Adaptive TTL Caches for Content Delivery

SANJAY SHAKKOTTAI

The University of Texas At Austin

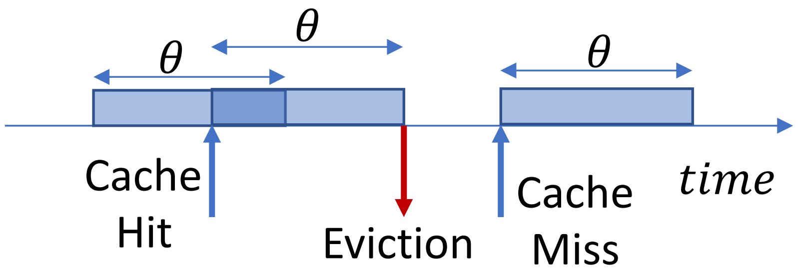
WITH SOUMYA BASU, ADITYA SUNDARARAJAN, JAVAD GHADERI, AND RAMESH SITARAMAN

Cache Design in Content Delivery

- Millions of objects of multiple types, each type has own requirement
- **25 m** objects of total size **25 TB** and **504 m** requests in a 9 day trace from one Akamai server (several thousand servers worldwide)
- Correlated arrivals with complex inter-arrival distribution
- Significant fraction of rare arrivals, e.g. objects with a few requests
- Guarantee **hit rate** for QoS and decrease **cache size** for reducing cost

Time-to-Live (TTL) Cache

- Popular caching scheme with good theoretical guarantees
- **Algorithm:** Fixed TTL value $\boldsymbol{\theta}$ for all objects
  - Cache miss: Object not in cache
    - Fetch object from server
    - Cache object with TTL θ
  - Cache hit: Object present in cache
    - Reset TTL of the object to θ
  - On timer expiry: Evict object from cache



TTL Cache Dynamics

Deficiencies in Existing Approach

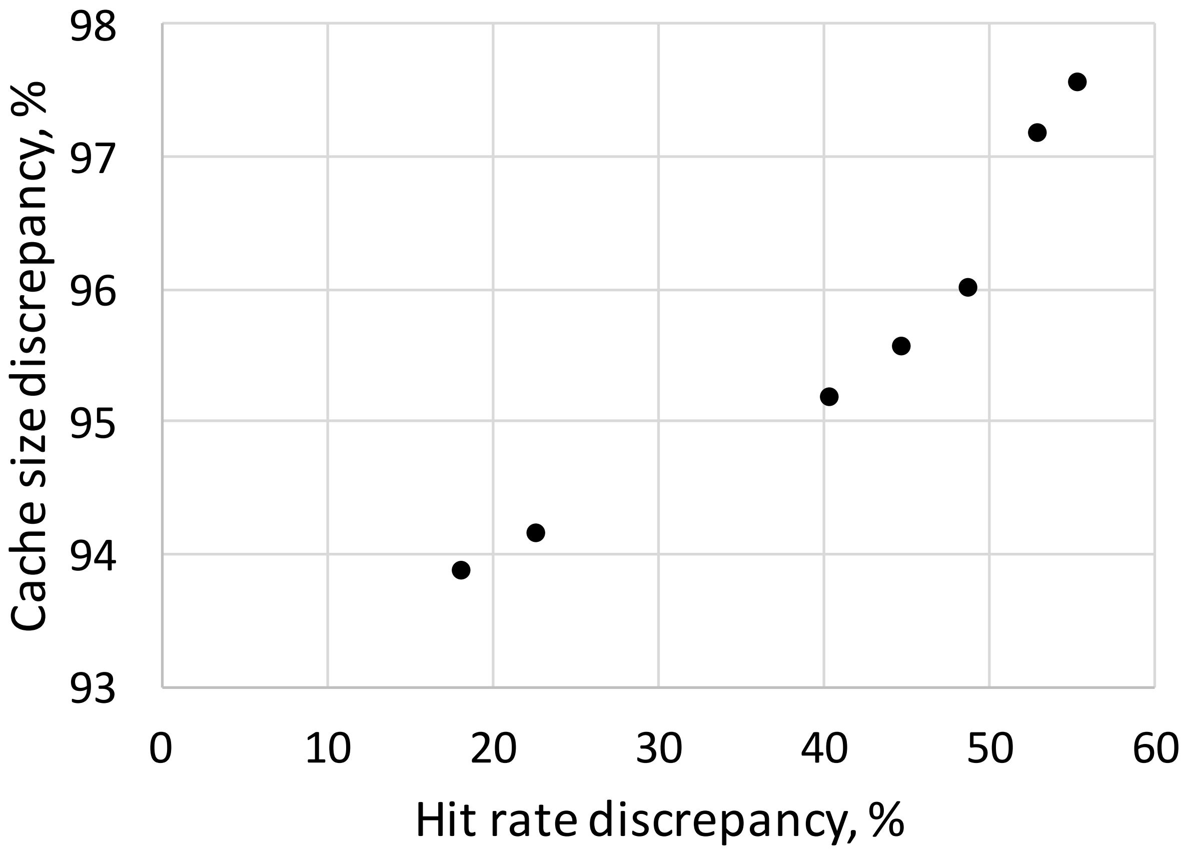
- A popular approach in designing cache is model based
  - Assume an underlying content request model
  - Tractable approximations, e.g. Che's formula/approximation
  - Design cache using analytical expression of hit rate and size

$$\boldsymbol{Hit\ rate} = \mathbf{1} - \mathbf{exp}(-\boldsymbol{\lambda_m T_c})$$
$$\boldsymbol{Cache\ size} = \sum_{m} (\mathbf{1} - \boldsymbol{exp}(-\boldsymbol{\lambda_m T_c}))$$

