HEELS: A Host-Enabled eBPF-Based Load Balancing Scheme

Rui Yang*  
Marios Kogias†

eBPF workshop, SIGCOMM  
September 10 2023

*EPFL  
†Imperial College London
Layer 4 load balancer

TCP connection
L4 load balancer: **Centralized Design**

Maglev [NSDI ’16], SilkRoad [Sigcomm ’17], Katran [Meta]
L4 load balancer: **Centralized Design**

Maglev [NSDI ’16], SilkRoad [Sigcomm ’17], Katran (Meta)

---

**Efficiency**

- ✔️ load balancer has a global view

**Scalability**

- ✗ easily result in IO bottleneck
L4 load balancer: **Decentralized Design**

IPVS Kube-proxy (Kubernetes)
L4 load balancer: **Decentralized Design**

IPVS Kube-proxy (Kubernetes)

- **Efficiency** ✗
  - load imbalance

- **Scalability** ✓
  - Every node acts as a load balancer
ABSTRACT

Load balancers are a ubiquitous component of cloud deployments and the cornerstone of workload elasticity. Load balancers can significantly affect the end-to-end application latency with their load balancing decisions, and constitute a significant portion of cloud tenant expenses.

We propose CRAB, an alternative L4 load balancing scheme that eliminates latency overheads and scalability bottlenecks while simultaneously enabling the deployment of complex, stateful load balancing policies. A CRAB load balancer only participates in the TCP connection establishment phase and stays off the connection's datapath. Thus, load balancer provisioning depends on the rate of new connections rather than the actual connection bandwidth. CRAB depends on a new TCP option that enables connection redirection. We provide different implementations for a CRAB load balancer on different technologies, e.g., PA-DPRK, and bpf, showing that a CRAB load balancer does not require many resources to perform well. We introduce the connection redirection option to the Linux kernel with minor modifications, so that it can be shipped with the V3 images offered by the cloud providers. We show how the same functionality can be achieved with a vanilla Linux kernel using a Netfilter module, while we discuss how CRAB can work while clients and servers remain completely agnostic based on functionality added on the host.

Our evaluation shows that CRAB pushes the IO bottleneck from the load balancer to the servers in cases where vanilla L4 load balancing does not scale and provides end-to-end latencies that are close to direct communication while retaining all the scheduling benefits of stateful L4 load balancing.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

1 INTRODUCTION

Load balancing is ubiquitous: nearly all applications today running in datacenters, public clouds, at the edge, or as core internet services rely on some form of load balancing for both availability and scalability. Load balancing can have different forms, e.g., L4, L7, DNS-based etc. and can be implemented in hardware or in software. There has been considerable research on load balancing [5, 9, 16, 24, 35, 42, 45, 47–49] both from academia and industry due to the demands for mass deployments, high throughput, and low latency variability, but also the demands to lower provider resources specifically dedicated to it. For instance, Google reports that software-based load balancing can take up to 3–4% of a datacenter’s resources [16].

This paper focuses on internal load balancers, which are deployed between clients and servers within the same datacenter or public cloud. Internal load balancers can have a significant impact on the end-to-end latency both due to their load-balancing decisions and the intermediate hop, while also constituting a major part of the infrastructure costs for cloud tenants. A common pattern includes the deployment of an internal cloud service, placed behind an internal load balancer, that spawn new service instances according to load requirements and registers them with the load balancer, leading to seamless scalability and elasticity.

Figure 1 illustrates a sample cloud-based, two-tier application. Users using their browsers hit the public IP of the external load balancer and their requests end up being served by the two web servers. Those servers act as internal clients to the backend-servers that are behind the internal load balancer and communicate with a managed database service. This design pattern allows the web tier and the back-end tier to scale independently and remain agnostic to each other due to the use of the two load balancers. Similar examples of such design patterns for services (or microservices) include ML inference to create recommendations, a user authentication microservice [23], generic application servers, and any workload orchestrated in containers such as Kubernetes [26].

