Mystique: A Fine-grained and Transparent Congestion Control Enforcement Scheme

Posco Tso PhD, SMIEEE, FHEA
Computers Science, Loughborough University

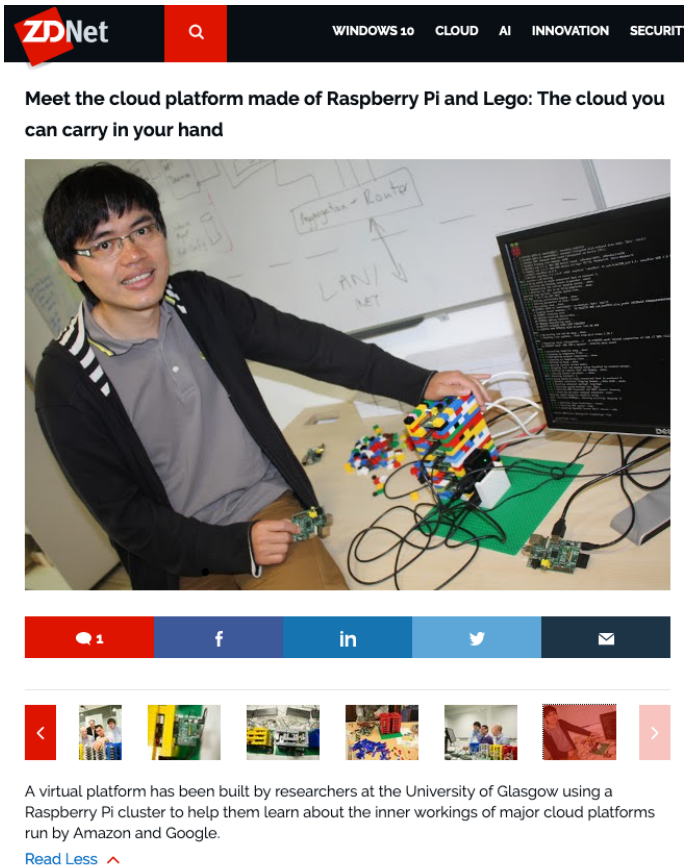
暨南大學
JINAN UNIVERSITY

Loughborough University

澳門大學
UNIVERSIDADE DE MACAU
UNIVERSITY OF MACAU

Who am I?

- Senior Lecturer (UK's equivalent to Associate Professor)
- Grew up and educated in Hong Kong
- Like building networked systems
- Received over £1M research funding



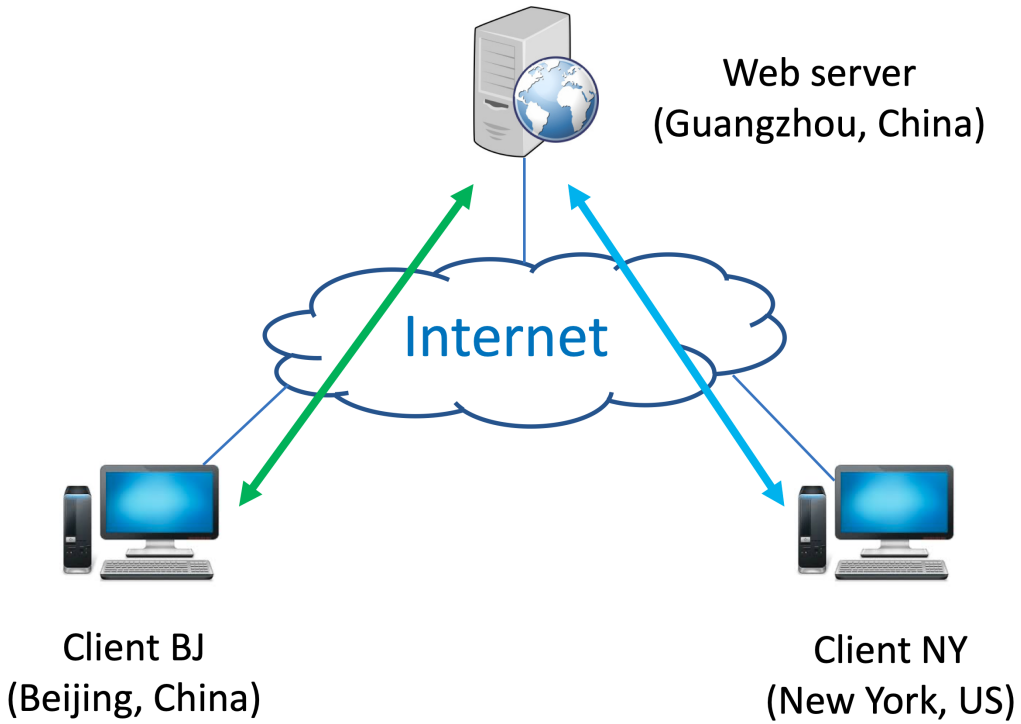
Where are we?

