Learning Algorithms for Optimal Network Control

Eytan Modiano
MIT, LIDS

Role of Learning in Network Control

- Learning network state and dynamics
  - Channels, connectivity, delays, etc.
  - Multi-arm bandit framework

    "multi-arm bandits with queues"

- Network control in uncooperative environments
  - Some of the nodes are uncontrollable and/or unobservable
  - Network optimization subject to stochastic queueing dynamics

- Performance optimization (i.e., delay)
  - Use ML/RL to solve stochastic optimization problem with large state space
  - Optimal routing, scheduling, etc.

- Control in adversarial environments
  - Nodes intentionally take adversarial actions

    "online learning framework"
  - Networks under attack (DoS, traffic injection)

Learning-based network control (talk outline)

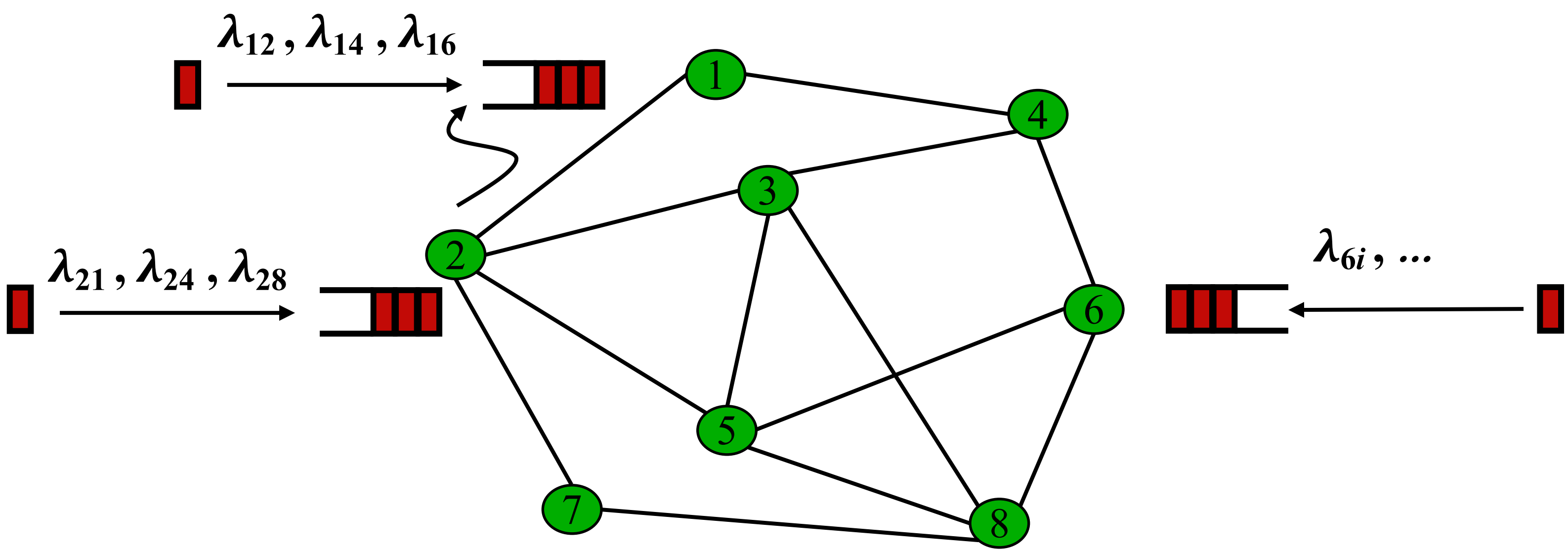
- Tracking Max-Weight (TMW): Learning-aided Max-Weight algorithm
  - Need to learn unknown underlay dynamics
  - Focus on network stability

- Gradient sampling Max-Weight: Learning-based network utility maximization
  - Need to learn unknown utility functions
  - Feedback/actions subject to queueing delay
  - Application to delay minimization

- Reinforcement learning algorithm for queueing networks
  - General optimal control framework for queueing systems

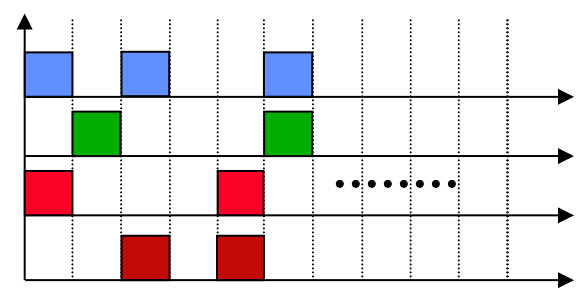
Network Model

- Multi-hop wireless network: Only a subset of the links can be activated simultaneously, due to interference
  - Need to make packet routing and link scheduling decisions



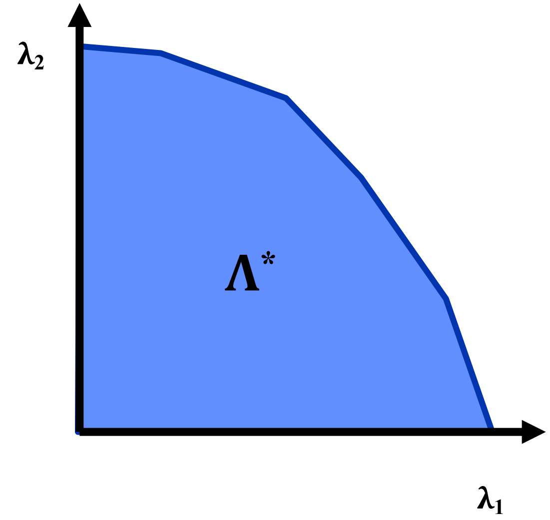
- Random arrivals with arrival rates λ_c
  - The λ_c's are not known in advance

- Time-slotted system



Throughput Maximization

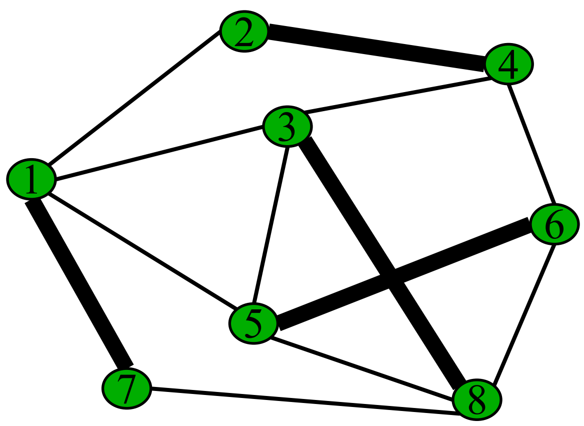
- Goal: Design a routing and scheduling policy that can support all arrival rates within the network stability region



- Stability Region (Λ^*) – the set of all admissible arrival rate vectors
  - There exists some policy that will "stabilize" the network with these arrivals
- Notions of stability
  - Bounded queue occupancy
  - Existence of steady state distribution
  - Rate stability: arrival rate = departure rate
- Tassiulas/Ephremides '92
  - Scheduling and routing algorithm that stabilizes the network under unicast traffic

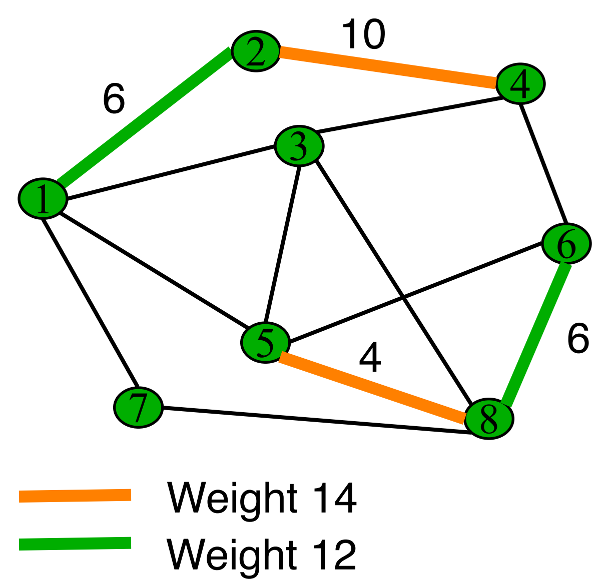
The Max-Weight Scheduling Algorithm (Tassiulas/Ephremides ’92)

- Only a subset of the links can be activated simultaneously. E.g.,
  - Primary interference constraints
    - A node transmits to a single neighbor at a time
    - Multiple transmissions can take place as long as they do not share a common node (e.g., Bluetooth)
  - Secondary (2-Hop) interference constraints
    - No two edges can be active if they can be joined by one or fewer edges (e.g., 802.11)



- Throughput optimal scheduling
  - Schedule the max-weight activation set in each time-slot
  - Weights are the queue backlogs

$$\pi^* = \arg\max_{\pi \in \Pi} \sum_{(i,j)} Q_{ij}(t)\pi_{ij}$$

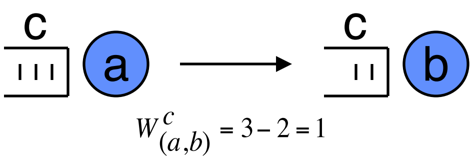


The Backpressure Routing Algorithm (Tassiulas/Ephremides ' 92)

- Route based on commodities: each commodity $C \in \{1,..,N\}$ corresponds to data associated with a given destination node

- Along each link (a,b) route commodity C that maximizes the differential backlog along that link. i.e.,

$$W^*_{(a,b)} = \max_{c\in\{1..N\}} W^c_{(a,b)} = (U^c_a - U^c_b)$$



  - Algorithms uses "back pressure" to find the routes

- Link activation: max-weight rule with differential backlogs as weights

  - Joint routing and scheduling $\pi^* = \arg\max_{\pi\in\Pi} \sum_{links(a,b)} W^*_{(a,b)}(t)\pi_{(a,b)}$

