Lincoln Tunnel L3 Software

Getting Started Guide

July 2009

Intel Confidential

Reference Number: XXXXXX-001US

Contents

Figures

Tables

Revision History

Date	Revision	Description
July 2009	001	Initial release

1.0 Introduction

1.1 About this Manual

This Getting Started Guide documents the instructions to obtain, build, install, and execute L3 Forwarding using Lincoln Tunnel.

1.2 Additional Information on Software

Lincoln Tunnel L3 Forwarding software has been validated using the 64-bit version of CentOS 5 with the 2.6.27 kernel.

1.2.1 Where to Find Current Software and Documentation

TBD:

1.2.2 Product Documentation

The following documentation is provided to support this software release:

- This Getting Started Guide
- TBD: What other documentation do we want to reference?

1.3 Related Software and Documentation

1.3.1 Open Source Software

The following table shows the required open source software.

Item	URL
CentOS 5.3 (64-bit version)	http://isoredirect.centos.org/centos/5/isos/x86_64
2.6.27.23 Linux* Kernel	http://kernel.org/pub/linux/kernel/v2.6/linux-2.6.27.23.tar.gz

1.4 Conventions

The following conventions are used in this manual:

- `Courier font` - commands and code examples
- *Italics* - directory names

1.5 Software Overview

1.5.1 Features Implemented

Lincoln Tunnel software is the Holland Tunnel L3 Forwarding ported to Nehalem. The changes between the two releases are:

- Lincoln Tunnel uses a newer Linux kernel version: 2.6.27
- Validated for 64-bit
- Uses a NUMA-aware driver

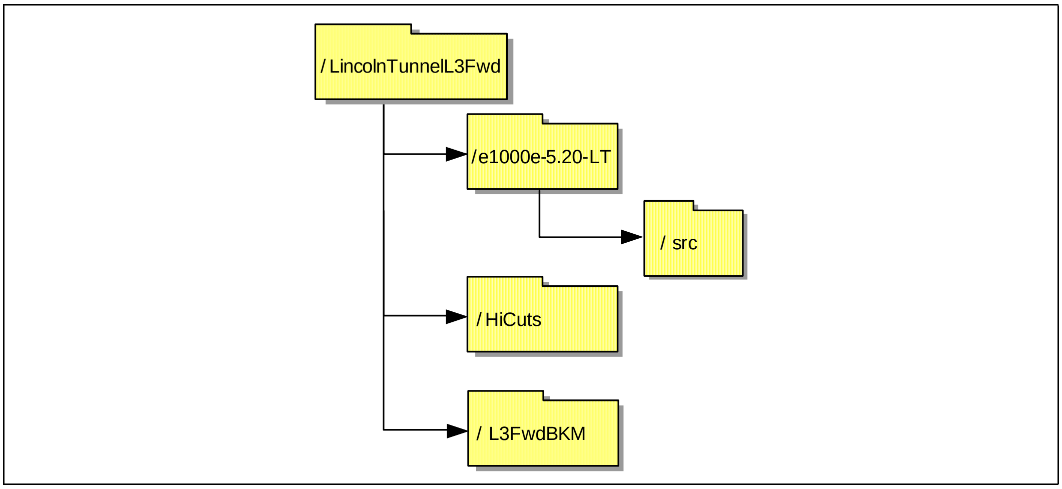
The software provides the following features:

- Flow classification (small table)
- By-pass of the Linux networking stack
- 64-bit NUMA-aware driver

1.5.2 Package Release Structure

The package release structure is shown in Figure 1.

Figure 1. Software Package Release Structure



2.0 Hardware

2.1 Pre-requisites

- A traffic generator with 16 Gigabit Ethernet* ports (for example, a SmartBits* or Ixia* chasis) that supports:
    - — Software for determining throughput limits (for example, SmartFlow* or IxNetwork*)
- 4x quad port network cards from Intel; these cards **must** support the e1000e Ethernet driver.

