Caching or pre-fetching? The role of hazard rates.

Andres Ferragut

joint work with Matias Carrasco and Fernando Paganini

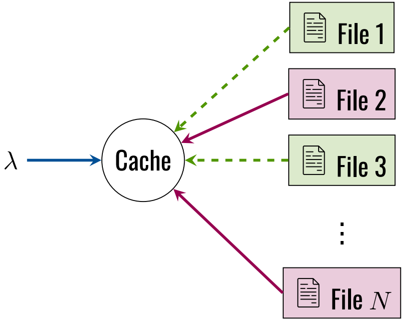
Universidad ORT Uruguay

60th Allerton Conference on Computing, Control and Communications – September 2024

Outline

The caching problem

- Consider a local memory system that handles items from a catalog of N objects.
- Requests for objects arrive as a random process.
- The memory (cache) can locally store $C < N$ of them.
- If item is in cache, we have a hit. Otherwise, it is a miss.

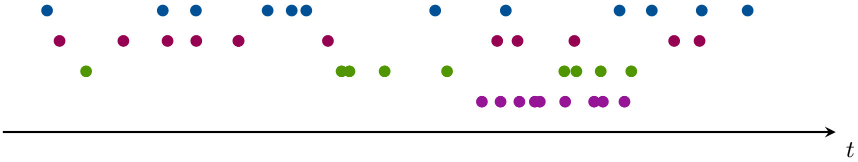


Objective: for a given arrival stream, maximize the steady-state hit rate.

Point process approach

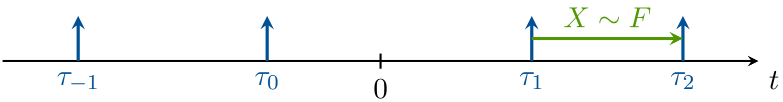
Introduced in [Fofack et al. 2014]

- Assume requests for item i come from a point process of intensity λ_i (popularities).



- At each point in time we must decide which items must be stored locally.

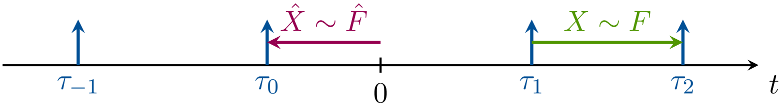
Two important distributions:



- **Inter-arrival distribution:** Typical distance between points...

$$X \sim F(t), \quad E[X] = 1/\lambda$$

Two important distributions:



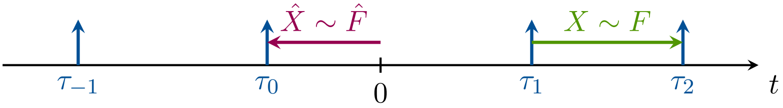
- **Inter-arrival distribution:** Typical distance between points...

$$X \sim F(t), \quad E[X] = 1/\lambda$$

- **Age distribution:** Distance from the last point in the current interval (sampling bias)!

$$\hat{X} \sim \hat{F}(t) := \lambda \int_0^t 1 - F(s)ds,$$

Two important distributions:



- **Inter-arrival distribution:** Typical distance between points...

$$X \sim F(t), \quad E[X] = 1/\lambda$$

- **Age distribution:** Distance from the last point in the current interval (sampling bias)!

$$\hat{X} \sim \hat{F}(t) := \lambda \int_0^t 1 - F(s)ds,$$

Note: you can formalize this under the Palm probability framework for stationary point processes.

Populating a cache: timer based (TTL) policies

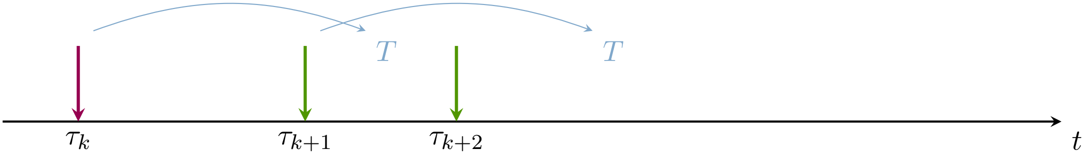
- Upon request arrival for item i, check for presence.

Populating a cache: timer based (TTL) policies

- Upon request arrival for item i, check for presence.
- If new, store item and start a timer T_i to evict.

