BTP Final Presentation

Adaptive Load Balancing in P4: The Power of Promethee-Prometheus and Probes

By:
Mangesh Dalvi
Yukta Salunkhe

Mentor:
Dr. Anish Hirwe

Introduction

- **Digital Age Growth:**
  - Industrial IoT, cloud computing, AI, and machine learning are driving network traffic.
- **Network Challenges:**
  - Traditional data planes lack flexibility and adaptability.
  - Fixed functions and limited protocol sets hinder performance and efficiency.
  - Vendor inflexibility leads to slow and expensive development.
- **Data Plane Programmability**
  - Opens new avenues for customization.
  - Enables efficient traffic management for virtual machines (VMs).

Motivation for Studying Data Planes, P4, and Virtualized Environments

- Network performance directly affect application performance. To keep applications running smoothly, researchers are exploring various approaches:
  - **Specialized Hardware**: Utilizing hardware like FPGAs and programmable ASICs to offload critical data plane operations (packet forwarding, processing) from the CPU.
  - **Kernel Bypass Techniques:** Techniques like SR-IOV, DPDK, and Netmap eliminate bottlenecks caused by context switching, packet copying, and interrupts between the kernel and user space.
  - **Efficient Data Structures:** Fast and Efficient data structures within the data plane allows for faster data lookup and manipulation.
  - **Data Plane Programmability (P4):** P4 empowers network engineers to customize network behavior. Its flexibility allows for faster detection of network disruptions and optimal rerouting of traffic

Load Balancing in Data Plane

Why Data Plane?

- Horizontal scaling for application servers improves performance.
- Line-rate performance for high traffic volumes.
- Customizable logic for directing traffic based on server load and other factors.
- Additional functionalities: traffic prioritization, real-time monitoring, filtering.

Load Balancing in Data Plane

Studying load balancing in P4 and virtualized environments is crucial because of limitations of Traditional Load Balancing :

- Uniform distribution may not reflect server load
- Scalability issues due to centralized systems
- Lacks granularity in considering server metrics
- Inefficient for dynamic cloud environments

Literature Review

- SmartNICs in Data Centers
  - Azure Accelerated Networking
  - P4xos: Consensus as a network service
- Reliability and Failure Recovery in Data Plane
  - BLINK: Fast Connectivity Recovery
  - P4 Neighbor: Efficient Link Failure Recovery
- Load Balancing and Congestion Management
  - HULA: Hop-by-hop Utilization-Aware Load balancing
  - CONGA: Distributed Congestion-Aware Load Balancing

Research Paper	Problem	Targeting Environment	Solution
Azure Accelerated Networking	Hardware performance with software programmability.	Data Centers	Sdn stack implemented on FPGA-based SmartNIC. Use of GFTs (Generic Flow Table)
P4xos: Consensus as a network service	Network I/O bottlenecks in software-based implementation of Paxos.	Distributed Systems	Offer consensus as a service within existing network switching devices. Use of custom headers.
BLINK: Fast Connectivity Recovery	Fast recovery from remote link failures.	-	Monitoring TCP retransmission pattern. Failure Detection and rerouting in data plane.
P4Neighbor: Efficient Link Failure Recovery	Switch storage overhead while tackling link failures.	-	Calculator and store only the backup paths for the neighbours.
HULA: Hop-by-hop Utilization-Aware Load balancing	Static load balancing is inefficient. Congestion aware LB cause memory overhead.	Fat Tree Topology	Link utilization based load balancing. Use of custom headers to propagate the max utilization of best path downstream.

Load Balancing in Data Plane

Studying load balancing in P4 and virtualized environments is crucial because of limitations of Traditional Load Balancing :

- Uniform distribution may not reflect server load- P4's programmability enables it to adapt to these changes in real-time, ensuring optimal resource utilization.
- Scalability issues due to centralized systems- P4-based solutions can be distributed across the network, enabling efficient load balancing for a vast number of virtual machines (VMs).
- Lacks granularity in considering server metrics- P4’s customizability allows for fine-grained control over data plane operations.

Problem Statement

The Problem

- Current load balancing solutions burden both the central controller and data plane with communication overhead.
- Traditional algorithms lack the dynamic flexibility to adapt to real-time network traffic conditions.

The Solution:

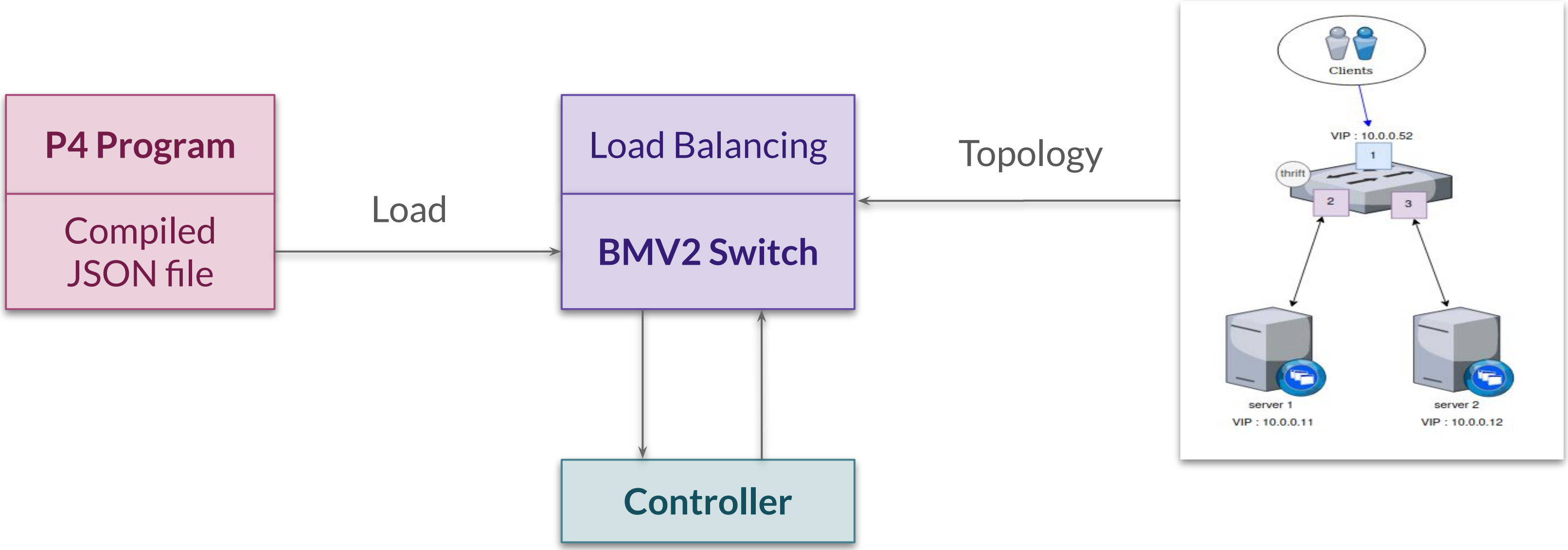
A dynamic, "load-aware" load balancing technique embedded entirely within the data plane.

