

Keynote Speech at the 13<sup>th</sup> International Conference on Evolving Internet  
(INTERNET 2021)  
July 18 – 22, 2021, Nice, France

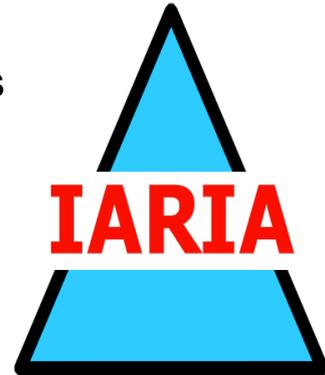
## Machine Learning for Distributed Software Networks (SDN)

Kin K. Leung

Electrical & Electronic Engineering, and Computing Departments  
Imperial College, London

[www.commsp.ee.ic.ac.uk/~kkleung](http://www.commsp.ee.ic.ac.uk/~kkleung)

**Acknowledgment:** Spike Zhang (Amazon; formerly Imperial),  
Liang Ma (Dataminr; formerly IBM U.S.), Paul Pritz (Imperial),  
Kostas Poularakis and Leandros Tassiulas (Yale University),  
and Elisa Bertino (Purdue University)



Queen's Tower  
Imperial College

## Keynote Speaker: Professor Kin K. Leung

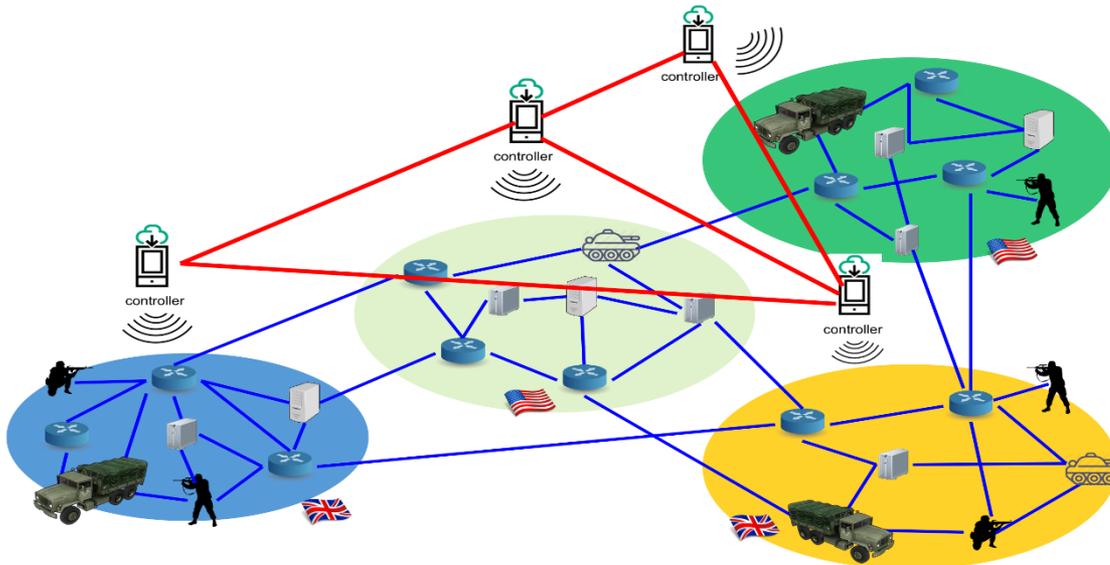


Professor Kin K. Leung  
Imperial College

- **Education and employment**
  - Ph.D. from University of California, Los Angeles (1985)
  - AT&T / Lucent Technologies Bell Labs (1986-2004)
  - Imperial College London (2004 – now)
- **Research interests**
  - Communications and computer networks
  - Machine learning for communications/computer networks
  - Wireless communications and cross-layer designs
  - Optimization, stochastic models and queueing theory
- **Major honors and awards**
  - IEEE Fellow, IET Fellow, Member of Academia Europaea
  - Bell Labs Distinguished Technical Staff Award (1994)
  - Royal Society Wolfson Research Merits Award (2004-09)
  - IEEE ComSoc Leonard G. Abraham Prize (2021)
  - Best paper awards at IEEE ICC 2019, ICDCS 2013, PIMRC 2012
  - Chairman, IEEE Fellow Evaluation Committee for ComSoc (2012-15)
  - Editor for 10+ IEEE and ACM journals

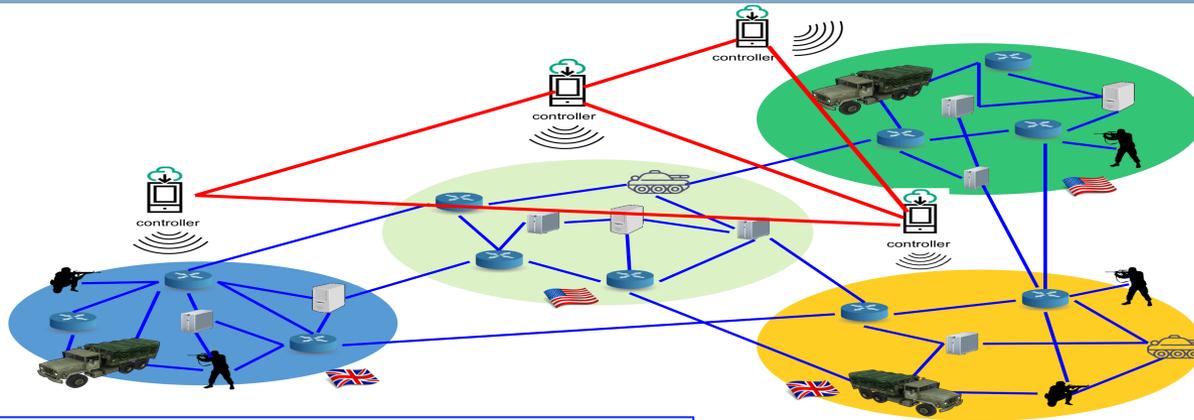
## U.S.-UK International Technology Alliance (ITA)

- ITA Programs Sponsored by U.S. Army & UK Dstl
  - Consortium leader: IBM U.S. & UK
  - NIS ITA: \$92M (2006-16)
  - DAIS ITA: \$40M (2016-21)
  - Now, 14 universities and industrial companies in U.S. & UK
- **Aim:** Develop new communications and computation infrastructures for **coalition defense operations**



- Software defined coalition (SDC)
  - Communications
  - Servers
  - Storage
  - Databases
- Secure, mobile, distributed analytics
- Edge computing

