UNIVERSITY OF CAMBRIDGE

Telefonica

C3PO: Computation Congestion Control (PrOactive)

an algorithm for dynamic diffusion of ephemeral in-network services

Liang Wang, Mario Almeida*, Jeremy Blackburn*, Jon Crowcroft

University of Cambridge, UK Telefonica Research, ES*

Presented by Suzan Bayhan (TU Berlin, Germany)

ACM ICN
2016 KYOTO

Question: Where Computations Happen

- "Fat client and thin server" or "thin client and fat server".
- Trend changes driven by the changes in *usage pattern*, *advances in hardware and software technologies*, and even new *business models*.
- Nowadays, both fat clients and fat servers: *high processing power* and *storage capacity*
- Still difficult to keep their pace with the ever-growing user demands.

UNIVERSITY OF CAMBRIDGE

Observation I: Pervasive Mobile Apps

Pervasive mobile clients have given birth to complex mobile apps. These apps are continuously generating, disseminating, consuming, and processing all kinds of information, in order to provide us convenient daily services.

Observation II: Battery Is The Bottleneck

- Unfortunately, given current battery technology, these demanding apps impose a huge burden on energy constrained devices.
- Power hogging apps are responsible for 41% degradation of battery life on average.
- Even popular ones such as social networks and instant messaging apps (e.g., Facebook and Skype) can drain a device's battery up to 9X faster due only to maintaining an online presence.

UNIVERSITY OF CAMBRIDGE

Observation III: MiddleBoxes Grow Stronger

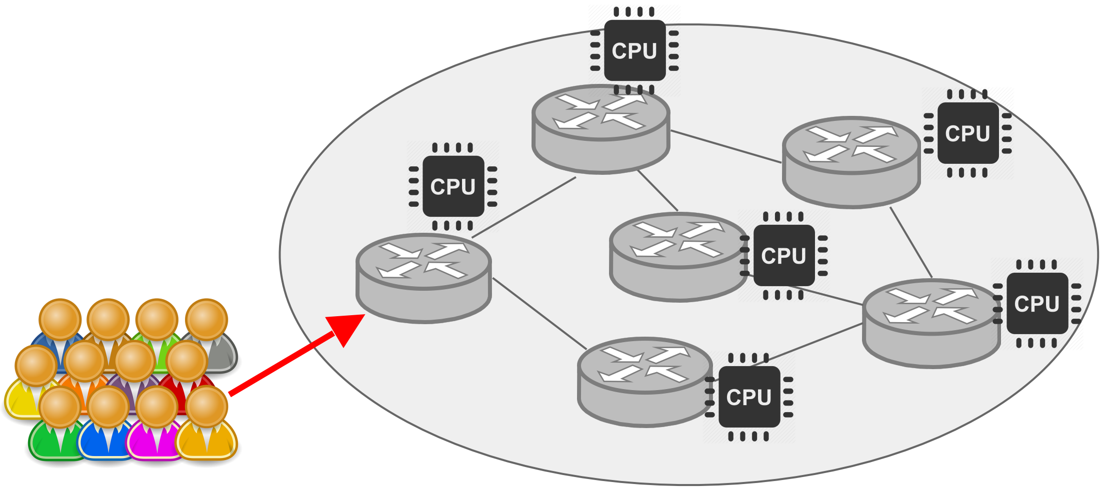
- Quite different from a decade ago, network middleboxes are no longer simple devices which only forward packets.

- ISPs' own network services have been shifting from specialized servers to generic hardware with the adoption of the NFV paradigm.

- E.g., Telefonica is shifting 30% of their infrastructure to NFV by 2016. Other providers such as AT&T, Vodafone, NTT Docomo, and China Mobile.

There are many underutilized in-network resources we can exploit

A New Paradigm: In-network Service Execution



- Ubiquitous in-network service execution

Challenge: How To Avoid Congestions

- Because service execution consumes multiple resources on a router, especially demands CPU cycles for computation intensive tasks, it introduces a new type of "congestion" in a network - "computation congestion".

- An obvious and important question:
  - How to avoid such congestions?

i.e., what would be the mechanism of "Computation Congestion Control" for those ephemeral computation-intensive in-network services?

UNIVERSITY OF CAMBRIDGE

How Is It Different From Previous Settings

- Traffic congestion: the solutions usually either
  - try to reduce the transmission rate or
  - take advantage of multiple paths.
  - congestions are avoided by the cooperation of both communication ends.

But: In-network services do not necessarily impose a point-to-point paradigm!

- Load balancing: a cluster often has a regular structure, i.e., network topology, central coordination, homogeneous configurations, uniform demands, and etc. Therefore, fully centralised control is the norm in the cluster. The jobs are often able to tolerate long scheduling delay.

