Port Triggering - impacts IPv6
    - Same considerations as IPv4 (open in-bound port based on configured out-bound port), require IPv6 equivalent impl.
  - UPnP IGD – IPv6 impact

- **Management/Accessibility**
  - SSH, TFTP
    - For management require both protocol types to work
  - Web (HTTP/HTTPS)
    - For UI require access from both protocols
  - SNMP
    - Manage SNMP MIBs for IPv6 as well
  - NTP – may not require IPv6

- **Many other features** - Gateways today do lot more work! - NAT Bypass, VPN Pass-Through, Routed Subnets, Parental Control, TR-69, Wireless Roaming,...
Well known IPv4 Limitations

- **Well known Issues with IPv4** –
  - Small IP Range – NAT implemented to reuse (private address range 10.xx..., 192.168...., 172.16. ...) – Basic NAT operation
  - Example

- **NAT introduces many issues** – Performance – SNAT on way out, DNAT on way in – packet rewrite, IP, TCP checksum update
  - NATs come in different flavors –
    - Symmetric (original dst ip/port only) – most restrictive
    - Full Cone (any ip/port) – least restrictive – can handover connection to other server
    - Restricted Cone NAT (original dst, any port) – handover within server
    - Restricted Port NAT (any dst, orig dst port) – handover across server with same port
    - NAT traversal discovery protocols STUN, TURN, ICE
    - IPv6 – Can connect from any IP/port to private device behind FW (TCP/UDP)
  - Packet Rewrite (aka ALGs) – few protocols affected – complicates processing, affects CPU offload engines – few examples
    - PPTP (uses GRE) – several clients behind FW with same Call ID connecting to Server – must rewrite Call ID (on way out and in)
Well known IPv4 Limitations

- ICMP – Public or Private IP can’t be revealed to client
  - Other important protocols – TFTP, FTP, RTSP, SIP, Kerberos, DNS – used under all sorts of circumstances by ISPs

![Diagram of a network with server, FW/NAT, and client.]
Related IPv6 Background

- **Primarily based on Cable eRouter standard for IPv6 delivery to premises**

- **IPv6 Addressing**
  - In theory IPv6 has $340,282,366,920,938,463,463,374,607,431,768,211,456$ address and IPv4 $4,294,967,296$ in practice
  - 64 bits are used for subnet mask and 64 bits for host, probable to further subdivide the subnet without OUI usage
  - IPv6 addresses can be huge - e.g. **128.91.45.157.220.40.0.0.0.252.87.212.200.31.20** - this would be an IPv6 address
  - **New Notation** – compact hex with 16 bits → **805B:2D9D:DC28:0000:FC57:D4C8:1FF**
  - IPv6 addresses can be huge - e.g. **805B:2D9D:DC28:0000:FC57:D4C8:1FFFF**
  - **Subnet notation CIDR like** → **805B:2D9D:DC28::/48** – routing prefix match is on first 48 bits
  - Address Space Allocation dictated by bit prefixes
    - **Loopback** → ::1/128 – the 0’s collapse to ::
    - **Global Unicast** → 2000::/3
    - **Link-Local unicast** → FE80::/10 – address in range can’t make it outside of subnet
    - **Multicast** → FF00::/8
      - 4-byte defines scope 2 – link local, ..., 8 site local – MC can go beyond local subnet: FF02::1 all nodes MC, ...
      - **Unspecified address** → ::128 – f.e. in DHCP messages when host does not know its IP address
  - **The elegant notation for ‘loopback’ interface comes from here** - 0:0:0:0:0:0:0:1 is reduced to ::1
  - IPv6 address Network/Host; Host is OUI constructed - for MAC = 39-A7-94-07-CB-D0 Host is: 3BA7:94FF:FE07:CB0
  - IPv6 Header is 40 bytes – twice IPv4 size
Relevant IPv6 Background

- **IPv6 Packet Structure** – combination of main header and extension header and option fields
  - It does not have a header checksum
  - Next Header → TCP, UDP, Hop by Hop Header, ESP, AH, ICMPv6
  - Couple Examples – of IPv6 Payload construction
Relevant IPv6 Background

• Additional Extension Headers
  – Hop by Hop – options that can be inspected by each node
  – Destination Options – options targeted for destination
  – Routing Option – similar to LSR in IPv4, also used in conjunction with destination option
  – Fragment Header – illustrated above
  – ICMPv6 – Neighbor & Router Discovery
  – AH, ESP – headers – transport/tunneled mode

• Routers don’t fragment, only hosts – minimum Fragment size changes from 576 to 1280

• Key Changes since IPv4
  – Renamed - Traffic class (IPv4 TOS), hop limit (IPv4 TTL)
  – Payload Length, Next Header, IPv4
  – Added - Flow Label; Removed - Internet Header Length, Identification, Flags (MF, ...), Fragment Offset, header checksum.