## Software Defined Coalition (SDC)



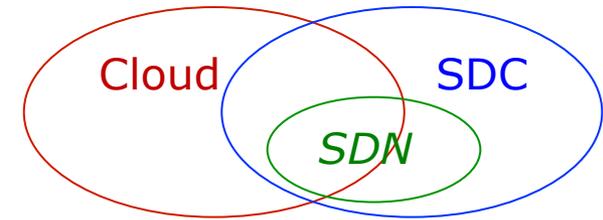
- Software Defined Coalition (SDC)
  - Architecture proposed by DAIS ITA Program <https://dais-ita-org/pub>
  - Extension of Software Defined Network (SDN), which focuses on communications
  - Resources (e.g., communication links, servers, storage) are grouped into domains
  - Resources in each domain are monitored and shared by a single domain controller, changeable through software, thus providing re-configurability and adaptability
  - Connect domains from owners to form SDC for a set of applications

- Key Technical Challenges for SDC
  - Controller synchronization: Domain controllers need to exchange status info for resource sharing across domains
  - Techniques for resource allocation and sharing across domains
  - Use SDC for machine learning applications
  - See SDC article by Kin Leung at <https://dais-ita.org/pub>

# What Is a Cloud?

Example cloud-powered applications:

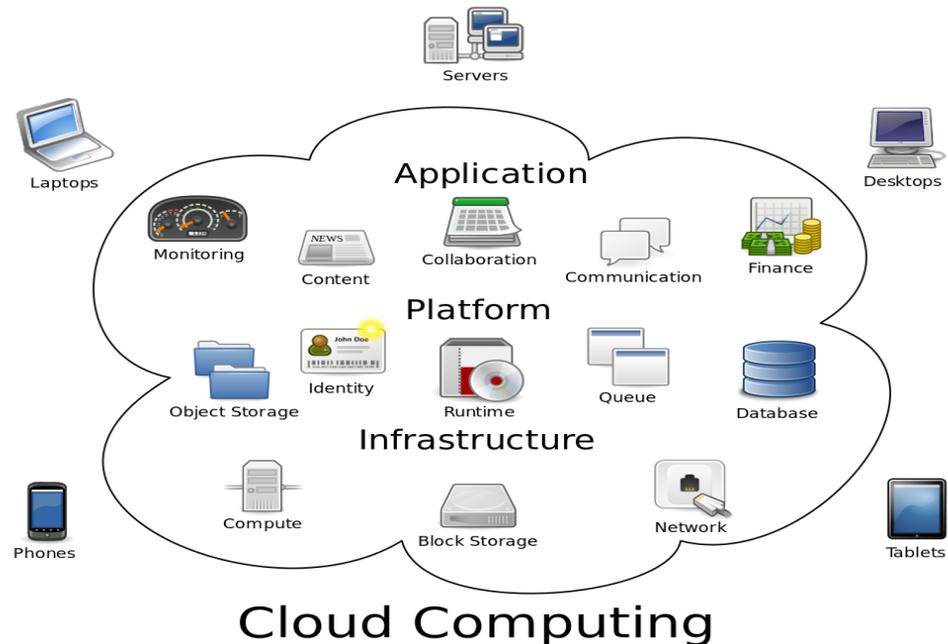
- Google map (map service)
- Dropbox (storage & content sharing)
- YouTube & Netflix (video streaming + encoding/decoding)



The NIST definition:

“**Cloud computing** is a **model for enabling** ubiquitous, convenient, on-demand **network access to a shared pool of configurable computing resources** (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.”

Source:  
<http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf>



Source: [https://en.wikipedia.org/wiki/Cloud\\_computing](https://en.wikipedia.org/wiki/Cloud_computing)

## Markov Decision Process (MDP) for Reinforcement Learning (RL)

- Markov Decision Process (MDPP)  $(S, A, P, R, \gamma)$

- $S$  : Set of states,  $A$ : Set of actions
- $P$  : State transition probability matrix

$$P_{SS'}^a = \mathbf{P}[S_{t+1} = s' \mid S_t = s, A_t = a]$$

- $R$  : Reward function

$$R_S^a = \mathbf{E}[R_{t+1} \mid S_t = s, A_t = a]$$

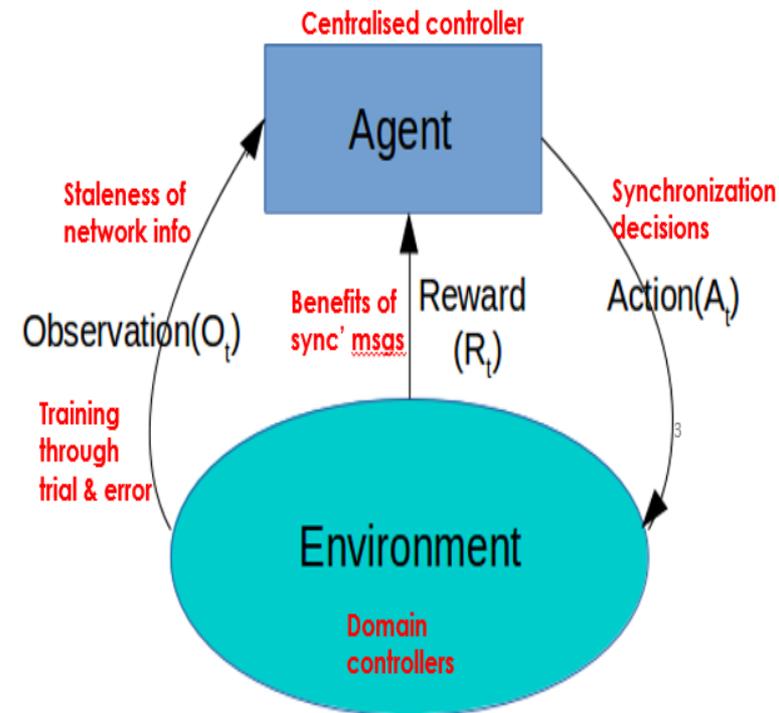
- $\gamma$ : Discount factor between 0 and 1