Loughborough University is a top 10 public research university in the UK.

TCP Congestion Control

- Congestion is not uncommon on the Internet [Singla, HotNets'14]
- Latency does matter
  - 80th latency exceeds 100 times the speed of light [Bozkurt, PAM'17]
  - 100ms latency penalty contributes to 1% revenue loss [Flach, SIGCOMM'13]
- New TCP congestion control (CC) schemes have been proposed
  - e.g., Sprout, PCC, BBR, PCC-vivace, Copa, etc.

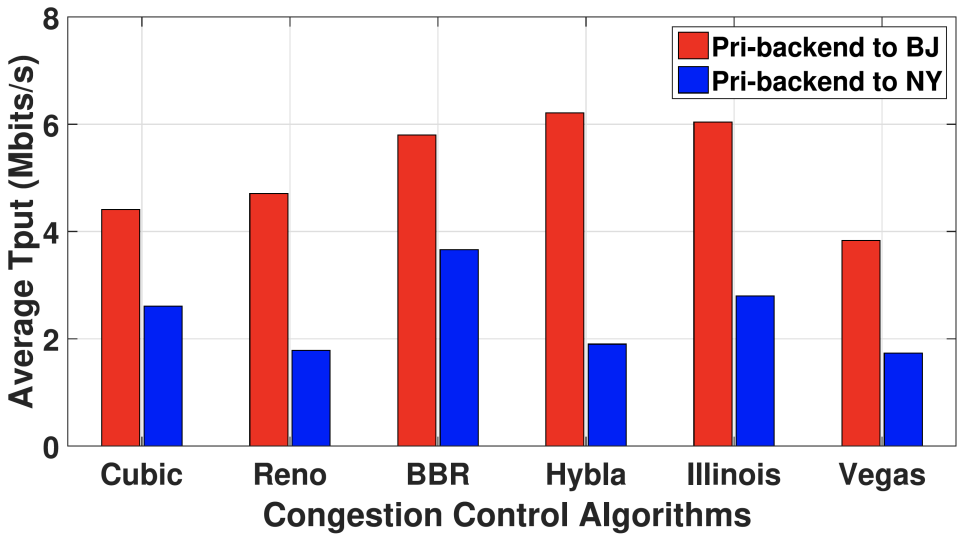
But, can one CC rule them all?



We set out to find the answer.

But, can one CC rule them all?

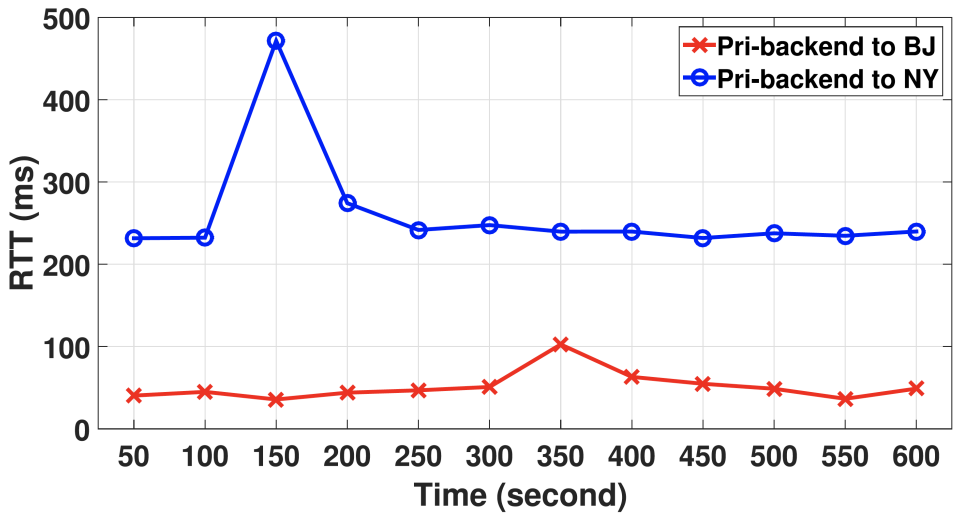
- CC only reaches peak performance in 'prescribed' scenarios.
  - e.g., RTT, loss ratio, ...