Internal load balancers must be able to handle low-latency, high-throughput IPv4/46 typically implemented on protocols such as gRPC [24], Thrift [57], HTTP, or even custom protocols.
L4 load balancers: **best of both worlds**

- Efficiency
- Scalability

CRAB

2020
L4 load balancers: **best of both worlds**

- Efficiency ✓
- Scalability ✓

CRAB

2020

TCP handshake

TCP data traffic

Server IP
L4 load balancers: **best of both worlds**

CRAB

2020

CRAB is designed for the *internal* cloud workloads
Poor deployability
Poor deployability

Requires a customized load balancer
incompatible with real-world ones
Poor deployability

Requires a customized load balancer
incompatible with real-world ones

Requires kernel changes at client side
through direct kernel patching or module loading
Comparison of existing L4 load balancers

<table>
<thead>
<tr>
<th></th>
<th>Centralized Designs</th>
<th>Decentralized Designs</th>
<th>CRAB 2020</th>
</tr>
</thead>
<tbody>
<tr>
<td>Efficiency</td>
<td>✔</td>
<td>✗</td>
<td>✔</td>
</tr>
<tr>
<td>Scalability</td>
<td>✗</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Deployability</td>
<td>✔</td>
<td>✔</td>
<td>✗</td>
</tr>
</tbody>
</table>
Comparison of existing L4 load balancers

<table>
<thead>
<tr>
<th>Feature</th>
<th>Centralized Designs</th>
<th>Decentralized Designs</th>
<th>CRAB 2020</th>
<th>HEELS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Efficiency</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Scalability</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Deployability</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
</tr>
</tbody>
</table>
HEELS bypasses the centralized load balancer.

**HEELS: A Host-Enabled eBPF-Based Load Balancing Scheme**

HEELS is designed for **internal** cloud workloads.
HEELS is readily deployable on the public cloud

Deployability

Compatible with a wide range of LBs
Both open-source and proprietary ones

Requiring no kernel modifications
Leveraging different eBPF hooks

HEELS: A Host-Enabled eBPF-Based Load Balancing Scheme
2023
HEELS is readily deployable on the public cloud

Deployability ✓

Compatible with a wide range of LBs
Both open-source and proprietary ones

Requiring no kernel modifications
Leveraging different eBPF hooks

HEELS: A Host-Enabled eBPF-Based Load Balancing Scheme
2023
Different mechanisms of L4 load balancers

Packet-encapsulation LB
Katran from Meta

Packet-rewriting LB
AWS Network Load Balancer
Different mechanisms of L4 load balancers

Packet-encapsulation LB
Katran from Meta

Packet-rewriting LB
AWS Network Load Balancer
HEELS is compatible with a wide range of LBs

<table>
<thead>
<tr>
<th>CIP</th>
<th>Client IP</th>
</tr>
</thead>
<tbody>
<tr>
<td>VIP</td>
<td>Load Balancer IP</td>
</tr>
<tr>
<td>DIP</td>
<td>Server IP</td>
</tr>
</tbody>
</table>

HEELS: A Host-Enabled eBPF-Based Load Balancing Scheme

HEELS relies on a customized TCP option
HEELS is compatible with a wide range of LBs.

HEELS: A Host-Enabled eBPF-Based Load Balancing Scheme

2023

HEELS requires no modifications to the load balancer itself.
HEELS is compatible with a wide range of LBs.

