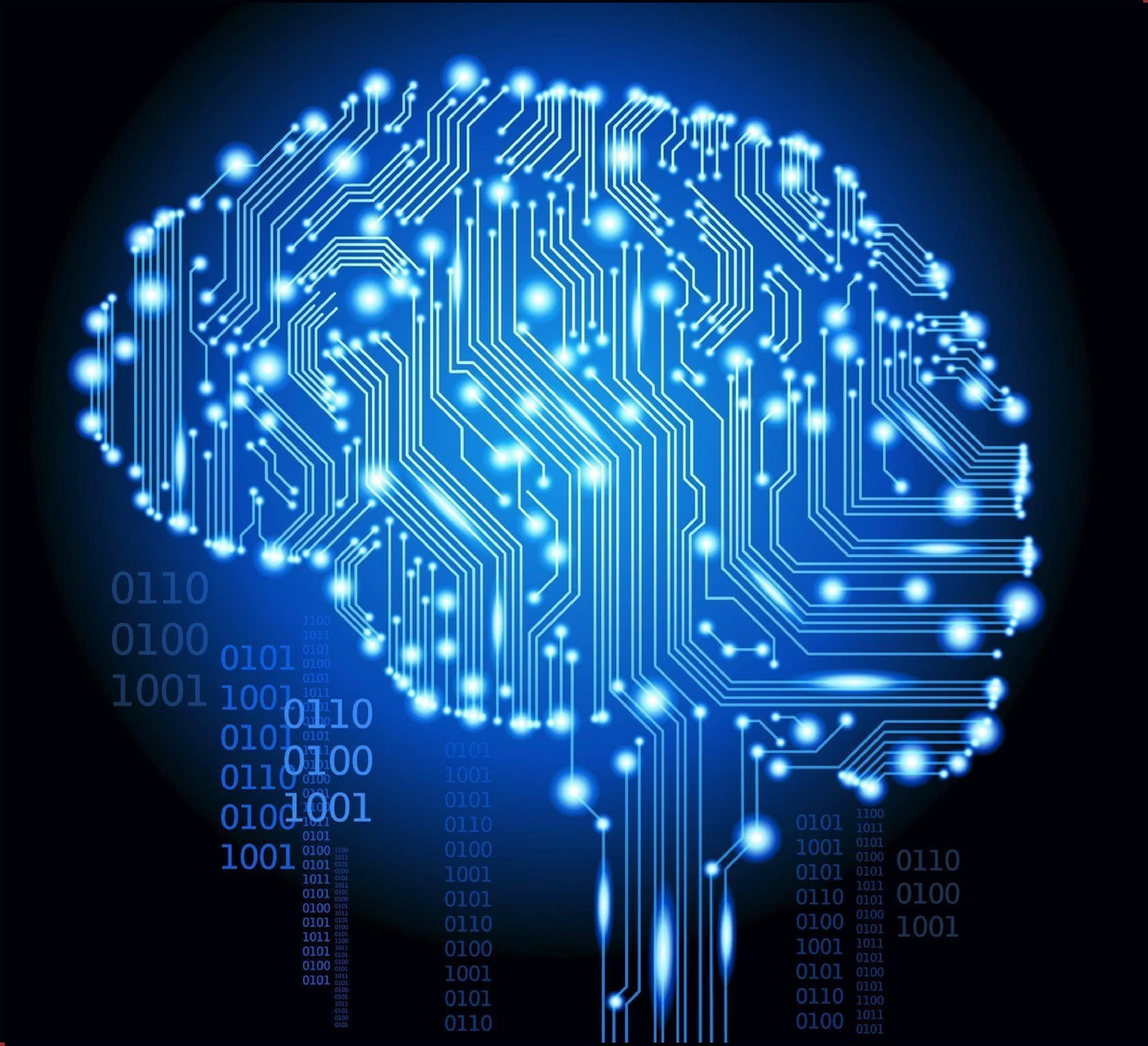
BTP PRESENTATION - 01

Reinforcement Learning Based SDN Controller Load Balancing

Department of Mathematics,
Indian Institute of Technology, Guwahati