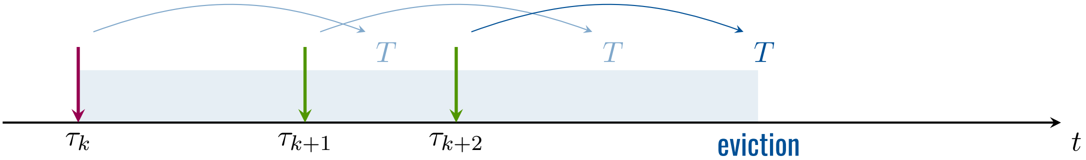
Populating a cache: timer based (TTL) policies

- Upon request arrival for item i, check for presence.
- If new, store item and start a timer T_i to evict.
- If present, reset timer to T_i.



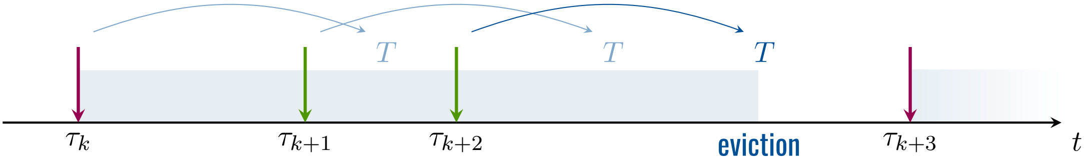
Populating a cache: timer based (TTL) policies

- Upon request arrival for item i, check for presence.
- If new, store item and start a timer T_i to evict.
- If present, reset timer to T_i.
- Upon timer expiration, evict the content.



Populating a cache: timer based (TTL) policies

- Upon request arrival for item i, check for presence.
- If new, store item and start a timer T_i to evict.
- If present, reset timer to T_i.
- Upon timer expiration, evict the content.
- Keep timers T_i such that average cache occupation is C.



Hit and occupation probabilities

Focus on a single item i:

- Hit probability: prob. that next request is still stored before timer expires:

$$P(X_i < T_i) = F_i(T)$$

- Memory usage: prob. that item is stored at a random point in time, i.e. prob. that timer has not expired since the last request:

$$P(\hat{X}_i < T_i) = \hat{F}_i(T)$$

Choosing the optimal timers

Requests come from independent sources with intensities λ_i and inter-arrival distribution F_i:

Problem (Optimal TTL policy)

Choose timers $T_i \geqslant 0$ such that:

$$\max_{T_i \geqslant 0} \sum_i \lambda_i F_i(T_i)$$

subject to:

$$\sum_i \hat{F}_i(T_i) \leqslant C$$

Remark: non-convex non-linear program. But it can be solved by a change of variables!!! [Ferragut et al. – QUESTA – 2018].

Structure of the optimal caching policy

- The crucial magnitude is the hazard rate of F:

$$\eta(t) := \frac{f(t)}{1 - F(t)}$$

- Likelihood of a request at time t, given the current interval has age t.

Structure of the optimal caching policy

- The crucial magnitude is the hazard rate of F:

$$\eta(t) := \frac{f(t)}{1 - F(t)}$$

- Likelihood of a request at time t, given the current interval has age t.

Theorem (F', Rodriguez, Paganini – 2016)

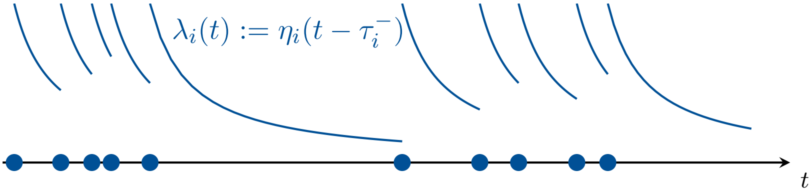
If the F_i have decreasing hazard rates, then the optimal TTL policy satisfies:

$$\eta_i(T_i^*) \geqslant \theta^*,$$

whenever $T_i^* > 0$ (i.e. the item is cached).
Moreover, inequality is strict iff $T_i^* = \infty$ (item always stored).

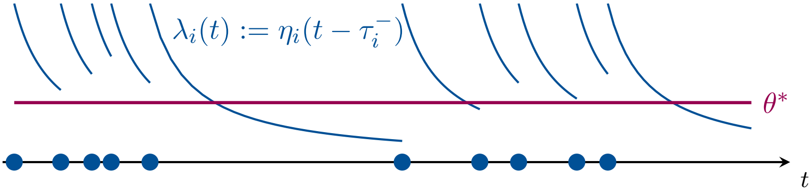
Why caching helps in this case?