Link Configuration - Both Stateful and Stateless

• difference is for stateful address assignment has state associated with it
• Both built upon: Link Local Address, multicasting, ND (ICMPv6), OUI address generation and DHCPv6 (even in stateless)
• Key ICMPv6 Neighbor Discovery Protocol Messages– RS actively solicits for Routers to send RAs – sent to MC address 0xFF02::2
  All routers MC address

ICMPv6 Router Solicitation (RS)

<table>
<thead>
<tr>
<th>Type=133</th>
<th>Code 0</th>
<th>Checksum</th>
<th>........</th>
</tr>
</thead>
</table>

ICMPv6 Router Advertisement (RA)

<table>
<thead>
<tr>
<th>Type=134</th>
<th>Code 0</th>
<th>Hop Limit</th>
<th>AutoConf Flags</th>
<th>Default Router Lifetime in Sec.</th>
<th>Neighbor Reach. Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>NS Transmit Time</td>
<td>OPTIONS:</td>
<td>Router Source Layer Address</td>
<td>MTU</td>
<td>Network Prefix(es)</td>
<td></td>
</tr>
</tbody>
</table>
Related IPv6 Background

- Neighbor Solicitation Messages, for Neighbor disc, DAD, IP addr change – msg types NS, NA
- Target address - 'Unsolicited Node MC Address' - FF02::1:FF<XX:YYYY> - followed by lower 24 bits of target IP address (33:33:FF:XX:YY:YY)
- Only Hosts subscribed to MC address receive – helpful to mobile devices in PM mode.

**ICMPv6 Neighbor Solicitation (NS)**

- Type=135
- Code 0
- Checksum
- Target Address

**ICMPv6 Neighbor Solicitation (NA)**

- Type=136
- Code 0
- Checksum
- Router Flag
- Solicited Flag
- Override Flag
- L2 Target Address

**Brief Overview Stateless Operation**

- Link Local Address is generated - prefix '0xFE80::OUI L2 MAC' when interface is up f.e. fe80::20c:29ff:fece:6446/64
- Address uniqueness test – NS issued, check for DAD
- Issue RS or listen for RA – check M flag for stateless or stateful, O flag for other (typically with stateless)
- Get and install advertised parameters: Network Prefix, Router L2 address for default GW, Router Lifetime, MTU & few ND params

**Stateful Configuration**

- RA from router sets the stateful flag host runs DHCP

**Stateless + Managed**

- Stateless configuration with DHCP to get various network config. params like DNS Server f.e. – common scenario
Related IPv6 Background

- **Primary Attributes of ICMPv6 ND**
  - Ethernet broadcasting eliminated
  - All hosts, All Routers, ND use Multi-Casting – host must be member of at least 3 MC groups – lower 24 bits of IP in solicitation messages used in MAC MC dest – f.e. for LL target Link Local: fe80::20f:3dff:fe7d:cbf MAC message: 0x33.33.FF.7D.0C.BF
  - Target host subscribes to this MC address – it’s the only one that will match
  - Only affected hosts are affected, good Mobile Devices in PM Mode

- **Security is built in – AH, ESP are another options that encapsulate the payload**
  - Allows even securing NS, NA functions – in IPv4 ARP can’t be secured, MIM attacks eliminated
  - Eliminates NAT issues - with several clients behind the FW (when same SPI is used)
  - Related to security – scanning IPv6 address space much harder – larger address space
Related IPv6 Background

- More on Stateful Configuration – More in Linux Dual-Stack GW
- DHCPv6 three primary goals
  - Address Configuration
  - Non-Address Configurable Parameters (like DNS Server, Domain Name, NTP server .....
  - Prefix delegation – provide several prefixes (eRouter implementation)
- Client known as “Request Router”, Server as “Delegating Router”
- DHCP Server and Relay agent MC address: FF02::1:2 – this MC Link scope
  - Used by IPv6 DHCP clients, LL addresses are acceptable in source field for clients, although client may use the unspecified address ::0/128
  - The recipient(s) may be a Server or Relay agent
- All DHCP servers MC address: FF05::1:3 – this site scope
  - Used by relay agents to contact a DHCP server, in this case the relay agent must have non-LL address
- UDP over IPv6, Client uses port 546 and server uses port 547, that is client sends to port 547, server replies to port 546.

![Diagram of DHCPv6 setup]
Related IPv6 Background

- **Architecture Principles**
  - Each subnet should not require a DHCP server, but at least a Relay agent
  - DHCP server in the cloud accessed through the FF02::1:2 relay agent. The relay agent then uses a site local address DHCP server.
  - There can be several DHCPv6 Relay agents in route.
  - Default DHCP client behavior is to send requests to local MC address
  - Clients in IPv6 are known by a DUID (as opposed to IPv4 MAC or client user identifier option)
    - Combination of MAC and some other string
  - For relay – interface identifier used to route back responses (in above figure DHCP Relay disambiguates between LAN Segment 1 or 2).