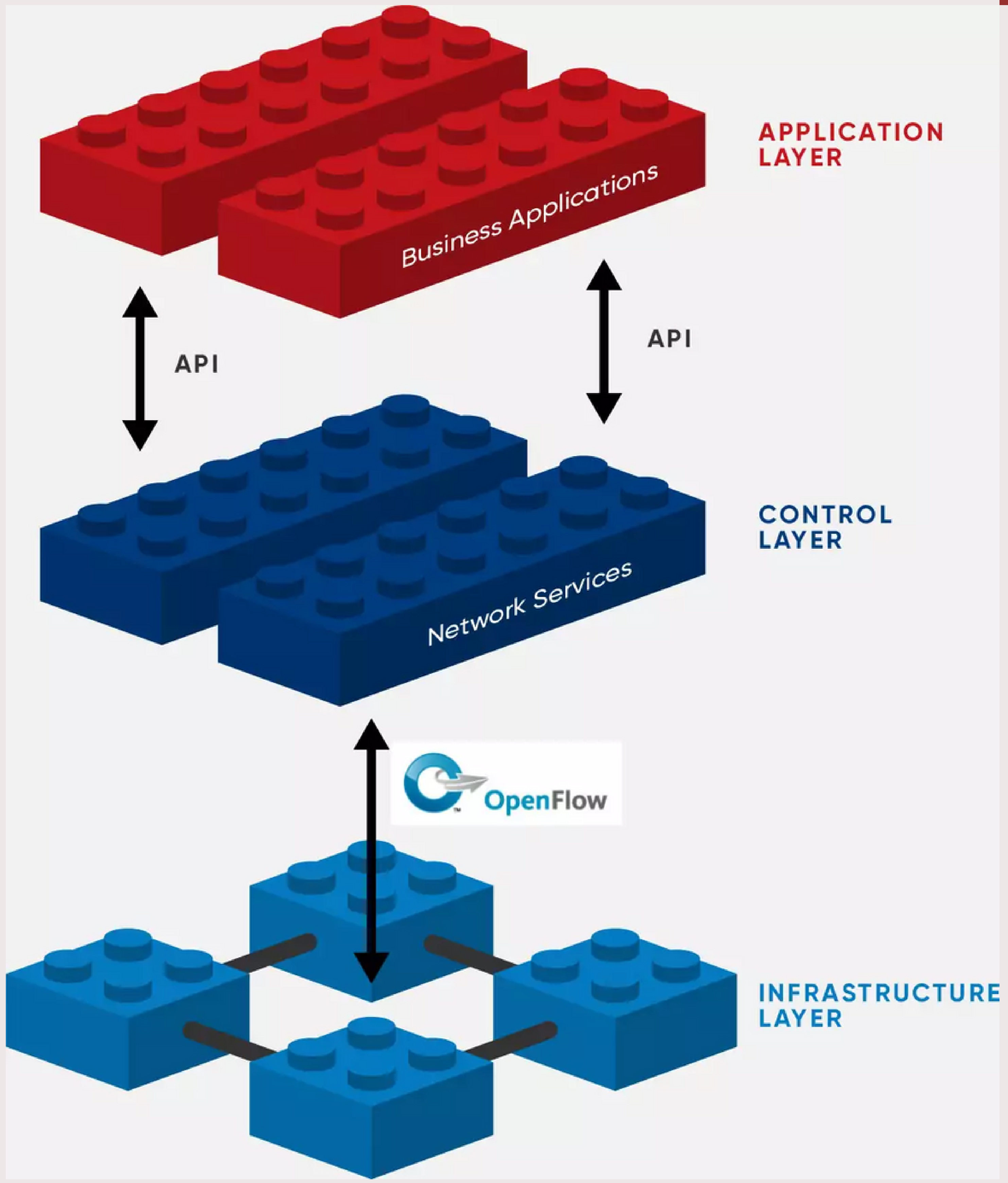
Presentation Outline

What will we discuss about today?

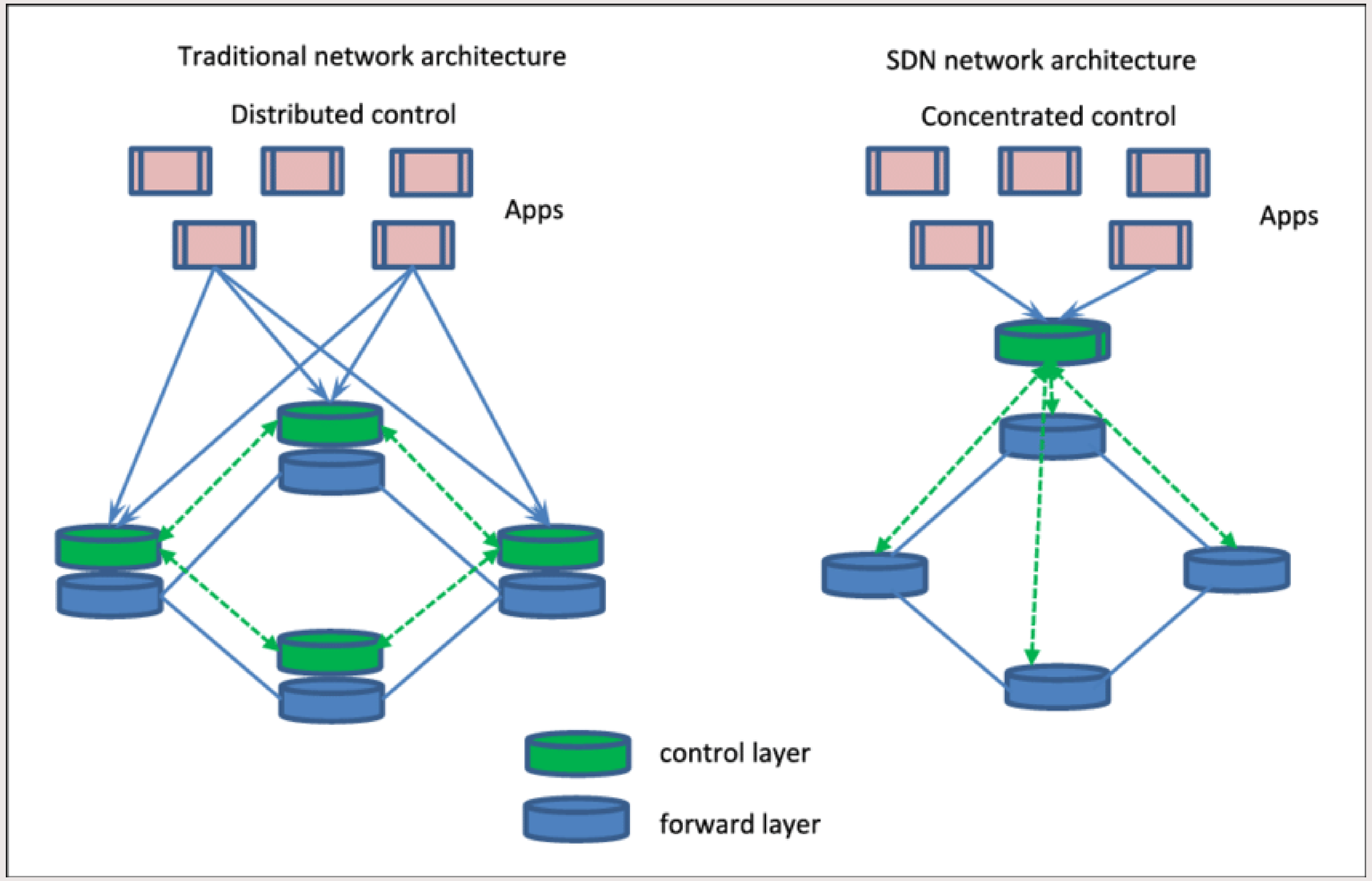
What is SDN?

