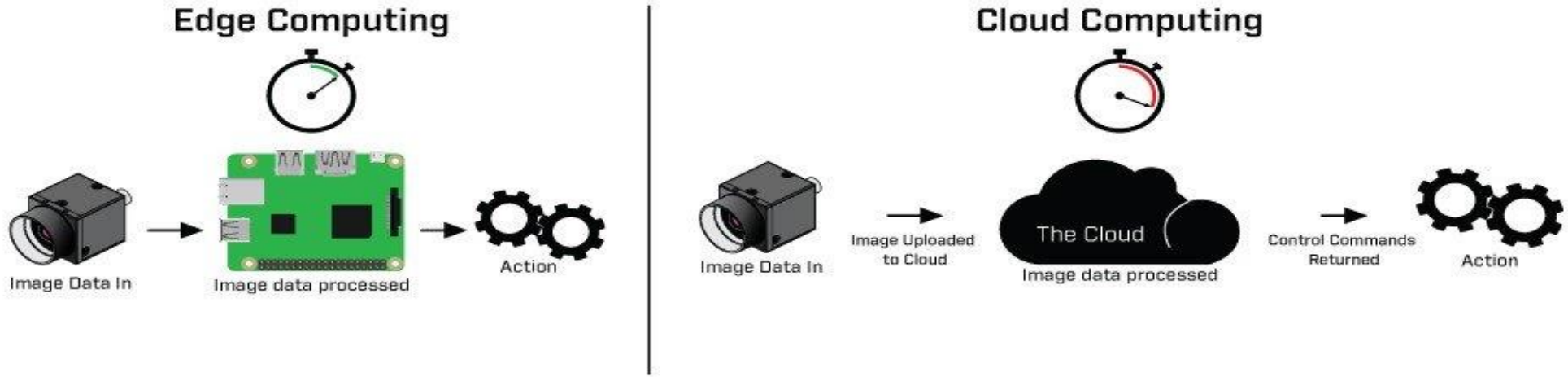
Energy Efficient Reconfiguration Algorithm using Reinforcement Learning in Federated Edge Cloud

Esrat Maria

Distributed and Cloud Computing Lab
(http://dcclab.sogang.ac.kr)

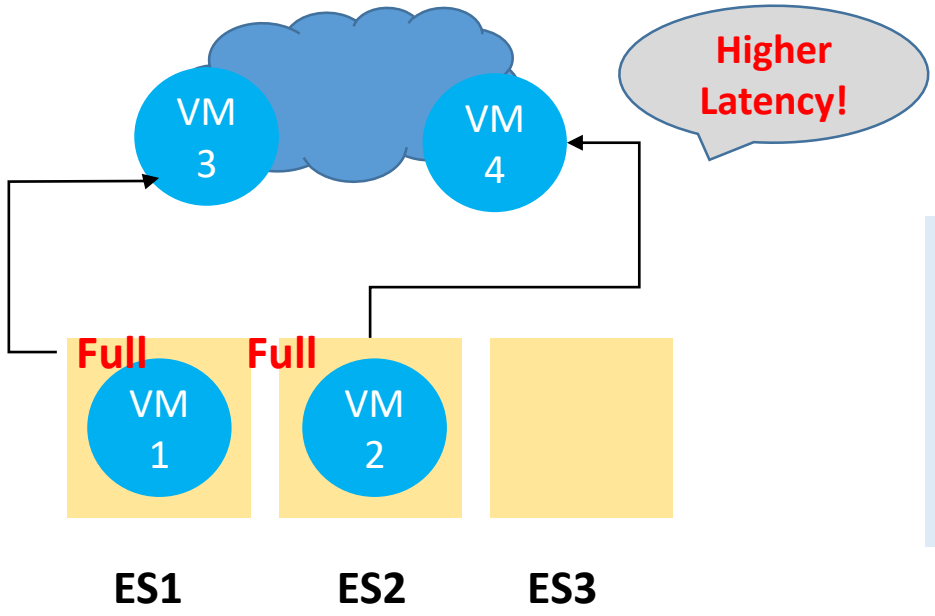
Background

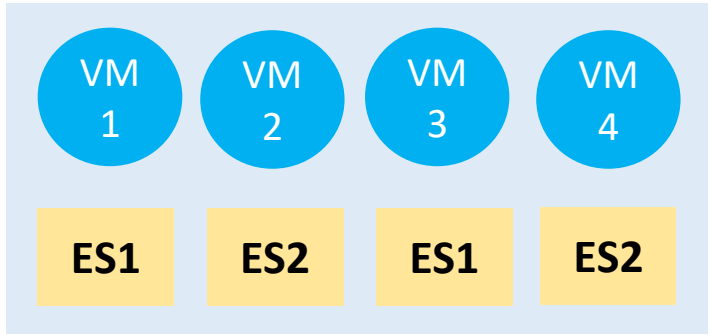
- **Traditional cloud computing limitations**
  - Physical distance between users and cloud servers
  - Service **latency is increased**



Background

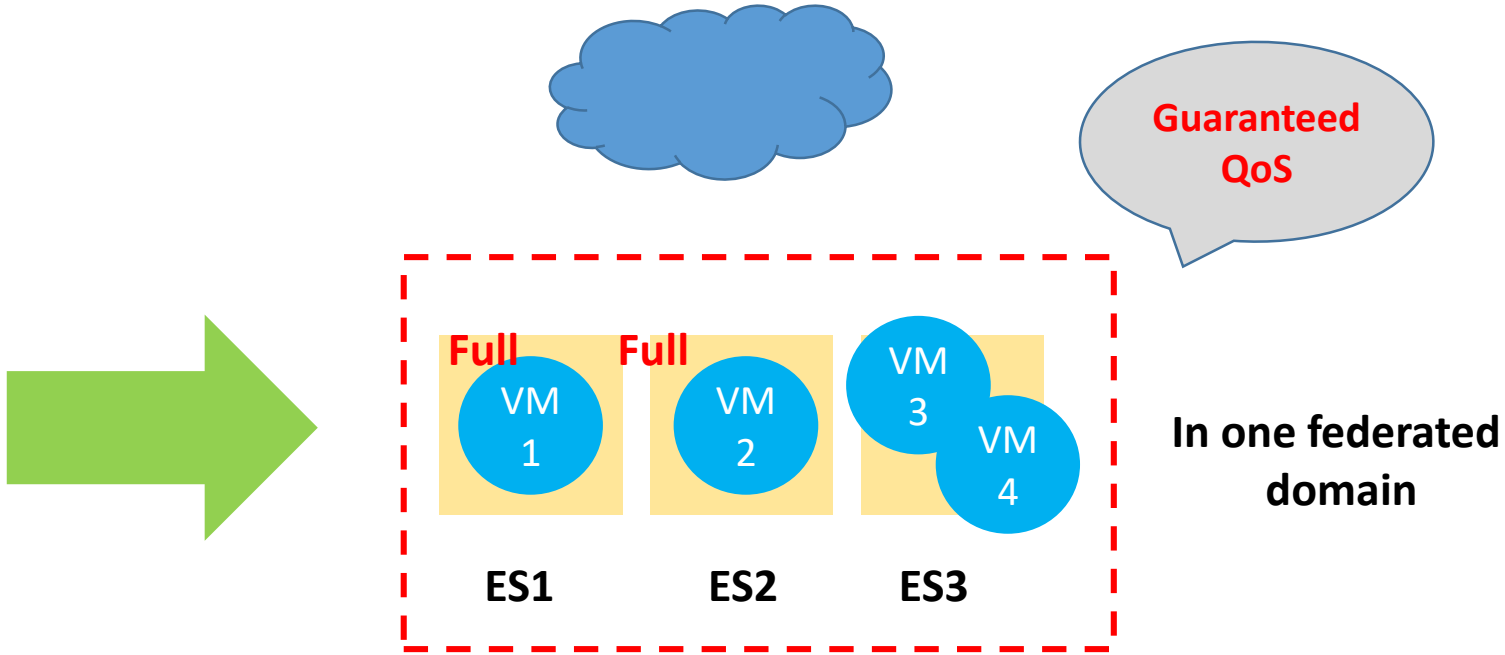
- **Edge Computing may not be enough**
  - Limited capacity
  - Edge server in each domain increases





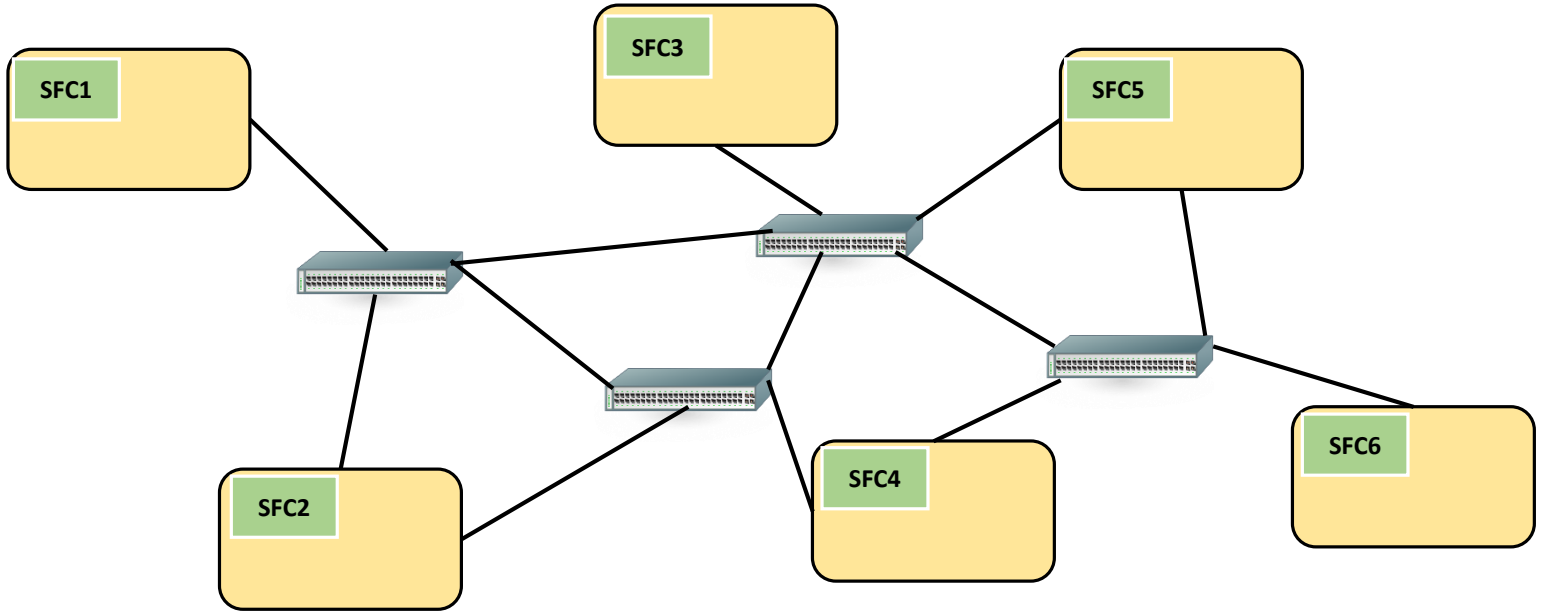
Background

- **Solution : Federated Edge Cloud (FEC)**
  - Offers collaboration of multiple edge servers
  - Nearby edge servers can share



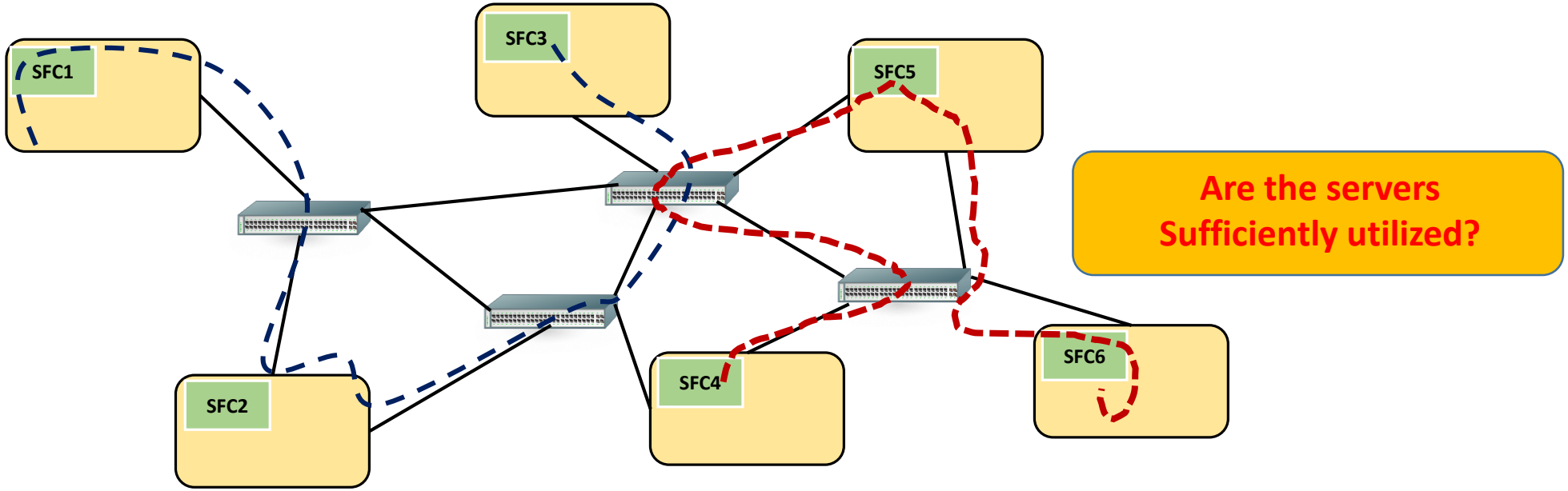
Motivation

- **Service functions are placed with their MAX requirement**
  - More servers are required to be **turned on**
  - One Service Function Chain (SFC) in one server
  - Inefficient when dealing with **varying number of traffic**