UNIVERSITY OF CAMBRIDGE

Characteristics of Our Context

Our context in an ISP network is more complicated:

1. the underlying topology is not regular
2. the node configurations can be heterogeneous
3. demands distribution is highly skewed hence the resources in a neighbourhood needs to be well utilised
4. central coordination is often expensive and reduces responsiveness of those networked nodes
5. services (and clients) are intolerable to long scheduling delay.

UNIVERSITY OF CAMBRIDGE

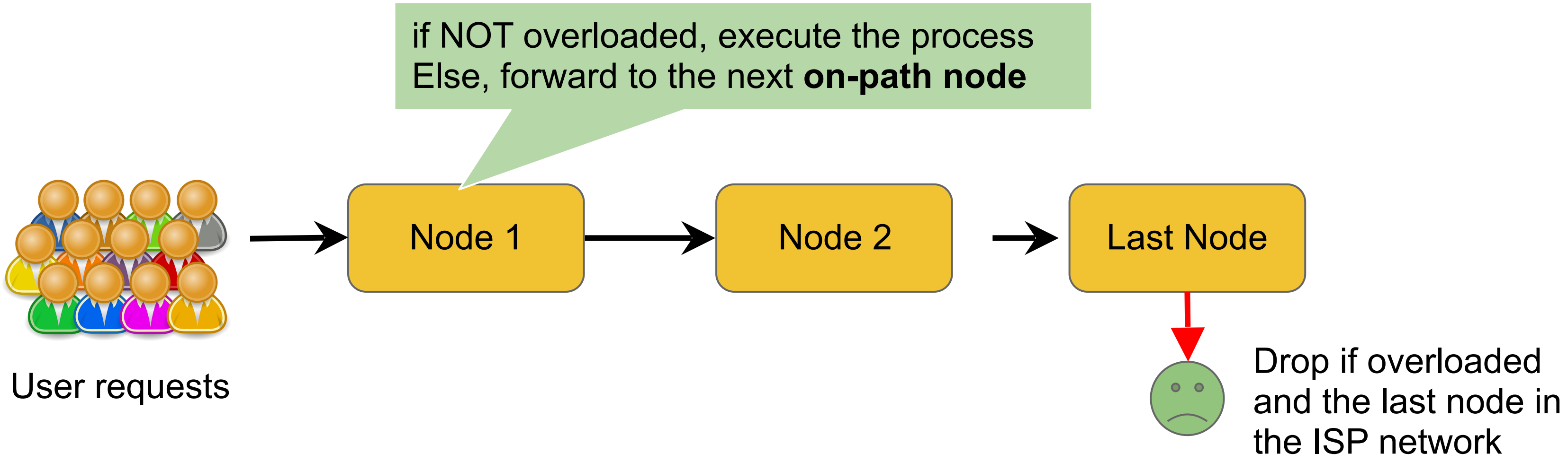
C3PO: Computation Congestion Control (PrOactive)

C3PO: a low-complexity distributed load balancer based on the pessimistic prediction of the service queue.

Why do we go for a distributed one instead of a centralised solver?

