Design and Implementation of a Software-Based Network Router

Vamsi Alluri, Yun Kim, Joseph Luo, Edward Kelley

May 1, 2025

1 Architecture

Our router employs a multi-process architecture where a central supervisor process manages several independent service processes.

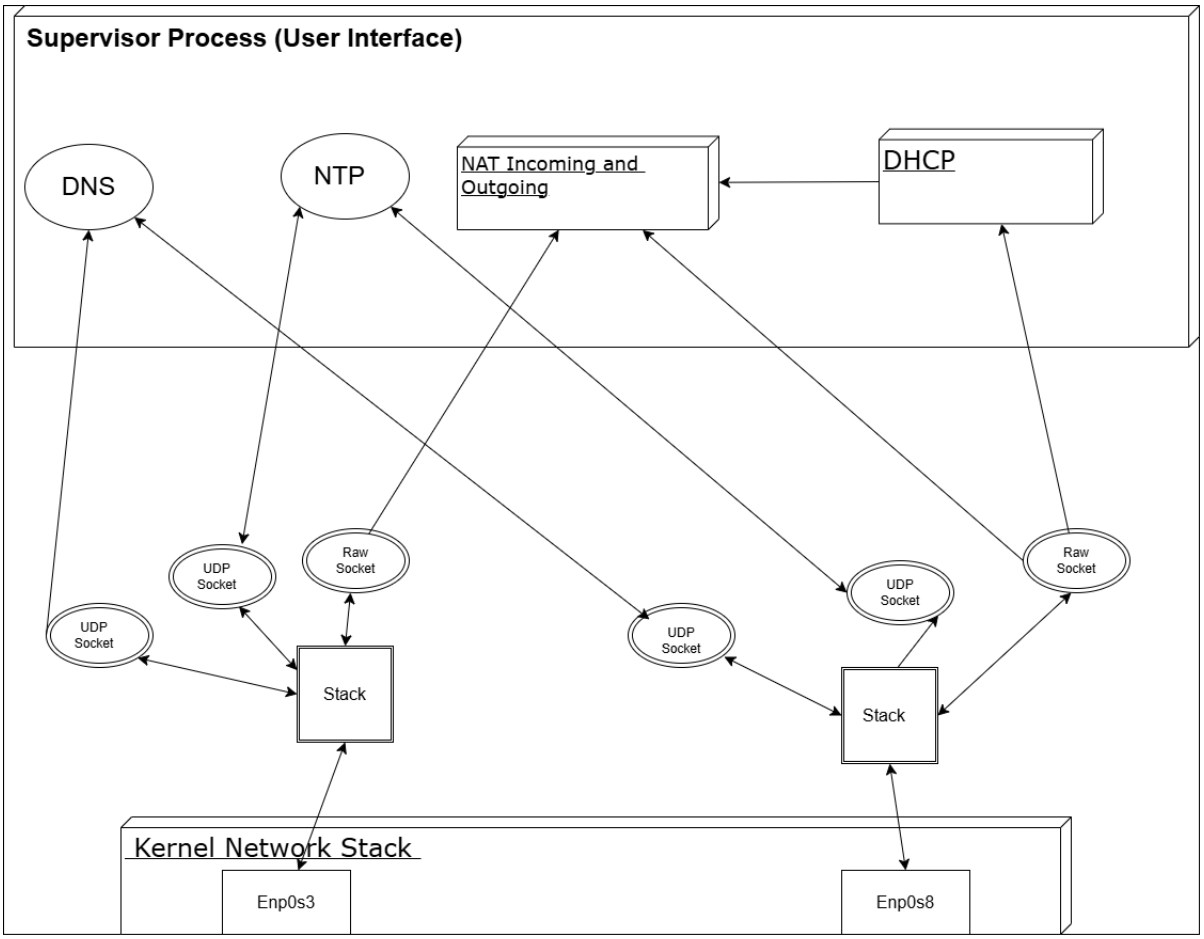


Figure 1: Overview of the router's multi-process architecture: a supervisor process spawns and monitors multiple service processes, isolating failures and enabling scalable growth.