- RL: Learning and control

- Model-based or model free

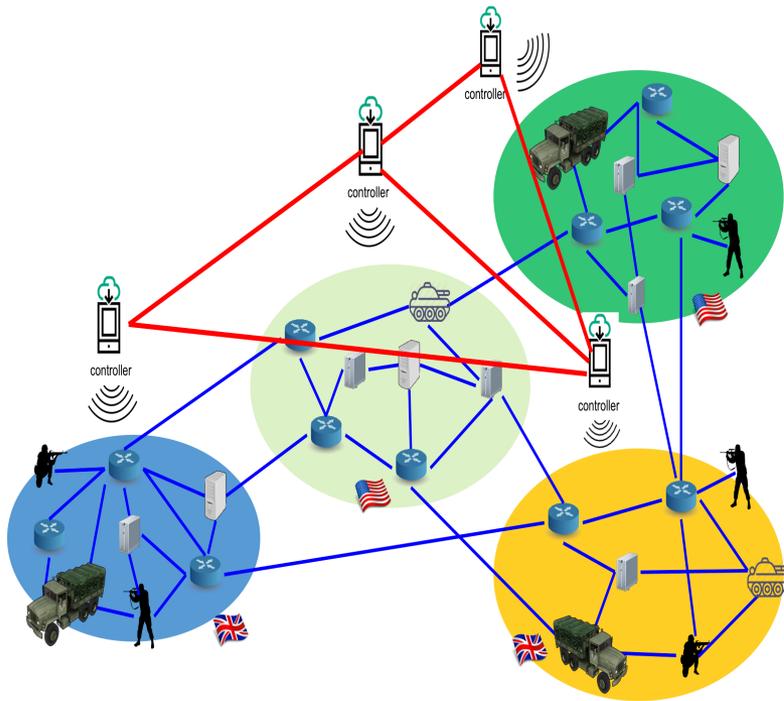
- Major problems

- Explosion of state-action space
- Require huge datasets & time for deep Q-learning and policy designs



*Possible solutions: State-action separation, state-action embedding, state decomposition, MDP decentralization, hierarchical RL, etc.*

# SDN / SDC Controller Synchronization



SDC: Software Defined Coalition

Each domain consists of network elements (e.g., switches, links, servers, databases) within a given administrative control

Software-defined Coalition (SDC) consists of a set of connected domains

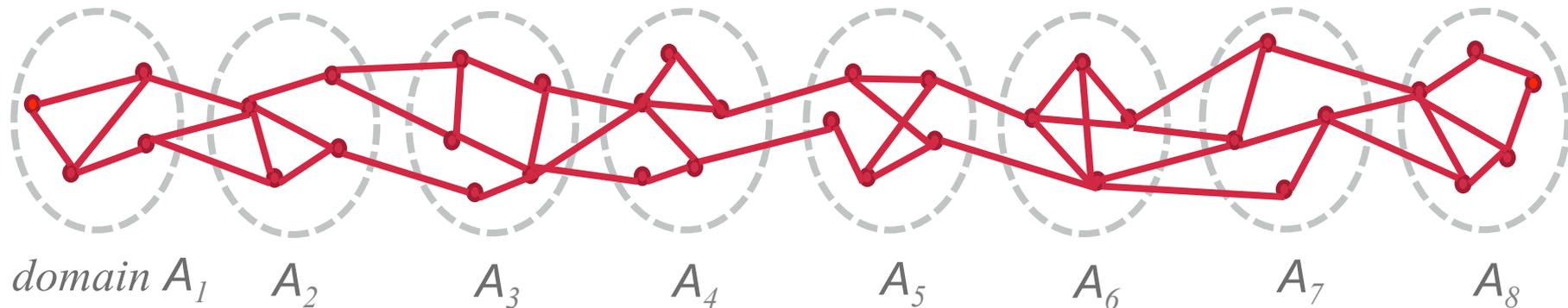
Network control functionalities reside on domain controllers, one for each domain => ***distributed SDC***

SDN domain controllers update each other from time to time with their current domain status => ***controller synchronization***

Always up-to-date synchronization among controllers infeasible due to high overheads

***Question:*** *What is a good controller synchronization policy (when and which controllers to sync) for the given performance metric and synchronization budget?*

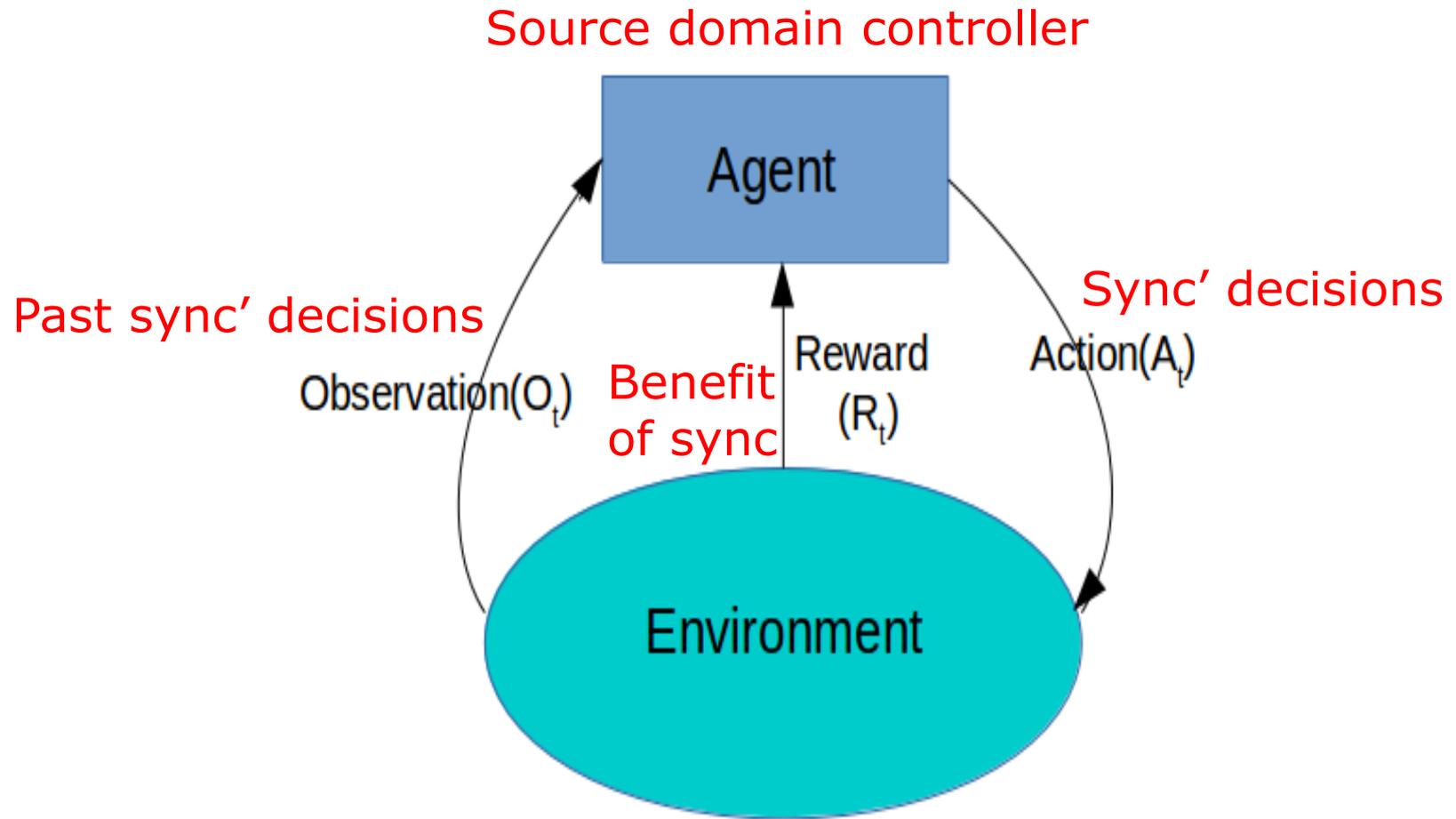
## Controller Synchronization: A Routing Example