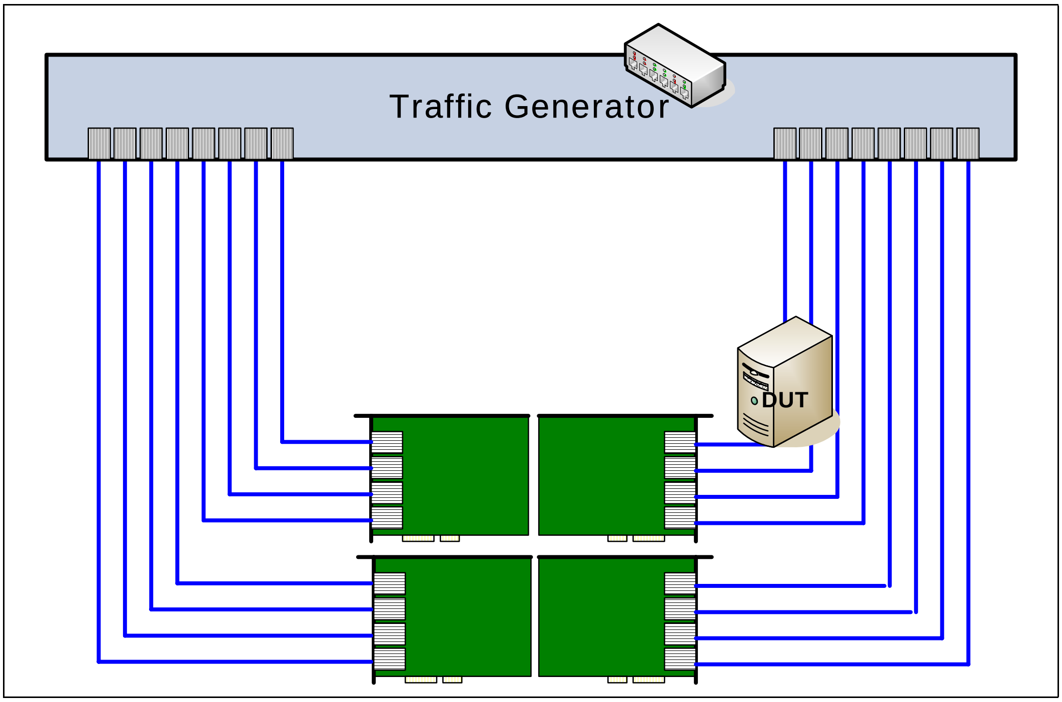
  The Intel® PRO/1000 PT Quad Port Server Adapter is an example of a quad port network card that supports the e1000e driver.
- A third-party network card (NIC) (not from Intel) that does not use the e1000e driver.

  Having a non-e1000e NIC enables the user to insert and remove the e1000e/ Lincoln Tunnel driver during testing without losing remote connectivity. Connections to the lab network should be configured through this port (eth0).
- 3 or 6 DIMMs of system memory, depending on the UP/DP configuration.

  1 DIMM of each available channel, 2 Gigabytes per channel, giving a total of 6 to 12 Gigabytes of system memory on the development system under test (DUT).

2.2 Installation

1. Configure the test network such that each Intel NIC port is connected to a Traffic Generator port.



2. In the system BIOS:

   a. Ensure that system memory is working at its maximum specified speed (since this can often be incorrectly detected) and ensure NUMA is enabled.

   b. Ensure the following BIOS settings are set (if available):

Setting	State
AC Prefetcher	DISABLED
HW Prefetcher	DISABLED
DCU Streamer Prefetcher	DISABLED
DCU UP Prefetcher	ENABLED
CPU C State	DISABLED
Symmetric MultiThreading	ENABLED
ACPI	DISABLED
Intel Speedstep	DISABLED
Direct Cache Access	DISABLED
ICH PCIE ASPM	L0 & L1 both DISABLED
Turbo Mode	DISABLED
BIST Selection	DISABLED
APIC Physical Mode	DISABLED
PCI Express* Port 3 (eth1, eth2) PCI-E Port Max Payload Request	128B

