Tutorial 1: Online Learning for Data Caching and Network Service Delivery

Parts I-II: intro+ single cache models

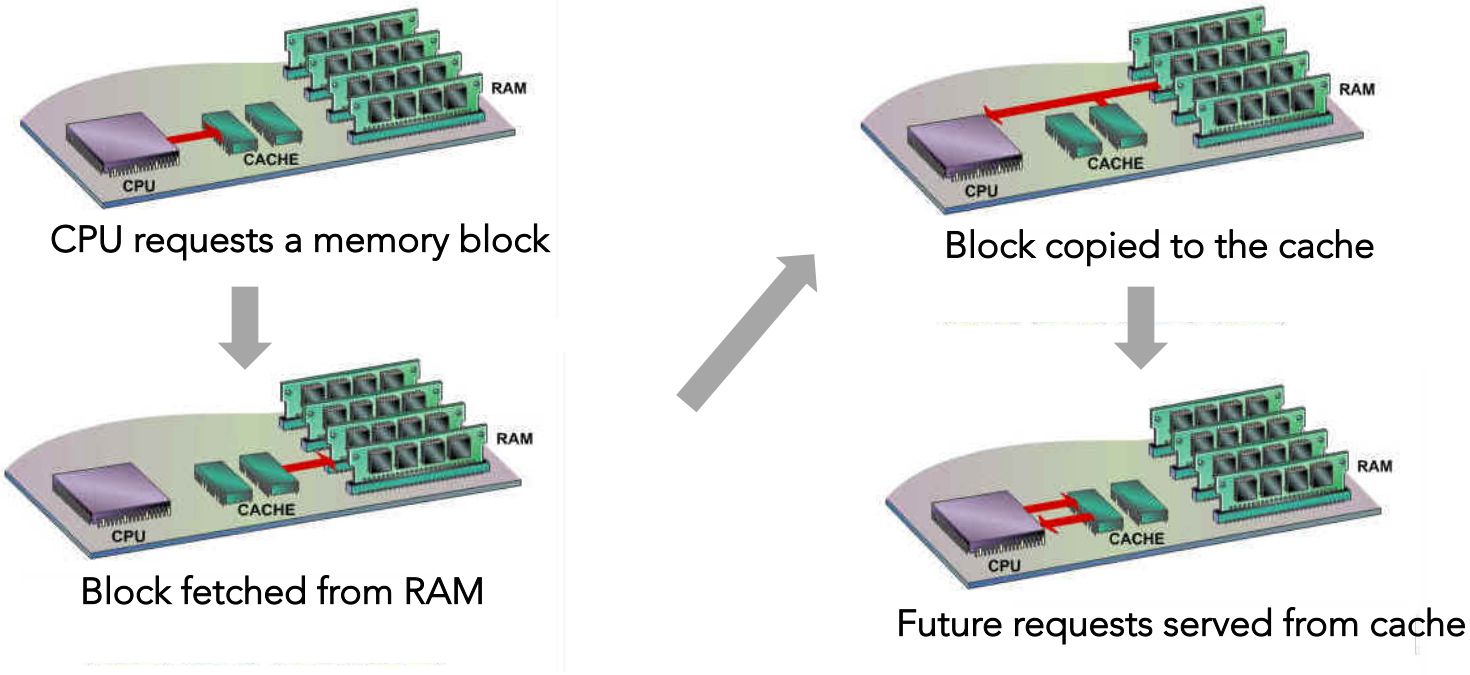
Georgios Paschos (Amazon, Luxembourg)*
George Iosifidis (Trinity College Dublin)
Apostolos Destounis (Huawei)

ACM ICN, 2020

*The work of Georgios was completed before joining Amazon

The Computer Cache of 70s

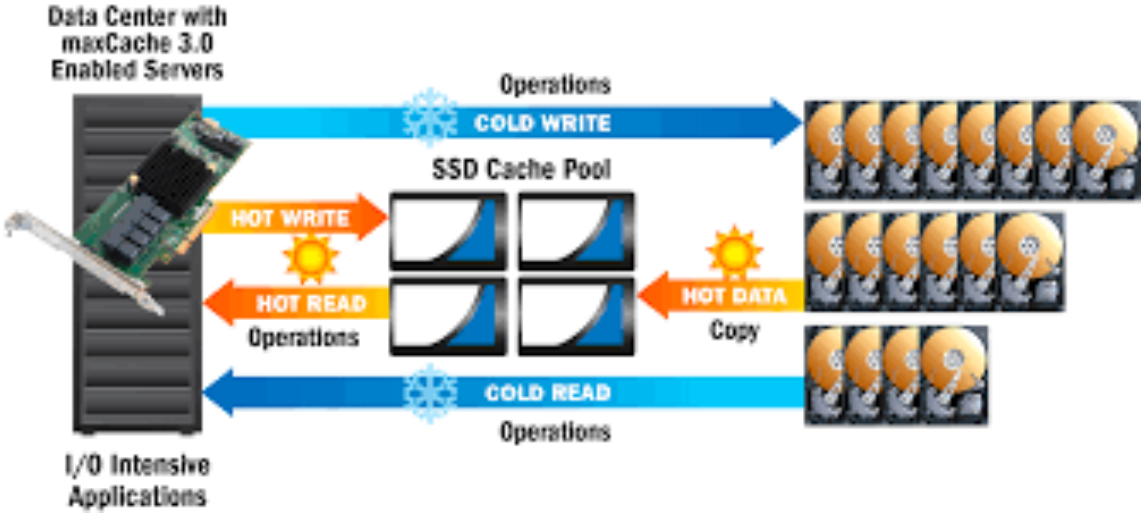
- Idea: a small fast memory can accelerate memory access



Why it works? Some memory blocks become hot (popular)

Caching in Datacenters

- **Benefit**: fast & cheap cloud storage with HDDs and SSDs

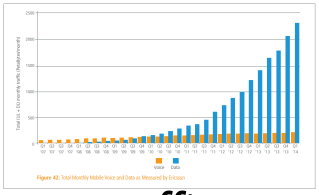


- Which block is "hot"?

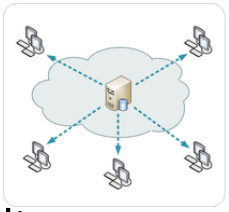
The Internet cache of 90s

Sir Tim Berners Lee
(inventor of www)

1990: This doesn't scale!

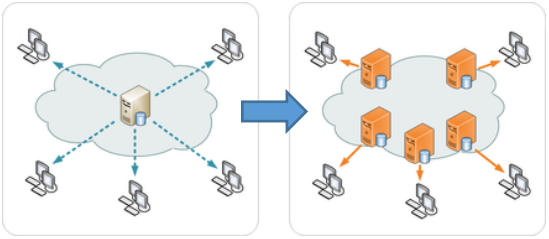


traffic



client-server

1995: Eureka!



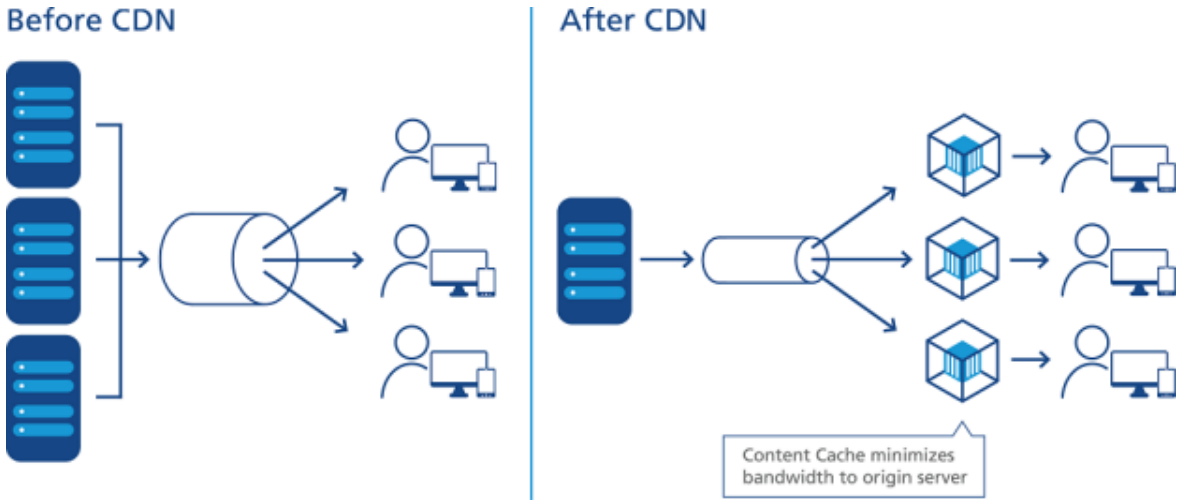
Content Delivery Networks

Prof. Tom Leighton
(Akamai founder)

Why it works? Some web pages become popular

Content Delivery Networks (e.g. Akamai)

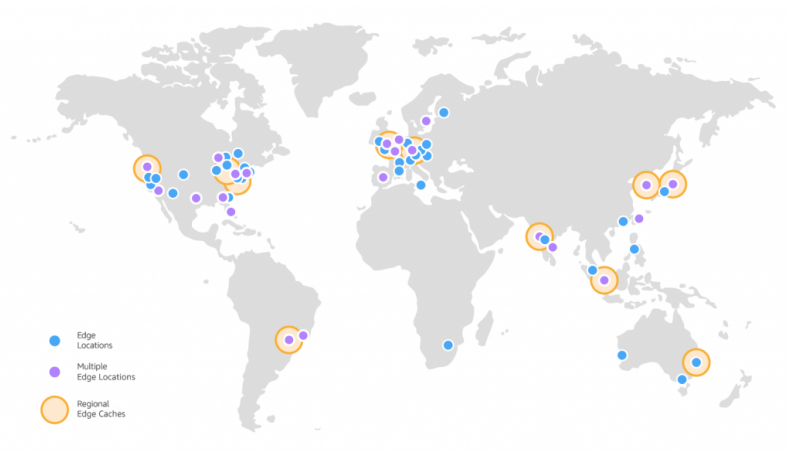
- **Benefit**: economize and accelerate content delivery

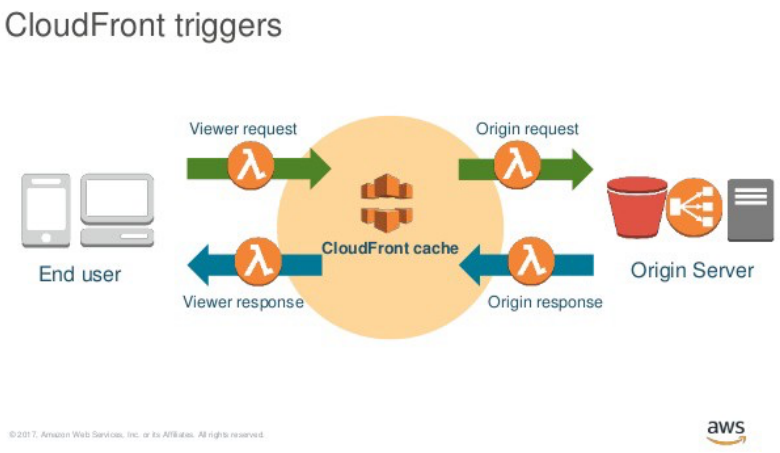


- Where to place servers to meet changing demand?

Cloud CDNs (e.g. AWS CloudFront)

- **Benefit:** CDN as a service

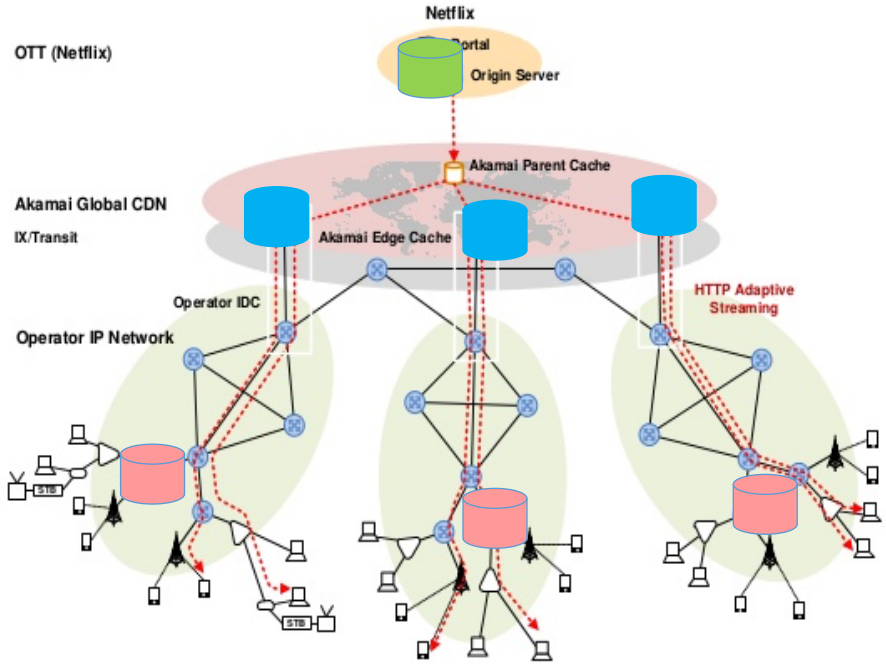




- Deploy virtual service to minimize renting cost subject to QoS

Telco CDNs

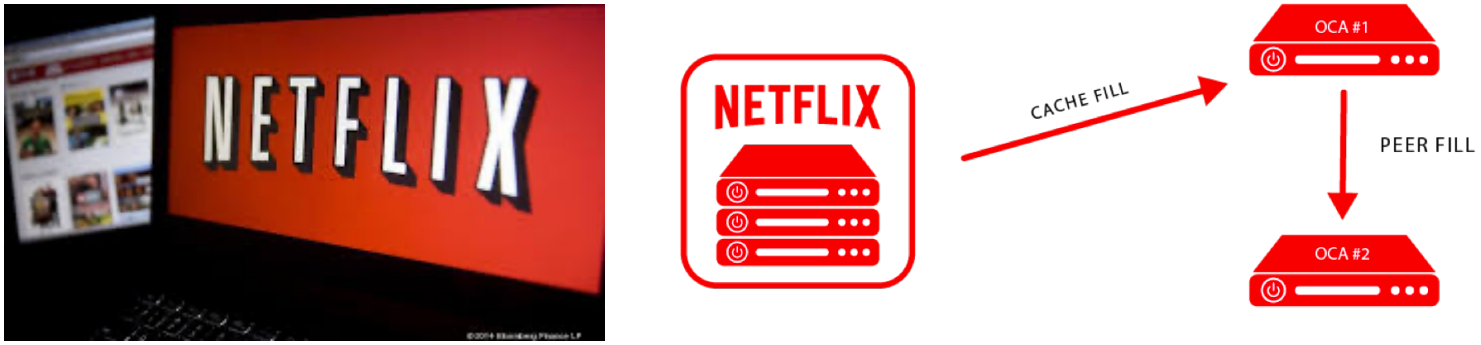
- **Benefit:** serve at the edge



- At which density/depth to replicate each content?

Private CDNs: Netflix Open Connect

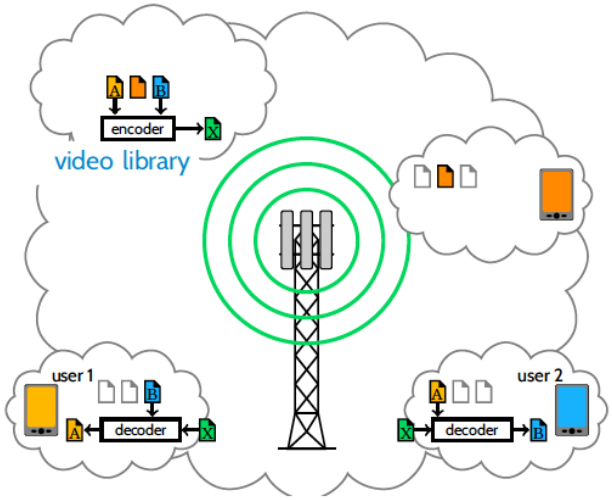
- **Benefit:** cheap movie delivery



Netflix "fills" Open Connect caches and "serves" the customers

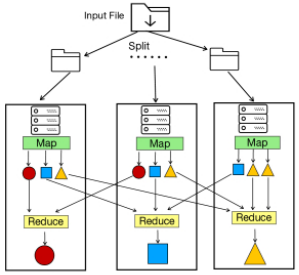
- Which movies to push for tomorrow's traffic?

Future: The role of caching



Wireless downlink

High bandwidth applications

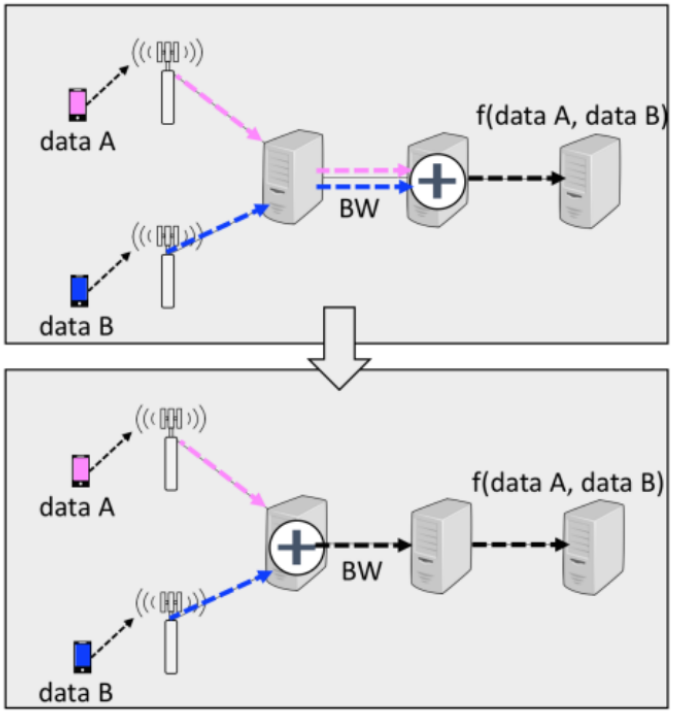


Parallel computing

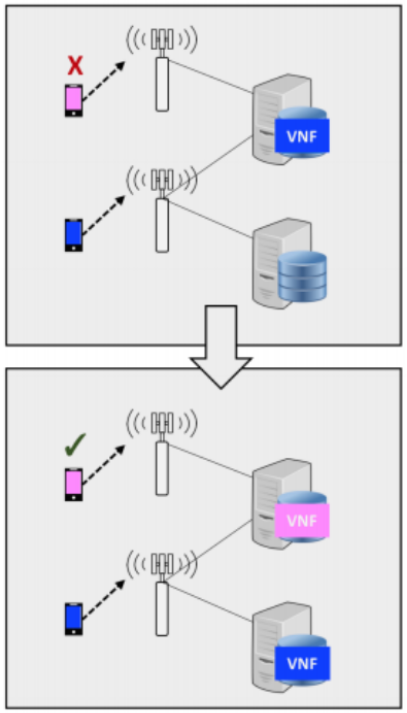
Memory provides side-information to enhance communications

Paschos et al., The role of caching in future communication systems and networks, IEEE JSAC, 36 (6), 2018

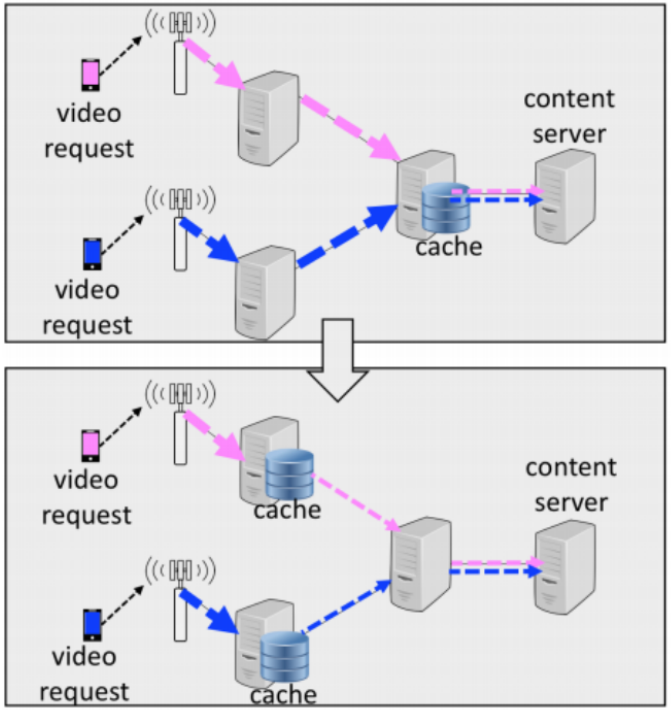
Future: network & memory



Data service placement



Functionality placement



Memory placement

Learn to manage network resources
- Network optimization
- Machine learning

This Tutorial

- Algorithms that can **learn** how to best allocate resources for network caching, delivery, and services

Agenda

- Part II: Single cache model (35min)

Q&A (10min)

Break (15min)

- Part III: Optimal placement in networks (35min)

Q&A (15min)

Break (10min)

- Part IV: Advanced learning algorithms for resour allocation (35min)

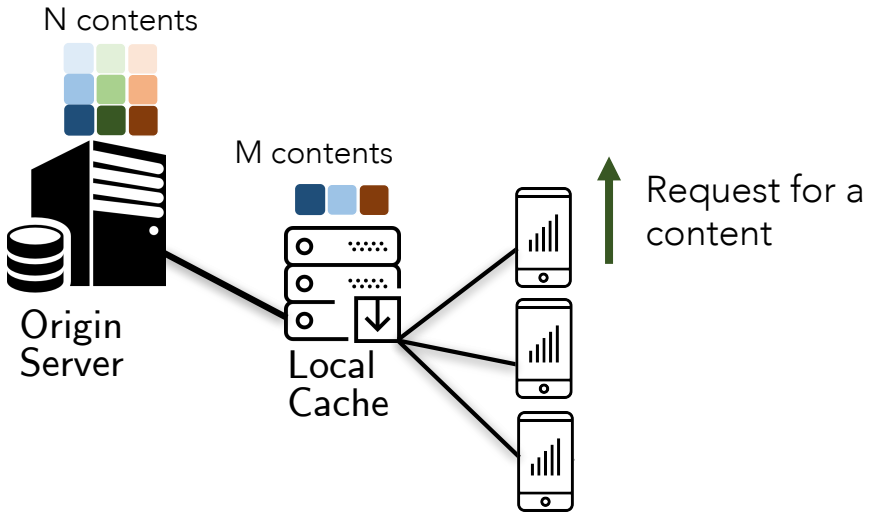
Q&A (15min)