1. Probe Enhanced Weighted Round Robin Load Balancing in P4.
2. Promethee-Prometheus Driven Load Balancing Using P4.

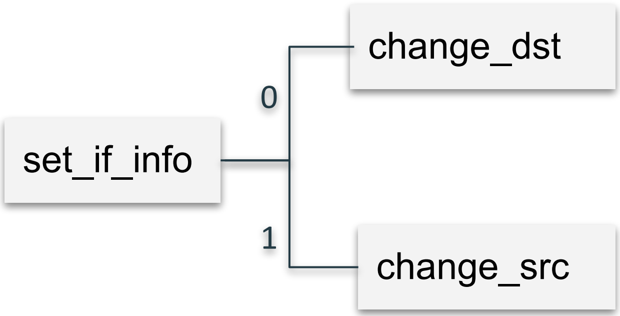
Design Challenges

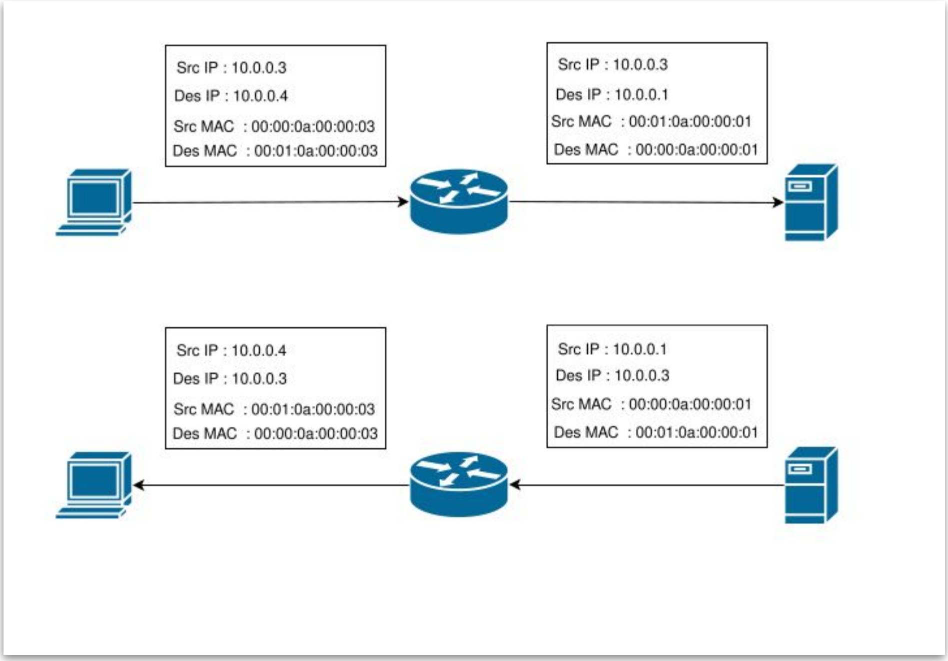
1. **Dynamic Weight Adjustment:** The load balancer must dynamically adjust weights based on real-time resource utilization metrics while minimizing overhead and ensuring smooth operation.
2. **Static or Controller Dependent Dynamic Updates**: Traditional hardware load balancers offer limited flexibility. Heavy reliance on the controller for sending and received updates can become a bottleneck if the controller is overloaded.
3. **Failure Handling**: The load Balancer should not route any new traffic to the failed VM, in order to ensure availability of the system.
4. **Programmability**: Developing hardware load balancers is cumbersome. Their fixed functionality (”one-size-fits-all”) requires network operators to deploy them as it is.
5. **Flowlet load balancing:** In dynamic environments, where workload conditions constantly change, static allocation of flows to virtual machines (VMs) may not remain optimal over time.