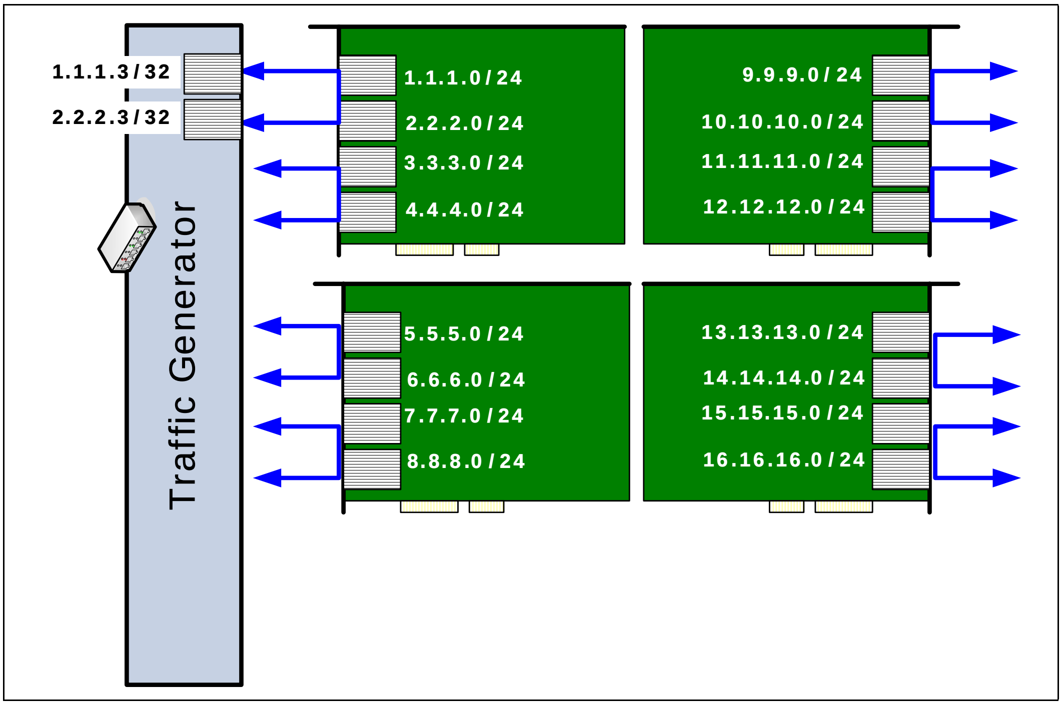
Setting	State
PCI Express Port 5 (eth3, eth4) PCI-E Port Max Payload Request	128B
PCI Express Port 7 (eth5, eth6) PCI-E Port Max Payload Request	128B
PCI Express Port 9 (eth7, eth8) PCI-E Port Max Payload Request	128B

3. Set the IP addresses on the Traffic Generator's ports as follows:

Traffic Generator Port	IP Address	Connected to...
1	1.1.1.3	eth1
2	2.2.2.3	eth2
3	3.3.3.3	eth3
4	4.4.4.3	eth4
5	5.5.5.3	eth5
6	6.6.6.3	eth6
7	7.7.7.3	eth7
8	8.8.8.3	eth8
9	9.9.9.3	eth9
10	10.10.10.3	eth10
11	11.11.11.3	eth11
12	12.12.12.3	eth12
13	13.13.13.3	eth13
14	14.14.14.3	eth14
15	15.15.15.3	eth15
16	16.16.16.3	eth16

4. Configure the traffic between ports in pairs, so that traffic from the left subnet is directed to only the right subnet and traffic from the right subnet is directed only to the left subnet (for example, 5.5.5.3 <-> 6.6.6.3 only).

Left Subnet	Right Subnet
1.1.1.3	2.2.2.3
3.3.3.3	4.4.4.3
5.5.5.3	6.6.6.3
7.7.7.3	8.8.8.3
9.9.9.3	10.10.10.3
11.11.11.3	12.12.12.3
13.13.13.3	14.14.14.3
15.15.15.3	16.16.16.3



5. The Traffic Generator should be configured to generate TCP traffic, as Lincoln Tunnel forwards anything other than TCP and UDP to the Linux* networking stack. Therefore, configure the packets to use the TCP protocol and port 80.

