**CHEETAH:*** A High-Speed Load-Balancer Design with Guaranteed Per-Connection-Consistency

**Tom Barbette**, Chen Tang, Haoran Yao, Gerald Q. Maguire Jr, Dejan Kostic, Panagiotis Papadimitratos & Marco Chiesa
**Datacenter Load-balancer**

**Uniform Load Balancing**

- **Per-Connection-Consistency (PCC)**
- **Efficient**
- **Dynamicity**

**This site can’t be reached**

The connection was reset.

Try:
- Checking the connection
- Checking the proxy and firewall
- Running Windows Network Diagnostics

**This webpage is not available**

ERR_CONNECTION_RESET

ERR_CONNECTION_TIMED_OUT
The challenge: Ensuring PCC

For each packet of an existing connection, the LB asks itself

« Which server is handling this connection? »

Per-Connection-Consistency
Today’s solutions cannot ensure all requirements at the same time
Stateless solutions
ECMP, WCMP [EuroSys’14]

Uniform

Efficient

PCC

Dynamic

[Image: Diagram showing Uniform, Efficient, PCC, and Dynamic with connections and symbols]
Stateless solutions

Consistent hashing [STOC’97], Beamer [NSDI’18], Faild [NSDI’18]

Uniform

Efficient

PCC

Dynamic
Stateful solutions

Silkroad [SIGCOMM’17], Ananta [SIGCOMM’13], Maglev [NSDI’16], Katran
CHEETAH

PCC « Which server is handling this connection? »

CHEETAH: ask the user to remember for us
key idea: store information about the load balancing decisions into a cookie

support any realizable load-balancing logic

high resilience

amenable to simple and fast implementation
CHEETAH IMPLEMENTATIONS

→ 5x faster software processing time while guaranteeing PCC

→ twice better tail latency compared to hashing thanks to more uniform load spreading

100G with 4 cores FastClick

12.8 Tb/s P4_Tofino

P4_16
CHEETAH
Cheetah Overview

Where to store the cookie?
- TCP timestamp
- QUIC connection ID
- IPv6 address

Load-Balancer
- set cookie
- get server id
- extract server id
- selected server 2

Users
- 2

Servers
- was server 2

Allows attackers to target a server...

2020-02-27
Stateless Cheetah

LoadBalancer

Cookies:
cookie = id XOR hash(4-tuple)

Get server id:

LoadBalancer:

Extract server id:
id = cookie XOR hash(4-tuple)

Set cookie:
s

Servers:

Users:
Servers extract server id
$id = \text{cookie XOR hash (4-tuple)}$

Load-Balancer

set cookie
$\text{cookie = id XOR hash (4-tuple)}$

get server id

Users

LB-logic

extract server id

previous server

Servers

selected server

selected server

Stateless CHEETAH
Stateless CHEETAH

The server can directly set up the cookie to enable Direct Server Return (DSR)

Load-Balancer

Users

LB-logic

set cookie

cookie = id XOR hash(4-tuple)

Servers

get server id

selected server

2020-02-27
There are two CHEETAHS

Stateless CHEETAH

Stateful CHEETAH

→ Keep per-connection state on the LB

∞ ◀️ ☭ ■

NAT
Statistics
Rate limiter
...

https://www.goodfreephotos.com
Stateful CHEETAH

Flow index

O(1) lookup

Per-connection state table

<table>
<thead>
<tr>
<th>Server ID</th>
<th>NAT</th>
<th>Statistics</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIP_3</td>
<td>1.2.3.4:5678</td>
<td>9 packets</td>
</tr>
<tr>
<td>DIP_6</td>
<td>9.0.1.2:3456</td>
<td>90 packets</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Stateful CHEETAH

https://www.goodfreephotos.com
Stateful CHEETAH

- Stack of Empty indexes
  - 1AB2
  - 39F0
  - ...