- **Messaging – many similarities to IPv4**
  - SOLICIT, ADVERTISE – client locate DHCP server, Server Advertises
  - REQUEST, REPLY – client request parameters including addresses, Server replies

- **INFORMATION-REQUEST** – client sends to server - reqst config params without an IP address assignment (F.e. Stateless + Managed)

- **Two and Four Message Exchanges**
  - Two Message – ‘Rapid Commit’ - both client and server need to be configured for ‘rapid commit’
    - The client sends a SOLICIT and internally sets the ‘rapid commit’ option,
    - server responds with REPLY IP and config info
  - Few other scenarios for 2-message exchange – like info request

- **Four Message Exchange – SOLICIT, ADVERTISE, REQUEST, REPLY**

- **DNS in IPv6**
  - New Resource Record ‘AAAA’ added for Forward Lookup – ‘A’ for IPv4
  - Reverse lookup – new reverse tree ‘ip6.arpa’ – ‘in-addr.arpa’ for IPv4
Background on Conn Tracking, NAT, FW for IPv4

- Connection tracking – key concepts – few practical examples will follow
  - Key to understanding stateful, FW, NAT and CPU Offload Engine
  - Clear up confusion on ALGs in IPv6
  - Foundation of Stateful Packet Inspection
  - Conn tracking few key concepts
    - concept of ORIGINAL, REPLY direction
      - Monitor REPLY packet on receive – if FW rules – Filter/FORWARD – ESTABLISHED let through
    - conntrack structure with a hash – associated with each flow/session
    - FW rules integrated with conntracking – facilitates generic stateful packet inspection (i.e. FW rules)
  - NAT – requires conntracking – to manage mapping for SNAT (MASQUERADE SNAT variant), DNAT
  - Basis of ALGs – two parts – although coined as one –
    - Inbound Port opening (for example FTP)
    - Packet rewriting - (also FTP another good example)
  - ALGs build on conntracking
    - Basis of ALGs – helper to monitor packet flow of ALG control stream
    - On match create an expectation, on hit mark as RELATED, - Filter/FORWARD – RELATED let through
  - Network Offload Engine – Several reasons for it
    - New Technologies DOCSIS, GPON – high DL/UL data rates (128,256Mbps)
    - BOM – must be low
    - Preferable CPUs/SMP – built for high data rates into future – CPU(s) dedicated to Apps - QAM tuning/Stream MPEG over IP, DLNA/UPnP, NAS
    - Offload Engine – tightly integrated to conntracking, ALGs
      - ALG Control stream let OS handle
      - Mansge TLU, entry into/out of OS
      - Mange conntrack states like – conntrack timeout
      - Many other – routing table updates, interfaces up/down, ....,
IPv4 Connection Tracking – SPI

- Key Connection Tracking Structures
  - tuplehash – has the original and reply tuples and links for hash table
  - status – and other fields pointers, important later

- Conntrack hash table – can hit in original or reply direction

```
struct nf_contrack_tuple_hash tuplehash[IP_CT_DIR_MAX];
```

```
unsigned long status; // IPS_EXPECTED_BIT, IPS_SEEN_REPLY_BIT,
timeout; // CT lifetime without activity
```

```
struct nf_conn
```

```
struct hlist_nulls_node hnnode
```

```
struct nf_conntrack_tuple tuple;
```

```
struct nf_conntrack_tuple_hash tuplehash[IP_CT_DIR_MAX];
```

```
struct hlist_nulls_node hnnode
```

```
struct nf_conntrack_tuple tuple;
```

```
Ex: IP=10.0.0.2/2001 proto=TCP
```

```
Ex: IP=1.2.3.4/3001 proto=TCP
```

```
dir = DIR_ORIGINAL
```

```
Ex: IP=10.0.0.2/2001 proto=TCP
```

```
dir=DIR_REPLY
```

```
Ex: IP=1.2.3.4/3001 proto=TCP
```

```
dir = DIR_REPLY
```

```
Default Namespace
init_net.ct.hash[]
```

```
original
```

```
reply
```

```
....
```

```
nf_conn
```

```
nf_conn
```

```
original
```

```
reply
```

```
....
```

Mario & TimoJ
IPv4 Connection Tracking – SPI

- Example of stateful firewall, with no NAT -- subset of Network stack Tables and Chains illustrated

Following Rule in Forward Chain:

- Filter Table in Forward Chain matches on skb->nfctinfo = IP_CT_ESTABLISHED and accepts the packet
- After DIR_REPLY statefulness established – means private host opened the FW
- The skb carries status – used by Filter Rules
- In summary generic statefullness means – hit in Conntrack hash on DIR_REPLY packet
- Protocol Statefullness – more extra work i.e. matching packets against TCP state machine
  - F.E TCP – state transitions, seq# ordering, .... – done in PREROUTING Connection tracking
IPv4 Connection Tracking – SPI+NAT