Outline of Part II: Single cache model

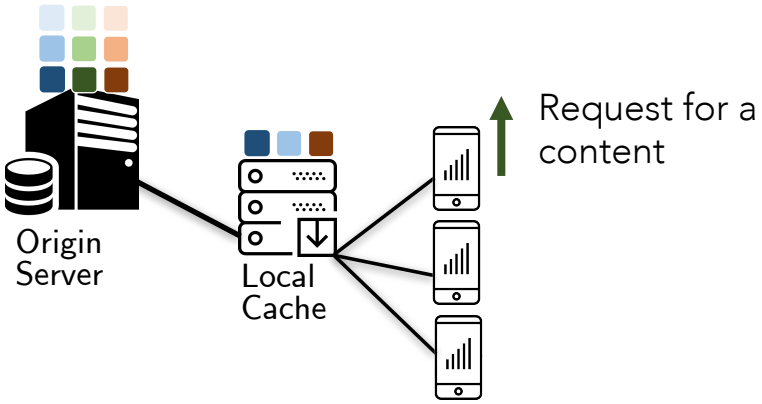
1. Snapshot optimization
2. Static popularity - LFU
3. Dynamic popularity – ABT
4. Adversarial requests - LRU
5. Universal caching based on online learning

The model of one cache

- Origin server holds N contents
- Local cache can store any M contents
- A content is requested with probability p_n - called popularity
  - "hit" if the requested content is stored in the cache
  - "miss" otherwise



Snapshot optimization



- Popularities are known. Which content placement y maximizes hit probability?

$$h(\boldsymbol{y}^*) = \max_{(y_n)\in\{0,1\}^N} \sum_{n=1}^{N} y_n p_n$$

$$\text{s.t.} \sum_{n=1}^{N} y_n = M$$

Generalizes to networks of caches – see Part II

Optimal placement

Cache most popular contents

$$h(\boldsymbol{y}^*) = \sum_{n=1}^{M} p_n$$

Zipf popularity

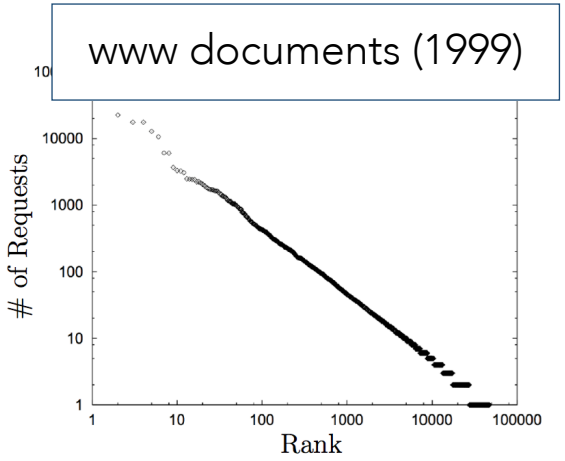
- Zipf law with parameter **τ**

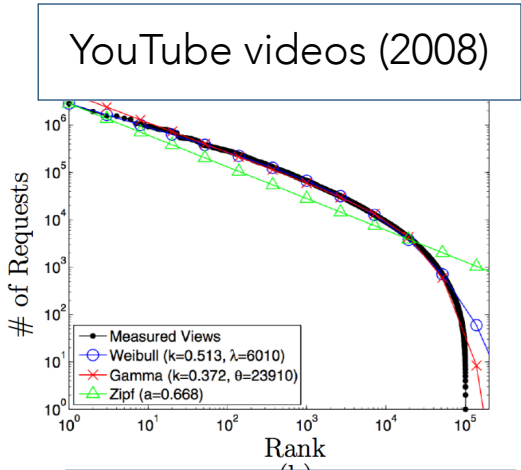
$$p_n = \frac{n^{-\tau}}{\sum_{i=1}^{N} i^{-\tau}}, \quad \underbrace{n = 1, \ldots, N}_{\text{popularity rank}}$$

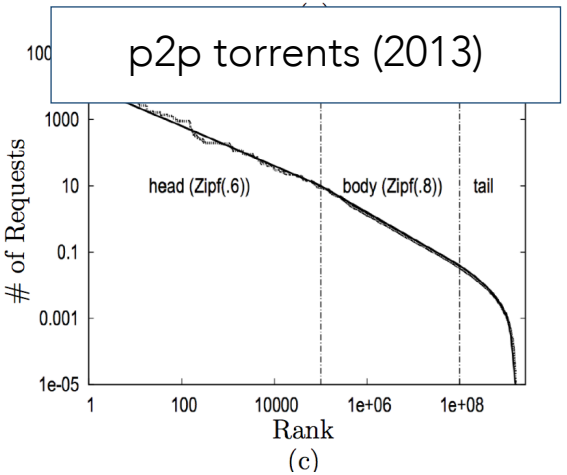
- Approximation of max hit prob.

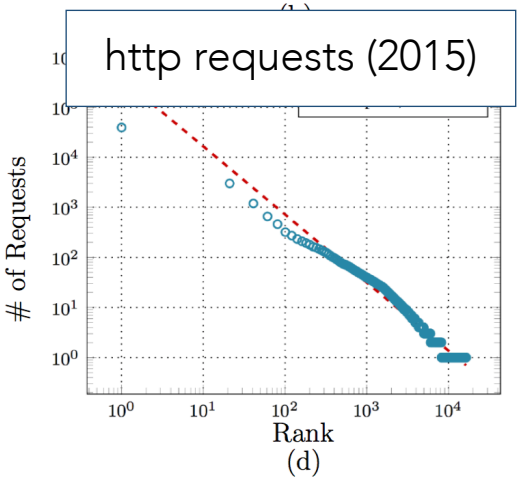
$$h(\boldsymbol{y}^*) \approx \begin{cases} \left(\frac{M}{N}\right)^{1-\tau} = \gamma^{1-\tau} & \text{if } 0 \le \tau < 1 \\ \frac{\ln M}{\ln N} = 1 - \frac{\ln \gamma^{-1}}{\ln N} & \text{if } \tau = 1 \\ 1 & \text{if } \tau > 1 \end{cases}$$

Depends on Zipf exponent and **γ**=M/N



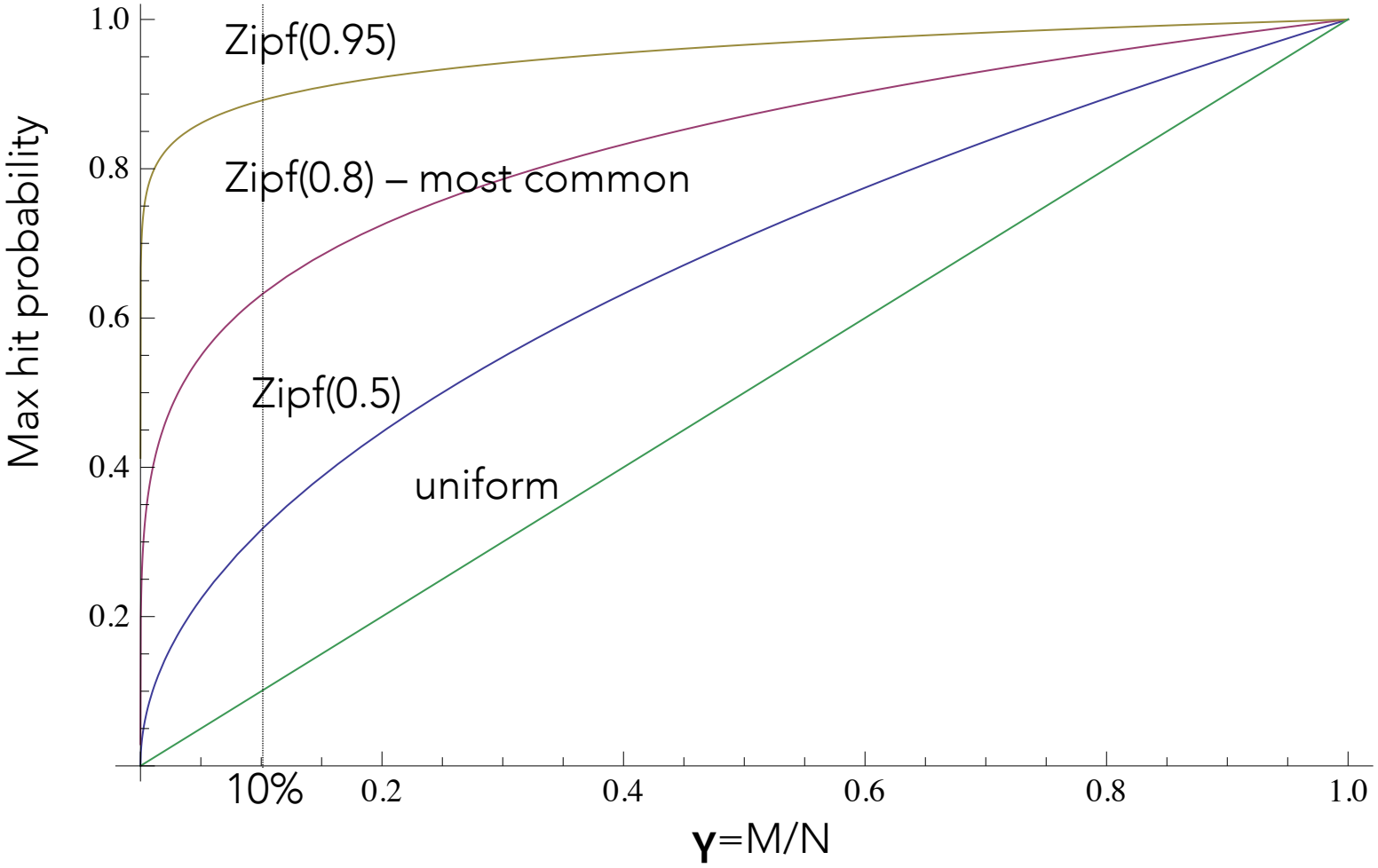






M. Mitzenmacher, A brief history of generative models for power law and lognormal distributions, Internet Mathematics, 1(2), 2003

Caches like skewed popularities



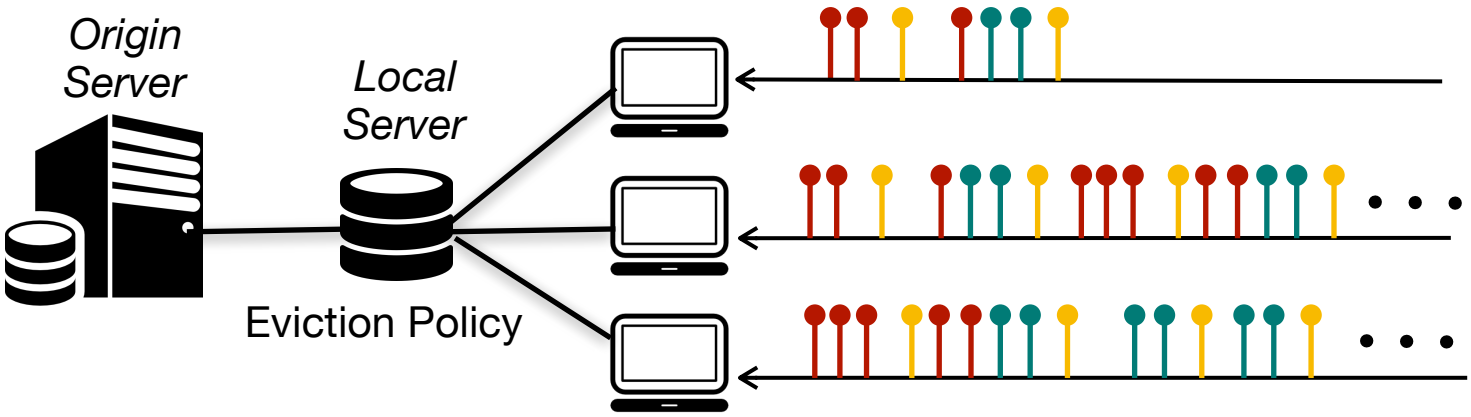
Summary: snapshot optimization

- Cache the most popular
- Extends to networks (a.k.a. femtocaching)

How do we learn which are the most popular contents?

Cache eviction policies

- The cache receives a request sequence $R = (R_1, \ldots, R_T)$

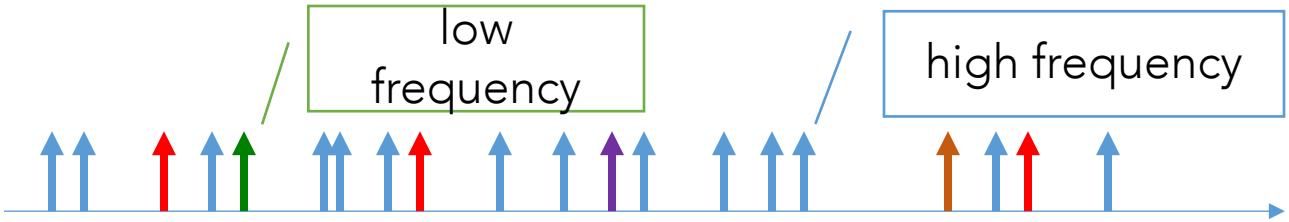


- Each request is hit or miss; for miss, select a content to evict

Eviction policies are learning algorithms;
to succeed they must learn which contents are popular

Least Frequently Used (LFU)

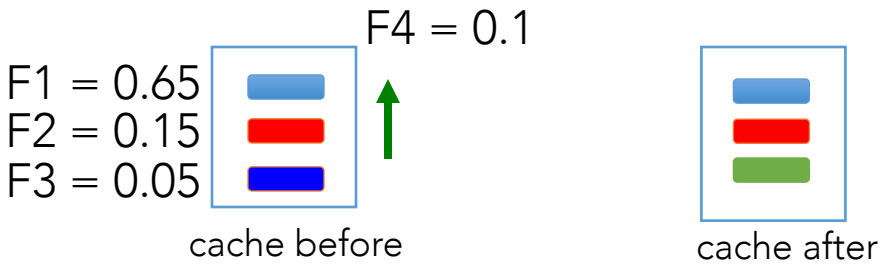
- Monitor the request frequency of contents



LFU

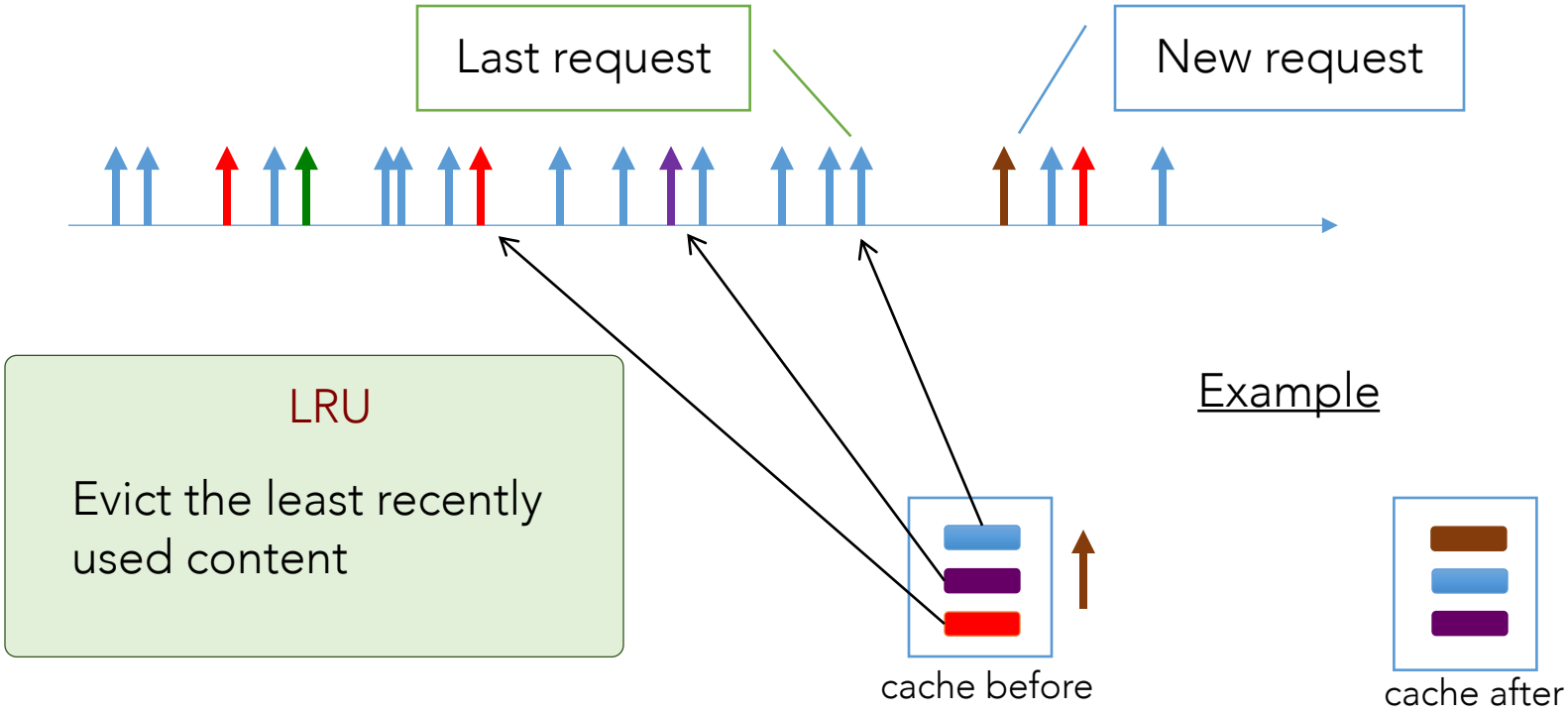
Evict the least frequently used content

Example



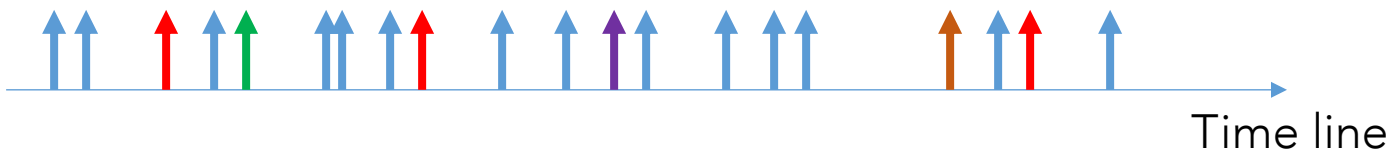
Least Recently Used (LRU)

- Keep contents in an ordered list



Independent Reference Model (p_n is time-invariant)

- IRM = Request sequence is i.i.d.



- Poisson IRM (similar): Marked Poisson process with rate $\boldsymbol{\lambda}$
  - Marks are IRM

Easy to fit. Effective in modeling systems with very large number of requests

LFU under IRM

- Intuition:
  - Frequency is an unbiased estimator of popularity
  - If popularity is static (IRM), frequencies converge to popularities by LLN
  - …and LFU eventually caches the most popular

Theorem (LFU is optimal for IRM):

Suppose the request sequence is IRM. LFU achieves the maximum limiting hit probability among all online eviction policies.