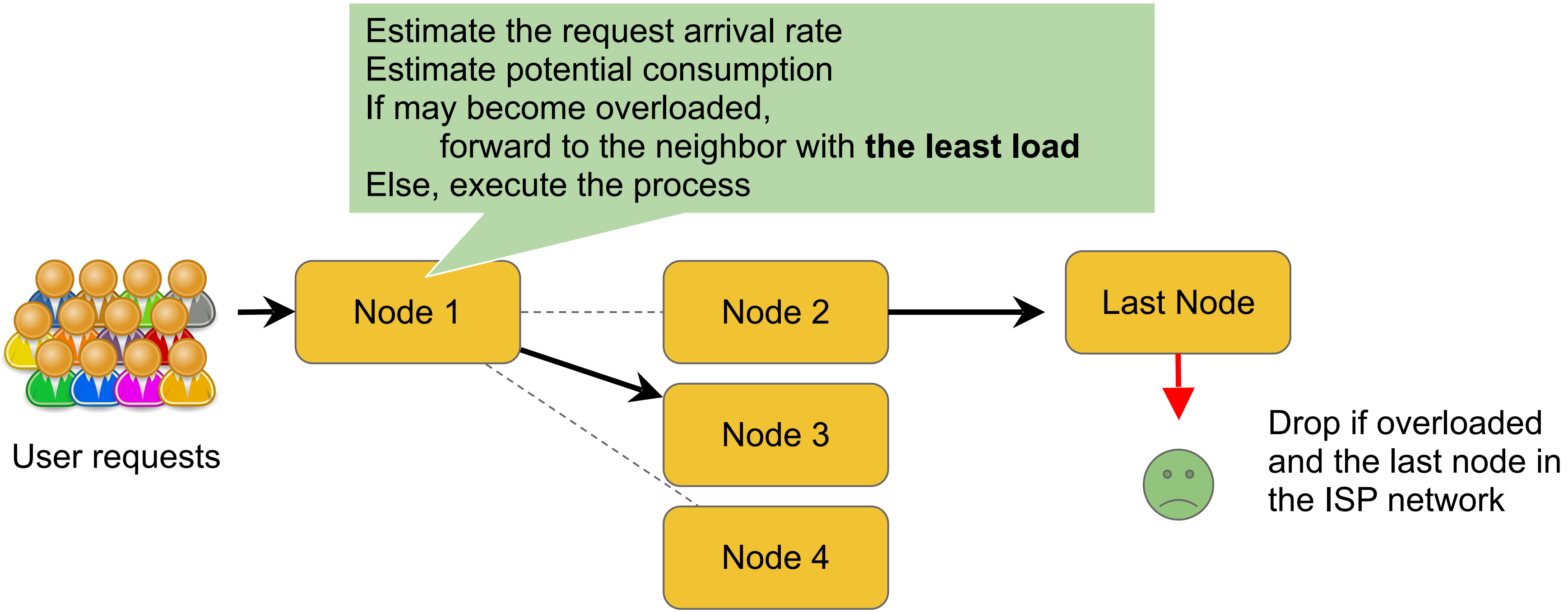
- A central solver needs global knowledge of all the nodes in a network
- the optimal strategy needs to be calculated periodically given the dynamic nature of a network and traffic
- there is a single point of failure
- there is often only marginal improvements over a smartly designed heuristic.

Two Strategies Are Studied: Passive Control



+ Very simple
- Reactive

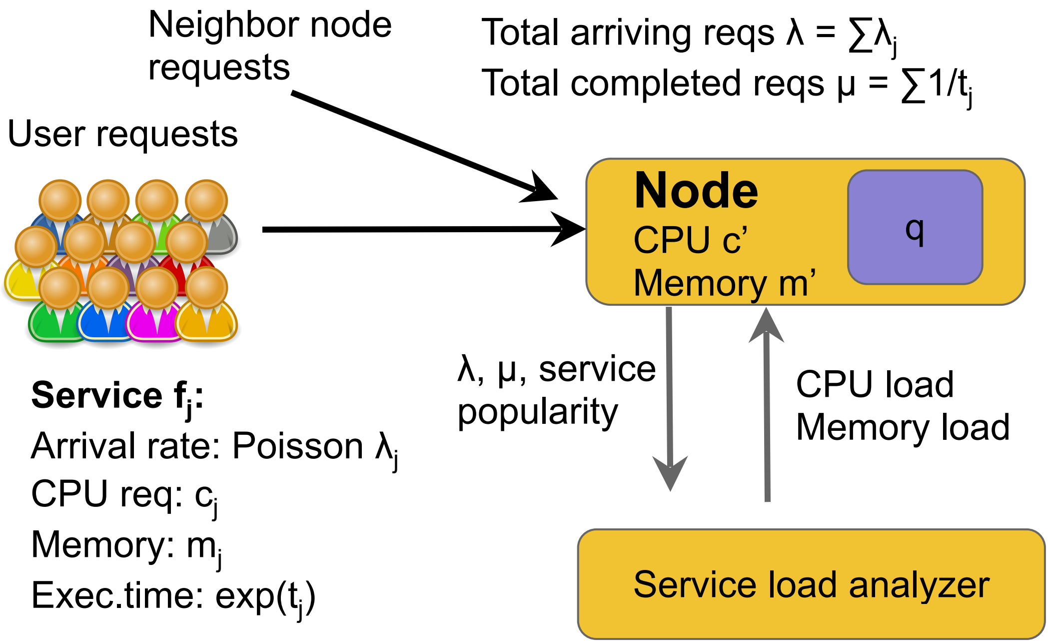
Proactive Control



- \+ Proactive, anticipatory
- \- Requires 1-hop information exchange

UNIVERSITY OF CAMBRIDGE

Proactive Control: A Closer Look



Service f_j:
Arrival rate: Poisson λ_j
CPU req: c_j
Memory: m_j
Exec.time: $\exp(t_j)$

