COGNITIVE RADIO NETWORK THROUGHPUT MAXIMIZATION WITH DEEP REINFORCEMENT LEARNING

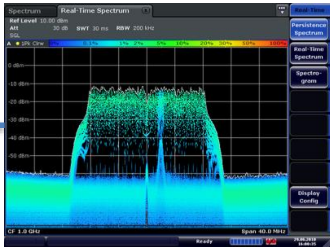
Kevin Ong, Zhang Yang, Dusit Niyato

22-Sep-2019

Background

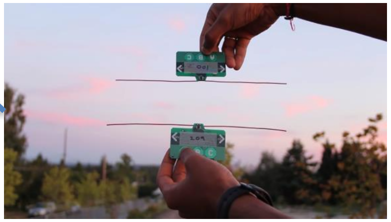
- Internet-of-Things (IoT) devices are forecasted at 50B devices by 2020.

KEY CHALLENGES



Wireless Spectrum Utilization

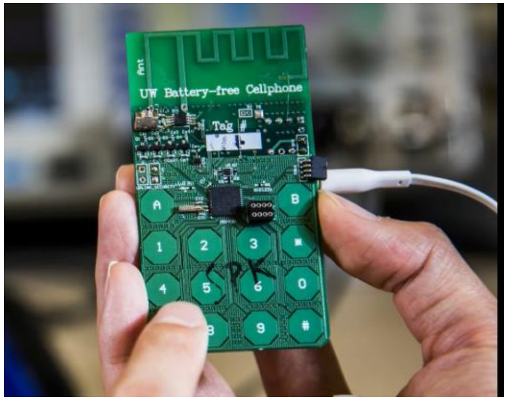
Security

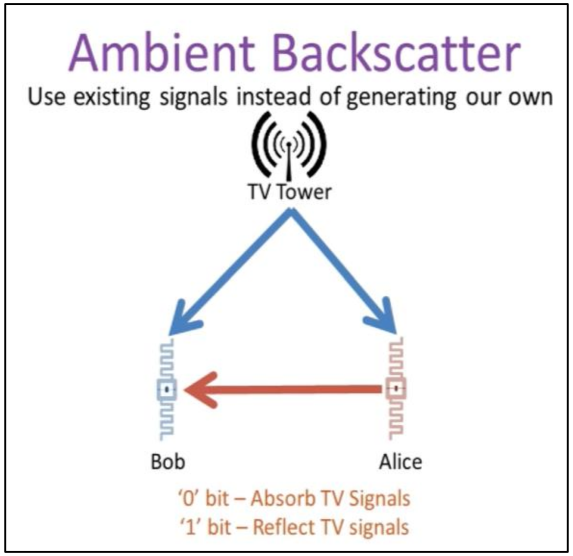


RF Energy Harvesting

Power Management

RF-Powered Cognitive Radio Networks with Ambient Backscatter





Motivation

- Low Data Throughput for Conventional CRN
  - ➢ Shared Channel Resource (Primary and Secondary Transmissions)