- Extend to NAT with Conntacking – typical scenario client using private IP

```
int ip_forward()
{
    return NF_HOOK(NFPROTO_IPV4, NF_INET_FORWARD,..., ip_forward_finish);

    iptable_filter_hook,()  -
    rule: input dev=wan, skb status ESTABLISHED or RELATED action ACCEPT
}
```

```
int ip_mc_output(struct sk_buff *skb)
{
    return NF_HOOK_COND(NFPROTO_IPV4, NF_INET_POST_ROUTING, ... , ip_finish_output, ..)
}
```

```
1. NAT TABLE/POSTROUTING: nf_nat_out()
   SNAT: rule: dev=wan MASQUERADE (on new pkt only)
```

```
2. CONNTRACK/POSTROUTING: ipv4_confirm()
```

```
ip_rcv()
{
    return NF_HOOK(NFPROTO_IPV4, NF_INET_PRE_ROUTING,  ip_rcv_finish)
}
```

```
2. NAT TABLE/PREROUTING: nf_nat_in()
1. CONNTRACK/PREROUTING: ipv4_conntrack_in()
```

- For Clarity L4 and protocol are not shown
- Following rule installed to NAT table (in addition to Filter): `iptables -t nat -A POSTROUTING -o $WAN-IF -j MASQUERADE`

1. In POSTROUTING hits \texttt{nf\_nat\_out()} –
   a) IP\_CT\_NEW – hits MASQUERADE rule, updates
      CT DIR\_REPLY tuple: Src: 1.2.3.4, Dst: 2.2.2.2
      Replies can now hit in conntrack hash
   b) Marks ct->status with IPS\_SRC\_NAT
   c) Invert reply tuple – Src: 2.2.2.2, Dst: 1.2.3.4 (and L4 proto)
      update IP hdr and checksums

2. In POSTROUTING hits ipv4\_confirm() gets CT from skb->nfct, inserts nf_conn on two hash chains, oblivious to NAT updates.

```
skb->nfct = &ct->ct_general
skb->nfctinfo = IP\_CT\_NEW
```

```
LAN\rightarrow WAN:  Dst: 1.2.3.4, Dst: 10.0.0.10; GW Public=2.2.2.2
1. Hits PREROUTING - ipv4\_conntrack\_in(), miss in contrack hash creates new
   nf\_conn ORIGINAL: Dst: 1.2.3.4, Src: 10.0.0.10 ; REPLY: Dst: 10.0.0.10, Src: 1.2.3.4 and
   ports and protocols
2. nf\_in\_in() – cstats not updated
```

```
1. Filter Table in Forward Chain matches on skb->nfctinfo
   IP\_CT\_ESTABLISHED and  accepts the packet
2. nf\_nat\_out() – does nothing due to IPS\_SRC\_NAT flag
3. ipv4\_confirm() – does nothing
```

```
skb->nfct = &ct->ct_general
skb->nfctinfo = IP\_CT\_ESTABLISHED + IP\_CT\_IS\_REPLY
```

```
SYN
```

```
WAN\rightarrow LAN:  Dst: 2.2.2.2  Src: 1.2.3.4
1. Hits PREROTING - ipv4\_conntrack\_in() hit in CT hash, dir=REPLY set IPS\_SEEN\_REPLY\_BIT,
   &ct->status for future reference.  Forward the packet up
2. For DIR\_REPLY reverse original direction tuple update ip hlr (L4) with reversed destination
   in this case 10.0.0.10 – update checksums
```

- For this NAT setup – general rule:
  - original dir – invert reply and use Source – update packet/recalc checksums
  - reply dir – invert original use Dst – update packet/recalc checksums
IPv4 Connection Tracking – SPI+NAT+ALG

- Add ALG to NAT with Conntracking – typical scenario client using private IP
- ALG Example – FTP – the problem

FTP Server

1.2.3.4
IPv4 Internet

TCP SYN 192.168.0.253 - 46107

192.168.0.1
4.3.2.1
192.168.0.100
1.2.3.4
IPv4 Internet

192.168.0.1
FW State for Control Session
EST, REL

192.168.0.100
PORT 192,100,0,253,180,27

Residential/SOHO User

Passive mode off

FTP client in Passive Mode – passes private IP in Payload
Server will attempt – private IP
Several Extra ALG Layers Added – to handle
  a. ALG helper scans for header sig. i.e. port 21
  b. Monitors commands and rewrites to Public IP
  c. Adds an expectation – incoming data connection
  d. For this scenario Creates DNAT mapping
IPv4 Connection Tracking – SPI+NAT+ALG