Reno has better performance than Cubic for Pri-backend$\longrightarrow$BJ, but worse from Pri-backend$\longrightarrow$NY.

But, can one CC rule them all?

- Also, network condition, i.e. RTT, oscillates dynamically
  - e.g., RTT, loss ratio, ...



Implication:Different congestion control algorithms can be the most efficient at certain times even for a single TCP connection

Adjusting servers' CC is difficult, if not impossible

- Consider multi-tenant cloud environment
  - Admins can not control server TCP stacks
  - Servers are setup and managed by different tenants
- For a single tenant:
  - Large amount of servers
  - Diverse versions of OS
- Per-socket basis CC:
  - Need to modify every application's implementation

We need a better solution

We aim to design a scheme to deploy and schedule CC with following properties:

- Transparent: NO modification to the OS of servers
- Fine-grained: flow-level CC enforcement
- Dynamic: network-awareness

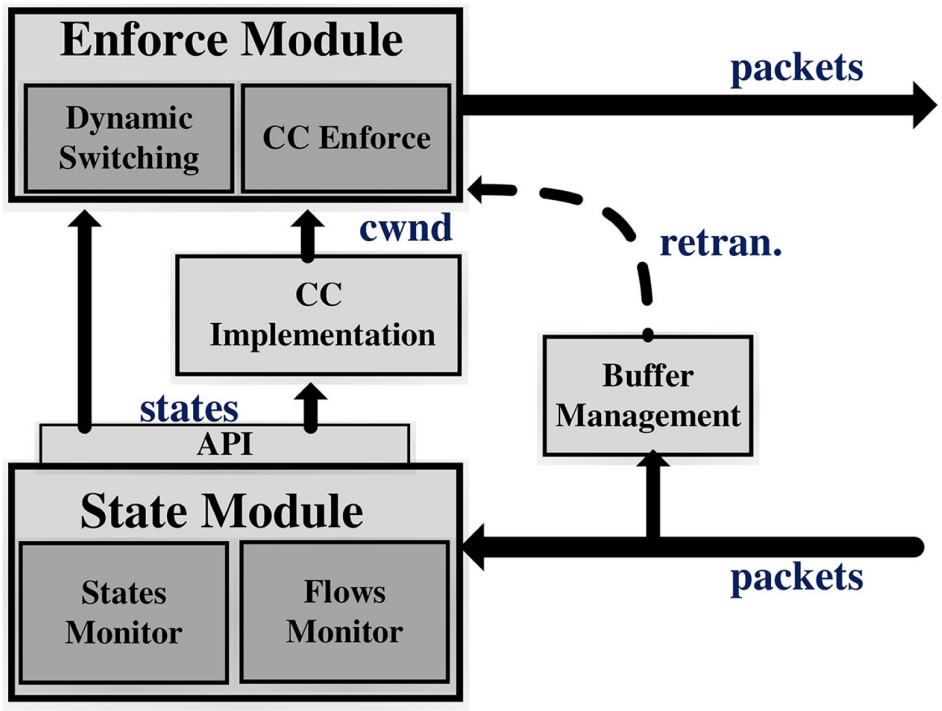
Design objectives

- Implements TCP congestion control in OVS
  - No modifications to Web servers' kernel
  - Per-flow basis congestion control switching
- Dynamically employ the most suitable CC
  - switch CC according to corresponding network condition

Comparing Mystique with other schemes

	Dynamic CC switching	**No TCP stack modification**	**Granularity**
AC/DC	✗	✓	OS Level
vCC	✗	✓	OS Level
NetKernel	✗	✗	OS Level
CCP	✗	✗	OS Level
MCC	✓	✗	Per-flow
Mystique	✓	✓	Per-flow

Mystique: High Level View



Implication:Implementing in OVS requires no modification on servers

Obtaining Congestion Control States

- Per-flow connection tracking
  - All traffic goes through the Open vSwitch (OVS)
  - We can reconstruct CC via monitoring all the packets of a connection

    Packets $\longrightarrow$ Flow classification $\longrightarrow$ Updating CC variables
- Maintain per-flow congestion control variables
  - E.g., CC-related RTTs, Bandwidth, loss etc.