**HEELS: A Host-Enabled eBPF-Based Load Balancing Scheme**

2023

The server modifies the incoming SYN packet *before* TCP/IP stack.
HEELS is compatible with a wide range of LBs

The server modifies the incoming SYN packet *before* TCP/IP stack

HEELS: A Host-Enabled eBPF-Based Load Balancing Scheme

2023
HEELS is compatible with a wide range of LBs

HEELS maintain its own state for TCP connections

HEELS: A Host-Enabled eBPF-Based Load Balancing Scheme

2023
HEELS is compatible with a wide range of LBs

HEELS: A Host-Enabled eBPF-Based Load Balancing Scheme

Kernel State
src: CIP
dst: VIP

Kernel State
src: CIP
dst: DIP

HEELS State
DIP → VIP

HEELS maintain its own state for TCP connections
HEELS is compatible with a wide range of LBs

HEELS: A Host-Enabled eBPF-Based Load Balancing Scheme

2023

HEELS rewrites every outgoing packet to match the kernel state of the other end
HEELS is compatible with a wide range of LBs

HEELS: A Host-Enabled eBPF-Based Load Balancing Scheme

2023

HEELS rewrites *every* outgoing packet to match the kernel state of the other end
HEELS is compatible with a wide range of LBs

HEELS: A Host-Enabled eBPF-Based Load Balancing Scheme

2023

HEELS rewrites *every* outgoing packet to match the kernel state of the other end
HEELS is compatible with a wide range of LBs

HEELS: A Host-Enabled eBPF-Based Load Balancing Scheme

2023

Direct communication after the handshake
Different mechanisms of L4 load balancers

Packet-encapsulation LB
Katran from Meta

Packet-rewriting LB
AWS Network Load Balancer
HEELS is readily deployable on the public cloud

Deployability ✓

Compatible with a wide range of LBs
Both open-source and proprietary ones

Requiring no kernel modifications
Leveraging different eBPF hooks

HEELS: A Host-Enabled eBPF-Based Load Balancing Scheme
2023

[Diagram showing TCP handshake and data traffic]
HEELS implements its design using a set of eBPF programs.

**SOCKOPS**

- Adding and extracting TCP options

**Traffic Control (TC)**

- Rewriting ingress SYN packet at the server

eBPF programs for handshake phrase
HEELS implements its design using a set of eBPF programs.

Traffic Control (TC)
Rewriting egress packets at end hosts

eBPF programs for data transmission phrase
HEELS implements its design using a set of eBPF programs

**SK_STORAGE**

Storing HEELS state at end hosts
Requires no changes to kernel state

Per-connection eBPF data structure
Same lifetime as the TCP connection

Created at TCP handshake phrase and accessed throughout the connection
We evaluate **HEELS** on both local testbed and public cloud

**Implementation**

- ~1.2k lines of eBPF code
- Supports both Katran and AWS Network Load Balancer (NLB)

**Questions**

1. **Q1**: Does HEELS bring significant overhead?  
   Deploy with Katran on local testbed

2. **Q2**: What benefits does HEELS bring on the cloud?  
   Deploy with AWS NLB on the cloud
HEELS introduces minimal performance overhead.

**Diagram:**

- **X-axis:** Message Size (Kbytes)
- **Y-axis:** Goodput (Gbps)

**Graph Details:**

- **Label:** All cores enabled

**Legend:**

- **Direct Data Transfer:** Vanilla Katran and IPVS
- **Vanilla Katran:**
- **IPVS:**
- **HEELS:**

**Notes:**

- HEELS introduces minimal performance overhead.
We begin our evaluation by studying the throughput and latency with a server. In our evaluation, we perform both CRR and RR with instance type Top-of-Rack (ToR) switch. Virtual machines are running Ubuntu 22.04 (kernel 5.15.0).

Figure 2a demonstrates that HEELS achieves the same goodput with direct data transfer and IPVS throughout enabled on each machine. Note that IPVS also achieves similar performance with vanilla katran. We compare HEELS with two baselines: HEELS running with Katran and HEELS by deploying it with AWS' Network Load Balancer and Katran on the public cloud, highlighting its immediate usefulness of deployment in the real world. First, we benchmark the throughput over-head brought by HEELS with two baselines: HEELS with instance type t3.large from AWS. All the above servers and load balancer machine. These three machines are connected by a Quanta/Cumulus 48x10GbE switch. For the cloud experiments, we employ a mechanism widely used in the cloud, as it brings a 2% cost-saving benefit.
We begin our evaluation by studying the throughput and latency with instance type Top-of-Rack (ToR) switch. Virtual machines are running Ubuntu 22.04 (kernel 5.15.0). We observe the performance of HEELS when all CPU cores are enabled on the client and server and compare them with a decentralized design where the clients perform load balancing. We observe minimal performance overhead introduced by HEELS, as it brings a 2% overhead to direct data transmission.