IPv4 FTP ALG example – covers both general ALG work with FTP specifics

- Register FTP ALG Helper manually

  • Register “FTP helper” – ‘nf_contrack_fip’ registers ‘helper’ in ‘nf_ct_helper_hash[]’ – monitor FTP pkts
  • Later “NAT helper” – ‘nf_nat_fip’ referenced (can be dynamically loaded)

```c
int ip_forward()
{
    return NF_HOOK(NFPROTO_IPV4, NF_INET_FORWARD,..., ip_forward_finish);
    iptable_filter_hook,() -
    rule: input dev=wan, skb status ESTABLISHED or RELATED action ACCEPT
}
```

```c
int ip_mc_output(struct sk_buff *skb)
{
    return NF_HOOK_COND(NFPROTO_IPV4, NF_INET_POST_ROUTING, ... , ip_finish_output, ..)
}
```

1. POST ROUTING ipv4_conntrack_in() executes helper to parse packet
   - f.e. PORT command with Private IP (i.e. PORT 192.168.0,253,180,27)
2. NAT TABLE/PREROUTING: nf_nat_in()
3. CONNTRACK/PREROUTING: ipv4_conntrack_in() – process expect, execute DNAT helper

```c
ip_rcv()
{
    return NF_HOOK(NFPROTO_IPV4, NF_INET_PRE_ROUTING,  ip_rcv_finish)
    2. NAT TABLE/PREROUTING: nf_nat_in()
    1. CONNTRACK/PREROUTING: ipv4_conntrack_in() – process expect, execute DNAT helper
}
```

LAN → WAN: Client (w/Private IP) issues SYN to FTP server with port 21
- PREROUTING ipv4_conntrack_in() - misses, helper hash searched “ftp_helper” associate with ‘CT’
- Do standard NAT work

```c
int ip_mc_output(struct sk_buff *skb)
{
    return NF_HOOK_COND(NFPROTO_IPV4, NF_INET_POST_ROUTING, ... , ip_finish_output, ..)
    SNAT rule: dev=wan MASQUERADE (on new pkt only)
    2. CONNTRACK/POSTROUTING: ipv4_confirm() – ftp helper
}
```

LAN → WAN: PREROUTING: Client issues PASSIVE mode PORT command
- modifies packet PORT IP with Public IP
- Creates a “nf_contrack_expect” – programs DNAT
- Associates a NAT helper with expectation (nf_nat_fip)
- enqueues ‘nf_contrack_expect’ on - init_net->ct. expect_hash[]

LAN → WAN: PREROUTING: Client issues PASSIVE mode PORT command
- modify packet PORT IP with Public IP
- Creates a “nf_contrack_expect” – programs DNAT
- Associates a NAT helper with expectation ( nf_nat_fip)
- enqueues ‘nf_contrack_expect’ on - init_net->ct. expect_hash[]

WAN → LAN: PREROUTING: FTP data connection from server will mis
- hit the expect hash table, execute the NAT helper to create DNAT entry
- Set IPS_EXPECTED_BIT in ct->status, causes skb to be marked
- RELATED

...... Data Connection Statefully established by
- Connection Tracking and FW Rules ....

POST ROUTING ipv4_conntrack_in() searches “ftp_helper” associate with ‘CT’
- Do standard NAT work

POSTROUTING: helper finds PORT command
- modified packet PORT IP with Public IP
- Creates a “nf_contrack_expect” – programs DNAT
- Associates a NAT helper with expectation (nf_nat_fip)
- enqueues ‘nf_contrack_expect’ on - init_net->ct. expect_hash[]

FILTER TABLE/ FORWARD CHAIN accepts packet
it’s marked RELATED

...... Data Connection Statefully established by
- Connection Tracking and FW Rules ....

LAN → WAN: Client (w/Private IP) issues SYN to FTP server with port 21
- PREROUTING ipv4_conntrack_in() - misses, helper hash searched “ftp_helper” associate with ‘CT’
- Do standard NAT work

WAN → LAN: PREROUTING: FTP data connection from server will mis
- hit the expect hash table, execute the NAT helper to create DNAT entry
- Set IPS_EXPECTED_BIT in ct->status, causes skb to be marked
- RELATED

...... Data Connection Statefully established by
- Connection Tracking and FW Rules ....
CPU Offload integrated with IPv4
Connection Tracking – SPI+NAT+ALG

- **General Principles of CPU Network Offload Integration**
  - ISPs want CPU heavy Apps on application processor – thus need to offload traffic
    - More predictable & cost effective – then adding CPUs i.e. as UL/DL rates go up
  - CPU should be IDLE with Max packet bandwidth – GPON, DOCSIS 8-Ch Bonding 240 Mbps/DL 104 Mbps UL – higher in future
  - Initial Packets – go up to kernel – to program offload engine – primarily L2/L3/L4 used
    - For DPI apps – Parental control – transparent proxy, content filtering more – kernel determines
  - Must be fully integrated into: IP stack – connection tracking, NAT, ALGs, Routing, ....,
  - Sessions Limited resource – must prioritize TLU – like streaming over casual browsing
    - Light DPI (f.e. HTTP persistent connection, Content-Type, ..., fields)