1.1 Supervisor Process (Router)

The main executable router acts as the central control system and user interface. This program's primary responsibilities are parsing command-line arguments, initializing and launching individual

network service processes including DHCP, NAT, DNS¡ NTP, establishing inter-process communication(IPC) channels (pipes) with each service, providing a command-line interface(CLI) for user interaction and service management, monitoring service status and handling message received from services, using select() to multiplex I/O between the standard input and the pipes connected to the service processes, handling signal for shutdown, and cleanup or termination of the all service processes before shutdown.

1.2 Service Processes

1.2.1 Core Network processes: (DHCP, NAT, DNS, NTP)

Each core network function (DHCP, NAT, DNS, NTP) runs as a separate, dedicated process. Each service is launched via fork from the Supervisor Process. Each service process communicates with the main supervisor process (router) using two pipes, one receiving from the router and one for sending status/responses back to the router. The entry point for each service's logic such as DHCP or NAT is called after daemonization, receiving the file descriptors for its communication pipes.

1.2.2 Inter-Process Communication(IPC):

Communication between the supervisor and service processes relies on pipes. This provides a simple, unidirectional channel for sending commands and receiving basic text-based responses or status updates.

This multi-process design provides isolation between the core router services, preventing a crash in one service (e.g., dnsd) from directly taking down others (e.g., natd or the main router process). The supervisor pattern with pipe-based IPC allows for centralized control and monitoring.

2 Additional Features Implementation Details

2.1 Network Time Protocol (NTP) Server

The NTP server opens a non-blocking UDP socket on the LAN interface (`enp0s8`) bound to port 123. The server filters incoming packets to only process those with a mode of 3, which means client. The server then constructs a reply packet with all fields set to zero except: the version number is 4, the stratum is 4, the origin timestamp is the transmit timestamp of the incoming packet, and the receive timestamp and the transmit timestamp are both set to the current time. A stratum of 4 indicates that this is a secondary server and fetches its time through an NTP server on the internet and does not use a GPS receiver to fetch its time. To populate the timestamp, the local time stored by the OS is converted from UNIX time to NTP epoch using the NTP_TIMESTAMP_DELTA and then host to network byte order conversion. The origin, receive, and transmit timestamps allow the client to properly use the NTP response to update its local time. This constructed reply is then sent back to the sender.

Additionally, as a secondary NTP server, the NTP server needs to refresh its local time routinely (as often as `DEFAULT_REFRESH` seconds) with another NTP server on the internet. To do this, it opens a UDP socket on the WAN interface (`enp0s3`) bound to a port unsurveiled by NAT and connects to the designated NTP refresh server using its domain name and `gethostbyname`. Then it sends a packet to the refresh server with a version number of 4 and a mode of 3 using `send` and receives the response packet using `recv`. Additonally, `setsockopt` is used to set the `recv` to timeout after 3 seconds if there is no reply. Then it converts the second component of the transmit time to UNIX time and uses `settimeofday` to set the local time to it. Then the socket is closed.

Note that socket operations (creation, binding, I/O errors), packet sends, receives, select timeouts, and `settimeofday()` outcomes are logged via `append_ln_to_log_file_ntp_verbose()`. Additionally, times are handled with one second precision.

2.2 Domain Name System (DNS) Server

The DNS server opens a non-blocking UDP socket bound to the LAN interface (`enp0s8`) bound to port 53. It only handles queries such that they are without errors, are a standard query, have zero as their zero flag, are a query, have one question, have no answers, have no authority resource records, and have no additional resource records. Then the question's type must be an A record and its class must be Internet Class. If any of this is not the case, the query will be sent back to the client with the not implemented RCODE flag. Then the reply is constructed by looking up the domain name in the domain name table using djb2 hash function and linear probing and constructing an answer associated with that domain for each IPv4 address stored in the table. If it was not in the table, then this will be fetched via the set domain name server using a UDP socket similar as I described for NTP's refreshing. Only A record answers and CNAME chaining will be handled (Where there are sequential CNAME answers connecting the question's domain name through other domain names and eventually to IPv4 addresses). The reply to the client appends these answers to the initial query and alters the header flags where appropriate. Then an entry for this domain name will be inserted into the table with its ttl and IPv4 addresses. Additionally, this table will be routinely cleaned to rid the table of entries with expired ttl. On shutdown, the entire table is freed. All cache hits, misses, upstream forwards, and errors are written in the log file.