- A source node in  $A_1$  constantly sends data to a destination node in  $A_8$
- Controller  $A_1$  constructs the path and establish a forwarding rule according to its view of the network
- Topologies and link conditions in all domains change over time, but at different rates (e.g., 5 edges rewire per second)
- Controller in  $A_1$  can only sync with **one** other domain every several seconds

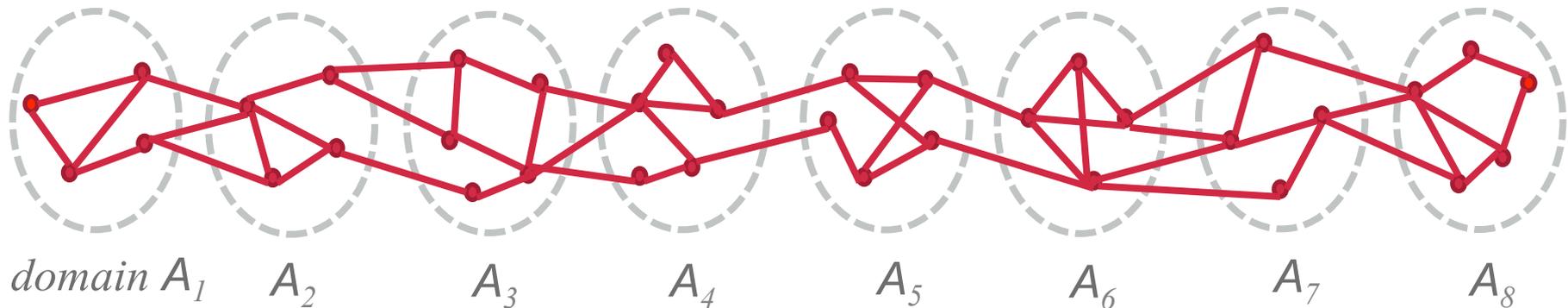
**Question:** How does the controller in  $A_1$  decide **which domain and when to synchronize with** for optimizing routing performance (e.g., minimizing the number of hops in communication path)?

## How can RL help controller synchronization?



Domains with dynamic topological changes

## Markov Decision Process (MDP) Formulation



**State:** Vector of time elapsed since last time  $A_1$  synchronizes with other domains

5	10	15	5	35	20	40
---	----	----	---	----	----	----

**Action:** Which domain to sync with?

0	0	0	0	1	0	0
---	---	---	---	---	---	---

**Immediate reward:** Reduction in transit delays after synchronization

The goal of the agent: **Maximize**

$$V(s_0) = \mathbb{E} \left[ \sum_{t=0}^T \gamma^t R(s_t, a_t) | s_0 \right]$$

## Q-function for evaluating a synchronization policy

- Specifically, the value (Q) function following policy  $\pi$  looks like this

$$\begin{aligned} Q^\pi(s, a) &= \mathbb{E} [r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \dots \mid s, a] \\ &= \mathbb{E}_{s'} [r + \gamma Q^\pi(s', a') \mid s, a] \end{aligned}$$

- Iterative value update to obtain the optimal Q-function

$$Q_{i+1}(s, a) = \mathbb{E}_{s'} \left[ r + \gamma \max_{a'} Q_i(s', a') \mid s, a \right] \quad \leftarrow \text{Bellman equation}$$

- The optimal value function for each state-action pair is

$$Q^*(s, a) = \mathbb{E}_{s'} \left[ r + \gamma \max_{a'} Q^*(s', a') \mid s, a \right]$$

Maximize:

$$V(s_0) = \mathbb{E} \left[ \sum_{t=0}^T \gamma^t R(s_t, a_t) \mid s_0 \right]$$

## Approximate Q-function by Deep Neural Network (DNN)

- Excessive size of the state-action space makes it impossible to store Q-values in a tabular setting
- Use **Deep Neural Network (DNN)** with weights  $w$  to approximate the Q-function

$$Q(s, a, w) \approx Q^\pi(s, a)$$

- Then, the goal is to train weights  $w$  of the DNN so that

$$Q(s, a, w) \approx Q^\pi(s, a) \quad \rightarrow \quad Q^*(s, a)$$

- The optimal policy is found when the following loss function (gap) is minimized close to 0

$$\mathcal{L}(w) = \mathbb{E} \left[ \left( \underbrace{r + \gamma \max_{a'} Q(s', a', w)}_{\text{target}} - Q(s, a, w) \right)^2 \right]$$

# Experimental Evaluation of RL Controller Synchronization

- **Three scenarios** with 6, 10, and 12 domains, respectively
- **Domain topologies** generated by using data extracted from Rocketfuel dataset
- Different patterns of dynamic link changes for domains
- **Performance benchmarks** for comparison
  - Anti-entropy algorithm (implemented in ONOS controller)
  - Fixed synchronization period
- Performance metric
  - **Accumulated time-discounted transit delay reductions over time**

# Evaluation Results

Compared with anti-entropy and fix sync, the proposed RL is:

31% & 91%  
better

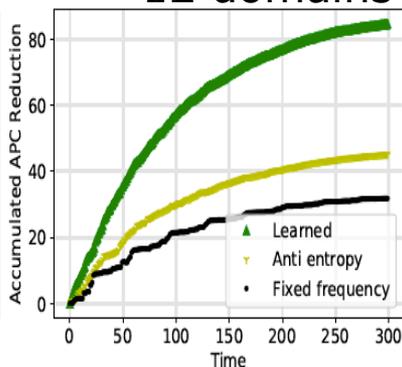
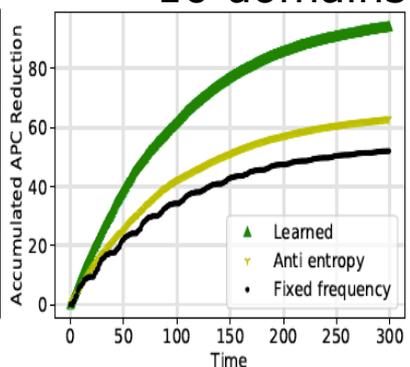
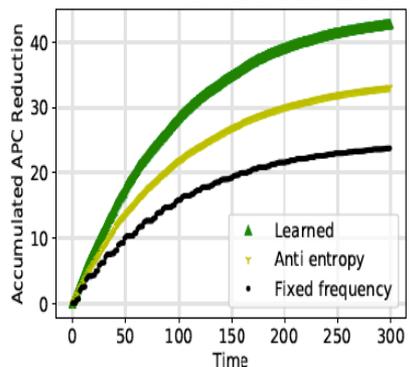
58% & 90%  
better