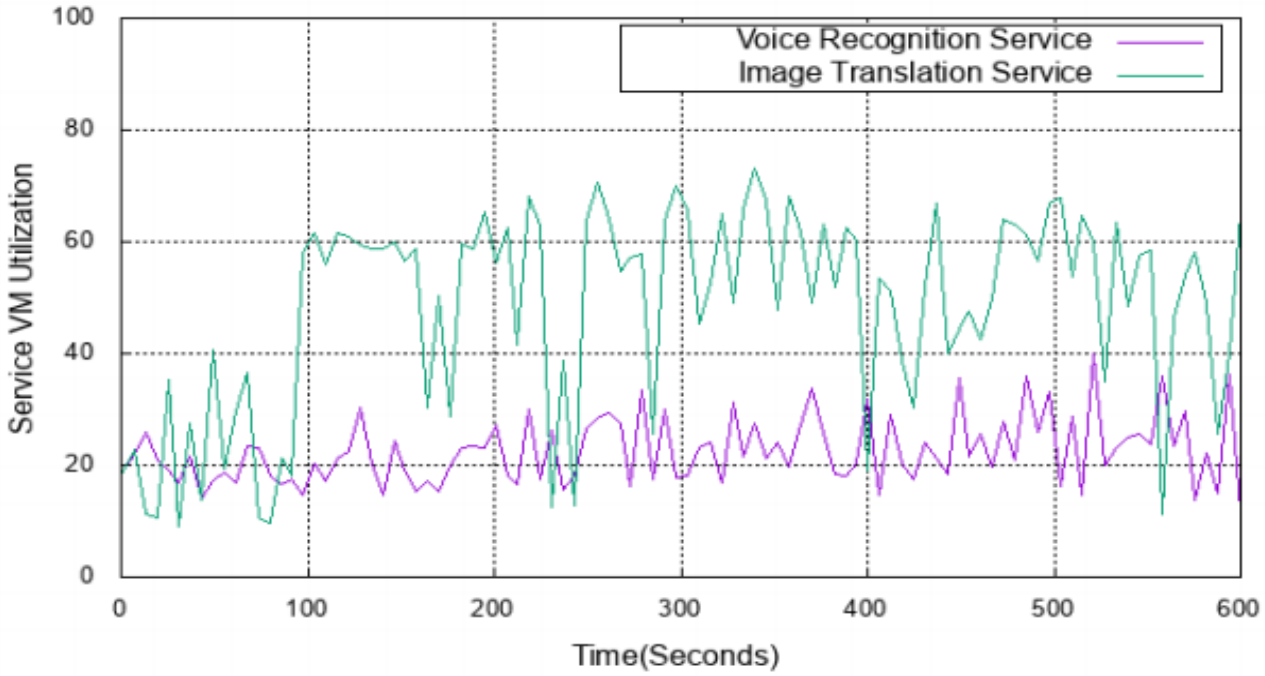
Motivation

- **Service functions are placed with their MAX requirement**
  - More servers are required to be **turned on**
  - One Service Function Chain (SFC) in one server
  - Inefficient when dealing with **varying number of traffic**



Motivation

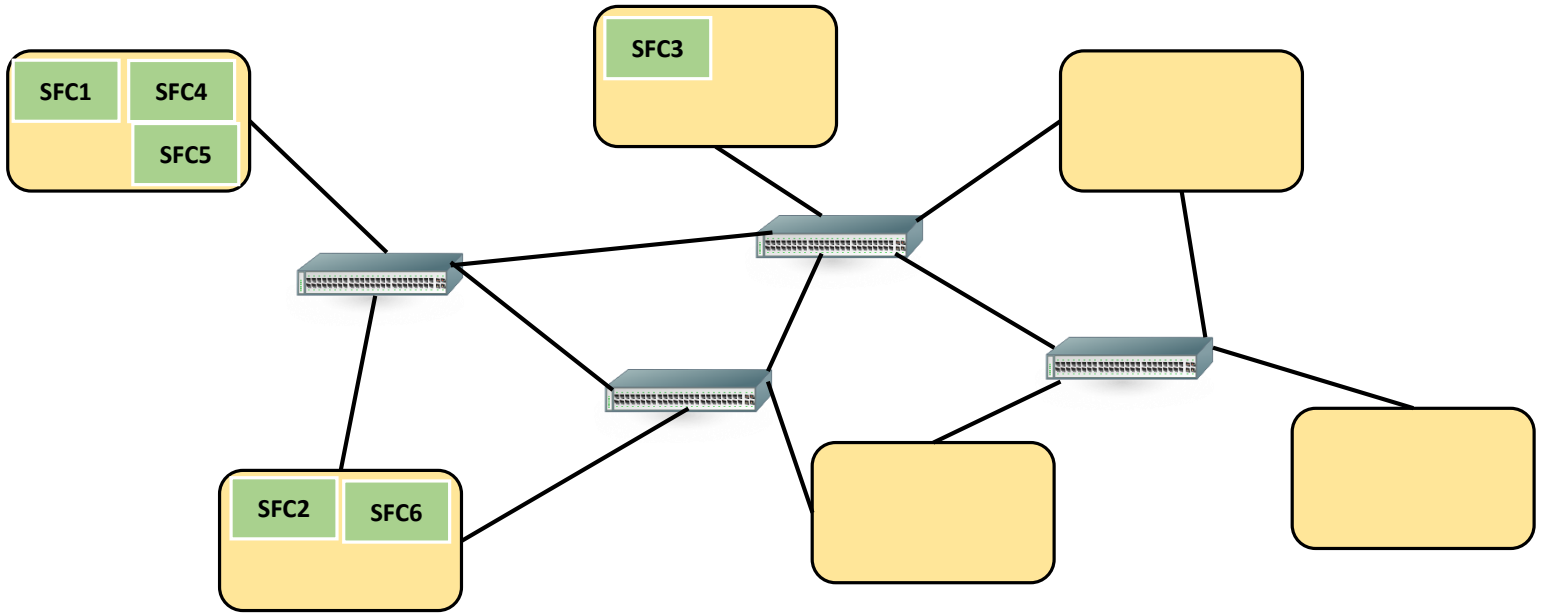
- **What happens when placed with MAX requirement**
  - Under-utilized servers
  - Inefficient when traffic is fluctuating



Usually 50~60% utilized

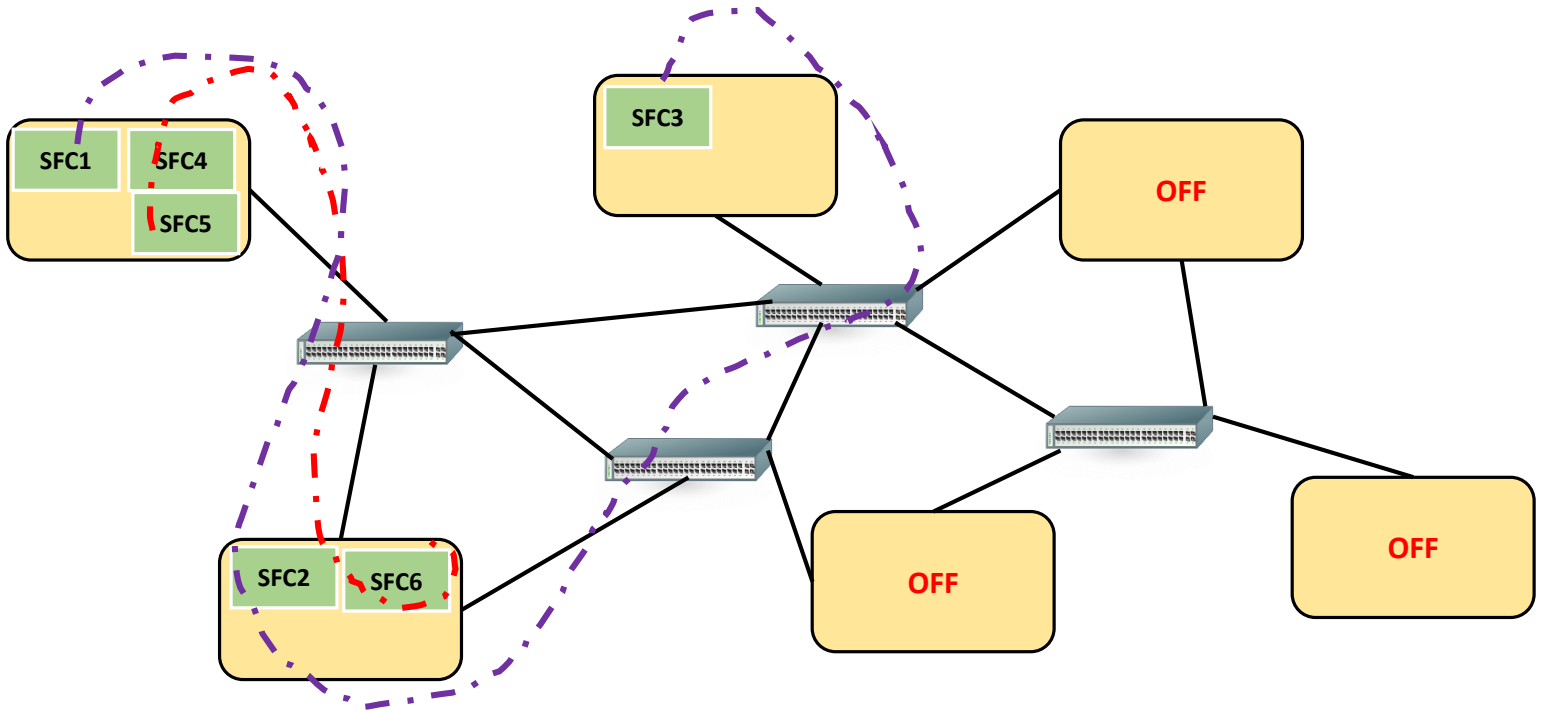
Motivation

- With **MIN** requirement the same service function placement example is much more efficient



Motivation

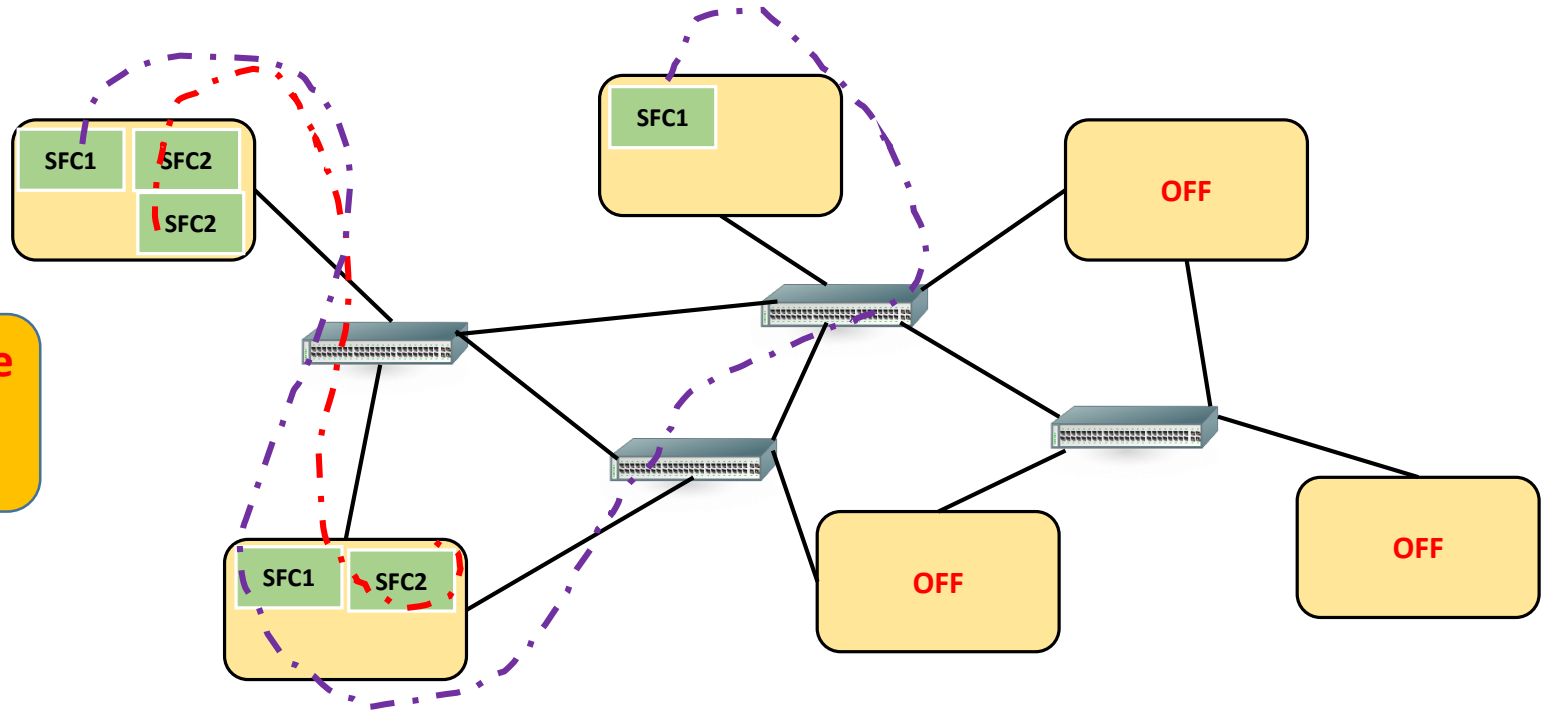
- With **MIN** requirement the same service function placement example is much more efficient