- Birth-death (λ, μ) system
- Analysis of M/M/1-PS model
- What is the **expected**
  - number of processes
  - CPU requirement
  - memory requirement

  running on the node?
- For a stable system,
  - workload < node capacity
- Probabilistically (q) execute requests to ensure stability
- Tune q based on expected load

UNIVERSITY OF CAMBRIDGE

Proactive Strategy: C3PO

Algorithm 1 C3PO - Proactive Computation Control

void **on_arrival** (request r):
    $\text{buf}_\lambda[i] \leftarrow \text{timestamp}(r)$
    $\lambda \leftarrow \text{mean\_rate}(\text{buf}_\lambda)$
    $\Delta\lambda \leftarrow \max(0, \lambda - \lambda')$
    $\lambda \leftarrow \lambda + \Delta\lambda$
    $q \leftarrow$ **eq.4** $(\lambda, \mu, c', c'', m', m'')$
    if draw_uniform $([0,1]) < q$ **then** execute (r)
    else forward_to_lightest_load_node (r)
    $i \leftarrow (i+1) \bmod k$
    if $i == 0$ **then** $\lambda' \leftarrow 0.5 \times (\lambda' + \lambda - \Delta\lambda)$

void **on_complete** (function s):
    $\text{buf}_\mu[i] \leftarrow \text{execution\_time}(s)$
    $\text{buf}_{c''}[i] \leftarrow \text{CPU\_consumption}(s)$
    $\text{buf}_{m''}[i] \leftarrow \text{memory\_consumption}(s)$
    $i \leftarrow (i+1) \bmod k$
    if $i == 0$ **then**
        $\mu \leftarrow 0.5 \times (\mu + \text{mean}(\text{buf}_\mu)^{-1})$
        $c'' \leftarrow 0.5 \times (c'' + \text{mean}(\text{buf}_{c''}))$
        $m'' \leftarrow 0.5 \times (m'' + \text{mean}(\text{buf}_{m''}))$
    forward_result (s)

on arrival, we record the time of arrival and update the mean arrival rate.
Upon each arrival, we calculate q - *probability of execution*

on service completion, we record the execution time, memory and CPU consumption, and calculate the average values

Why pessimistic?

Algorithm 1 C3PO - Proactive Computation Control

```
void on_arrival (request r):
    buf_λ[i] ← timestamp (r)
    λ ← mean_rate (buf_λ)
    Δλ ← max(0, λ − λ')
    λ ← λ + Δλ
    q ← eq.4 (λ, μ, c', c'', m', m'')
    if draw_uniform ([0,1]) < q then execute (r)
    else forward_to_lightest_load_node (r)
    i ← (i + 1) mod k
    if i == 0 then λ' ← 0.5×(λ' + λ − Δλ)

void on_complete (function s):
    buf_μ[i] ← execution_time (s)
    buf_c''[i] ← CPU_consumption (s)
    buf_m''[i] ← memory_consumption (s)
    i ← (i + 1) mod k
    if i == 0 then
        μ ← 0.5×(μ + mean(buf_μ)^−1)
        c'' ← 0.5×(c'' + mean (buf_c''))
        m'' ← 0.5×(m'' + mean (buf_m''))
    forward_result (s)
```

Always considers the increase in arrival rate!

Implementation Details

Algorithm 1 C3PO - Proactive Computation Control

1: void **on_arrival** (request r):
2: $\quad \mathrm{buf}_{\lambda}[i] \leftarrow \mathrm{timestamp}\ (r)$
3: $\quad \lambda \leftarrow \mathrm{mean\_rate}\ (\mathrm{buf}_{\lambda})$
4: $\quad \Delta\lambda \leftarrow \max(0, \lambda - \lambda')$
5: $\quad \lambda \leftarrow \lambda + \Delta\lambda$
6: $\quad q \leftarrow$ **eq.4** $(\lambda, \mu, c', c'', m', m'')$
7: $\quad$ **if** draw_uniform $([0,1]) < q$ **then** execute (r)
8: $\quad$ **else** forward_to_lightest_load_node (r)
9: $\quad i \leftarrow (i+1) \bmod k$
10: $\quad$ **if** $i == 0$ **then** $\lambda' \leftarrow 0.5 \times (\lambda' + \lambda - \Delta\lambda)$
11:
12: void **on_complete** (function s):
13: $\quad \mathrm{buf}_{\mu}[i] \leftarrow \mathrm{execution\_time}\ (s)$
14: $\quad \mathrm{buf}_{c''}[i] \leftarrow \mathrm{CPU\_consumption}\ (s)$
15: $\quad \mathrm{buf}_{m''}[i] \leftarrow \mathrm{memory\_consumption}\ (s)$
16: $\quad i \leftarrow (i+1) \bmod k$
17: $\quad$ **if** $i == 0$ **then**
18: $\qquad \mu \leftarrow 0.5 \times (\mu + \mathrm{mean}(\mathrm{buf}_{\mu})^{-1})$
19: $\qquad c'' \leftarrow 0.5 \times (c'' + \mathrm{mean}\ (\mathrm{buf}_{c''}))$
20: $\qquad m'' \leftarrow 0.5 \times (m'' + \mathrm{mean}\ (\mathrm{buf}_{m''}))$
21: $\quad$ forward_result (s)