- Backpressure "learns" the "optimal" routes and schedules using queue backlog as feedback
  - Requires all nodes to cooperate:
    - Share queue information
    - Implement the same policy

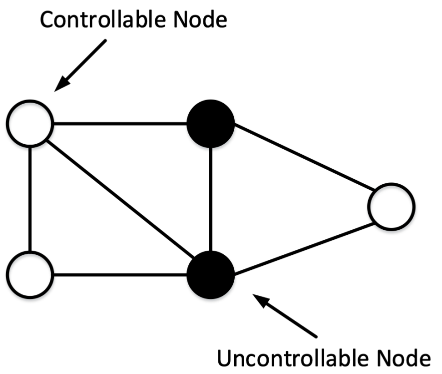
Impact of Uncontrollable Nodes

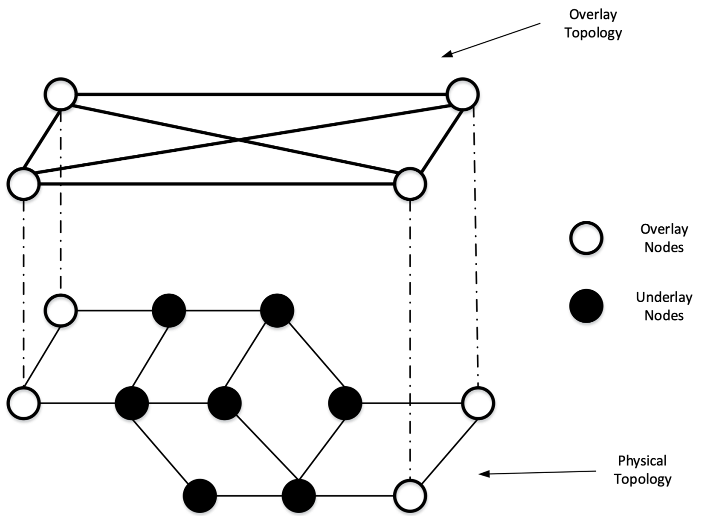
- Increasingly networks are only partially controllable

- A subset of nodes are not managed by the network operator and could use some unknown network control policy

- Existing optimal control policies may yield poor performance

- **Overlay-underlay network:** MaxWeight algorithm may lead to throughput loss



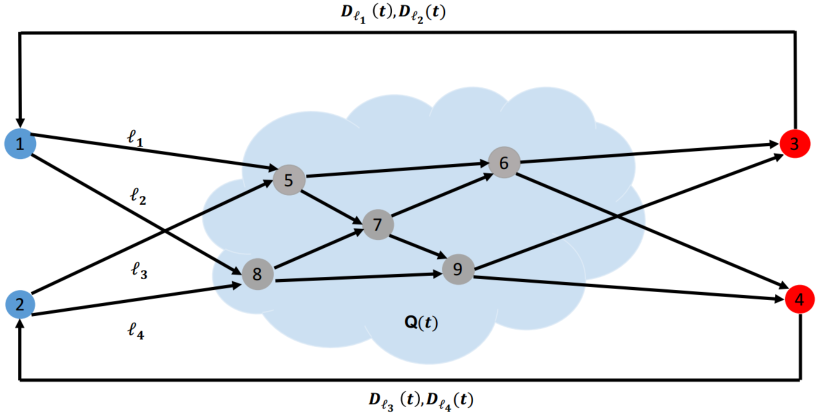


Overlay Framework

- Overlay architecture is extremely common
  - Operate over a black-box whose internal dynamics are not known and may not even be observable
  - e.g., over the top service providers, coalition networks

- Network control based on end-to-end feedback
  - Need to learn the dynamics of the underlay network

- Approach: a combination of reinforcement learning and Lyapunov optimization to develop control algorithms based on end-to-end feedback
  - Stability: keep queues bounded
  - Utility maximization



Learning-based network control (talk outline)

- **Tracking Max-Weight (TMW): Learning-aided Max-Weight algorithm**
  - Need to learn unknown underlay dynamics
  - Focus on network stability
- Gradient sampling Max-Weight: Learning-based network utility maximization
  - Need to learn unknown utility functions
  - Feedback/actions subject to queueing delay
  - Application to delay minimization
- Reinforcement learning algorithm for queueing networks
  - General optimal control framework for queueing systems

Q. Liang, E. Modiano, "Optimal Network Control in Partially-Controllable Networks," Infocom, 2019.
B. Liu, Q. Liang, E. Modiano, "Tracking MaxWeight: Optimal Control for Partially Observable and Controllable Networks," IEEE/ACM Transactions on Networking," 2023.

Model

- Consider a queueing network with N nodes and K flows
- $Q_{ik}(t)$ is the queue length of flow k at node i in slot t
- In each time slot t, we observe a network event ω_t which includes information about link capacities, external packet arrivals, etc.
  - $\{\omega_t\}_{t\geq 0}$ follow a stationary stochastic process
- Each node i needs to make a routing decision $f_{ijk}(t)$ indicating the offered transmission rate for flow k over link $i \rightarrow j$
  - $\tilde{f}_{ijk}(t)$ = actual transmitted packets, may be smaller than $f_{ijk}(t)$

Partially-Controllable Network

- The set of all nodes is denoted by $\mathcal{N}$
  - Network routing vector is $\boldsymbol{f}(t) = \{f_{ijk}(t)\}_{i \in \mathcal{N}}$
- The set of **controllable** nodes is denoted by $\mathcal{C}$
  - Controllable action $\boldsymbol{f}^c(t) = \{f_{ijk}(t)\}_{i \in \mathcal{C}}$
  - Controllable policy $\pi_c: (\omega, \boldsymbol{Q}) \mapsto \boldsymbol{f}^c$
- The set of **uncontrollable** nodes is denoted by $\mathcal{U}$
  - Uncontrollable action $\boldsymbol{f}^u(t) = \{f_{ijk}(t)\}_{i \in \mathcal{U}}$
  - Uncontrollable policy $\pi_u: (\omega, \boldsymbol{Q}) \mapsto \boldsymbol{f}^u$

Objective: design controllable policy π_c such that the entire network is rate stable:

$$\lim_{t \to \infty} \frac{\mathbb{E}[Q_{ik}(t)]}{t} = 0, \qquad \forall i \in \mathcal{N}, k.$$

Queue-Agnostic Uncontrollable Policy

- **Queue-agnostic** uncontrollable policy (ω-only policy):

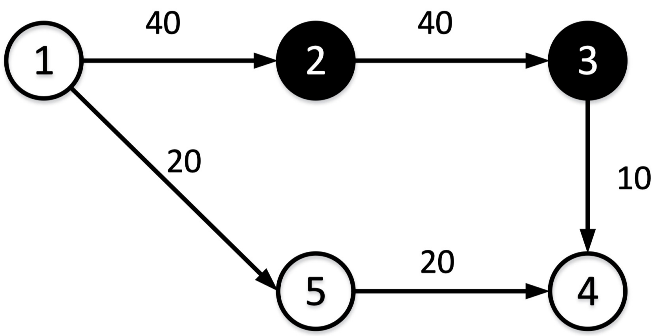
$$\pi_u : \omega \mapsto \boldsymbol{f}^u$$

- Uncontrollable node simply observe the current network event ω_t and makes a routing decision
  - "stateless"
- Simple yet cover a wide range of practical protocols:
  - Shortest-path routing (OSPF, RIP)
  - Multi-path routing (ECMP)
  - Randomized routing

Failure of BP Algorithm

Failure of Backpressure (BP) Algorithm

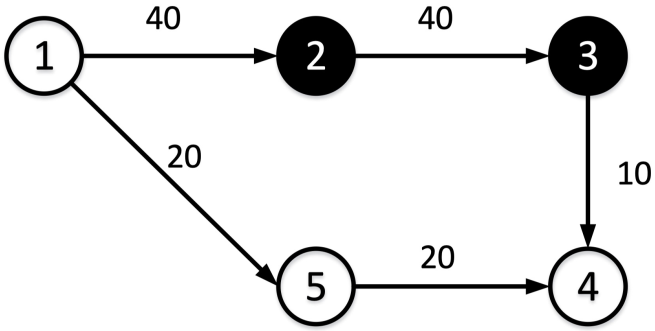
- Each node can only transmit to one of its neighbors in each slot.
- Only one flow: 1 → 4 (with rate 20)
- Uncontrollable node 2 transmits to node 3 at full line rate.
- Uncontrollable node 3 holds any packets it received.
- Backlogs are always zero at node 2, so BP always sends packets to node 2 although they cannot be delivered.



Classic Methods Might Fail

Why Backpressure (BP) Algorithm fails?

- Node 3 uses a non-work-conserving policy such that flow conservation law is not preserved at node 3.
- However, BP is not aware of the behavior of node 3 since node 2 hides this fact from node 1.
- **Lesson learned :** A good network control algorithm must be aware of the uncontrollable policy and react accordingly.