Paschos, Iosifidis, Caire, Cache Optimization Models & Algorithms, Foundations & Trends in Comm. & Inf. Theory, 2020

LRU under IRM: Markovian analysis

- Cache configuration is a Markov chain
  - Hit probability is given by stationary probability
  - Requires enumeration of all possible cache permutations

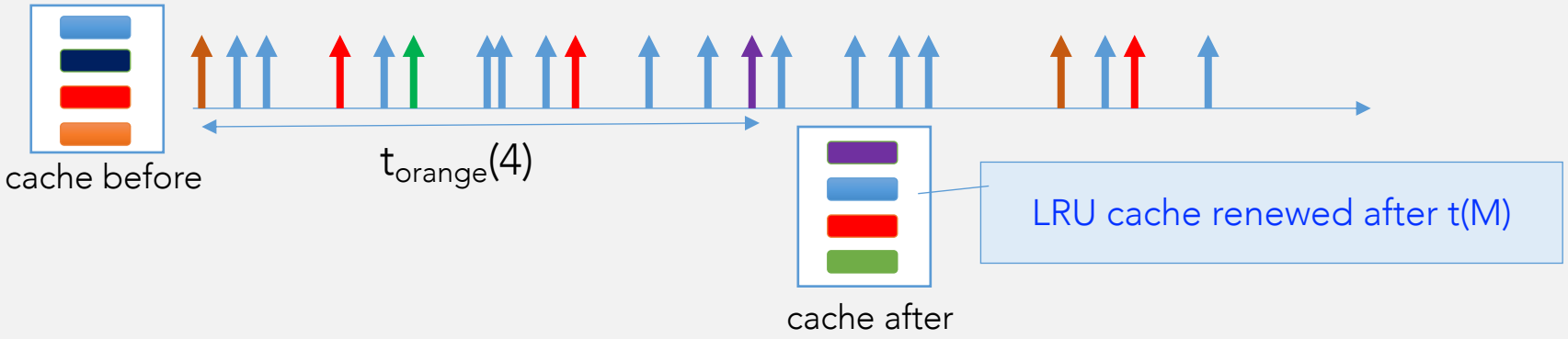
$$\sum_{\sigma} \frac{p_{\sigma(1)} p_{\sigma(2)} \cdots p_{\sigma(M)} \left(1 - p_{\sigma(1)} - p_{\sigma(2)} - \cdots - p_{\sigma(M)}\right)}{\left(1 - p_{\sigma(1)}\right)\left(1 - p_{\sigma(1)} - p_{\sigma(2)}\right) \cdots \left(1 - p_{\sigma(1)} - p_{\sigma(2)} - \cdots - p_{\sigma(M-1)}\right)}$$

Markovian analysis is computationally expensive
works only for small caches

W. F. King, Analysis of Paging Algorithms, Computer Science, 1971

LRU under IRM: Che approximation

- Characteristic time = elapsed time of M different requests without a request for n
  - Example M=4

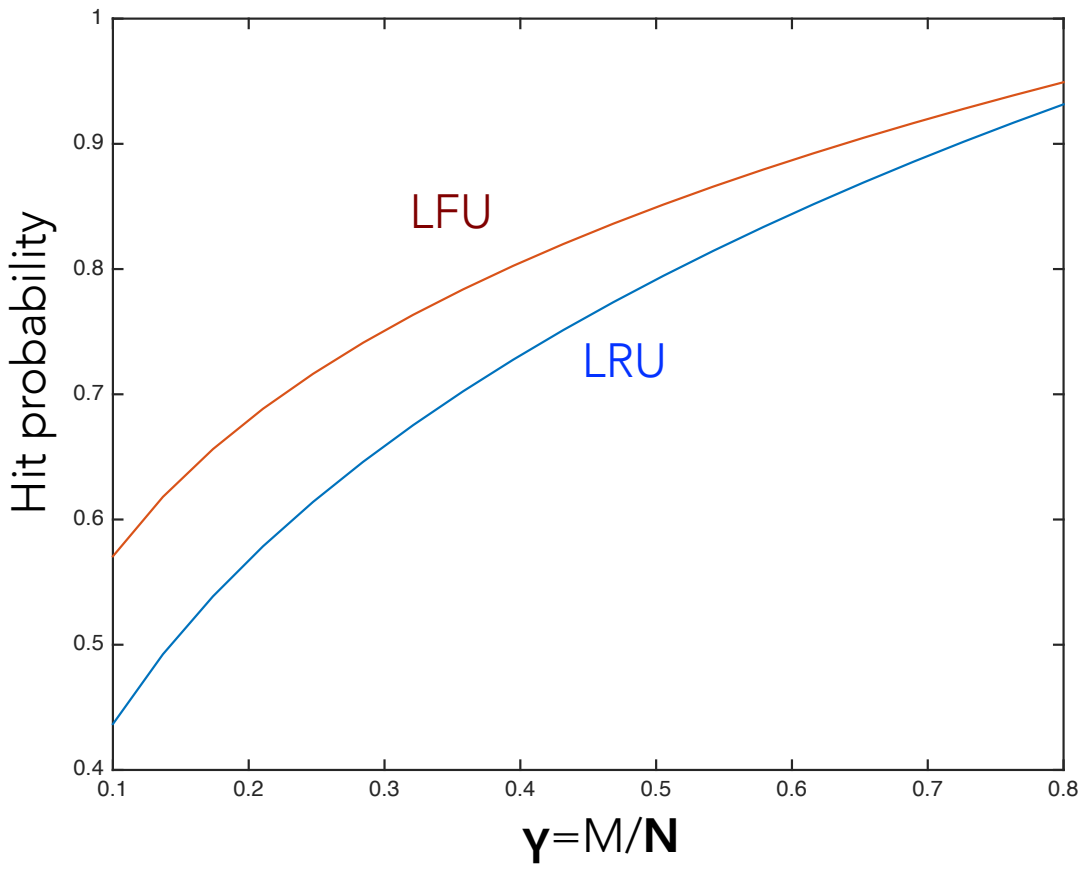


- [Che02]: characteristic time RV is concentrated around its mean
  - Hit probability of LRU: $h^{LRU} = \sum_n p_n h_n(M)$ where $h_n(M) = 1 - e^{-\lambda p_n t(M)}$
  - Characteristic time is the unique solution of : $M = \sum_j (1 - e^{-\lambda p_j t(M)})$

- Asymptotically exact [Fricker12]

R. Fagin, Asymptotic miss ratios over independent references, Jrn. of Comp. & System Sc., 1977
H. Che et al, Hierarchical web caching systems: modeling, design and experimental results, IEEE JSAC, 2002

LRU vs LFU under IRM [τ=0.8]



For static popularity, LFU is optimal. Why people use LRU?

Summary: static popularity

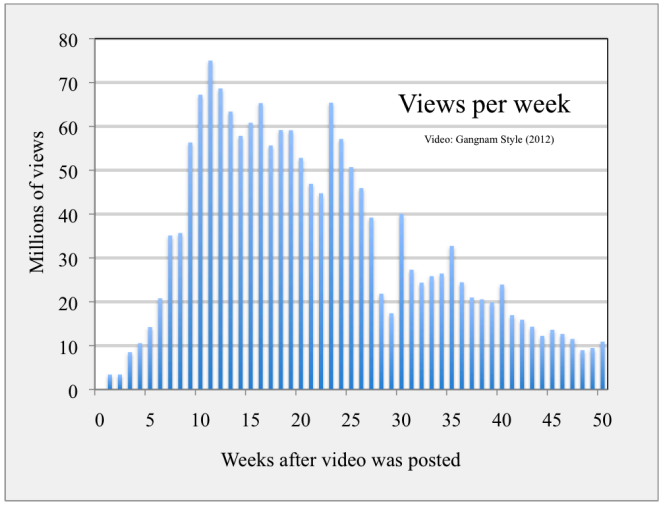
- LFU (mean estimator) is optimal for static popularity
- LRU performance can be approximated

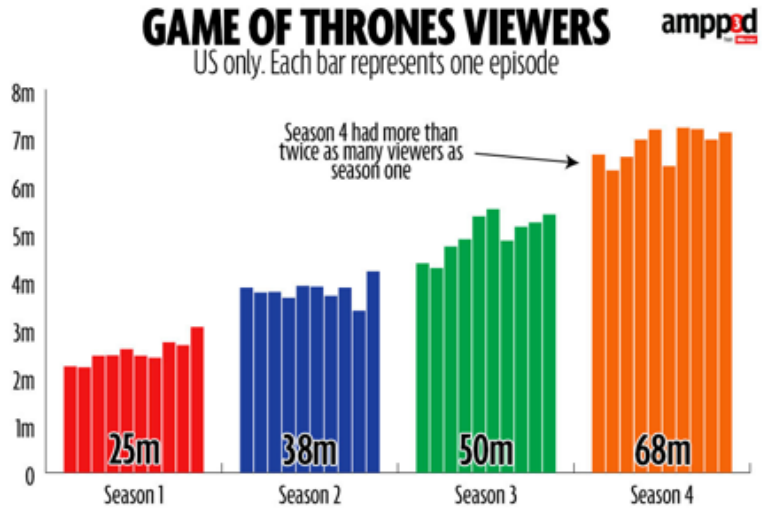
Is popularity static?

Not really!
..but can be assumed static in short time frames

Popularity is time-varying

"Gangnam style" weekly views

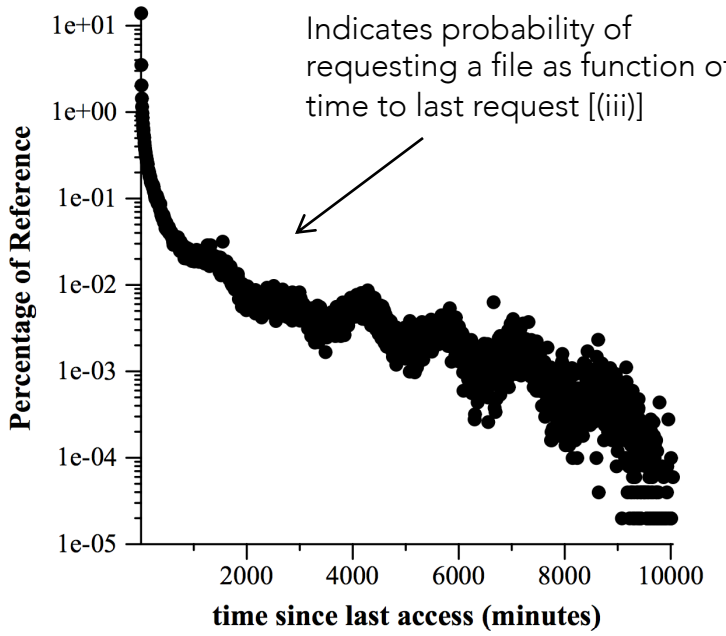




- Viral content
  - Series episodes
  - Viral Youtube videos
  - Trending News
  - Social media (Twitter, Fb, Instagram)

Temporal Locality

- Time correlations: recently requested are locally popular (hot)



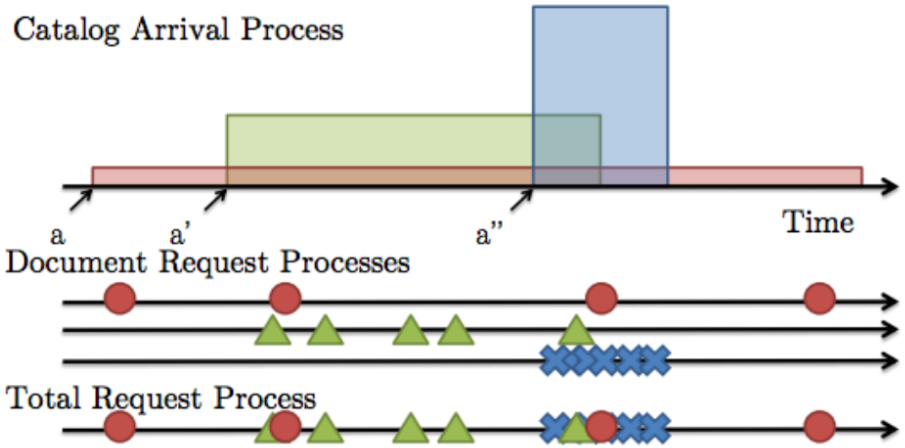
Youtube popularity changes at time-scale of days [(ii)]

Half of the top-25 wikis are different from day to day [(i)]

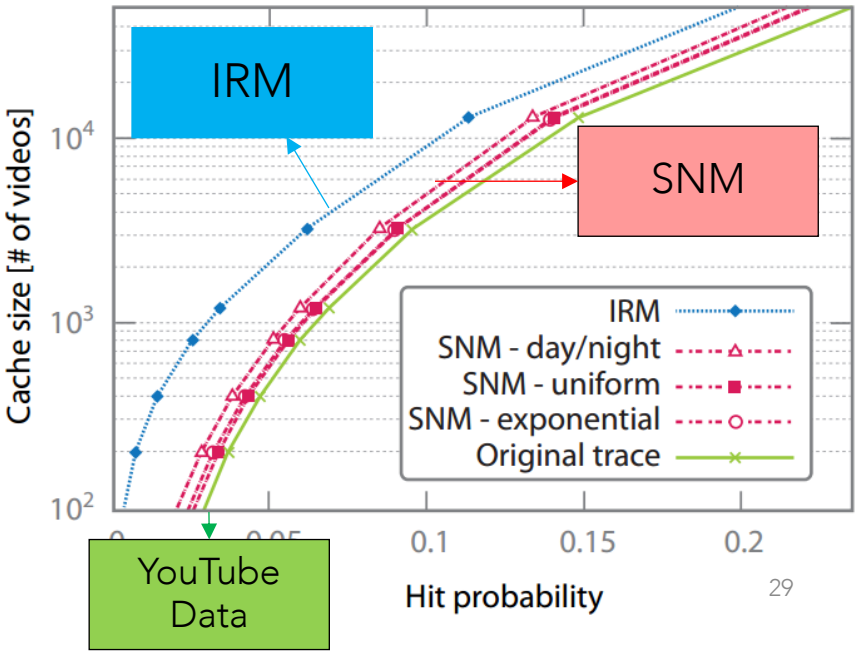
(i) G. Hassinger et al, Performance evaluation of new web caching strategies, ITC' 2016
(ii) S. Traverso et al, Temporal locality in today's content caching: Why it matters and how to model it, ACM CCR, 2013
(iii) C. Cao, et al., Cost-aware www proxy caching algorithms, in Proc. of Usenix, 1997

Shot Noise Model (p_n varies with time)

- "shot height" determines request intensity



Fits YouTube traces better than IRM



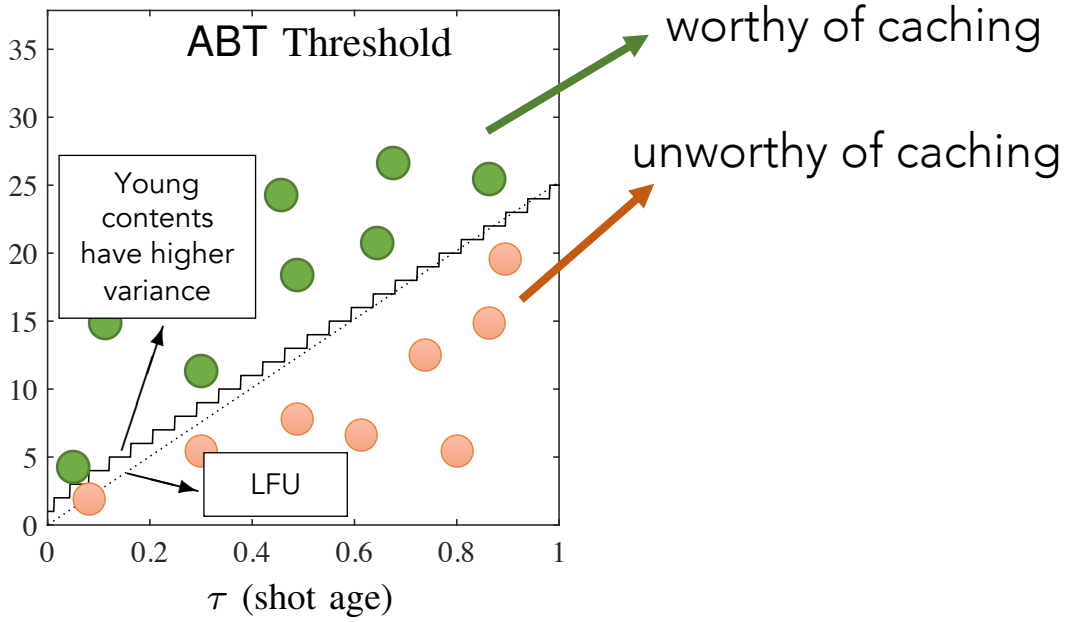
[Traverso13] "Temporal locality in today's content caching: Why it matters and how to model it"

[Olmos14] "Catalog dynamics: Impact of content publishing and perishing on the performance of a LRU cache"

Content classification for SNM

Age-Based Thresholding

LFU with underbias on young contents



Theorem (ABT is optimal for SNM):

Suppose the request sequence is rectangular-SNM. ABT achieves the optimal hit rate asymptotically as N->inf.

M. Leconte et al, Placing dynamic content in caches with small population, INFOCOM, 2016

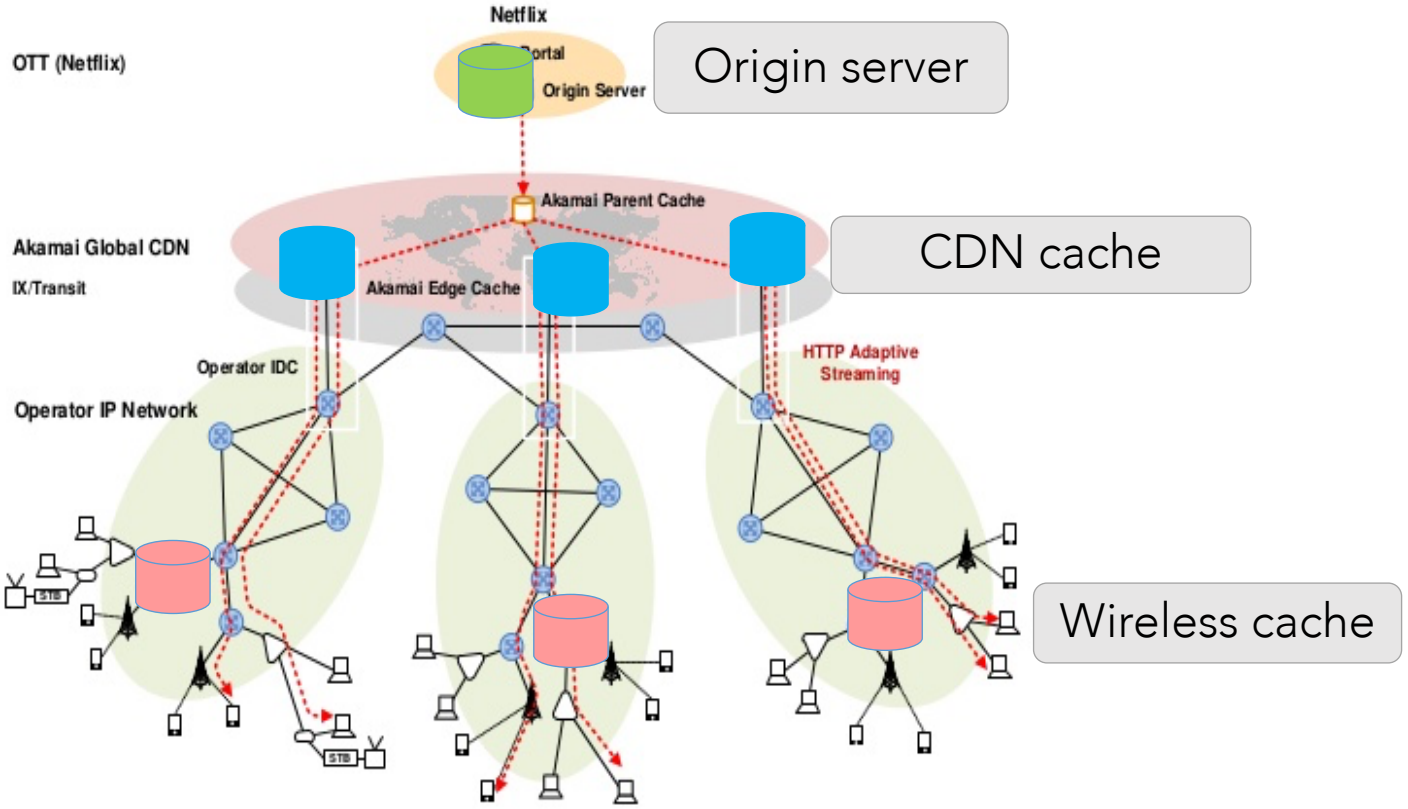
Popularity learning in metro CDNs

- Local LRU ineffective: Cache local, learn global

50 req/day/item

10 req/day/item

0.1 req/day/item



S. E. Elayoubi et al, Performance and cost effectiveness of caching in Mobile Access Networks, in ACM ICN'15
M. Leconte et al, Placing dynamic content in caches with small population, INFOCOM'16
Paschos et al, Wireless Caching: Technical Misconceptions and Business Barriers, IEEE Comm. Mag., 2016

Summary: non-stationary models

- SNM captures temporal locality
- ABT is optimal for SNM
- LRU analysis [Leonardi14]
- Intensity of requests affects learning

Challenges with non-stationary models

- Complex fitting process
- Model-dependent learning

E. Leonardi, et al, Least recently used caches under the Poisson Shot Noise model, INFOCOM, 2015

The paging problem (worst-case)

- Find a policy that maximizes the hits under any sequence R

Belady's oracle (MIN)

Evict the content requested further in the future

Theorem (Belady pathwise optimality):

For any sequence R, Belady has at least as many hits as any other eviction policy.

Belady is an offline benchmark.
What can we say about online policies?

[Belady66] "A study of replacement algorithms for virtual storage computers"
[Mattson70] "Evaluation techniques for storage hierarchies"
[Kao08, pp. 601–604] "Encyclopedia of algorithms"

Competitive ratio

- For every online policy there is a 0-hit sequence where Belady has non-zero hits
  - All online policies have same competitive ratio! ☹

- Cache size disadvantage. A policy is (**γ**,**ρ**)-competitive if for any R it has **γ** more misses than a **ρ** smaller Belady cache

Theorem (Lower bound on competitive ratio):

Suppose Belady is given cache size x. Every policy with cache size M>x misses at least:

$$m^{\pi}(R, M) \geq \left(\frac{M}{M - x + 1}\right) m^{*}(R, x)$$

[Sleator85] "Amortized efficiency of list update and paging rules"

LRU has optimal competitive ratio

Theorem (LRU is $\left(\frac{M}{M-x+1}, \frac{x}{M}\right)$ -competitive):

Suppose Belady is given cache size x. For any sequence R, LRU satisfies:

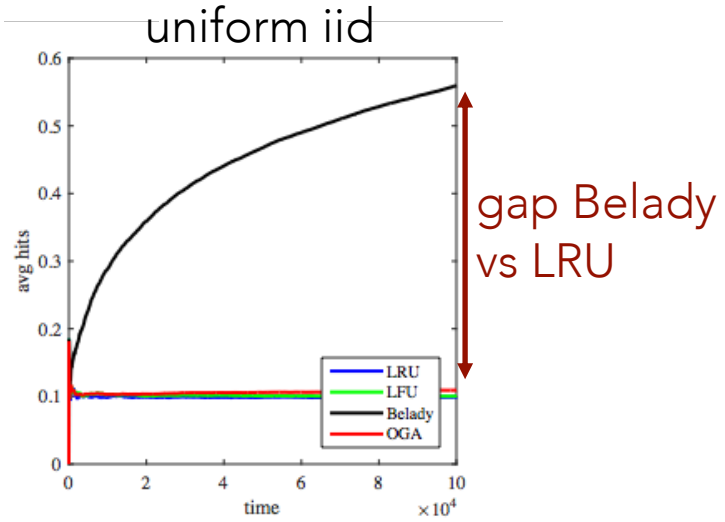
$$m^{LRU}(R, M) \leq \frac{M}{M-x+1} m^{*}(R, x), \quad \forall R.$$

- GreedyDualSize extends LRU to content size
- Landlord to hit costs

[Cao97] "Cost-aware www proxy caching algorithms"
[Young12] "The k-server dual and loose competitiveness for paging"

Summary: adversarial requests

- LRU achieves the optimal competitive ratio, but…



Conservative eviction
No more than M misses on any sequence with at most M different contents
e.g. FIFO, LRU, CLOCK

However:
- Every "conservative" policy has optimal competitive ratio
- Gap to Belady can be pretty large

