

# PoC of New IP

Bhaskar Kataria  
Rohit M. P.  
Leslie Monis  
Mohit P. Tahiliani  
NITK Surathkal

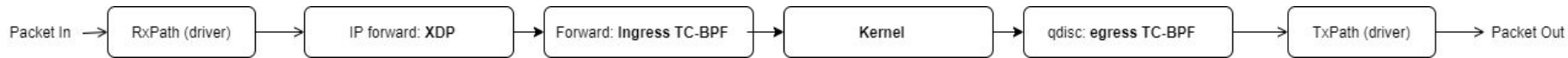
Kiran Makhijani  
Futurewei Technologies, Inc.

# Goals

Implement a programmable data plane for New IP packet processing

- **Address customization:** applications and routers should be able to forward packets between hosts with different address formats.
- **Design an end-to-end model to meet service delivery guarantees:** routers should be able to process various in-network New IP contracts as described by the applications.
- **Rapid experimentation** of the New IP components should be possible.

# IP packet flow in Linux



# Main Components

1. Topology setup - Network Stack Tester ([NeST](#))
2. [Scapy](#) - Host side API
3. eXpress Data Path [XDP](#) - shipping and contract processing
4. Traffic Control-BPF ([TC](#)) - LBF queue discipline

# New IP packet flow

Topology-setup: NeST



# Topology setup: NeST

- NeST is a python3 package designed for emulating real-world networks
- Uses Network namespaces to create the topology
- Makes testing easier

For this project, NeST is used to:

- Assign IP addresses to the nodes
- Populate the routing tables
- Form the packet by providing necessary information like MAC addresses

# Host side API: Scapy

- Python library for powerful interactive packet manipulation
- Helps forming custom packet formats
- Uses Raw sockets internally

For this project, Scapy is used to:

- Craft and send the packets from the virtual interface setup by NeST
- Sniff and decode the packet at the receiver

# XDP

- XDP is an eBPF based high performance data path
- Merged in the Linux kernel
- Provides packet processing at the lowest point in the software stack

For this project, XDP is used to:

- Update the Ethernet header in the packet
- Identify the interface from which the packet is to be sent using kernel routing tables or BPF maps
- Store the exit interface index (ifindex) in metadata of the packet
- Pass the packet on to the TC BPF hook



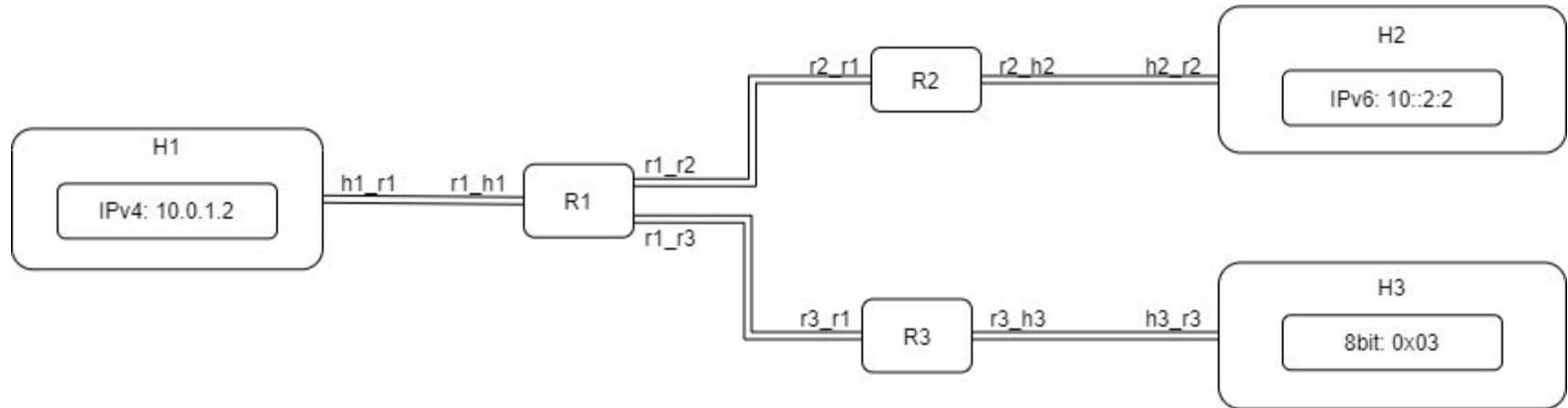
# TC-BPF

- BPF programs attached to the traffic control (tc) ingress and egress hook
- Runs after the networking stack has done the initial processing of the packet
- Has access to the metadata of the packet