3.0 Software Installation Instructions

Installing and executing the Lincoln Tunnel L3 Forwarding software involves:

- Installing CentOS 5.3 and 2.6.27.23 kernel
- Updating IRQ affinities
- Updating the policies file
- Launching the application

3.1 Installing CentOS 5.3 and 2.6.27.23 Kernel

The 64-bit version (x86_64) of CentOS 5.3 can be found at:

http://isoredirect.centos.org/centos/5/isos/x86_64/

The Vanilla Linux* 2.6.27.23 kernel can be found at:

http://kernel.org/pub/linux/kernel/v2.6/linux-2.6.27.23.tar.gz

1. Install CentOS 5.3 64-bit:
    a. Choose to customize the packages selected for installation. Select Development Tools and Development Libraries to be installed.
    b. Setup the lab network on the third-party NIC so that the DUT is remotely accessible.
    c. Disable SELinux and firewalls during the firstboot stage of the setup (occurs after initial reboot during installation customization).
2. Configure the third-party NIC to use port eth0. The network ports that Lincoln Tunnel uses start at eth1.
    a. Determine the current port of the third-party NIC with the **ethtool -i** command:

```
# ethtool -i eth0
driver: r8169
version: 2.3LK-NAPI
firmware-version:
bus-info: 0000:08:02.0
```

    b. The third-party NIC uses a driver this is not provided by Intel, in this case r8169. Configure the DUT's networking scripts such that eth0 is the third-party NIC, that is, /etc/sysconfig/network-scripts/ifcfg-eth0 and all the Intel NIC ports range from eth1 and above.
    c. It may be necessary to disable the onboard NIC in the BIOS to make this change.
3. Configure ssh to permit root access:
    a. Change the sshd_config file so that PermitRootLogin is uncommented and set to “yes”.

b. Restart the sshd daemon:

```
# service sshd restart
```

c. Verify that the machine is now accessible over the lab network.

4. Extract the Lincoln Tunnel software archive into the */root/* directory.

5. Disable any unneccesary system services:

a. List the currently enabled system services with the following command:

```
# chkconfig --list | awk '{if($7=="5:on") print $0}'
acpid           0:off   1:off   2:on    3:on    4:on    5:on    6:off
haldaemon       0:off   1:off   2:on    3:on    4:on    5:on    6:off
kudzu           0:off   1:off   2:on    3:on    4:on    5:on    6:off
lvm2-monitor    0:off   1:on    2:on    3:on    4:on    5:on    6:off
network         0:off   1:off   2:on    3:on    4:on    5:on    6:off
readahead_early 0:off   1:off   2:on    3:on    4:on    5:on    6:off
readahead_later 0:off   1:off   2:on    3:on    4:on    5:on    6:off
sshd            0:off   1:off   2:on    3:on    4:on    5:on    6:off
syslog          0:off   1:off   2:on    3:on    4:on    5:on    6:off
```

b. Only the following services should be enabled:

- Acpid
- Haldaemon
- Kudzu
- Lvm2-monitor
- Network
- Readahead_early
- Readahead_later
- Sshd
- Syslog

Disable any other services with the `chkconfig <service name> off` command.

c. Reboot when complete.

6. Disable X:

a. Edit /etc/inittab and remove the following lines:

```
# run xdm in runlevel 5
x:5:respawn:/etc/X11/prefdm -nodaemon
```

7. Add kernel optimizations to the system initializations scripts:

a. Edit /etc/inittab and add the following lines:

```
net.ipv4.neigh.default.base_reachable_time=3000
net.ipv4.neigh.default.locktime=9000
net.ipv4.neigh.default.gc_stale_time=240
net.ipv4.neigh.default.gc_interval=250
net.ipv4.neigh.default.gc_thresh1=1024
net.ipv4.neigh.default.gc_thresh2=4096
net.ipv4.neigh.default.gc_thresh3=8192
net.ipv4.route.error_cost=1000
net.ipv4.route.error_burst=5000
net.ipv4.route.gc_interval=6000
net.ipv4.route.gc_min_interval=500
net.ipv4.route.gc_thresh=20480
net.ipv4.route.gc_timeout=3000
net.ipv4.route.max_size=327680
net.core.optmem_max=65536
net.core.rmem_max=1048576
net.core.wmem_max=1048576
net.core.rmem_default=524288net.core.wmem_default=524288
net.core.netdev_max_backlog=256
net.core.netdev_budget=256
net.core.dev_weight=256
net.ipv4.ip_forward=1
```