Motivation

- With **MIN** requirement the same service function placement example is much more efficient

Consolidation of more VMs in One server



Motivation

- **Cons in MIN approach**
  - With high traffic **MIN** approach is not suitable
    - ✓ Resources are overloaded faster
    - ✓ More energy required
  - Small server capacity in a FEC
    - Increased number of migration and migration overhead

Migration energy increased!

How to settle to an optimal phase?

Previous Works

- **Few efforts made to minimize energy in FEC**
  - **"Sla-aware and energy-efficient dynamic overbooking in sdn-based cloud data centers" – proposed:**
    - Dynamic overbooking algorithm to allocate host and network resources
    - Over/Under provisioning of resources
    - Workload variation not considered
  - **Various threshold based approaches**

Previous Works

Few efforts made to minimize energy in FEC

- Use of Reinforcement Learning / ML approaches
  - **"Network aware approach for the scheduling of virtual machine migration during peak loads" – proposed:**
    - ✓ Agent that learns optimal migration time
      - Overall migration minimization not involved
      - Service path reconfiguration not involved
  - **"A reinforcement learning approach for dynamic selection of virtual machines in cloud data centres" – proposed:**
    - ✓ Agent that chooses optimal VM to migrate
  - **"Using reinforcement learning for autonomic resource allocation in clouds: Towards a fully automated workflow" – proposed:**
    - ✓ RL controller that dynamically allocates and de-allocates resources based on workload variation.

FEC-RL(Reinforcement Learning)

This paper proposes a RL based service reconfiguration algorithm

1. Minimizes energy consumption along the service path
2. Guarantees service QoS

❖ **Through trial and error FEC-RL founds an optimal solution that reduces energy**

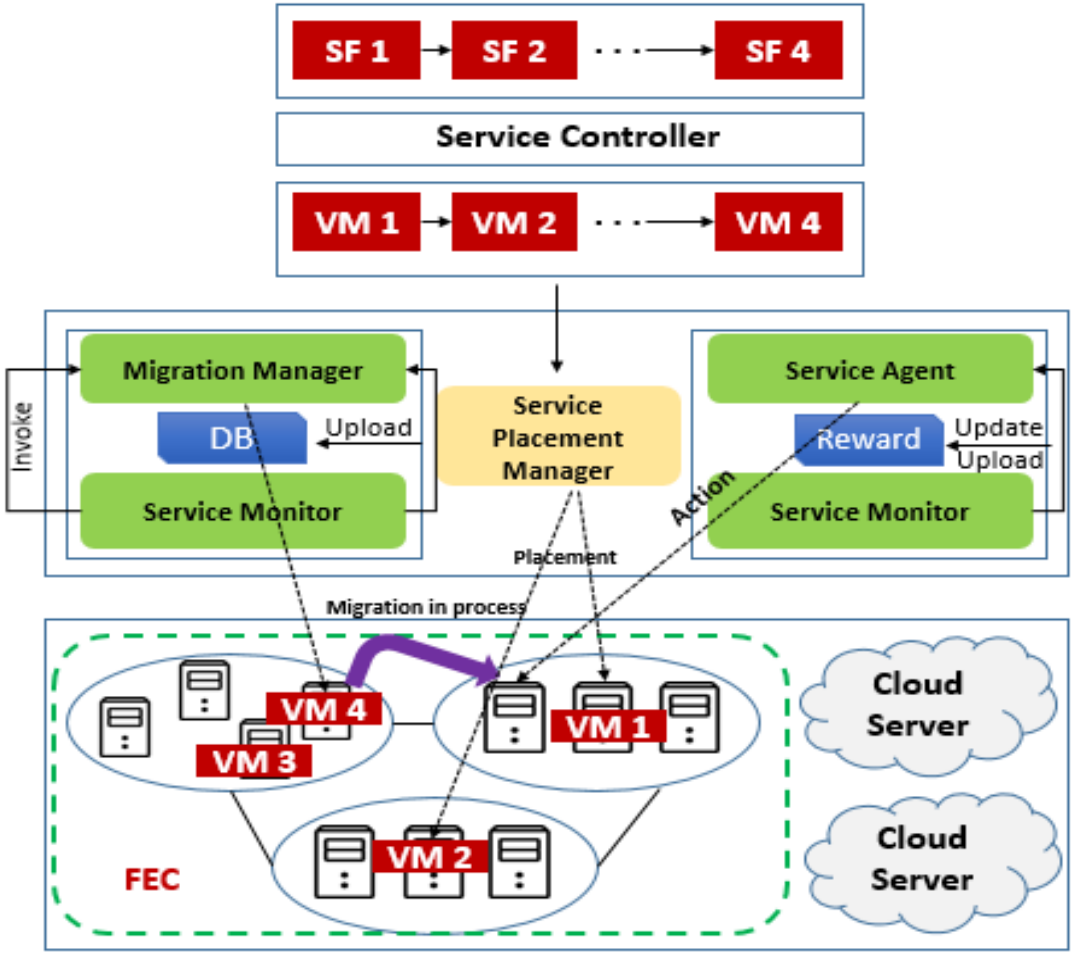
❖ **A relatively low number of migration and migration overhead while dealing with traffic fluctuation**

- Reinforcement learning based reconfiguration algorithm that can reduce the service migration overhead

Proposed Approach – FEC-RL (Architecture)

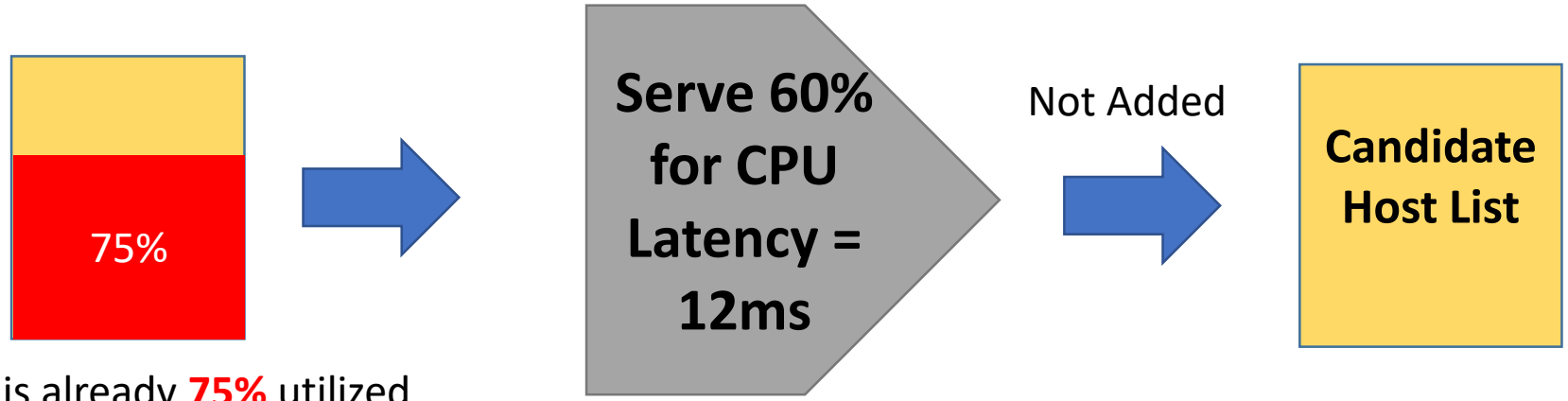
Four Major components

- **Service Placement Manager**
  - ✓ Using MIN traffic requirement
- **Migration Manager**
  - ✓ Invokes migration when host over utilized
- **Service Monitor**
- **Service Agent**
  - ✓ Activated by Migration Manager to choose the optimal Migration strategy



Filtering Server List

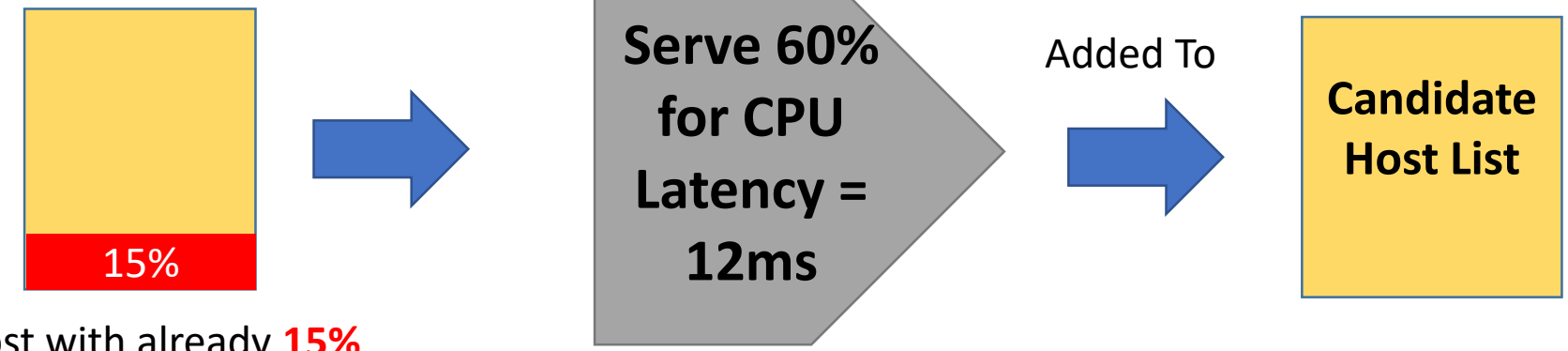
- **Before FEC-RL even starts, a filtering is done**
  - **Based on CPU and latency requirement by the service**
  - **Eligible servers make *Candidate Host List***
  - **For example, a service requires 60% Host CPU and latency 12ms**



If a host is already **75%** utilized
It can't fulfil the service request
Or if latency requirement cant be met

Filtering Server List

- **Before FEC-RL even starts, a filtering is done**
  - **Based on CPU and latency requirement by the service**
  - **Eligible servers make *Candidate Host List***
  - **For example, a service requires 60% Host CPU and latency 12ms**



A host with already **15%** utilization
And also in the nearby edge domain

State

- Total state number = Total number of host in Candidate Host List
- State represents → total physical energy consumption
  - Physical Energy = Server Energy + Link Energy
    - ✓ $E_{SP_i}^{physical} = E_{SP_i}^{server} + E_{SP_i}^{link}$
    - ✓ $E_i^{server} = E_{server_i}^{static} + \left(E_{server_i}^{max} - E_{server_i}^{static}\right) \times \frac{CPU_i^{used}}{CPU_i^{total}}$
    - ✓ $E_i^{link} = E_{switch_i}^{static} + E_{switch_i}^{port} \times num_{port}$
  - State represents total energy consumption of physical resources along the service path
- Server utilization = CPU utilization