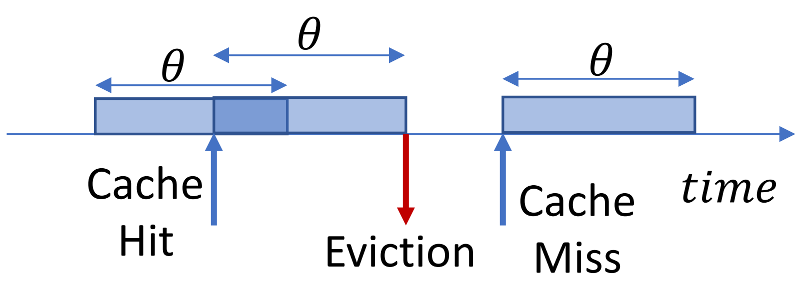
- Approximate request model lead to error in analytical expression
- Tractable approximations difficult for complex caching models

Deficiencies in Existing Approach: Fixed TTL

- Compute TTL value for hit rate and estimate size for the TTL value
- Performance of TTL cache with approx. on a 9 day long Akamai trace



Dynamic TTL Adaptation



TTL Cache Dynamics

- **Goal:** Achieve target hit rate h^*
- **Adaptation of TTL:** TTL $\uparrow \Rightarrow$ Hit rate $\uparrow$, TTL $\downarrow \Rightarrow$ Hit rate $\downarrow$
  - Cache hit on l^{th} arrival
    - Decrease TTL : $\theta(l) = \left(\theta(l-1) - \frac{1}{l}(1-h^*)\right)^+$
  - Cache miss on l^{th} arrival
    - Increase TTL : $\theta(l) = \theta(l-1) + \frac{1}{l}(h^*)$

Converges to TTL $\boldsymbol{\theta^*}$ providing hit rate $\boldsymbol{h^*}$ for stationary traffic

Drawback: Transient Arrivals

- Rare objects have negligible contribution to hit rate
- **70%** of all objects, **10%** of the traffic, are requested only once
- Size occupied by rare objects = (TTL value) $\times$ (Arrival rate)
- Significant number of rare objects leads to large cache wastage

Which objects are rare?
How to filter rare objects?

Filtering Rare objects: Shadow Cache

- Separate shadow cache along with main TTL cache – Deep cache
- On a new arrival cache object **label** in shadow cache with TTL θ
- Upon a hit in shadow cache object enters deep cache with TTL θ
- Similar to LRU-2Q or Bloom filter + LRU

Result: Dynamic TTL with $\theta(l)$ projected on [0,L] converges to TTL θ^* in the presence of rare object traffic if h^* is feasible (w.r.t. L).

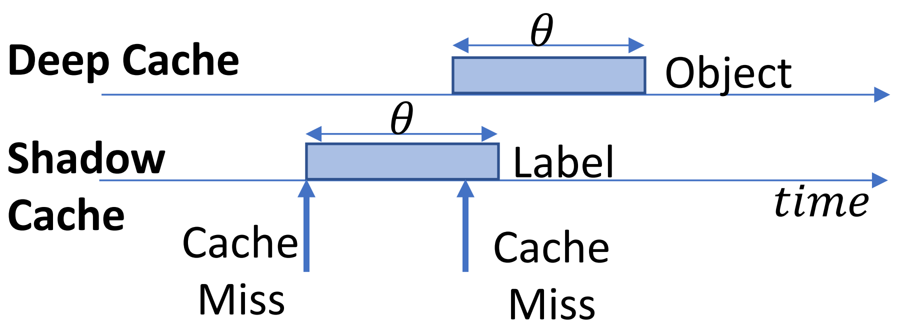
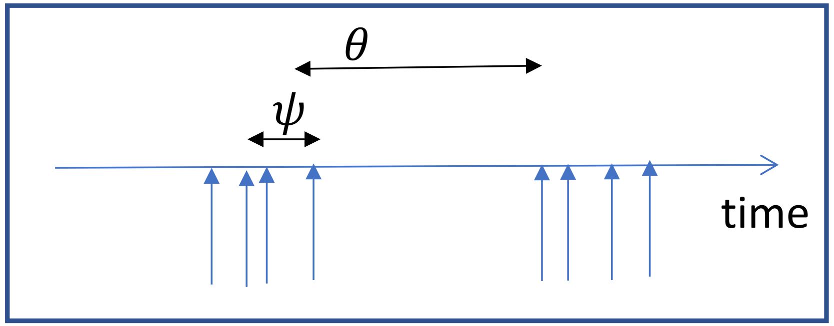