Software Defined Networking

Software-defined networking (SDN) separates the control plane and data plane of network devices, whose refined centralized control and network programmability bring unique advantages compared with traditional networks. Along with the explosive growth of network traffic and the emergence of network applications of big data, strictly centralized control planes can't meet the requirements of the existing network, the semi-centralized or logically centralized control plane and the fully distributed control plane become the development trend of future enterprise network and carrier network architecture.



How is SDN different from Traditional networks?



Communication between switches (forwarding devices) and controllers occurs through a southbound API, of which OpenFlow is a well-known example.

Northbound API provides communication between the control plane and the application layer.

In large networks, a single controller may not suffice and multiple controllers will be required. EastWest API is used as an interface among the multiple controllers.

Load Balancing in SDN Controllers

The distributed deployment based on multiple controllers **improves the scalability of the control plane** and **avoids single-point failures** in centralized deployment

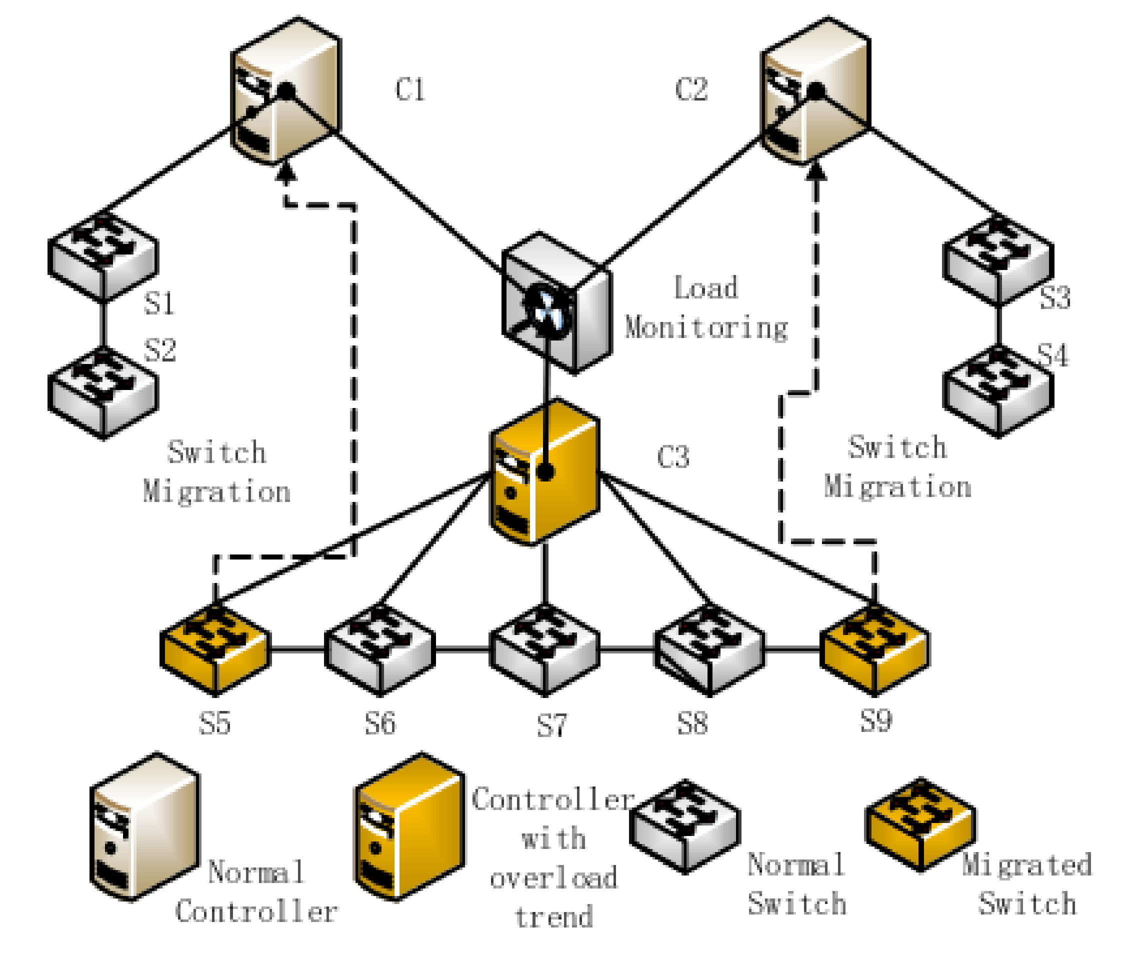


Fig. 1. Controller load diagram.

The load of the SDN controller mainly comes from the following aspects:

(i) **When a controller generates an overload, it is generally because a large number of switches cannot find a corresponding flow entry for the newly arrived packet, thereby send a packet-in message to the default controller.**
(ii) Controllers communicate with each other in the control plane.
(iii) The controller installs flow entries for the switch.
(iv) All controllers maintain and share the same network view of the entire control platform.