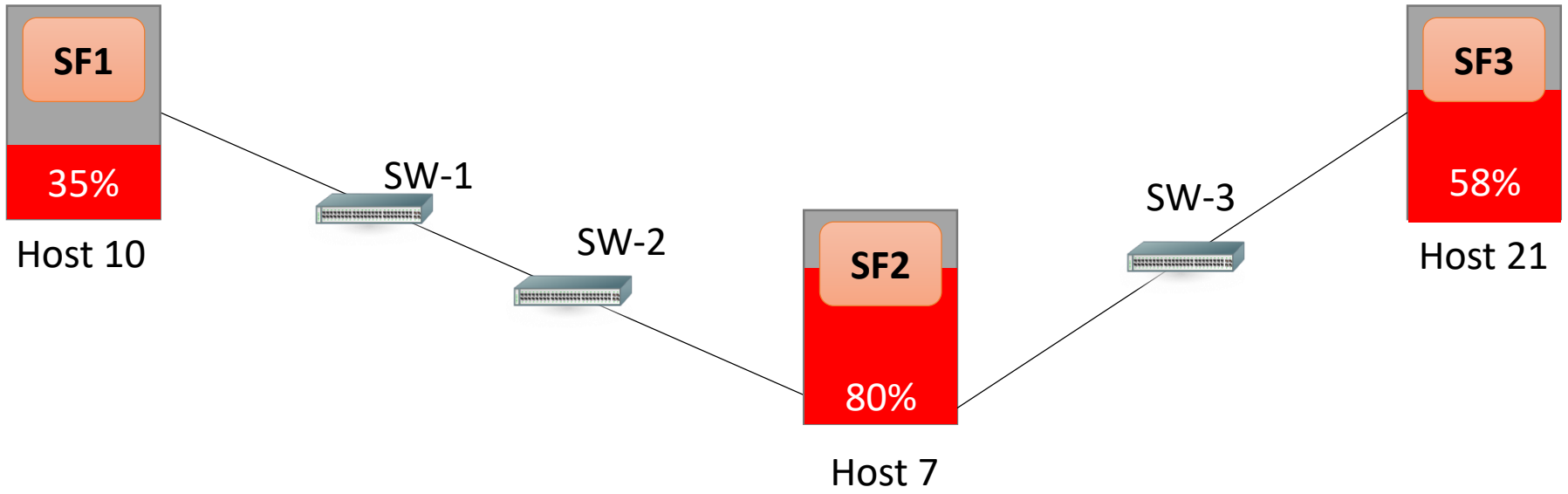
Action

- **Number of actions = Total number of host in Candidate Host List**
- **Two ways choosing action by the agent**
  - Exploration: random action by agent
    - ❖Works with ***no idea*** about the environment
    - ❖Initial learning stage
  - Exploitation: action taken by an experienced agent
    - ❖Able to choose optimal action
    - ❖Has more experience during this phase
    - ❖Decides from looking at the reward value