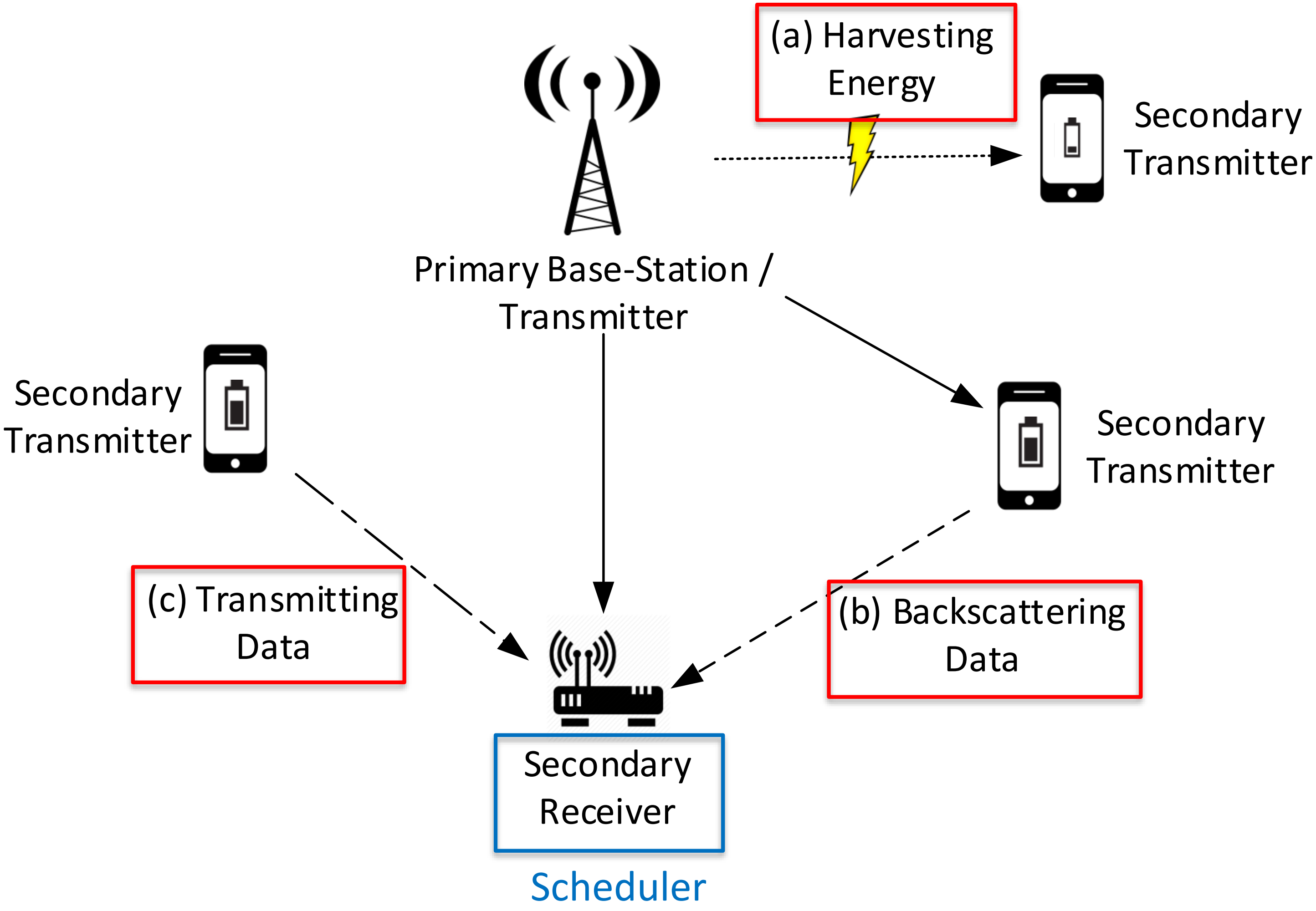
- Improved Management of Channel Resources with mode-switching enabled Secondary Transmitters (STs) :
  - ➢ Backscatter (***Back***)
  - ➢ Harvest-then-Transmit (***HTT***)
  - ➢ Active Transmission during Idle (***TX***)

Problem-Statement

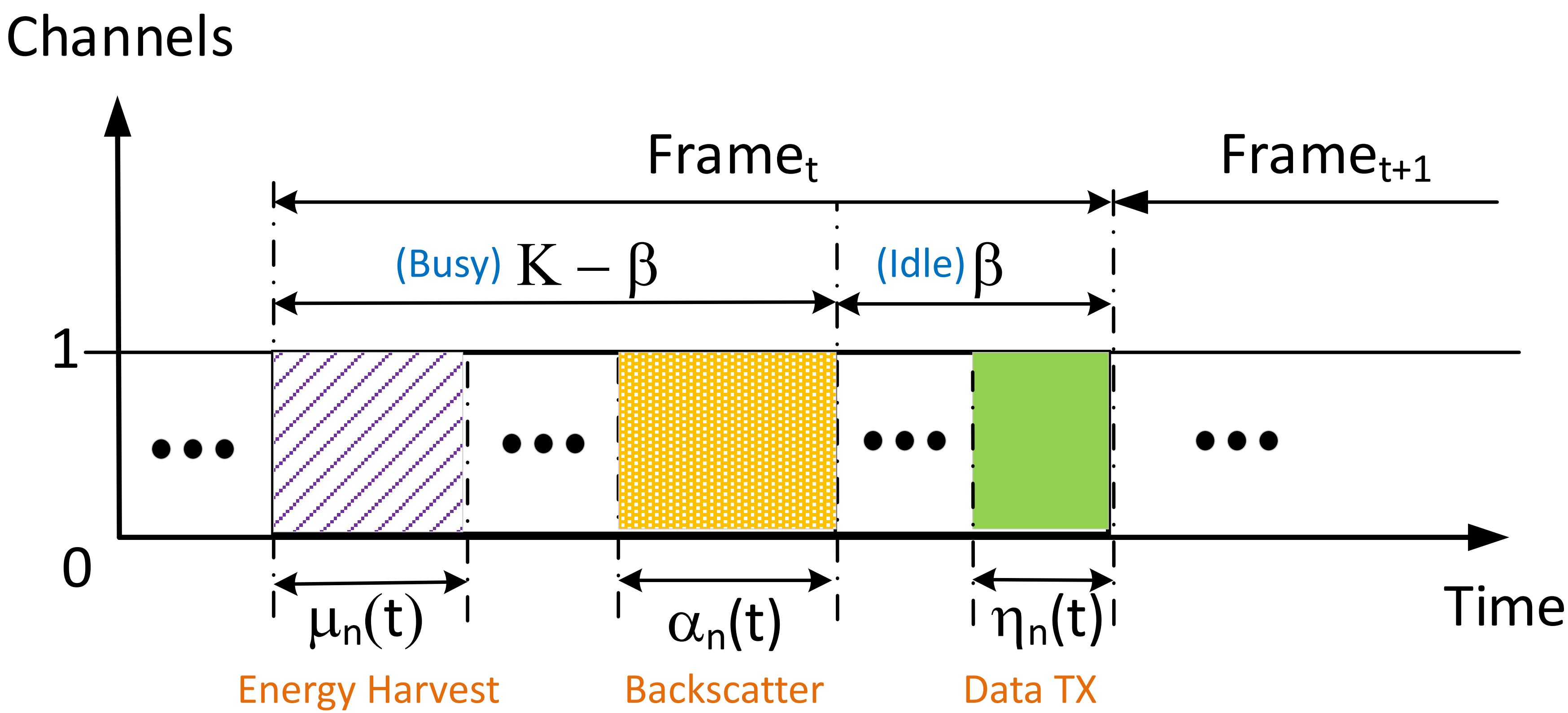
- Existing methods model the behaviour by assigning backscatter time as an auction resource, optimizing CRN data throughput as a concave function and ***Markov Decision Process (MDP)***[3].
- Recent prior work proposes Reinforcement-Learning[3] and tabular Q-Learning[4]:
  - ➢ Huge state space → ∞ Large Computation time
  - ➢ Unknown Channel States → Challenging to Model in MDP
  - ➢ Modelling large-scale RF powered CRN and devices becomes infeasible
- Deeper Neural Network = Consistently Better Performance?