APIs provided by Mystique

- Implement and deploy new CC schemes
- CC switching logic

Methods	**Descriptions**
getCRTT()	Obtain state *c_rtt*'s value
getCBW()	Get state *c_bw*'s value
getMinRTT()	Get state *min_rtt*'s value
getMaxBW()	Get state *max_bw*'s value
setPeriod()	Set parameter *period*'s value
setTstep()	Set parameter *t_step*'s value
setBWstep()	Set parameter *bw_step*'s value
isLoss()	Offer loss feedback
setCwnd()	Set new congestion window

Enforcing Congestion Control

- TCP sends min(cwnd, rwnd)
  - cwnd is congestion control window (congestion control)
  - rwnd is receiver's advertised window (flow control)
- Mystique reuses rwnd for congestion control purpose
  - Web servers with unaltered TCP stacks will naturally follow our enforcement
- Congestion signals should not be sent to web servers
  - Keep the servers' cwnd at high level
  - Flows' sending rate is throttled by Mystique

Dynamic Switching – an example

Algorithm 1 Congestion control switching logic example

Require: Congestion states, e.g, loss, RTT
  if no loss **then**
    Return Hybla
  else if RTT $< 50ms$ **then**
    Retrun Illinois
  else
    Return BBR
  end if

Implementable locations

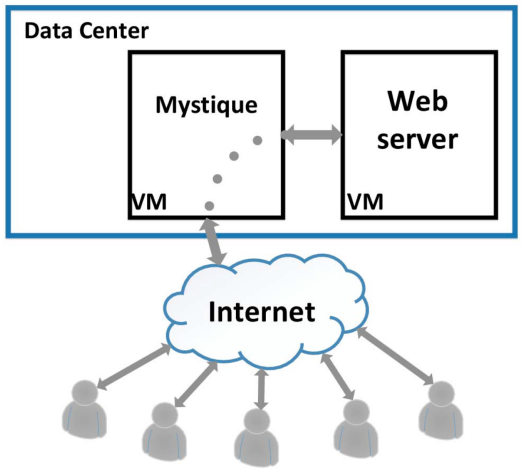
- VMs
  - allowed to setup new servers or release old ones dynamically
  - requires routers/switches redirecting desired traffic
- Hypervisors
  - easy scaling with numbers of servers
  - no route redirection is required
  - flexibility and scalability are limited
- Routers/Switches
  - easily enforce congestion control without route redirection
  - difficult to perform load balancing

Implementation

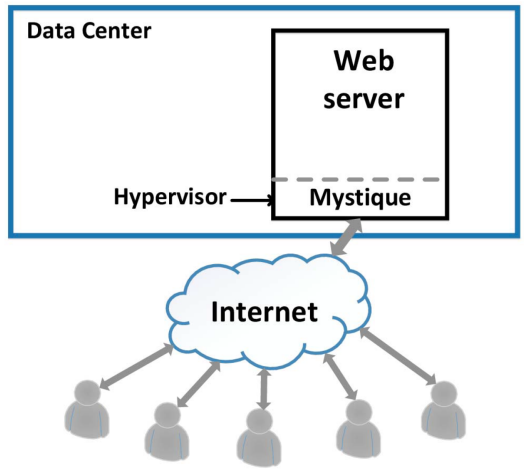
- Prototype implementation in Open vSwitch (OVS) kernel datapath
  - 1400 LoC added
- Leverages available techniques to improve performance
  - RCU-enabled hash tables to perform connection tracking
  - skb_clone is used for packet buffering to prevent deep-copy of data
  - Multi-threading technique is used for faster processing
  - Leverages NIC TSO (TCP segmentation offload) to increase performance

Testbed setting

- Mystique and Web servers
  - Private cloud: Campus DC in Jinan University, Guangzhou
  - Public cloud: AWS at Singapore
- Five clients
  - Guangzhou, China (GZ)
  - Shenzhen, China (SZ)
  - Beijing, China (BJ)
  - London, UK (Lon)
  - New York, US (NY)