Usage of state aggregation → Mapping of 10 states into one

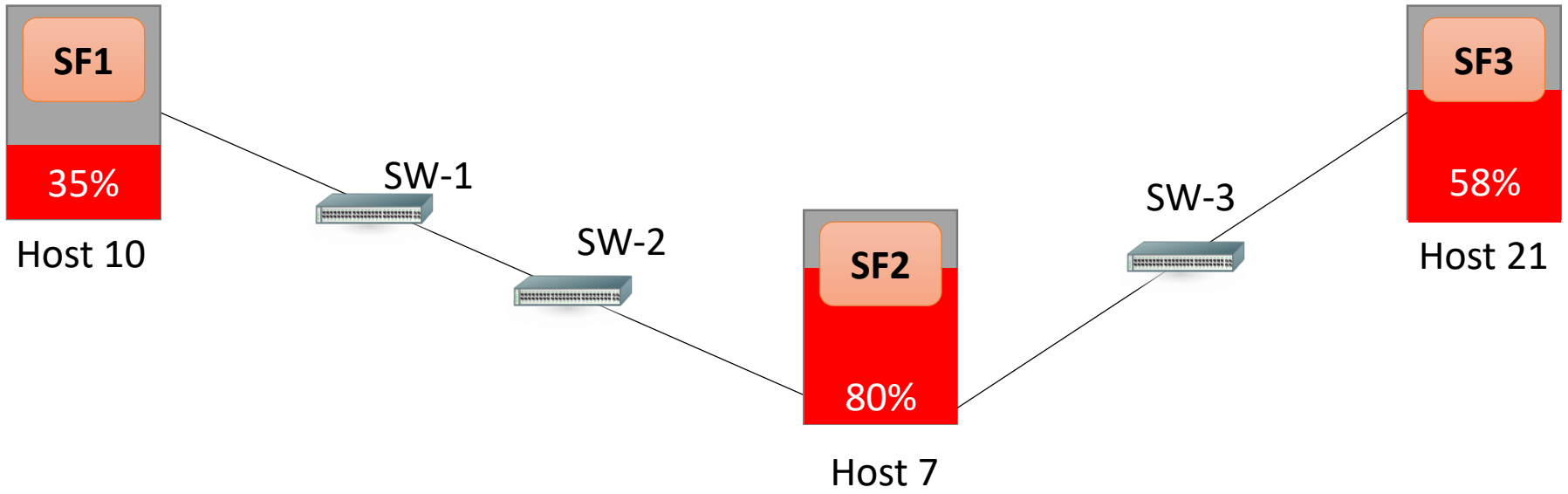
Service Path Construction

Service Path, *SP*



Service Path Construction

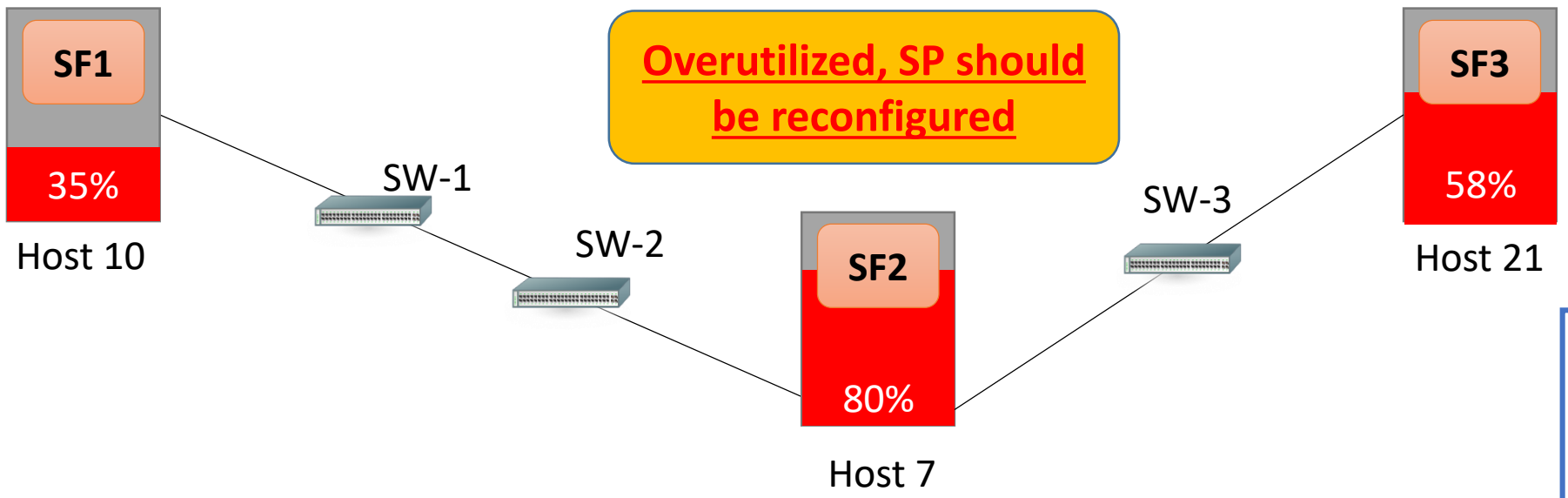
Service Path, *SP*



Path, ***SP*** = Host 10 –> SW-1 –> SW-2 –> Host 7 –> SW-3 –> Host 21

Service Path Construction

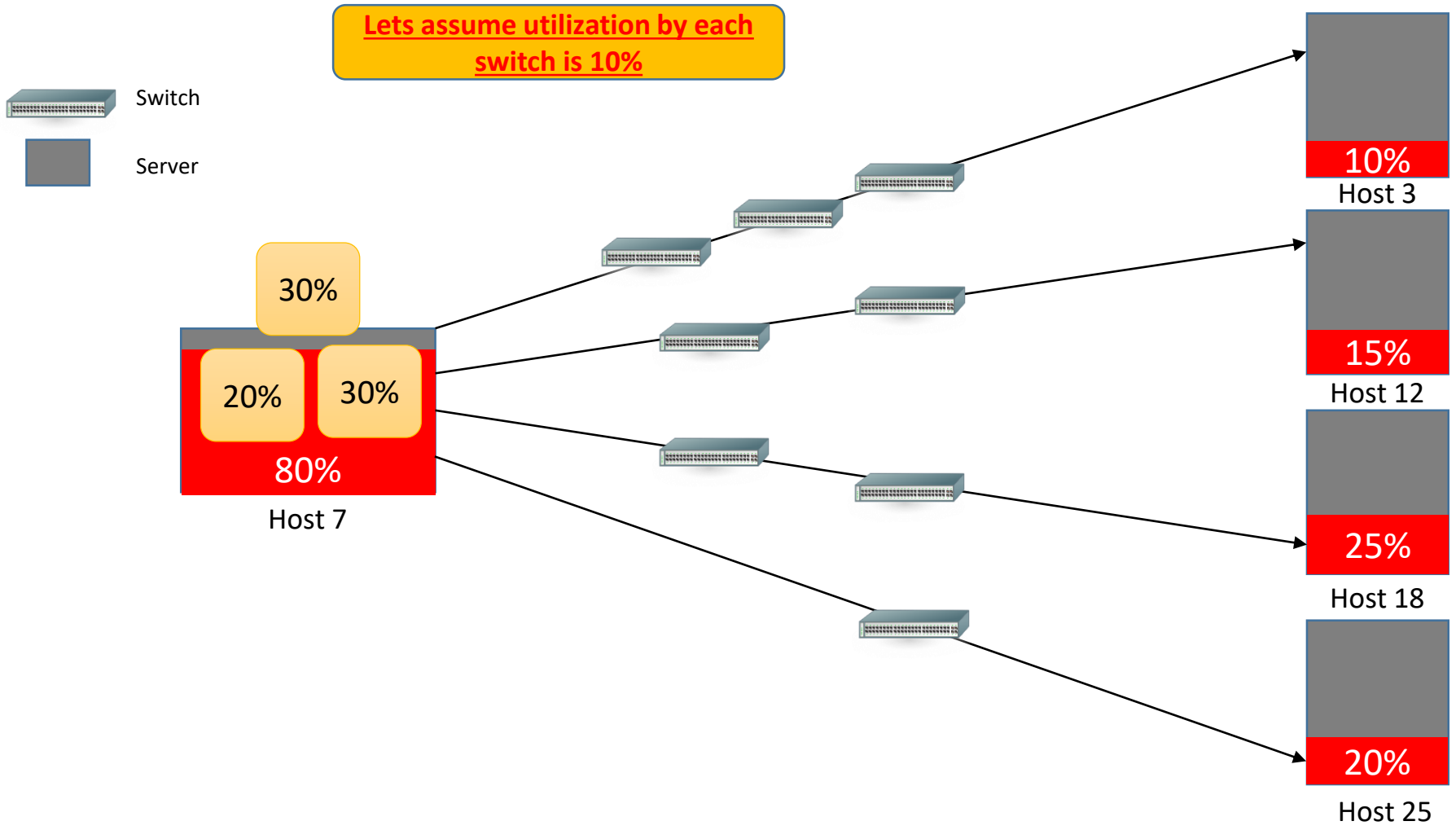
Service Path, *SP*



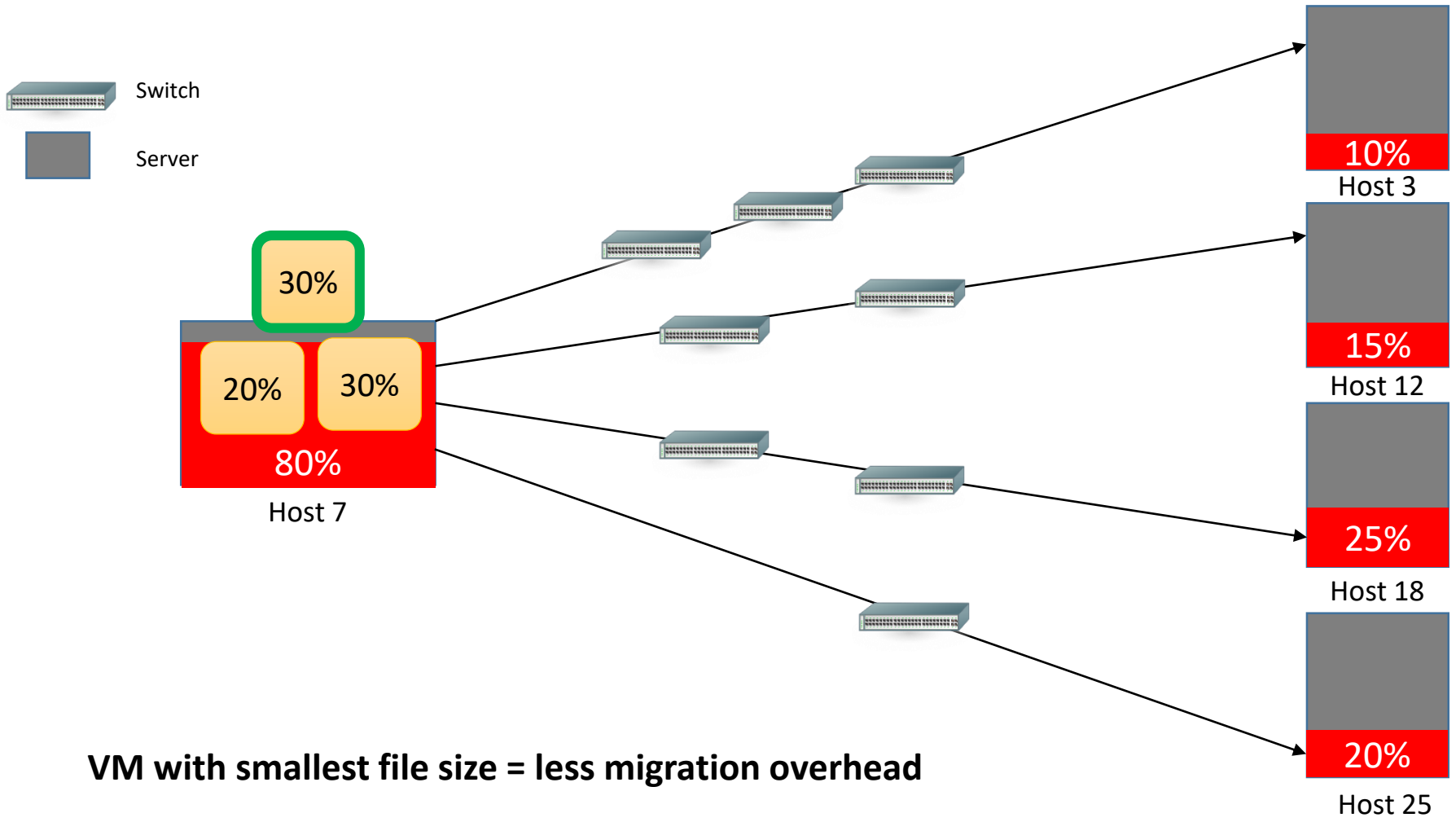
Path, ***SP*** = Host 10 –> SW-1 –> SW-2 –> Host 7 –> SW-3 –> Host 21

When a host is over utilized, migration is triggered and **FEC-RL** is invoked

Reward – (Calculation Example)

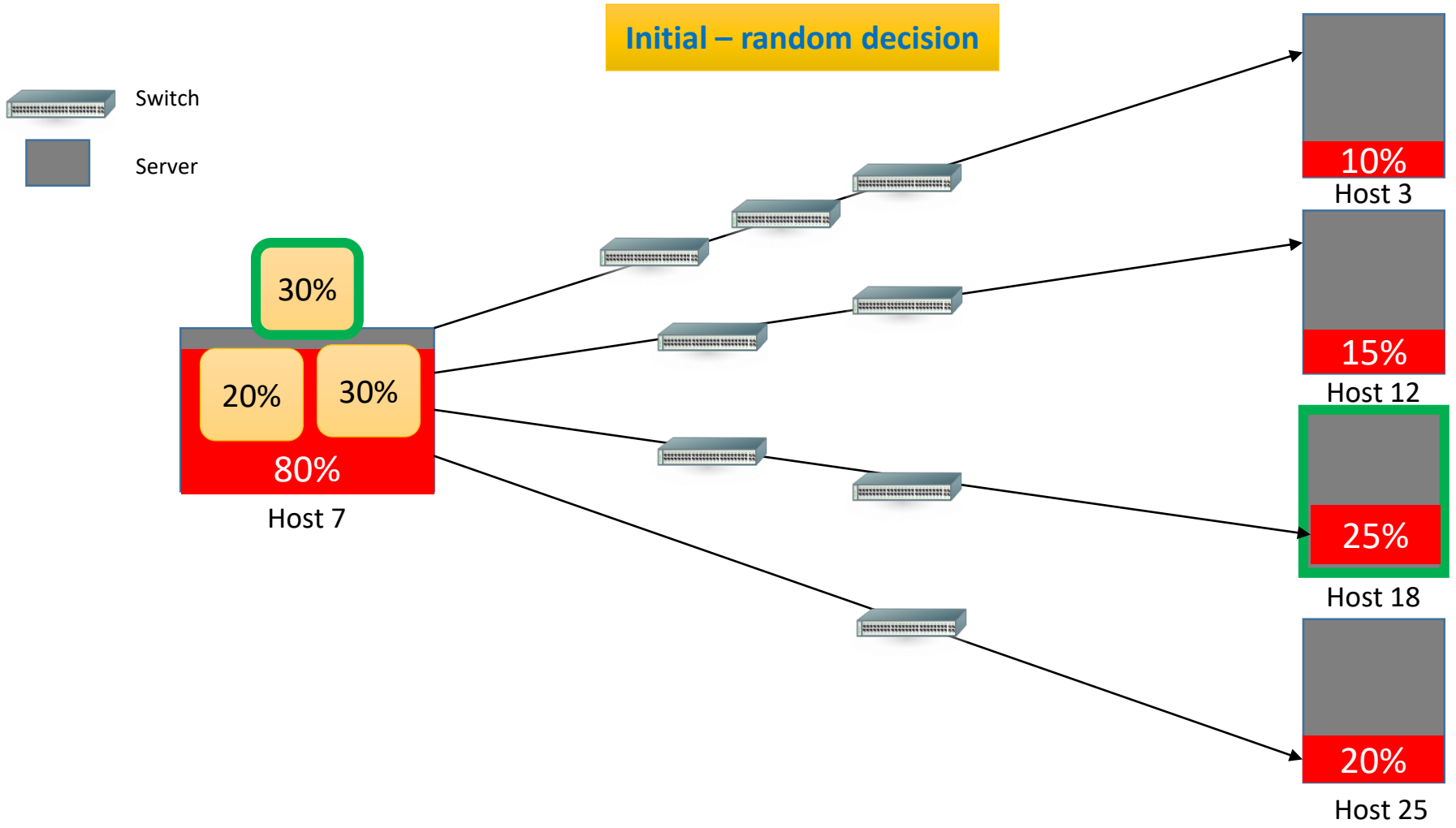


Reward – (Calculation Example)



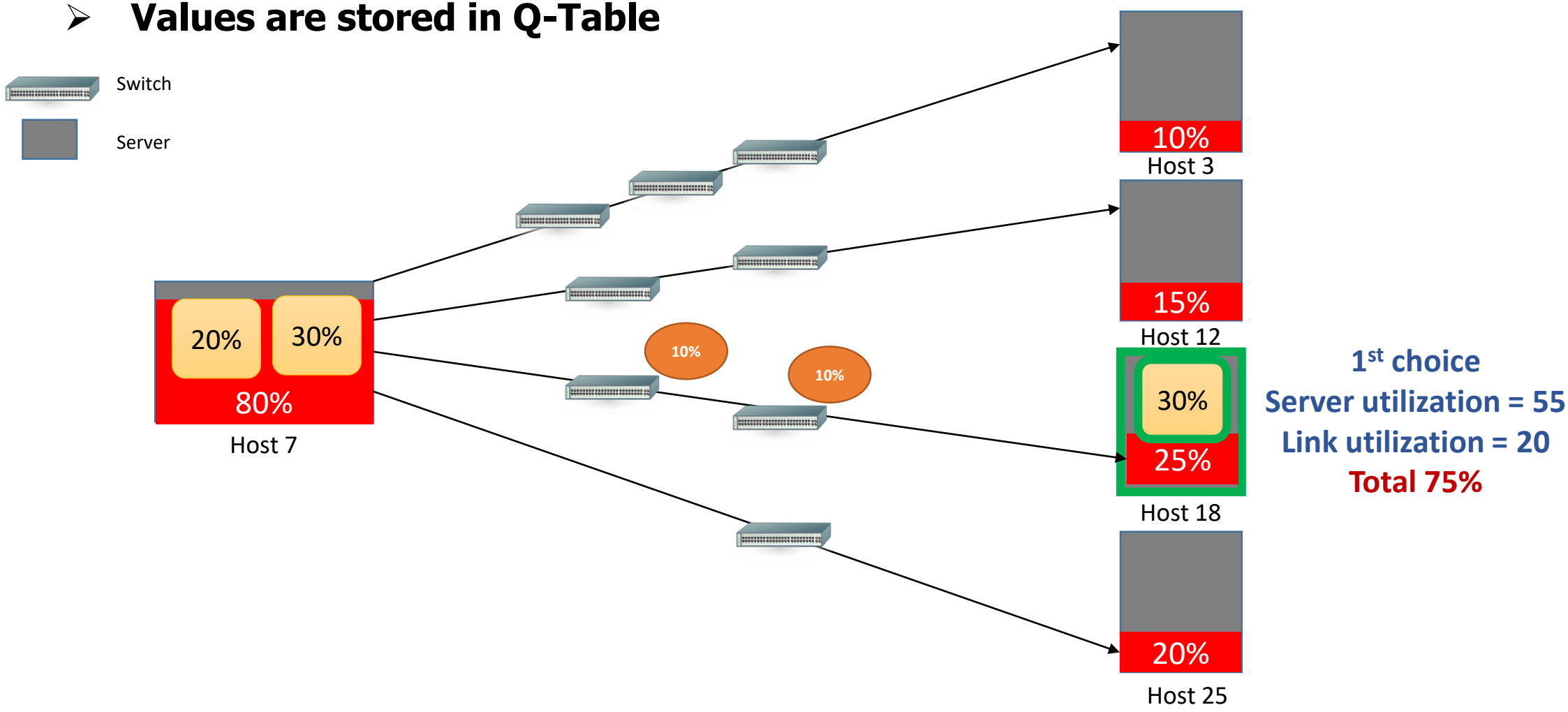
VM with smallest file size = less migration overhead

Reward – (Calculation Example)



Reward – (Calculation Example)

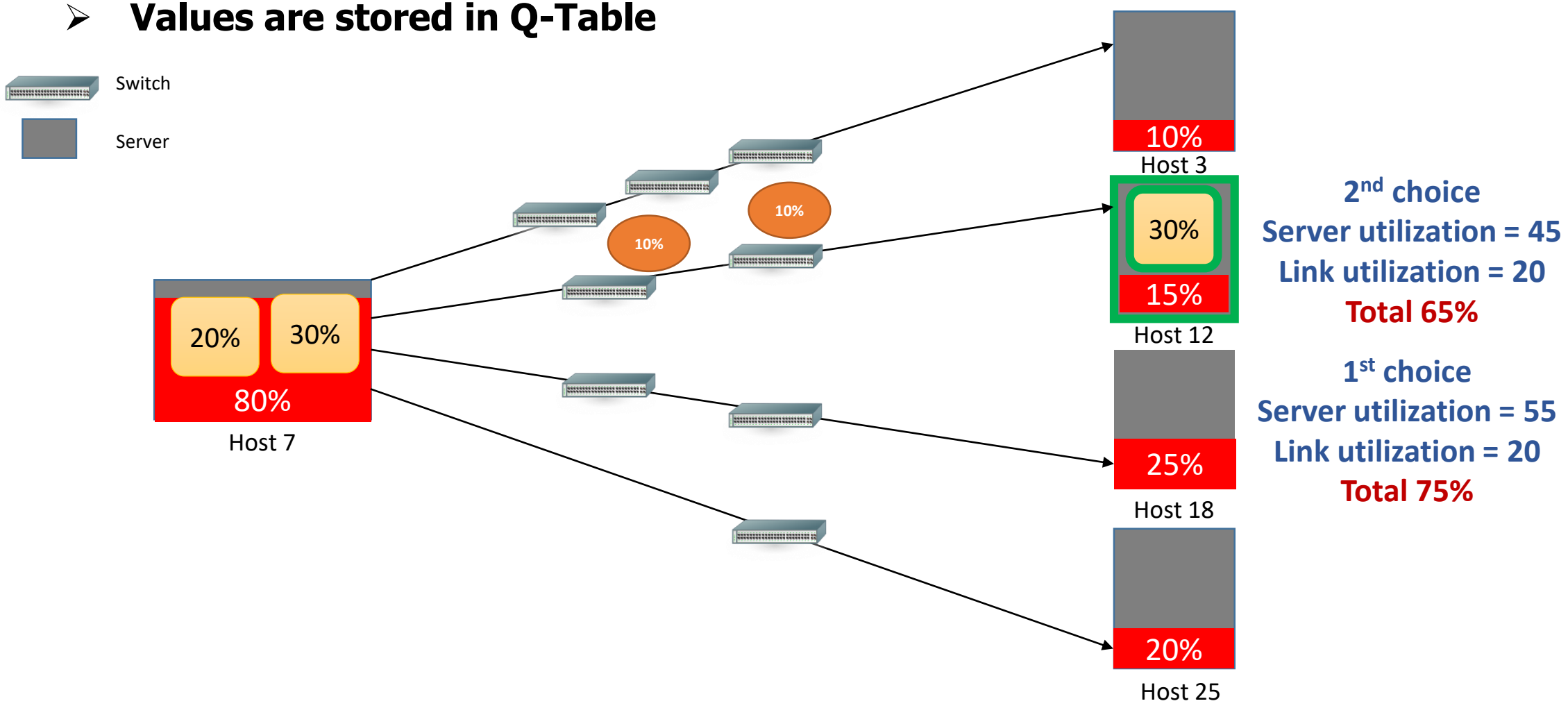
- **Values are stored in Q-Table**



1st choice
Server utilization = 55
Link utilization = 20
Total 75%

Reward – (Calculation Example)