[3] N. Van Huynh, D. T. Hoang, D. N. Nguyen, E. Dutkiewicz, D. Niyato, and P. Wang, "Reinforcement learning approach for RF-powered cognitive radio network with ambient backscatter," In 2018 IEEE Global Communications Conference (GLOBECOM) (pp. 1-6). IEEE.
[4] X. Wen, S. Bi, X. Lin, L. Yuan, and J. Wang, "Throughput maximization for ambient backscatter communication: A reinforcement learning approach," arXiv preprint arXiv:1901.00608, 2019.

System Model



System Model



SR observes and controls TX Scheduling of STs

Problem Formulation (MDP)

Objective Function:

$$\boxed{Throughput_{max} = \sum_{ST=1}^{N} PacketsTransmitted \quad (1)}$$

State Space:

of pkts queue

Energy Units

ST $\boxed{\mathcal{S}_n = (q_n, c_n)}$

CRN network $\boxed{\mathcal{S} = \mathcal{S}^c \times \prod_{n=1}^{N} \mathcal{S}_n}$

Busy slots (i.e channel state)

Action Space:

$$\mathcal{A} = \left\{ \begin{array}{l} (\mu, \alpha_1 ..., \alpha_N, \eta_1, ..., \eta_N)| \\ \mu + \sum_{n=1}^{N} \alpha_n \leq b, \mu + \sum_{n=1}^{N} (\alpha_n + \eta_n) \leq K \end{array} \right.$$

State Transition Probability Distribution:

Described in Eqns (5) to (8)

Packet Arrival follows Binomial Distribution Eqn (9)

Problem Formulation (MDP)

Reward Function:

pkts TX wrt op-mode

$$\mathcal{R}(s,a) = \underbrace{\sum_{n=1}^{N} S_n^b (q_n^{(1)} - q_n)}_{\text{backscatter}} + \underbrace{\sum_{n=1}^{N} S_n^a (q_n^{(2)} - q_1)}_{\text{active}}$$