Fig. TTL Cache with Filtering

Infrequent Bursty Arrivals

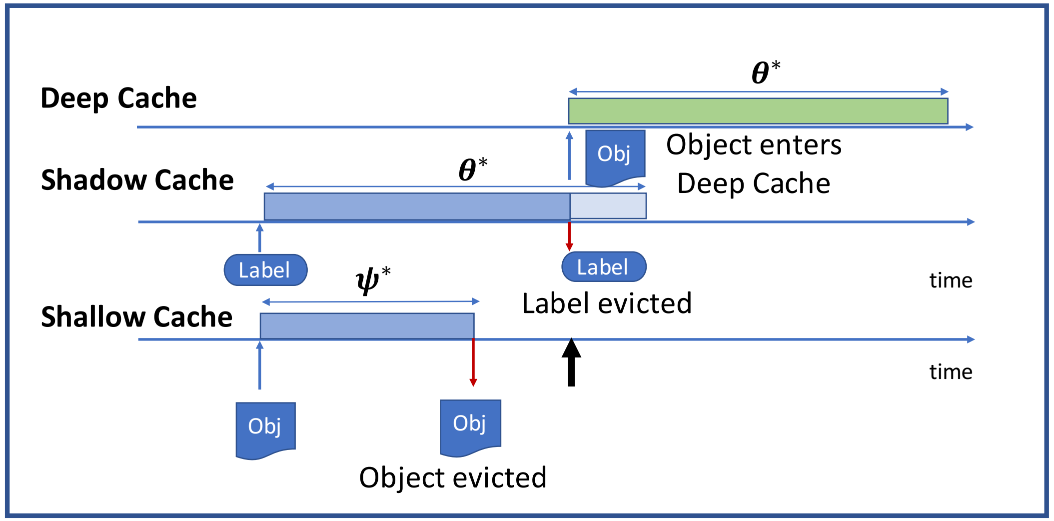
- Filtering differentiates rare and popular objects to reduce cache size
- Problem: Infrequent bursty arrivals suffer from filtering

Example: Bursty requests with typically 4 arrivals, and bursts separated in time beyond TTL.



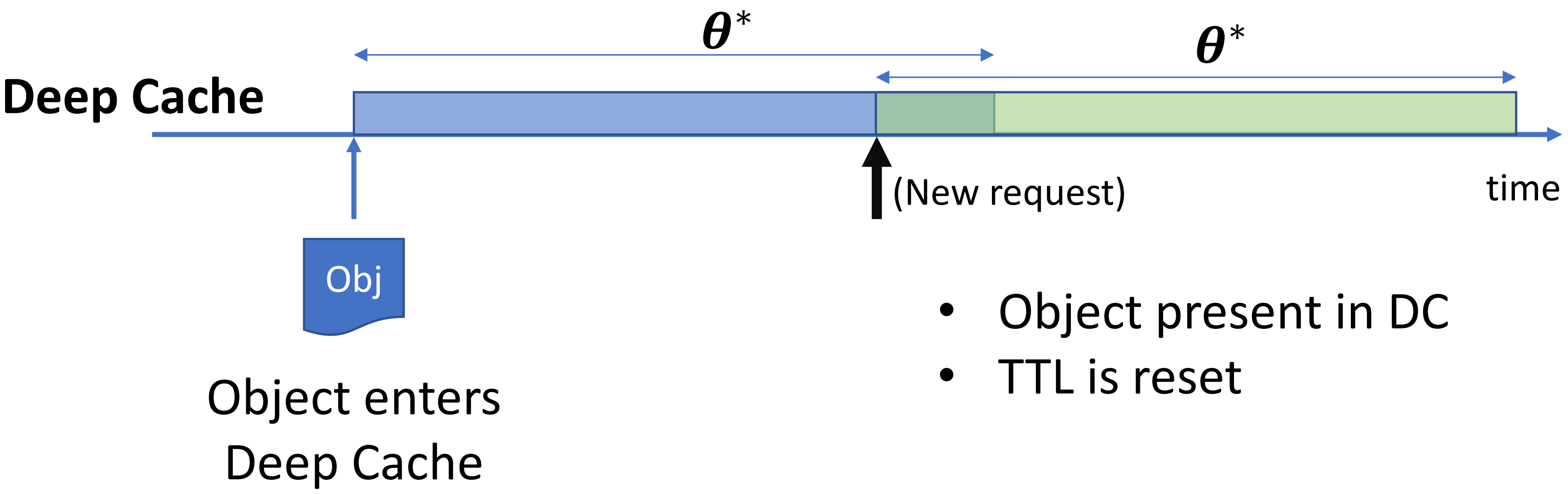
Can we capture multi-time-scale dynamics?

Filtering TTL Cache



- A filtering TTL cache has three parts:
  - Deep Cache (DC), Shadow Cache (SdC), and **Shallow Cache (SC)**
- Deep Cache captures the popular objects with TTL θ
- Shadow Cache filters out rare objects with TTL θ
- **Shallow Cache captures infrequent burstiness with TTL $\psi < \theta$**
- **Joint adaptation of ψ and θ to meet hit rate and size targets**