Other cache eviction policies

Name	Eviction rule	Guarantee	Cplx	Ref.
Belady	max stack distance	max hits pathwise	Not Appl.	[24, 166]
LRU	least recently requested	competitive	★	[221]
FIFO	first-in	competitive	★	[88]
WSCLOCK	linked list LRU	competitive	★	[45]
GreedyDualSize	least size-related credit	competitive	★★★	[42]
LANDLORD	least size & utility credit	competitive	★★★	[247]
LFU	least frequently requested	opt. stationary	★★	–
q–LRU	cache with prob q, evict using the LRU policy	opt. stat. $q \to 0$	★	[88]
LRU-k	k LRU stages	opt. stat. $k \to \infty$	★★	[180, 88]
TTL	upon timer expiration	opt CUM stat.	★	[80, 69]
RANDOM	random	None	★	[88]
CLIMB	hit moves up one position, evict tail	None	★	[11, 89, 224]
Partial LRU	LRU per chunk	None	★★	[238, 162]
Score-gated	threshold-based	None	★★	[106]
ARC	split cache to LRU and LFU	4–competitive	★★	[167, 60]
CAR	combine ARC and CLOCK	18–competitive	★★	[15, 60]

Paschos, Iosifidis, Caire, Cache Optimization Models & Algorithms, Foundations & Trends in Comm. & Inf. Theory, 2020

Summary so far

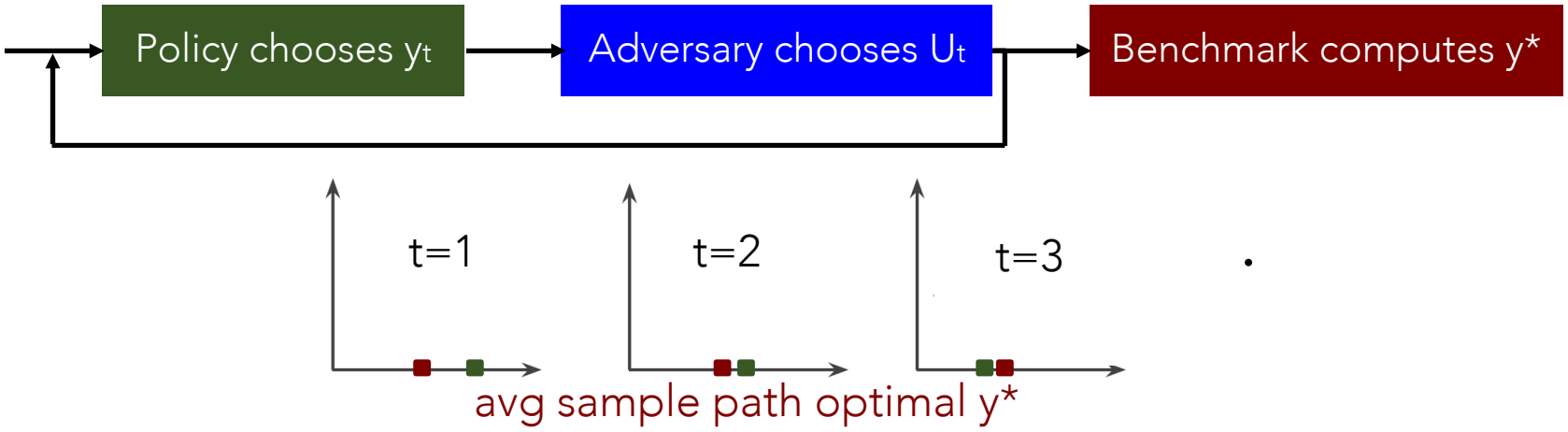
- Belady is path-wise optimal, needs knowledge of the future
- LFU is optimal for IRM, suffers from stale contents
- ABT is optimal for SNM, but model-dependent
- LRU has optimal competitive ratio, but caches too many unpopular contents

Challenge

Design a caching policy that works under any request model

Online Convex Optimization

- At each slot choose action y_t and receives a loss $U_t(y_t)$ where U_t is convex and chosen from a time-variant distribution (even by an adversary)

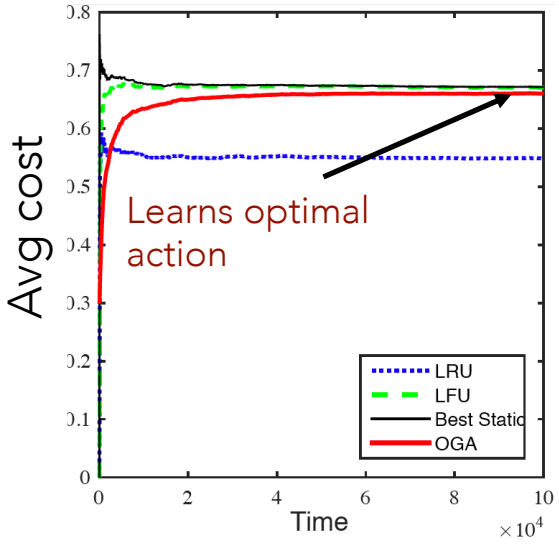


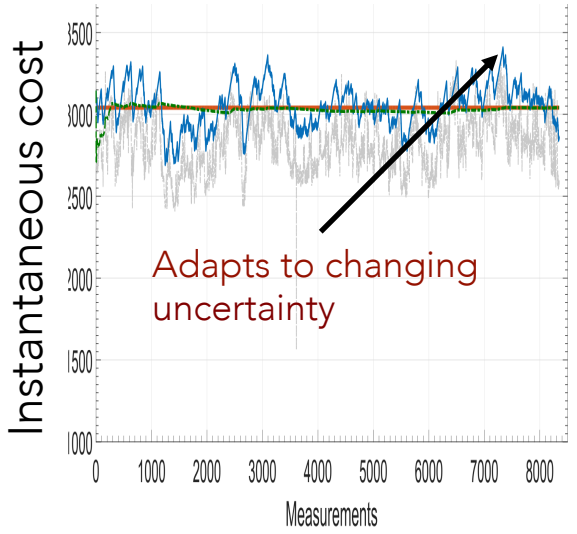
- Key idea: compare policy loss vs sample path opt action $y^* = \arg\max \sum_t U_t(y)$

Regret: $\sum_t U_t(y_t) - \sum_t U_t(y^*)$

"No regret" policies

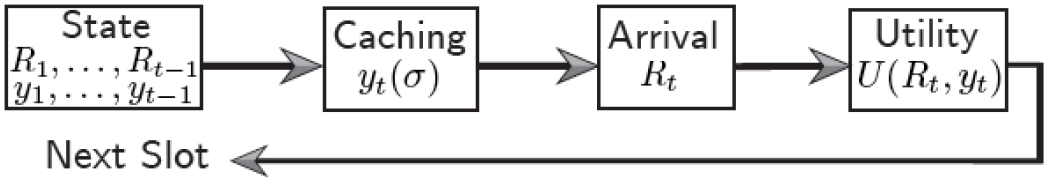
- A policy satisfying $\sum_t U_t(y_t) - \sum_t U_t(y^*) = o(T)$ is called no regret policy
- As $T \to \infty$, learns the optimal average action





How do we derive a no-regret cache eviction policy?

OCO formulation of caching



storage constraint

1. **Policy chooses:** $y_t^n \in [0,1]$ = fraction of content n to cache $\sum_n y_t^n \leq M$ (cache size)
2. **Adversary** chooses a file request $R_t^n = 1$, iff n requested
   - A linear reward is collected $U_t(y) = \sum_n w^n R_t^n y^n$ → only requested provides reward
   - Choose $w^n = 1$ for hit maximization
3. **Benchmark** y^* : cache the most frequently requested

$$y^* = \arg\max_{\substack{y \in [0,1]^N \\ \sum_n y^n \leq M}} \sum_t U_t(y)$$

Regret of classical policies

Lemma (Regret of LRU, LFU):

There exist sequences R for which LRU and LFU have regret $\Omega(T)$.

- *There exist request models where LRU, LFU take consistently suboptimal actions and do not improve over time*
  - Periodic requests with period M+1 for both LRU, LFU
  - Stationary requests for LRU
  - SNM for LFU
- *LFU resembles "Follow the Leader" (y that maximizes past sum rewards)*
  - FtL is regretful in Online Linear Optimization problems

S. Shalev-Schwarz, Online learning and online convex optimization, Foundations and Trends in ML, 2011

Regret lower bounds

- Convex functions on ball: $\Omega(\sqrt{T})$ [Shalev-S12]
- Strong convexity: $\Omega(\log T)$ [Hazan14]
- Our problem has a restricted adversary (due to shape of requests)

Theorem (Regret lower bound for caching) [Paschos19]:

There exist sequences R_t for which every caching policy has regret

$$Reg \geq w\sqrt{\frac{\gamma}{\pi}}\sqrt{MT}$$

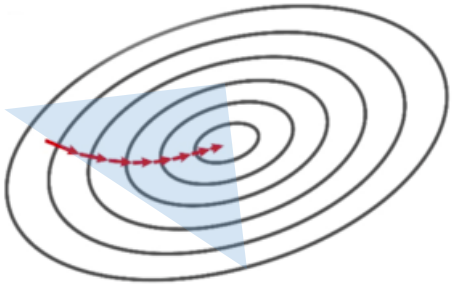
as $T \to \infty$.

This is the fastest rate at which any caching policy can learn

G. Paschos et al, Learning to cache with no regrets, INFOCOM, 2019.
S. Shalev, Online learning and online convex optimization, Foundations and Trends in ML, 2011
E. Hazan et al, Beyond the regret minimization barrier:…, PMLR, 2011.

Zinkevich Gradient (OGA)

- Follow the direction of previous gradient:
  - Cache update: $y_{t+1} = \Pi_{\mathcal{Y}}(y_t + \eta \nabla U_t(y_t))$
  - FtL with Euclidean regularization



Theorem (OGA Regret) [Paschos19]:

The Online Gradient Ascent caching policy satisfies:

$$Reg(OGA) \leq w\sqrt{2}\sqrt{MT}$$

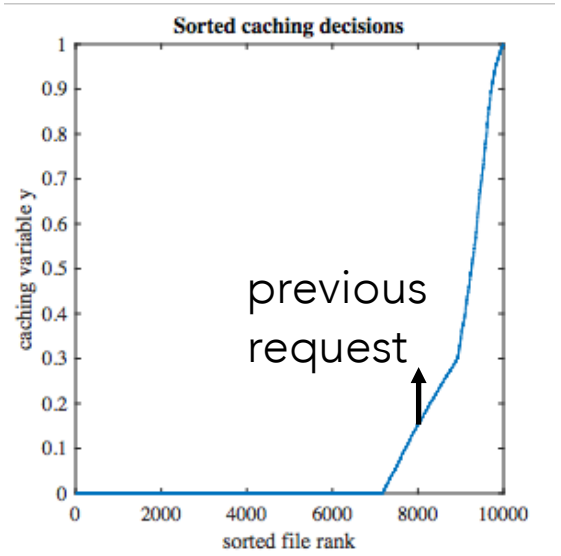
Zinkevich gradient is optimal to within $2\sqrt{\pi}$ [Bhattacharjee20]

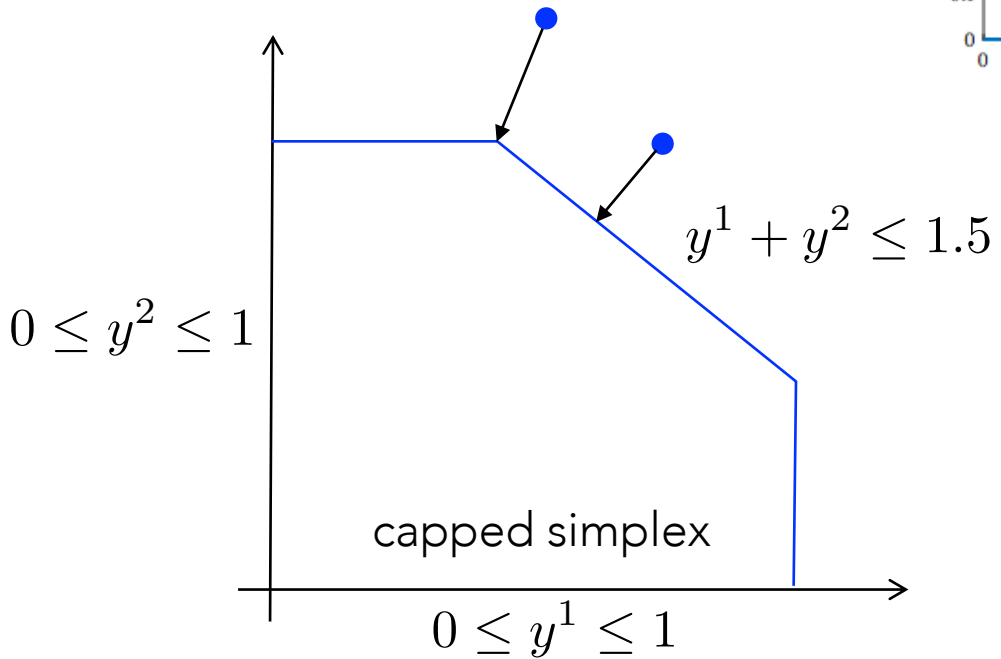
[Zinkevich03] "Online convex programming and generalized infinitesimal gradient descent"

[Bhattacharjee20] "Fundamental limits of online network-caching"

No regret cache eviction

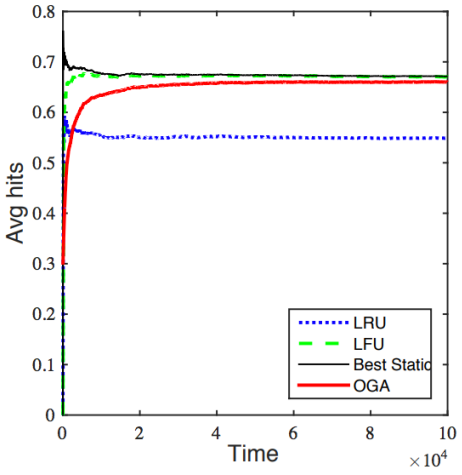
- Observe previous request $R_t^n = 1$
- Compute gradient: $\nabla(U_t) = (0, \ldots, w^n, 0, \ldots, 0)$
- Increase caching of content n: $y_{t+1}^n = y_t^n + \eta w^n$
- All other contents remain same $y_{t+1}^k = y_t^k$
- Euclidean projection on capped simplex



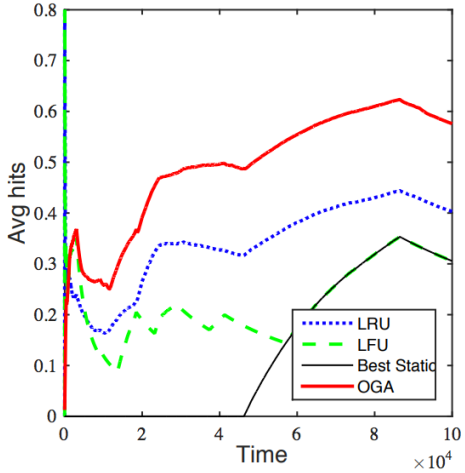


Universality of OGA

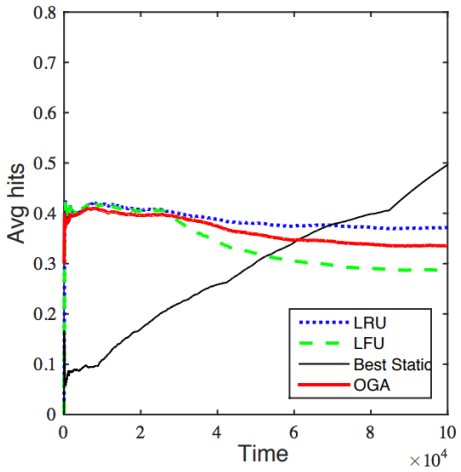
IRM



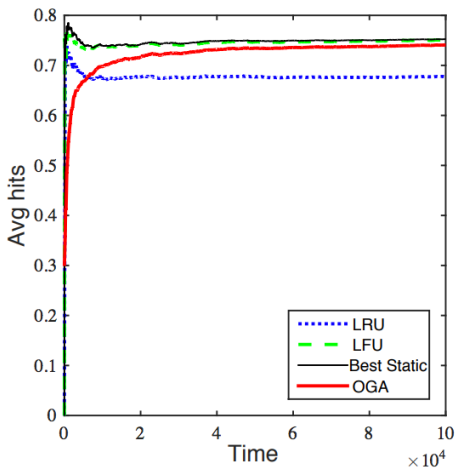
SNM



Youtube dataset



mixture



Parts I-II: Takeaway messages

- Learning framework for universal cache eviction
- Online gradient: optimal regret for one cache
- Both extend to networks of caches (see Part II)

QUESTIONS?

CONNECT
Centre for Future Networks

Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

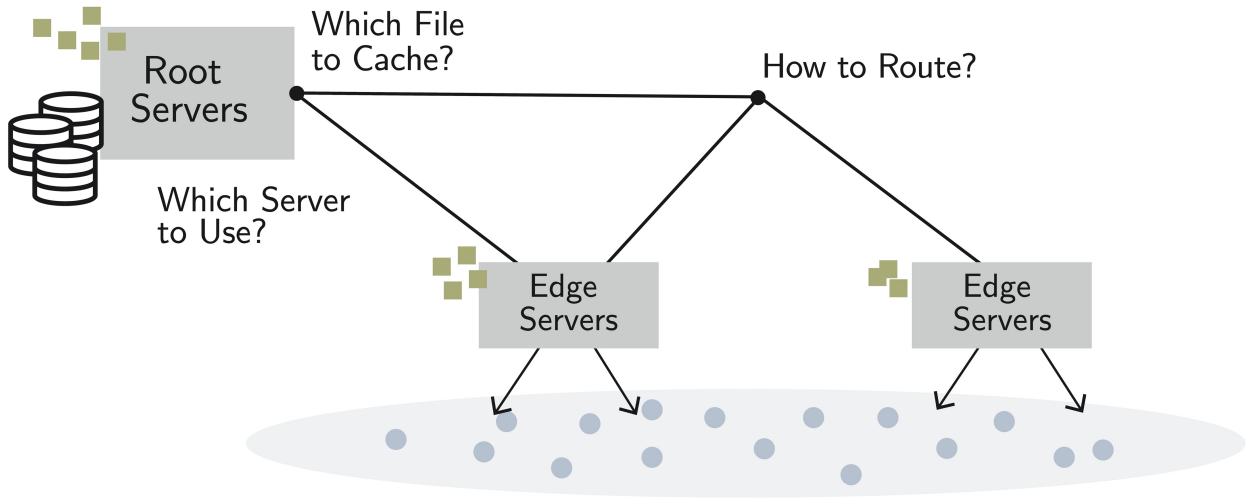
Tutorial 1: Online Learning for Data Caching and Network Service Delivery

Part III: Optimal Placement in Networks

Georgios Paschos (Amazon, Luxembourg)
George Iosifidis (Trinity College Dublin)
Apostolos Destounis (Huawei, Paris)

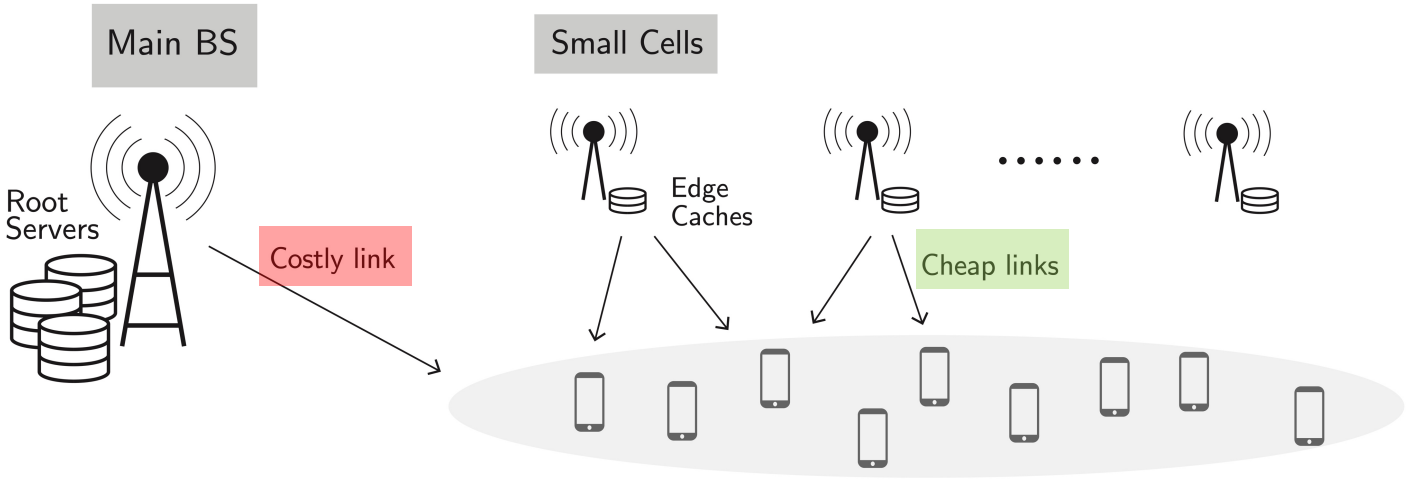
ACM ICN, 2020

Caching Decisions in Networks



- Cache Deployment: where to deploy caches? how large should they be?
- Caching Files: which files each cache will store?
- Content Routing: how to route the files from caches to requesters?

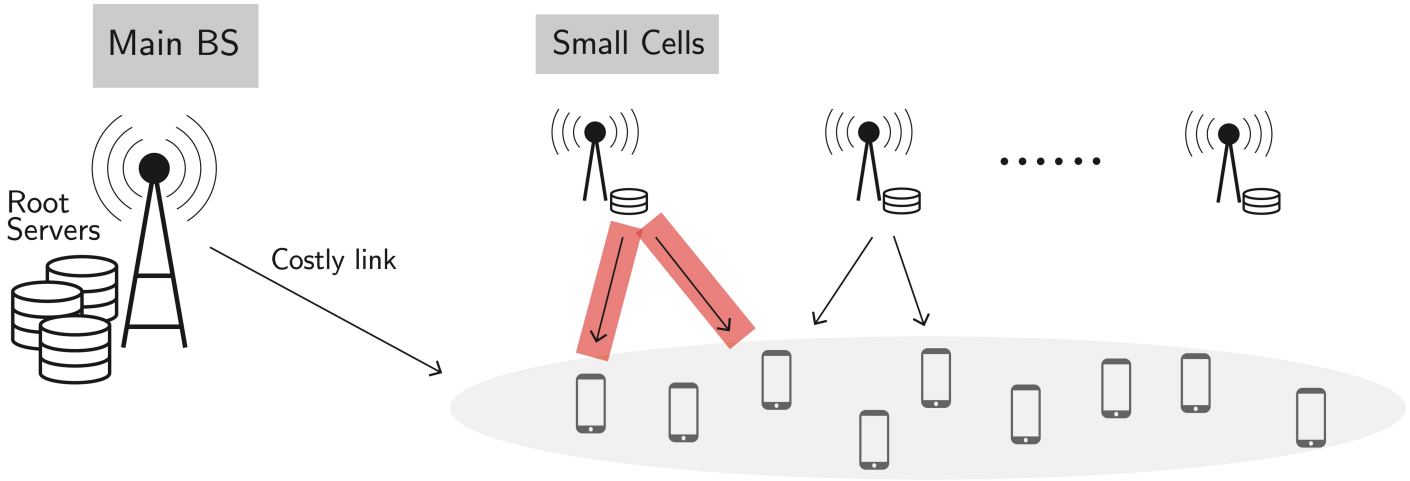
Focus on Wireless Edge Caching



- Femtocaching: proactive caching at storage-endowed small cells
- Gains: increases performance for users; reduces costs for networks
- Challenges: multiple paths, wireless effects, moving users, etc.