- **Values are stored in Q-Table**

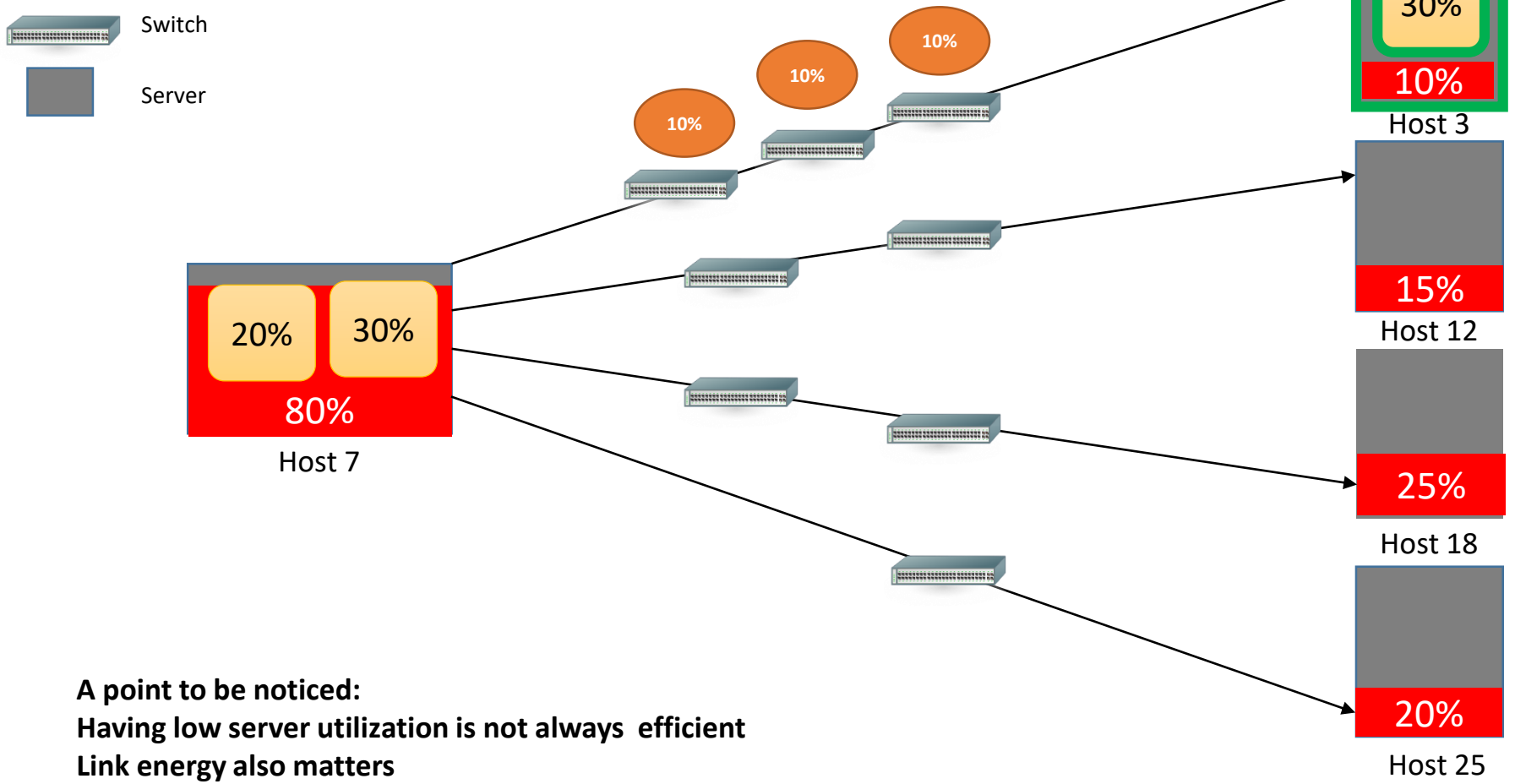


2nd choice
Server utilization = 45
Link utilization = 20
Total 65%

1st choice
Server utilization = 55
Link utilization = 20
Total 75%

Reward – (Calculation Example)

- **Values are stored in Q-Table**



3rd choice
Server utilization = 40
Link utilization = 30
Total 70%

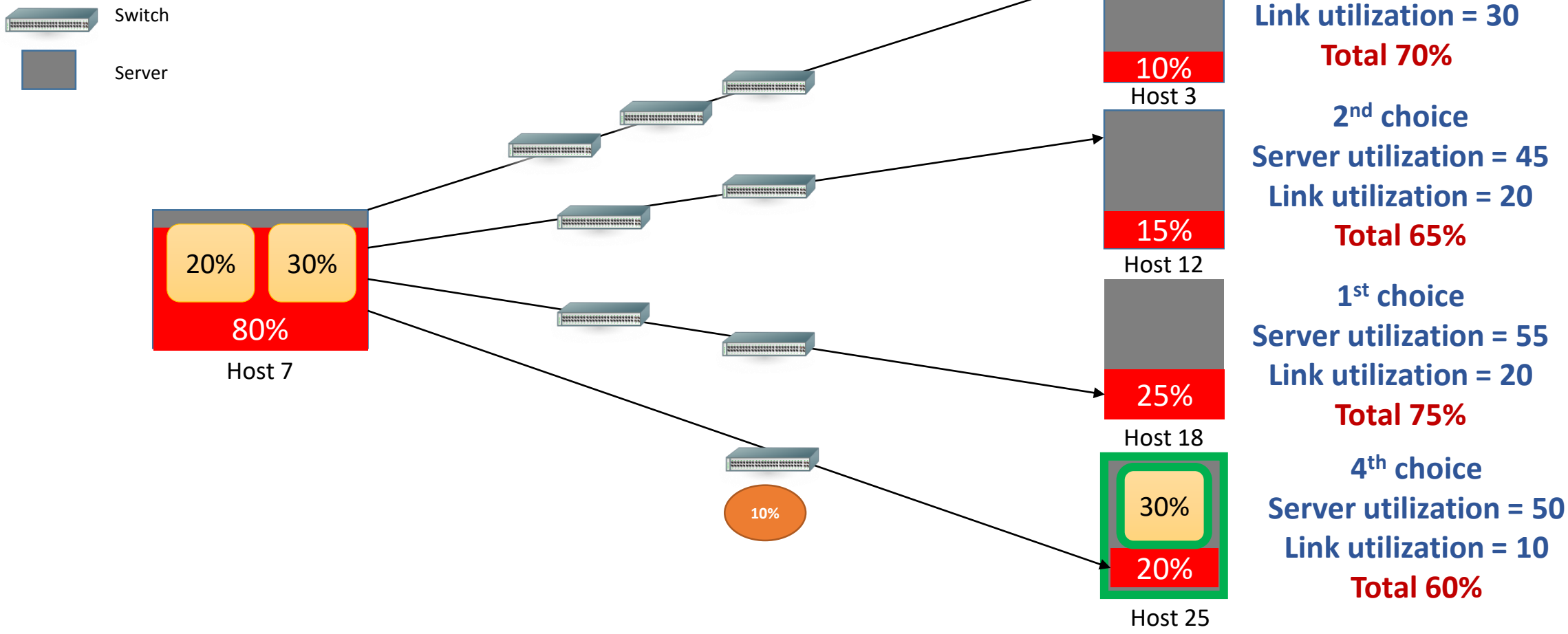
2nd choice
Server utilization = 45
Link utilization = 20
Total 65%

1st choice
Server utilization = 55
Link utilization = 20
Total 75%

A point to be noticed:
Having low server utilization is not always efficient
Link energy also matters

Reward – (Calculation Example)

- **Values are stored in Q-Table**



3rd choice
Server utilization = 40
Link utilization = 30
Total 70%

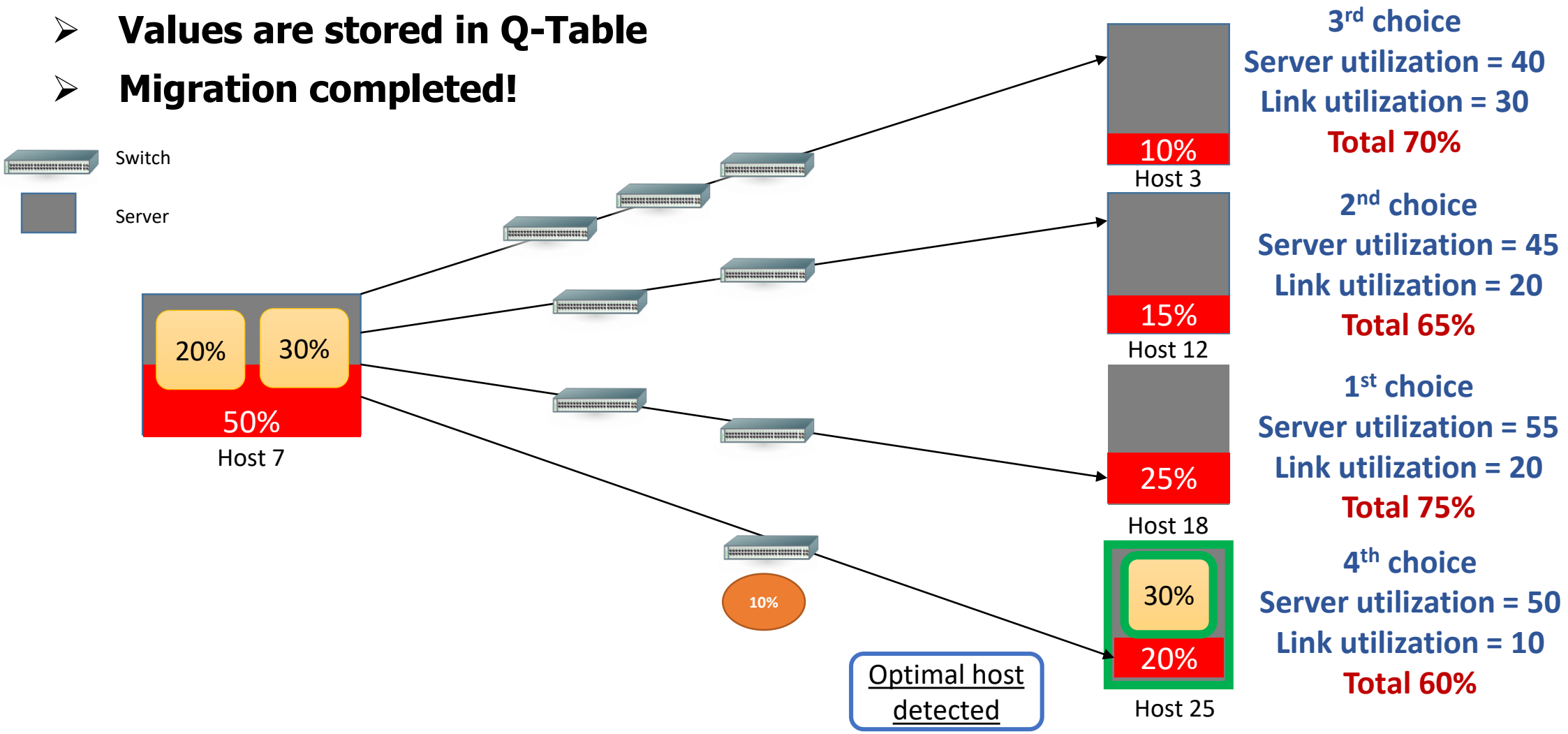
2nd choice
Server utilization = 45
Link utilization = 20
Total 65%