P4


P4 Program
Compiled JSON file
Load
Load Balancing
BMV2 Switch
Topology
Controller
Clients
VIP : 10.0.0.52
thrift
1
2
3
server 1
VIP : 10.0.0.11
server 2
VIP : 10.0.0.12

NATing


change_dst
0
set_if_info
1
change_src


Src IP : 10.0.0.3
Des IP : 10.0.0.4
Src MAC : 00:00:0a:00:00:03
Des MAC : 00:01:0a:00:00:03
Src IP : 10.0.0.3
Des IP : 10.0.0.1
Src MAC : 00:01:0a:00:00:01
Des MAC : 00:00:0a:00:00:01
Src IP : 10.0.0.4
Des IP : 10.0.0.3
Src MAC : 00:01:0a:00:00:03
Des MAC : 00:00:0a:00:00:03
Src IP : 10.0.0.1
Des IP : 10.0.0.3
Src MAC : 00:00:0a:00:00:01
Des MAC : 00:01:0a:00:00:01

Maintaining Connection

```
//Calculate flowId using hash
bit<16> flow_id;
hash(flow_id, HashAlgorithm.csum16, (bit<16>)0, {
hdr.ipv4.srcAddr,
hdr.ipv4.dstAddr,
hdr.tcp.srcPort,
hdr.tcp.dstPort,
hdr.ipv4.protocol},(
bit<16>)65535);
}
```

```
//Populate the table entry with egress port for the flow

//mark to check if the flow is already seen
flowId_to_is_assigned.write((bit<32>) flow_id, 1);

//store the mapping between flowId and egress spec
flowId_to_port_assigned.write((bit<32>) flow_id,
(bit<9>) standard_metadata.egress_spec);

//get the port assigned directly from above register if
present
```

Breaking Flows Into Flowlets

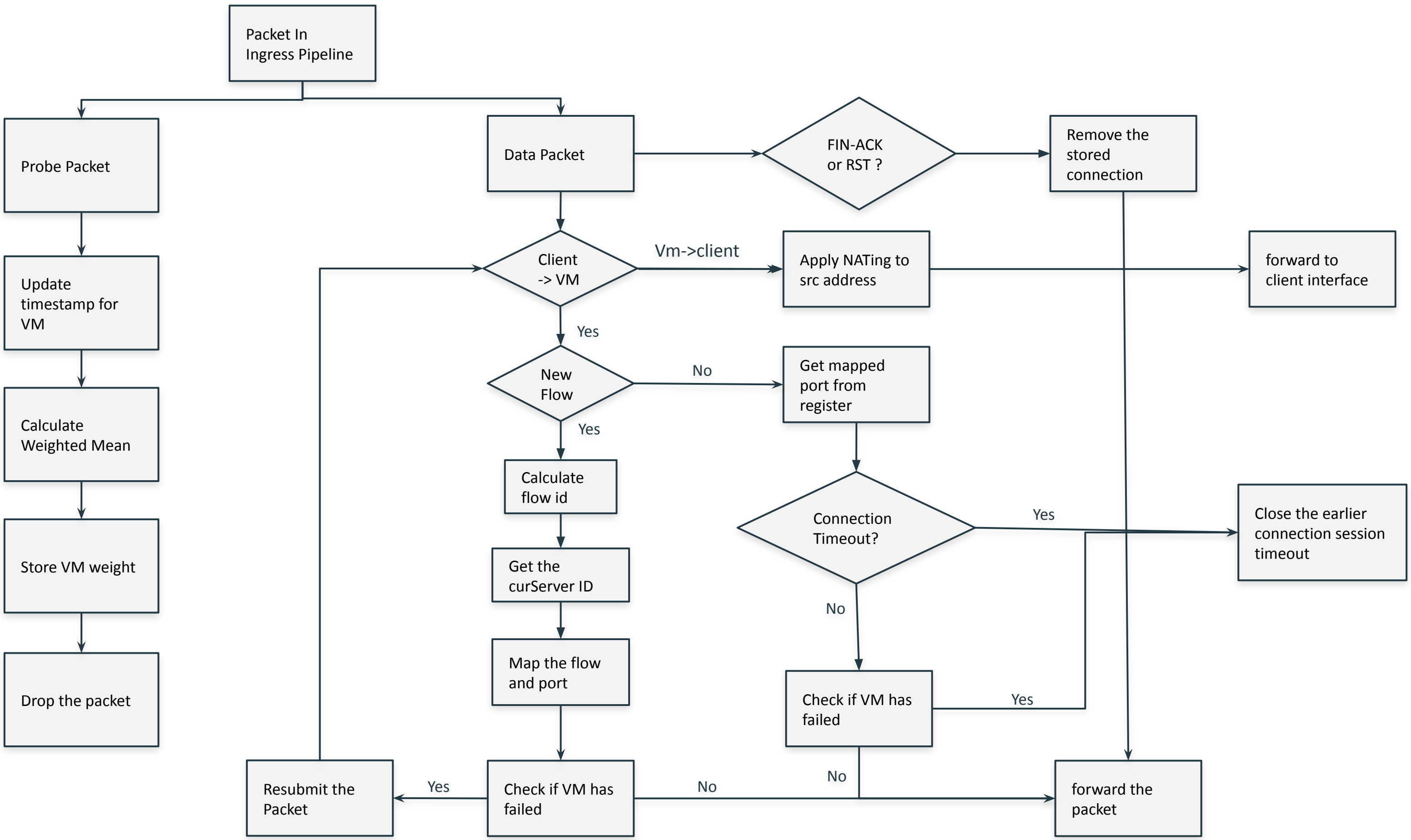
- Set aging threshold*.
- Monitor packet arrival for each flow.
- Check if elapsed time exceeds the threshold.
- If so, terminate the original session.
- Start new session, assign to best server.
- This helps to enhance load balancing due to dynamic allocation.