N. Golrezaei, et al. FemtoCaching: Wireless video content delivery through distributed caching helpers, INFOCOM' 12

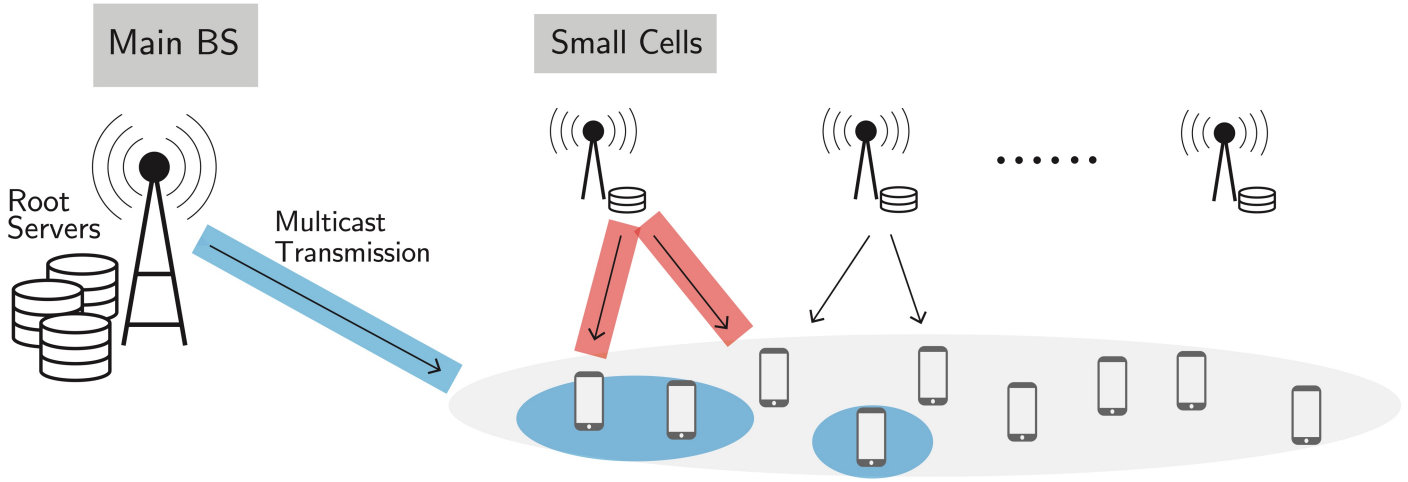
Focus on Wireless Edge Caching



- Capacitated Femtocaching: massive demand; limited wireless capacity
- Challenges: more "hard" constraints; routing is compounded

K. Poularakis, et al. Approximation Algorithms for Mobile Data Caching in Small Cell Networks, IEEE Trans. Comm.' 14

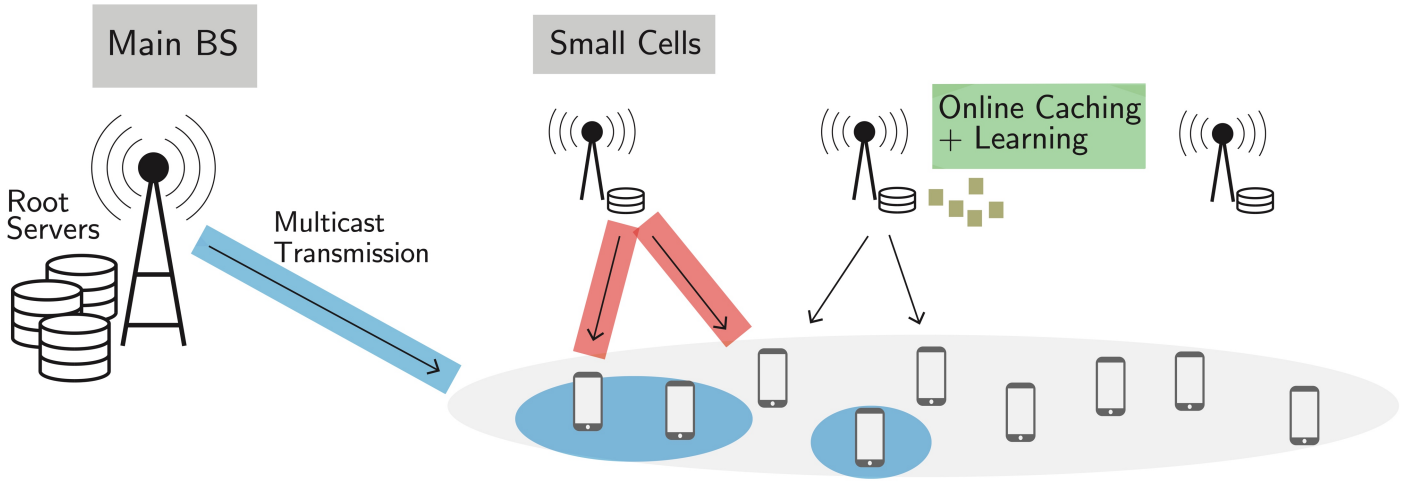
Focus on Wireless Edge Caching



- Multicast + Caching: two sides of the same coin
- Multicast: concurrency in time: users request the same file at same time
- Caching: concurrency in space: co-located users request the same file

K. Poularakis, et al. Exploiting Caching and Multicast for 5G Wireless Networks, IEEE Trans. Wirel. Comm.' 16

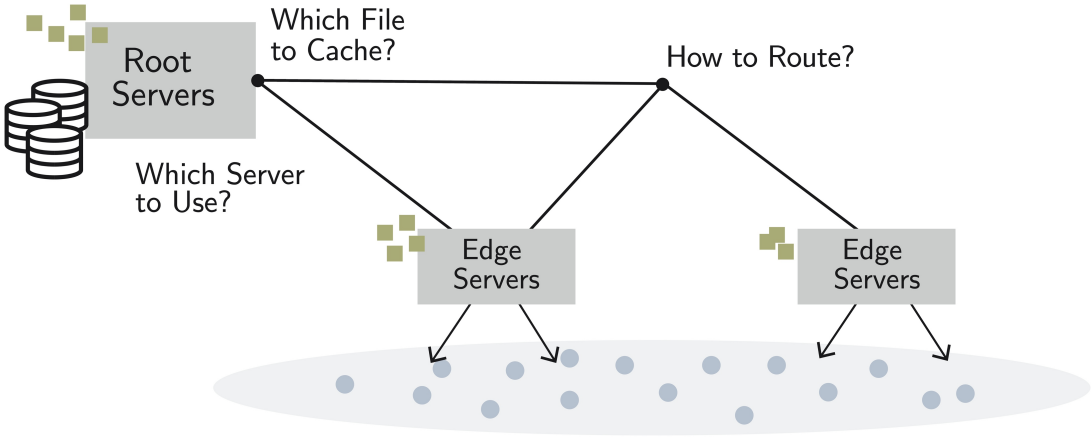
Focus on Wireless Edge Caching



- Previous models: "populate-during-night" policy; known demand
- Online Femtocaching: update caches dynamically by adapting to requests
- Learning: how to cache & route optimally, i.e., as if we knew the future!

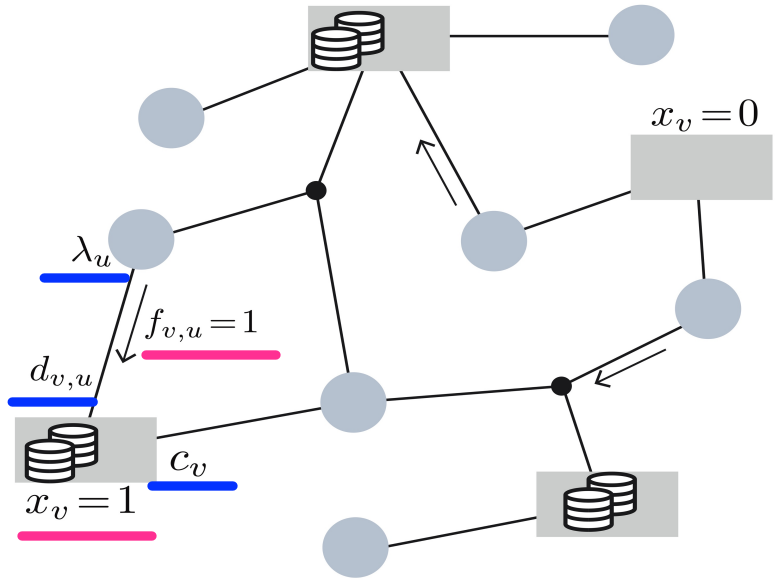
G. Paschos, et al. Online Convex Optimization for Caching Networks, IEEE/ACM Trans. on Networks, 2020

This Tutorial is not only about Caching



- NFV/Middleboxes: deploying gateways, in-network servers, etc.
- Edge Computing: dimensioning edge servers; assigning users to servers
- SDN architectures: deploying SDN controllers
- ML/Code Libraries: caching code to support services, e.g., AI-assisted driving
- But: each problem has its own specifics and challenges!

The Cache Deployment Problem



- Parameters: network graph; demand; routing costs; storage costs
- Decisions: cache placement/dimensioning; demand–cache assignment

$$\mathcal{X}_{D1} = \left\{ \boldsymbol{x} \in \{0,1\}^V \right\}, \; \mathcal{F}_{D1} = \left\{ \boldsymbol{f} \in \{0,1\}^{V \times V} \;\middle|\; \begin{array}{l} \sum_{v \in \mathcal{V}} f_{v,u} = 1, \quad u \in \mathcal{V} \\ f_{v,u} = 0 \text{ if } \mathcal{P}_{v,u} = \emptyset \end{array} \right\}$$

The Cache Deployment Problem

Deployment (D1)

$$\min_{\boldsymbol{x}\in\mathcal{X}_{D1},\ \boldsymbol{f}\in\mathcal{F}_{D1}} \quad J_{D1}(\boldsymbol{x},\boldsymbol{f}) \triangleq \sum_{v\in V} x_v c_v + \sum_{v,u\in\mathcal{V}} f_{v,u}\lambda_u d_{v,u}$$

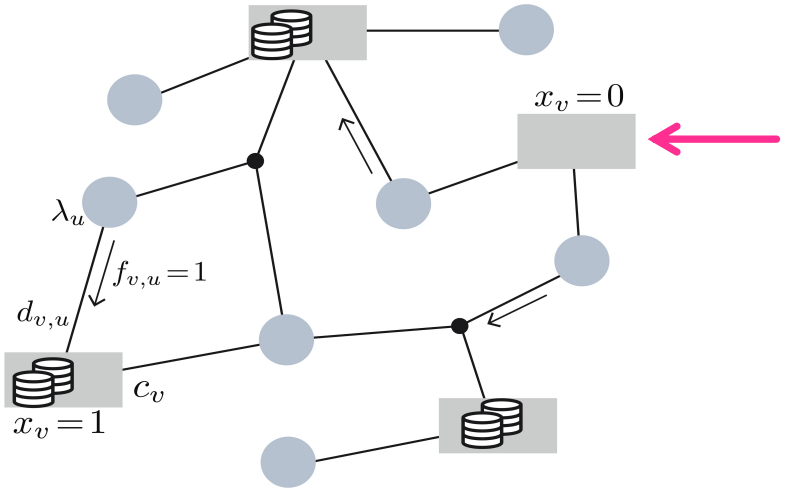
$$\text{s.t.} \qquad f_{v,u} \leq x_v, \quad v,u\in\mathcal{V}.$$

$$\mathcal{X}_{D1} = \left\{\boldsymbol{x}\in\{0,1\}^V\right\},\ \mathcal{F}_{D1} = \left\{\boldsymbol{f}\in\{0,1\}^{V\times V} \,\middle|\, \begin{array}{l} \sum_{v\in\mathcal{V}} f_{v,u} = 1, \quad u\in\mathcal{V} \\ f_{v,u} = 0 \text{ if } \mathcal{P}_{v,u} = \emptyset \end{array}\right\}$$

- A discrete optimization problem
- How difficult is to find the optimal solution?
- NP-Hard; but we can rely on the theory of Facility Location

Facility Location Problems

- *k-center*. Given a set $\mathcal{V}$ of locations and distances $\mathcal{P}$, open k facilities and select the demand-facility assignment $\mathcal{F}$, to minimize the maximum distance $P_{v,u}$ for any pair (v, u) with $f_{v,u} = 1$.
- *k-median*. Given a set $\mathcal{V}$ of locations and distances $\mathcal{P}$, open k facilities and select the demand-facility assignment $\mathcal{F}$, to minimize the aggregate distance $\sum_{v,u} P_{v,u}$ for all (v, u) pairs with $f_{v,u} = 1$.
- *Uncapacitated fixed-charge*. Given a set $\mathcal{V}$ of locations, distances $\mathcal{P}$, and deployment costs $\boldsymbol{c}$, decide how many and which facilities to open, and select the assignment $\mathcal{F}$, to minimize the aggregate distance and deployment costs.



The Cache Deployment Problem

Deployment (D1)

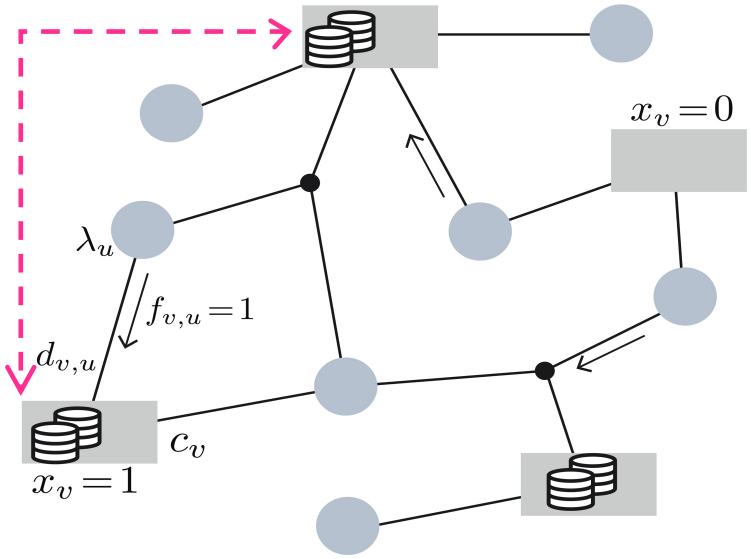
$$\min_{\boldsymbol{x}\in\mathcal{X}_{D1},\ \boldsymbol{f}\in\mathcal{F}_{D1}} \quad J_{D1}(\boldsymbol{x},\boldsymbol{f}) \triangleq \sum_{v\in V} x_v c_v + \sum_{v,u\in\mathcal{V}} f_{v,u}\lambda_u d_{v,u}$$

$$\text{s.t.} \qquad f_{v,u} \leq x_v, \quad v,u \in \mathcal{V}.$$

- Best known algorithm achieves 1.488-approximation ratio
- I.e., at most 1.488-times higher cost than the minimum of (D1)
- Different versions of (D1) admit different solutions – check FL results

(i) S. Li, "A 1.488 Approximation Algorithm for the Uncapacitated Facility Location Problem". Inf. & Comp. 2013
(ii) D. Shmoys, et al, "Approximation Algorithms for Facility Location Problems", ACM STOC, 1997

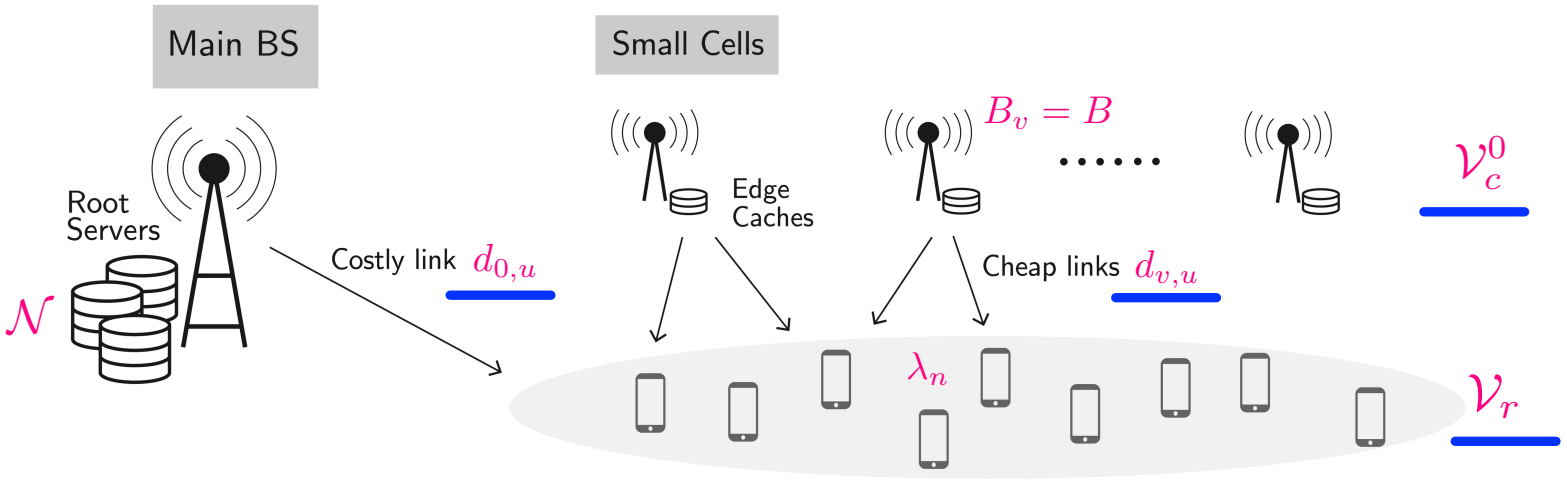
A Twist: Deploying Synching Services



- Synching: the different servers need to synchronize; e.g., to ensure consistent service state (code; libraries; updates, etc.)
- Trade off: placing servers close to demand? or close to each other?
- Solution: model it as a co-located FL problem (Co-FLP)

C. Swamy et al., Primal-dual Algorithms for Connected Facility Location Problems, Algorithmica, 40(4), 2004

Wireless Edge Caching (WEC)



- Which files to cache at each SC, in order to maximize the delay savings?
- No storage cost; SCs have equal capacity but different delay; Main BS stores all files; file popularity is the same across all users
- Decide only the caching: $\mathcal{Y}_{C1} = \left\{ \boldsymbol{y} \in \{0,1\}^{V_c^0 \times N} \;\middle|\; \sum_{n \in \mathcal{N}} y_{v,n} \leq B,\ v \in \mathcal{V}_c^0 \right\}$

N. Golrezaei, et al. FemtoCaching: Wireless video content delivery through distributed caching helpers, INFOCOM' 12

WEC – Problem Formulation

Uncapacitated Caching (C1)

$$\max_{\boldsymbol{y}\in\mathcal{Y}_{C1}} \quad J_{C1}(\boldsymbol{y}) \triangleq \sum_{u\in\mathcal{V}_r} \Big(d_{0,u} - D_u(\boldsymbol{y})\Big)$$

$$D_u(\boldsymbol{y}) = \sum_{n\in\mathcal{M}_1} \lambda_n d_{1,u} + \sum_{n\in\mathcal{M}_2\setminus\mathcal{M}_1} \lambda_n d_{2,u} + \ldots + \sum_{n\notin\{\mathcal{M}_j\}_{j=1}^{V_c^0}} \lambda_n d_{0,u}$$

$$\mathcal{Y}_{C1} = \left\{ \boldsymbol{y} \in \{0,1\}^{V_c^0\times N} \;\middle|\; \sum_{n\in\mathcal{N}} y_{v,n} \le B,\ v\in\mathcal{V}_c^0 \right\}$$

- Bad news: NP-complete problem
- Good news: but we can use a simple greedy algorithm with ½ approx. ratio!