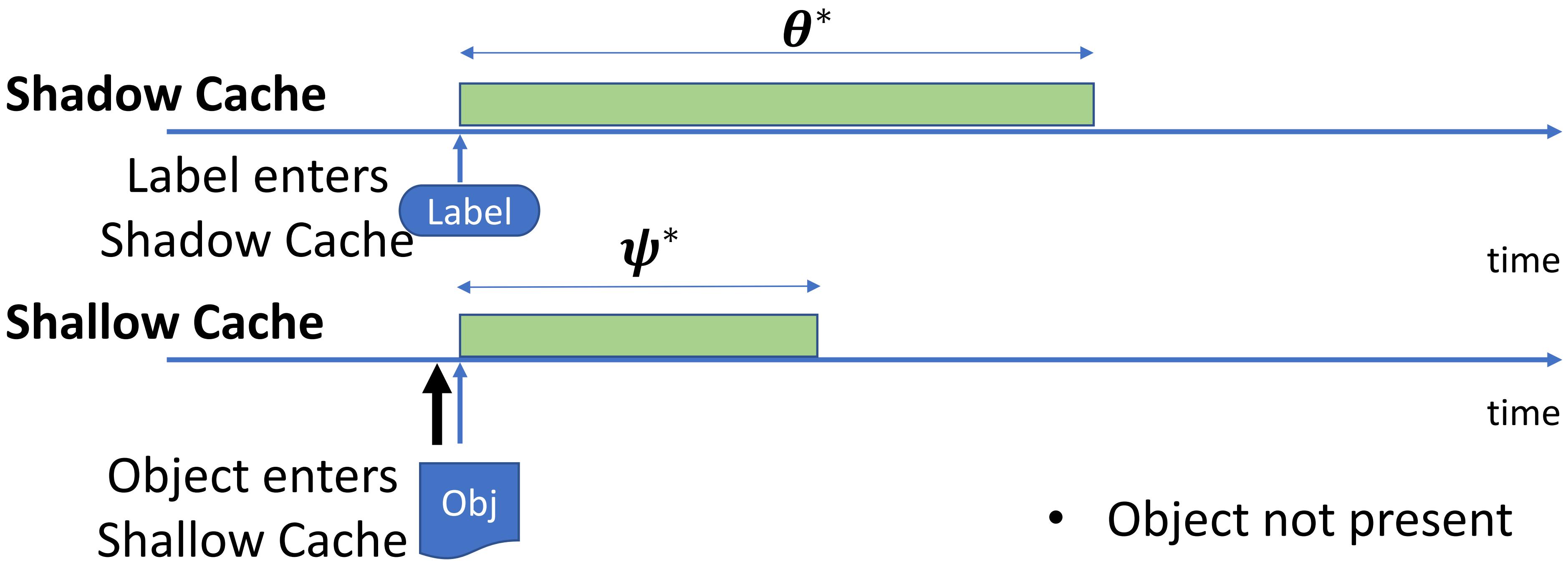
Filtering TTL Cache (f-TTL): Deep Cache Hit