The SDN controller load balancing mechanism based on reinforcement learning achieves the following three objectives: <u>(I) better load balancing, (II) without migration conflicts, (III) without new overload.</u>

Switch Migration Design

The **controller load** is the sum of the controller's packet-in message rates.

$$L_{C_i} = \sum_{k=1}^{l} L_{S_k}$$

Different controllers have different CPU performance, the number of processors, memory size, etc., which results in different load capacities. **The controllers' load ratio Rci is the ratio of the controller's real-time load Lci to the controller's load capacity Cci**

If the switch is migrating between controllers, there will be migration overhead. **Because the hop count between the switch and the controller is different, the load of the switch to the controller is different, the migration efficiency is expressed as a ratio of these two.**

The discrete coefficient of all controllers is defined as the degree of the load balance of the entire control plane. The smaller the value is, the better the load balancing of the control plane.

$$D = \sqrt{\left(\sum_{i=1}^{n} \left(R_{C_i} - \bar{R}\right)^2 / n\right)} / \bar{R}$$

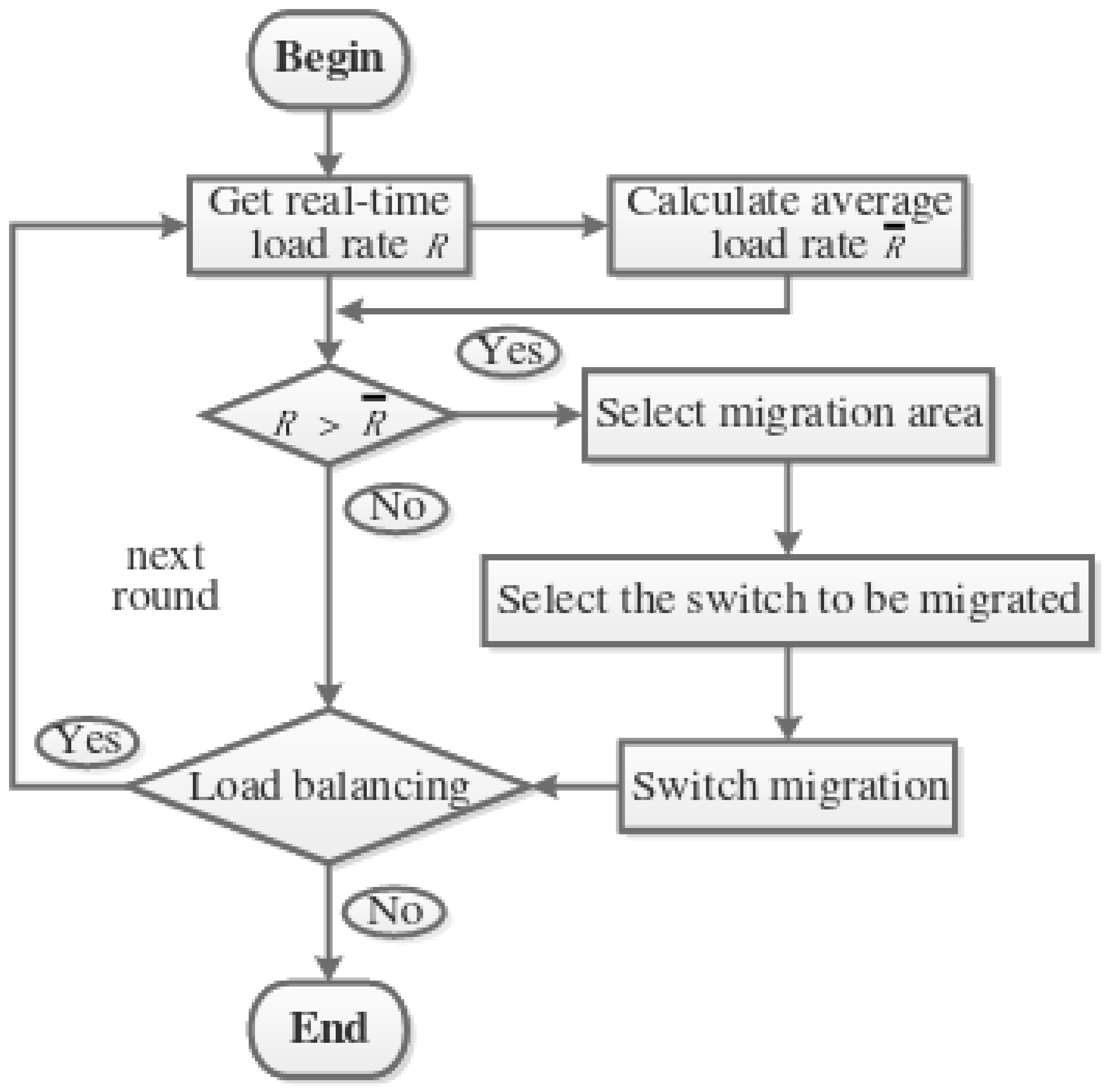
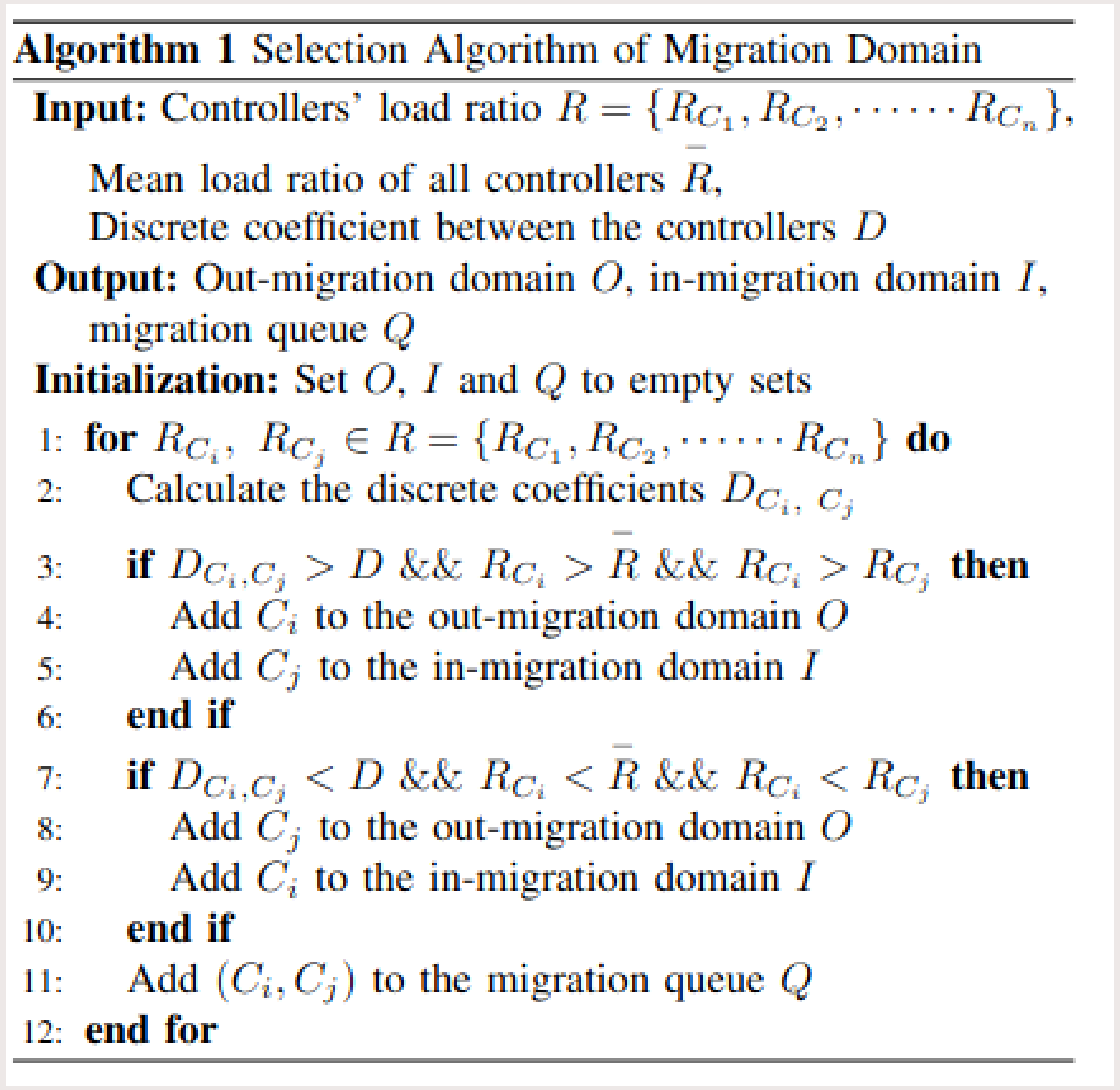