(a) *Mystique* in VM



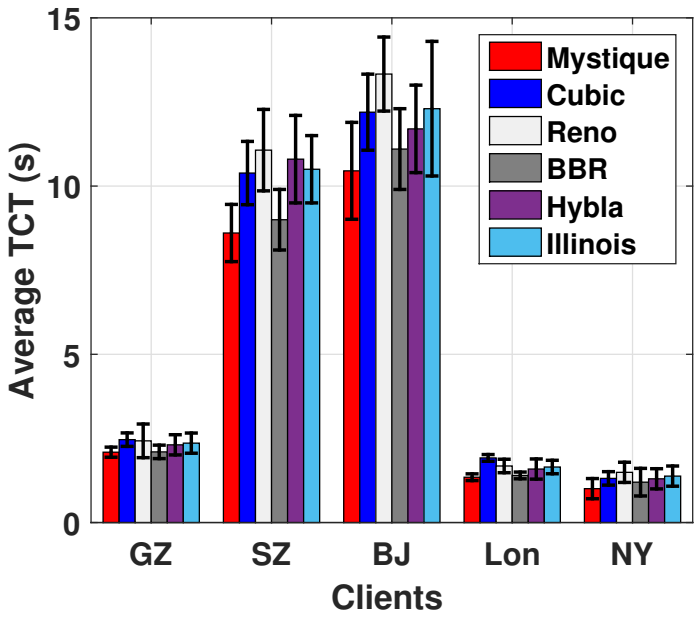
(b) *Mystique* in Hypervisor

Metrics

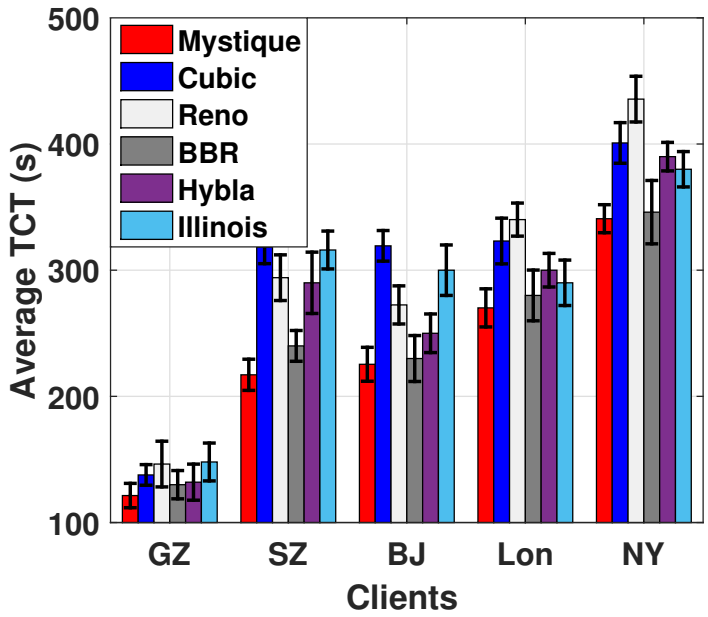
- Transfer Completion Time (TCT) and throughput
  - Transfer time and throughput of both large file (1.2MB) and small file (95.9MB)
- Loading time of websites
  - Loading time of five websites, e.g., CNN, Guardian, Stack Overflow, Walmart, Yahoo.

- We compare Mystique with Cubic, Reno, BBR, Hybla, and Illinois, respectively.
  - Run corresponding stack on Web server kernel
  - Mystique
    - Add additional one hop to Web server
    - Run Algorithm1 for dynamic switching
    - Run Cubic stack on the Web server