95% & 173%  
better

6 domains

10 domains

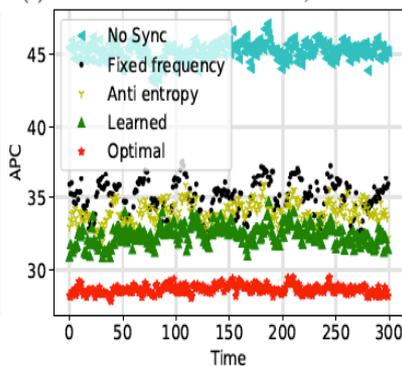
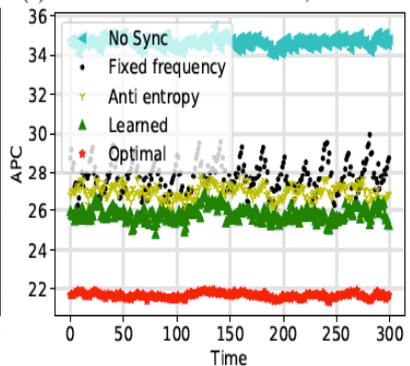
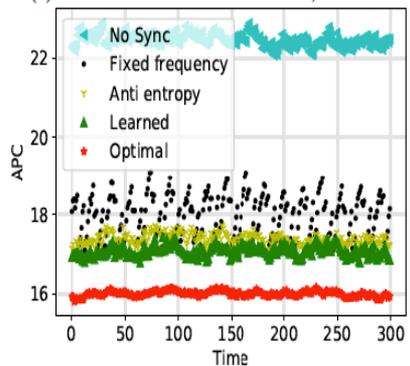
12 domains



(a) Accumulated APC reduction,  $m = 6$ .

(b) Accumulated APC reduction,  $m = 10$ .

(c) Accumulated APC reduction,  $m = 12$ .



(d) Immediate APC,  $m = 6$ .

(e) Immediate APC,  $m = 10$ .

(f) Immediate APC,  $m = 12$ .

Fig. 3: Evaluation results.

Optimization Goal

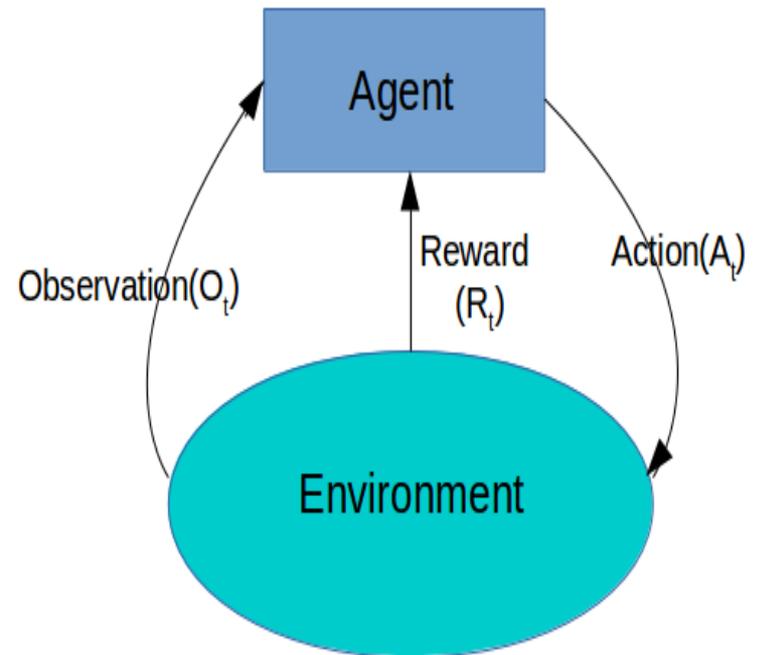
$$V(s_0) = \mathbb{E} \left[ \sum_{t=0}^T \gamma^t R(s_t, a_t) | s_0 \right]$$

Observations:

- Superiority of the DQ scheduler for long and short-term routing quality
- Performance degradation due to lack of synchronization is more severe when domain-wise path is longer, as “no sync” performance are worsened by 37.5%, 59.1%, and 61%, respectively

## RL Challenges and Possible Solutions

- **Big Challenge of RL** for control of large communications and computer infrastructures
  - **Huge** state and action spaces
  - **Excessive** training / learning time, if possible
- Solution techniques include
  - **State-action separable RL (sasRL)**
  - Joint state-action embeddings
  - State space decomposition
  - Decentralized MDP
  - **Hierarchical RL**
  - .....



# Background: Serial Decision-Making Problems and RL-based Solutions

An example serial decision-making problem



*RL agent's interaction with the env*

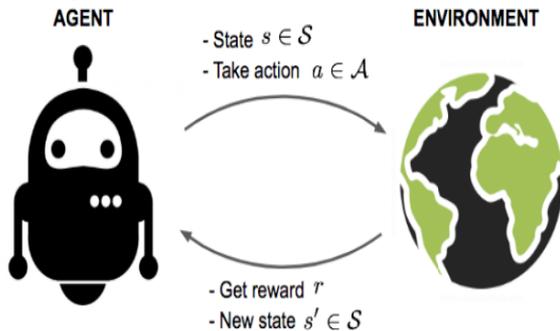


Image credit:  
<https://lilianweng.github.io/>

- Serial decision-making problems can be described by a series of tuples consisting of the following elements
  - State: the agent's perception of the environment
  - Action: what the agent does (the agent's action policy)
  - Reward: what the agent earns after taking actions
- RL methods guide the agent to behave in certain ways by **reinforcing good behaviors (actions) and penalizing bad ones** (think about how human/animals learn)
- Conventionally, the goal of RL methods is to maximize the cumulative reward received by the agent
- Two types of RL method
  - Model-based: Explicitly use a model of the environment to assist the agent's learning process (e.g., planning based on dynamic programming)
  - Model-free: It does not require nor depend on any explicit environment model (most mainstream RL algorithms)

# Conventional Approach to RL Problems

## Conventional Approach to RL Problems

- Value function: Estimate the goodness of RL agent's behaviors
- Conventionally, State + Action  $\rightarrow$  Reward;  
State-Action-Value (SAV) function is used:

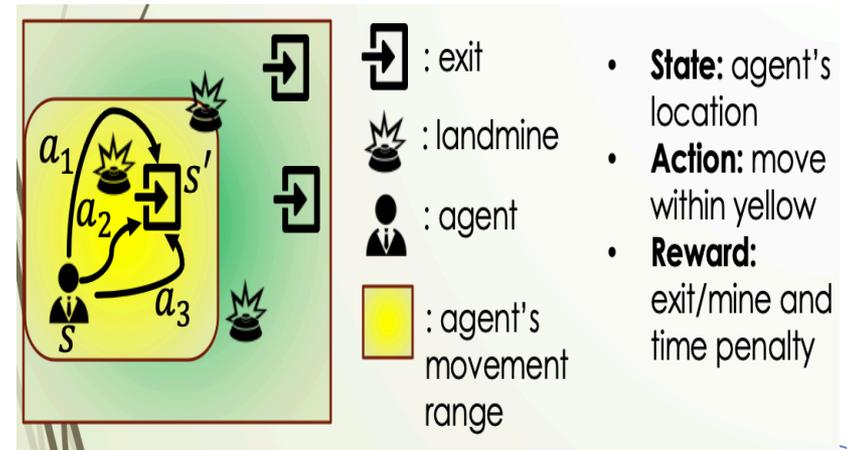
$$G_t = R_{t+1} + \gamma R_{t+2} + \dots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

$$q_{\pi}(s, a) = \mathbb{E}_{\pi} [G_t \mid S_t = s, A_t = a]$$

- Why most existing RL algorithms use the SAV function to derive and evaluate policies?
  - State-Action pairs directly define the environment and the agent's behaviors
  - Implicitly, it is assumed that the reward of a state transition is a function of current state and action, which is generally true

## What is the problem using SAV function?

- Enormous state action space (combinatorial increase in size)
- For many problems, the next state of the state transition matters more than the action for determining rewards
- For instance, think about the following example



# A New Value-Function Approach for RL Problems

## Our proposal: State Transition-based Value Function for RL Problems

- State + Next state  $\rightarrow$  Reward; State-Transition-Value (STV) function
- STV function: Estimate the goodness of state-next state combinations

$$\mathcal{R} : \mathcal{R}_{ss'} = \mathbb{E}[r_{t+1} | s_t = s, s_{t+1} = s']$$

- STV function better captures the return dynamics for problems where reward is directly related to the state transition, and action only causes the transition

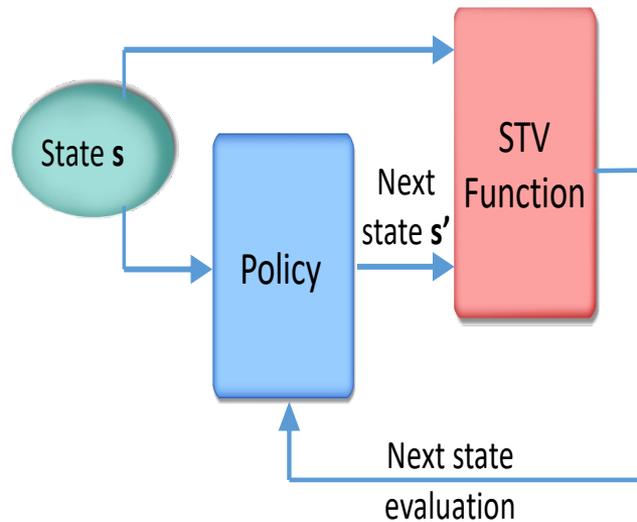
## State-Action Separable RL (sasRL)

- STV function only involves the state space, but not the state-action space  $\rightarrow$  STV simpler than SAV
- The dynamic of state + action  $\rightarrow$  next state dynamic can be learned separately, using simple supervised learning techniques
- The light-weight transition model
  - $\tau_\omega : \mathcal{S} \times \mathcal{S} \rightarrow \mathcal{A}$  parameterized by DNN weights  $\omega$
  - The model is trained via supervised learning to minimize

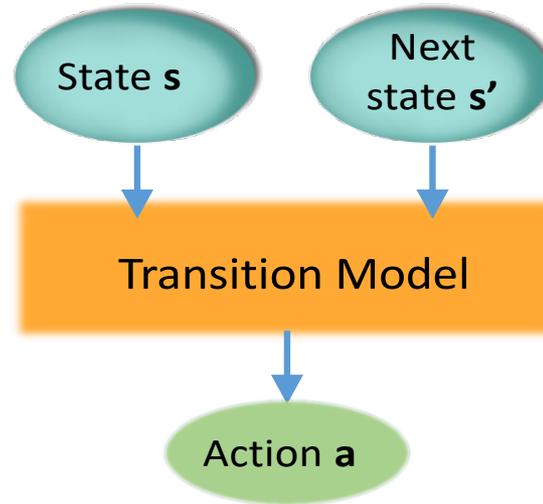
$$\mathcal{L}_\omega = L(\tau_\omega(s, s'), a)$$