Decreasing hazard rates corresponds to bursty traffic:



Why caching helps in this case?

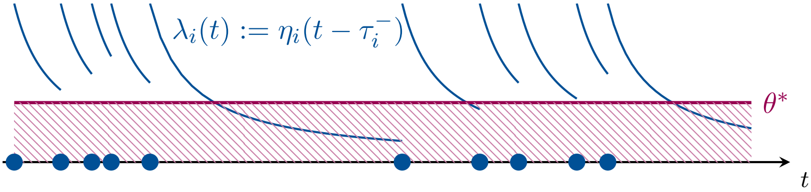
Decreasing hazard rates corresponds to bursty traffic:



- An arrival makes a subsequent arrival more likely.
- Store it while its likelihood is high enough (above a threshold).

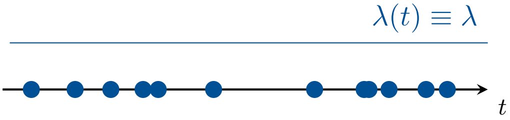
Why caching helps in this case?

Decreasing hazard rates corresponds to bursty traffic:

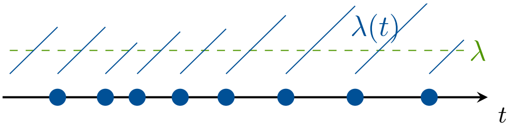


- An arrival makes a subsequent arrival more likely.
- Store it while its likelihood is high enough (above a threshold).

What about other types of traffic?

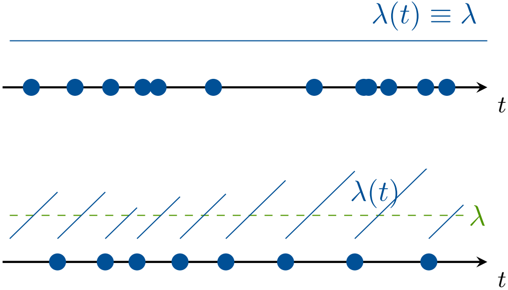


Constant hazard rate $\rightarrow$ Poisson process.



Increasing hazard rate $\rightarrow$ more periodic!

What about other types of traffic?

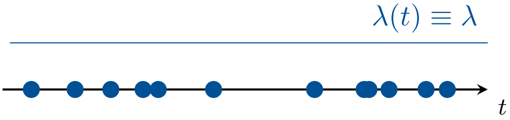


Constant hazard rate $\rightarrow$ Poisson process.

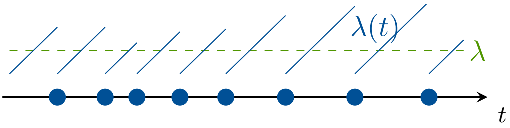
Increasing hazard rate $\rightarrow$ more periodic!

Theorem: for these types of traffic, keep the most popular is the optimal caching policy.

What about other types of traffic?



Constant hazard rate $\rightarrow$ Poisson process.

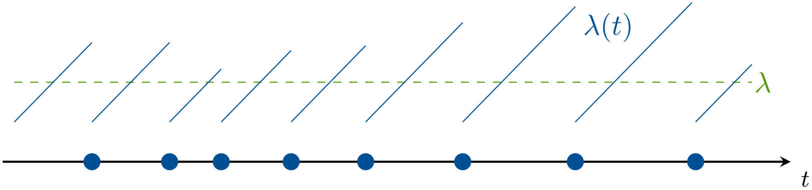


Increasing hazard rate $\rightarrow$ more periodic!

Theorem: for these types of traffic, keep the most popular is the optimal caching policy.

Can we improve upon this?

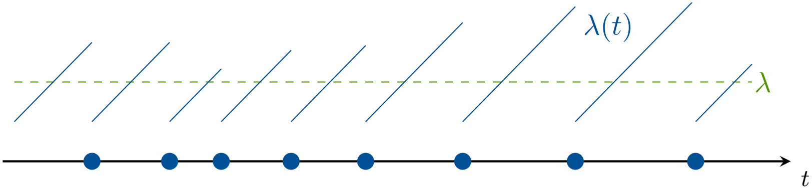
- Once you have seen a request, it's less likely to see the same item again for a while.



- What is the timer based equivalent of this case?

Thinking about increasing hazard rates...

- Once you have seen a request, it's less likely to see the same item again for a while.



- What is the timer based equivalent of this case?