Tracking-Max-Weight (TMW) Algorithm

- TMW enhances the original Max-Weight algorithm with an implicit learning of the policy used by uncontrollable nodes

- TMW produces control actions for controllable nodes and **generates an "emulated" action for uncontrollable nodes**

- TMW aims to
  - Stabilizing a virtual system with "emulated" uncontrollable actions
  - Minimizing the gap between the "emulated" and the true uncontrollable action

TMW - Details

- Let $\boldsymbol{f}^u(t)$ be the true action taken by uncontrollable nodes in slot t
- Let $\boldsymbol{g}(t) = (\boldsymbol{g}^c(t), \boldsymbol{g}^u(t))$ be the routing decisions generated by TMW
  - $\boldsymbol{g}^c(t)$ is the action for controllable nodes
  - $\boldsymbol{g}^u(t)$ is the "emulated" action for uncontrollable nodes

- Gap between $\boldsymbol{f}^u(t)$ and $\boldsymbol{g}^u(t)$:

$$\Delta_{ijk}(t) = g_{ijk}(t) - \tilde{f}_{ijk}(t), \qquad \forall i \in \mathcal{U}$$

where $\tilde{f}_{ijk}(t)$ is the actual number of transmitted packets under offered rate $f_{ijk}(t)$

TMW - Virtual Queues

TMW maintains two types of virtual queues

- Virtual queue $\boldsymbol{X}(t)$ is the backlog in the "emulated" system:

$$X_{ik}(t+1) = \left[X_{ik}(t) + a_{ik}(t) + \sum_{j\in\mathcal{N}} g_{jik}(t) - \sum_{j\in\mathcal{N}} g_{ijk}(t)\right]^{+}$$

- Virtual queue $\boldsymbol{Y}(t)$ characterizes the cumulative difference between the "emulated" action and the true action:

$$Y_{ijk}(t+1) = Y_{ijk}(t) + \Delta_{ijk}(t),$$

  where $\Delta_{ijk}(t) = g_{ijk}(t) - \tilde{f}_{ijk}(t), \ \forall i \in \mathcal{U}$

- TMW requires ability to observe underlay ($\tilde{f}_{ijk}(t)$)
  - Sparse and noisy observations

TMW Algorithm

1. In each slot t, obtain $\boldsymbol{g}(t)$ by solving the following problem:

$$\max_{\boldsymbol{g}(t)\in\mathcal{F}_{\omega_t}} \sum_{(i,j)} \sum_{k} g_{ijk}(t) W_{ijk}(t),$$

where

$$W_{ijk}(t) = X_{ik}(t) - X_{jk}(t) - Y_{ijk}(t).$$

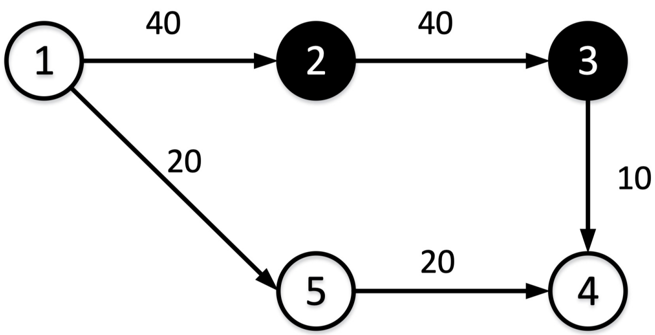
2. Apply $\boldsymbol{g}(t)$ to controllable nodes

3. Update virtual queues $\boldsymbol{X}(t)$ and $\boldsymbol{Y}(t)$

- TMW uses BP routing on the virtual queues, offset by Y
- The offset Y drives the emulated actions g^u toward the actual actions f^u, i.e., drives $\Delta \to 0$

Example Revisited: Failure of BackPressure

- Uncontrollable node 2 transmits to node 3 at full line rate
- Uncontrollable node 3 holds any packets it received
- Backlog is always zero at node 2, so BP always sends packets to node 2 although they cannot be delivered



- With TMW Y_{34} will continue to grow because node 3 does not send
- Create "backpressure" away from node 3 in "emulated" system

$$W_{ijk}(t) = X_{ik}(t) - X_{jk}(t) - Y_{ijk}(t).$$

- Eventually node 1 will stop sending to node 2 and g_{34}^{u} will go to 0

Performance of TMW

Theorem

If uncontrollable nodes use an ω-only policy, and their state can be observed, then the TMW algorithm can stabilize the physical queue $\boldsymbol{Q}(t)$ whenever possible

Proof

- Show that TMW can stabilize the two virtual queues $\boldsymbol{X}(t)$ and $\boldsymbol{Y}(t)$
- Show that if the two virtual queues $\boldsymbol{X}(t)$ and $\boldsymbol{Y}(t)$ can be stabilized, then the physical queue $\boldsymbol{Q}(t)$ can also be stabilized

Performance of TMW with Sparse Observations

- **Delayed/Sparse Observations:**
  - Denote by $\tau_i(t)$ the most recent time we have an observation of node i
  - Denote by $L_i(t) = t - \tau_i(t)$, the delay at t

Theorem: When $\sum_{t=0}^{T-1} L_i(t)/T = o(T)$ for every $i \in \mathcal{U}$, then the TMW algorithm can stabilize the physical queues $\boldsymbol{Q}(t)$ whenever possible

- As long as the average observation delay is sublinear in T, TMW is throughput-optimal for partially observable and controllable setting

Performance of TMW with Noisy Observations

- **Noisy observations**
  - Denote by $\epsilon_{ijk}(t)$ the estimation error in $W_{ijk}(t)$
  - Estimation error in observation of underlay queues

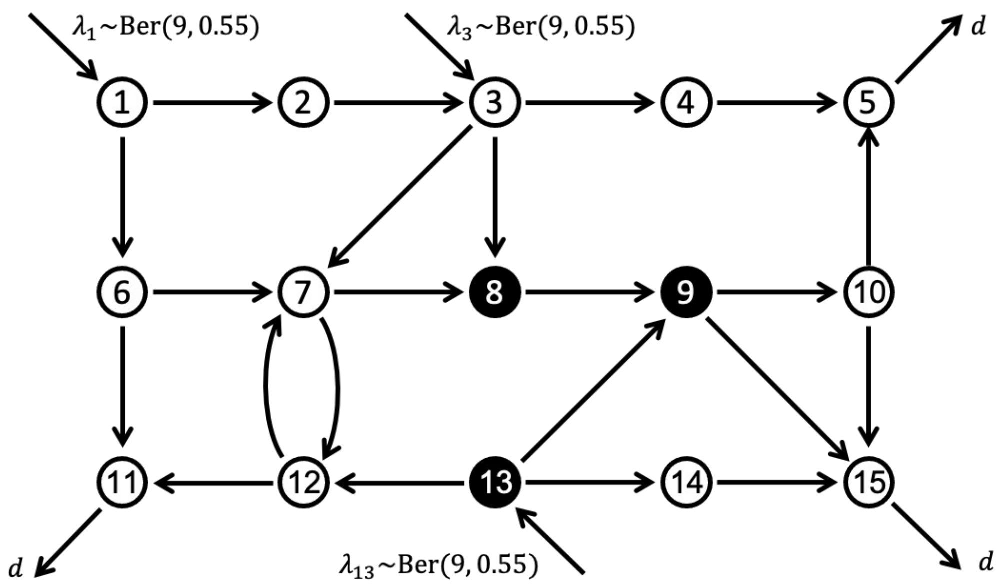
Theorem: When $|\epsilon_{ijk}(t)| = o(t)$ for every $i \in \mathcal{U}$ and k, then TMW can stabilize the physical queues $\boldsymbol{Q}(t)$ whenever possible

- If the estimation error grows sublinearly in t, TMW is throughput-optimal
  - Includes case of constant noise (O(1))

Simulation

Model

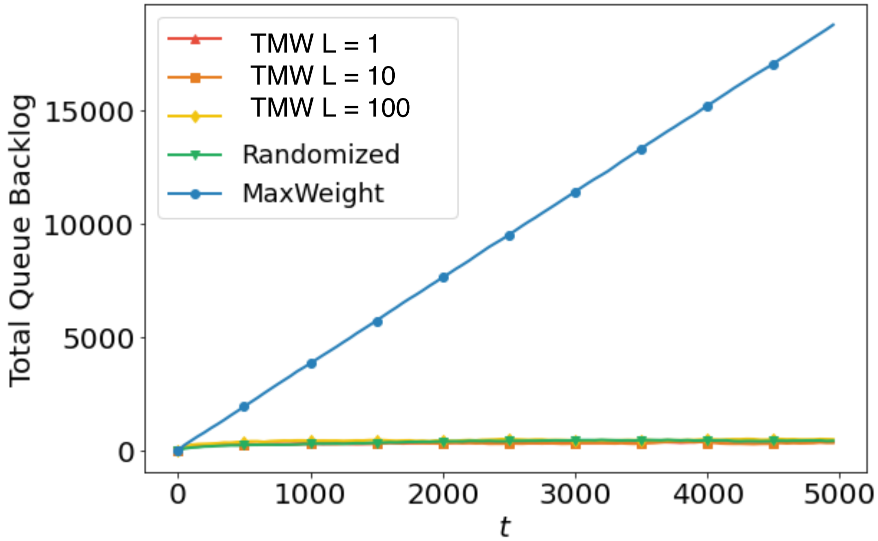
- All links have the capacity of 5
- Node 8, 9 and 13 are uncontrollable and unobservable
  - Uniformly route 0~5 packets on each outgoing link

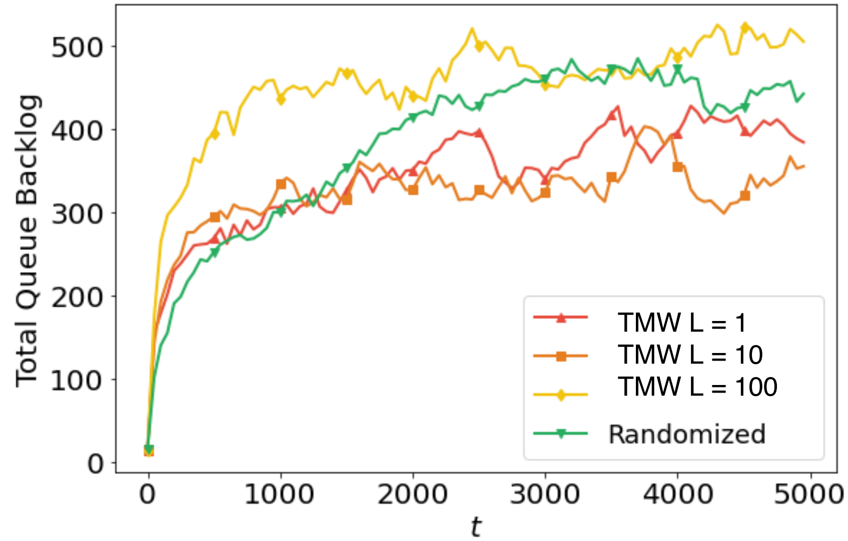


Simulation: stability

Stability performance (sparse observation)

- Suppose the observations are sparse (nodes in $\mathcal{U}$ can only be observed every L time units).
- Max-Weight (i.e., BackPressure) fails to stabilize the system.