1st choice
Server utilization = 55
Link utilization = 20
Total 75%

4th choice
Server utilization = 50
Link utilization = 10
Total 60%

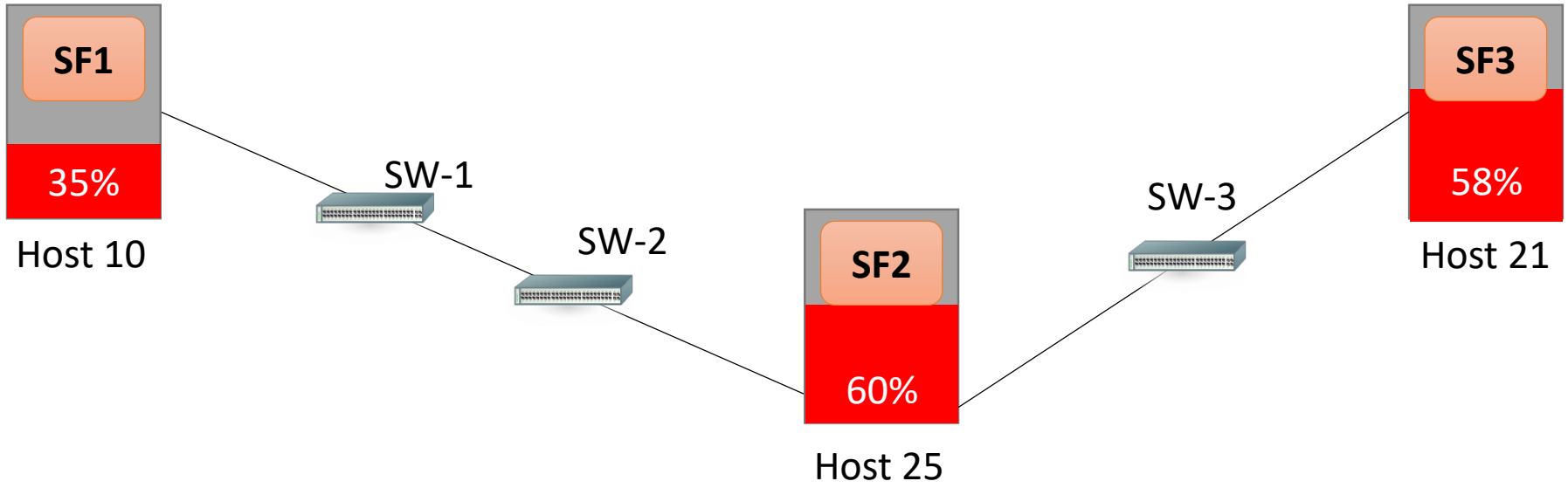
Reward – (Calculation Example)

- **Values are stored in Q-Table**
- **Migration completed!**



Final Reward ($E_{SP_i}^{physical}$ calculation)

Service Path, *SP*

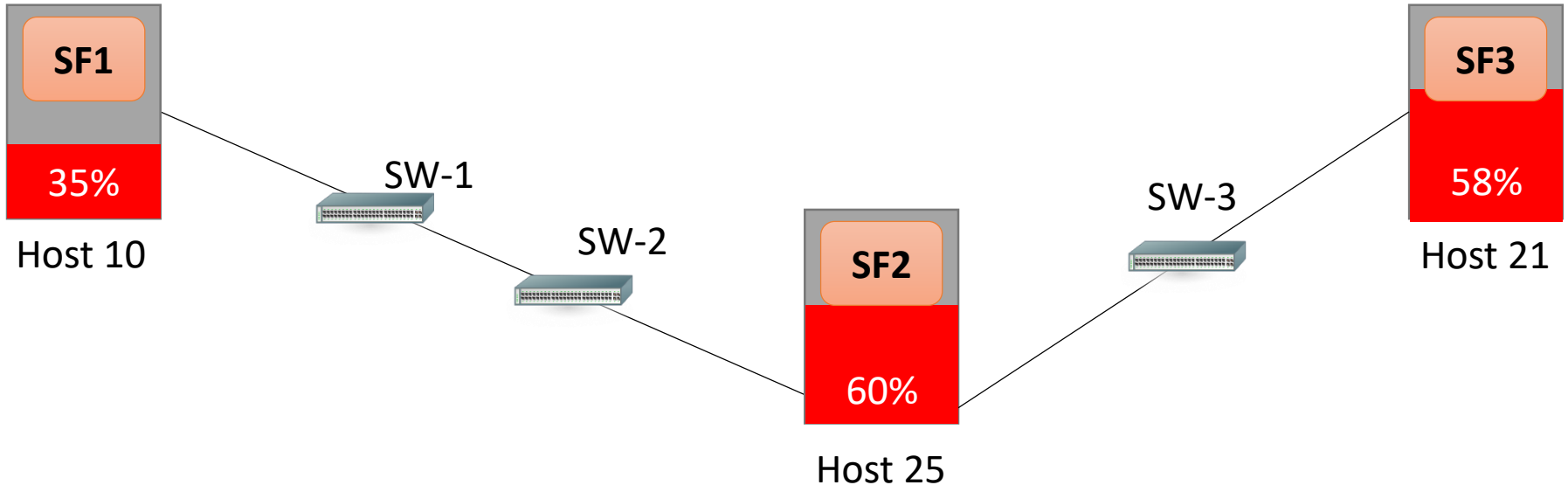


(1st choice) Path, ***SP*** = Host 10 –> SW-1 –> SW-2 –> Host 18 –> SW-3 –> Host 21
Optimal Path, ***SP*** = Host 10 –> SW-1 –> SW-2 –> Host 25 –> SW-3 –> Host 21
Server utilization with 1st choice: (0.35 + 0.75 +0.58) = 1.68
Server utilization with optimal choice: (0.35 + 0.6 +0.58) = **1.53 (reduced)**

The Lower the reward value the better the model (not considering link energy for this example)

Final Reward (Lat_{SP_i} calculation)

Service Path, *SP*

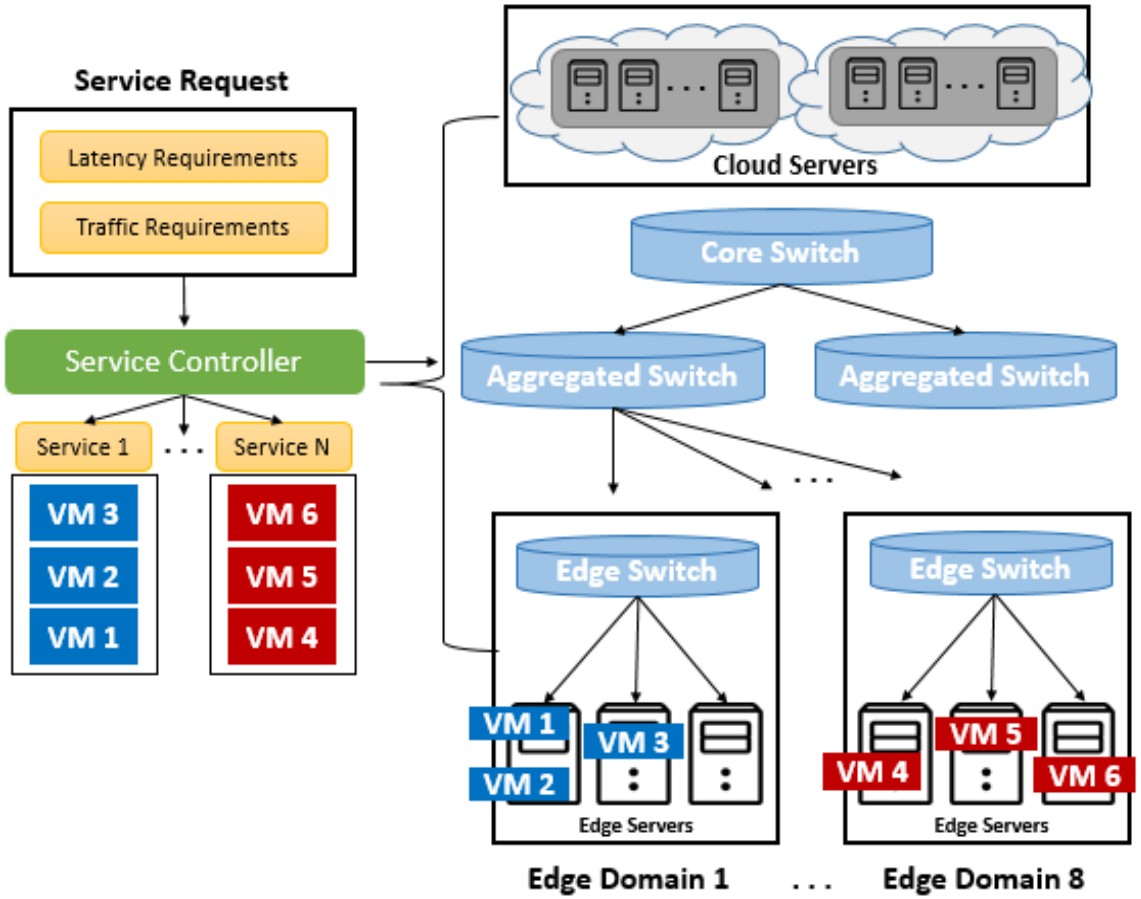


(1st choice) Path, ***SP*** = Host 10 –> SW-1 –> SW-2 –> Host 18 –> SW-3 –> Host 21
Optimal Path, ***SP*** = Host 10 –> SW-1 –> SW-2 –> Host 25 –> SW-3 –> Host 21
The higher the **BandWidth** the lower the latency value.

The Lower the reward value the better the model (not considering link energy for this example)

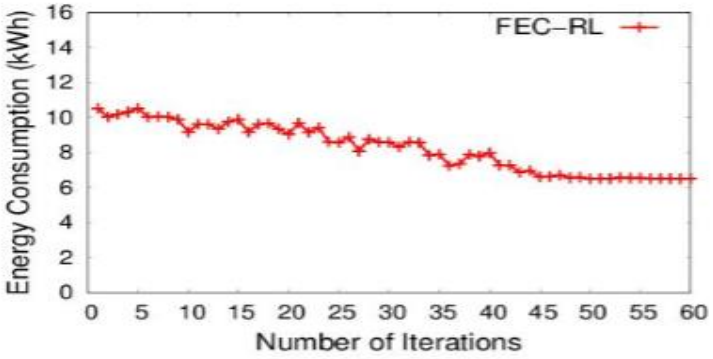
Evaluation

- **Simulator: CloudSimSDN**
- **FEC Topology**
  - 800 host servers
    - ✓ Distributed among 8 domains
  - PlanetLab(**for Traffic generation**)
    - VM utilization of 2 months (March, April)
    - CPU data from 3rd April, 2011
      - Low, Medium, High traffic
- **Reinforcement Learning**
  - Iterations: Total 60, for all traffic
  - learning rate: 0.05
- **Compared with**
  - Traditional **MAX** approach
    - Max Traffic requirement
    - Migration overhead not considered
  - **ESFEC-EF/MF**

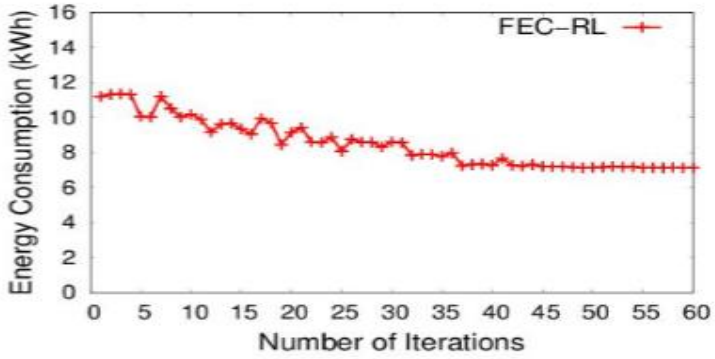


FEC-RL Convergence Model

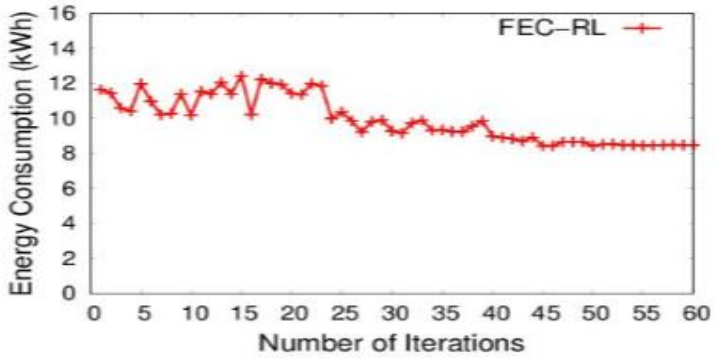
- The model converges starting from 50th iteration