Key insight

The question now is not how long we should remember something, but instead how long we should forget about it!

Timer based pre-fetching policy

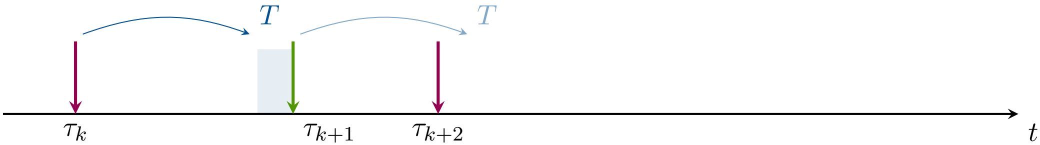
- Upon request arrival for item i, check for presence.

Timer based pre-fetching policy

- Upon request arrival for item i, check for presence.
- If not-present: start a timer T_i.

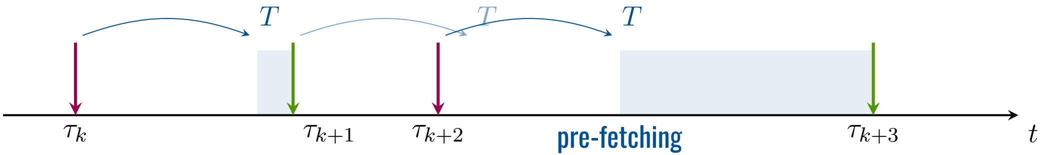
Timer based pre-fetching policy

- Upon request arrival for item i, check for presence.
- If not-present: start a timer T_i.
- If present: remove content and reset timer to T_i.



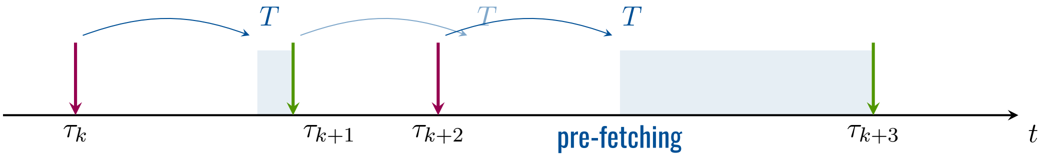
Timer based pre-fetching policy

- Upon request arrival for item i, check for presence.
- If not-present: start a timer T_i.
- If present: remove content and reset timer to T_i.
- Upon timer expiration, pre-fetch the content.



Timer based pre-fetching policy

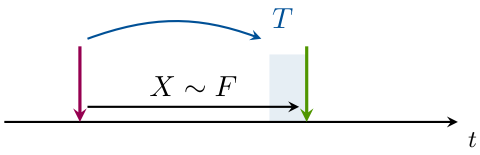
- Upon request arrival for item i, check for presence.
- If not-present: start a timer T_i.
- If present: remove content and reset timer to T_i.
- Upon timer expiration, pre-fetch the content.
- Keep timers T_i such that average cache occupation is C.



Timer based pre-fetching

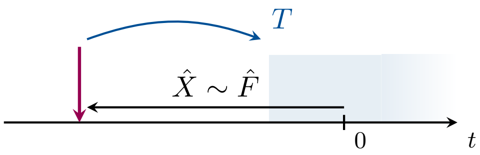
Consider a single item with a timer T and its request process:

Hit probability: next arrival occurs after timer expires.



Hit probability $= 1 - F(T)$

Occupation probability: probability that timer has expired by 0 since last arrival.



Avg. occupation $= 1 - \hat{F}(T)$

Choosing the optimal timers

Requests come from independent sources with intensities λ_i and inter-arrival distribution F_i:

Problem (Optimal pre-fetching policy)

Choose timers $T_i \geqslant 0$ such that:

$$\max_{T_i \geqslant 0} \sum_i \lambda_i (1 - F_i(T_i))$$

subject to:

$$\sum_i (1 - \hat{F}_i(T_i)) \leqslant C$$

Choosing the optimal timers

Requests come from independent sources with intensities λ_i and inter-arrival distribution F_i:

Problem (Optimal pre-fetching policy)

Choose timers $T_i \geqslant 0$ such that:

$$\min_{T_i \geqslant 0} \sum_i \lambda_i F_i(T_i)$$

subject to:

$$\sum_i \hat{F}(T_i) \geqslant N - C$$

Remark: we can use the same change of variables again!