- **General Operation**
  - LAN -> WAN
    - ingress hook: saves pre-SNAT’ed SRC L3/L4 (i.e. private IP), local MACs attaches info to skb
      - PREROUTING drops packet if ALG, offloading stops here;
    - egress hook: saves SNAT’ed L3/L4 (i.e. Public IP), WAN, GW MACs – programs Offload Engine
  - WAN-> LAN (Path not shown)
    - ingress hook: saves pre DNAT’ed L3/L4 (i.e. public IP), GW, WAN MAC attaches to skb
      - PREROUTING drops packet if ALG, offloading stops here;
    - egress hook: saves DNAT’ed L3/L4 (i.e. Private IP), local MACs – Programs Offload Engine
  - Two Offload Entries programmed – on match packet headers update, switched to egress port
  - Must see at least 2 packets – ORIGINAL/REPLY
  - Must have: src/dst macs, src/dst IPs, src/dst ports (if applicable), protocol type, ingress, and egress port(s)
  - Local packets to/from GW – obviously not accelerated
CPU Offload integrated with IPv4 Connection Tracking – SPI+NAT+ALG

- General integration of Offload Engine into Linux Network Stack

![Diagram of Offload Engine in Linux Network Stack]

- IPv4 Network Stack
- Offload Engine
- General Flow:
  - **Ingress**
    - L2: Src, Dst Mac
    - L3: Src, Dst IP
    - L4: Ports or others, protocol#
  - **Wan Egress Packet**
    - Src IP NAT’ed
    - Dst MAC changed
    - TTL Updated
    - Checksums recalculated
  - **Lan Ingress Packet**
    - L2: Src, Dst Mac
    - L3: Src, Dst IP
    - L4: Ports or others, protocol#
    - **Lan Egress Packet**
    - Dst IP NAT’ed
    - Src, Dst MAC changed
    - TTL Updated
    - Checksums recalculated

- **dev_queue_xmit()**
- **ip_output()**
- **ip_forward()**
- **ip_local_deliver()**
- **netif_receive_skb()**
- **ip_rcv()**
- **ip_local_deliver()**
- **ip_output()**

- **Table Missing**
- **Table Hit**

- L2: Src, Dst IP
- L3: Src, Dst IP
- L4: Ports or others, protocol#
CPU Offload integrated with IPv4 Connection Tracking – SPI+NAT+ALG

- **Additional kernel hooks for Offload Engine**
  - NETDEV_UP/DOWN (netdev_chain) – must flush TLU – prevent forwarding
  - Address change (inetaddr_chain) – lan, wan – flush TLU – prevent wrong addr use - for offloaded sessions
  - Routes added/deleted – static/dynamic – flush TLU – sessions out of sync
  - Conntrack timeout – extend timeout – if session offloaded
    - offloaded session invisible to OS
  - NAT – changes – flush TLU – sessions out of sync
  - MC routing – when added insert one entry – several egress ports
  - Few problems
    - GRE (PPTP) – may need to drop out – if call id’s collide – must let ALG handle (previous slide)
    - Similar issue with IPsec Tunneled Mode – ESP
    - Once offloaded – not secure
  - Handle session re-entrancy – OS ↔ Offload engine – disable Protocol Connection tracking
  - Parental Control/DPI – requires more logic – inspect packets longer – before offload
IPv6 Connection Tracking – SPI

- Same conntrack structure – used for IPv6, same hash table

```
nf_conn
```

```
struct nf_conntrack_tuple_hash tuplehash[IP_CT_DIR_MAX];
```

- **IPv6 & IPv4 SPI fully integrated**
  - common nf_conn, conntrack table – key structure ‘nf_conntrack_tuple’ - common
IPv6 Connection Tracking – SPI

- Example of IPv6 stateful firewall

```
int ip6_forward()
{
    return NF_HOOK(NFPROTO_IPV6, NF_INET_FORWARD,..., ip6_forward_finish);
}
```

```
ip6table_filter_hook,()  - rule: input dev=wan, skb status ESTABLISHED or RELATED action ACCEPT
```

```
int ip6_output(struct sk_buff *skb)
{
    return NF_HOOK_COND(NFPROTO_IPV6, NF_INET_POST_ROUTING, ... , ip6_finish_output,...)
}
```

```
CONNTRACK/POSTROUTING Chain: ipv6_confirm()
```

```
ipv6_rcv()
{
    return NF_HOOK(NFPROTO_IPV6, NF_INET_PRE_ROUTING, ip6_rcv_finish)
}
```

```
CONNTRACK/PREROUTING: ipv6_conntrack_in()
```

- Following Rule in Forward Chain:  `ip6tables -A FORWARD -i $WAN-IFACE -m state -state ESTABLISHED,RELATED -j ACCEPT`

- After DIR_REPLY statefulness established – means private host opened the FW
- The skb carries status – used by Filter Rules
IPv6 Connection Tracking – SPI+ALG