8. Ensure that SELinux is disabled:
    a. SELinux should have been disabled during firstboot.
    b. Ensure that /etc/selinux/config reads as follows:

```
# This file controls the state of SELinux on the system.
# SELINUX= can take one of these three values:
#       enforcing - SELinux security policy is enforced.
#       permissive - SELinux prints warnings instead of enforcing.
#       disabled - SELinux is fully disabled.
SELINUX=disabled
# SELINUXTYPE= type of policy in use. Possible values are:
#       targeted - Only targeted network daemons are protected.
#       strict - Full SELinux protection.
SELINUXTYPE=targeted
```

9. Remount drives to reduce time-stamping of files:
    a. Add the following lines to /etc/rc.local:

```
mount -o remount,noatime /
mount -o remount,noatime /boot
```

    b. Reboot when complete.
10. Extract the Vanilla Linux 2.6.27.23 kernel. This document assumes that the kernel is exacted into the */usr/src* directory.
11. Create a symbolic link to the extracted kernel:

```
$ ln -sf /usr/src/linux-2.6.27.23 /usr/src/linux
```

12. Copy the config-2.6.27.23 file from */root/LincolnTunnelL3Fwd/* into the */usr/src/linux* directory:

```
$ cp /root/LincolnTunnelL3Fwd/config-2.6.2.23 /usr/src/linux
```

13. Change to the */usr/src/linux* directory:

```
$ cd /usr/src/linux
```

14. Rename the config-2.6.27.23 to .config:

```
$ mv config-2.6.27.23 .config
```

15. Edit the Makefile to give the Kernel a distinguishing name:
    a. Edit the Makefile, changing the EXTRAVERSION parameter to read "EXTRAVERSION = .23-LT":

```
VERSION = 2
PATCHLEVEL = 6
SUBLEVEL = 27
EXTRAVERSION = .23-LT
NAME = Trembling Tortoise

# *DOCUMENTATION*
# To see a list of typical targets execute "make help"
```

16. Type the following command:

```
$ make menuconfig
```

17. Exit out of the make menuconfig tool saving the configuration.
18. Type the following commands to build the kernel:

```
$ make clean
$ make all
$ make modules_install
$ make install
```

19. After the kernel build and install has completed, update the grub loader configuration file so that the new kernel is the default kernel:
    a. Edit the file /etc/grub.conf and change the default parameter to 0:

```
#
# Note that you do not have to rerun grub after making changes to this file
# NOTICE:  You have a /boot partition.  This means that
#          all kernel and initrd paths are relative to /boot/, eg.
#          root (hd0,0)
#          kernel /vmlinuz-version ro root=/dev/VolGroup00/LogVol00
#          initrd /initrd-version.img
#boot=/dev/sda
default=0
timeout=5
splashimage=(hd0,0)/grub/splash.xpm.gz
hiddenmenu
title CentOS (2.6.27.23-lt)
    root (hd0,0)
    kernel /vmlinuz-2.6.27.23-lt ro root=/dev/VolGroup00/LogVol00 rhgb quiet
    initrd /initrd-2.6.27.23-lt.img
```

20. Reboot the machine into the new kernel. After the reboot, verify the new kernel is being used with the `uname -a` command.

```
Linux localhost.localdomain 2.6.27.23-LT #17 SMP Fri Jul 10 21:20:56 IST 2009
x86_64 x86_64 x86_64 GNU/Linux
```

3.2 Updating IRQ Affinities

This section describes the updating of IRQ affinities to match the test setup defined in *./LincolnTunnelL3Fwd/L3FwdBKM/irq_setup.sh*.

For example, on a system with two Intel® quad-core processors that are SMT-enabled, the affinity string might look like:

```
affinity=(ff 1 100 4 400 10 1000 40 4000 2 200 8 800 20 2000 80 8000)
```

This indicates that:

- `eth0` has affinity mask of `ff`
- `eth1` has affinity mask of `1`
- `eth2` has affinity mask of `100`

Each core's affinity mask can be found in the following table:

Core	Affinity Mask
0	1
1	2
2	4
3	8
4	10
5	20
6	40
7	80
8	100
9	200
10	400
11	800
12	1000
13	2000
14	4000
15	8000

Note: The core to which each interface is assigned is important.