- We use **four fixed-size circular buffers** instead of fixed time window to prevent the memory usage from being subject to service arrival/completion rate. The reason is the number of arrived requests can vary in a fixed time window.

- Parameter *k* (buffer size) represents a trade-off between **stability** and **responsiveness**.
  - Larger *k* → more stable estimates (longer history considered)
  - Smaller *k* → higher responsiveness to the changes in λ and μ.

- Although λ needs to be calculated at every request arrival (line 3), further optimisations can be easily done to reduce the complexity.

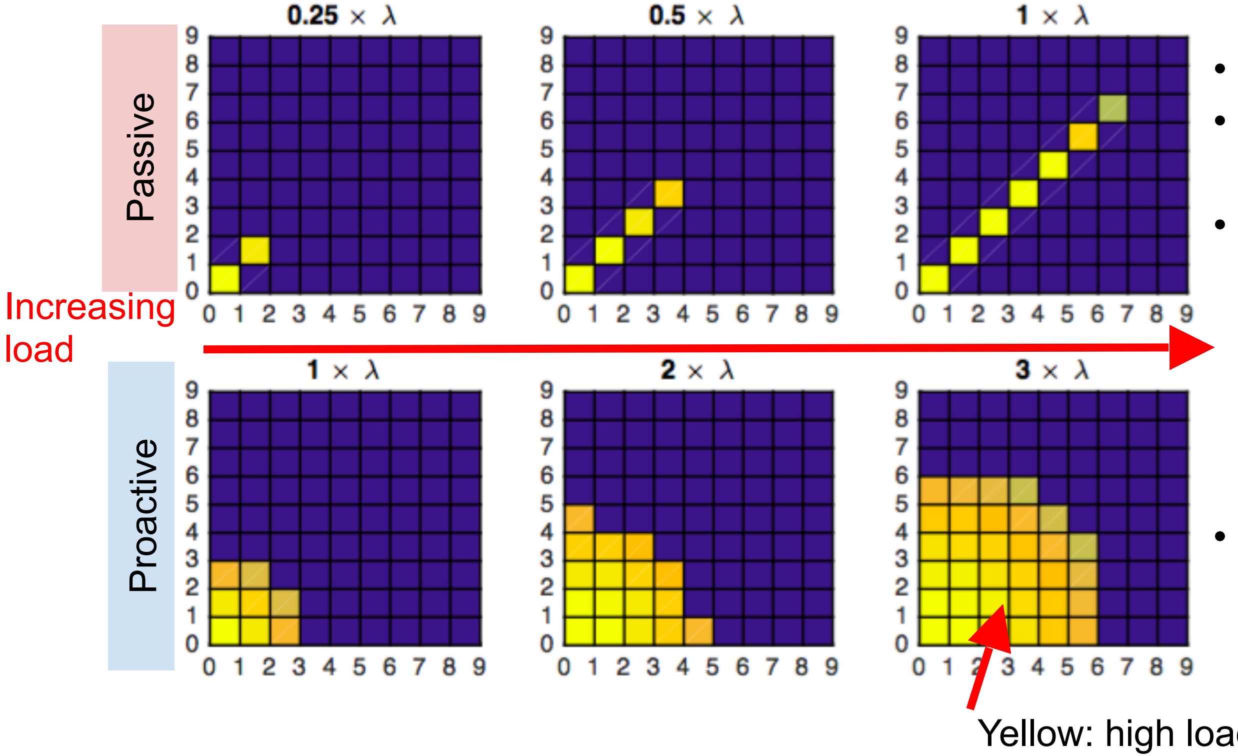
UNIVERSITY OF CAMBRIDGE

Evaluation Setup

- Aim: analyze how different strategy impacts **load distribution** as well as **latency**, **drop rate**, **responsiveness to jitters**.
- ICARUS simulator
- We test three strategies on both synthetic and realistic topologies
    - None (only edge routers execute services),
    - Passive,
    - Proactive
- Poisson request stream with $\lambda = 1000$ as arrival rate
- We assume CPU is the first bottleneck in the system for computation intensive services, and only present the results of using Exodus network in the following.