Derive Optimal Policy (π*) by maximizing value-state function

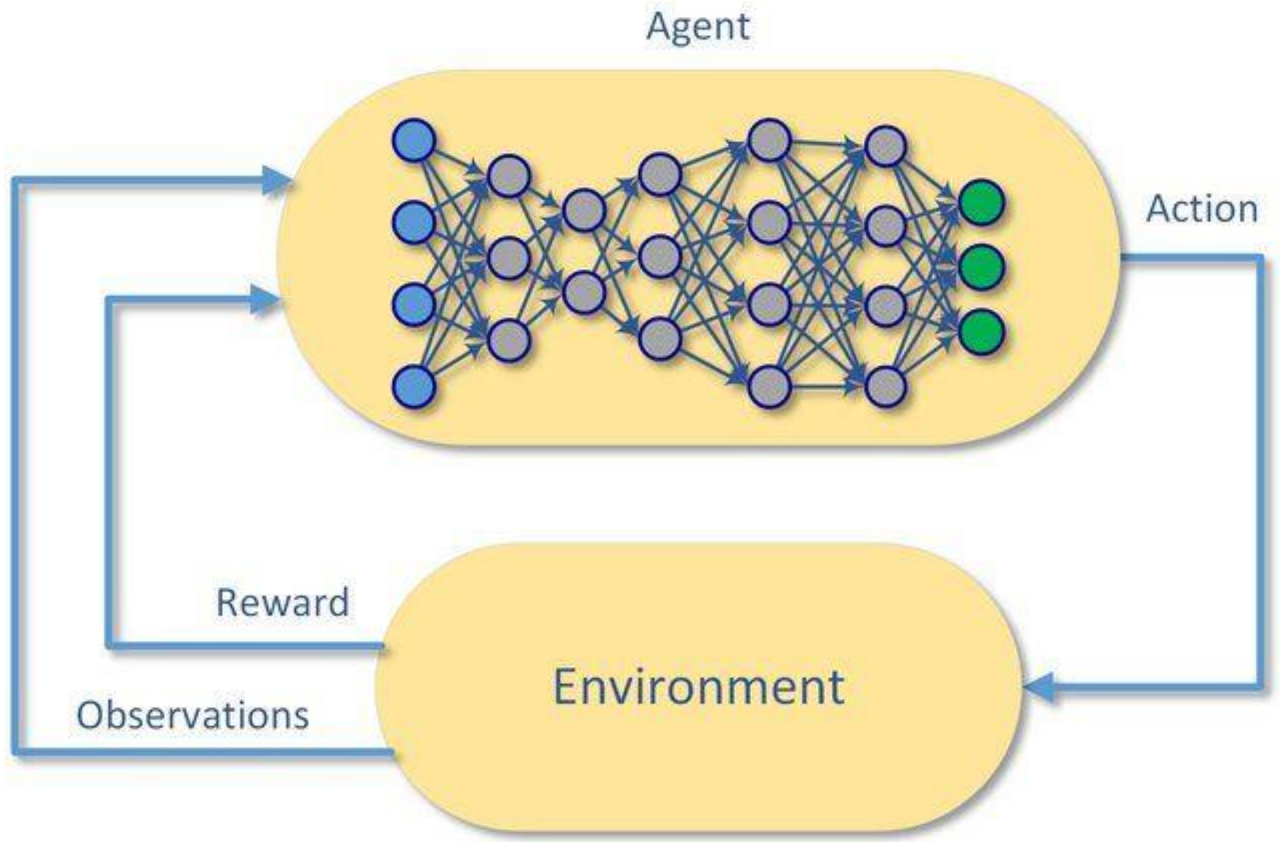
$$\mathcal{V}(s) = \sum_{s' \in S} \mathcal{P}_{\pi(s)}(s,s')(\mathcal{R}(s,a) + \gamma \mathcal{V}(s'))$$

Update Q^{new} using Bellman Equation

$$\overbrace{\mathcal{Q}^{new}(s,a)}^{\text{New Q value}} = (1-\alpha) \overbrace{\mathcal{Q}(s,a)}^{\text{Current Q value}} + \alpha[\overbrace{\mathcal{R}(s,a)}^{\text{Reward received}} + \gamma \underbrace{\max_{a' \in \mathcal{A}} \mathcal{Q}'(s',a')}_{\text{Max(Expected future reward)}}]$$

DQN

Deep Reinforcement Learning Approach - DQN



Source: https://www.researchgate.net/publication/319121340/figure/fig3/AS:547264844500992@1507489514833/A-conceptual-structure-of-a-deep-reinforcement-learning-system.png

Why Deep Q-Networks?

- Agent's intelligence is stored in Q-table.
  - Exponential Computation and Memory Resource requirements for environments with large State-Action pairs.