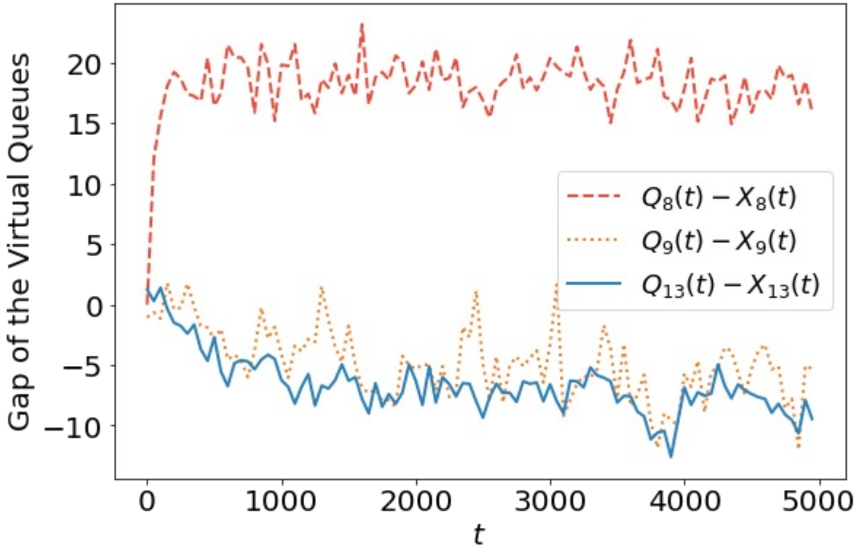




Simulation: tracking of underlay

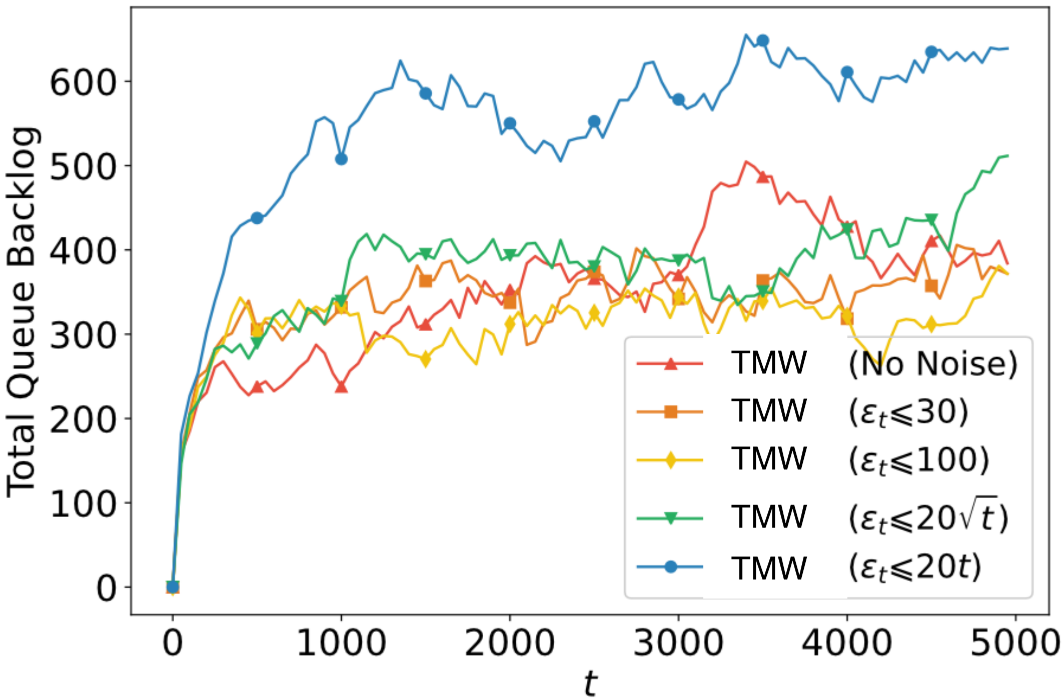
Tracking of underlay queue backlogs (sparse observation with $L = 10$)

- TMW quickly controls the gap between X_{ik} and Q_{ik}.



Simulation: noisy observations

Stability performance with noisy observations



Learning-based network control (talk outline)

- Tracking Max-Weight (TMW): Learning-aided Max-Weight algorithm
  - Need to learn unknown underlay dynamics
  - Focus on network stability
- **Gradient sampling Max-Weight: Learning-based network utility maximization**
  - Unknown utility functions
  - Application to minimum delay routing
- Reinforcement learning algorithm for queueing networks
  - General optimal control for queueing systems

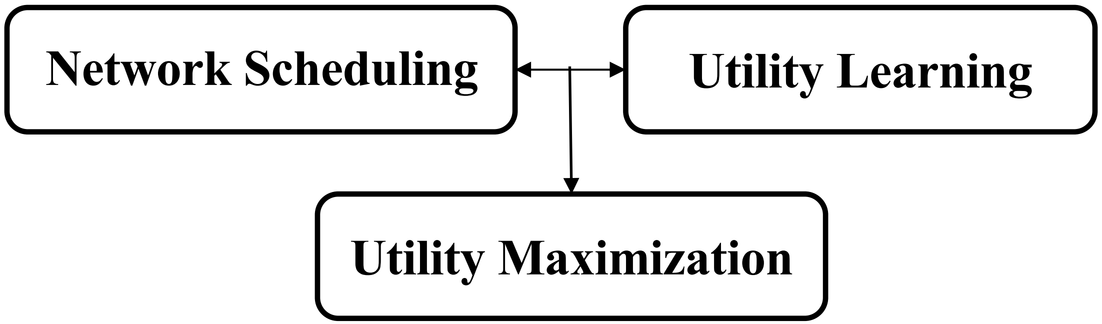
[3] Xinzhe Fu, E. Modiano, "Learning-NUM: Network Utility Maximization with Unknown Utility Functions and Queueing Delay," IEEE/ACM Transactions on Networking," 2022.
[4] Xinzhe Fu, E. Modiano, "A Learning Approach to Minimum Delay Routing in Stochastic Queueing Networks," Infocom 2023.

Network Utility Maximization (NUM) with Unknown Utilities

- NUM objective: maximize sum utilities subject to capacity constraints
  - $g_i(r)$ the "utility" of allocating rate r to class i traffic
- Previous works consider known utility functions
  - E.g., proportional fairness: $g(r) = \log(r)$
- The utility functions may be unknown in advance
  - User satisfaction (e.g., video quality)
  - Average delay

$$\textbf{maximize}: \sum_i g_i(r_i)$$
$$\textbf{Subject to}: \underline{r} \in \Lambda, \ \underline{r} \le \underline{\lambda}$$

- Key challenges/novelty:
  - Unknown utility functions: Power consumption of links, delay, user satisfaction
  - Feedback delay: Function values are observed after decisions are made



Gradient Sampling Max-Weight (GSMW)

$$\text{maximize}: \sum_i g_i(r_i)$$
$$\text{Subject to}: \underline{r} \in \Lambda, \ \underline{r} \le \underline{\lambda}$$

- Goal: Minimizing regret over time-horizon T
- The Gradient Sampling Max-Weight Algorithm
  - Choose user rates: r_i's
  - Use feedback to construct approximate gradients

$$\text{gradient} = \frac{[g_i(r_i+\delta) - g_i(r_i-\delta)]}{2\delta}$$

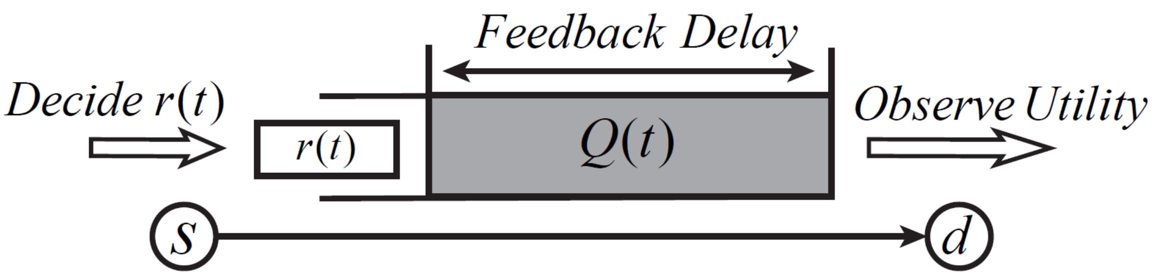
  - Use Max-Weight to determine the network routing and scheduling decisions:
    Ensure network stability: constraint $r \in \Lambda$
  - Update the rate variables based on approximate gradients and queue lengths

$$r_i(t+1) := r_i(t) + \frac{1}{\alpha} \cdot (gradient - V \cdot queue\ length)$$

- Primal-Dual Interpretation:
  - Primal variables: rates r_i's
  - Dual variables: queue lengths Q_n's, corresponding to constraint $r \in \Lambda$
  - Update primal and dual variables based on gradient of the Lagrangian
    Primal: gradient – V * queue lengths
    Dual: queue length dynamic

Gradient Sampling Max-Weight (GSMW)

- Dealing with feedback delay



- The Parallel-Instance GSMW
  - If feedback delay is Z slots, initiate Z instances of GSMW
  - Since delay is unknown and time-varying, can generate “new” instances dynamically while waiting for feedback