Begin
Get real-time load rate R
Calculate average load rate $\bar{R}$
Yes
$R > \bar{R}$
Select migration area
No
next round
Select the switch to be migrated
Yes
Load balancing
Switch migration
No
End

Fig. 2. Load balancing mechanism.

The selection algorithm selects a series of switch migration triples (**the migrate-out domain, the migrate-in domain, and the migrating switch**), then establishes an optimal migration model based on reinforcement learning in the switch migration decision phase.

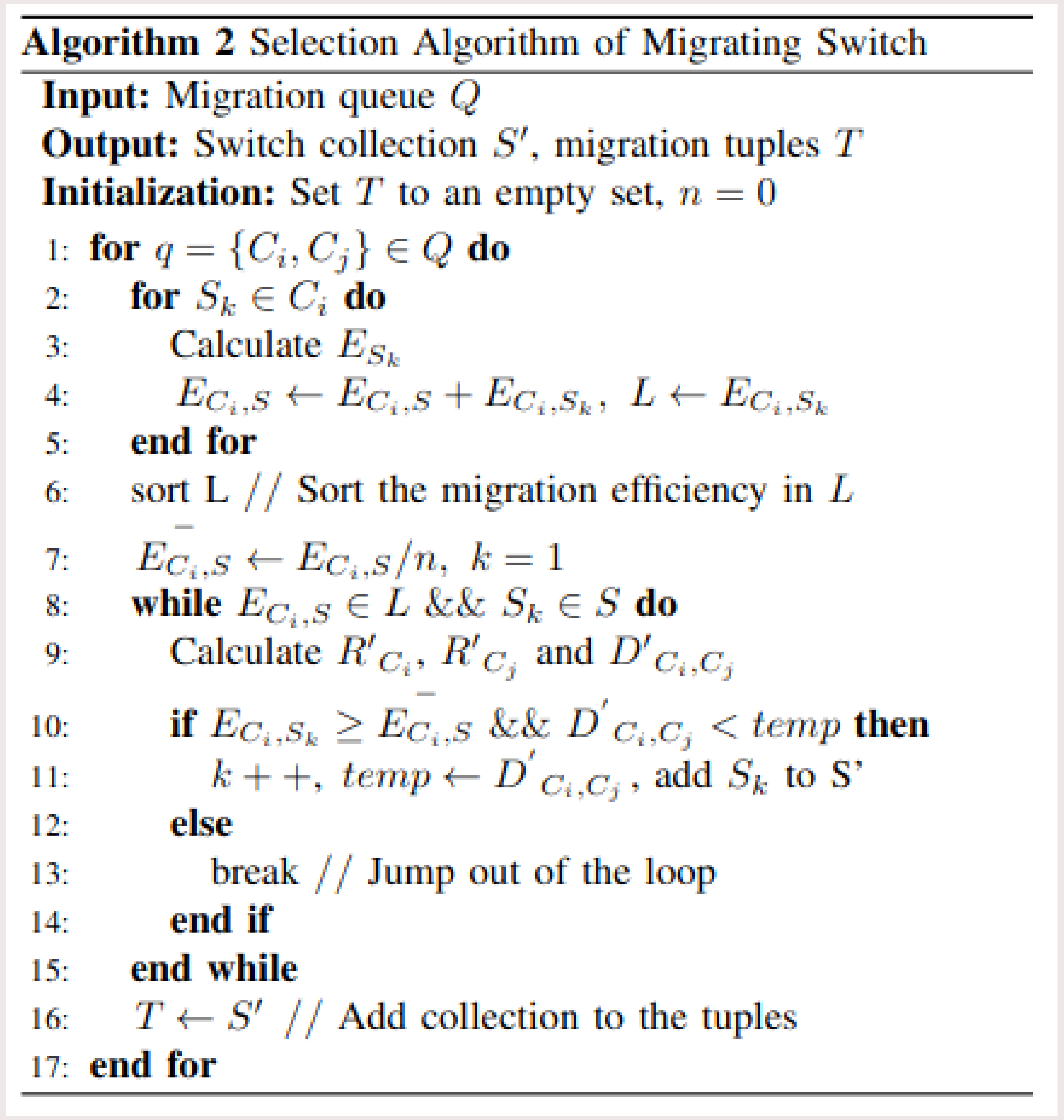
The selection algorithm proposed will add an anti-collision mechanism in the judgment part to avoid the consequences of a bad migration.

A switch combination is managed by the out-migration controller, whose migration efficiency has greater than the average efficiency and it maximizes the reduction of the discrete coefficients between controllers.



Algorithm 1 Selection Algorithm of Migration Domain

```
Input: Controllers' load ratio $R = \{R_{C_1}, R_{C_2}, \cdots\cdots R_{C_n}\}$,
    Mean load ratio of all controllers $\bar{R}$,
    Discrete coefficient between the controllers $D$
Output: Out-migration domain $O$, in-migration domain $I$,
    migration queue $Q$
Initialization: Set $O$, $I$ and $Q$ to empty sets
1: for $R_{C_i}$, $R_{C_j} \in R = \{R_{C_1}, R_{C_2}, \cdots\cdots R_{C_n}\}$ do
2:     Calculate the discrete coefficients $D_{C_i,\ C_j}$
3:     if $D_{C_i,C_j} > D$ && $R_{C_i} > \bar{R}$ && $R_{C_i} > R_{C_j}$ then
4:         Add $C_i$ to the out-migration domain $O$
5:         Add $C_j$ to the in-migration domain $I$
6:     end if
7:     if $D_{C_i,C_j} < D$ && $R_{C_i} < \bar{R}$ && $R_{C_i} < R_{C_j}$ then
8:         Add $C_j$ to the out-migration domain $O$
9:         Add $C_i$ to the in-migration domain $I$
10:    end if
11:    Add $(C_i, C_j)$ to the migration queue $Q$
12: end for
```



Algorithm 2 Selection Algorithm of Migrating Switch

```
Input: Migration queue $Q$
Output: Switch collection $S'$, migration tuples $T$
Initialization: Set $T$ to an empty set, $n = 0$
1: for $q = \{C_i, C_j\} \in Q$ do
2:     for $S_k \in C_i$ do
3:         Calculate $E_{S_k}$
4:         $E_{C_i,S} \leftarrow E_{C_i,S} + E_{C_i,S_k}$, $L \leftarrow E_{C_i,S_k}$
5:     end for
6:     sort L // Sort the migration efficiency in $L$
7:     $\bar{E}_{C_i,S} \leftarrow E_{C_i,S}/n$, $k = 1$
8:     while $E_{C_i,S} \in L$ && $S_k \in S$ do
9:         Calculate $R'_{C_i}$, $R'_{C_j}$ and $D'_{C_i,C_j}$
10:        if $E_{C_i,S_k} \geq \bar{E}_{C_i,S}$ && $D'_{C_i,C_j} < temp$ then
11:            $k++$, $temp \leftarrow D'_{C_i,C_j}$, add $S_k$ to S'
12:        else
13:            break // Jump out of the loop
14:        end if
15:    end while
16:    $T \leftarrow S'$ // Add collection to the tuples
17: end for
```