- Same ALG Example as for IPv4, NAT not applicable

```
FILTER Table/FORWARD Chain
int ip6_forward()
{
    return NF_HOOK(NFPROTO_IPV6, NF_INET_FORWARD, ..., ip6_forward_finish);
    ip6table_filter_hook,() - rule: input dev=wan, skb status ESTABLISHED or RELATED action ACCEPT
}
```

```
int ip6_output(struct sk_buff *skb)
{
    return NF_HOOK_COND(NFPROTO_IPV6, NF_INET_POST_ROUTING, ..., ip6_finish_output, ...)
    CONNTRACK/POSTROUTING Chain: ipv6_confirm() – execute helper
}
```

```
ipv6_rcv()
{
    return NF_HOOK(NFPROTO_IPV6, NF_INET_PRE_ROUTING, ip6_rcv_finish)
    CONNTRACK/PREROUTING: ipv6_conntrack_in() - process expect
}
```

- Register an ALG helper – ‘nf_contrack_ftp’ – registers FTP helper - on “nf_ct_helper_hash[ ]” - monitor for port 21

```
dev_queue_xmit()
```

```
netif_receive_skb()
```

- POST ROUTING ipv6_confirm() executes helper to parse packet – (i.e. command with EPRT |2|2001:1:2:3::2|46107)

```
FILTER TABLE/FORWARD CHAIN accepts packet it’s marked RELATED
```

```
skb->nfct = &ct->ct_general (with associated helper)
skb->nfctinfo = IP_CT_NEW
- Several Packets go by before ALG command executed
```

```
skb->nfct = &ct->ct_general (with associated helper)
skb->nfctinfo = IP_CT_ESTABLISHED + IP_CT_IS_REPLY
```

```
skb->nfct = &ct->ct_general (with associated helper)
skb->nfctinfo = IP_CT_RELATED
```

```
FILTER Table/FORWARD Chain
```

- Client issues SYN to FTP server with port 21
- POSTROUTING ipv6_confirm() - misses , helper hash searched “ftp_helper” associate with “CT”

- PREROUTING: Client issues PASSIVE mode EPRT command
- Creates a “nf_contrack_expect” –
- enqueues ‘nf_contrack_expect’ on - init_net->ct.expect_hash[]

- POSTROUTING: helper finds EPRT command
- Creates a “nf_contrack_expect” –
- enqueues ‘nf_contrack_expect’ on - init_net->ct.expect_hash[]

- PREROUTING: FTP data connection from server will mis contrack hash table.
- will hit the expect hash table,
- Set IPS_EXPECTED_BIT in ct->status, causes skb to be marked RELATED

```
To WAN/LAN
```
CPU Offload integrated with IPv6 SPI, ALGs

- **General Operation**

  - LAN -> WAN - *ingress hook*: similar to IPv4, although deals with IPv6 header structures
    - PREROUTING drops packet if ALG, offloading stops here – allow helper to follow ALG control connection – pkt modified
    - *egress hook*: similar to IPv4 – programs Offload Engine

  - WAN-> LAN (Path not shown) *ingress hook*: similar to IPv4
    - PREROUTING drops packet if ALG, offloading stops here; *egress hook*: similar to IPv4– Programs Offload Engine

  - Two Offload Entries programmed – on match packet headers update, switched to egress port
  - Must see at least 2 packets – ORIGINAL/REPLY
  - Must have: src/dst macs, src/dst IPs, src/dst ports (if applicable), protocol type, ingress, and egress port(s)
  - Local packets to/from GW – obviously not accelerated
  - Issues with protocols like IPsec/ESP go away
  - Configuration and source directories

**Kernel Configuration**
- must enable IPv6 to see IPv6 Net filter Options
  - The IPv6 protocol
  - Network packet filtering framework (Net filter)
    - Core Net filter Configuration
    - IP: Net filter Configuration
    - IPv6: Net filter Configuration

**Source Directories**
- connection tracking, ALG helpers (port opening) – both IPv4/IPv6 supported,
  - some generic match modules
    - ipv4 specific – NAT, NAT helpers, IPv4 specific match modules,
    - IPv4 table registration
    - ipv6 match modules, IPv6 Table registration

- For FTP ALG: net/net filter/nf_conntrack_ftp – required for both IPv4 & IPv6
  - net/net filter/ipv4/nf_nat_ftp -
CPU Offload integrated with IPv6 SPI, ALGs