1. Use `cat /proc/cpuinfo` to figure out which physical processor each logical processor belongs to.
2. In the `cat /proc/cpuinfo` output, look at the "processor", "physical id", and "core id" lines.

   The lines below are a partial output for four processors on an eight-core system that has SMT enabled:

```
processor      : 6
physical id    : 0
core id        : 3

processor      : 7
physical id    : 1
core id        : 3

processor      : 14
physical id    : 0
core id        : 3
```

```
processor       : 15
physical id     : 1
core id         : 3
```

This indicates that:

- — Processor 6 belongs to the physical processor in socket 0 and is an SMT thread on core 3.
- — Processor 7 belongs to the physical processor in socket 1 and is an SMT thread on core 3.
- — Processor 14 belongs to the physical processor in socket 0 and is an SMT thread on core 3.
- — Processor 15 belongs to the physical processor in socket 1 and is an SMT thread on core 3.

If the flows look like eth1 <--> eth2, where eth1 and eth2 are a full-duplex flow, interrupts for eth1 and eth2 should be enabled on the same physical socket, and on two separate SMT threads on the same core. So in this example, interrupts for eth1 should be assigned to core 6 and interrupts for eth2 should be assigned to core 14.

3.3 Updating Policies

This section describes how to update the policies.in file, which is located in the *./LincolnTunnelL3Fwd/HiCuts/* directory. Within the policies.in file, destination MAC addresses for the traffic generator are specified. These MAC addresses must be updated to match your traffic generator. See either Appendix A, "Policy Grammar" or the README file located in the *./LincolnTunnelL3Fwd/HiCuts/* directory for additional information on policies.

It is recommended to use interfaces eth1, eth2, eth3, and so on. Similarly, eth1's IP address should be 1.1.1.1, eth2's IP address should be 2.2.2.1, eth3's IP address should be 3.3.3.1, and so on. Doing this ensures that other scripts do not need to be modified. If different values are used, the following scripts need to be updated:

- TODO: List each other script that needs to be updated...

3.4 Launching Application

To run the application, execute the ./LincolnTunnelL3Fwd/runLT_L3Fwd.sh script file. Be sure to specify the number of ports you want to use along with the NUMA node assignment for each port. For example, for 16 ports, you would probably want to have eth1 to eth8 allocating from socket 0, and eth9 to eth16 allocating from socket 1. To accomplish this, the following command would be used:

```
$ bash runLT_L3Fwd.sh 16 0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1
```

Appendix A Policy Grammar

A.1 Policy Grammar

A policy file consists of multiple lines with the following structure:

```
id sip dip sport dport protocol ftype {rules}
```

The fields are defined as follows:

- **id**:
  Numerical policy id. This is only used for display/debug purposes. It should be, but doesn't have to be, unique.
- **sip**:
  Source IP address. It can be "any", a standard dot separated IP address (a.b.c.d) or IP address range. Ranges are specified as ip_address/bits, bits being the number of most significant bits in the IP address that matter. For example, 192.168.10.0/24 is equivalent to netmask 255.255.255.0.
- **dip**:
  Destination IP address. This follows the same format as "sip" above.
- **sport**:
  Source Port. It can be "any", a decimal port number or a port range. Ranges are specified as decimal_number-decimal_number. For example, 1024-2000.
- **dport**:
  Destination Port. This follows the same format as "sport" above.
- **protocol**:
  Can be "any", "tcp", "udp" or "ip". Please note that the current driver does not support "ip".
- **ftype**:
  Can be "normal", "nat" or "loadb". This affects how the "permit" directive is handled; please see "permit" section below.
- **rules**:
  List of rules to execute if a packet matches the specified policy. The minimum number of rules is 1 and the maximum number is 6. See the following.

Rule Format Detail

A rule is specified as:

```
(cond) ? true_action : false_action ;
```

cond:
Consists of 1) an offset and 2) a condition check (cond_check). The offset affects an internal offset counter (initialized to 0 on every packet). The data at this offset is then used in the condition check. If the condition is true, then the true_action is taken, else the false_action is taken.