2.3 Log Feature

Each service logs major changes to configurations or error it encountered to the log file and there is an option to enable verbose using cli arguments when starting the server, which enables more clear logging for debugging.

2.4 Router Management Application with command line interface

The router process plays a central role in managing the services and handling commands from the CLI. Users can start or shutdown the services, and can interact with the sub services using ¡service name¿:¡command¿ type commands.

3 Implementation Details

3.1 NAT Implementation

The NAT module operates as a transparent gateway between the LAN and WAN interfaces by capturing all Ethernet frames in promiscuous mode on each side. Outbound IPv4 packets from the LAN are parsed to extract the IP header, protocol, and transport-layer port or ICMP identifier. For each outbound packet, the router looks up an existing translation (matching LAN source IP, source port/ID, and protocol) in the hash-indexed NAT table. If none exists, it allocates a fresh ephemeral port (or ICMP ID) on the WAN side, records a new `nat_entry` containing the original LAN tuple, the translated WAN tuple, protocol, and a timestamp, and inserts it into a power-of-two-sized hash table using chaining. The packet's source IP and port/ID fields are then rewritten to the router's WAN address and the allocated port, checksums are recomputed, and the frame is forwarded out the WAN interface (after ARP resolution for the gateway MAC). Inbound packets destined to a translated WAN port are similarly matched via the reverse lookup, their destination rewritten back to the original LAN host, checksums updated, and forwarded on the LAN interface.

Edge cases for TCP packets more than MTU and TTL expiration:

- When TCP packets received are more than the supported MTU, NAT returns an ICMP packet to set the MTU.
- When time to live is 1 or less in ip header, returns time limit exceeded ICMP packet.

3.2 NAT Table Management

Each NAT entry tracks its last usage time and carries a per protocol timeout: TCP entries default to four hours (to cover long-lived connections), UDP entries to five minutes (reflecting typical short transactions), and ICMP entries to one minute (for simple echo requests). A periodic cleanup routine scans the entire table every minutes, removing any entry whose `time(NULL) - last_used` exceeds its protocol's and state management timeout. This approach balances resource usage preventing stale mappings from accumulating with functionality, allowing legitimate long-lived TCP flows to persist while quickly reclaiming ports used by transient UDP and ICMP traffic.

3.3 DHCP Implementation

The DHCP service implements a renew-first, allocate-second strategy. It first searches its lease table for an existing binding to the client's MAC address and, if found and untainted by recent conflicts, simply renews that lease for another hour. If no valid binding exists, it selects the first free slot in a fixed-size array, and computes the candidate IP as:

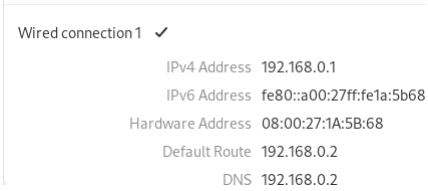
$$\texttt{network\_addr} + \texttt{index} + 1,$$

and then passing the data with that IP with an ICMP Echo Request (ping) to detect live hosts before offering it. Leases are tracked with start, end, and conflict timestamps to prevent reusing an address too soon after a collision.

3.4 IP Assignment Flexibility

tldr, Yes, the router works with different subnet masks. It first checks the slot, and since it is unavailable, it skips to the next slot and checks. As a result, the code correctly assigns the correct slots. The router will work correctly if it has assigned address 192.168.0.2/29 or 192.168.0.2/16, and both of the cases worked.

Result for 192.168.0.2/29:



Limitation: Current logic uses network address + i + IP ALLOCATE START OFFSET where offset is +2. This could be too simple of logic to giveout the IP addresses.

4 Testing Methodology

4.0.1 Performance Metrics

We evaluate each task along two dimensions:

- Correctness
- Efficiency

Correctness Correctness is defined as the router's ability to complete the specified operation under test conditions without error.

Efficiency Efficiency captures how fast the router carries out each task. We use iPerf3 to generate controlled traffic loads and measure:

- Throughput (Mbps): the achieved data rate under sustained transfer.
- Latency (ms): the round-trip time for small probe packets.

By combining correctness percentages with throughput and latency measurements, we obtain a comprehensive view of both functional and performance behavior of our router under test.