 ![Diagram of CPU Offload integrated with IPv6 SPI, ALGs]
6RD

- Dual-Stack long term Preferable option for ISPs (example spec DOCSIS eRouter)
- Expected both protocols will co-exist for years to come – few reasons - applications not migrated to IPv6, embedded devices
- DS-Lite another option – tunneling IPv4 in IPv6 – breaks IPv4 features – UPnP IGD, DDNS, DMZ, ...
- 6RD – Rapid Deployment another option
  - Tunneling IPv6 over IPv4 similar to 6to4
  - tun6to4 device – used predefined IPv4 Anycast router to reach IPv6 internet
  - 6RD allows ISP specific prefix (instead of 2002::/16) used w/IPv4 addr – i.e. 2001:4dc2::/32, 192.0.2.100 → 2001:4dc2:c000:264::/64
    - New DHCPv4 option with prefix, border router IPv4 address
  - 6RD supported– ‘ip’ tool supports 6rd tunnel mode
    - IPv6: IPv6 Rapid Deployment (6RD) (EXPERIMENTAL) under “IPv6 protocols” enables 6RD
  - Offload Engines – no support IPv6 ↔ IPv4

- 6rd delegated Prefix: 2002:4dc2; 6rd IPv4

![Diagram showing IPv6 and IPv4 networking with 6RD and IPv6 Rapid Deployment]
IPv6 Provisioning Cablelabs eRouter Spec

- eRouter is a dual stack spec – strong reference for IPv6 implementation
- Can be configured for IPv4 only, IPv4+IPv6 or IPv6 only
  - There are TLVs in TFTP config which determine mode – mode assumed here is Dual-Stack
- IPv4 not covered – standard dhcp client /handler script - proxy DNS server, private IP DHCP server...
- Procedure for ISP Facing Interface – most likely flow – other variants not practical
  1. Construct link local address (LL) – ipv6/conf/wan/autoconf=1 – 0xfe80::<OUI host> - join ND and all Hosts MC group
  2. Get RAs – confirm managed mode (M flag set), get default router other params like Hop Limit, MTU
  3. The M bit must be set – issue DHCPv6 request – get IA_NA, IA_PD – perm. IPv6 address, prefixes and DNS server
     - Router may use Rapid Commit option in future – discussed earlier
  4. Run DAD NS, join ND and all hosts MC group
  5. DHCPv6 handler script (dibbler or dhcp6c KAME) – retrieves values later used to configure LAN side
 IPv6 Provisioning & Routing
Cablelabs eRouter Spec

### Procedure for Customer Facing Interface(s)

- The Customer Facing Interface configuration follows ISP server configuration
  
  1. Create LL address w/DAD, subscribe to ND & All hosts MC Groups
  2. Construct IPv6 address for each interface
     - Use IA_PD + interface OUI, run DAD, subscribe to ND & All hosts MC Groups
  3. Generate RAs – with O=1 and provide Prefix option
     - IA_PD – from DHCP on ISP facing interface
     - Client use SLAAC

  For example RADVD configuration:

  ```
  interface eth0.3
  {
    AdvSendAdvert on;
    MinRtrAdvInterval 30;
    MaxRtrAdvInterval 100;
    AdvOtherConfigFlag on;
    prefix 2001:1:2:3::/64
    ...
  };
  ```

  4. **Start up DHCPv6 Server**
     - At very least pass DNS Server – determined from ISP Interface configuration
     - Other acceptable option – run proxy DNS server – update /etc/resolv.conf - pas router as DNS server
     - Example from dhcp6s –
       
       ```
       option domain-name-servers 2001:1:2:3::50;
       ```

### Routing

- IPv6 addresses are globally routable – nothing special – of link ND for GW, on-link ND for destination
- MLD – similar to IGMP must – manage LAN membership – provide reports to queries – on ISP facing interface
Managing Dual Stack Gateway – SW solution

- ISPs concerned Dual-Stack will require more upgrades
  - Limit Service Calls, sending out technicians
- Prefer to isolate both stacks
- Parameter changes – require total reboot – safest approach
- With Dual-Stack – shared components (DNS, SNMP, TR-69, SSH, …) – updates impact both stacks
- Upgrade Flexibility – update kernel, apps keep other stack running
- Intelligent upgrade – constant interface
- Virtualization one solution
  - introduces new challenges

- BR_FILTER in/out – Forward Filter ‘-p <ipv4,arp>’ drop on ipv6 interface; ‘-p<ipv6>’ drop on ipv4 interfaces
- BR_FILTER local in – BROUTER drop
- Same WAN MAC on both VMs
- Allows independent management of stacks
- CPU offload need backend/frontend driver
Managing Dual Stack Gateway – HW Support

- Virtual devices with own memory mapped I/O – programmable MAC
- IPv4/IPv6 – routed at hardware level
- Physical Devices used for LAN/WAN Management interface
Managing Dual Stack Gateway – Upgrade

- Typical Image Format without virtualization

- Typical Upgrade of CPE – complex procedure
- TR-69 RPC used – communicate image download
- HTTPS – used to download image
- CPE typically upgraded at image level
- After download – image burned to ‘other’ side, boot side switched
- During reboot blackout period - On Fault recovery involved
- Virtualization Management VM – reboot has not blackout, under constant surveillance