Gradient Sampling Max-Weight (GSMW) Performance Guarantees

- Regret: $R(T)$
  - The cumulative difference between the utility achieved by the algorithm and the optimal over a time horizon of T time slot

- When the feedback is noiseless: GSMW achieves $\mathrm{R}(T) = \mathrm{O}(\sqrt{T})$

- When the feedback is noisy: GSMW achieves $\mathrm{R} = \mathrm{O}\left(\max\left\{\sqrt{T}, T^{(2+2\beta)/3}\right\}\right)$
  - β = noise parameter

    Each observation is corrupted by an i.i.d. zero-mean random noise with standard deviation bounded by T^{β} $(\beta \leq 0)$
  - Regret increases from $\mathrm{O}(\sqrt{T})$ to $\mathrm{O}(T^{\frac{2}{3}})$ as noise increases

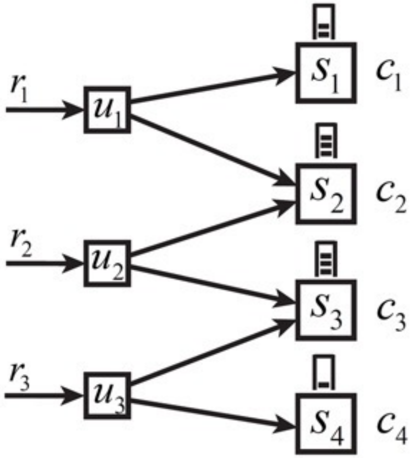
- Sublinear regret corresponds to “optimality” as the regret per unit time goes to zero

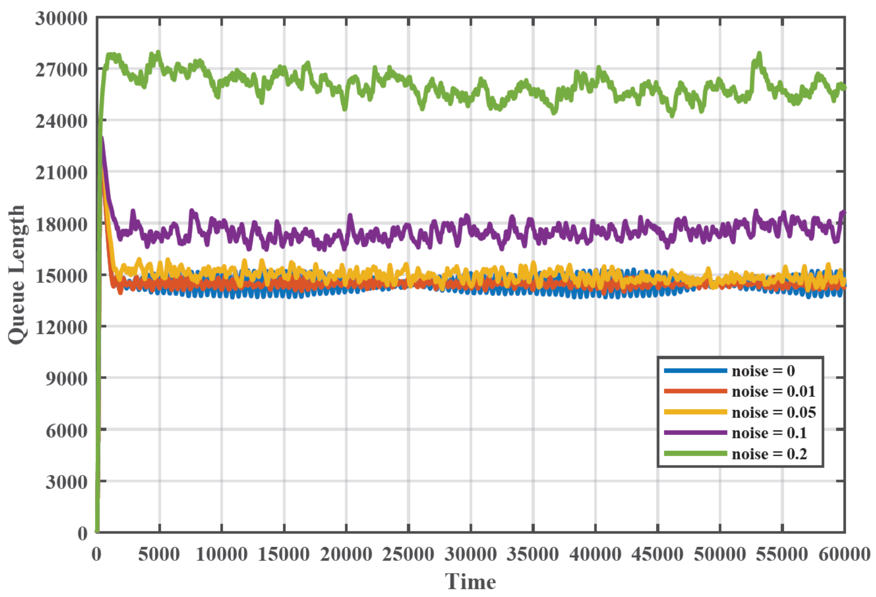
- Noise can represent imprecise measurement or feedback errors

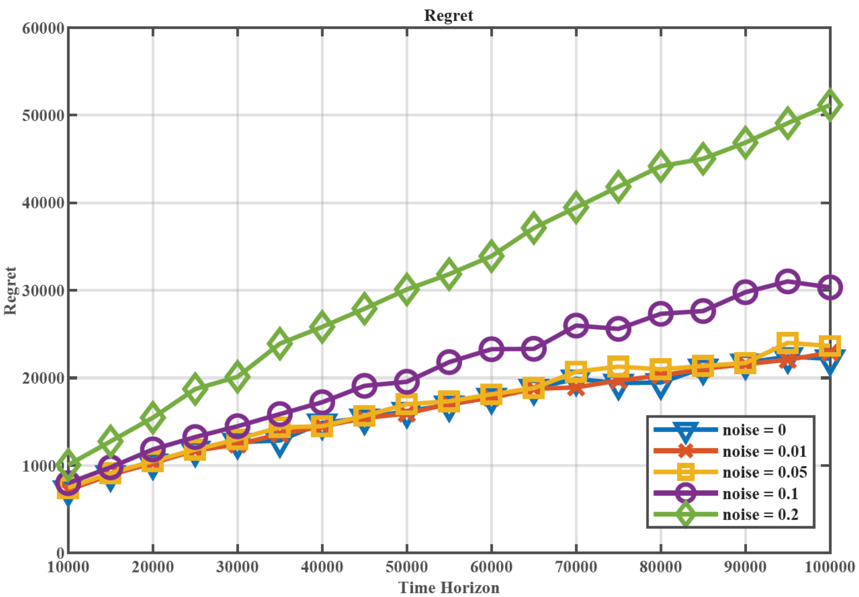
Simulation Results

- Parallel server network
- Utility is function of server and data rate
  - Mix of logarithmic, polynomial and linear functions

- GSMW stabilizes the network and achieves sublinear regret
  - Sublinear regret = asymptotic optimality







Queue-length **Regret**

- I.I.d noise Uniform [-noise, noise]

GSMW: Minimum Delay Routing

- Minimum Delay Routing [1]
  - K paths $\{P_1, \dots, P_K\}$ from source s to destination d
  - Route incoming traffic of rate λ along the paths
  - Capacity region Λ
  - Compute the optimal rates $r = (r_1, \dots, r_K)$
  - Flow on link e associated with rate vector r: $f_e^r = \sum_{k: e \in P_k} r_k$

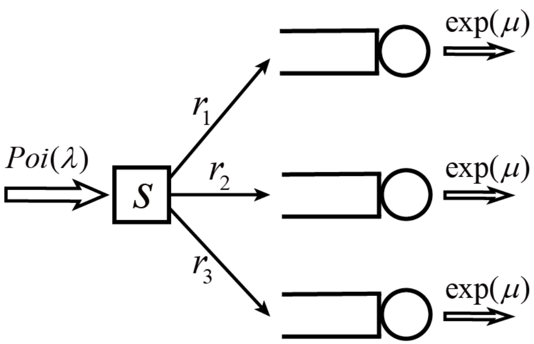
- Assumptions:
  - $D_e(f_e)$ is the delay of link e when the rate is f_e
  - D_e is convex, non-decreasing and known

$$\begin{aligned} \textbf{Minimize } & \sum_{e \in E} D(f_e^r) \\ \textbf{s. t. } & \sum_{k=1}^{K} r_k = \lambda, \\ & r \in \Lambda, \\ & r_k \geq 0, \forall k. \end{aligned}$$

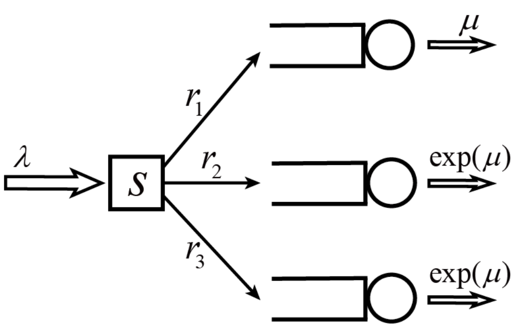
[1] R. Gallager, “A minimum delay routing algorithm using distributed computation." 1977.

Minimum Delay Routing in Stochastic Queueing Networks

- Parallel M/M/1 Queues
  - $\rho_i = \mathrm{r_i}/\mu$
  - $\mathbb{E}_{\pi_r}[Q_i(t)] = \rho_i/(1-\rho_i)$
  - Optimal: $r_1 = r_2 = r_3 = \frac{\lambda}{3}$.



- M/M/1 queues and deterministic queue
  - Route more traffic to the deterministic queue.



The delay function depends on link characteristics that are unknown apriori.

Minimum Delay Routing in Stochastic Queueing Networks

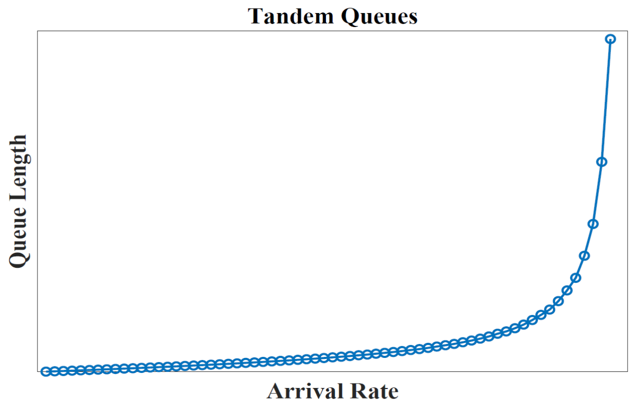
- Network Model
  - Arrival rate $a(t)$, i.i.d., with $\mathbb{E}[\,a(t)] = \lambda$
  - Route the incoming packets along K paths $\{P_1, \dots, P_k\}$
  - Static routing policy parameterized by routing vector $r = (r_1, \dots, r_K)$
  - Queue length of link e at time t: $Q_e(t)$
  - Steady-state queue length distribution under rate r: π_r
  - By Little's law, λ*(steady-state delay) = $\sum_{e\in E} \mathbb{E}_{\pi_r}[Q_e] \coloneqq D(r)$

- Problem Formulation
  - Find r that minimizes $D(r)$
  - $D(r)$ is unknown, but queue lengths are observable
  - *Learn* the delay function and the optimal static routing policy

$$
\begin{aligned}
\text{Minimize } & \mathrm{D(r)} \coloneqq \textstyle\sum_{e\in E} \mathbb{E}_{\pi_r}[Q_e] \\
\text{s. t. } & \textstyle\sum_{k=1}^{K} r_k = \lambda, \\
& r \in \Lambda, \\
& r_k \ge 0, \forall k.
\end{aligned}
$$

Minimum Delay Routing in Stochastic Queueing Networks

- Assumptions: $D(r)$ is a convex function of r
  - Proved for single queues [1]
  - We show the convexity for tandem queues via stochastic coupling
  - It follows that convexity holds for networks with disjoint paths
- Static Routing vs. Dynamic Routing
  - We study the optimal static routing policy that makes decisions independent of queue lengths
  - Dynamic policies can outperform the optimal static policy, but few results are known
  - In simulations, the optimal static policy outperforms common dynamic policies



Minimize $\mathrm{D(r)} := \sum_{e \in E} \mathbb{E}_{\pi_r}[Q_e]$

s. t. $\sum_{k=1}^{K} r_k = \lambda$,

$r \in \Lambda$,

$r_k \geq 0, \forall k.$

[1] M. Neely and E. Modiano, "Convexity in queues with general inputs." 2005.