- Apply the change of variables $u_i = \hat{F}_i(T_i)$.
- Note that u_i is the probability of not being stored.
- The problem becomes:

$$\min_{u_i \in [0,1]} \sum_i \lambda_i F_i(\hat{F}_i^{-1}(u_i))$$

subject to:

$$\sum_i u_i \geqslant N - C$$

- Objective gradient:

$$\frac{\partial}{\partial u_i}\lambda_i F_i \circ \hat{F}_i^{-1}(u_i) = \frac{\lambda_i f_i(\hat{F}_i^{-1}(u_i))}{\lambda_i \left(1 - F_i(\hat{F}_i^{-1}(u_i))\right)} = \eta_i(\hat{F}_i^{-1}(u_i))$$

- Increasing! $\rightarrow$ Proper convex optimization problem.

- Objective gradient:

$$\frac{\partial}{\partial u_i}\lambda_i F_i \circ \hat{F}_i^{-1}(u_i) = \frac{\lambda_i f_i(\hat{F}_i^{-1}(u_i))}{\lambda_i\left(1 - F_i(\hat{F}_i^{-1}(u_i))\right)} = \eta_i(\hat{F}_i^{-1}(u_i))$$

- Increasing! $\rightarrow$ Proper convex optimization problem.
- Lagrangian duality:

$$\begin{aligned}\mathcal{L}(u,\theta) &= \sum_{i=1}^{N} \lambda_i F_i\left(\hat{F}_i^{-1}(u_i)\right) + \theta\left(N - C - \sum_{i=1}^{N} u_i\right)\\ &= \sum_{i=1}^{N}\left[\lambda_i F_i\left(\hat{F}_i^{-1}(u_i)\right) - \theta u_i\right] + \theta(N-C).\end{aligned}$$

Main result

Theorem

If the F_i satisfy the IHR property, there exists a unique threshold $\theta^* \geqslant 0$ such that the optimal timers satisfy:

$$\eta_i(T_i^*) \geqslant \theta^*,$$

whenever $T_i^* < \infty$ (pre-fetching).
The inequality is strict if and only if $T_i^* = 0$, i.e. the content is always stored.

Main result

Theorem

If the F_i satisfy the IHR property, there exists a unique threshold $\theta^* \geqslant 0$ such that the optimal timers satisfy:

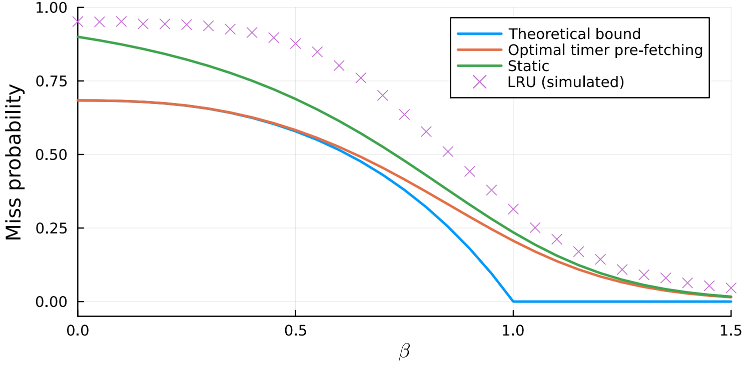
$$\eta_i(T_i^*) \geqslant \theta^*,$$

whenever $T_i^* < \infty$ (pre-fetching).
The inequality is strict if and only if $T_i^* = 0$, i.e. the content is always stored.

Remark: Again the policy is a threshold policy.

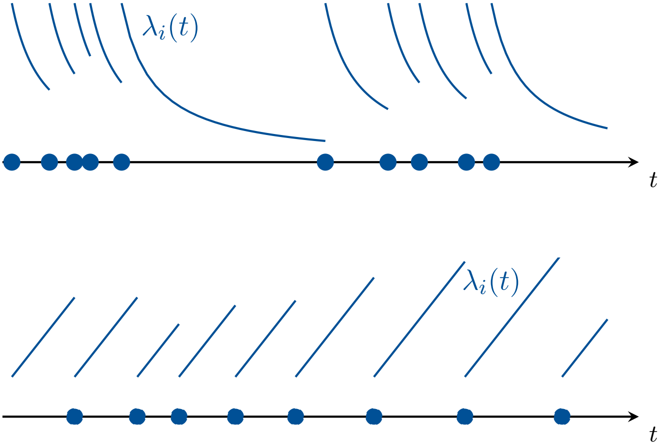
Simulation example

Erlang ($k = 5$) inter-arrival times, Zipf $\propto n^{-\beta}$ popularities, varying β, $N = 10000$, $C = 1000$.