- Object present in DC
- TTL is reset

Cache Hit in Deep Cache

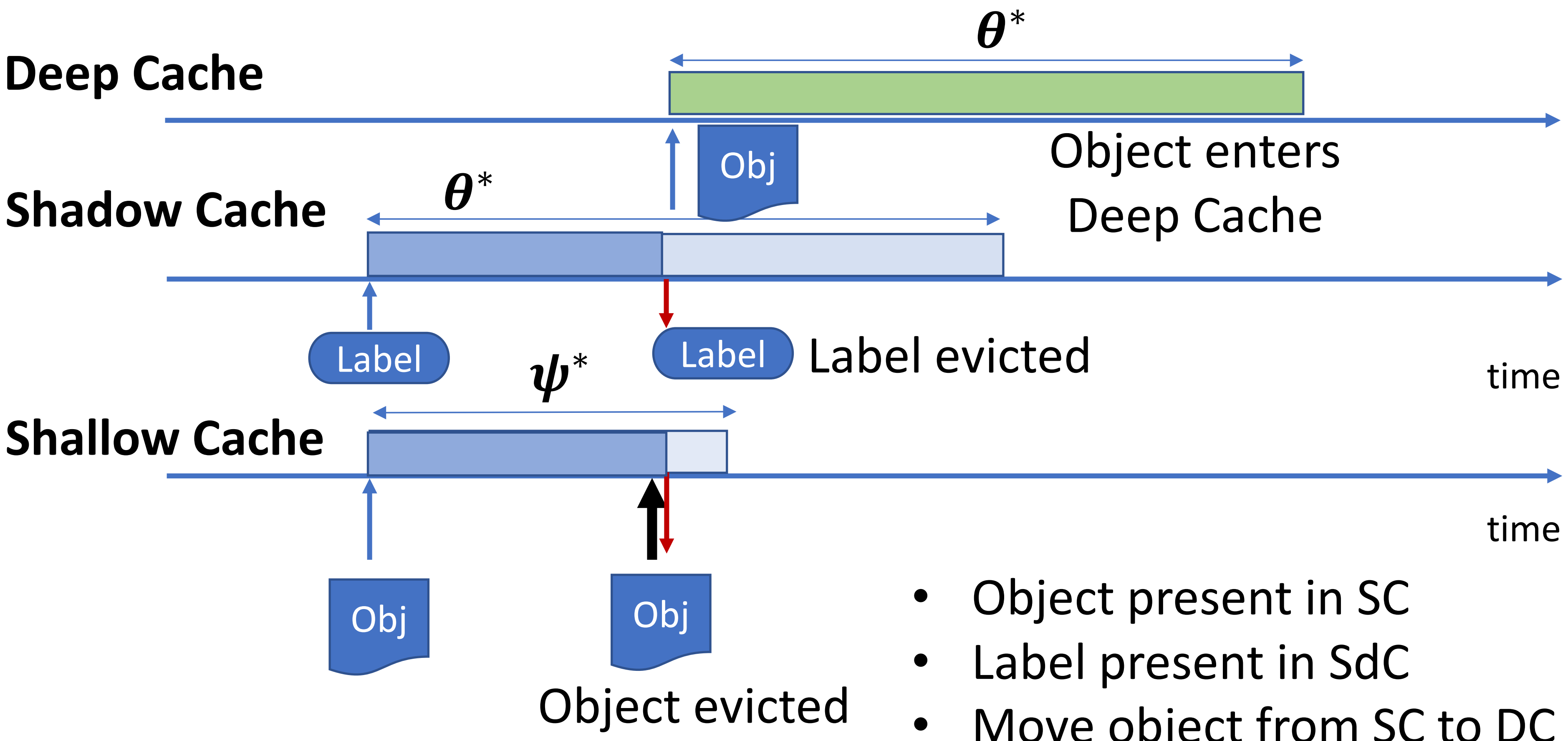
Filtering TTL Cache (f-TTL): Cache Miss



- Object not present
- Label not present
- Fetch from server

Cache Miss

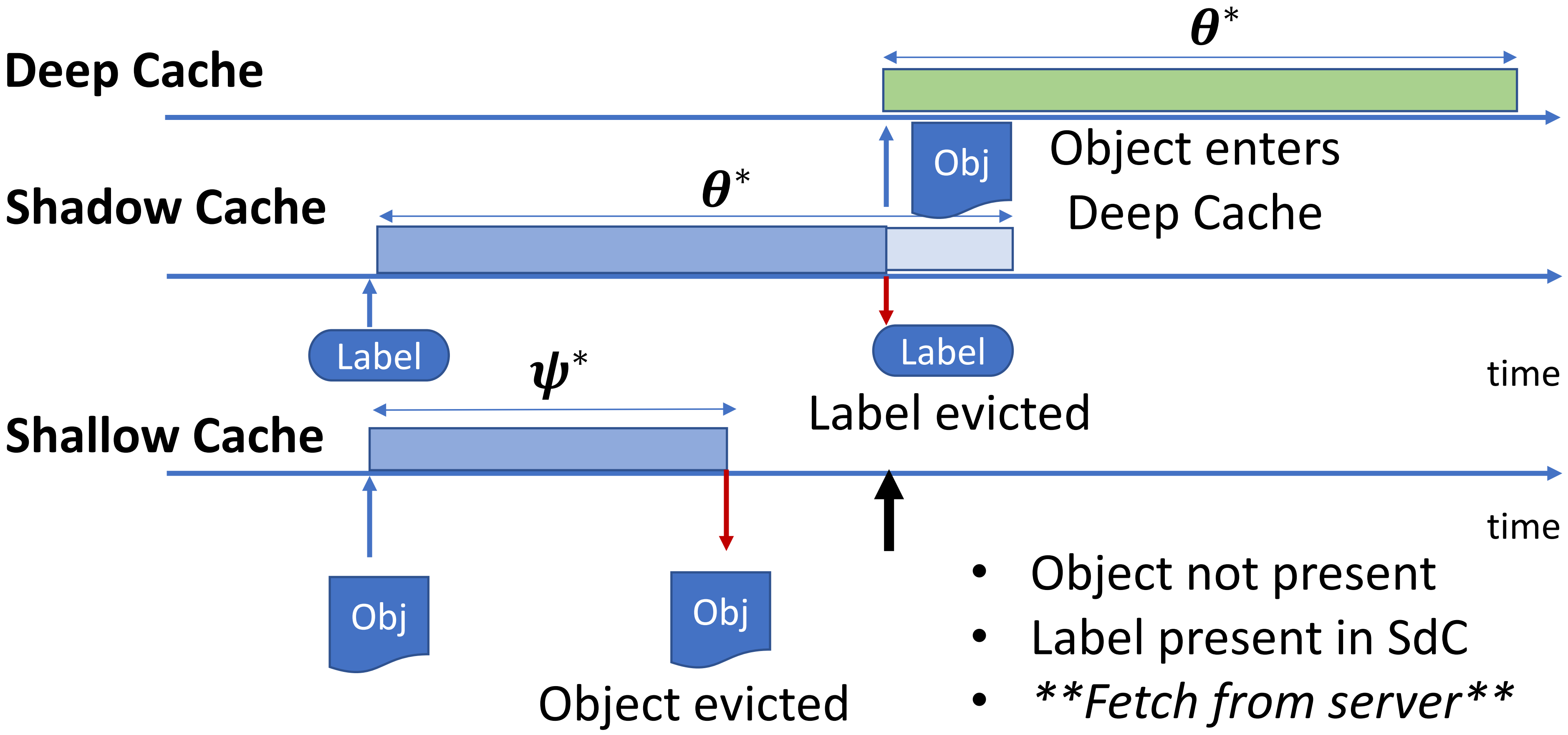
Filtering TTL Cache (f-TTL): Shallow Cache Hit



