Icarus: a Caching Simulator for Information Centric Networking

Lorenzo Saino, Ioannis Psaras and George Pavlou

Department of Electronic and Electrical Engineering
University College London

http://icarus-sim.github.io
Introducing Icarus

What is Icarus?

• Simulator for evaluating caching performance in ICN
• Not bound to any specific ICN architecture
• Design is generic enough to make it suitable to simulate any generic networked caching systems (KV stores, CDNs, content routers)

What Icarus is not?

• Not a suitable tool to evaluate other aspects of ICN architectures such as security, naming, congestion control, routing scalability
Requirements for caching simulators

General requirements:
• Reliability and accuracy
• Easy to use, fast iteration cycles
• Rich library of models, algorithms, protocols

Specific requirements:
• Large realistic topologies
• Large content catalogues and many content requests to allow caches to reach steady-state
• Support trace-driven simulations
Icarus objectives

Use cases:
- Caching and routing strategies
- Cache replacement policies
- Cache placement algorithms
- Analytical models

Non-functional requirements:
- Extensibility
- Scalability
High-level architecture
Extensibility

- Python-based, built based on fnss and networkx
- Plug-in registration system and extensive use of bridge pattern to provide loose-coupling

```python
@register_cache_policy('FOO')  # config
class FooCache(Cache)
    POLICIES = ['LRU', 'FOO']

    def get(self, k):
        ...

    def put(self, k):
        ...
```
Pluggable components

- Caching and routing strategies
- Cache replacement policies
- Topologies
- Workloads (synthetic and trace-driven)
- Cache placement strategies
- Content placement strategies
- Performance metrics
- Results readers/writers
Caching and routing strategies

Currently implemented strategies:

• Leave Copy Everywhere (LCE)
• Leave Copy Down (LCD)
• ProbCache
• Cache Less for More (centrality-based caching)
• Hash-routing
• Random (choice and Bernoulli)
• Nearest Replica Routing (NRR)
• No Cache
Cache replacement policies

Replacement policies:
- Least Recently Used (LRU)
- Segmented LRU (SLRU)
- Least Frequently Used (LFU)
- First In First Out (FIFO)
- Random

Add-ons:
- Probabilistic insertion
- TTL expiration
Logically centralized strategy implementation

- Strategies implemented as logically centralized entities
- Network implemented using Model-View-Controller (MVC)

**Common agent-based designs**

**Icarus design**
Scalability

- Flow-level abstraction
- No buffering
- Parallel execution of experiments
- Minimized I/O operations
Modelling tools

**Cache performance**
- Che’s approximation
- Laoutaris’ approximation
- Numerical hit ratio

**Workloads**
- Zipf fit
- Trace parsers
  - Wikibench
  - YouTube
  - Squid
  - URL list
  - GlobeTraff
Performance evaluation

Scalability

Extensibility

Accuracy
Simulators cross-comparison

Source:
ACM ICN’14, demo session

<table>
<thead>
<tr>
<th>Simulator</th>
<th>Flow/packet</th>
<th>Buffers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Icarus</td>
<td>Flow</td>
<td>No</td>
</tr>
<tr>
<td>ccnSim</td>
<td>Packet</td>
<td>No</td>
</tr>
<tr>
<td>ndnSim</td>
<td>Packet</td>
<td>Yes</td>
</tr>
</tbody>
</table>
CPU utilization

Source: Tortelli et al., ICN’14
## Memory utilization

<table>
<thead>
<tr>
<th>Simulator</th>
<th>SP+LCE</th>
<th>SP+RAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>ndnSim</td>
<td>9.82 GB</td>
<td>7.82 GB</td>
</tr>
<tr>
<td>ccnSim</td>
<td>53.68 MB</td>
<td>53.7 MB</td>
</tr>
<tr>
<td>Icarus</td>
<td>111.05 MB</td>
<td>110.98 MB</td>
</tr>
</tbody>
</table>

Source: Tortelli *et al.*, ICN’14
Parallel execution speedup

- Ideal speedup
- Actual speedup

Graph showing speedup vs. number of cores.