The Gradient Sampling Framework

- Projected Gradient Descent:
  - $r_{t+1} := r_t - \eta \cdot \nabla D(r_t)$
  - Projected r_t onto the feasibility region

- Gradient Sampling:
  - Approximate $\nabla D(r_t)$ using values of D
  - Randomly sample a perturbation vector ϵ of unit length
  - Approximate $\nabla \mathrm{D}(\mathrm{r}_t)$ by $\hat{\nabla} D(r_t) := \frac{D(r_t+\delta\epsilon)-D(r_t-\delta\epsilon)}{2\delta} \cdot \epsilon$
  - $r_{t+1} := r_t - \eta \cdot \hat{\nabla} D(r_t)$

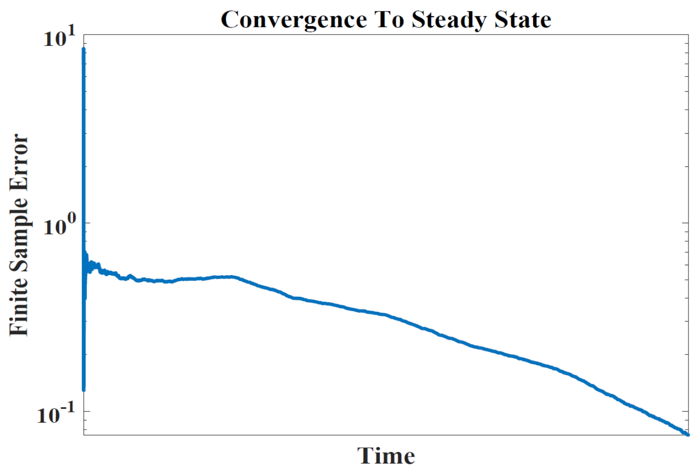
- Challenges:
  - How to obtain the value of $D(r)$?
  - Performance guarantee of the whole procedure

[1] X. Fu and E. Modiano, "Learning-NUM: Network utility maximization with unknown utility functions and queueing delay." 2022.
[2] A. Flaxman, A. Kalai, and H. McMahan, "Online convex optimization in the bandit setting: gradient descent without a gradient." 2004.

Estimating the Steady-State Delay

- Using queue-length observations to estimate steady-state delay
  - Starting from t_0, employs routing vector r for duration τ
    $$\lim_{\tau \to \infty} \mathbb{E}[Q_e(t_0 + \tau)] = \mathbb{E}_{\pi_r}[Q_e]$$
  - Use queue-length observation at $t_0 + \tau$ (for a large enough τ) to approximate $\mathbb{E}_{\pi_r}[Q_e]$

- Proposition:
  - The error $\mathbb{E}\left|Q_e(t_0 + \tau) - \mathbb{E}_{\pi_r}[Q_e(t)]\right|$ decreases exponentially with τ
  - Analyze the convergence of countable-state Markov chain using Lyapunov drift arguments



Convergence of the average total queue lengths in a tandem network

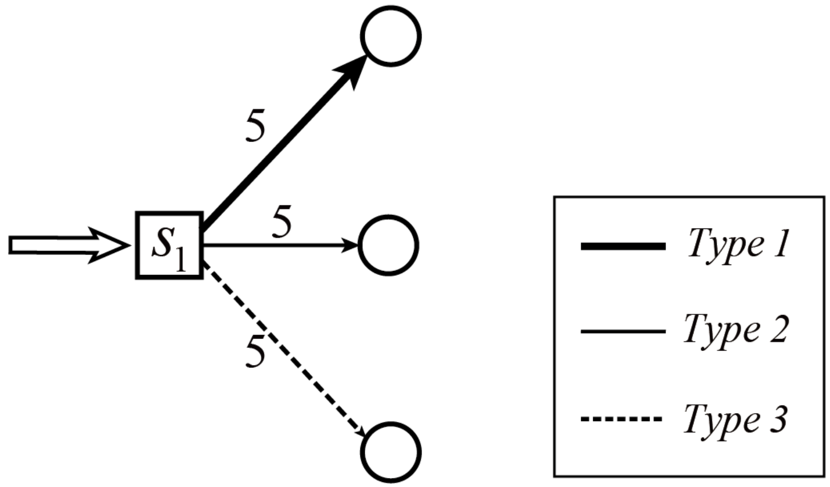
Gradient Sampling For Delay Minimization

- The Gradient Sampling Policy
  - For each iteration $t = 1, \dots T$:
  - Randomly sample a perturbation vector ϵ
  - Employ the static routing vector $r_t - \delta\epsilon$ for $\tau = \log T$ time slots $t_0 + 1, \dots, t_0 + \tau$
    $\widehat{D}(r_t - \delta\epsilon)$ as the total queue lengths at $t_0 + \tau$.
  - Employ the static routing vector $r_t + \delta\epsilon$ for $\tau = \log T$ time slots $t_0 + \tau + 1, \dots, t_0 + 2\tau$
    $\widehat{D}(r_t - \delta\epsilon)$ as the total queue lengths at $t_0 + 2\tau$.
  - Approximate $\nabla D(r_t)$ by $\widehat{\nabla} D(r_t) \coloneqq \frac{\widehat{D}(r_t+\delta\epsilon)-\widehat{D}(r_t-\delta\epsilon)}{2\delta} \cdot \epsilon$
  - $r_{t+1} \coloneqq r_t - \eta \cdot \widehat{\nabla} D(r_t)$. (Projected onto the feasibility region)

- Theorem: Let r^* be the optimal routing vector. $D(r_T) - D(r^*) = O\left(\frac{log\ T}{T}\right)^{\frac{1}{4}}$
  - Suitable values for δ, η
  - Proof: Bias and variance of the approximate gradients plugging in the dynamics of the gradient descent workflow

Single-hop Network

- Link Type:
  - Type 1: deterministic
  - Type 2: uniform
  - Type 3: bursty
- Load Level:
  - Low: arrival = 4
  - Medium: arrival = 8
  - High: arrival = 12
- Policy:
  - Uniform
  - JSQ
  - Gradient Sampling (GS)



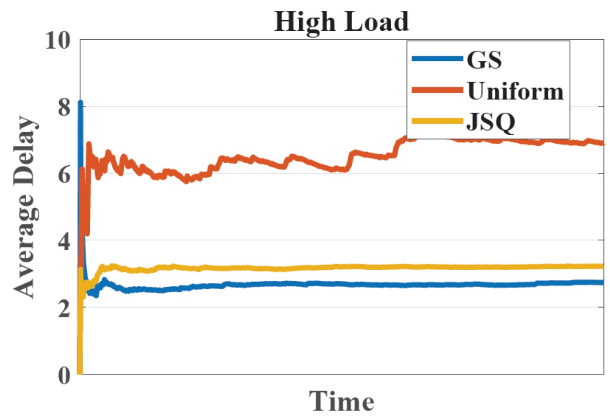
Minimize $\sum \mathbb{E}_{\pi_r}[Q_i]$

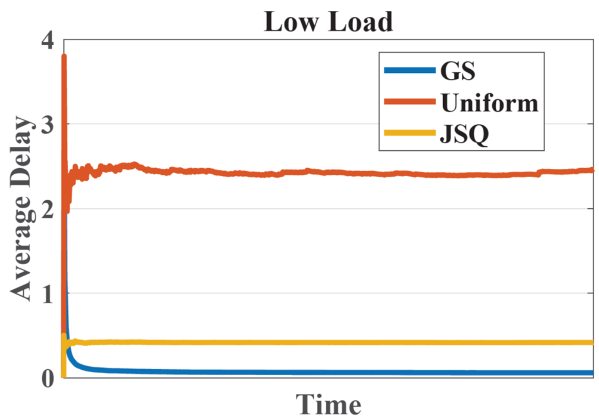
s. t. $r_i \leq 5, \quad i = 1,2,3$

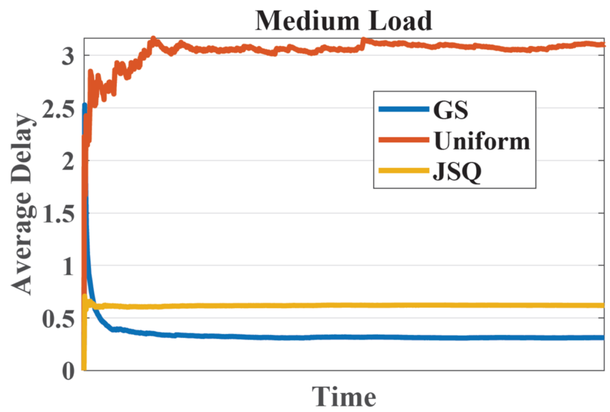
$\sum_{i=1}^{3} r_i = \lambda$

Single-hop Network

- GS can "learn" the link type
  - The bursty link should be avoided if possible
  - GS converges to the optimal static policy, which outperforms JSQ
- The gap decreases with the load