*Threshold value chosen as 500s

Probe Enhanced Weighted Round Robin Load Balancing in P4

Overview

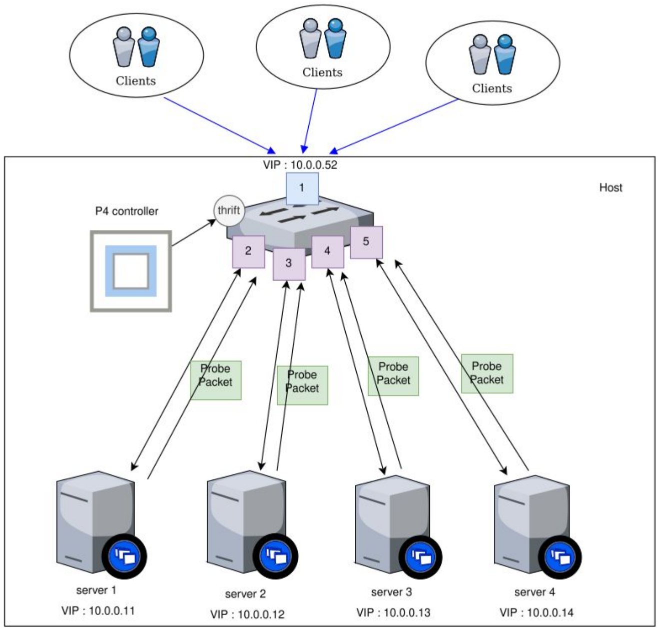
- VM sends metrics in probe packets, inside the custom header.
- The protocol field of IP header is set to 0x42 for the custom header.
- Weighted Mean is calculated in switch and is normalized to a range of 0-12.
- These weights are considered to be proportional to the capacity of the VMs, i.e. number of connection that can be made to VM.
- Leveraged Dynamic Weighted Round Robin in switch.



Packet In Ingress Pipeline
Probe Packet
Update timestamp for VM
Calculate Weighted Mean
Store VM weight
Drop the packet
Data Packet
FIN-ACK or RST ?
Remove the stored connection
Client -> VM
Vm->client
Apply NATing to src address
forward to client interface
Yes
New Flow
No
Get mapped port from register
Yes
Calculate flow id
Get the curServer ID
Map the flow and port
Connection Timeout?
Yes
Close the earlier connection session timeout
No
Check if VM has failed
Yes
No
Resubmit the Packet
Yes
Check if VM has failed
No
forward the packet

Introduction	Background	Approach 1	Edge Cases	Evaluation	Conclusion

Probe Enhanced Weighted Round Robin Load Balancing in P4

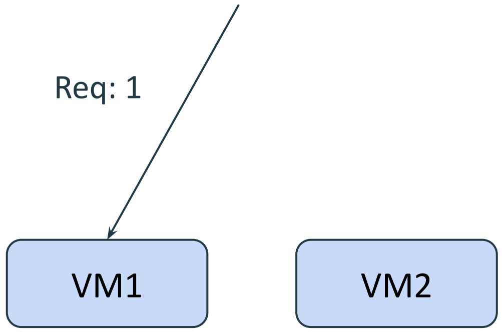



```
header probe_t {
        bit<4> id;
        bit<4> cpu_percent_left;
        bit<4> memory_percent_leftt;
        bit<4> link_util_left;
}
```

Weights = 6 (network), 3 (cpu), 1(memory)
Weighted mean lies in range of 0-100.

Link_bandwidth = 1000Mbps
$T_{probeFreq}$ = 5s

Weighted Round Robin



Vm weights

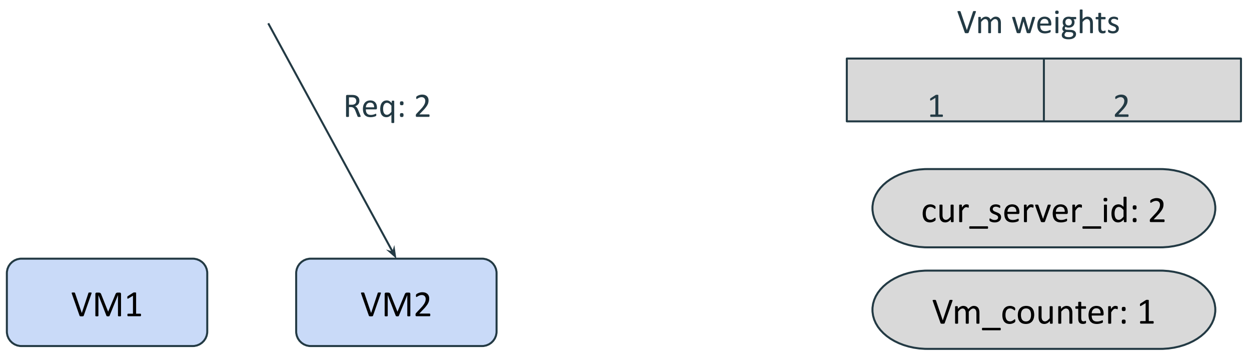
1	2

cur_server_id: 1

Vm_counter: 1

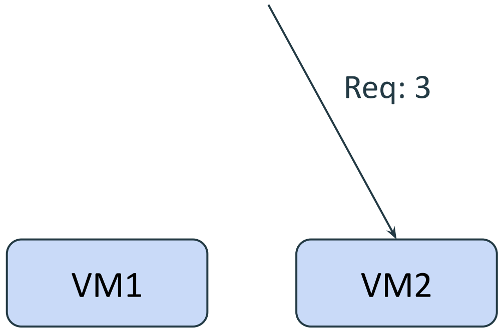
Req1-VM1				….

Weighted Round Robin



Req1-VM1	**Req2-VM2**			….

Weighted Round Robin



Vm weights

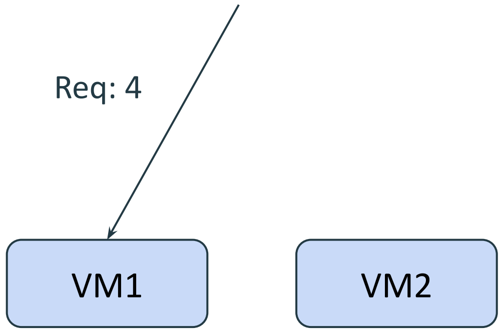
1	2

cur_server_id: 2

Vm_counter: 2

Req1-VM1	Req2-VM2	**Req3-VM2**		….

Weighted Round Robin



Vm weights

1	2

cur_server_id: 1

Vm_counter: 1

Req1-VM1	Req2-VM2	Req3-VM2	**Req4-VM1**	….