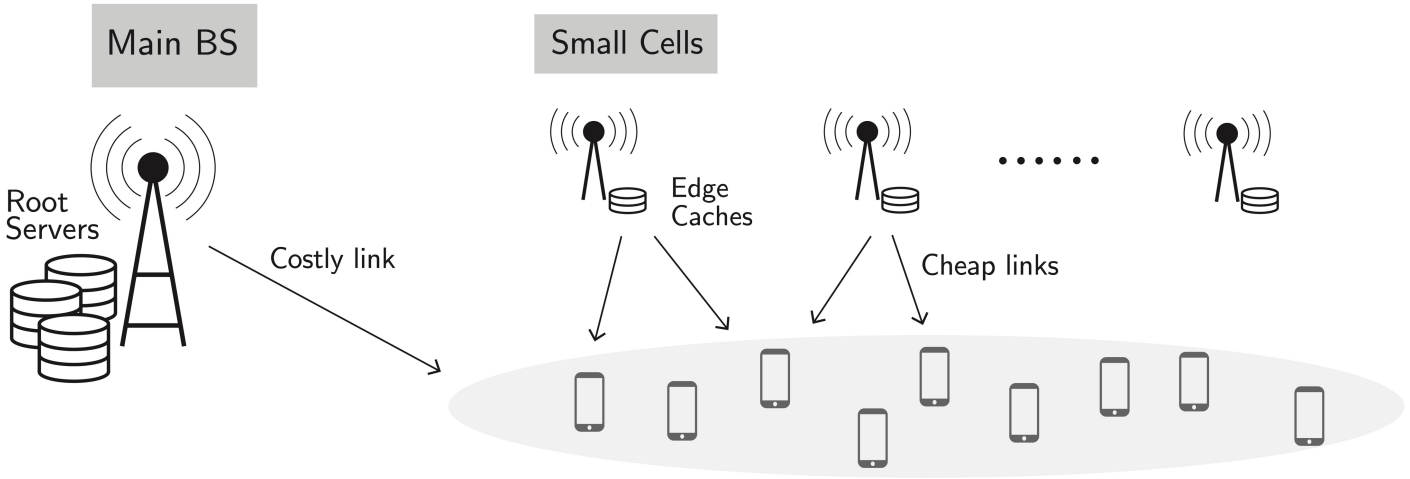
WEC - Greedy Algorithm

$$S = \left\{ s_1^1, s_2^1, \ldots, s_N^1; \ \ldots; \ s_1^{V_c^0}, s_2^{V_c^0}, \ldots, s_N^{V_c^0} \right\} \qquad S_v = \{ s_1^v, s_2^v, \ldots, s_N^v \}$$

$$\mathcal{I} = \left\{ Y \subseteq S \;\middle|\; |Y \cap S_v| \leq B, \ \ v \in \mathcal{V}_c \right\} \qquad J(Y) = \sum_{u \in \mathcal{V}_r} \left(d_{0,u} - D_u(Y) \right)$$

Initialize: $Z \leftarrow S$; $Y \leftarrow \emptyset$; $Z_v \leftarrow S_v$, $Y_v \leftarrow \emptyset$, $\forall\, v = 1, \ldots, V_c$
for $i = 1, 2, \ldots, B \cdot V_c$ **do**
  Find the placement maximizing the marginal utility:
  $s_{n^*}^{v^*} = \arg\max_{z \in Z} J_Y(z) \triangleq J\big(Y \cup \{z\}\big) - J\big(Y\big)$
  if $J_Y\left(s_{n^*}^{v^*}\right) = 0$ **then**
    Terminate
  end
  $Y_{v^*} \leftarrow Y_{v^*} \cup \{ s_{n^*}^{v^*} \}$ %Update the cached files of cache v
  $Y \leftarrow Y \cup \{ s_{n^*}^{v^*} \}$ %Update the caching policy
  $Z \leftarrow Z \setminus \{ s_{n^*}^{v^*} \}$
  $Z_{v^*} \leftarrow Z_{v^*} \setminus \{ s_{n^*}^{v^*} \}$
  if $|Y_{v^*}| = B$ **then**
    $Z \leftarrow Z \setminus Z_{v^*}$ %Remove elements of caches with no space
  end
end
Output: optimal content placement Y

WEC Extensions



- Capacitated Femtocaching: we use facility location theory
- Hierarchical Caching: exploit the structure of routing costs; trade off: caching higher serves more requests, but caching lower serves cheaper
- General Network Models: routing delays change with the load; objective functions are non-linear (e.g., energy costs); etc.

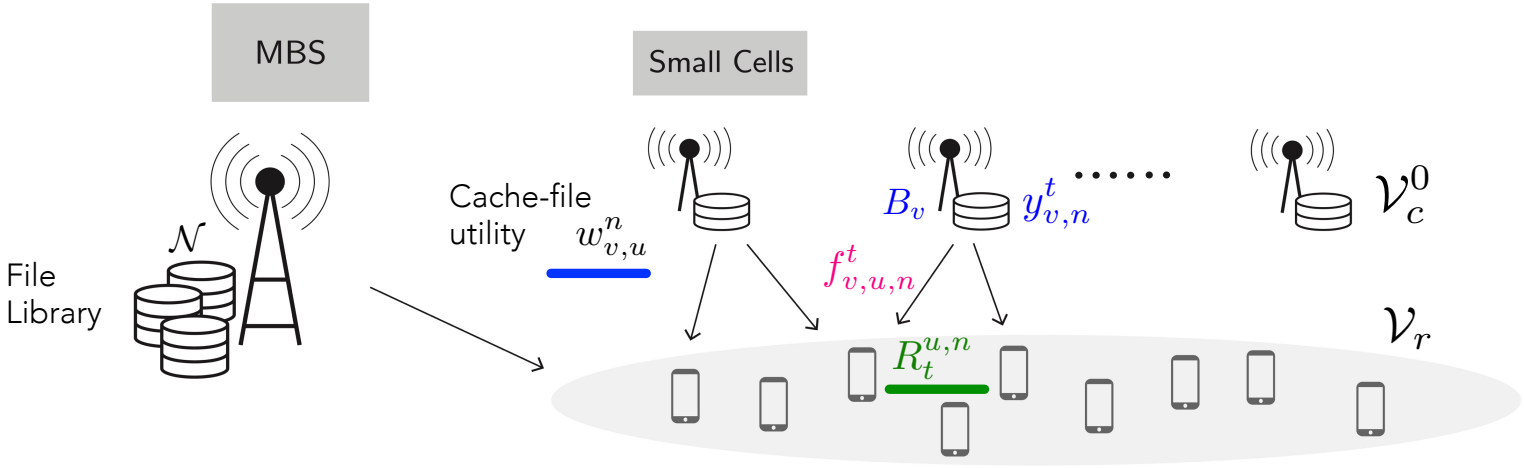
Paschos, Iosifidis, Caire, Cache Optimization Models & Algorithms, Foundations & Trends in Comm. & Inf. Theory, 2020

Online Learning for WEC

- What happens when we do not know the expected demand?
- When we do not even know the model that generates the requests?
- We can leverage the OCO theory to build a robust policy:
  - No need to assume anything about the file popularity;
  - No need to assume anything about the request intensity;
  - Low-complexity reactive (not proactive) caching-routing policy
- Powerful framework, with many applications/extensions
  - Dynamic networks: moving caches (e.g. vehicular netw.), changing links, etc.
  - Cache update costs; important for service/code deployment
  - General performance criteria, beyond cache hit ratio

Paschos, Iosifidis, Destounis, Online Convex Optimization for Caching Networks, IEEE/ACM Trans. on Networks, 2020

Learning-WEC: Model



- Request model:

$$\mathcal{R}_{C3} = \left\{ \boldsymbol{R} \in \{0,1\}^{N \times V_r} \;\middle|\; \sum_{n \in \mathcal{N}} \sum_{u \in \mathcal{V}_r} R^{u,n} = 1 \right\}$$

- Caching decisions:

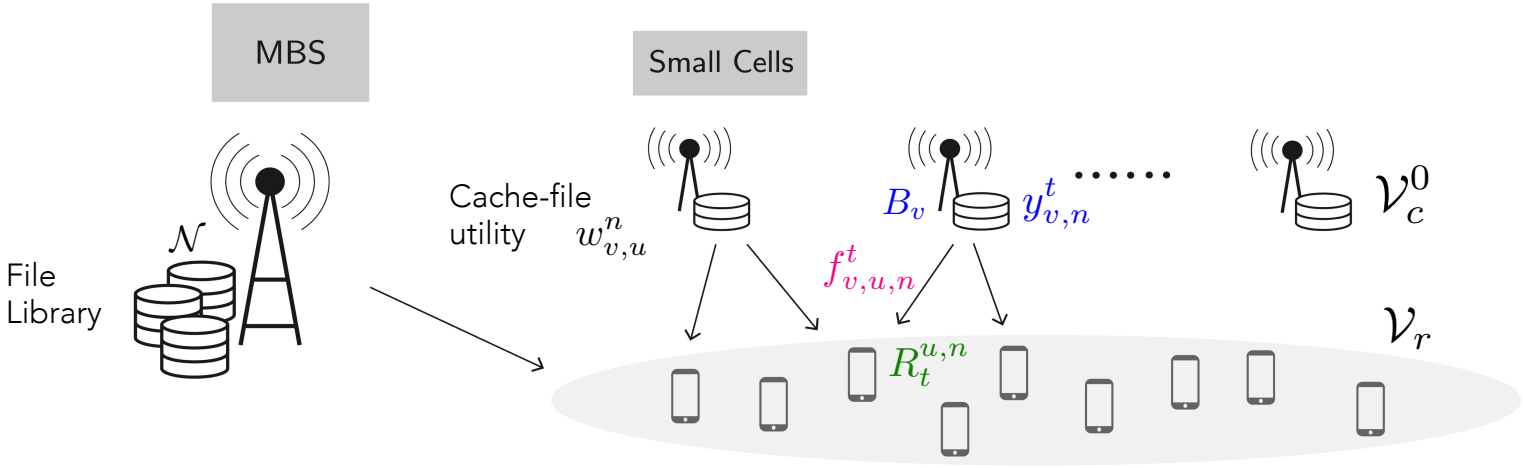
$$\mathcal{Y}_{C3} = \left\{ \boldsymbol{y} \in [0,1]^{N \times V_c} \;\middle|\; \sum_{n \in \mathcal{N}} y_{v,n} \leq B_v, \; v \in \mathcal{V}^0_c \right\}$$

- Routing decisions

$$\mathcal{F}_{C3}(\boldsymbol{y}_t) = \left\{ \boldsymbol{f} \in [0,1]^{N \times V_c \times V_r} \;\middle|\; \begin{array}{l} \sum_{v \in \mathcal{V}_c} f_{v,u,n} = R^{u,n}_t, \; u \in \mathcal{V}_r, \; n \in \mathcal{N} \\ f_{v,u,n} \leq y_{v,n}, \; v \in \mathcal{V}^0_c, \; u \in \mathcal{V}_r, \; n \in \mathcal{N} \\ f_{v,u,n} = 0, \; u \in \mathcal{V}_r, v \in \mathcal{V}_c \setminus \mathcal{V}_c(u), \; n \in \mathcal{N} \end{array} \right\}$$

Note: we follow the coded caching or chunk-based caching model, where decisions are continuous variables

Learning-WEC: Model



- Utility function: $J_t(\boldsymbol{y}_t) = \max_{\boldsymbol{f} \in \mathcal{F}(\boldsymbol{y}_t)} \sum_{n \in \mathcal{N}} \sum_{u \in \mathcal{V}_r} \sum_{v \in \mathcal{V}_c} w^n_{v,u} R^{u,n}_t f^t_{v,u,n}$
  - For each caching vector, we find the optimal routing vector
- Online caching Policy: $\sigma : (\boldsymbol{R}_1, \boldsymbol{R}_2, \ldots, \boldsymbol{R}_{t-1}, \boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_{t-1}) \longrightarrow \boldsymbol{y}_t \in \mathcal{Y}_{C3}$

A step back: Demystifying OCO

- OCO tackles problems with time-varying unknown objectives. How is it related to other optimization tools?

- (i) (Standard) Network Optimization
  - Use a gradient algorithm to solve the dual
  - Apply the solution *after convergence*

$$\max_{y \in \mathcal{Y}} J(y) \qquad x^* \to y^* \to J(y^*)$$

$$\min_{x \in \mathcal{X}} H(x) \qquad x_{t+1} = x_t - a_t \nabla H(x_t)$$

- (ii) Primal Averaging
  - As (i), but we apply the solution *in each slot*

$$\max_{y \in \mathcal{Y}} J(y) \qquad \frac{1}{T} \sum_{t=1}^{T} J(y_t)$$

$$\min_{x \in \mathcal{X}} H(x) \qquad x_t \to y_t \to J(y_t)$$

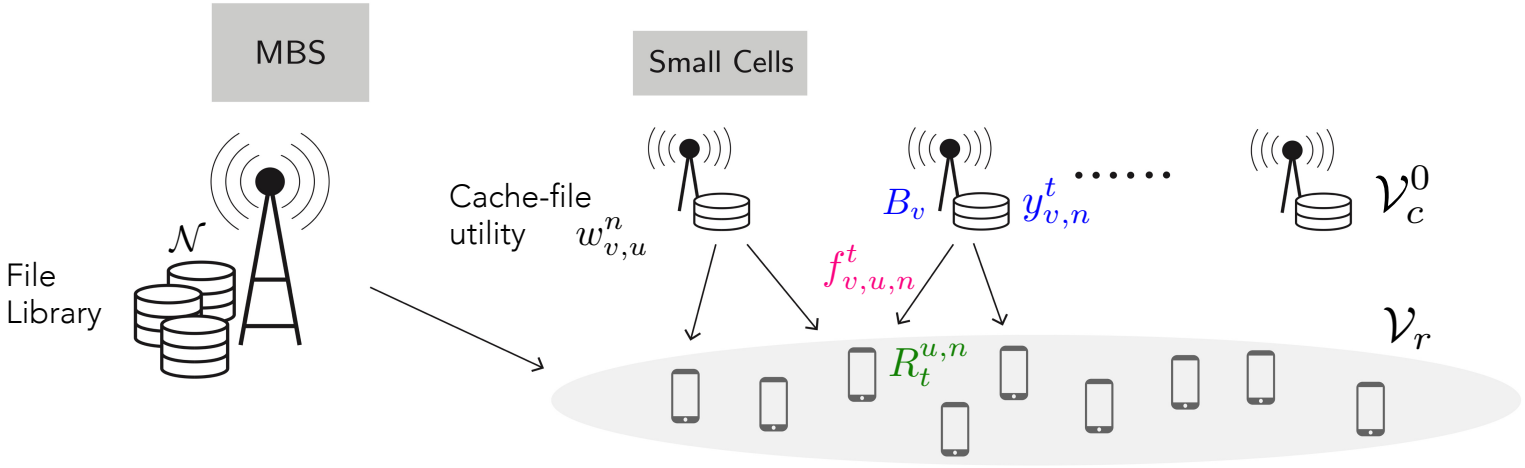
- (iii) OCO
  - As (ii), but the objective *changes in each slot*

$$\max_{y \in \mathcal{Y}} J_t(y) \qquad \frac{1}{T} \sum_{t=1}^{T} J_t(y_t) - J_t(y^*)$$

$$\min_{x \in \mathcal{X}} H_t(x) \qquad x_t \to y_t \to J_t(y_t)$$

(i) M. Chiang, et al, A Mathematical Theory of Network Architectures, Proceedings of IEEE, 2007
(ii) A. Nedic, et al, Approximate Primal Solutions & Rate Analysis for Dual Subgrad. Methods. SIAM Jrn. on Optim. 2009
(iii) E. Hazan, Introduction to Online Convex Optimization, Foundations and Trends in Optimization, 2016

Learning-WEC: Solution



- Goal: achieve sublinear regret

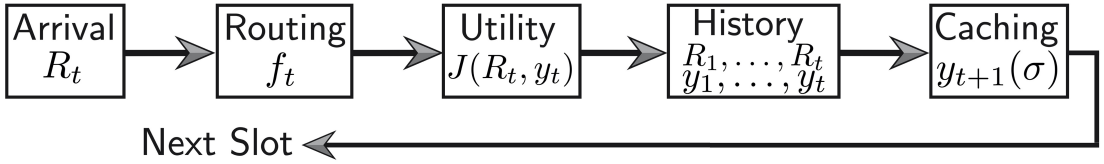
$$\mathtt{Reg}_T(\sigma) = \max_{P(\boldsymbol{R}_1,\ldots,\boldsymbol{R}_T)} \mathbb{E}\left[\sum_{t=1}^{T} J_t(\boldsymbol{y}^*) - \sum_{t=1}^{T} J_t(\boldsymbol{y}_t(\sigma))\right] \qquad \boldsymbol{y}^* \in \arg\max_{\boldsymbol{y}\in\mathcal{Y}} \sum_{t=1}^{T} J_t(\boldsymbol{y})$$

Online Femtocaching (C3)

Find a policy σ that satisfies:

$$\mathtt{Reg}_T(\sigma) = o(T).$$

Bipartite Subgradient Caching Algorithm



Input: $\mathcal{E}$; $\mathcal{N}$; $\{w^n_{v,u}\}_{(v,u,n)}$; $\eta_t = \Delta y / K\sqrt{T}$.

Initialize: $\hat{n}, \hat{u}$, $\boldsymbol{y}_1$ arbitrarily

for $t = 1, 2, \ldots$ **do**

  Receive $\boldsymbol{R}_t$ and set $n = \hat{n}, u = \hat{u}$: $R_t^{\hat{u},\hat{n}} = 1$

  Find $\boldsymbol{f}_t$

  Calculate utility $J_t(\boldsymbol{y}_t)$

  Calculate the supergradient $\boldsymbol{g}_t$

  $\boldsymbol{q}_{t+1} = \boldsymbol{y}_t + \eta_t \boldsymbol{g}_t$

  $\boldsymbol{y}_{t+1} = \Pi_{\mathcal{Y}}(\boldsymbol{q}_{t+1})$

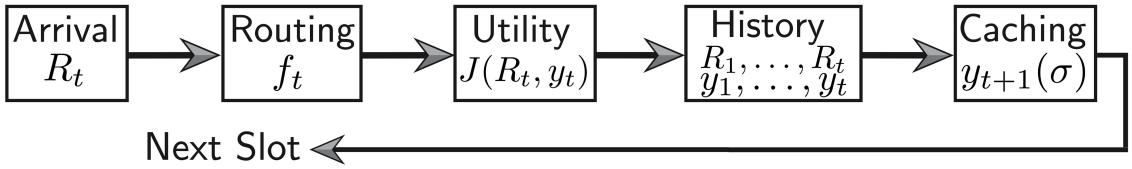
end

Output: $\{\boldsymbol{y}_t\}_{t=1}^T$

$$J_t(\boldsymbol{y}_t) = \max_{\boldsymbol{f} \in \mathcal{F}(\boldsymbol{y}_t)} \sum_{n \in \mathcal{N}} \sum_{u \in \mathcal{V}_r} \sum_{v \in \mathcal{V}_c} w^n_{v,u} R_t^{u,n} f^t_{v,u,n}$$

Paschos, Iosifidis, Destounis, Online Convex Optimization for Caching Networks, IEEE/ACM Trans. on Networks, 2020

Bipartite Subgradient Caching Algorithm



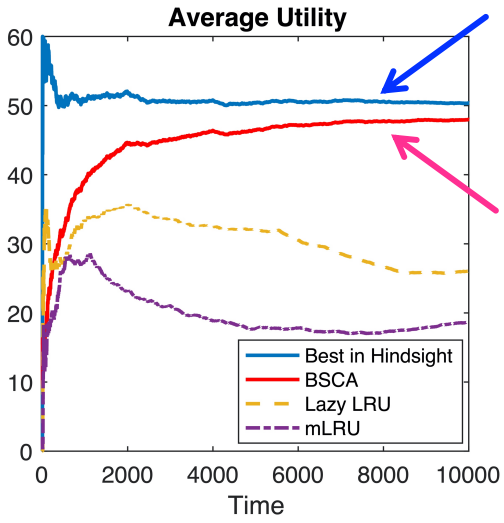
- The BSCA algorithm learns to perform as good as the optimal static policy (hindsight policy); and its learning rate is sqrt (T).

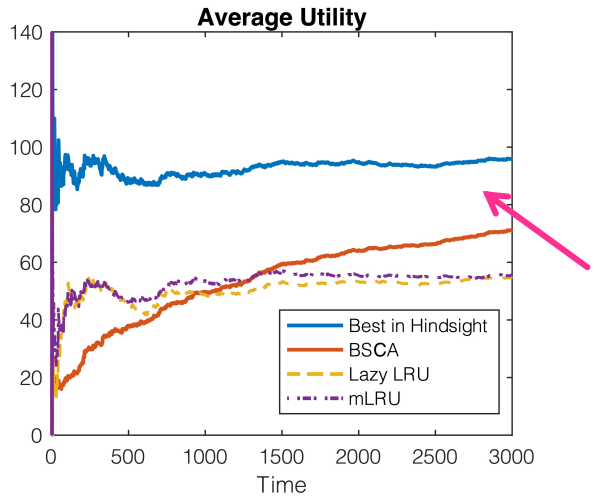
$$Reg_T(\mathrm{BSCA}) \leq w^{(1)} \sqrt{2 \mathrm{deg} V_c B T}, \quad B = \max_v B_v$$

- Where the bound parameters are independent of the file library size N.

Experiments with Traces

- The BSCA algorithm learns to perform as good as the optimal static policy (hindsight policy); and its learning rate is sqrt (T).



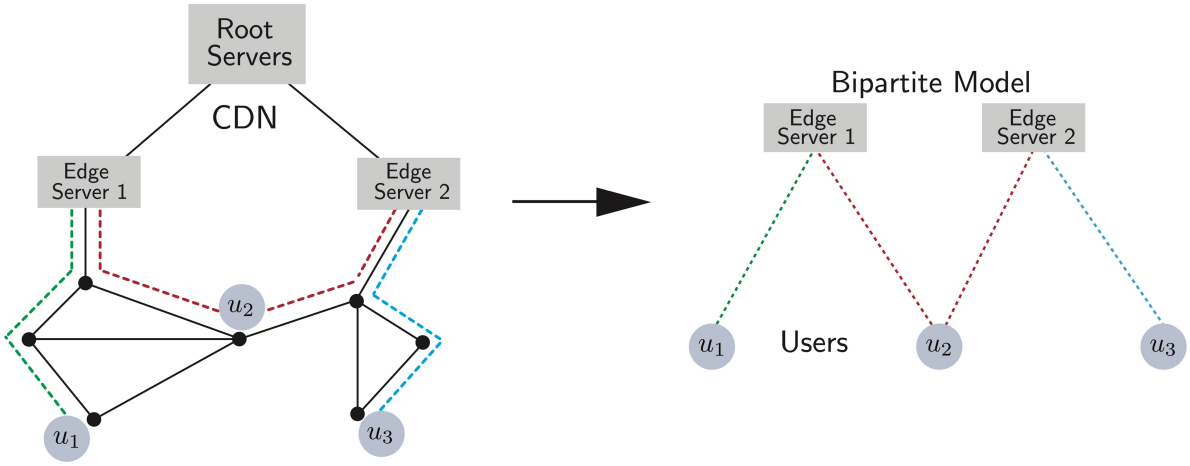


Lazy LRU: Leonardi, Neglia, Implicit coordination of caches in small cell networks under unknown popularity profiles," IEEE JSAC, 36 (6), 2018

mLRU: Giovanidis, Avranas, Spatial multi-LRU: Distributed caching for wireless networks with coverage overlaps arXiv:1612.04363, 2016

Beyond: 1-hop Networks

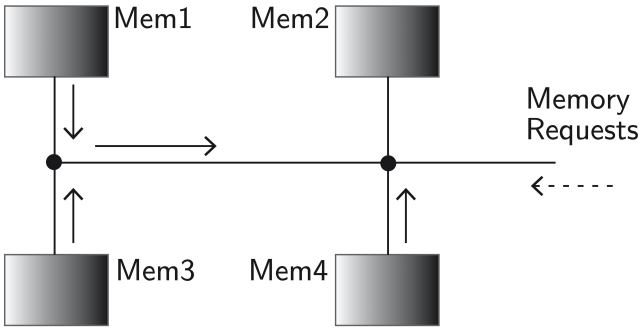
- A large range of general networks can be modeled with a bipartite graph.
  - Non-capacitated links;
  - and non-congestible links, i.e., load-independent routing costs.