UNIVERSITY OF CAMBRIDGE

C3PO Exploits Its Neighbourhood



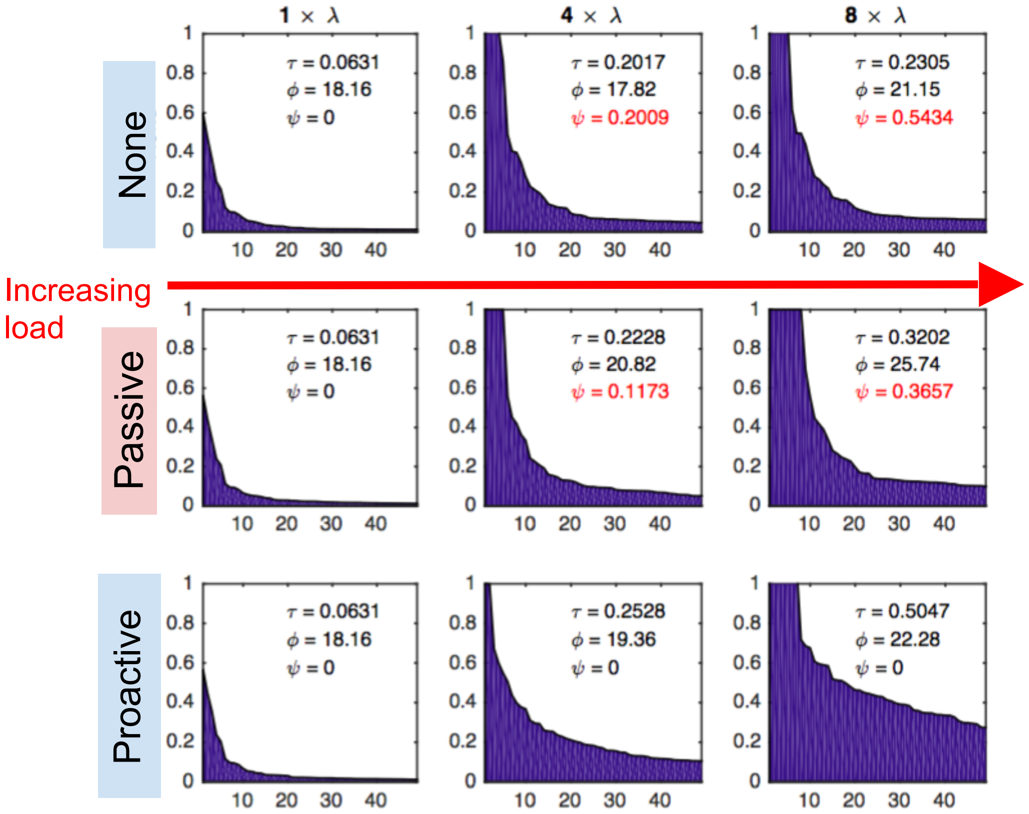
- Client connected to router at (0,0)
- Server connected to router at (9,9).

- Proactive is more capable at utilizing the nearby resources within its neighborhood, leading to *better load balancing and smaller latency*.