- Object present in SC
- Label present in SdC
- Move object from SC to DC

Shallow Cache Hit

Filtering TTL Cache (f-TTL): Shadow Cache Hit

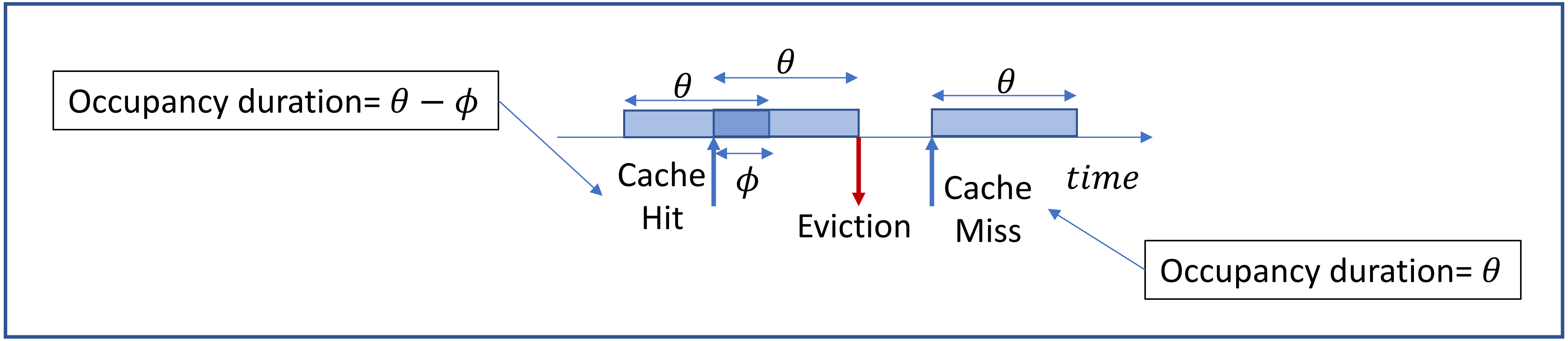


- Object not present
- Label present in SdC
- ****Fetch from server*****
- Not counted as cache hit

Shadow Cache Hit

Filtering TTL Cache (f-TTL):Objectives

- Achieve target hit rate $\boldsymbol{h}^*$
- Achieve size $\lambda \boldsymbol{s}^*$ (λ is arrival rate)
- $\boldsymbol{s}^*$ is average (over time and objects) occupancy duration of a request
- Occupancy duration is duration from arrival to eviction or TTL reset



Filtering TTL Cache (f-TTL): Adaptation

- The adaptation of θ is the same as the dynamic TTL cache
  - Increase θ on cache miss and shadow cache hit
  - Decrease θ on shallow or deep cache hit
- The adaptation of ψ is to meet s^* (size target == occupancy duration)
  - Estimate the occupancy duration $s_{est}(l)$
  - Increase ψ if $s^* > s_{est}(l)$ and decrease otherwise

F-TTL Cache: Time Scale Separation

- Faster adaptation of θ compared to ψ – Time scale separation
  - Deep Cache Adaptation: In faster time scale, ψ (shallow cache) is quasi-static while θ adjusts to attain hit rate
  - Shallow Cache Adaptation: In slower time scale, ψ adapts to attain size

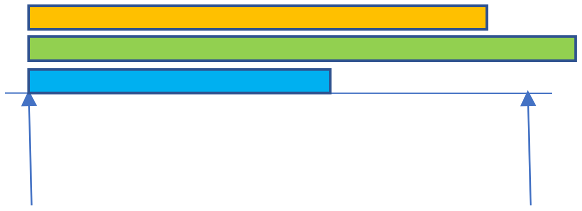
- $\theta(l) = \left(\theta(l-1) - \frac{1}{l^{\alpha}}(Y(l) - h^*)\right)^+$

  $\alpha \in (0.5,1)$ and $Y(l)$ = 1 if Deep cache/ Shallow cache hit, 0 o/w

- $\psi(l) = \min\{\theta(l), \left(\psi(l-1) + \frac{1}{l}(s^* - s_{est}(l))\right)^+\}$

Filtering TTL Cache (f-TTL): Estimating s_{est}

- At adaptation instance, checking the size is misleading
- Remaining TTL timer value for request object $\phi(l)$
- Estimating the occupancy duration
  - Deep/ Shallow cache hit: $s_{est}(l) = \theta(l-1) - \phi(l)$
  - Shadow cache hit: $s_{est}(l) = \theta(l-1)$
  - Cache miss: $s_{est}(l) = \psi(l-1)$



- Size 3 at the first arrival instance
- Size changes between two arrival instances

Truncating Parameters – Towards Actor-Critic

- TTL value θ may become unbounded in presence of rare objects
  - α% of the traffic from rare objects $\Longrightarrow h^* > (100 - \alpha)\%$ infeasible
  - We need to project $\theta(l)$ on [0,L] with large but finite L

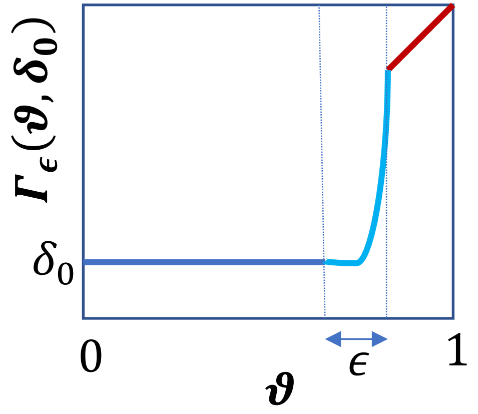
Projection for robustness against rare objects