Throughout the experiments, we introduce minimal performance overhead imposed by HEELS, as it brings a 2% overhead to direct data transmission. Note that IPVS also achieves similar performance with usage and their goodput is 6.248 Gbps and 6.043 Gbps, respectively. The goal of the experiment in Fig. 2b is to measure the overhead introduced by HEELS. Now we proceed to measure the latency overhead.

**Figure 2:** The goodput achieved by direct data transfer and IP Virtual Server (IPVS). Throughout the experiments, we achieve the same goodput with direct data transfer and IPVS throughout enabled on each machine. Fig. 2a demonstrates that machines, while Fig. 2b shows the results with only one CPU core.

**Figure 4:** TCP handshake latency breakdown.

In this section, we evaluate the performance of HEELS. We compare it with IPVS and direct data transfer, vanilla katran, and HEELS running with Katran.

**Figure 3:** Unloaded median latency measured for direct data transfer, vanilla katran, and HEELS.

**Figure 1:** Goodput (Gbps) for all servers and load balancer. All the above servers and load balancer machine runs Katran. We compare HEELS with two baselines: SK_STORAGE and Katran. We compare the performance and the ease of its deployment in the real world. First, we benchmark the throughput and latency overhead imposed by HEELS in the public cloud, highlighting its immediate usefulness of cost-saving benefits.
We begin our evaluation by studying the throughput and latency with instance type `t3.large` at 3.50GHz with 8 cores (16 hyper threads) and a 10G NIC. The virtual machines are running Ubuntu 22.04 (kernel 5.15.0).

Throughout the experiments, we observe the performance of HEELS, IPVS, and vanilla katran. We compare HEELS running with Katran. We limit the number of concurrent TCP connections to 100. Fig. 2a represents the results when all CPU cores are enabled on the client and server, while Fig. 2b shows the results with only one CPU core.

In the RR benchmark, clients establish a connection once and then request/response with an 8-byte payload, and closing this connection. The latency of establishing a connection, exchanging a single request/response with an 8-byte payload, and closing this connection.

In this section, we evaluate the cost-saving benefits of HEELS. Throughout the experiments, we observe the performance of HEELS's performance and the ease of its deployment.

HEELS introduces minimal performance overhead. Note that since HEELS only deploys eBPF programs on the client and server, there is no overhead brought by HEELS's performance and the ease of its deployment. HEELS's performance and the ease of its deployment.

To properly benchmark the performance, we use a customized implementation of Netperf's CRR (Connect-Request-Response) and RR benchmarks. In the RR benchmark, clients establish a connection once and then request/response with an 8-byte payload, and closing this connection.
HEELS introduces minimal performance overhead

![Graph 1](image1)

**Goodput (Gbps)**

- **HEELS**
- **Vanilla Katran**

![Graph 2](image2)

**Graph 2**

**Goodput (Gbps)**

- **HEELS**
- **Vanilla Katran**

3.2% overhead introduced by HEELS
HEELS improves the latency introduced by centralized LBs
HEELS improves the latency introduced by centralized LBs
HEELS improves the latency introduced by centralized LBs

![Graph showing HEELS saves >10% latency](image)

- HEELS saves >10% latency
- HEELS vs Vanilla AWS NLB
- Message size (Kbytes)
  - 4096
  - 1024
  - 8
- Time (ms)
  - 0
  - 5
  - 10