- BSCA can be directly applied to these networks as well.

Beyond: Wireless Caching

- Bipartite connected caches appear in a range of various systems
  - CDNs, where the "edge" is identified at the level of city, or city-area.
  - Connected memory segments, as in emerging disaggregated data centers.

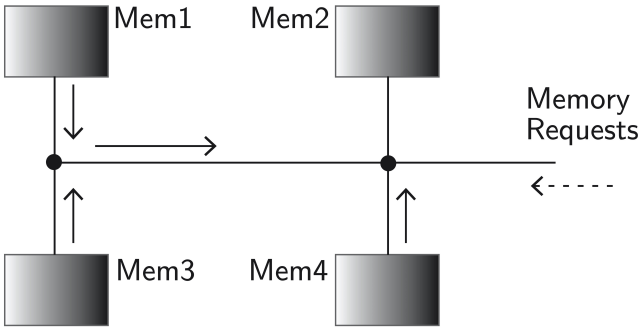


- BSCA can be used to optimize the operation of these systems.

M. Bielski et al, dReDBox: Materializing a full-stack rack-scale system prototype of a next-generation disaggregated datacenter, in Proc. of DATE, 2018.

Beyond: Caching

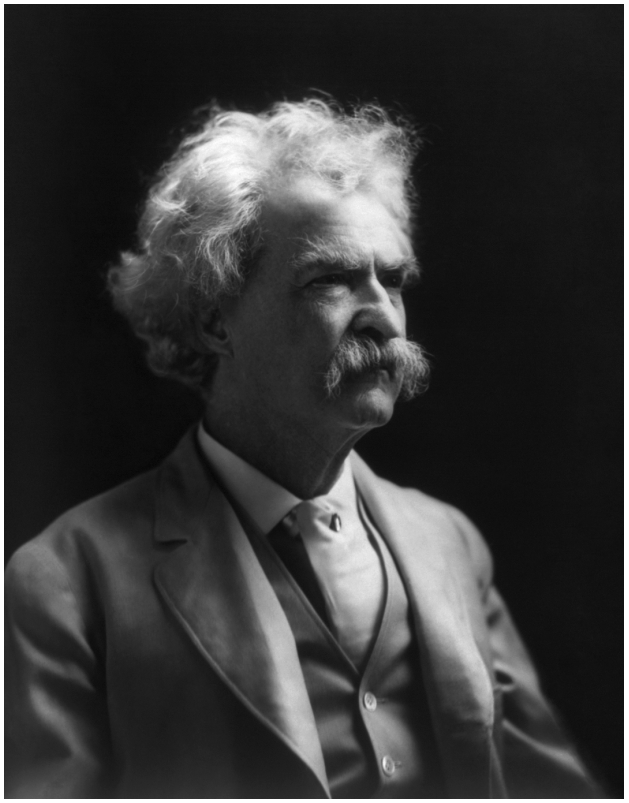
- What if there is cost to update the caches? E.g., if we store large ML libraries?
  - We need to balance utility improvements with prefetching costs.



- What if the caching decisions need to be discrete?

Paschos, Iosifidis, Destounis, Online Convex Optimization for Caching Networks, IEEE/ACM Trans. on Networks, 2020

Questions?



The more you explain it, the more I don't understand it

Mark Twain

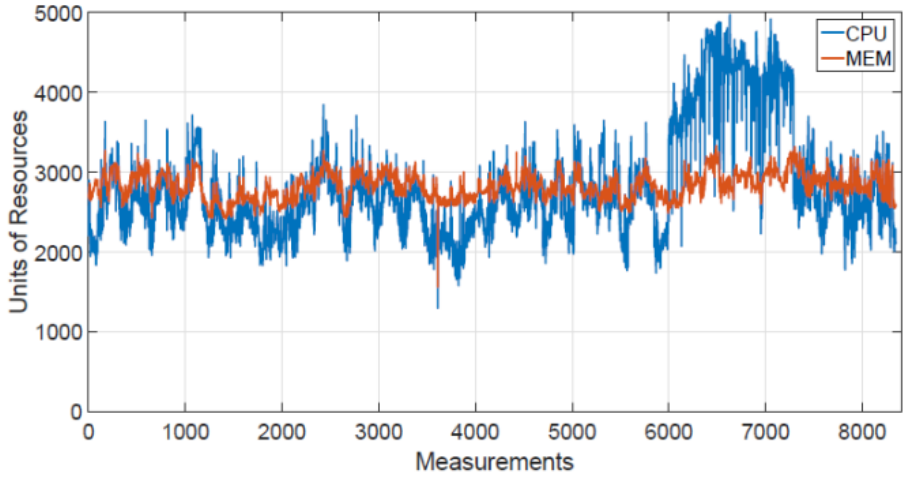
Tutorial 1: Online Learning for Data Caching and Network Service Delivery

Part IV: online learning for network service delivery

Georgios Paschos* (Amazon, Luxembourg)
George Iosifidis (Trinity College Dublin)
Apostolos Destounis (Huawei, Paris)

ACM ICN, 2020

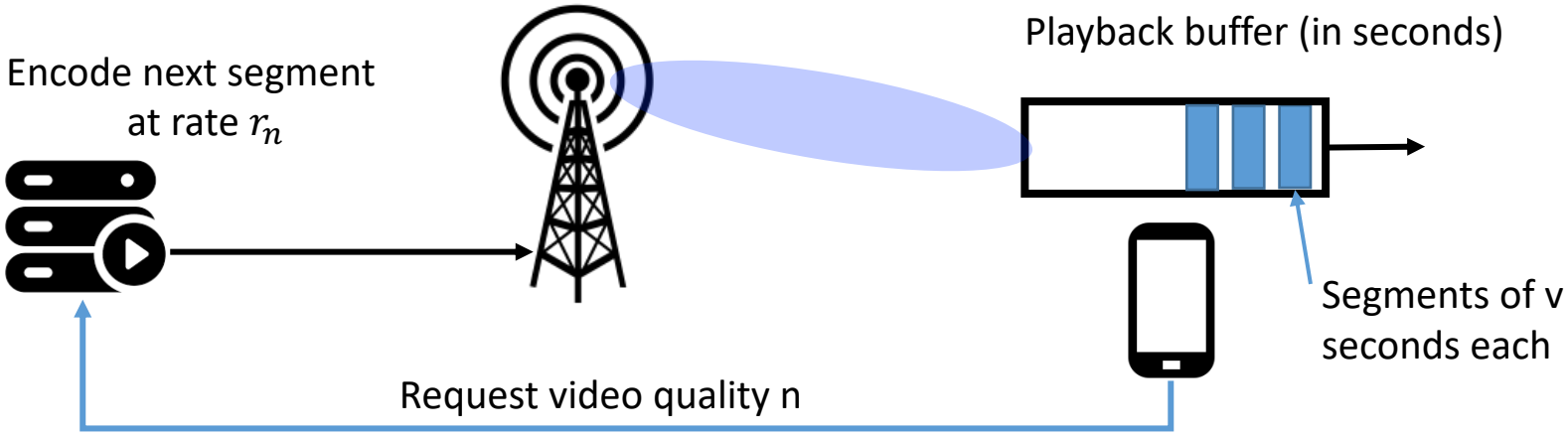
SLA provisioning in cloud services



- Reserve resources (e.g. CPU and memory) in the cloud for future incoming computation tasks.
- Goal: how to reserve just enough resources while guaranteeing an SLA in the form of not under-provisioning often.
- Challenge: Resource requests have a non-stationary behavior and unpredictable patterns

C. Reiss et al. "Heterogeneity and Dynamicity of Clouds at Scale: Google Trace Analysis," *in ACM Symposium on Cloud Computing (SoCC)*, San Jose, CA, USA, Oct. 2012

Mobile video streaming

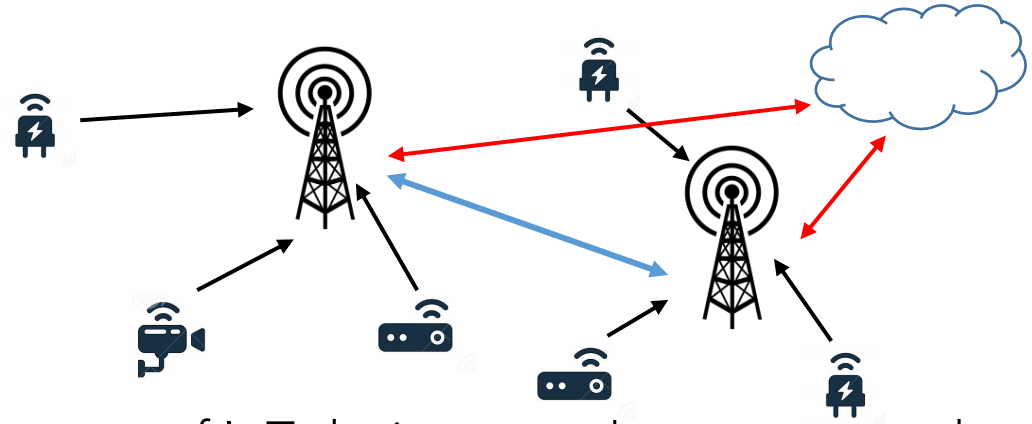


- HTTP adaptive streaming: the streaming client at the device requests the quality (encoding rate) for the next video segments.

- Goal: Maximize quality while avoiding buffer underflows (stalls) and overflows and not changing quality levels often.

- Challenges: Client works in the application layer, with no view of network condition
  - How fast the buffer fills will depend on congestion, channel states and other users in the network.
  - Duration of segments is a priori unknown to the client.

Radio resource allocation in IoT



- Large amount of IoT devices, need uncoordinated energy efficient ways for medium access.

- Goals:
  - Each IoT device must learn how to adapt to the environment without wasting energy when active.
  - Access points need to learn how to offload tasks from the IoT devices.

- Challenges:
  - IoT activation patterns and background interference can be unknown and unpredictable.

A. Marcastel et al. "Online power optimization in feedback-limited, dynamic and unpredictable IoT networks", *IEEE Transactions on Signal Processing*, 2019.

T, Chen et al., "Learning and management for Internet of Things: Accounting for adaptivity and scalability", *Proceedings of the IEEE,* 2019

Unifying theme

- Common aspect: Resources must be reserved but the environment (demands, service rates, …) is **unknown** and **unpredictable**.

- Online Learning: Treat the environment as if controlled by an adversary.
  - Performance measured in terms of **regret**: Compare with the best static assignment with knowledge of the future environment.

- Comparison to other relevant ML frameworks:
  - Predict the environment: Not feasible in many applications.
  - Reinforcement learning: Assumes stationary (Markovian) dynamics.

Online Learning with adversarial constraints

- Framework: At every round t
  - The agent selects an action x_t
  - The adversary selects a cost function $f_t(.)$and possibly a constraint function $g_t(.)$.

- Benchmark action: $x_* = arg\min_{x \in X} \sum_{t=1}^{T} f_t(x)$

  s.t. $\sum_{t=1}^{T} g_t(x) \leq 0$

  (Best action in hindsight)

- Regret: $R_*(\mathrm{T}) = \sum_{t=1}^{T} f_t(x_t) - \sum_{t=1}^{T} f_t(x_*)$

- Constraint Residual: $C(T) = \sum_{t=1}^{T} g_t(x_t)$

- Objective: Achieve regret and constraint residual that scale sub-linearly with T.
  - The long-run performance of the algorithm approaches the benchmark while approximately satisfying the constraint.

Lower bounds

- In unconstrained settings, no regret better than $O(\sqrt{T})$ can be achieved.
- Can we have the no regret property and sublinear scaling in the constraint residual?
- When both are selected by and adversary, in general **no.**
- Counterexample where the adversary can force $\Omega(T)$ regret if constraints are to be satisfied.

S. Mannor, J. N. Tsitsiklis, and J. Y. Yu, "Online Learning with Sample Path Constraints, " *Journal of Machine Learning Research*, 2009

Example where "no regret" is impossible (1)

adversary picks a column

- (Payoff, Constraint) Matrix: $\begin{bmatrix}(1,-1) & (1,1)\\ (0,-1) & (-1,-1)\end{bmatrix}$ agent picks a row: $x_t \in \{1,2\}$

- Main Idea:
  - The adversary has a policy that chooses each column for the same fraction of time.
    - Best action in hindsight has reward t.
  - Switches to column 2 at "bad" rounds for the agent.

- If the agent plays often row 1 while adversary plays column 1 (high payoff), force her to play a low payoff action if she wants to satisfy the constraint (by repeatedly playing column 2)

S. Mannor, J. N. Tsitsiklis, and J. Y. Yu, "Online Learning with Sample Path Constraints, " *Journal of Machine Learning Research*, 2009

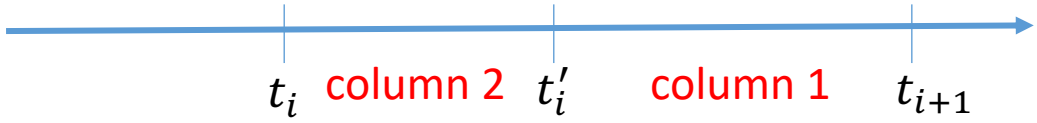
Example where "no regret" is impossible (2)

adversary picks a column

- (Payoff, Constraint) Matrix: $\begin{bmatrix}(1,-1) & (1,1)\\ (0,-1) & (-1,-1)\end{bmatrix}$ agent picks a row: $x_t \in \{1,2\}$

$$\hat{r}_t = \frac{1}{t}\sum_{t'=1}^{t} r_t$$

- Policy for adversary:
  - Keep $\tilde{a}_t = \frac{1}{t}\sum_{l=1}^{t} 1(x_l = 1)$
  - Initialize k=1
  - While k=1 or $\tilde{a}_{t-1} > 3/4$: } Step 1
    - Pick column 2.
    - k=k+1
  - Else: Choose column 1 for the next k rounds and then reset k=1 } Step 2



- Regret: $\lim_{t\to\infty} \sup\left(1-\hat{r}_t\right) \geq \lim_{i\to\infty} \inf\left(1-\hat{r}_{t_{i+1}}\right) \geq \frac{1}{8}$

S. Mannor, J. N. Tsitsiklis, and J. Y. Yu, "Online Learning with Sample Path Constraints, " *Journal of Machine Learning Research*, 2009

K-benchmark

- Use a more restrictive benchmark to obtain non-trivial guarantees for the regret.

- Characterize the performance with respect to a benchmark x_*^K that satisfies the constraint in all sliding windows of K consecutive rounds.

- $K = T$: impossibility result.

- $K = 1$: the benchmark satisfies $g_t\ (x) \leq 0$ at every epoch.
  - $O(\sqrt{T})$ regret w.r.t. x_*^K for $O(\sqrt{T})$ constraint residual is achievable

- $K = o(T)$: covers all intermediate cases.
  - Tradeoffs between having a regret closer to the original one and tight constraint residual guarantees

M. J. Neely and H. Yu, "Online Convex Optimization with Time-Varying Constraints", *arXiV e-prints*, 2017
N. Liakopoulos et al. "Cautious regret minimization: Online optimization with long-term budget constraints," *ICML,* 2019

General algorithm

- Main template:
  - Use gradient descent on the augmented Lagrangian

$$L(x) = Vf_t(x_t) + Q(t)g_t(x_t) + a\|x - x_t\|_2^2$$

  - Predict the cost and constraints of the current epoch as linear approximations $\tilde{f}_t(x), \tilde{g}_t(x)$ of the ones inflicted in the previous epoch.

$$\tilde{f}_t(x) = f_{t-1}(x) + \langle f'_{t-1}(x_{t-1}), x - x_{t-1}\rangle \qquad \tilde{g}_t(x) = g_{t-1}(x) + \langle g'_{t-1}(x_{t-1}), x - x_{t-1}\rangle$$

  - Update the predictor queue with the prediction of the constraints.

$$Q(t) = [Q(t-1) + \tilde{g}_t(x_{t-1})]^+$$

- Action update: $x_t = \Pi_X\left(x_{t-1} - \frac{Vf'_{t-1}(x_{t-1}) + Q(t)g'_{t-1}(x_{t-1})}{2a}\right)$

- Parameters:
  - V: cautiousness parameter, trades off regret vs. constraint residual.
  - a: regularization parameter.

N. Liakopoulos et al. "Cautious regret minimization: Online optimization with long-term budget constraints," *ICML,* 2019

General algorithm: main ideas

- Predict future costs/constraints as first order approximations
  - The main idea in online learning without constraints.

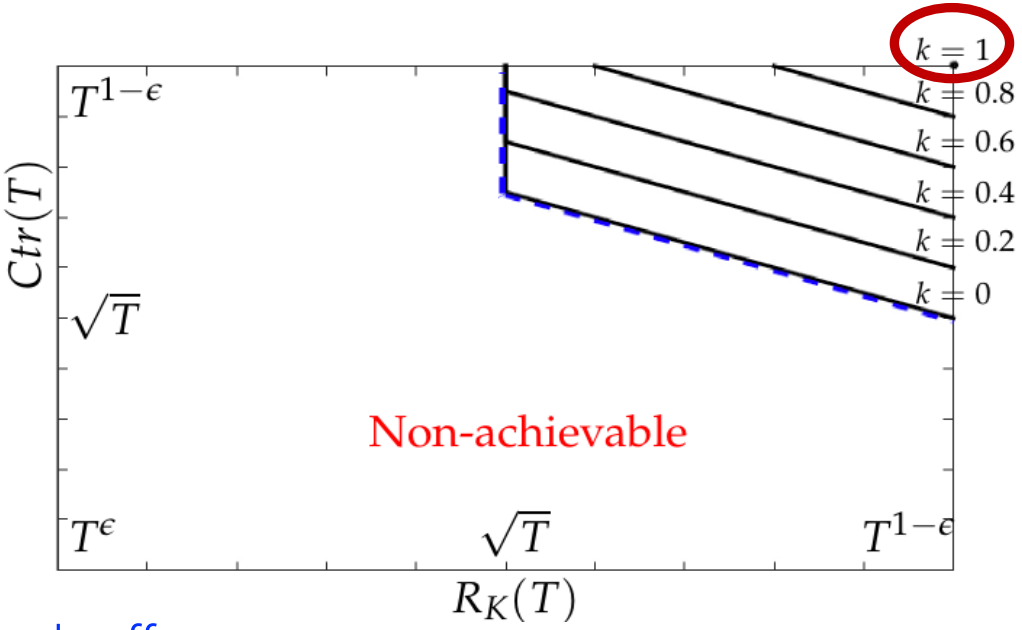
- If Q(t) was fixed, one step of online gradient w.r.t the primal variables in the Lagrangian.

- Multiplier Q(t) to accumulate constraint violations
  - Inspired by the use of queues in stochastic network optimization.
  - Q(t) grows sublinearly with T implies long term constraints are satisfied.
  - Analysis via bounds on the quantity of drift of Q(t) + cost function.
  - Q(t) as a "predictor queue": predicts the violation for the next round.

L. Georgiadis, M.J. Neely, L. Tassiulas, "Resource Allocation and Cross-Layer Control in Wireless Networks", *Foundations and Trands in Networking*, 2006

M. Zinkevich, "Online convex programming and generalized infinitesimal gradient ascent," *ICML*, 2003.

Performance guarantees



- Achievable tradeoff:
  - $O\left(\frac{KT}{V} + \sqrt{V}\right)$ regret w.r.t. the K-benchmark
  - $O\left(\sqrt{VT}\right)$ constraint residual

- Setting $K = T^{1-\varepsilon}$
  - $O\left(T^{1-\varepsilon/2}\right)$ regret for $O\left(T^{1-\varepsilon/4}\right)$ constraint residual.

N. Liakopoulos et al. "Cautious regret minimization: Online optimization with long-term budget constraints," *ICML,* 2019

Proof Sketch

- Predicted drift plus penalty plus smoothness (PDPPS):

$$D(x) = \frac{1}{2}(Q^2(t+1) - Q^2(t)) + V\tilde{f}_t(x) + a\|x - x_{t-1}\|_2^2$$

- Analysis based on upper bound of PDPPS:
  - $D(x) \leq B + V\tilde{f}_t(x) + Q(t)\tilde{g}_t(x) + a\|x - x_{t-1}\|_2^2 = B + r(t)$
  - The algorithm minimizes r(t).

- Derive an upper bound for Q(t+1):
  - Compare K consecutive rounds of the algorithm and x_*^K using sample path properties.
  - Constraint residual follows from the upper bound.

- Regret bound:
  - $r(t) \leq r_{benchmark}(t)$
  - Telescopic sums for the above inequalities.
  - Use of the queue length bounds and constraint residuals to obtain the expressions

Related results

- Non-adversarial constraints
  - Same constraint function per round: $R(T) = O(\sqrt{T} + T/V)$, $C(T) = O(\sqrt{VT})$
  - Stochastic constraint functions: $R(T) = O(\sqrt{T})$, $C(T) = O(\sqrt{T})$
- Adversarial constraints, benchmark with K=1
  - General case: $R(T) = O(\sqrt{T})$, $C(T) = O(T^{3/4})$
  - Slater Assumption: $R(T) = O(\sqrt{T})$, $C(T) = O(\sqrt{T})$
- Perturbed constraints: $g_t(x) = g(x) - b_t$
  - Compare against the best action in hindsight for T rounds.
  - $R(T) = O(\min(T^{\epsilon}, T^{1-\epsilon}))$, $C(T) = O(T^{\epsilon})$
- Use Online convex optimization on a Lagrangian is a common idea when there are constraints.