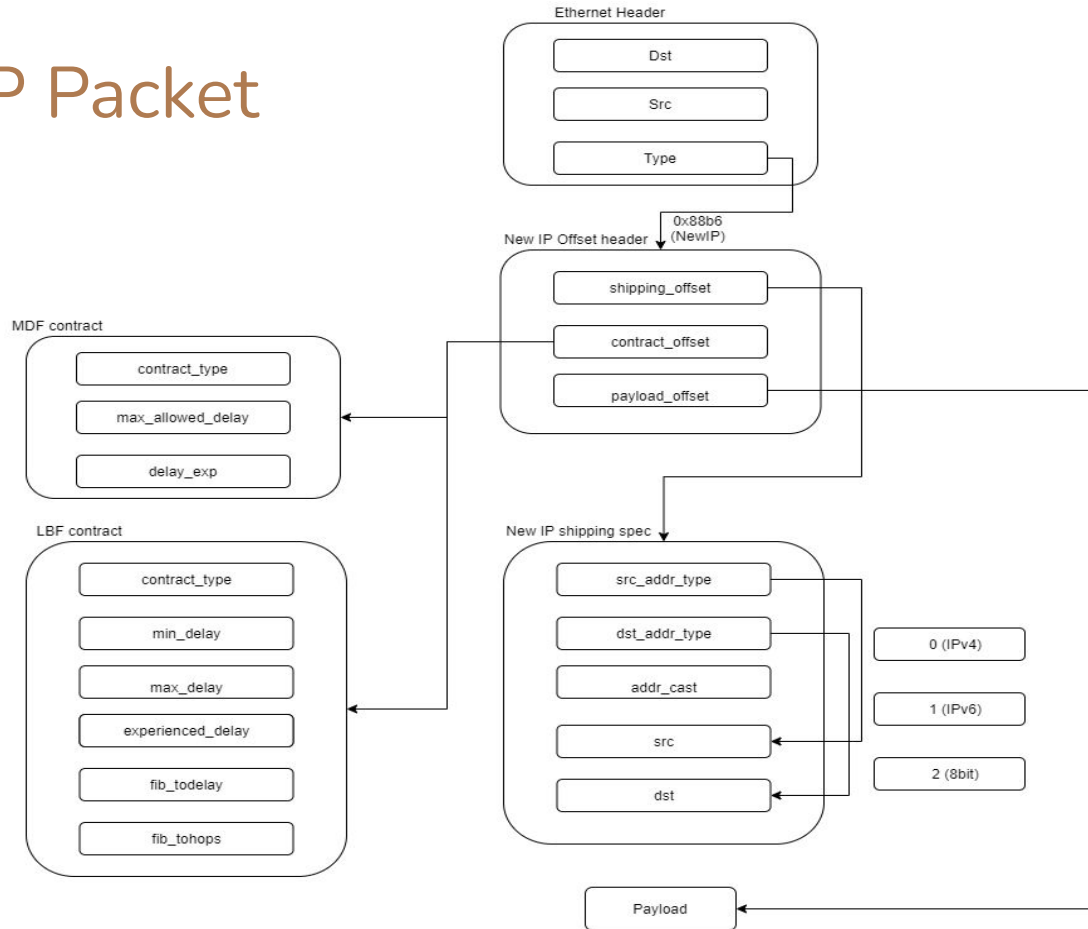
For this project, TC-BPF is used to:

- Add our queueing discipline (we cannot have queue discipline with just XDP)
- Read the ifindex from metadata at ingress hook
- Redirect the packet to the egress hook of the interface associated with ifindex
- Run our queue discipline on the egress hook

# Topology Setup - NeST



# New IP Packet



# New IP Packet in Wireshark

The image shows a Wireshark packet capture window. The top menu bar includes File, Edit, View, Go, Capture, Analyze, Statistics, Telephony, Wireless, Tools, and Help. Below the menu is a toolbar with various icons. A display filter is applied: `Apply a display filter ... <Ctrl-/>`. The packet list pane shows a single packet, packet 17, with the following details:

No.	Time	Source	Destination	Protocol	Length	Info
17	1.887838	10.0.1.2	10::2:2	New IP	93	New IP

The packet details pane shows the following structure:

- Frame 17: 93 bytes on wire (744 bits), 93 bytes captured (744 bits)
- Ethernet II, Src: f6:4c:82:c1:6d:94 (f6:4c:82:c1:6d:94), Dst: ca:79:49:92:7b:2f (ca:79:49:92:7b:2f)
  - Destination: ca:79:49:92:7b:2f (ca:79:49:92:7b:2f)
  - Source: f6:4c:82:c1:6d:94 (f6:4c:82:c1:6d:94)
  - Type: Local Experimental Ethertype 2 (0x88b6)
- New IP Packet
  - New IP Offset
    - Shipping Offset: 4
    - Contract Offset: 28
    - Payload Offset: 40
    - Type: 1
  - New IP Shipping Spec
    - Source Address Type: IPv4 (0x00)
    - Destination Address Type: IPv6 (0x01)
    - Address Cast: Unicast (0x00)
    - Type: Latency Based Forwarding Contract (0x02)
    - Source Address: 10.0.1.2
    - Destination Address: 10::2:2
  - Latency Based Forwarding Contract
    - Contract Type: 2
    - Minimum Delay: 500
    - Maximum Delay: 800
    - Delay Experienced: 166
    - Total Delay: 200
    - Total Hops: 2
- Data (39 bytes)