- Problem: Suppose the operator sets attainable h^* but sets s^* too small
- We can argue that $(\theta(l), \psi(l)) \rightarrow (L, 0)$, resulting in an achieved hit rate of $h < h^*$

Approach: Fictitious dynamics through an actor-critic algorithm

An Actor-Critic Algorithm

- **Approach:** Separate observation and action – **Actor-critic algorithm**
- Critic parameters ϑ and δ record hit rate and size rate, resp.
- Actor parameters θ and ψ are used in the f-TTL algorithm
- Actors are functions of critics: $\theta = L\vartheta$ and $\psi = \mathrm{L}\,\Gamma_\epsilon(\vartheta, \delta)$
  - Saturation function $\Gamma_\epsilon(\vartheta, \delta) = \begin{cases} \delta, & if\ \vartheta \le 1 - 1.5\epsilon \\ \vartheta, & if\ \vartheta, \ge 1 - 0.5\epsilon \\ interpolation, & o/w \end{cases}$



- Time scale separation in ϑ and δ adaptation to ensure convergence

Details: Actor Critic Adaptation

- **Cache hit:** Let the object be of type t , with size w and at the time of request its TTL timer be $\phi > 0$
  - $\vartheta(l) = \max\left(0, \vartheta(l-1) - \frac{1}{l^\alpha}(1-h^*)\right), \alpha \in (0.5,1)$
  - $\delta(l) = \min\left(1, \max\left(0, \vartheta_t(l-1) + \frac{1}{l}(s^* - \theta(l-1) + \phi)\right)\right)$
- **Shadow hit or miss:** Let the object be of type t , with size w
  - $\vartheta(l) = \min\left(1, \vartheta(l-1) + \frac{1}{l^\alpha} h^*\right)$
  - $\delta(l) = \min\left(1, \max\left(0, \vartheta(l-1) + \frac{1}{l}\left(s^* - \psi(l-1)\right)\right)\right)$

Filtering TTL Cache (f-TTL): Guarantees

- Bursty arrivals, and rare objects following a 'Rarity condition' (objects with asymptotic zero hit rate) present
- Two-timescale stochastic approximation based proof technique (using methods in Borkar 97; conditions from Kushner-Clark, Kushner-Yin)

Filtering TTL with actor-critic adaptation converges to $(\boldsymbol{\vartheta}^*, \boldsymbol{\delta}^*)$ a.a.s.

- **If $\boldsymbol{h}^*$ feasible with threshold $\boldsymbol{L}$ then $\boldsymbol{h}^*$ is achieved**
- **Either achieves size $\boldsymbol{\lambda s}$ where $\boldsymbol{s} \leq \boldsymbol{s}^*$**
- **Or collapses to a pure shadow cache mode, i.e. $\boldsymbol{\psi} = \mathbf{0}$**

- Generalize to multiple types with different h^* and s^*
- Generalize to different object sizes with modified adaptation

Performance on Akamai Traces

- Modified Algorithm with constant step size adaptation
- 9 day trace with 504m requests from 25m distinct objects
- Average error for hit rate targets 0.4, 0.5, 0.6, 0.7, and 0.8 : < 1.3%

