Example workloads for networking applications

Pravin Shinde

Contents

Chapter 1

Workloads

1.1 Why Haskell?

One of the common question that appears is why use Haskell for modeling NIC device?

1.2 The Big picture

Applications have some requirements, and hardware has some features

1.2.1 Application requirements:

Following are the possible application requirements.

1. Number of listen ports
2. Number of incoming clients
3. Number of outgoing active connections expected
4. Average connection lifetime
5. Average number of packets in each connection
6. Expected volume of incoming traffic
7. Expected volume of outgoing traffic
8. Protocol type
9. Number of cores involved
10. Memory locations
11. Low latency preference

12. High throughput preference
13. Low jitter requirements

1.2.2 Hardware features:

1. Dedicated RX/TX queues
2. Hardware filters
3. Large receive offload
4. Large send offload
5. TCP offload

1.2.3 Questions for the hardware model:

The Mathematical (ie: Haskell based) model of NIC hardware should lead to many alternate network stack configurations. And one should be able ask following questions from any of these alternate network stack configurations.

1. What will be the **flow setup cost**?
2. What will be the **Flow tear-down cost**?
3. What are the **scalability limits**? (ie: How many parallel flows can be handled?)
4. What is the **Scale up cost** when resources are increased during application execution?
5. What is the **Scale down cost** when resources are decreased/ removed during application execution?
6. What is the **packet classification/processing cost**?
7. What is the **latency** in delivering single packet?
8. How many packets can be delivered in one second. Eg: **Throughput**.

1.2.4 Application: DNS Server

Requirements:

1. UDP protocol
2. Single listen port
3. Large number of incoming clients
4. Single packet request/response

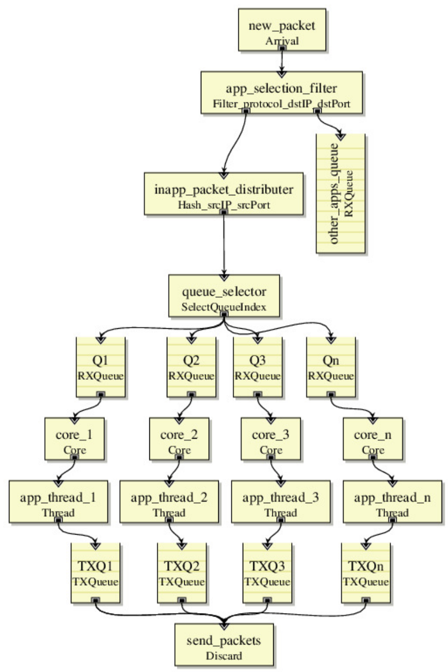

Figure 1.1: Ideal configuration for DNS server

5. Preference to throughput

6. Load balancing with multiple cores

Example of this server will be bind

Ideal hardware setup

1. One dedicated RX queue for each load balancing core.

2. Hardware filter for (protocol, destination IP, destination port) to separate packets for this particular application.

3. Ideal : Give each separated packet to next core in round-robin fashion

Alternate setup: 1

1. Hash (source ip, source port) (assuming uniform distribution of hash values)

2. Use hash to select one of the dedicated RX queues.

Alternate setup: 2

1. Use one RX queue and one filter (protocol, destination IP, destination port) for this application, and let all the cores share the same RX queue. (not a good idea due to contention on updating RX)

1.2.5 Application: HTTP server (eg: apache)

A server responding with small sized static pages

Requirements:

1. TCP protocol
2. Single listen port
3. Preference to throughput
4. Large number of incoming clients
5. Small request, larger response.
6. Load balancing with multiple cores

Ideal hardware setup

1. One dedicated RX queue for each load balancing core.
2. Hardware filter for (protocol, destination IP, destination port)
3. Ideal : Give each new connection to next core in round-robin fashion

Alternate setup: 1

1. Hash (source ip, source port)
2. Use hash to select one of the dedicated RX queues.

Alternate setup: 2

1. Use syn filter to separate syn packets
2. Give all syn packets to load balancer
3. Load balancer will distribute them in round-robin manner
4. Insert new flow directing filter to ensure rest of the packet of this connection goes directly to proper core.

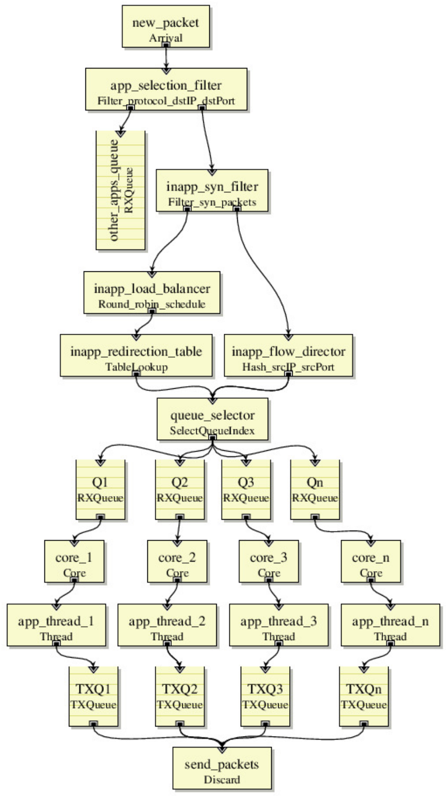

Figure 1.2: Ideal configuration for HTTP server

1.2.6 Application: Web crawler

Requirements:

1. HTTP/TCP Protocol
2. Large number of outgoing connections
3. Large incoming data
4. Relatively small connection lifetime (HTTP requests)
5. Scale by adding more cores running same application with different targets
6. Preference to throughput

Ideal hardware setup

1. A dedicated RX queue for every core involved in load-balancing

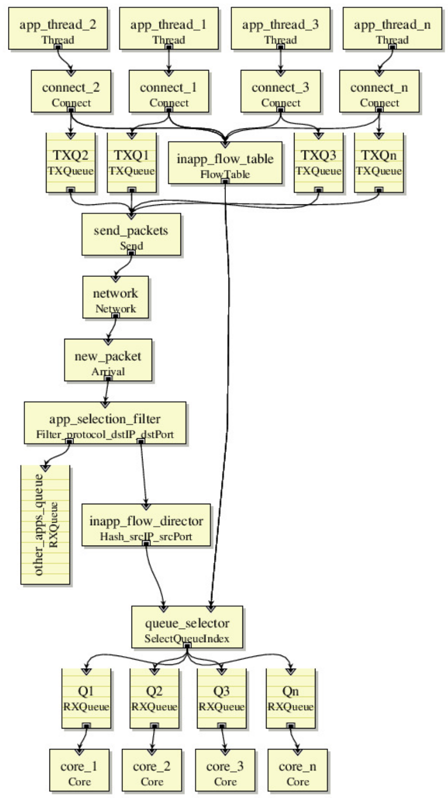

Figure 1.3: Ideal configuration for Web crawler

2. A dedicated hardware filter for every outgoing connection
3. Filtering based on $(protocol, src - ip, src - port, dest - IP, dest - port)$
4. Dedicated queue for each load-balancing core

1.2.7 Application: Web Proxy

Requirements:

1. HTTP/TCP Protocol
2. Single listen port
3. Large number of incoming connections
4. Large number of outgoing connections
5. Relatively small connection lifetime for outgoing connections.

6. Larger connection lifetime for incoming connections (client connections)
7. Scaling??
8. Preference to latency and throughput (not sure)
9. With high probability, Incoming and outgoing connections are on different interfaces
10. Examples: Squid

Ideal hardware setup

1. One dedicated hardware filter for incoming connections
2. Filtering based on (protocol, destination IP, destination port)
3. Hashing to load balance connections hash(source IP, source port)
4. Dedicated queue for each load-balancing core
5. Distribution of connections across cores in round-robin fashion.
6. Does not make much sense to have a dedicated queues for outgoing connections.

1.2.8 Application: NFS filesystem client

A kernel code which connects to NFS server and gets the contents of files based on application requests.

Requirements:

1. UDP protocol
2. Single connect port (outgoing connection)
3. Preference to throughput
4. Small request, large response. (reading data)
5. Load balancing: Increase number of kernel threads doing IO over NFS. The queries and responses are marked by RPC transaction-IDs which can be used to map the responses to proper kernel thread.

Ideal hardware setup

1. One dedicated RX queue for each load balancing core.
2. Hardware filter for (protocol, destination IP, destination port)
3. Give each packet to proper kernel thread based on RPC transaction ID.

Alternate setup: 1

If there is only one application

1. Use hash(source IP, source port) to select the destination core.

1.2.9 Application: Key-value store

Requirements:

A multi-core key value store like memcached

1. Protocol??

Ideal hardware setup

1. How many filters?

1.2.10 Application: MPI application

A class of scientific applications which communicate with each other using runtimes like MPI and perform some computation in distributed fashion.

Requirements:

1. TCP Protocol
2. Small messages
3. Large number of messages
4. Preference to low latency
5. Scalability with number of nodes
6. Long connection lifetime. (Assuming all messages are using same channel established once per node)

Ideal hardware setup

1. How many filters?

1.2.11 Application: Database server (eg: mysql!!)

A server handling small number of clients with large number of queries

Requirements:

1. TCP protocol
2. Single listen port
3. Preference to throughput
4. Small request, large response.
5. Load balancing with ??

1.2.12 Application: Firewall

Requirements:

1. Protocol??

Ideal hardware setup

1. How many filters?

1.2.13 Application: Intrusion Detection System

Requirements:

1. Protocol??

Ideal hardware setup

1. How many filters?

test citation [1]

Bibliography

[1] Baumann, A., Barham, P., Dagand, P.-E., Harris, T., Isaacs, R., Peter, S., Roscoe, T., Schüpbach, A., and Singhania, A. The multikernel: a new os architecture for scalable multicore systems. In *Proceedings of the 22nd Symposium on Operating Systems Principles* (New York, NY, USA, 2009), SOSP '09, ACM, pp. 29–44.