(a) Low Traffic

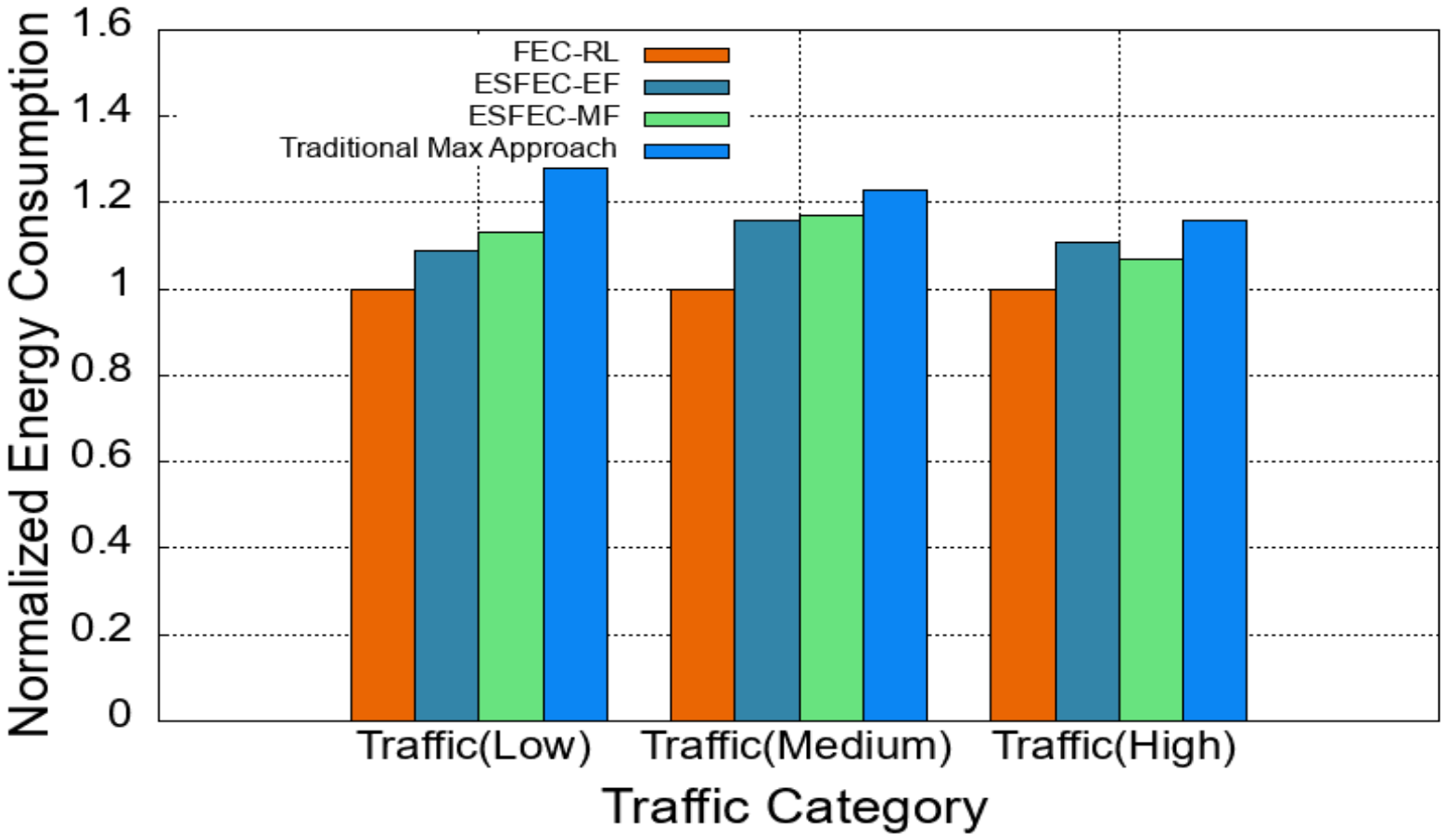


(b) Medium Traffic

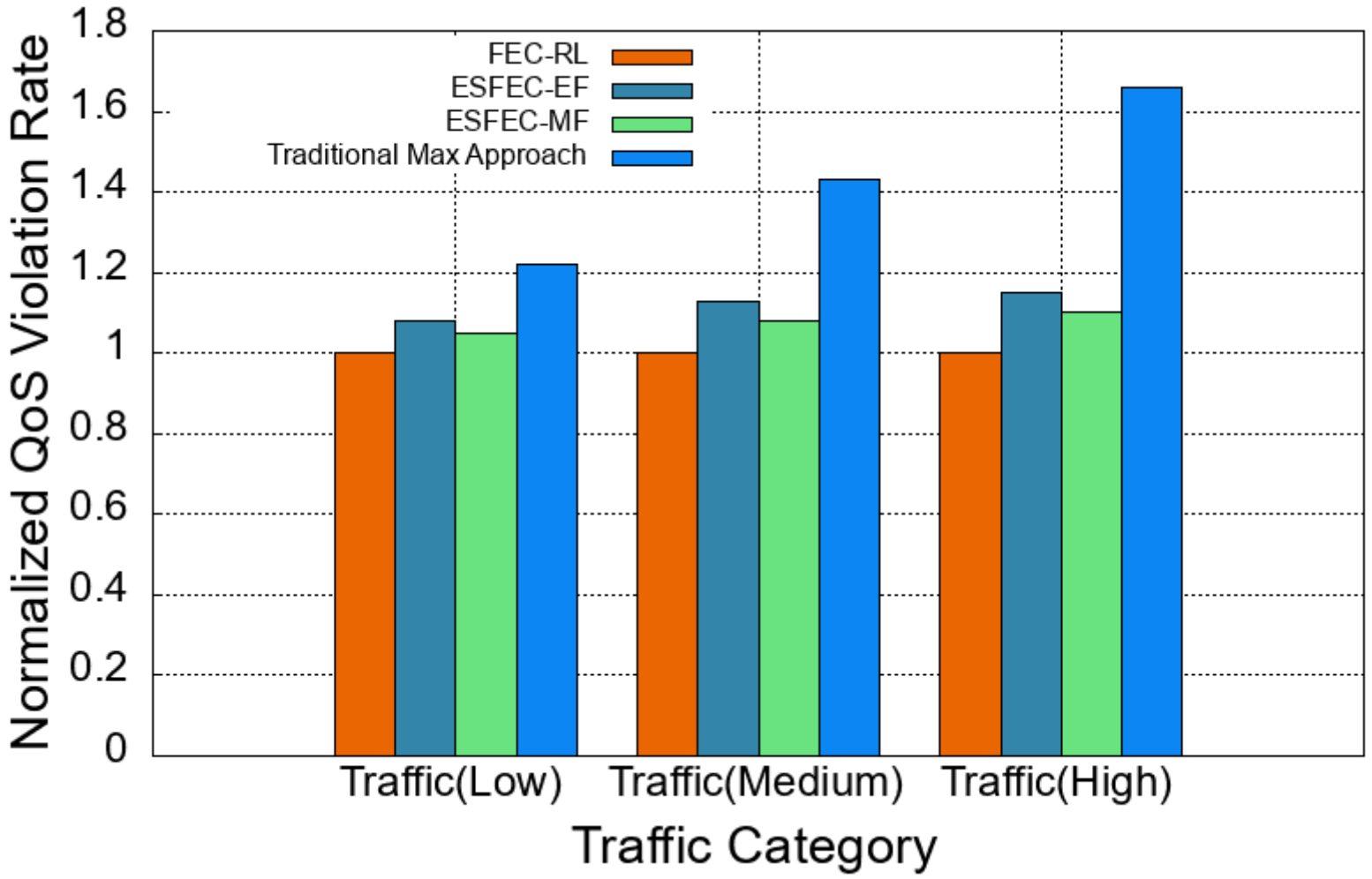


(c) High Traffic

FEC-RL Energy Efficiency

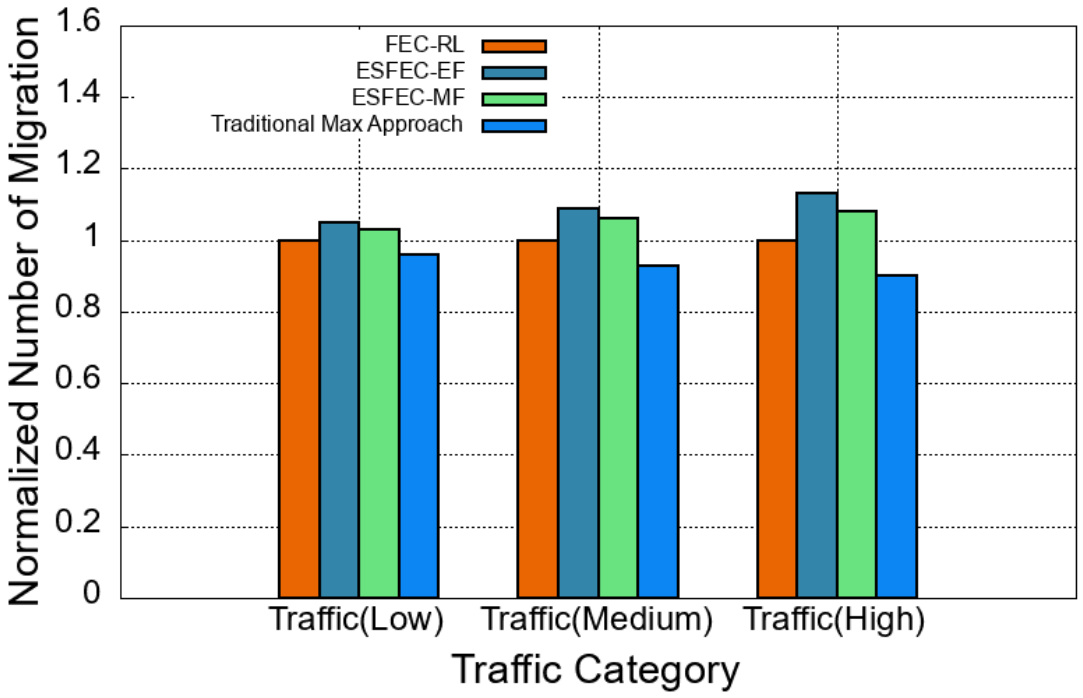


FEC-RL QoS Efficiency

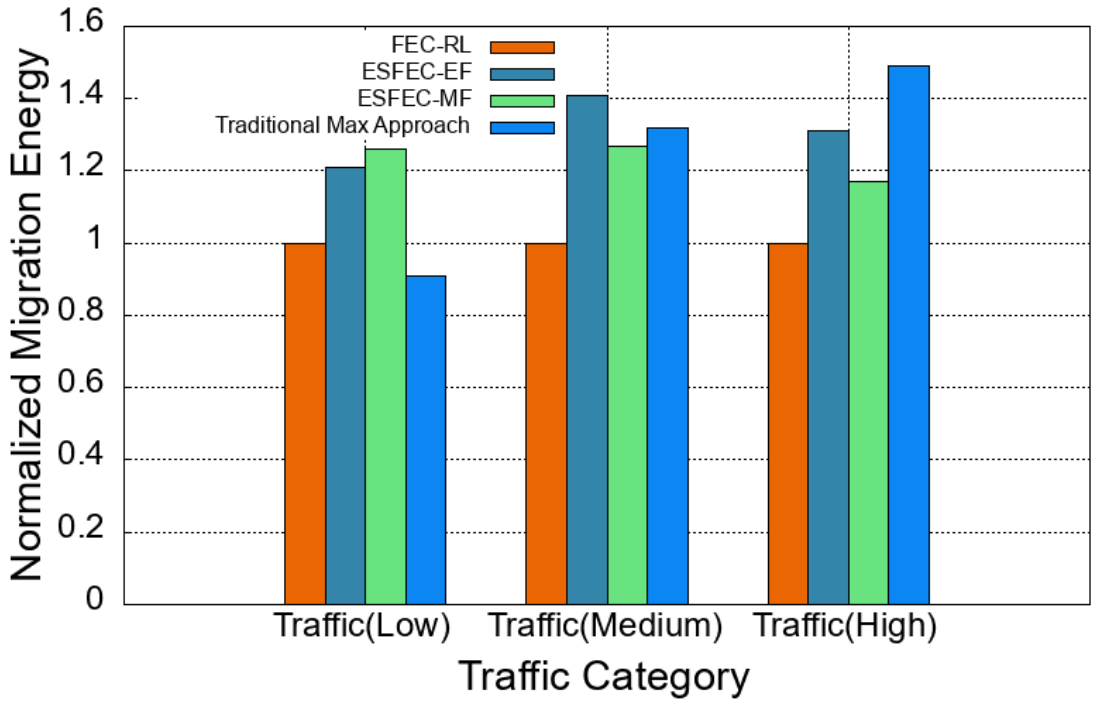


FEC-RL Migration number & Migration Energy

Migration Number



Migration energy



Conclusion

- **A learning based algorithm called FEC-RL**
  - In an unknown environment finding an optimal solution
  - Offered **Energy efficiency**
  - Offered **QoS** efficiency
  - Less **migration energy** consumed
  - Less migration **overhead**

Thank You