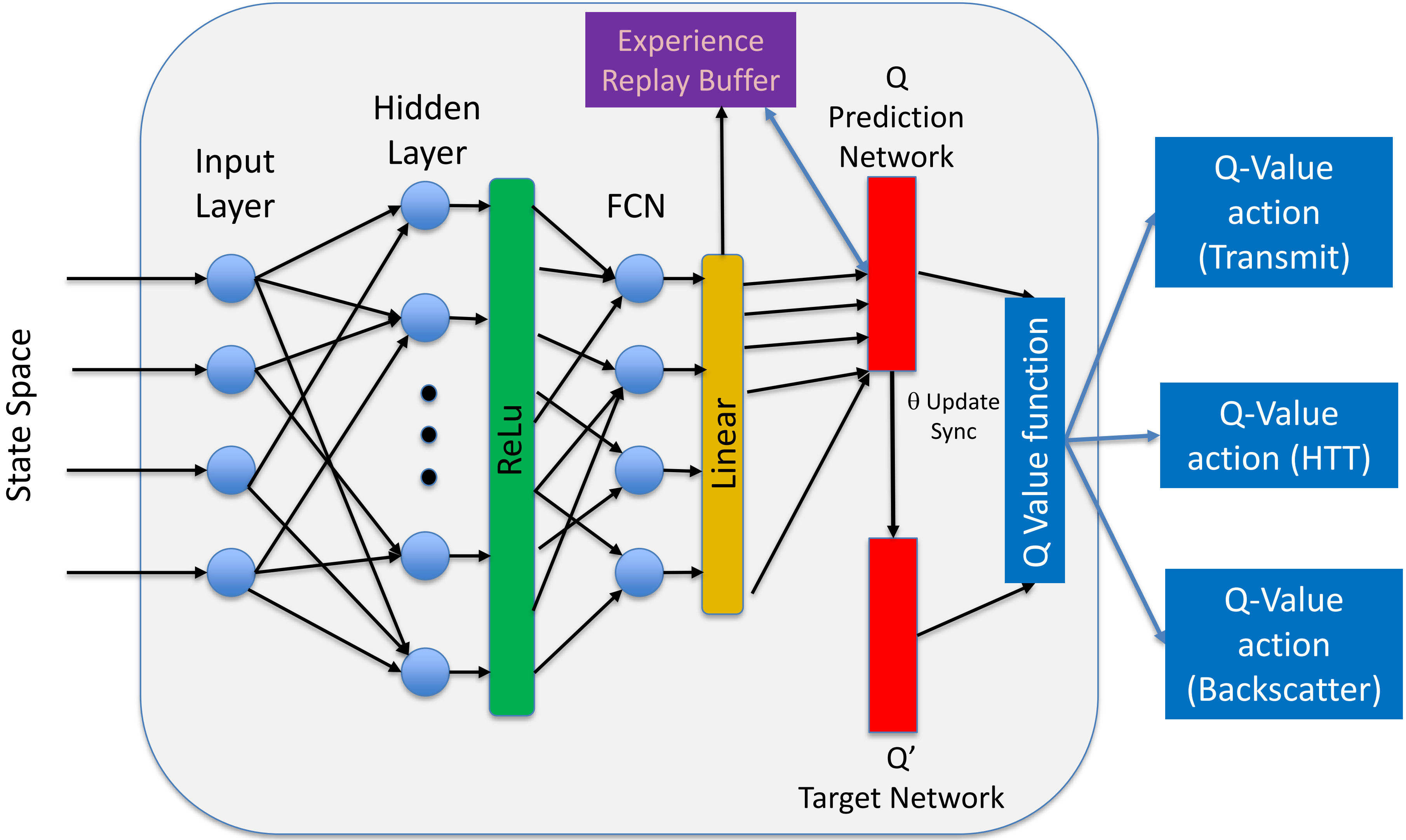
- Minh et al. [2] proposed the idea of using neural network to approximate the Q-value function.

- Specifically, the backpropagation will be used to update its gradient and converge to a solution in the Bellman equation:

$$\alpha\left[\boxed{R_{t+1}+\gamma \max_{a} Q(S_{t+1},a)}\right.$$

[2] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G. and Petersen, S., 2015. Human-level control through deep reinforcement learning. Nature, 518(7540), p.529.