- Flow index
  - Fast in software
  - Entirely doable from hardware dataplane

- O(1) insertion
- O(1) deletion
- O(1) lookup

Per-connection state table

<table>
<thead>
<tr>
<th>Server ID</th>
<th>NAT</th>
<th>Statistics</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIP_3</td>
<td>1.2.3.4:5678</td>
<td>9 packets</td>
</tr>
<tr>
<td>DIP_6</td>
<td>9.0.1.2:3456</td>
<td>90 packets</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
EVALUATION

Stateful at the price of stateless
Packet processing performance analysis

Requests for a 8K file to 64 servers using HTTP, 640 000 req/s
Packet processing performance analysis

Requests for a 8K file to 64 servers using HTTP, 640 000 req/s
We have the advantages of stateful classification at the price of stateless:

- PCC
- Dynamicity
- Uniformity

Requests for a 8K file to 64 servers using HTTP, 640 000 req/s
Packet processing performance analysis

2 to 3 times faster classification than a cuckoo hash table

Requests for a 8K file to 64 servers using HTTP, 640 000 req/s
EVALUATION

Reducing load imbalances with arbitrary LB mechanisms
Uniform workload:
Variance across server load with hashing

20-30% variance with medium to high load

In line with Maglev [NSDI’16] which shows up to a ~30% overprovision

Requests for a 8K file using HTTP to 64 servers, at 100 to 200 req/s per servers
Uniform workload:
Variance across server load with Round-Robin (RR)

By using round-robin, CHEETAH can bring down the variance to reach a near-uniform load balancing.

Requests for a 8K file using HTTP to 64 servers, at 100 to 200 req/s per servers
Uniform workload:
Tail latency

CHEETAH lowers the tail latency by 2 to 3x

Requests for a 8K file using HTTP to 64 servers, at 100 to 200 req/s per servers
Bimodal workload:
Round-Robin (RR) does not help
Bimodal workload:
Least loaded

100req/s per servers with 10% of heavy request and 90% of small requests
Bimodal workload:
Average Weighted Round-Robin and Power-of-2-choice

Cheetah - AWRR  Cheetah - Least Loaded  Hash
Cheetah - Pow2  Cheetah - RR

CHEETAH allows to use your preferred advanced server selection techniques

a factor of ~2X

2020-02-27
100req/s per servers with 10% of heavy request and 90% of small requests
In the paper

Large-scale simulations
- 468 servers, motivation, # broken connections when ensuring uniformity with hashing

More experiments
- PCC w/ dynamicity, comparison with Beamer [NSDI’18]

Details about the cookie encoding and limitations
- Other possible cookie encodings

Multi-tier load-balancing considerations
- Stateful load balancers still break connections at scale!
Exploited a network cookie to:

> Guarantee PCC and support any implementable LB mechanism

> Present a fast design to allow $O(1)$ flow insertion and lookup from the dataplane

Cookie as a standard?

Implemented on both software switches and programmable Tofino ASIC

Thank you!

github.com/cheetahlb

includes dataset and experiments

Tom Barbette
Backup slides
TESTBED
Uniform workload experiment

Machine 1
Machine 2
Machine 3
Machine 4

LB

Machine 5
CPU 1
CPU 2
CPU 3
CPU 4
CPU 5
CPU 6
CPU 7
CPU 8
CPU 9
CPU 10
CPU 11
CPU 12
CPU 13
CPU 14
CPU 15
CPU 16

Machine 6
Machine 7
Machine 8

64 servers

8K HTTP requests

2020-02-27
FULL STATEFUL DESIGN
**CHEETAH stateful load balancer**

Client-side:
- IP src, VIP
- TCP src, dst
- payload

Server-side:

- **ConnTable[0]**
  - id | hash | DIP
  - ...
  - 39EF FEFE DIP_6
  - 39F0  -  -
  - 39F1 A7A7 DIP_4
  - ...

- **ConnTable[m-1]**
  - id | hash | DIP
  - ...
  - 39EF  -  -
  - 39F0 3434 DIP_1
  - 39F1 5656 DIP_3
  - ...

- **ConnStack[0]**
  - cookie
    - 39F0
    - 1D15
    - ...

- **ConnStack[m-1]**
  - cookie
    - 1A2B
    - 39EF
    - ...