Deployment in VMs



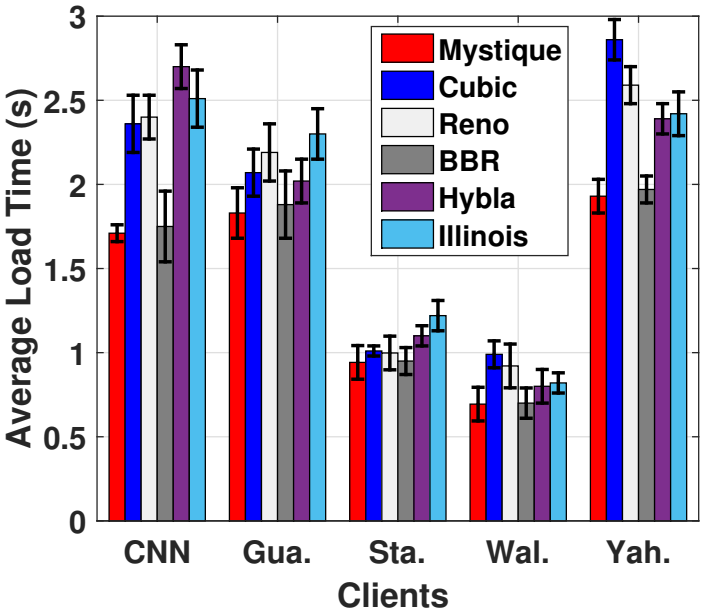
(a) Small file in private cloud



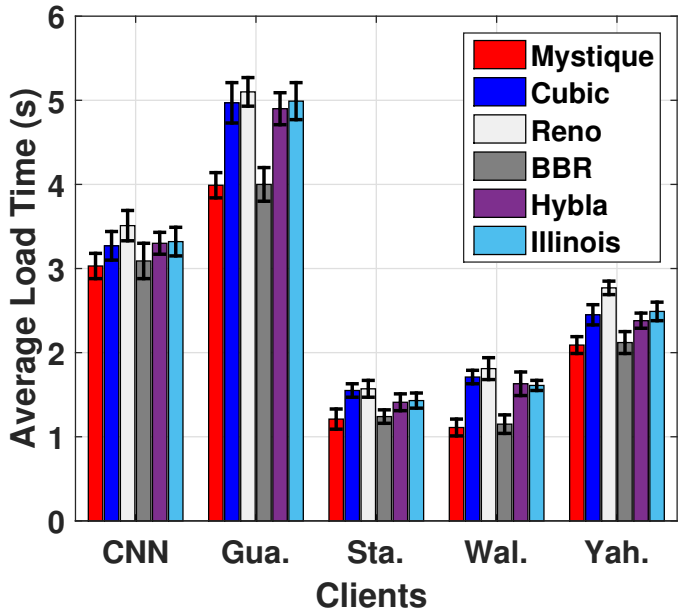
(b) Large file in private cloud

Mystique outperforms Cubic more than **20%** on average, even with additional one-hop delay.

Deployment in Hypervisors



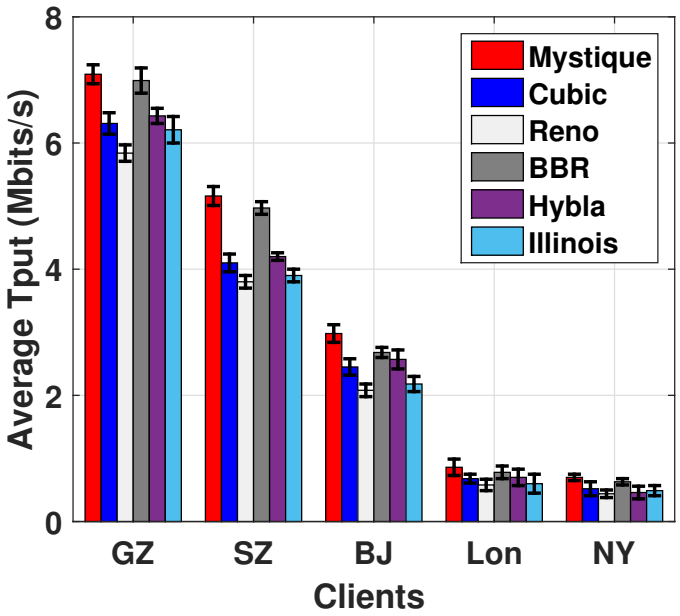
(a) Private cloud



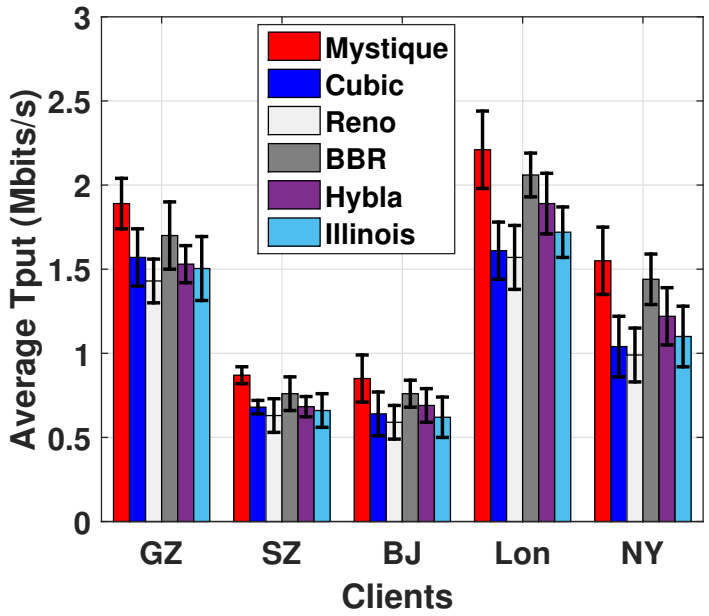
(b) Public cloud

Mystique on Hypervisors reduces
the load time by up to **28.75%**, compared to Cubic.