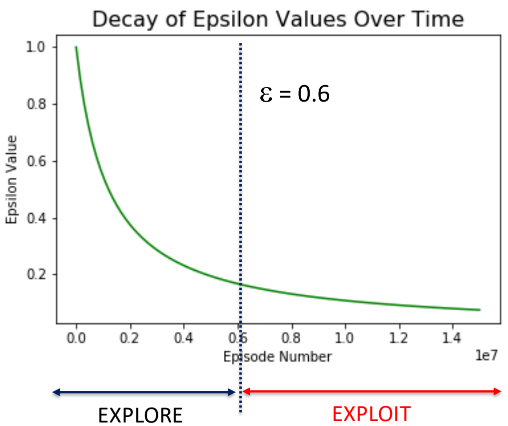
Proposed DRL Architecture



Algorithm 1 Deep Q-Learning with Experience Replay for Gateway Time-scheduling

Input: Action space $\mathcal{A}$, mini-batch size L_b, target network replacement frequency L^-
Output: Optimal policy π^* for N Secondary Transmitters
Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$
for Episode=1 to E **do**
    Initialize sequence $s_1 = x_1$ and preprocessed sequence $\Phi_1 = \Phi(s_1)$
    for timestep=1 to T **do**
        Choose an action a_t
        With probability ϵ, a random action is performed
        Otherwise, choose $a_t = argmax_a Q(\Phi(s_t, a)$ from $Q(s, a; \theta)$
        Broadcast messaging time-schedules for N secondary transmitters
        Execute chosen action a
        Receive reward r
        Receive state messages from primary transmitter and N secondary Transmitters
        Update next network state s'
        Store tuple (s, a, r, s') in replay memory D
        Randomly sample tuple (ss, aa, rr, ss') of mini-batch size (L_b) from replay memory $\mathcal{D}$
        Calculate target Q-value for each mini-batch transition
        $y_t^{DQN} = \begin{cases} r, \text{if episode i terminates at timestep+1} \\ r + \gamma max_{a'} \hat{Q}(\phi_{j+1}, a', \theta^-), \text{else} \end{cases}$
        Train the Q-Network using $(y_t^{DQN} - Q(ss, aa)^2)$ as loss and update the weights θ
        Reset $\theta^- = \theta$ every L^- steps
        Update $s \leftarrow s'$
        Increment timestep by 1
    repeat until timestep is > T, terminate
repeat until Episode is > E, terminate

Epsilon-Greedy Policy



Experience Replay Memory Buffer

- Stores state-transition tuple that leads to best reward
- Buffer size affects speed of learning and quality

Setup and HyperParameters

Objective

To evaluate and propose Hyperparameter combinations which yields high data throughput for the given system model and benchmark against state-of-the art.

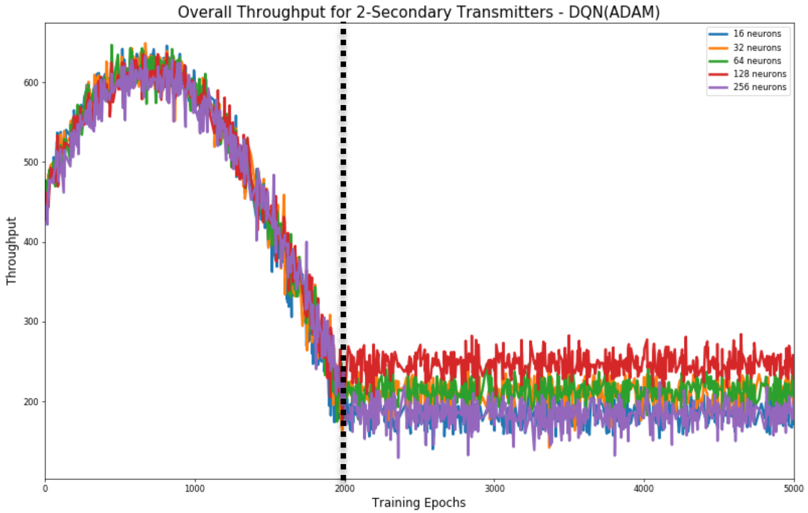
Parameter	Value
Hidden Layers	1(DQN), 3(Comparison)
Number of Hidden Neurons (H_n)	16, 32 ,64, 128, 256
Optimizer	Adam, SGD
ϵ-Greedy decay	0.9$\rightarrow$0
ϵ-Greedy decay steps	4×10^5
Learning Rate (α)	$1e^{-3}$, $1e^{-4}$
Discount rate (γ)	0.9
Target Network Update Rate	$1e^{-4}$
Mini-batch size	32
Replay Memory size	5×10^5
Iteration steps per Episode	200
Training iterations	10^6
Secondary Transmitters (N)	2,3
Time slots within single time frame	10
Idle time slots within single time frame	[1;9]
Packet Arrival Probability (λ_n)	[0.1;0.9]

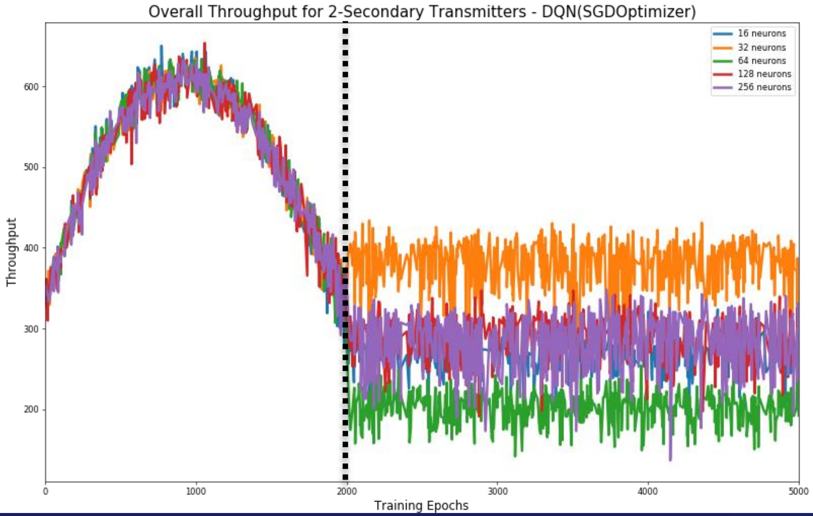
TABLE I: DQN Model Simulation Parameters