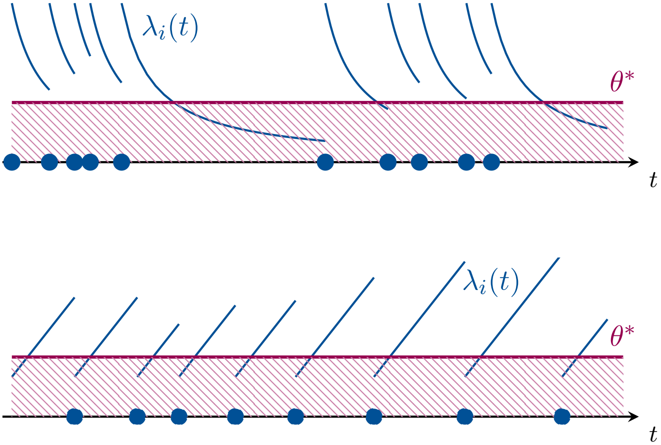


- Pre-fetching improves over the static policy.
- Classical caching (e.g. LRU) is a very bad idea for regular traffic.

A tale of two policies...



A tale of two policies...



Both policies are just the same policy!

- Keep a hazard rate threshold θ for storing a content
- Compute θ^* such that avg. memory occupation is C.

Asymptotic optimality

- Threshold policies in fact are related to recent results by [Panigrahy et al. 2022], about the optimal causal policy.
- They also identify a notion of hazard rate, the stochastic intensity, as defininf the optimal policy.

Asymptotic optimality

- Threshold policies in fact are related to recent results by [Panigrahy et al. 2022], about the optimal causal policy.
- They also identify a notion of hazard rate, the stochastic intensity, as defininf the optimal policy.

Theorem (F', Carrasco, Paganini – In preparation – check ArXiv soon)

Under an appropriate scaling regime, as $N \to \infty$ and $C = cN$, the optimal causal policy converges to a fixed threshold policy. Moreover, the limit threshold θ^* coincides with its timer based counterpart.

Therefore, the above policies give a universal asymptotic upper bound on caching performance.

Final remarks

- You have to know your traffic before deciding on caching or pre-fetching!

Final remarks

- You have to know your traffic before deciding on caching or pre-fetching!
- Classical caching is not well suited to regular traffic.

Final remarks

- You have to know your traffic before deciding on caching or pre-fetching!
- Classical caching is not well suited to regular traffic.
- We identified the hazard rate as the crucial indicator of regularity, and devised a new policy for IHR, that is also asymptotically optimal among all causal policies.

Final remarks

- You have to know your traffic before deciding on caching or pre-fetching!
- Classical caching is not well suited to regular traffic.
- We identified the hazard rate as the crucial indicator of regularity, and devised a new policy for IHR, that is also asymptotically optimal among all causal policies.
- A lot of open questions, in particular:
  - How we can learn the hazard rates online?
  - How we can estimate the appropriate threshold?
  - What about mixtures of IHR and DHR traffic?

Tante grazie!

Andres Ferragut (and Diego Goldsztajn)
ferragut@ort.edu.uy
aferragu.github.io

References I

P. Brémaud.
Point process calculus in time and space.
Springer, NY, 2020.

H. Che, Y. Tung, and Z. Wang.
Hierarchical web caching systems: Modeling, design and experimental results.
IEEE Journal on Selected Areas in Communications, 20(7):1305–1314, 2002.

A. Ferragut, I. Rodriguez, and F. Paganini.
Optimizing TTL caches under heavy tailed demands.
In Proc. of ACM/SIGMETRICS 2016, pages 101–112, June 2016.

A. Ferragut, I. Rodríguez, and F. Paganini.
Optimal timer-based caching policies for general arrival processes.
Queueing Systems, 88(3–4):207–241, 2018.

References II

N. C. Fofack, P. Nain, G. Neglia, and D. Towsley.
Performance evaluation of hierarchical TTL-based cache networks.
Computer Networks, 65:212–231, 2014.

N. K. Panigrahy, P. Nain, G. Neglia, and D. Towsley.
A new upper bound on cache hit probability for non-anticipative caching policies.
ACM Trans. Model. Perform. Eval. Comput. Syst., 7(2–4), November 2022.