What should ideal $T_{probeFreq}$ be?

Should be frequent enough for optimal load balancing
At the same time, should not be too high to overwhelm the network
Through experiment, considered $T_{probeFreq}$ = 5s most suitable.

How are the weights calculated?

Weights can be calculated using AHP, by giving pairwise preference matrix of metrics, which depends on the application running on the server

Edge Case: What if weight of any VM comes out to be 0?

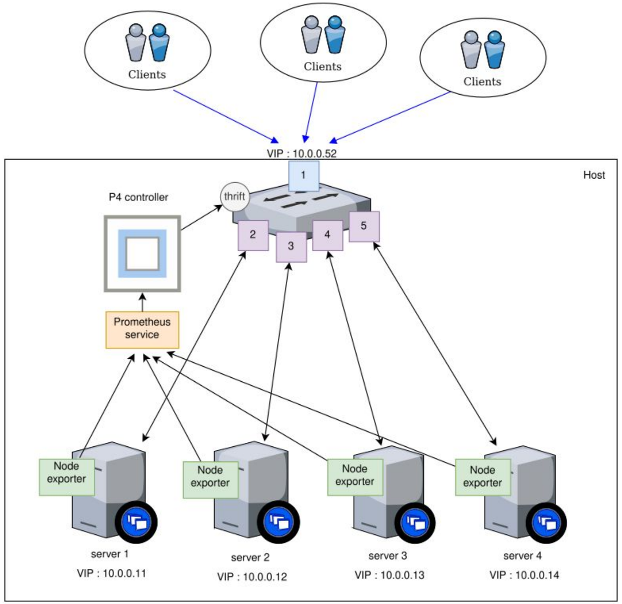
No flow is assigned to that VM. The cur server id value is incremented and the packet is resubmitted to the ingress pipeline until it finds a valid VM (one whose weight is >0).

Promethee-Prometheus Driven Load Balancing Using P4

Overview

- Monitors: Gathers real-time resource metrics from VMs **using prometheus**
- Decides: Uses **AHP and Promethee II** to rank VMs based on utilization.
- Directs: **Guides new traffic to the most underutilized** VM via P4 switch.
- Adapts: Continuously updates load distribution based on changing demands.
- Optimizes: Enhances performance and resource utilization.

Promethee-Prometheus Driven Load Balancing Using P4



- **Prometheus** : To collect system metrics from virtual machines.
- **Promethee II :** For ranking VMs.
- **Analytic Hierarchy Process (AHP)** : To determine weights for decision criteria.

Prometheus Client - Queries

cpu load : avg_over_time(100 - (avg by(instance)(irate(node_cpu_seconds_total{mode="idle"}[1m])) * 100)[1m:])

ram usage : (1 - (node_memory_MemFree_bytes + node_memory_Cached_bytes + node_memory_Buffers_bytes) / node_memory_MemTotal_bytes) * 100

Network transmitted bytes : ((rate(node_network_transmit_bytes_total[1m]) * 8) / (1000 * 1024 * 1024)) * 100

Analytic Hierarchy Process (AHP)

-	cpu_load	ram_usage	network_usage
cpu_load	1	4	1/2
ram_usage	1/4	1	1/6
network_usage	2	6	1

we obtained consistency ratio (CR) of 0.009 which is less than 0.1, indicating that the judgment matrix is consistent.

Promethee II

Using Promethee II min_utilized register value gets updated. min_utilized register that indicates the port associated with the virtual machine (VM) possessing the lowest current resource consumption.

1. Idle Timeout Mechanism

- To better load balance the occasional flows.
- The server which was better at the time of allocation, may not still be the better one for handling the flow.
- Idle timeout mechanism is used to age out the older unused connection entries.

TIME_THRESHOLD = 500s

In ingress

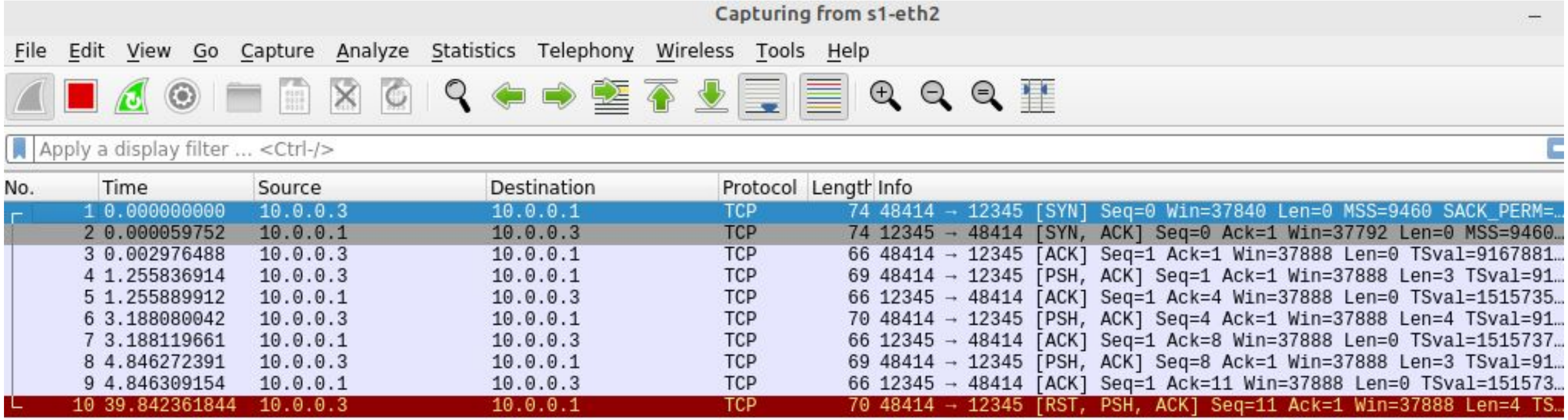
```
action clone_packet() {
        const bit<32> REPORT_MIRROR_SESSION_ID = 500;
        clone(CloneType.I2E, REPORT_MIRROR_SESSION_ID);
}

if(cur_time - last_time > TIME_THRESHOLD) {
        clone_packet();
        //send reset packet to server
        hdr.tcp.rst = 1;
        flowId_to_port_assigned.read(standard_metadata.egress_spec, (bit<32>) flow_id);
        ...
}
```