**Figure 5: Unloaded median latency measured for AWS Network Load Balancer (NLB) and Katran.**

**Table 2: Deploying costs for vanilla AWS NLB, HEELS, and vanilla Katran.**

<table>
<thead>
<tr>
<th>Packet Size (Kbytes)</th>
<th>Vanilla AWS NLB</th>
<th>HEELS</th>
<th>Vanilla Katran</th>
</tr>
</thead>
<tbody>
<tr>
<td>1024</td>
<td>0.027/hr</td>
<td>0.013/hr</td>
<td>0.010/hr</td>
</tr>
<tr>
<td>4096</td>
<td>0.055/hr</td>
<td>0.027/hr</td>
<td>0.020/hr</td>
</tr>
<tr>
<td>1024</td>
<td>0.027/hr</td>
<td>0.013/hr</td>
<td>0.010/hr</td>
</tr>
<tr>
<td>8</td>
<td>0.006/hr</td>
<td>0.003/hr</td>
<td>0.002/hr</td>
</tr>
</tbody>
</table>

**Conclusion:**

HEELS reduces latency compared to vanilla AWS NLB and Katran. The savings increase with larger packet sizes, making it a cost-effective solution for high-latency and bandwidth-intensive applications.
**HEELS** offers significant cost benefits for cloud users

### AWS NLB pricing

Cost for using AWS NLB
a flat rate of $0.027/hr

Cost for data traversing AWS NLB
a $0.006/hr rate for every GB processed.

<table>
<thead>
<tr>
<th>Message size (Kbytes)</th>
<th>Price per hour ($/hr)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>Vanilla AWS NLB</td>
</tr>
<tr>
<td></td>
<td>HEEL</td>
</tr>
<tr>
<td>1024</td>
<td></td>
</tr>
<tr>
<td>4096</td>
<td></td>
</tr>
</tbody>
</table>
HEELS offers significant cost benefits for cloud users

**AWS NLB pricing**

Cost for using AWS NLB
a flat rate of $0.027/hr

Cost for data traversing AWS NLB
a $0.006/hr rate for every GB processed.

<table>
<thead>
<tr>
<th>Message size (Kbytes)</th>
<th>Price per hour ($/hr)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Vanilla AWS NLB</td>
</tr>
<tr>
<td>8</td>
<td>0.028</td>
</tr>
<tr>
<td>1024</td>
<td>0.135</td>
</tr>
<tr>
<td>4096</td>
<td>0.459</td>
</tr>
</tbody>
</table>
**HEELS** offers significant cost benefits for cloud users

**AWS NLB pricing**

Cost for using AWS NLB
a flat rate of $0.027/hr

Cost for data traversing AWS NLB
a $0.006/hr rate for every GB processed.

<table>
<thead>
<tr>
<th>Message size (Kbytes)</th>
<th>Price per hour ($/hr)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Vanilla AWS NLB</td>
</tr>
<tr>
<td>8</td>
<td>0.028</td>
</tr>
<tr>
<td>1024</td>
<td>0.135</td>
</tr>
<tr>
<td>4096</td>
<td>0.459</td>
</tr>
</tbody>
</table>

constant costs
HEELS offers significant cost benefits for cloud users

**AWS NLB pricing**

Cost for using AWS NLB
- A flat rate of $0.027/hr

Cost for data traversing AWS NLB
- A $0.006/hr rate for every GB processed.

<table>
<thead>
<tr>
<th>Message size (Kbytes)</th>
<th>Price per hour ($/hr)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Vanilla AWS NLB</td>
</tr>
<tr>
<td>8</td>
<td>0.028</td>
</tr>
<tr>
<td>1024</td>
<td>0.135</td>
</tr>
<tr>
<td>4096</td>
<td>0.459</td>
</tr>
</tbody>
</table>

Increasing costs as the message size grows versus constant costs.
HEELS: A Host-Enabled eBPF-Based Load Balancing Scheme

A new eBPF-based load balancing scheme

Readily deployable on the cloud

Bringing both performance and cost benefits to users