The packet bytes pane shows the raw data in hexadecimal and ASCII:

Offset	Hex	ASCII
0000	ca 79 49 92 7b 2f f6 4c 82 c1 6d 94 88 b6 04 1c	.yI·{/·L ··m·····
0010	28 01 00 01 00 02 0a 00 01 02 00 10 00 00 00 00	(················
0020	00 00 00 00 00 00 00 02 00 02 00 02 01 f4 03 20	················
0030	00 a6 00 c8 00 02 69 70 76 34 20 74 6f 20 69 70	······ip v4 to ip
0040	76 36 20 66 72 6f 6d 20 68 31 20 74 6f 20 68 32	v6 from h1 to h2
0050	20 6d 6f 72 65 20 6c 61 74 65 6e 63 79	more la tency

# API for Crafting a New IP packet

```
sender_obj = Sender()
```

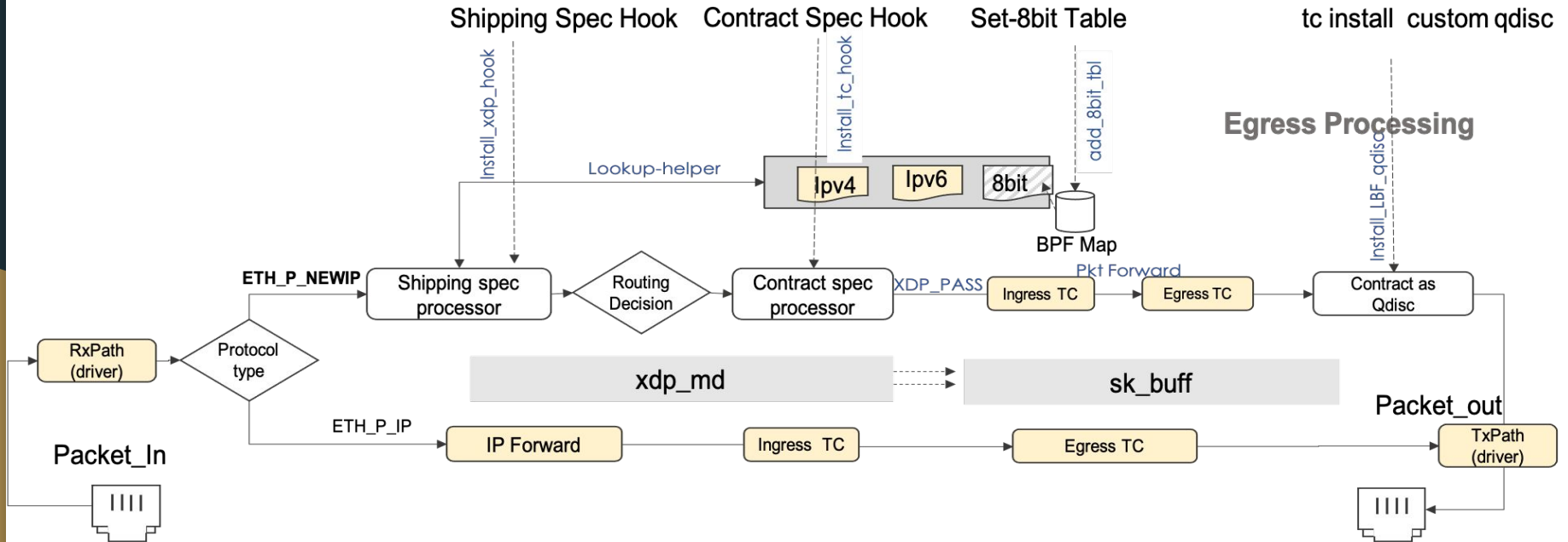
```
sender_obj.make_packet(src_addr_type="ipv4", src_addr="10.0.1.2",  
dst_addr_type="ipv6", dst_addr="10::2:2",content="ipv4 to ipv6 from h1  
to h2 more latency")
```

```
lbf_contract = LatencyBasedForwarding(min_delay = 500, max_delay = 800,  
fib_todelay = 0, fib_tohops = 3)
```

```
sender_obj.set_contract ([lbf_contract])
```

```
sender_obj.send_packet(iface='h1_r1', show_pkt=True)
```

# New IP Packet processing



# Demo

Thank you!