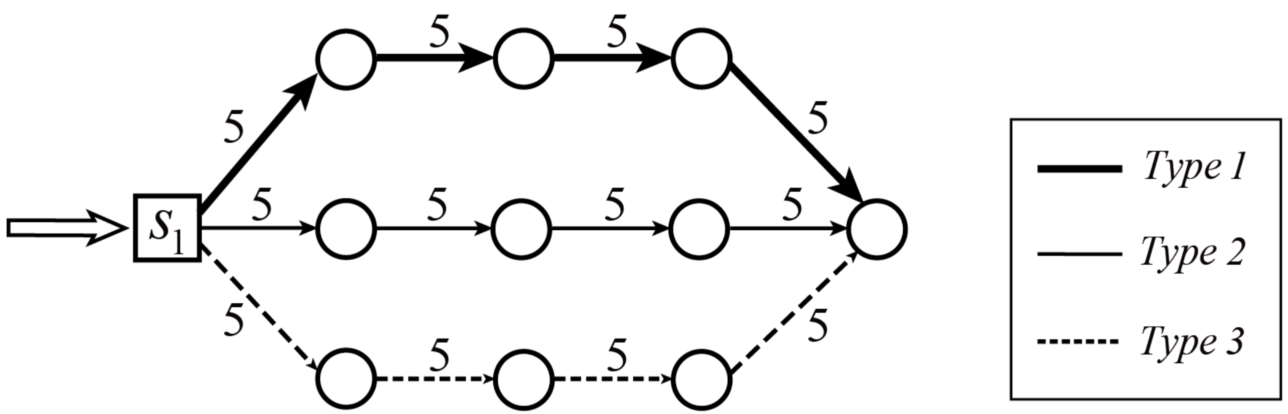
Disjoint Paths

- Link Type:
  - Type 1: deterministic
  - Type 2: uniform
  - Type 3: bursty
- Policy:
  - Uniform
  - UMW
  - GS
- Load Level:
  - Low: arrival = 4
  - Medium: arrival = 8
  - High: arrival = 12

Minimize $\sum_{e=1}^{12} \mathbb{E}_{\pi_r}[Q_e]$

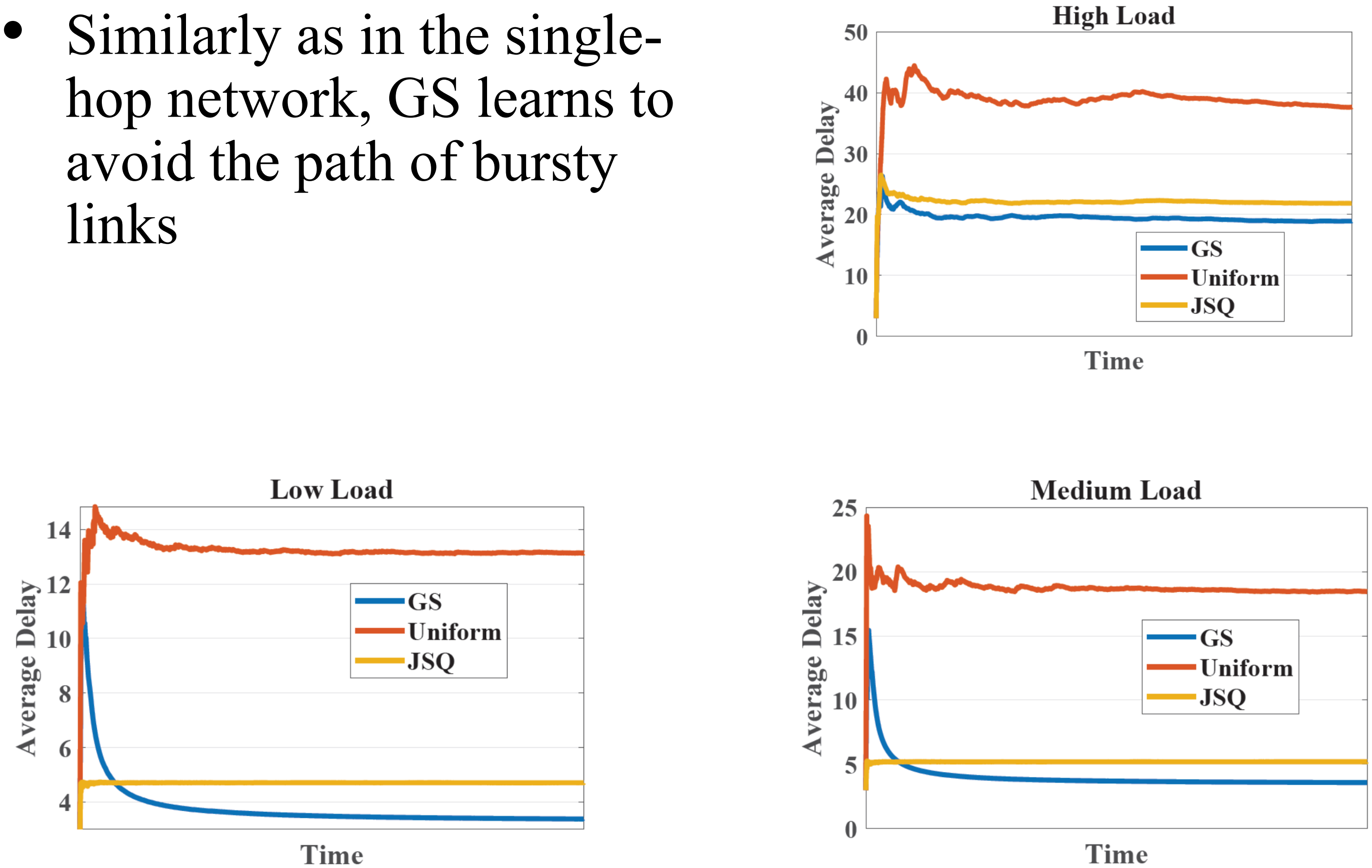
s. t. $r_i \leq 5, \quad i = 1,2,3$

$\sum_{i=1}^{3} r_i = \lambda$



Disjoint Paths

- Similarly as in the single-hop network, GS learns to avoid the path of bursty links



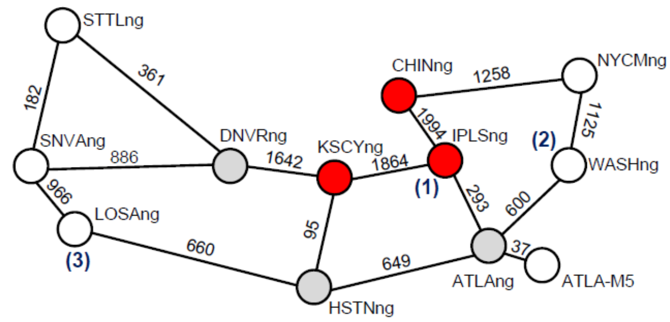
The Abilene Network

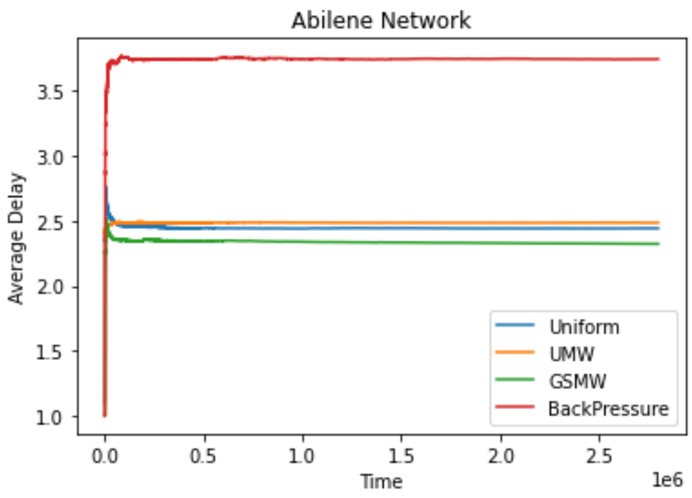
- The Abilene Network
  - Link rates scaled down by 10. Offered transmissions are generated from Poisson distributions.
  - Two sources and one destination.

    S_1: STTLng, arrival = 30

    S_2: CHINng, arrival = 40

    D: ATLAng
- Policy:
  - Uniform
  - UMW
  - BackPressure
  - GSMW [1]





[1] A. Sinha and E. Modiano, "Optimal control for generalized network-flow problems." 2017.