Switch Migration Decision using Q-Learning

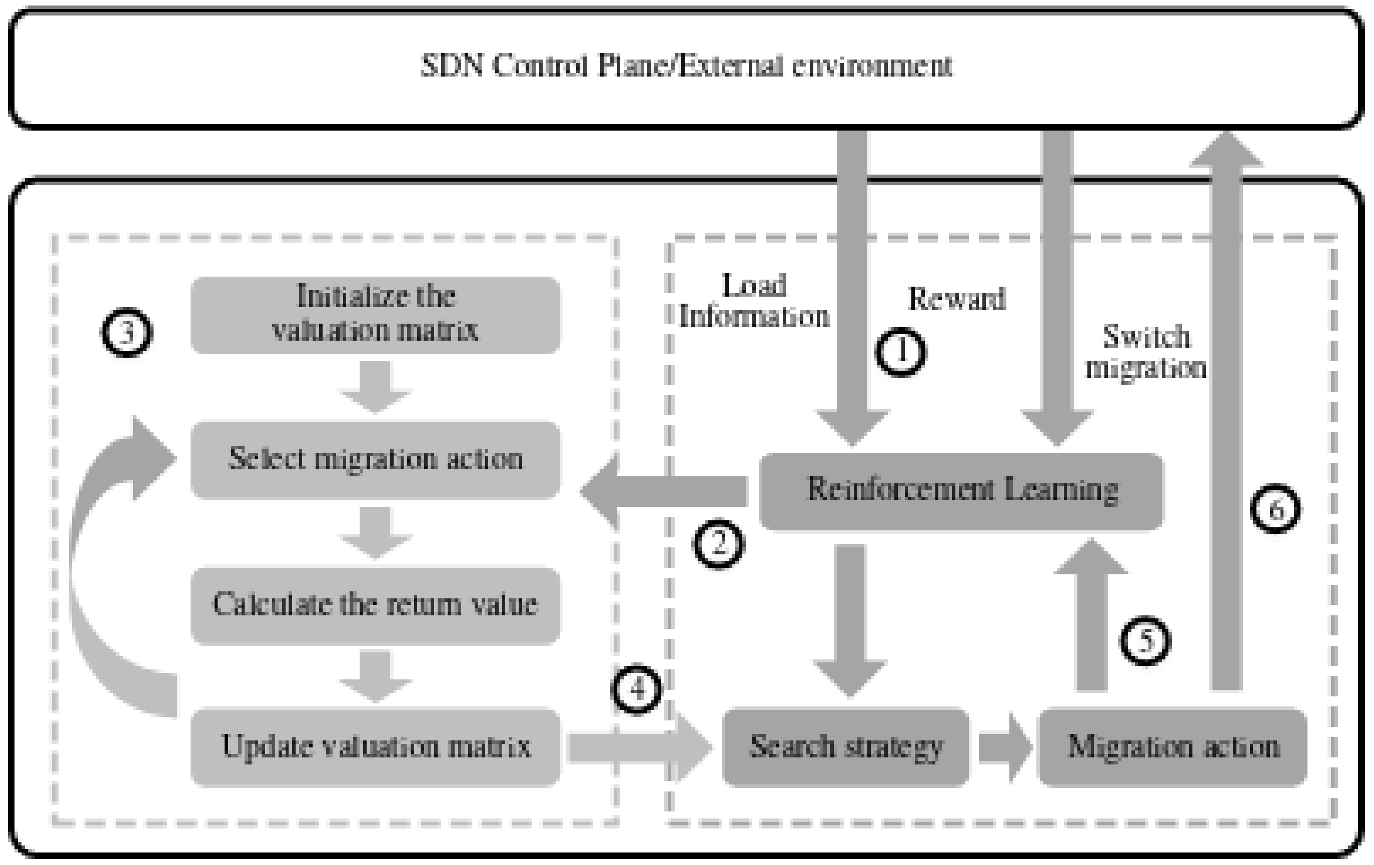


Fig. 3. Optimal migration model based on reinforcement learning.

Q-learning is a model-free reinforcement learning algorithm to learn the value of an action in a particular state. Reinforcement Learning is a decision-making algorithm. It's all about figuring out how to behave optimally in a given situation in order to maximize reward. This optimal behavior is learned through experiences with the environment and observations of how it reacts

Problem formulation in MDP

State Space (S) - S is the finite state space. In our problem, state-space S is the migration domain, and a state ST ∈S corresponds to the index of the controller i.e. if we have 5 controllers then S = c0,c1,...c5.