- On-path routers: a few due to small-world structure of networks

UNIVERSITY OF CAMBRIDGE

Scalable to Workload on Exodus ISP network

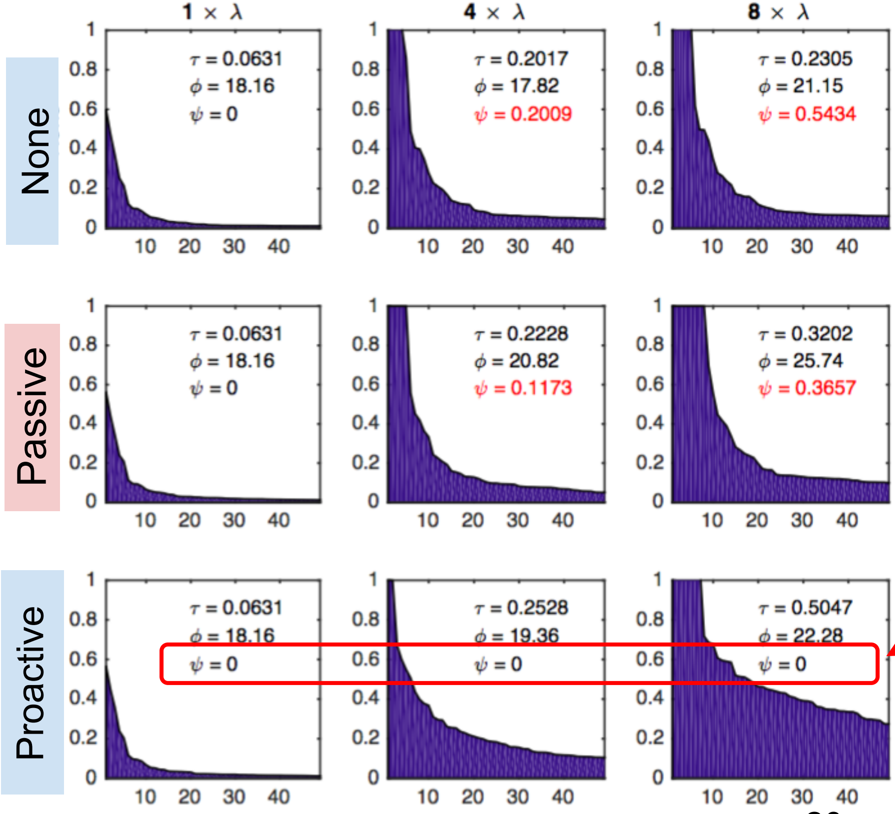


- *x*-axis is node index and *y*-axis is load.
- Top 50 nodes (of 375) of the heaviest load are sorted in decreasing order
- For None, there are 60-80 edge routers
- 5 ms link latency

τ : average load;
φ : average latency (in *ms*);
ψ : ratio of dropped requests.