Additional Information

- Mean Average over 10 runs.
- Random Policy results are omitted as previous comparisons were reported in [16].
- Optimal policy is assumed when results do not experience large fluctuations for 100 episodes, after exploration steps have elapsed (ε=0).
- Results for 2ST and 3ST scenario

Results (Performance Comparison of Optimizer Selection wrt HL size)



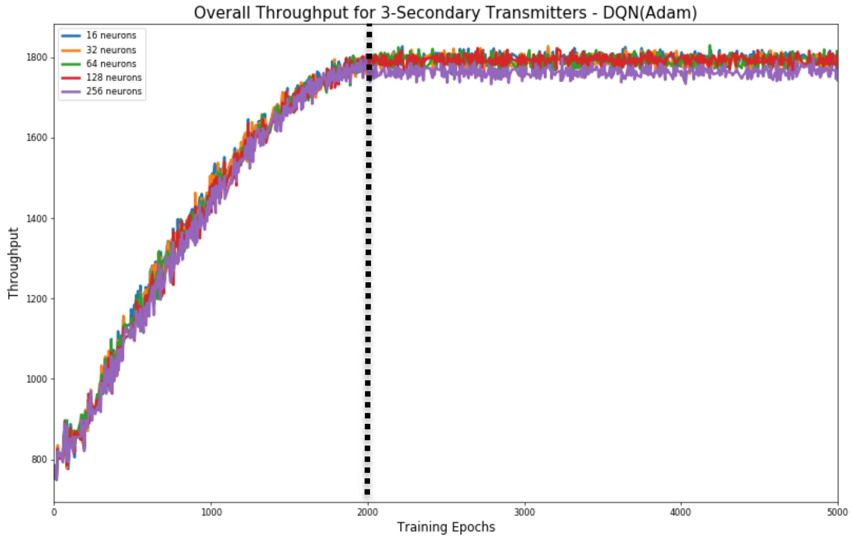


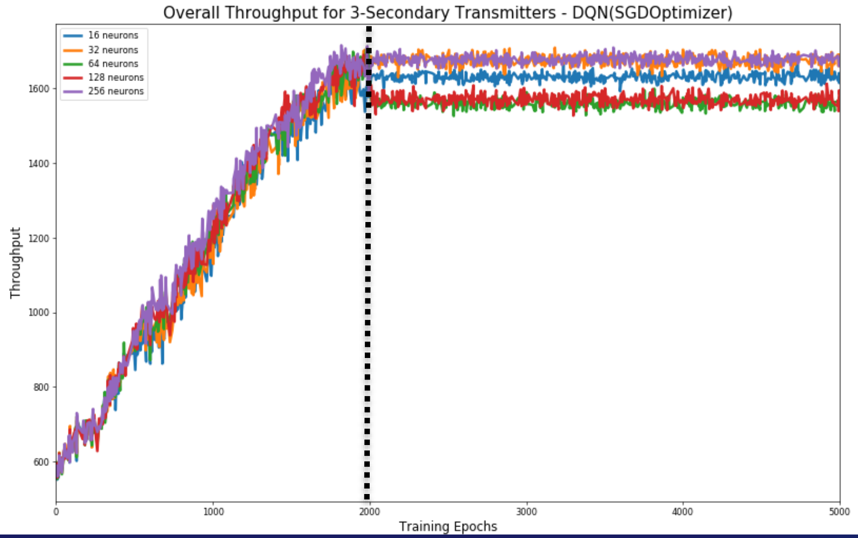
- Results are for **2STs**.

Environment	Number of Neurons	Adam	SGD	Speedup
2ST	16	183	269	~1.5x
2ST	32	210	379	~1.8x
2ST	64	212	203	~0.96x
2ST	128	246	288	~1.2x
2ST	256	184	283	~1.5x

$$Speedup = \frac{SGD\ Results}{ADAM\ Results}$$

Results (Performance Comparison of Optimizer Selection wrt HL size)





- Results are for **3STs**.