C = \text{hash}_c \mod m
CHEETAH stateful load balancer

Client-side:
- IP src, VIP
- TCP src, dst
- payload

Server-side:
- LB-logic
- hash$_5$(c)
- connStack[0]
- connStack[m-1]

ConnTable[0]

<table>
<thead>
<tr>
<th>id</th>
<th>hash</th>
<th>DIP</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>39EF</td>
<td>FEFE</td>
<td>DIP_6</td>
</tr>
<tr>
<td>39F0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>39F1</td>
<td>A7A7</td>
<td>DIP_4</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

ConnTable[m-1]

<table>
<thead>
<tr>
<th>id</th>
<th>hash</th>
<th>DIP</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>39EF</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>39F0</td>
<td>3434</td>
<td>DIP_1</td>
</tr>
<tr>
<td>39F1</td>
<td>5656</td>
<td>DIP_3</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
CHEETAH stateful load balancer

ConnTable[0]

<table>
<thead>
<tr>
<th>id</th>
<th>hash</th>
<th>DIP</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>39EF</td>
<td>FFEF</td>
<td>DIP_6</td>
</tr>
<tr>
<td>39F0</td>
<td>11D7</td>
<td>DIP_3</td>
</tr>
<tr>
<td>39F1</td>
<td>A7A7</td>
<td>DIP_4</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

ConnTable[m-1]

<table>
<thead>
<tr>
<th>id</th>
<th>hash</th>
<th>DIP</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>39EF</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>39F0</td>
<td>3434</td>
<td>DIP_1</td>
</tr>
<tr>
<td>39F0</td>
<td>3434</td>
<td>DIP_1</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

ConnStack[0]

<table>
<thead>
<tr>
<th>cookie</th>
<th>id</th>
<th>hash</th>
<th>DIP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1D15</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>cookie</td>
<td>39F0</td>
<td>11D7</td>
<td>DIP_3</td>
</tr>
<tr>
<td>1A2B</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>39EF</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

ConnStack[m-1]

<table>
<thead>
<tr>
<th>cookie</th>
<th>id</th>
<th>hash</th>
<th>DIP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1A2B</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>39EF</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>cookie</td>
<td>39F1</td>
<td>5656</td>
<td>DIP_3</td>
</tr>
<tr>
<td>39F1</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

LB-logic

IP src, VIP
TCP src, dst
payload

hash₅(c)

hash₅(c) % m

client-side

server-side

IP src, DIP_3
TCP src, dst
cookie q
payload

payload

IP src, VIP
TCP src, dst
payload
TS PARSING
Implementation: storing the cookie

Where to store the cookie?

- **TCP timestamp**, QUIC connection-id, IPv6 addresses
- quickly extracting the TCP timestamp is key to high performance

<table>
<thead>
<tr>
<th>SYN packets</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>MSS SACKOK Timestamp [NOP WScale]</td>
<td>49.86%</td>
</tr>
<tr>
<td>MSS NOP WScale NOP NOP Timestamp [SACK EOL]</td>
<td>44.49%</td>
</tr>
<tr>
<td>MSS NOP WScale SACKOK Timestamp</td>
<td>4.53%</td>
</tr>
<tr>
<td>Slow path</td>
<td>1.12%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SYN-ACK packets</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>MSS SACKOK Timestamp [NOP WScale]</td>
<td>76.85%</td>
</tr>
<tr>
<td>MSS NOP WScale SACKOK Timestamp</td>
<td>18.79%</td>
</tr>
<tr>
<td>MSS NOP NOP Timestamp [SACK EOL]</td>
<td>1.69%</td>
</tr>
<tr>
<td>MSS NOP WScale NOP NOP Timestamp [SACK EOL]</td>
<td>1.55%</td>
</tr>
<tr>
<td>Slow path</td>
<td>1.12%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Other packets</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>NOP NOP Timestamp</td>
<td>98.46%</td>
</tr>
<tr>
<td>NOP NOP Timestamp [NOP NOP SACK]</td>
<td>1.49%</td>
</tr>
<tr>
<td>Slow path</td>
<td>0.05%</td>
</tr>
</tbody>
</table>

Based on a 1-hour KTH packet trace