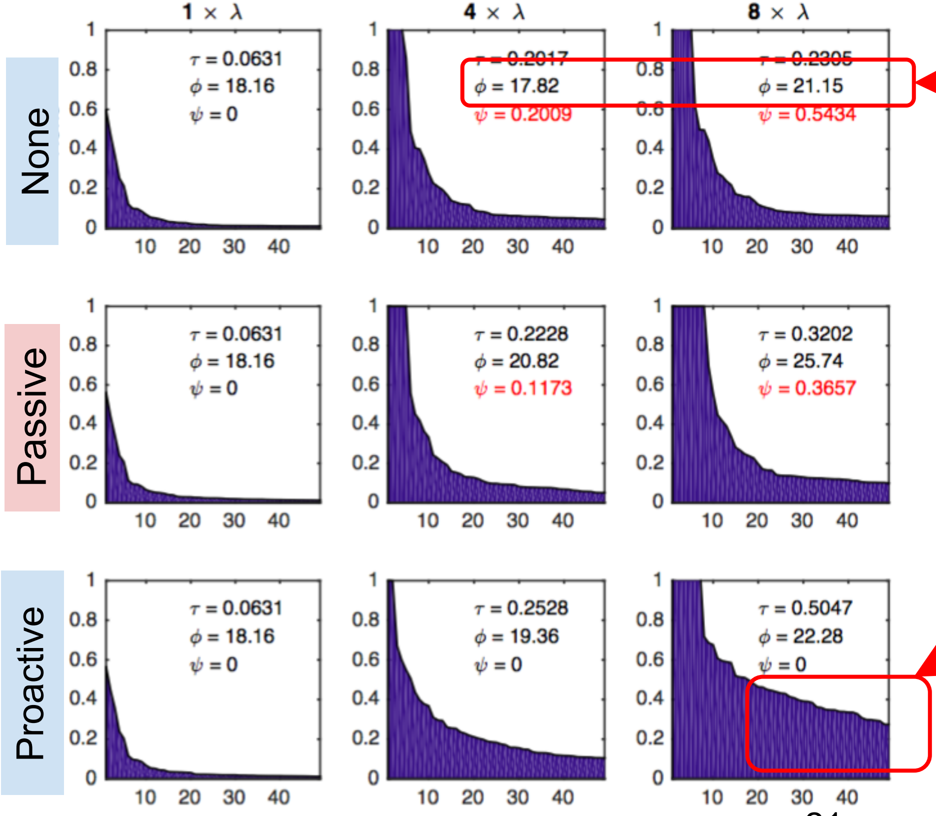
UNIVERSITY OF CAMBRIDGE

Scalable to Workload on Exodus ISP network



Proactive: No service drops due to load balancing capability

Scalable to Workload on Exodus ISP network

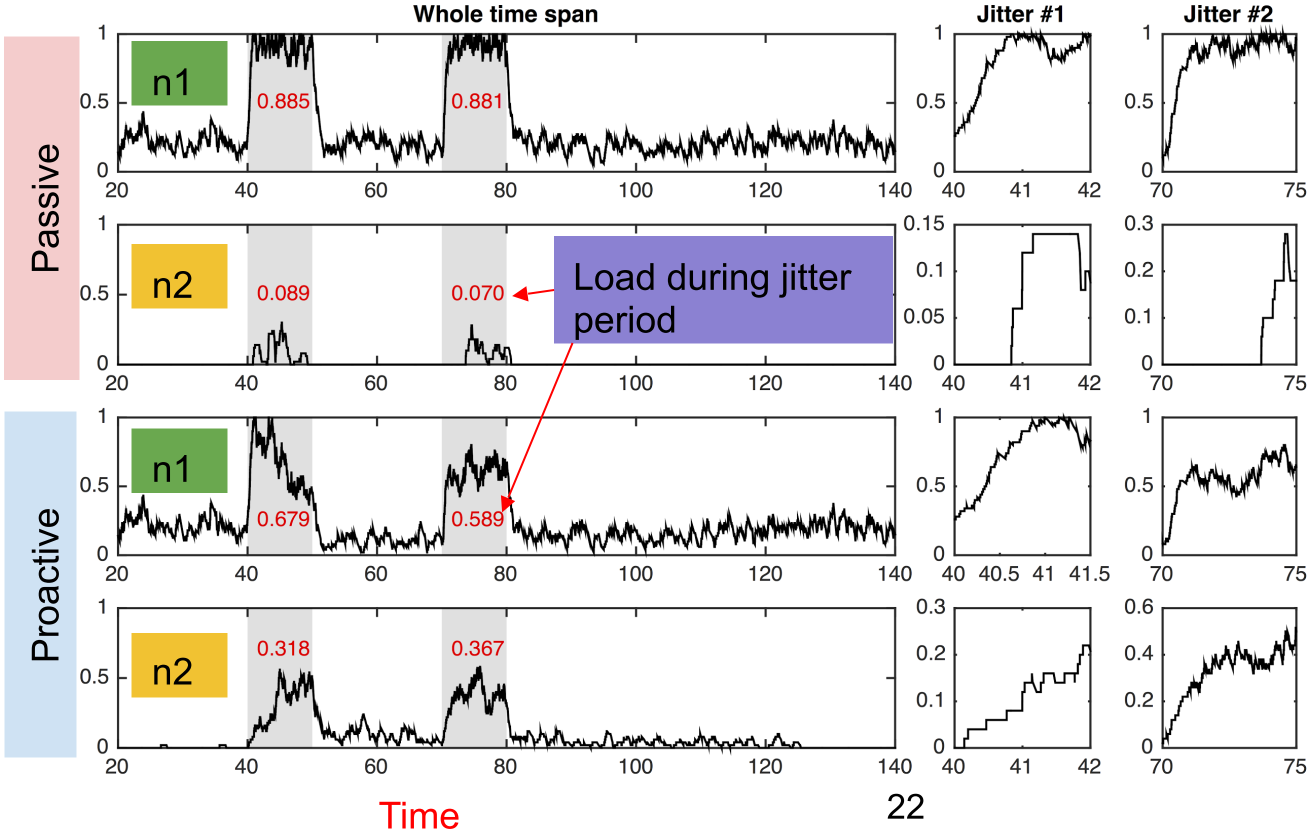


Lower latency because only edge routers execute services

Heavy Tail proves the success of load balancing in the network

UNIVERSITY OF CAMBRIDGE

Responsiveness to Jitters



Simple line topology:
client → $n1$ → $n2$ → server.

Jitters (6λ arrival for 10ms) are injected at time 40 ms and 70 ms.

C3PO balances the load on n1 and n2 whereas Passive cannot!

UNIVERSITY OF CAMBRIDGE

Conclusion

- We studied two control strategies (Passive and Proactive) to control computation congestion in in-network service execution.
- Based on the Proactive control, we proposed a fully distributed, low complexity, and responsive load controller to avoid potential computation congestions when executing in-network services.
- Our results showed that
  - the proposed solution C3PO can effectively take advantage of available resources in a neighbourhood to balance the service load and further reduce service latency and request drop rate.

UNIVERSITY OF CAMBRIDGE

Thank you. Questions?

For more information: Liang Wang

liang.wang@cl.cam.ac.uk

UNIVERSITY OF CAMBRIDGE

Intuitive Explanations of C3PO

Stationary analysis of M/M/1-PS model on the service queue on a router.

Intuitively:

- Try to estimate the future incoming request rate based on the previous observations,
- If a router thinks it is going to be overloaded based on its capacity and service requirements, it only probabilistically executes some of the service requests.
- When estimating the future request rate, a router takes into account the second-order information (i.e., the derivative) of the rate. Essentially, it is smoothing. But it only considers the positive derivatives (line 4 in the code), so it is “pessimistic”, therefore “proactive”. Why :)

UNIVERSITY OF CAMBRIDGE