4.1 Experiment Procedure

Setting: Run config.sh file in the Router-side VM to set up the proper configuration environment to set up the iptables. Manually set up the router interface to have an ip of 192.168.1.1 and set the subnet mask to 255.255.255.0.

1. Open VM Host A, B, and C.
2. Check the LAN interface settings to confirm DHCP has correctly assigned the address on the host machine. Type dhcp:list_leases in the supervisor process cli to check it has been correctly received.
3. Test UDP bandwidth using Iperf and our built-in router.
4. In the cli, run dns:set hi 8.8.8.8. Then run ping hi from Host A.

4.2 Evaluation

4.2.1 Correctness:

All router operations have been completed successfully.

4.2.1.1 DHCP Correctness:

```
root@router# dhcp:list_leases
DHCP: Active leases:
  IP: 192.168.1.2    , MAC: 08:00:27:1a:5b:68, Expires in: 3488 s
  IP: 192.168.1.3    , MAC: 08:00:27:27:16:f2, Expires in: 3536 s
  IP: 192.168.1.4    , MAC: 08:00:27:56:d8:bb, Expires in: 3591 s
DHCP: End of lease list
```

The result shows that the DHCP has correctly assigned IP address to the VM hosts A, B, C.

4.2.1.2 DNS Correctness

```
root@vbox:~# ping hi
PING hi (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=255 time=18.2 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=255 time=12.3 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=255 time=13.8 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=255 time=18.7 ms
64 bytes from 8.8.8.8: icmp_seq=5 ttl=255 time=21.5 ms
^C
--- hi ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4014ms
rtt min/avg/max/mdev = 12.326/16.936/21.543/3.377 ms
```

The results shows that the ping to hi goes through to 8.8.8.8 successfully. This shows that our DNS server is working correctly.

```
root@osboxes:/home/osboxes/cs536/lab5/tcp_server_client# iperf3 -c 192.168.56.1 5201 -u
Connecting to host 192.168.56.1, port 5201
[  5] local 192.168.10.2 port 51846 connected to 192.168.56.1 port 5201
[ ID] Interval           Transfer     Bitrate         Total Datagrams
[  5]   0.00-1.00   sec   128 KBytes  1.05 Mbits/sec  90
[  5]   1.00-2.00   sec   128 KBytes  1.05 Mbits/sec  90
[  5]   2.00-3.00   sec   128 KBytes  1.05 Mbits/sec  90
[  5]   3.00-4.00   sec   127 KBytes  1.04 Mbits/sec  89
[  5]   4.00-5.00   sec   128 KBytes  1.05 Mbits/sec  90
[  5]   5.00-6.00   sec   128 KBytes  1.05 Mbits/sec  90
[  5]   6.00-7.00   sec   128 KBytes  1.05 Mbits/sec  90
[  5]   7.00-8.00   sec   128 KBytes  1.05 Mbits/sec  90
[  5]   8.00-9.00   sec   127 KBytes  1.04 Mbits/sec  89
[  5]   9.00-10.00  sec   128 KBytes  1.05 Mbits/sec  90
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Jitter    Lost/Total Datagrams
[  5]   0.00-10.00  sec  1.25 MBytes  1.05 Mbits/sec  0.000 ms  0/898 (0%)  sender
[  5]   0.00-10.03  sec  1.25 MBytes  1.05 Mbits/sec  0.646 ms  0/898 (0%)  receiver
```

Figure 2: UDP throughput test (1 Mbit/s offered load, 10 s).

4.2.2 Efficiency:

Bandwidth Evaluation using Iperf
The result shows as follows

- **Average throughput:** 1.05 Mbit/s (close to the 1 Mbit/s target).
- **Packet loss:** 0 / 898 datagrams (0 %).
- **Jitter:** 0.646 ms (negligible).

These results show our router introduces no loss or delay under a 1 Mbit/s UDP load. CPU utilization remained below 5 % (measured on the router VM), indicating the forwarding path is efficient for this traffic level.

5 Cascaded Router Configuration

We have not tried cascaded router configuration due to the limitations of Virtual Box having a single internal adapter. But, we strongly think it works because all the services support dynamic ip address and subnet masks.

6 Sign Off

The code is original and there is not copied directly from internal sources.

All group members contributed equally to the project. Vamsi Alluri, Edward Kelley, Yunhu Kim, Joseph Luo

---