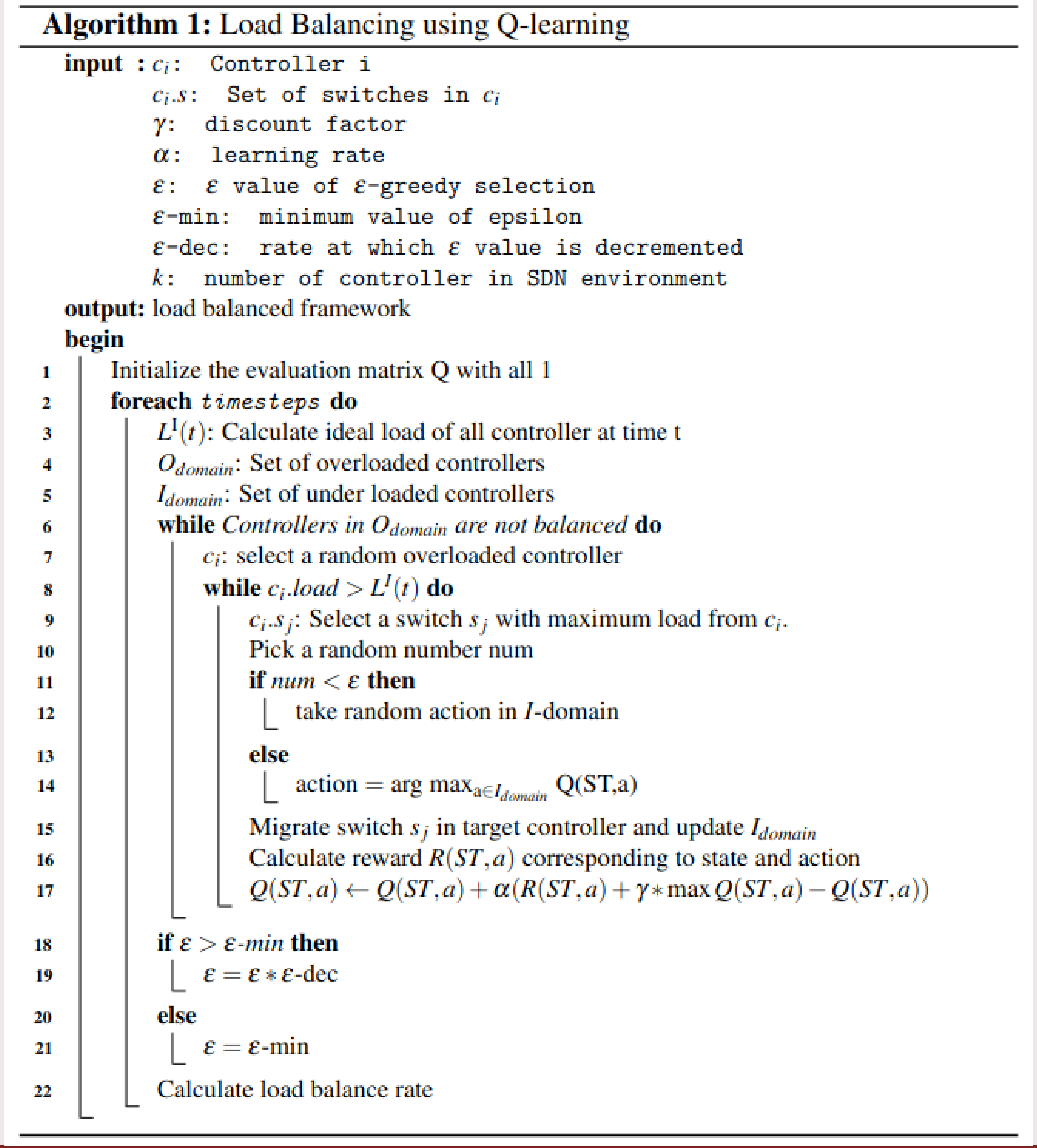
Action Space (A) - A is the finite action space. In our problem, an action is defined as selecting the controller for migration from the in-migration domain (i.e. set of underloaded domains). Each action a ∈ A is represented by an index of underloaded controller selected for migration (i.e. target controller).

Reward Function (R) - R represents the immediate reward associated with state-action pairs, denoted by R(ST, a), where ST ∈S and a ∈ A. R(ST,a) is degree of change of load balancing after migration.

$$R(ST,a) = D_{i,j} - D'_{i,j}$$

where, $D_{i,j} := D_{c_i,c_j} = \left(\sqrt{\sum_{k=i,j}(L_{c_k} - L^{\mathrm{I}}_{c_i,c_j})^2/2}\right)/L^{\mathrm{I}}_{c_i,c_j}$

here, $D_{i,j}$ is load deviation coefficient between two controllers c_i, c_j



Algorithm 1: Load Balancing using Q-learning

```
input : $c_i$:  Controller i
        $c_i.s$:  Set of switches in $c_i$
        $\gamma$:  discount factor
        $\alpha$:  learning rate
        $\varepsilon$:  $\varepsilon$ value of $\varepsilon$-greedy selection
        $\varepsilon$-min:  minimum value of epsilon
        $\varepsilon$-dec:  rate at which $\varepsilon$ value is decremented
        $k$:  number of controller in SDN environment
output: load balanced framework
begin
    Initialize the evaluation matrix Q with all 1
    foreach timesteps do
        $L^I(t)$: Calculate ideal load of all controller at time t
        $O_{domain}$: Set of overloaded controllers
        $I_{domain}$: Set of under loaded controllers
        while Controllers in $O_{domain}$ are not balanced do
            $c_i$: select a random overloaded controller
            while $c_i.load > L^I(t)$ do
                $c_i.s_j$: Select a switch $s_j$ with maximum load from $c_i$.
                Pick a random number num
                if $num < \varepsilon$ then
                    take random action in $I$-domain
                else
                    action = $\arg\max_{a \in I_{domain}}$ Q(ST,a)
                Migrate switch $s_j$ in target controller and update $I_{domain}$
                Calculate reward $R(ST,a)$ corresponding to state and action
                $Q(ST,a) \leftarrow Q(ST,a) + \alpha(R(ST,a) + \gamma * \max Q(ST,a) - Q(ST,a))$
        if $\varepsilon > \varepsilon$-min then
            $\varepsilon = \varepsilon * \varepsilon$-dec
        else
            $\varepsilon = \varepsilon$-min
        Calculate load balance rate
```

Here, we apply an ε-greedy approach to train the Q-network by exploring a random action with probability ε, and exploiting an action that maximizes the expected long-term reward with probability 1 – ε.

For each action taken, we shall correspondingly obtain its new state and the resultant reward. This is treated as the experience in RL and shall be reserved for future use.

Load balancing rate:

The load balancing rate computes the variation, of the controller load from its ideal value at time 't'. In this report, we consider an ideal load of the controller as a mean load of system.

$$\frac{\sum_{j=1}^{k}(|L_{c_j}(t) - L^I(t)|)}{k}$$

Progress & Planning

1. Have received and understood the code that is currently working.
2. Have gone through (and also going through), more research papers to understand better about the work done or look at different techniques used by research papers.

1. Need to refactor parts of the code, and reducing time-complexity at places possible.
2. Make a more modular project structure, instead of keeping everything in 1 single python file
3. Identified a few metrics (eg. switch migration efficiency) which can improve results.

AB Satyaprakash

B.Tech - Mathematics and Computing,
IIT Guwahati
180123062
satyapra@iitg.ac.in