In egress

// send reset packet to source (client)

```
if(standard_metadata.instance_type == PKT_INSTANCE_TYPE_INGRESS_CLONE) {
        set reset bit 1
        hdr.tcp.rst = 1;
        //swap src, dst -> ip, mac, port
        ...
        standard_metadata.egress_spec = 1;
}
```

1. Idle Timeout Mechanism



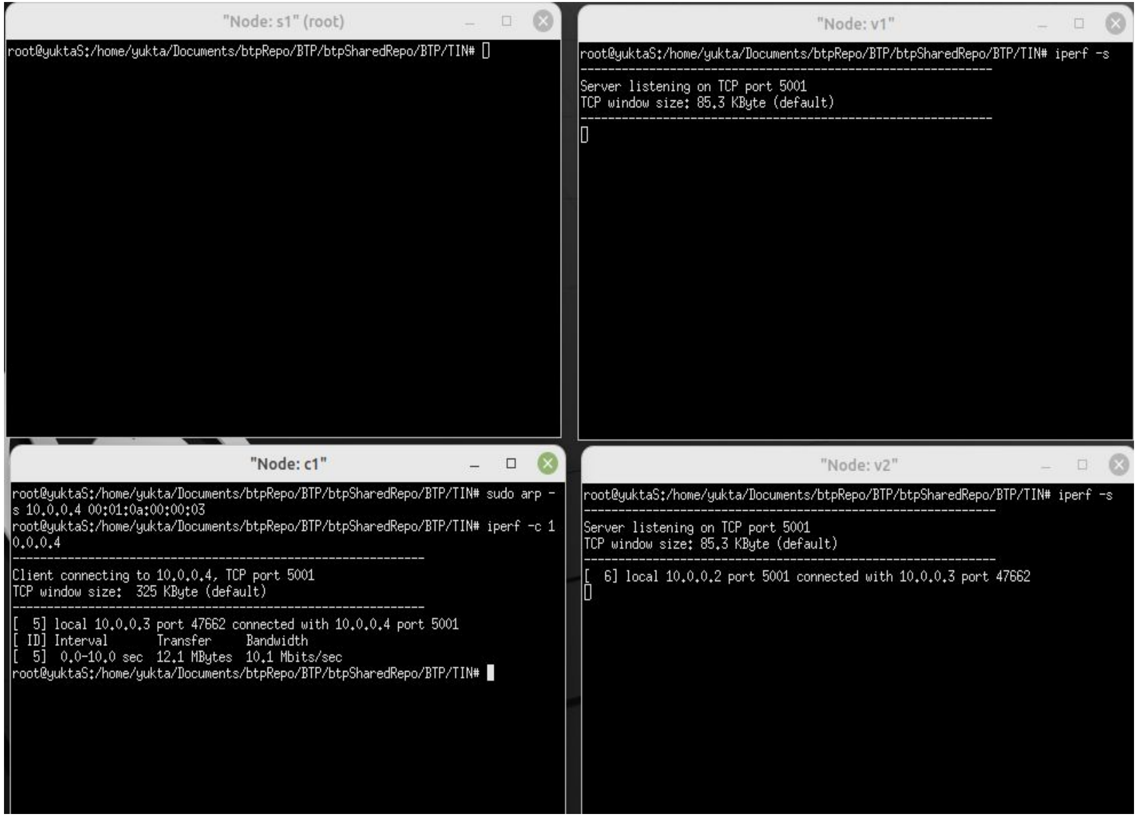
2. Failure of VM

- Prevent routing traffic to failed or malfunctioning VMs, ensuring high availability and reliability of the system.

$T_{probeThreshold}$ = 2*probeFreq (= 10s, considering 5s as the Probe Freq).

In ingress

```
if((cur_time-last_update_time) > PROBE_TIMEOUT) {
        has_failed.write((bit<32>)_cur_server_id, 1);
        check_failed = 1;
}

if(check_failed == 1){
        _cur_server_id = _cur_server_id + 1;
        cur_server_id.write((bit<32>) 0,_cur_server_id);
        vm_conn_counter.write((bit<32>)0, 0);
        resubmit_preserving_field_list((bit<8>)1);
}
```



2. Failure of VM

```
controller.register_write('vm_weights',1, 0)
controller.register_write('vm_weights',2, 2)
```

```
Condition "_vm_weight <= 0" (node_18) is true
```

```
Primitive resubmit_preserving_field_list((bit<8>)1)
```

```
Resubmitting packet
Processing packet received on port 1
```

Demo

Evaluation Setup & Methodology

Hardware:

* PC: 11th Gen Intel i5-1135G7 (8 cores), 16GB RAM
* Virtualization: VirtualBox with four virtual machines (VMs)

Inside virtual machine:

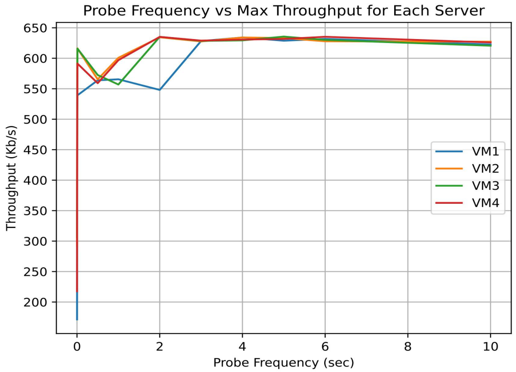
* VMs: Ubuntu 16.04.7 LTS (Xenial Xerus)
* Base memory: 2405 MB
* Processor: 1 (100% execution cap)
* TCP Server: Python socket library, listening on port 5007

