Chameleon: Predictable Latency and High Utilization with Queue-Aware and Adaptive Source Routing

A. Van Bemten*, N. Đerić*, A. Varasteh*, S. Schmid^, C. Mas Machuca*, A. Blenk*, W. Kellerer*

Contact: nemanja.deric@tum.de

* Technical University of Munich
^ University of Vienna
Chameleon: **Predictable Latency** and High Utilization with Queue-Aware and Adaptive Source Routing

We want to provide **strict per-packet latency guarantees**
Chameleon: Predictable Latency and High Utilization with Queue-Aware and Adaptive Source Routing

Shortcomings of SotA → Unexploited optimization opportunities available in current networks

1. Sub-optimal Resource Usage
2. Static Flow Embedding

These drawbacks could lead to unnecessarily low utilization
Most SoA (Silo \textbf{SIGCOMM15}) do not exploit optimally “advanced” switch features such as priority queueing

All outgoing packets on one port will be served by the same queue → Resource & Demand Oblivious
Most SoA (Silo [SIGCOMM15]) do not exploit optimally “advanced” switch features such as priority queueing.

Per-Queue Topology: 1) More embedding opportunities & 2) Higher offered delay diversity
Chameleon is Queue-Aware → Higher Utilization
Second problem of the SoA (Silo [SIGCOMM15], QJump [NSDI15]) is that it is inflexible and static.

Once a decision is taken, it is **never reevaluated**

---

Flow/Network reconfigurations have the potential to greatly **increase**: the number of **accepted flows** → network **utilization** & operator **revenue**
Can we actually exploit **network reconfiguration**?

That requires **reconfiguration** of switches that are forwarding delay-sensitive traffic.

Use **Source Routing** → Reconfigure the hosts instead of the switches to avoid their unpredictability (that also circumvents the problem of consistent network updates).
Flow Embedding Strategy

When the centralized controller receives a flow request, the following algorithm is run:

```plaintext
1: function EMBEDDINGSTRATEGY(request)
2:     response ← ROUTE(request)
3:     if response ≠ NULL then
4:         RESERVE(response), return response
5:     for each flowToReroute in LIM(SORT(GETFLOWSTOREROUTE(request))) do
6:         INCREASEGRAPHCOSTS(flowToReroute, request)
7:         reroutingResponse ← ROUTE(flowToReroute)
8:         if reroutingResponse ≠ NULL then
9:             RESERVE(reroutingResponse)
10:            FREE(flowToReroute.originalPath)
11:            response ← ROUTE(request)
12:            if response ≠ NULL then
13:                RESERVE(response), return response
14:     return NULL
```

Firstly, we use per-queue level topology and our network calculus model to check if we can embed a flow. If a flow can be embedded, we simply reserve resources and configure the corresponding end host.

If a flow can’t be embedded, we run a rerouting algorithm.
Chameleon

End-Host Implementation

1. DPDK App (VMDq based) with tagging and shaping (leaky token bucket)
2. Vagrant software to deploy VMs
3. vhost-net/virtio-net architecture to interconnect DPDK app and VMs

(a) Server.  
(b) DPDK application.
1. Fat tree topology with $k = 4$ (or more in simulations)
2. Various flow types, listed in Table
   - Flows are added in an online fashion until the first embedding failure
3. Live Monitoring is done with EndaceDAG measurement card

<table>
<thead>
<tr>
<th>Flow description</th>
<th>Rate</th>
<th>Burst</th>
<th>Deadline</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Category 1: Industrial applications (IA) [1, 34]</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Database operations</td>
<td>[300, 550] Kbps</td>
<td>[100, 400] byte</td>
<td>[80, 120] ms</td>
</tr>
<tr>
<td>Production control</td>
<td>[100, 500] Kbps</td>
<td>[100, 400] byte</td>
<td>[10, 20] ms</td>
</tr>
<tr>
<td>Control and NTP</td>
<td>[1, 100] Kbps</td>
<td>[80, 120] byte</td>
<td>[10, 20] ms</td>
</tr>
<tr>
<td><strong>Category 2: Clock synchronization (CS) [51]</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Category 3: Control plane synchronization (CPS) [2, 55]</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Category 4: Bandwidth-hungry applications (BH) [4, 5, 45, 65]</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hadoop, data-mining</td>
<td>[100, 150] Mbps</td>
<td>[1000, 5000] byte</td>
<td>[10, 100] ms</td>
</tr>
<tr>
<td>Hadoop, data-mining</td>
<td>[100, 200] Mbps</td>
<td>[1000, 3000] byte</td>
<td>[10, 100] ms</td>
</tr>
<tr>
<td>Hadoop, data-mining</td>
<td>[80, 200] Mbps</td>
<td>[1000, 3000] byte</td>
<td>[50, 100] ms</td>
</tr>
</tbody>
</table>

Table 1: Considered flow types and their characteristics.
Evaluations – Results

Does Chameleon actually work? Yes

Does Chameleon bring any benefits? Yes, up to 15 times more flows compared to SoA.

Resource-aware and reconfigurable networks can improve cloud network utilization while providing predictable latency.
Thank you for your time!

For more details check the paper:

Chameleon: Predictable Latency and High Utilization with Queue-Aware and Adaptive Source Routing