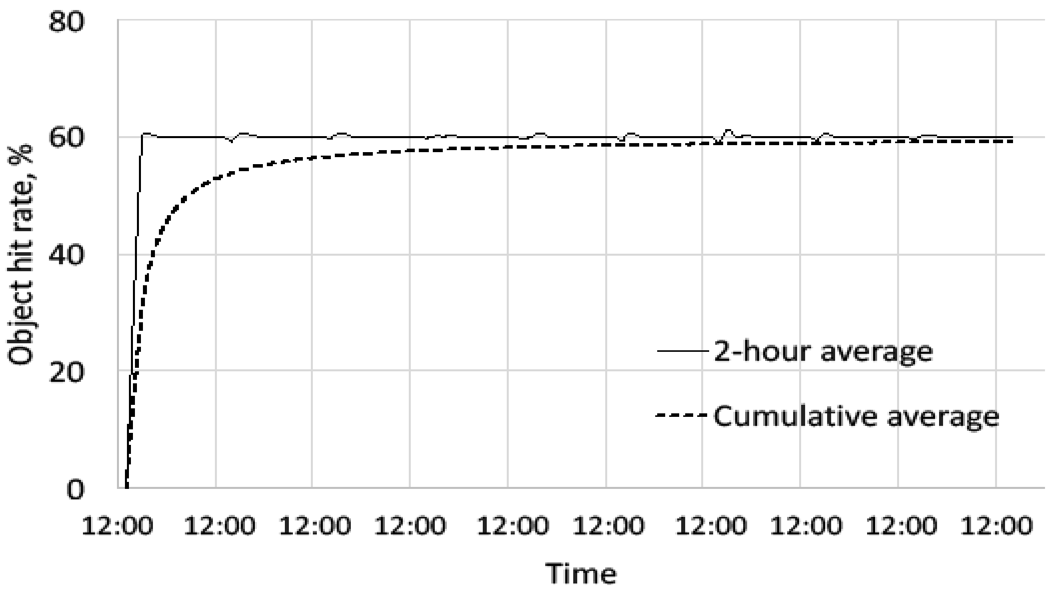


Fig 1: d-TTL Convergence Plot

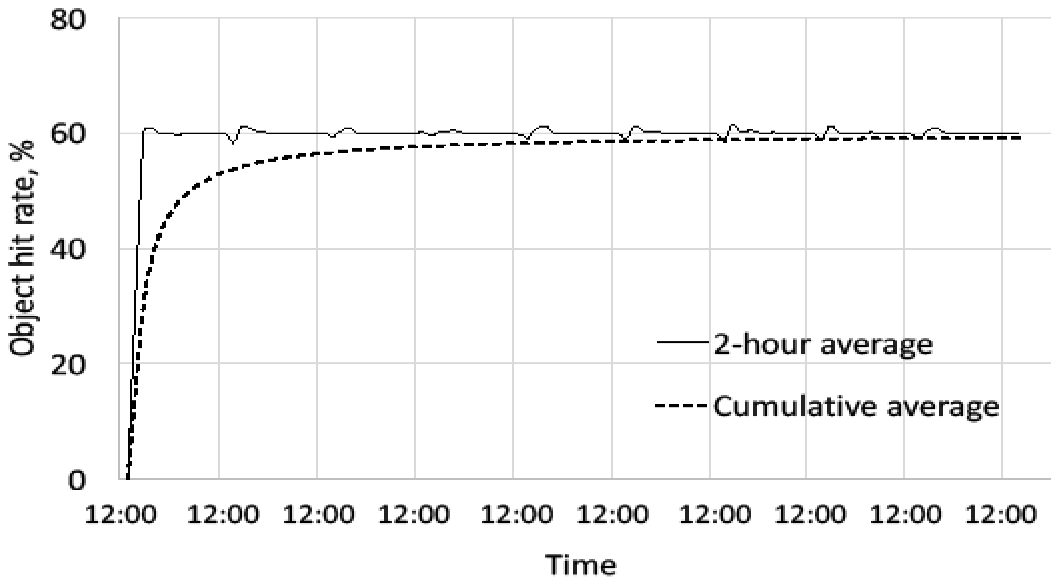


Fig 2: f-TTL Convergence Plot

Performance on Akamai Traces

- Variable sized version of the d-TTL and f-TTL Algorithms
- Size rate target = 50% of d-TTL : Size rate achieved = 49% of d-TTL

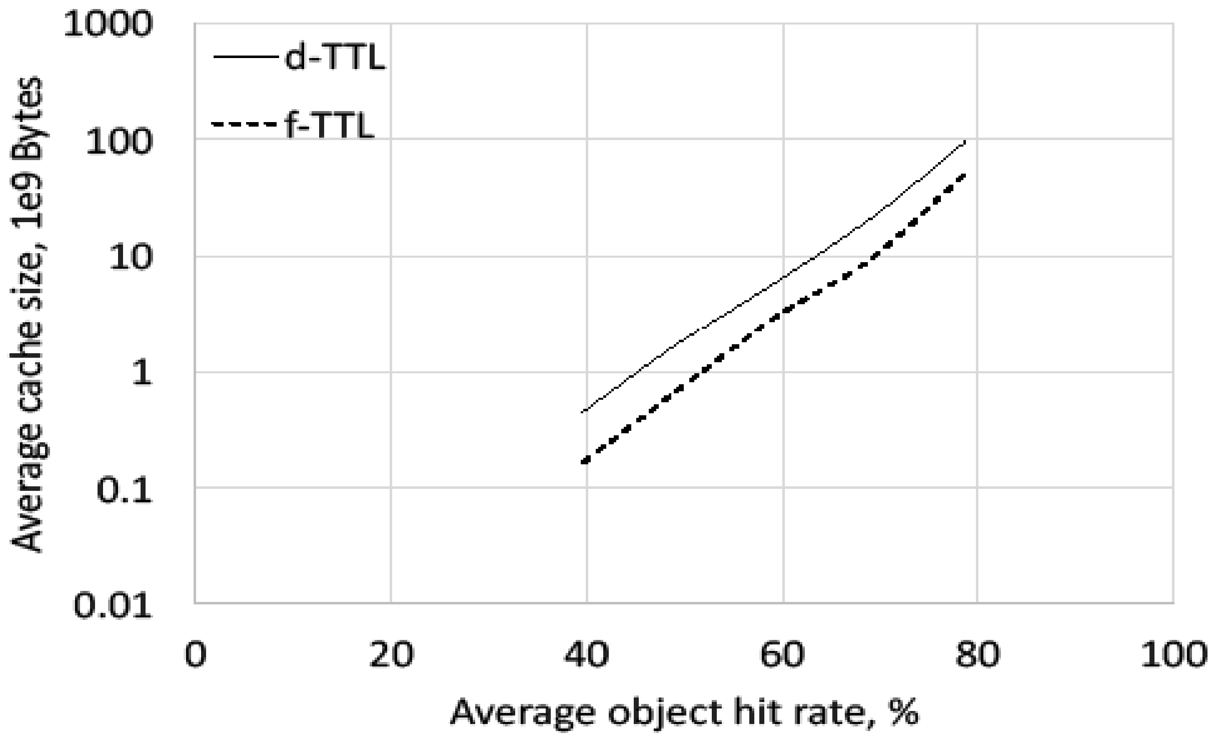


Fig 3: Hit rate vs Average Cache Size curve

Thank You