Client-Server Interaction:

* Client sends 250-byte string every 1 second
* Server responds with 5 MB of data
* Duration: 30 seconds per client
* Load Increase: Clients connect in 0.5-second intervals

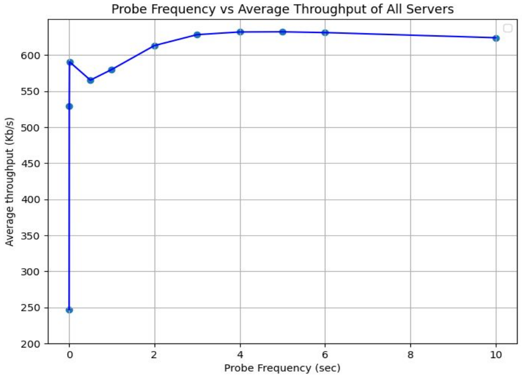
Evaluation Setup & Methodology - Approach 1

Probe Frequency vs Server Throughputs



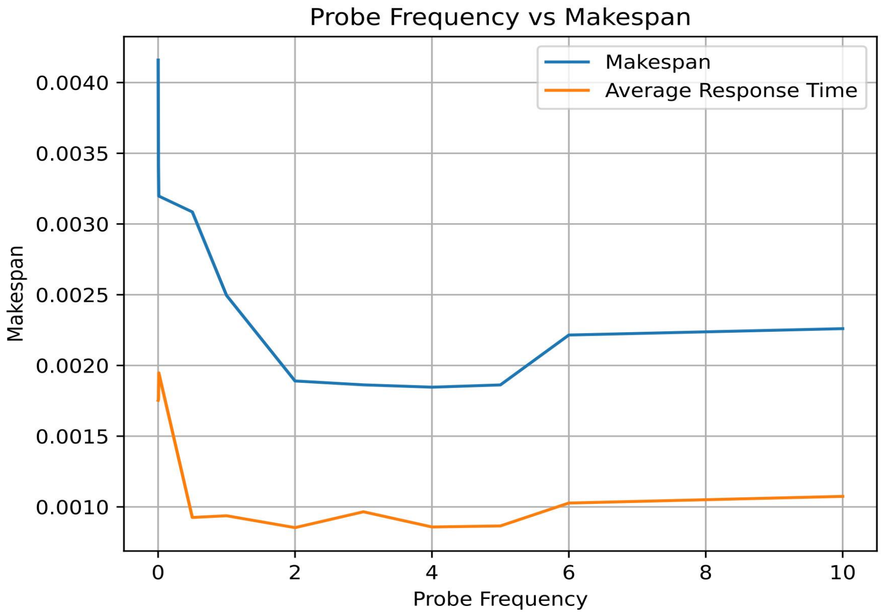
With increasing number of probe packets, the network gets congested, thereby decreasing the throughput.

Probe Frequency vs Average Throughput



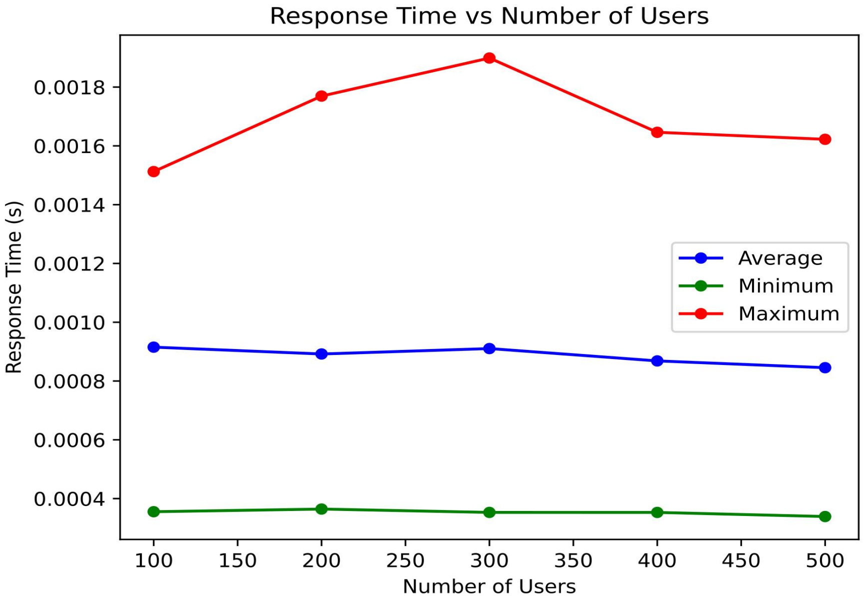
Avg throughput becomes roughly constant for all probe frequencies greater than 4.