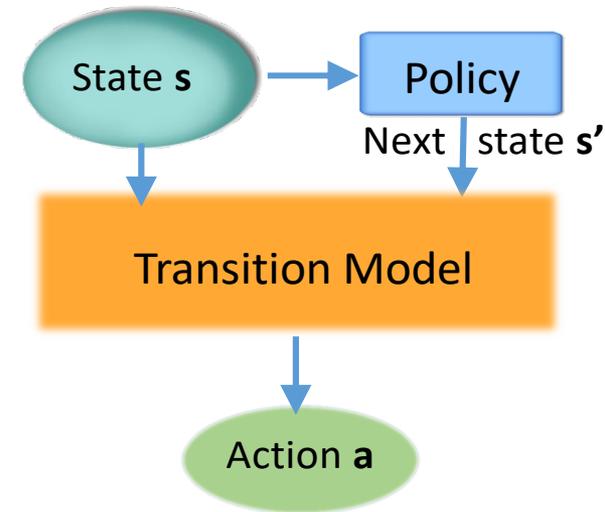
# sasRL Architecture



Model-free RL:  
Train the STV function



Model-based  
supervised  
learning to  
assist RL task



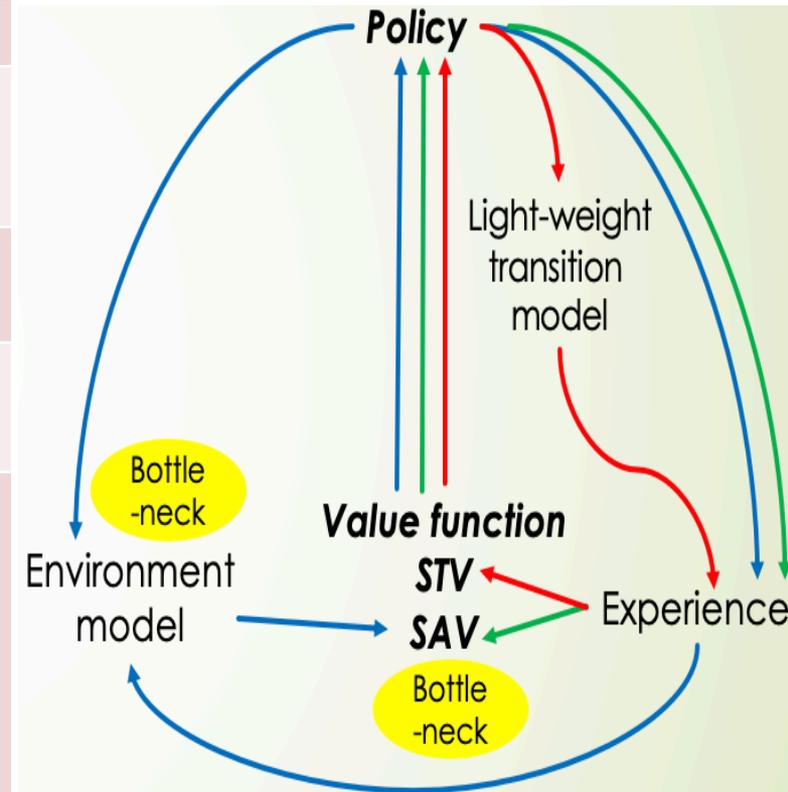
sasRL in operation

State-Action Separable RL (sasRL) transforms a complex RL problem into a simpler RL problem and a simple supervised learning problem



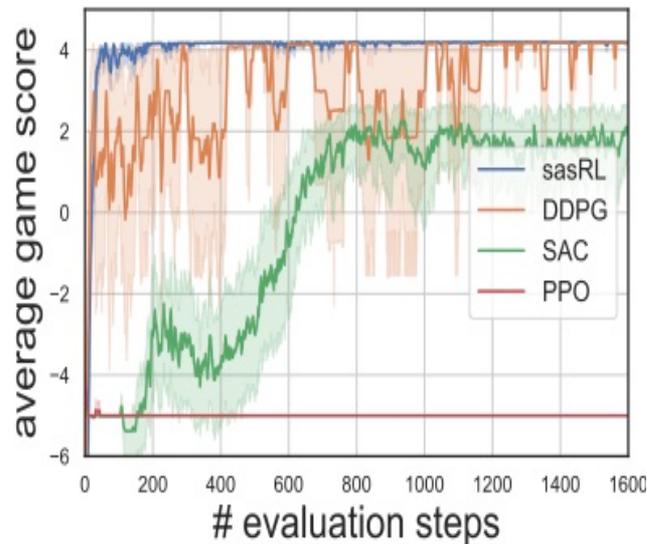
# Comparing sasRL with Other RL Algorithms

	Model-based RL	Model-free RL	sasRL
Symbol			
Env. model needed?	Full env. model	No	Light-weight transition model
Value function	SAV	SAV	STV
Learning bottleneck	SAV, env. model	SAV	n/a
Pros	1) Less interactions with env. 2) Fuller use of data	1) Simple, direct 2) No bias (caused by env. model)	Address 2 bottlenecks by: 1) define & employ more efficient STV function 2) a light-weight transition model
Cons	Env. model difficult to train	Inefficient	
Examples	Dyna-Q	Q-learning, DQN, DDPG, SAC	

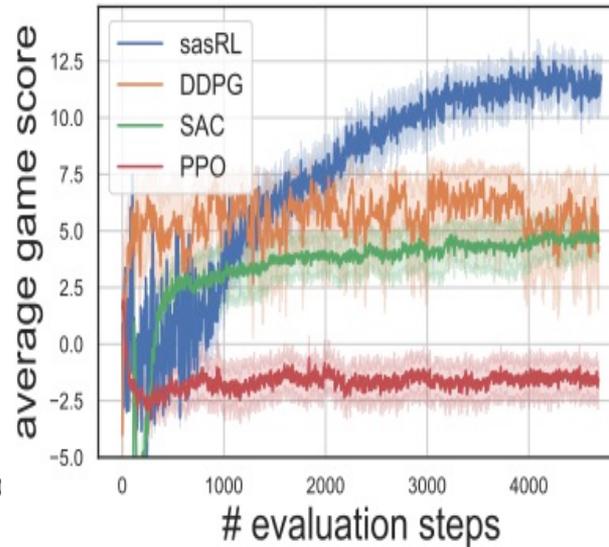


## Comparative Evaluation Results

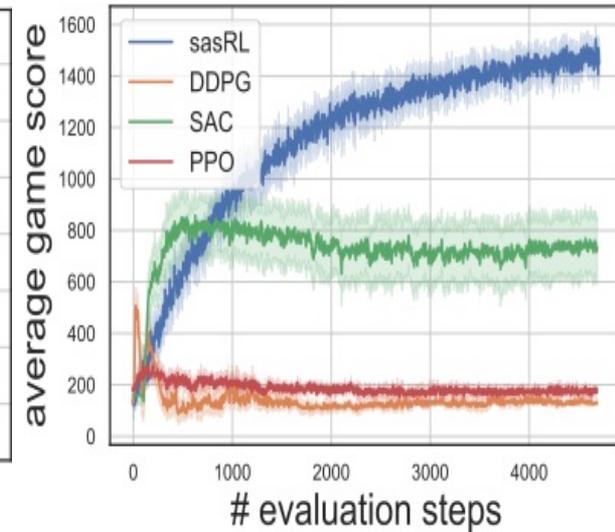
*Gird world exit*



*Berzerk*



*Slot machine*



- Remarks on comparative evaluation results
  - **sasRL** performance is consistently superior
  - **PPO** (Proximal Policy Optimization Algorithm) fails in all three scenarios
  - **SAC** (Soft Actor-Critic) produces the most stable results on average
  - **DDPG** (Deep Deterministic Policy Gradient) outperforms SAC in two cases, but is unstable and brittle

## sasRL+Transfer Learning for Environment Changes

- **Joint Reinforcement and Transfer Learning (RL+TL)**
  - Consider **SDC fragmentation with 2 domains, focusing on data servers**
  - Combine RL (e.g., sasRL) and TL based on generative adversary network (GAN) to synthesize data for learning in new environments
  - Combined RL+TL can **significantly speed up RL** when operating environment changes (e.g., SDC domain fragmentation and re-connection)



Note:

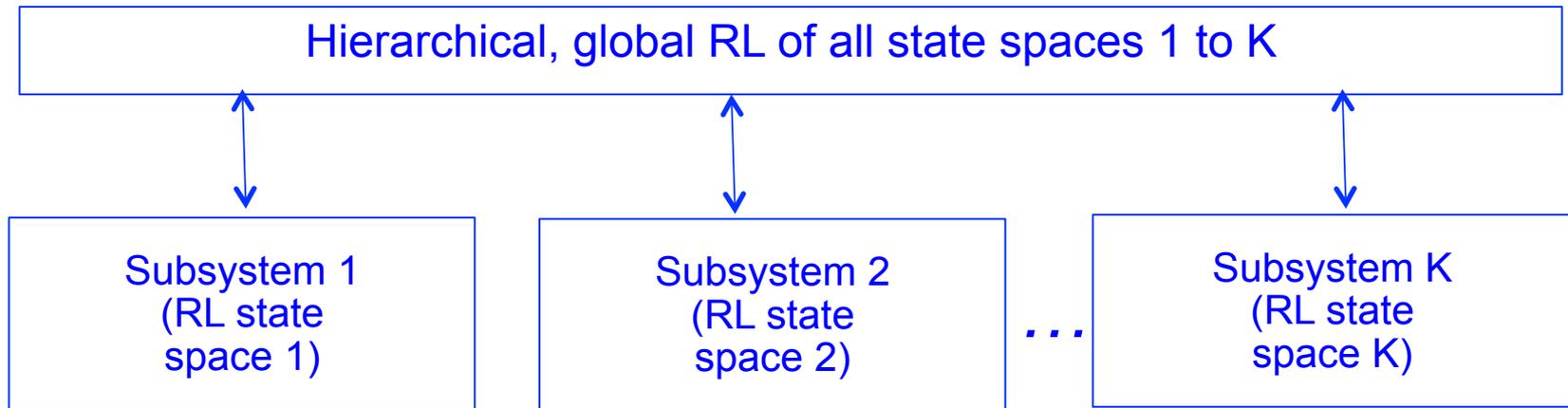
- The reward is inversely proportional to the service delay
- Real Explorations = 10,000 data samples
- Augmented (RL+TL) or Limited Explorations = 100 data samples (1% of Real Exploration sample size)

## Summary of Key Features of sasRL

- **RL problem formulation under sasRL:** modified Markov Reward Process (mMRP), in contrast to commonly used MDP
- **Value function:** state-transition-value (STV) function, instead of commonly used state-action-value-function (SAV) function
- **Convergence property analysis:** STV function's convergence time is  $O(T^{1/k})$ , where T is the convergence time for updating SAV function, and k is a weighting factor
- Model-free learning for the STV function, model-based learning for the transition model; **two training process can take place separately** (thus enables parallelism)
- sasRL **enables the concurrent optimizations** of both STV function and policy (e.g., suitable for the actor-critic framework)
- The STV function, policy and transition model for sasRL can be **approximated by DNNs**
  - DNN for STV function is easier to train than that of SAV function, due to reduced input vector space size
  - The transition model is easy to train via mature and standard supervised learning routines
- In summary, sasRL achieves better performance by breaking down an inefficiency-prone model-free RL problem into a more sample efficient and easier to train model-free RL problem and a simple supervised learning problem



# Hierarchical RL for Control of Large System



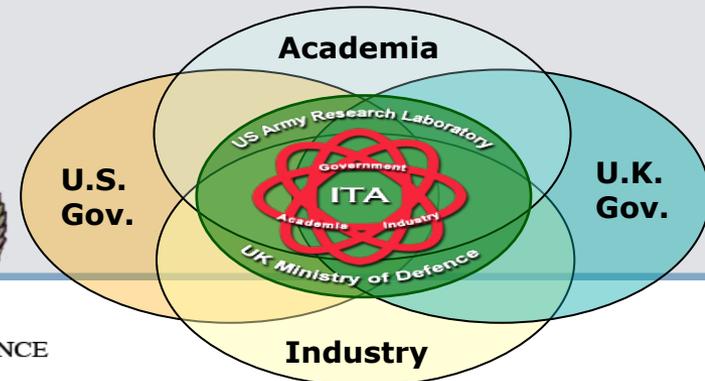
- SDC is a large system of many subsystems (domains), each subsystem with its own state subspace controlled by an RL agent
- The global RL agent interacts with subsystem agents to control the whole system – tightly-coupled hierarchical RL
- Open issues:
  - Compress states and actions to avoid space explosion for the global agent
  - Control policies for multi-agents



# Acknowledgments



MINISTRY OF DEFENCE



## Acknowledgments

- Spike Zhang (Amazon; formerly Imperial), Liang Ma (Dataminr; formerly IBM U.S.), Paul Pritz (Imperial), Kostas Poularakis and Leandros Tassiulas (Yale University), and Elisa Bertino (Purdue University)
- Research funding: U.S./U.K. ITA Project

## Publications

1. Z. Zhang, L. Ma, K.K. Leung, F. Le, S. Kompella and L. Tassiulas, “How Advantageous Is It? An Analytical Study of Controller-Assisted Path Construction in Distributed SDN,” IEEE/ACM Trans. on Networking, 2019.
2. Z. Zhang, L. Ma, K.K. Leung, and F. Le, “More Is Not Always Better: An Analytical Study of Controller Synchronizations in Distributed SDN,” IEEE/ACM Trans. on Networking, 2021.
3. Z. Zhang, L. Ma, K. Poularaki, K.K. Leung and L. Wu, “DQ Scheduler: Deep Reinforcement Learning Based Controller Synchronization in Distributed SDN,” IEEE ICC, China, June 2019. (Best Paper Award)
4. Z. Zhang, L. Ma, K Poularakis, K.K. Leung, J. Tucker and A. Swami, “MACS: Deep Reinforcement Learning based SDN Controller Synchronization Policy Design,” IEEE ICNP, USA, October 2019.
5. Z. Zhang, L. Ma, K.K. Leung, K. Poularakis, and M. Srivatsa, “State Action Separable Reinforcement Learning,” IEEE BigData 2020.
6. Z. Zhang, A. Mudgerikar, A. Singla, K. Leung, E. Bertino, D. Verma, K. Chan, J. Melrose, and J. Tucker, “Reinforcement and Transfer Learning for Distributed Analytics in Fragmented Software Defined Coalitions,” SPIE, April 2021, Florida.
7. P. Pritz, L. Ma and K.K. Leung, “Joint Sate-Action Embedding for Efficient Reinforcement Learning,” submitted for publication.

Please google “Kin K Leung” for my website to find these and other papers.



*Thank you*