- Speedup increases linearly with the number of cores.
- The actual speedup remains close to the ideal speedup.
Accuracy

Source: Tortelli et al., ICN’14
## Extensibility

Implementing planned features

<table>
<thead>
<tr>
<th>Strategy</th>
<th>LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Edge</td>
<td>23</td>
</tr>
<tr>
<td>LCE</td>
<td>17</td>
</tr>
<tr>
<td>LCD</td>
<td>20</td>
</tr>
<tr>
<td>ProbCache</td>
<td>32</td>
</tr>
<tr>
<td>Centrality-based</td>
<td>30</td>
</tr>
<tr>
<td>NRR</td>
<td>24</td>
</tr>
</tbody>
</table>
# Extensibility

Implementing unplanned features

<table>
<thead>
<tr>
<th>Feature</th>
<th>LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>User-specified seed</td>
<td>3</td>
</tr>
<tr>
<td>User-defined experiment queue</td>
<td>7</td>
</tr>
<tr>
<td>Centrality-based cache placement</td>
<td>4</td>
</tr>
<tr>
<td>Results collector for debugging</td>
<td>20</td>
</tr>
<tr>
<td>Save results in CSV format</td>
<td>35</td>
</tr>
</tbody>
</table>
Summary and conclusions

• We presented Icarus, a caching simulator for ICN
• Designed for extensibility and scalability
• Independent cross-comparison validates soundness of design decisions
• Comprises a set of modelling tools for cache performance and workloads analysis
http://icarus-sim.github.io
Icarus tutorial

Lorenzo Saino

Department of Electronic and Electrical Engineering
University College London

http://icarus-sim.github.io
Agenda

• Architecture overview
• How to download, install, use
• Code walk-through
• Implement new components
• Configuring a simulation campaign
• Using modelling tools
• Q/A
Architecture and design

Code organized in four loosely-coupled subsystems:

- Orchestration
- Scenario generation
- Execution
- Results collection and analysis
Orchestration
Scenario generation
Scenario generation

- content placement
- cache placement
- topology factory
- data
- settings
- topology
- events
- parser
- workload
- Zipf Distr
- trace
- settings
Execution

- scenario
- orchestration
- results

- conf
- settings
- topology events
- topology events settings
- results
- execution
Execution

- Engine
- Strategy
- Network View
- Network Controller
- DataCollectorProxy
  - CacheHits Collector
  - Latency Collector
  - ... Collector
Results collection and analysis

scenario

orchestration

execution

results

conf

settings
topology
events
topology
events
settings
results
results
Results collection and analysis

```
results -> ResultSet -> plot
```

- reader
- writer
- file
Let’s look at the website

http://icarus-sim.github.io
Let’s look at the code

• Code overview
• Pluggable components
• Network API
Let’s run a sample simulation

LCE vs ProbCache

- Topology: RocketFuel (1221)
- Cache placement: uniform
- Content placement: uniform
- Workload: synthetic, $\alpha = 0.8$
- Replacement policy: LRU
- Metrics: cache hit ratio, latency
Modelling tools

Cache performance

Workloads
Modelling tools

Cache performance

• Che’s approximation

Workloads

```python
>>> import icarus as ics
>>> ics.che_cache_hit_ratio(
     ics.TruncatedZipfDist(alpha=0.8, n=1000).pdf, 100)
0.36482948293429832
```
Modelling tools

Cache performance

• Che’s approximation
• Laoutaris’ approximation

Workloads

>>> import icarus as ics
>>> ics.laoutaris_cache_hit_ratio(
    ics.TruncatedZipfDist(alpha=0.8, n=1000).pdf,
    100)
0.359348209359255
Modelling tools

Cache performance

• Che’s approximation
• Laoutaris’ approximation
• Optimal hit ratio

>>> import icarus as ics
>>> ics.optimal_cache_hit_ratio(
    ics.TruncatedZipfDist(alpha=0.8, n=1000).pdf,
    100)
0.52582651157679017
Modelling tools

Cache performance

- Che’s approximation
- Laoutaris’ approximation
- Optimal hit ratio
- Numeric hit ratio

```python
>>> import icarus as ics
>>> ics.numeric_cache_hit_ratio(
    ics.TruncatedZipfDist(alpha=0.8, n=1000).pdf,
    ics.LruCache(100))
0.37861264056574684
```
Modelling tools

Cache performance

- Che’s approximation
- Laoutaris’ approximation
- Optimal hit ratio
- Numerical hit ratio

>>> import icarus as ics
>>> ics.zipf_fit(ics.TruncatedZipfDist(alpha=0.8, n=1000).pdf)
(0.799999999571759, 1.0)

Workloads

- Zipf fit
Modelling tools

Cache performance
- Che’s approximation
- Laoutaris’ approximation
- Optimal hit ratio
- Numerical hit ratio

Workloads
- Zipf fit
- Trace parsers

```python
>>> import icarus as ics
>>> ics.parse_wikibench('wikibench.txt')
```
http://icarus-sim.github.io