Probe Frequency vs Makespan



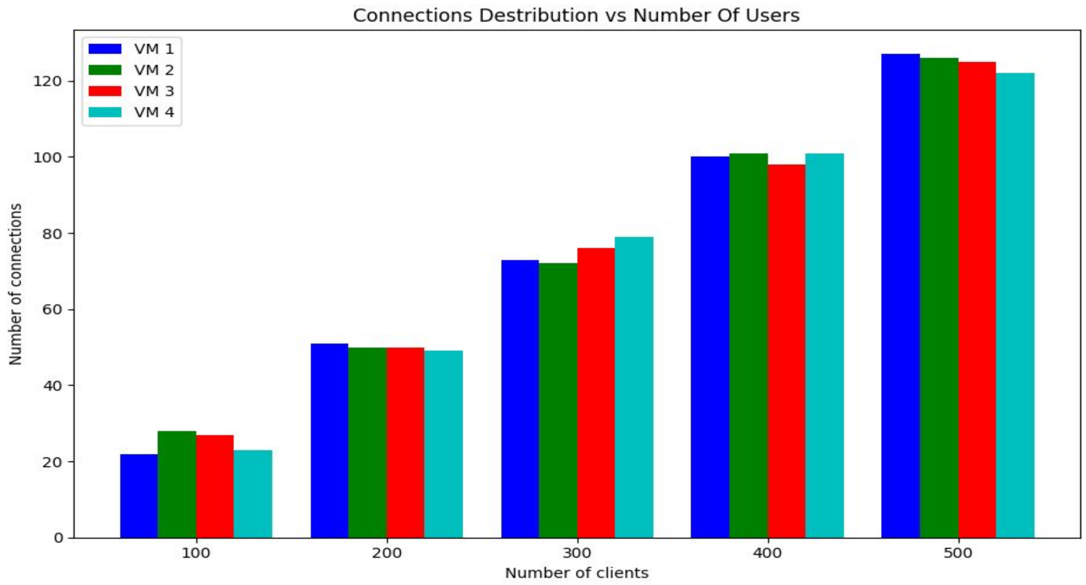
- Makespan refers to the maximum time it takes for a single client request to be fully handled and a response returned.
- Makespan is less for higher probe frequencies. A shorter makespan directly translates into faster response times from the clients’ perspective.

Response Time vs Clients



- There isn’t much variation in the average response time of server even when the number of client connections are increased till 500.

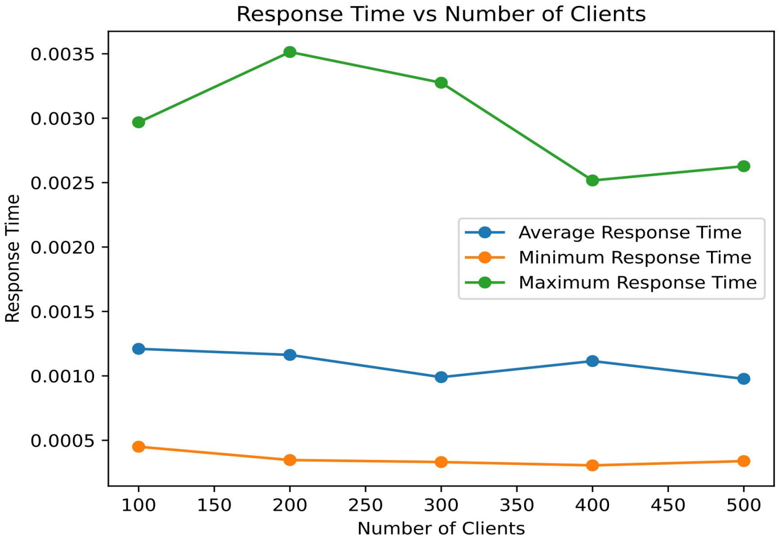
Connections vs Clients



- Distribution by load balancer during equivalent load on each machine is fairly equal.

Evaluation Setup & Methodology - Approach 2

Response Time vs Number of Connections



- Indicates a slightly elevated average response time compared to the previous approach.
- This increase may be attributed to the Prometheus server sharing resources with the virtual machine under evaluation.
- We expect that testing within a fully isolated environment should yield better results.

Comparing Our Approach with other load balancers implemented in data plane
Setup : 2 VM, 100 clients and stress is 7 workers spinning malloc()

Load Balancer	Makespan(s)	Avg Response Time(s)	Throughput (Kb/s)	Distribution
Hash based	0.0027873992919921	0.0013046917915344424	321.7085, 332.34453	48, 52
Hash with stress	0.25374268492062	0.013996536783908022	320.4093, 363.096	47, 53
Round Robin	0.0015149513880411	0.0010721955299377443	477.117, 476.9	50, 50
RR with stress	0.2559712429841359	0.011239712076618072	477.4921, 478.1265	50, 50
Probe enhanced WRR	0.0005086819330851	0.0002260793050130208	510.5625, 510.8984	50, 50
Probe enhanced WRR with stress	0.0019080718358357	0.0004536112149556475	610.6585, 466.0875	58, 42
Promethee-Prometheus based	0.0007852395375569	0.000386993646621	943.74, 813.68	67, 33
Promethee-Prometheus based with stress	0.0016802390416463	0.0004763094584147136	943.77, 403.71	83, 17

Robustness

- Does not depends on number of virtual machine.
- Idle timeout, connection tracking, Handling RST and FIN packet in switch.
- Metrics based dynamic load balancing in data plane.

Limitations

- Given that it's a hardware offload approach, it's difficult to assess its efficiency on a standard PC.
- Being open source and new technology, many version and libraries with varying dependencies exist, thus making it difficult to follow online documentations.

Conclusion

- Explored the implementation of dynamic load balancing techniques in the data plane using P4 switch.
- Significant advantages over traditional application-layer approaches, especially: Speed & Granularity, Scalability & efficiency, Flexibility & Resilience.
- Network operators can customize weight parameters for VMs to optimize performance for their specific use cases. Offloading load balancing algorithms to smart NICs has the potential to further improve efficiency by reducing CPU cycles required on the host.
- Overall, this BTP project provided us with valuable insights into advanced networking concepts and technologies, particularly those related to faster and more efficient network management.

References

1. Azure accelerated networking: SmartNICs in the public cloud, 2019
2. P4neighbor:Efficient link failure recovery with programmable switches, 2021.
3. Conga: Distributed congestion-aware load balancing for datacenters, 2014.
4. Hula: Scalable load balancing using programmable data planes, 2016.
5. Study the effect of parameters to load balancing in cloud computing, 2016.
6. P4xos: Consensus as a network service, 2020.
7. Blink: Fast Connectivity Recovery Entirely in the Data Plane
8. P4 based Load Balancing Strategies for Large Scale Software-Defined Networks