- **offset**:
  An offset can be specified as:
  - "nochange"
    Does not change the internal offset, which is initialized to 0 prior to the first rule.
  - "rel <offset>"
    Adds the number to the internal offset:
    int_offset += <offset>;
  - "compute <offset> <shift> <mask>"
    Sets the new offset to:
    int_offset = ((data[int_offset] & <mask>) >> <shift>) + <offset>;
  - "absolute <offset>"
    Just sets the internal offset to <offset>:
    int_offset = <offset>;
- **cond_check**:
  A conditional check can be empty, "<op> <mask> <res>" or "<op> <mask> <res> <intval>".
  - When the condition is empty the true_action is taken by default.
  - In the second case the condition is tested as:
    (<res> <op> (data[int_offset] & <mask>))
  - In the last case the condition is tested as:
    (<res> <op> (internal_val_array[<intval>] & <mask>))
    Note: This case is not fully implemented.
- op:
  Can be one of the following:
  - "=" (equal)
  - "<" (less than)
  - ">" (greater than)
  - ">=" (greater than or equal)
  - "<=" (less than or equal)
  - "!=" (not equal)
- action:
  Can be: "none", "none <next_rule>", "drop", "permit <ethname> <ip> <mac>", "modify <type> <mod_offset> <mask> <val> <next_rule>", "crypto <next_rule>" or "exception".
  - none:
    Performs no action. If followed by a number then it denotes to run the rule at index <next_rule> next (0-based).
  - drop:
    Frees the packet and stops processing.
  - permit:
    The actions of this rule depend on the value of ftype above.

    permit(normal):
    This performs simple forwarding. It modifies the source MAC address to that of the interface specified by <ethname>, the destination MAC address to that specified by <mac>, updates the TTL value and checksum and sends the packet out the <ethname> device. The <ip> value is not used in this case.

permit(loadb):
This performs simple load balancing. In this mode we assume the traffic is directed to the machine running the pipeline. This modifies the source MAC address to that of the interface specified by <ethname>, the destination MAC address to that specified by <mac>, changes the destination IP address to <ip>, updates the TTL value, updates the checksums and sends the packet out the <ethname> device. It also adds a corresponding rule to redirect all returning traffic back to the device, modifying the packet so that it appears to be coming from the machine running the pipeline.

permit(nat):
This performs simple NAT'ing. This modifies the source MAC address to that of the interface specified by <ethname>, the destination MAC address to that specified by <mac>, changes the source IP address to <ip>, changes the source port to a unique value, updates the TTL value, updates the checksums and sends the packet out the <ethname> device. It also adds a corresponding rule to redirect all returning traffic back to the originating device.

— modify:
If the type is "value" then the value at moff is modified as:
  data[moff] = (data[moff] & <mask>) | <val>
If the type is "index" then the value at moff is modified as:
  data[moff] = (data[moff] & <mask>) | internal_val_array[<val>]
Note: This case is not fully implemented.

The value of moff is determined by <mod_offset> as defined below:

mod_offset:
This offset is also based upon the internal offset counter and is specified the same way as "offset" above, the main difference being that it does not modify the internal offset counter:

- "nochange" and "compute" mean that moff is the same as the internal offset:
  moff = int_offset;

- "rel <offset>" adds the number to the internal offset:
  moff = int_offset + <offset>;

- "absolute <offset>" just sets the offset to <offset>:
  moff = <offset>;

— crypto:
Not yet implemented

— exception:
Not yet implemented

Note: White space is ignored and any line starting with "#" is considered a comment and is also ignored.

A.2 Simple Rules Examples

Example 1.

```
1 any 192.168.11.0/24 any any any normal { (nochange)? permit eth1 0.0.0.0
00:00:04:00:00:01 : none;}
```

This forwards all received traffic destined for the 192.168.11.0/255.255.255.0 network out via the eth1 interface, addressed to the device at MAC 0:0:4:0:0:1.