Environment	Number of Neurons	Adam	SGD	Speedup
3ST	16	1794	1631	~0.91x
3ST	32	1792	1675	~0.93x
3ST	64	1792	1561	~0.87x
3ST	128	1792	1571	~0.88x
3ST	256	1763	1678	~0.95x

Benchmark Results with Advanced DQN

Environment	DQN Method	Optimizer	Hidden Neurons	Layers	Mean Throughput(pkts)	Speedup wrt DoubleDQN
2ST	DQN-SGD32	SGD	32	1	379	~3.1x
2ST	DQN-Adam128	Adam	128	1	246	~2.0x
2ST	DoubleDQN	Adam	32	3	124	NA
2ST	DuelDQN	Adam	32	3	224	~1.8x
2ST	DoubleDuelDQN	Adam	32	3	173	~1.4x

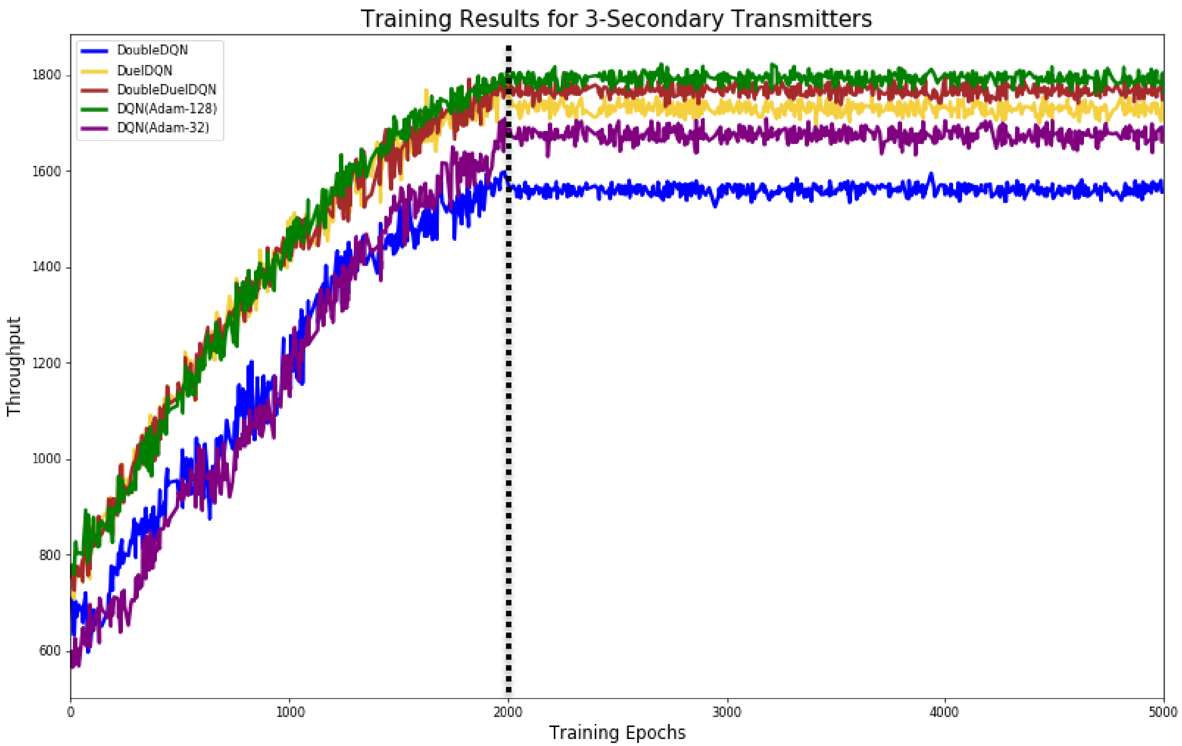
$$Speedup = \frac{Proposed\ Method\ Results}{DoubleDQN\ Results}$$

- For **2STs**
- Better performance than deeper networks in [16]

Benchmark Results with Advanced DQN

Environment	DQN Method	Optimizer	Hidden Neurons	Layers	Mean Throughput(pkts)	Speedup wrt DoubleDQN
3ST	DQN-SGD32	SGD	32	1	1675	~1.07x
3ST	DQN-Adam128	Adam	128	1	1793	~1.15x
3ST	DoubleDQN	Adam	32	3	1560	NA
3ST	DuelDQN	Adam	32	3	1731	~1.11x
3ST	DoubleDuelDQN	Adam	32	3	1767	~1.13x

- For **<u>3STs</u>**.



Discussion

- Empirical proof for proposed DQN configurations:
  - Optimizer choice affects simulation results
  - Performance speed-ups → 1.07x to 3.1x
  - Deeper Neural Network $\neq$ better performance
  - Implicit Reduction in Model Training Time

- Explanations for better performance:
  - Low Problem Dimensionality and Complexity [1]
  - Optimized Hyper-parameters (one of many combinations)

Source: [1] Mhaskar, Hrushikesh, Qianli Liao, and Tomaso Poggio. "When and why are deep networks better than shallow ones?." In Thirty-First AAAI Conference on Artificial Intelligence. 2017.

Future Work

- Future research work could include:
  - Performance evaluation for increasing number of STs.
  - Slightly more complex model, such as multiple STs and SRs.
  - Multi-channel.
  - Test on Real datasets.

Thank you!

Source: [Logo]