Deployment in Routers/Switches



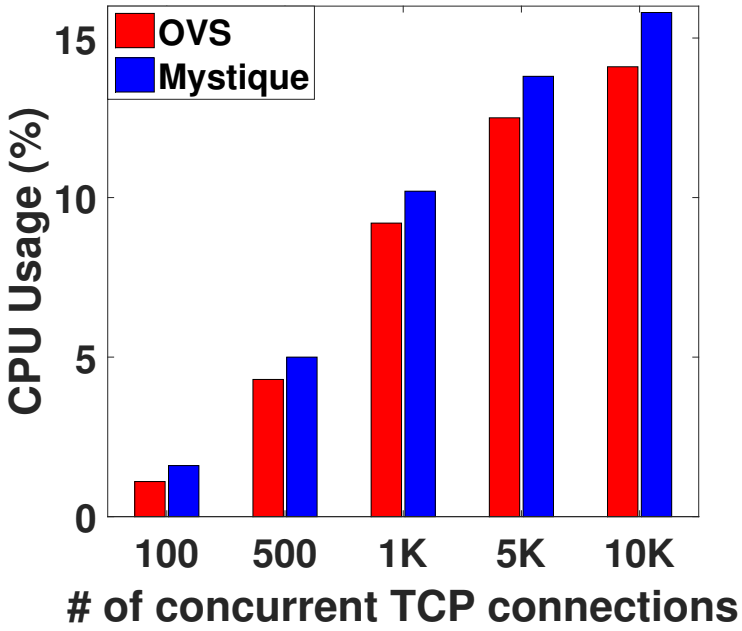
(a) Private cloud



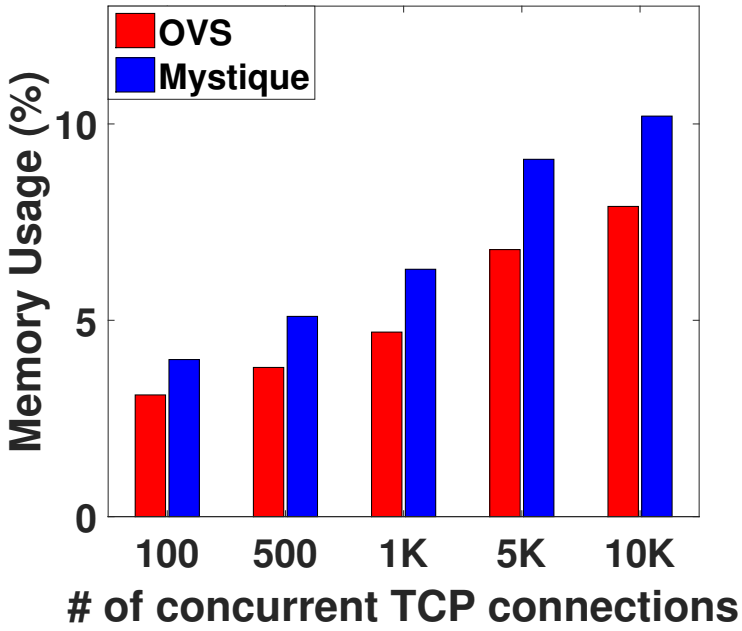
(b) Public cloud

Mystique on Routers/Switches increases the throughput by up to **20.38%**, compared to Cubic.

Overhead



(a) CPU overhead



(b) Memory overhead

CPU: The largest difference is **2**%. Memory: The largest difference is **3**%.

Conclusion

- Mystique enforces more appropriate congestion control algorithm for corresponding network environment
- Extensive test-bed results have demonstrated the effectiveness of Mystique
- We plan to investigate and implement more newly proposed congestion controls
- We would like to hear about your comments too

Questions and suggestions are welcome

Thank You

@fptso
p.tso@lboro.ac.uk
https://www.poscotso.com