Example 2.

```
2 192.168.10.0/24 192.168.11.1 any 80 tcp loadb { (nochange)? permit eth1
192.168.15.2 00:00:04:00:00:01 : none;}
```

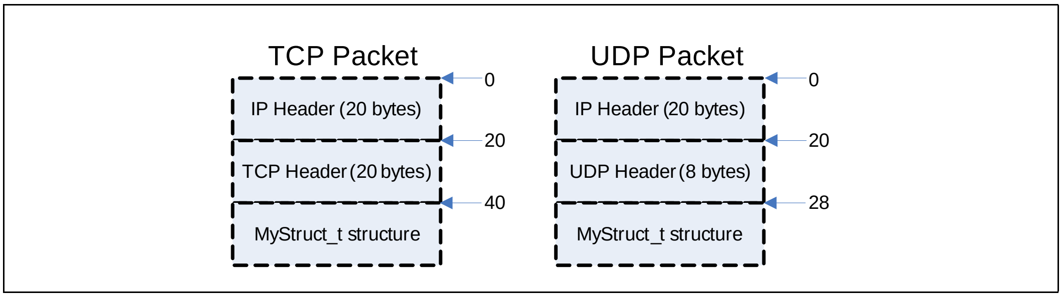
This redirects all HTTP traffic from the 192.168.10.0/255.255.255.0 network originally destined for 192.168.11.1 to the machine at 192.168.15.2 via eth1, again rewriting the destination MAC address to 0:0:4:0:0:1.

A.3 Advanced Rule Example

Assume that the payload of the target traffic consists of the following structure:

```
typedef struct
{
    u16 m_u16SomeType;
    u8  m_u8Key;
    ...
} MyStruct_t;
```

Also, assume that the m_u8Key value MUST be 42. If it is, then the traffic should be forwarded, otherwise, it should be dropped. Finally, assume that the traffic consists of both TCP and UDP packets as shown:



Note that the payload starts at different offsets based on the protocol type. Therefore, the packet needs to be first examined to see which type of packet it is (TCP or UDP).

Based on that, we can then offset into the payload part of the packet and check the m_u8Key value. This can be accomplished with the following rule:

```
1 any any any any any normal
{
    # Rule 0:
    (absolute 10 = 255 6  ) ? none 2 : none ;

    # Rule 1:
    (nochange    = 255 17 ) ? none 3 : drop ;

    # Rule 2:
    (rel      30          ) ? none 4 : none ;

    # Rule 3:
    (rel      18          ) ? none   : none ;

    # Rule 4:
    (rel       2 = 255 42 ) ? permit eth1 0.0.0.0 00:03:47:F4:F7:63 : drop;
}
```

Rule 0 sets the internal offset to 10 (the protocol byte of the IP header) and then checks to see if the byte at that offset is 6 (TCP). If it is, it jumps to Rule 2. If it does not match, it goes to the next rule, Rule 1. Note that Rule 0 could also be written as:

```
(rel 10 = 255 6  ) ? none 2 : none ;
```

since the starting offset is always 0.

Rule 1 does not modify the internal offset (still 10) and then checks to see if the byte at that offset is 17 (UDP). If it is it jumps to Rule 3. If it is not, then the packet is dropped.

Rule 2 adds 30 to the internal offset to skip over the remaining IP header and the TCP header. It has no condition so the True segment is taken which causes a jump to Rule 4.

Rule 3 adds 18 to the internal offset to skip over the remaining IP header and the UDP header. It has no condition and no actions so it goes to the next rule, Rule 4.

By the time we get to Rule 4, the internal offset should be pointing to the start of the payload. Rule 4 then adds 2 to the internal offset (the offset of m_u8Key) and then checks whether the byte at that offset is 42. If it is, then the packet is forward to the machine at MAC address 00:03:47:F4:F7:63 out of interface eth1. If it is not, then the packet is dropped.

The equivalent pseudo code is:

```
{
    int offset = 0;

    // Offset into the protocol byte of the IP header
    offset = 10;

    // Check for TCP
    if ( (data[ offset ] & 255) == 6) {
        // offset past the remaining IP header and the TCP header
        offset += 10 + 20;
    }

    // Check for UDP
    else if ( (data[ offset ] & 255) == 17) {
        // offset past the remaining IP header and the UDP header
        offset += 10 + 8;
    }
    else {
        // Drop the packet
    }

    // Offset to the m_u8Key value
    offset += 2;
    if ( (data[ offset ] & 255) == 42) {
        // Forward to the machine at MAC address 00:03:47:F4:F7:63
        // out interface eth1
    }
    else {
        // Drop the packet
    }
}
```

§ §