J. Yuan, A. Lamperski, "Online Comvex Optimization with cumulative constraints," *NeurIPS*, 2018
H. Yu, M.J. Neely, X. Wei, "Online Convex Optimization with Stochastic Constraints," *NeurIPS,* 2017
W. Sun, D. Dey, A. Kapoor, "Safety-Aware Algorithms for Adversarial Contextual Bandit, " *ICML,* 2017
M. J. Neely and H. Yu, "Online Convex Optimization with Time-Varying Constraints", *arXiV e-prints*, 2017
V. Valls et al., "Online Convex Optimization with Perturbed Constraints: Optimal Rates against Stronger Benchmarks ," *AISTATS,* 2020

Cloud reservation: Model

- Action: How much to reserve from each resource.
- Cost: $f_t(x) = \sum_{i=1}^{I} c^i \cdot x_t^i$

  cost of resource i

SLA constraint: resource i can be underprovisioned for no more than fraction ϵ_i of the time

- Constraint functions: $g_t^i(x) = E\{1(x_t^i < \lambda_t^i)\} - \varepsilon_i$

  actual demand for resource i

- Adversary: Selects the demand for each resource.
  - From set $[0, \Lambda_t^i]$
  - Λ_t^i follows a (known) Gaussian distribution

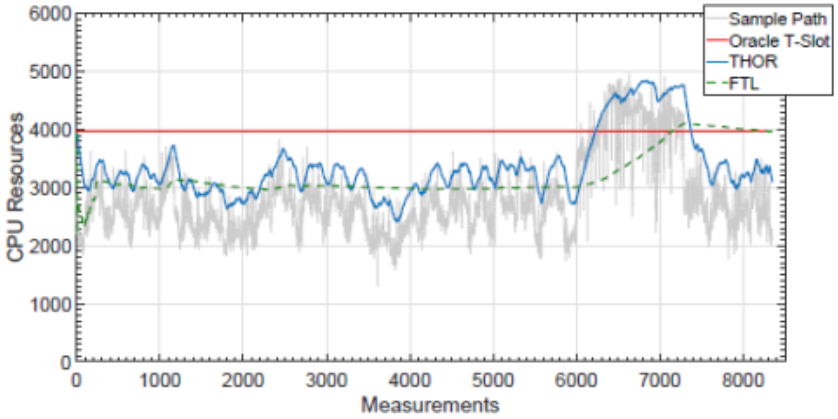
N. Liakopoulos, G. Paschos, T. Spyropoulos, "No Regret in Cloud Resources Reservation with Violation Guarantees, " *IEEE INFOCOM*, 2019

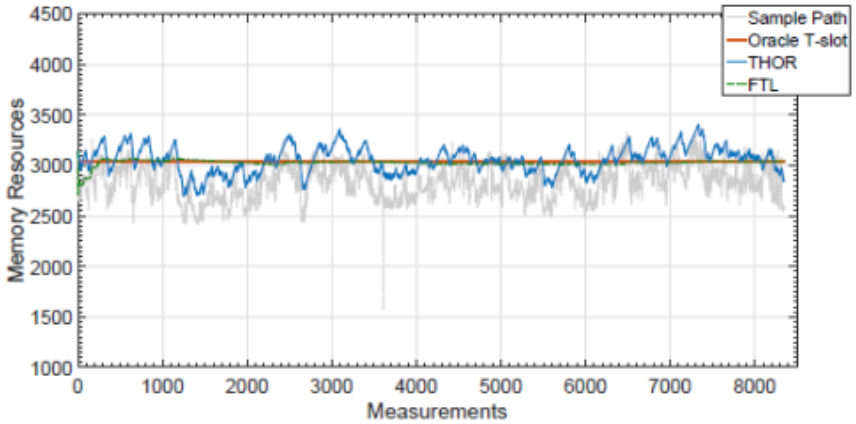
Cloud reservation: Algorithm

- Convexification: $g_{i,t}(x_t^i) = \begin{cases} G_t^i x_t^i +, \beta^i & x_t^i < \mu^i \\ F_{\Lambda_t^i}(x_t^i), & \text{otherwise} \end{cases}$
  - $G_t^i = F'_{\Lambda_t^i}(\mu^i)$
  - $F_{\Lambda_t^i}(z)$: CCDF of the Gaussian upper bound for the demand for resource i
- Resource allocation update: $x_t = \Pi\left(x_{t-1} - \frac{Vc+Q(t)F'_{t-1}(x_{t-1})}{2a}\right)$
- Predictor queue update: $Q_i(t+1) = \left[Q_i(t) + g_{i,t}(x_t^i)\right]^+$

N. Liakopoulos, G. Paschos, T. Spyropoulos, "No Regret in Cloud Resources Reservation with Violation Guarantees, " *IEEE INFOCOM*, 2019

Cloud reservation: Performance

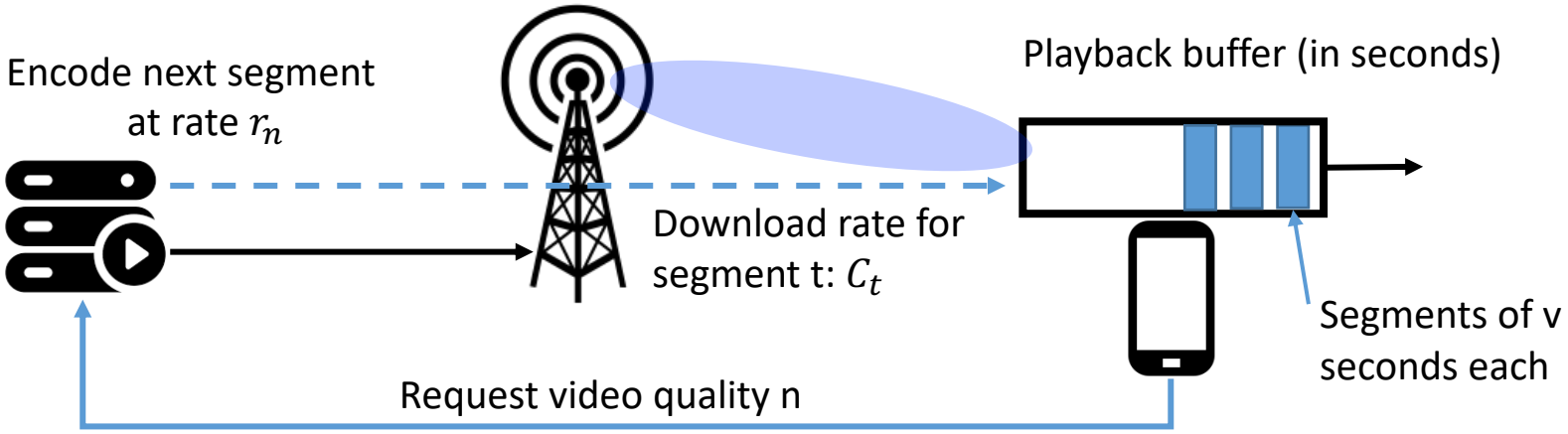




- Results on a Google public dataset.
- Online learning algorithm tracks the changes in resource demands efficiently.

N. Liakopoulos, G. Paschos, T. Spyropoulos, "No Regret in Cloud Resources Reservation with Violation Guarantees, " *IEEE INFOCOM*, 2019

Mobile video streaming

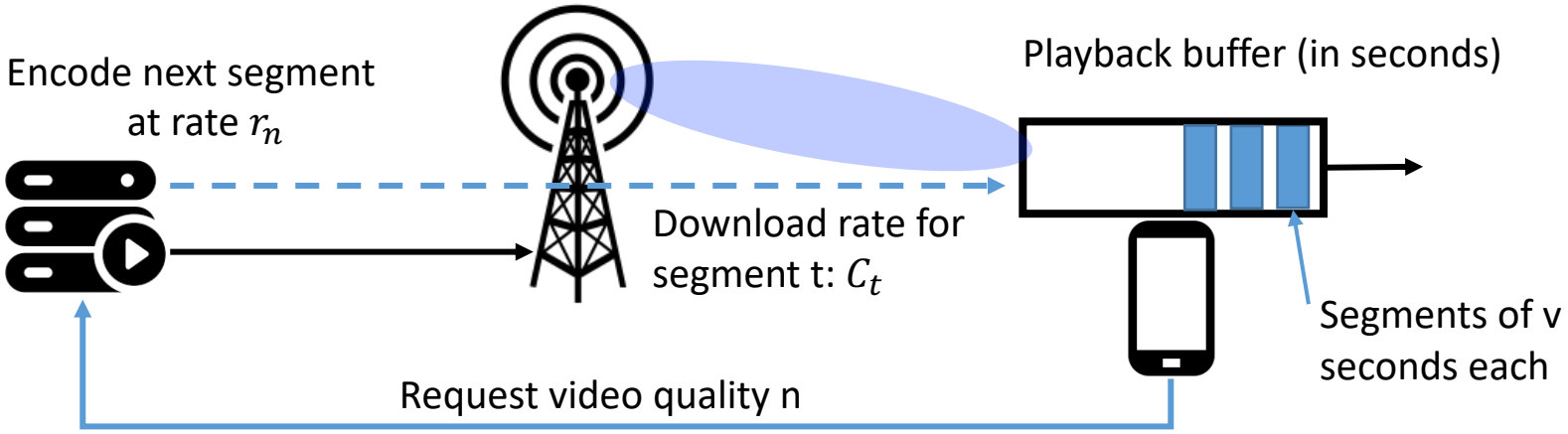


- Client-side adaptation of the video encoding rate:
  - Video client requests the quality for the next segment.

- Buffer based heuristics: Decrease requested quality as buffer empties
  - Segments are downloaded faster to avoid stalls

- Rate based heuristics: Estimate/predict the download rate and adjust the requested quality accordingly.

Mobile video streaming: Model



- Action: Quality level for the t-th segment
- Playback buffer: $B_{t+1} = \left[B_t - \frac{S_{x_t,t}}{C_t}\right]^+ + v - \Delta t$

Accounts for the finite buffer size

size of segment t under the chosen quality level

- Cost: $f_t(x_t) = -r_{x_t}$
- Overflow: $g_t^1(x_t) = \frac{S_{x_t,t}}{C_t} - v$
- Underflow: $g_t^2(x_t) = v - \frac{S_{x_t,t}}{C_t} - \frac{B_{max}}{T}$

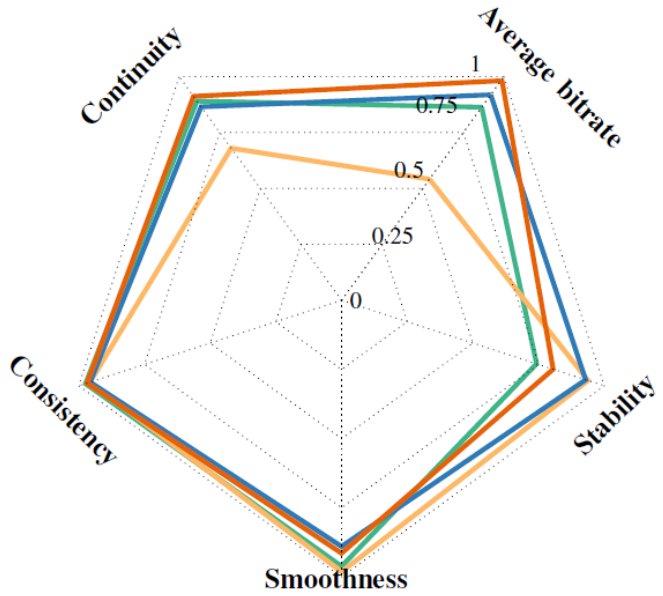
The adversary selects the delivery rate and segment size.

Mobile video streaming: Algorithm

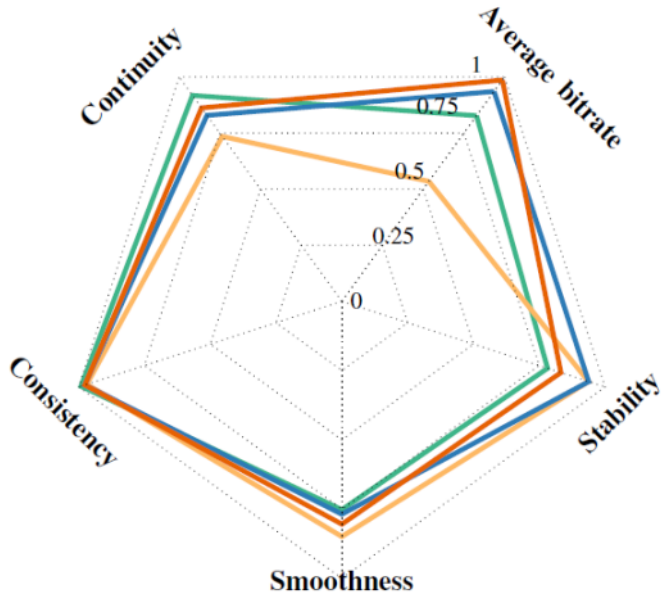
- At most βT quality changes are allowed
- Convexification: Update a probability distribution ω_t over segment qualities
  - Replace $r_{x_t}, S_{x_t,t}$ with $\sum_n \omega_{t,n} r_n$, $\sum_n \omega_{t,n} S_{n,t}$
- Initialization:
  - k=0 (counts the quality changes), t′=1
  - Q(t) = 0
- If $\frac{k}{t} < \beta$: $\omega_{t+1} = \Pi\left[\omega_t - \frac{\sum_{j=t'}^{t} V \cdot f'_{j-1}(\omega_{j-1}) + Q^1(t) \cdot g_{j-1}^{t,'}(\omega_{j-1}) + Q^2(t) \cdot g_{j-1}^{t,'}(\omega_{j-1})}{2a}\right]$, k++, t′=1
- Else: $\omega_t = \omega_{t-1}$
- $Q^i(t) = \left[Q^t(t-1) + g_t^i(\omega_t)\right]^+$

T. Karagkioules et al., “Optimizing Adaptive Video Streaming in Mobile Networks via Online Learning”, *arXiV e-prints*, 2019

Mobile video streaming: Performance



Pedestrian user



User in a car

BB RB L2A ($\beta = 0.3$) L2A ($\beta = 1$)

- Online learning algorithm achieves stable streams with consistently good performance in all metrics.

T. Karagkioules et al., "Optimizing Adaptive Video Streaming in Mobile Networks via Online Learning", *arXiV e-prints*, 2019

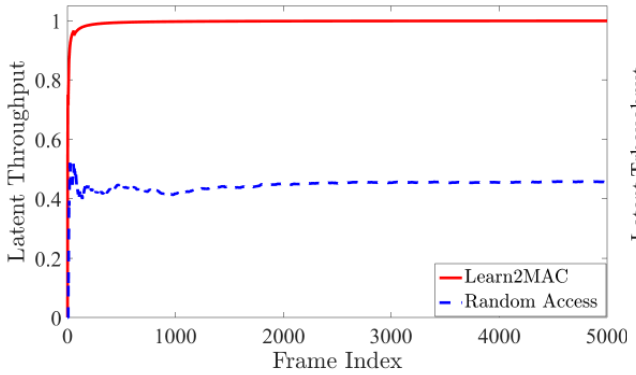
Uncoordinated Medium Access: Setting

- Each device has a deadline of K slots to deliver its packet.
  - Needs N uncontested transmissions.
  - Each transmission has an energy cost.
- Each device has a codebook: transmission pattern over the K slots.
  - Probability distribution over which codes to transmit.
- At the end of each epoch, update the distribution according to slots occupancy and how the other codes would have performed.

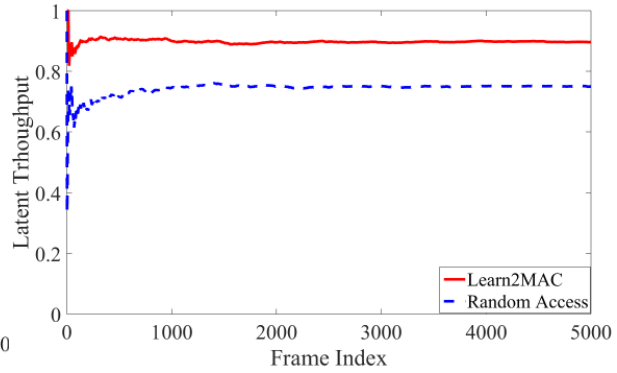
A. Destounis, D. Tsilimantos, M. Debbah, G. Paschos, "*Learn2MAC: Online Learning Multiple Access for URLLC Applications*", Workshop on Ultra Low Latency in Wireless Networks (ULLWN), 2019

Uncoordinated Medium Access: Results

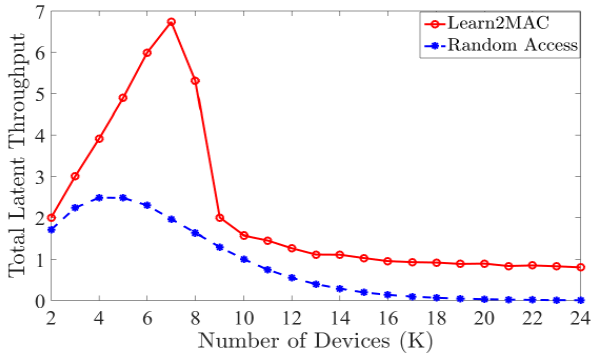
TDMA interference



Random Access Interference



Multiple learning devices



- Online learning algorithm outperforms random access baselines.
- Single learning device:
  - Can learn to adapt to background interference.

- Multiple learning Devices
  - Learn to transmit in non-overlapping slots in low loading (enough resources for all devices)
  - Performance deteroriates sharply in high load and overload though still better than random access.

A. Destounis, D. Tsilimantos, M. Debbah, G. Paschos, "*Learn2MAC: Online Learning Multiple Access for URLLC Applications*", Workshop on Ultra Low Latency in Wireless Networks (ULLWN), 2019

Beyond: Distributed Learning

- Setting: Cost is the sum of costs of agents, which can exchange information over a communication graph.
  - Extension of distributed optimization in the time-varying adversarial setting.

- When constraints muse be satisfied at each round:
  - Distributed online primal-dual mirror descent achieves sublinear regret and constraint residual.

- What can we say about more general long term constraints?

X. Yi et al. “Distributed Online Convex Optimization With Time-Varying Coupled Inequality Constraints ," *IEEE Transactions on Signal Processing,* 2020

Beyond: Multiple Agents

- Motivating example: Uncoordinated multiple access for delay-limited services and energy-constrained devices in IoT.
  - Choose the resource block(s) to transmit to and/or code to use.

- Without long-term constraints:
  - Online gradient/mirror descent generally achieves a coarse correlated Nash equilibrium.
  - Stronger guarantees (e.g. convergence to Nash equilibria) in specific settings (e.g. MIMO precoding).

- How can we incorporate constraints and what guarantees can we get?

P. Mertikopoulos, E. V. Belmega, R. Negrel and L. Sanguinetti, "Distributed Stochastic Optimization via Matrix Exponential Learning," *IEEE Transactions on Signal Processing,* May 2017

Takeaway messages

- Online learning is a framework that can be used to address practical problems in service delivery.
  - Especially cases where modelling is hard and no underlying structure can be assumed.
  - Long term constraints can be accommodated.

- Obtained algorithms are very simple: similar to a gradient descent step.

- Moreover, they do come with performance guarantees.
  - Useful for tuning the parameters.

- Extensions to multiple agents with and without communication among them are of theoretical and practical importance.

Backup Slides

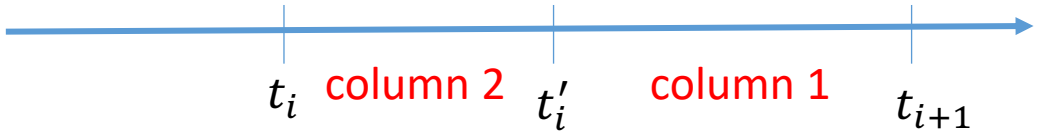
Example where "no regret" is impossible (3)

adversary picks a column

- (Payoff, Constraint) Matrix: $\begin{bmatrix}(1,-1) & (1,1)\\ (0,-1) & (-1,-1)\end{bmatrix}$ agent picks a row: $x_t \in \{1,2\}$

- Analysis as $T \to \infty$

- Step 2 (column 1 for k rounds) happens infinite amount of times
  - There exists infinite sequences of $t_i, t_i', i = 1,2, \ldots$ where the adversary switches the column:



- Columns 1 and 2 are chosen for equal fractions of time: $t_i' \geq t_{i+1}/2$

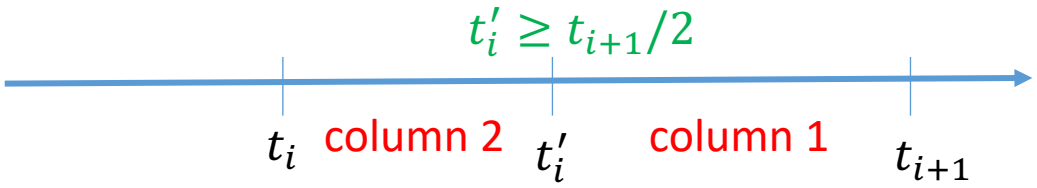
S. Mannor, J. N. Tsitsiklis, and J. Y. Yu, "Online Learning with Sample Path Constraints, " *Journal of Machine Learning Research*, 2009

Example where "no regret" is impossible (4)

adversary picks a column

- (Payoff, Constraint) Matrix: $\begin{bmatrix}(1,-1) & (1,1)\\ (0,-1) & (-1,-1)\end{bmatrix}$

agent picks a row: $x_t \in \{1,2\}$



$$\hat{r}_t = \frac{1}{t}\sum_{t'=1}^{t} r_t$$

- Best action in hindsight: always choose row 1
  - Payoff after t rounds = t.
- Up to round t_{i+1}, the row 1 is played for at most $\frac{3t_i'}{4} + (t_{i+1} - t_i') \leq \frac{7t_{i+1}}{8}$ times
- Regret: $\lim_{t\to\infty} \sup (1 - \hat{r}_t) \geq \lim_{i\to\infty} \inf \left(1 - \hat{r}_{t_{i+1}}\right) \geq \frac{1}{8}$

S. Mannor, J. N. Tsitsiklis, and J. Y. Yu, "Online Learning with Sample Path Constraints, " *Journal of Machine Learning Research*, 2009

Beyond: Fairness among objectives

- Motivating example: resource reservations for network slices.
  - Assign bandwidth and/or computation resources to slices.
  - A priori unknown demand arrives.
  - Cost for each slice (e.g. delay or amount of rejected traffic).

- Objective: balance the costs/payoffs
  - $\sum_k g_k\left(\sum_{t=1}^{T} f_t^k(x_t)\right)$
  - Network utility maximization framework to the adversarial setting.

- How can the online learning framework be extended in this case ?