Extension To Wireless Networks

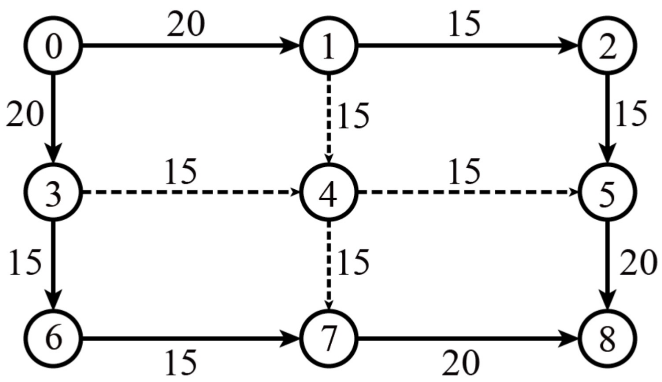
- The problem formulation and the gradient sampling policy can be extended to wireless networks

$$\begin{aligned} \text{Minimize } \; & \mathrm{D(r)} := \sum_{e\in E} \mathbb{E}_{\pi_r}[Q_e] \\ \text{s. t. } & \sum_{k=1}^{K} r_k = \lambda\,, \\ & r \in \Lambda\,, \\ & r_k \geq 0, \forall k. \end{aligned}$$

- Compute the optimal routing policy for the network with a given scheduling policy
  - The queues still evolve following some underlying Markov chain
  - The gradient sampling policy has the same guarantee if the delay function is convex

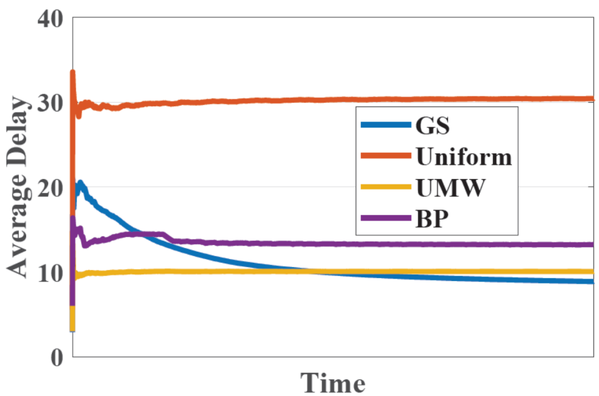
Grid Topology: Wireless

- 3*3 Grid Network
  - Type 2 (Poisson) and Type 3 (bursty) links
  - Source: 0, Destination: 8
    6 paths
  - Arrival rate: 8



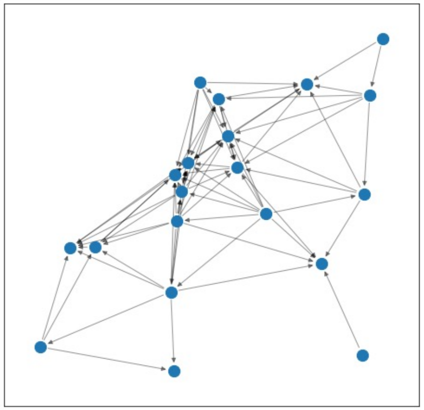
- Scheduling Policy: Max-Weight

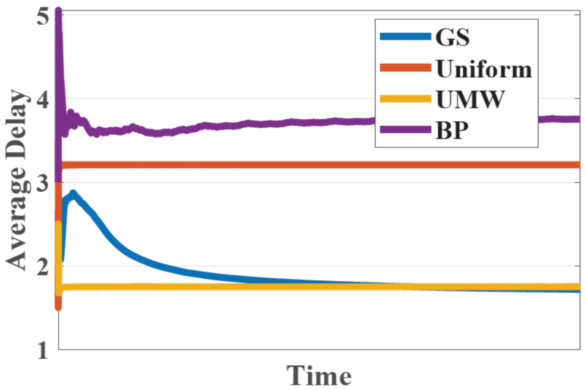
- Routing Policy:
  - Uniform (source routing)
  - UMW
  - BackPressure (BP)
  - GS



Random Geometric Graph: Wireless

- Random Geometric Graph
  - 20 nodes in a unit square
  - Distance threshold 0.4
  - Poisson links of rate 20
  - Two source-destination pairs with arrival rates 4
- Scheduling Policy:
  - Max-Weight
- Policy:
  - Uniform (source routing)
  - UMW
  - BackPressure
  - GS
  - AugGS





Learning-based network control (talk outline)

- Tracking Max-Weight (TMW): Learning-aided Max-Weight algorithm
  - Need to learn unknown underlay dynamics
  - Focus on network stability

- Gradient sampling Max-Weight: Learning-based network utility maximization
  - Need to learn unknown utility functions
  - Feedback/actions subject to queueing delay

- **Reinforcement learning algorithm for queueing networks [5]**
  - **General optimal control for queueing systems**

[5] Bai Liu, Qiaomin Xie, E. Modiano, "RL-QN: A Reinforcement Learning Framework for Optimal Control of Queueing Systems," ACM Trans on Modeling and Performance Eval of Computing Systems (TOMPECS), 2022.

Network performance optimization

- Most previous work focused on long-term throughput, utility
  - Infinite time horizon, coarse performance metric
- Optimizing finer granularity metrics (e.g., queue-size) is challenging due to curse of dimensionality
  - Limited results for idealized settings
- Reinforcement learning has the potential to solve this problem
  - Neural Nets: promising but little insight
  - Model-based RL (e.g., Upper confidence RL) holds promise for low-complexity insightful solutions
- Approach: Use RL to optimize performance in networks with unknown dynamics
  - Challenge: dealing with unbounded state-space due to queue-size
  - Control actions affect the dynamics of uncontrollable nodes (through the queues)

Queue-Dependent Uncontrollable Policy

$$\pi_u : (\omega, \boldsymbol{Q}) \mapsto \boldsymbol{f}^u$$

- Policy takes queue size into account
- Covers state-of-the-art dynamic routing and scheduling algorithms (e.g., BackPressure routing)
- Queue evolution dynamics may be unknown and arbitrary

$$\boldsymbol{Q}(t+1) \;=\; h(\boldsymbol{f}^c(t), \boldsymbol{Q}(t), \omega_t),$$

  where $h(\cdot)$ is some unknown function that depends on our controllable routing action $\boldsymbol{f}^c(t)$, the current queue length vector $\boldsymbol{Q}(t)$, and the observed network event ω_t

- Optimization is a Markov Decision Problem (MDP)

MDP Formulation

- **Action**: $\boldsymbol{f}^c(t)$
- **State**: $\boldsymbol{Q}(t)$
- **State Transition Probabilities**:

$$P(\boldsymbol{Q}'|\boldsymbol{Q}, \boldsymbol{f}^c(t)),$$

  evolve according to the queueing dynamics $\boldsymbol{Q}(t+1) \; = \; h(\boldsymbol{Q}(t))$.

- **Objective**: find a policy π^* that minimizes the long-term average queue length

$$J^\pi \; = \lim_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E}\left[\sum_{ik} Q_{ik}^\pi(t)\right].$$

Challenges in Solving the MDP

- This is an MDP with unknown dynamics

$$\boldsymbol{Q}(t+1) = h\big(\boldsymbol{Q}(t)\big),$$

  i.e., Reinforcement Learning (RL) problem

- The state space (i.e., queue length vector space) $\mathcal{Q}$ is countably-infinite
- Existing RL algorithms do not have any performance guarantees in face of countably-infinite state space

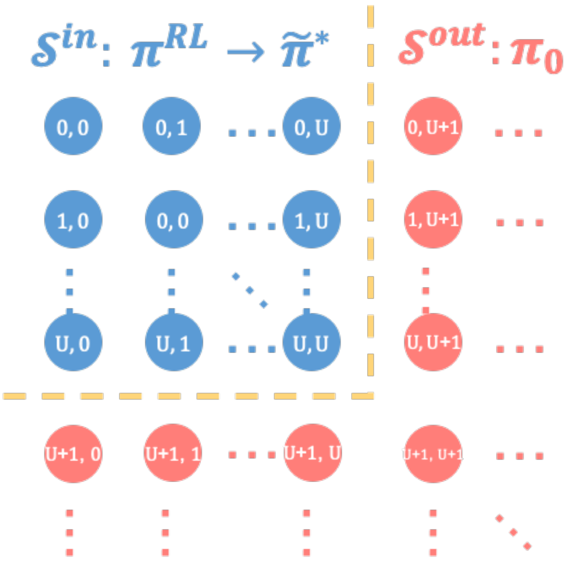
- Possible approaches:
  - Truncation: Solve MDP for truncated system [Liang, Modiano, Infocom '18]
  - RL-QN [Liu, Xie, Modiano, Allerton 19]

    Optimal performance for queueing networks with unbounded state space

Reinforcement Learning for Queueing Networks (RL-QN) Algorithm

For a Bounded State Space $\mathcal{S}^{in}$

- Apply model-based reinforcement learning scheme
  - E.g., Upper-confidence RL (UCRL); episodic exploration/exploitation scheme
- Converges to the optimal policy $\tilde{\pi}^*$



Average cost goes to optimal as $\mathcal{S}^{in}$ grows

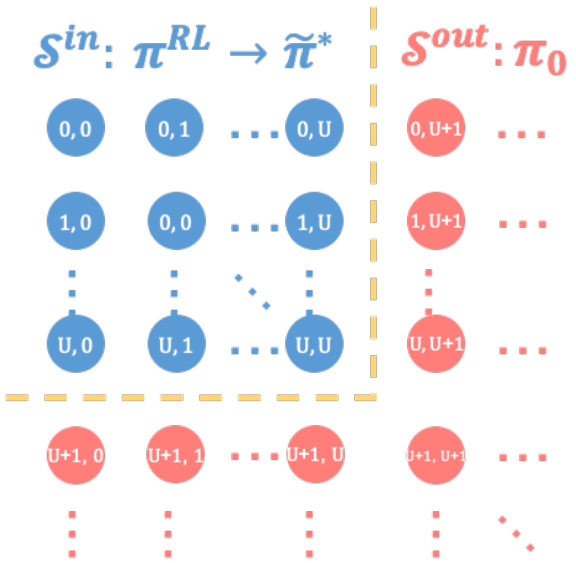
For the Rest of the State Space $\mathcal{S}^{out}$

- Use a known stabilizing policy π_0 (common in communication network)
- Apply π_0 to the rest of the states
- Intuition: in stable system the probability of the queue exceeding U decays exponentially in U

RL-QN Algorithm (exploration vs. exploitation)

For episodes $k = 1, 2, \cdots$

- *w.p.* $l/\sqrt{k}$, do **exploration**
  - Apply π_{rand} to $\mathcal{S}^{in}$
  - Apply π_0 to $\mathcal{S}^{out}$
- *w.p.* $1 - l/\sqrt{k}$, do **exploitation**
  - Use history data to **estimate** the dynamics of $\widetilde{M}$
  - **Solve** for estimated optimal policy $\tilde{\pi}_k$
  - Apply $\tilde{\pi}_k$ to $\mathcal{S}^{in}$
  - Apply π_0 to $\mathcal{S}^{out}$
- When visits to $\mathcal{S}^{in}$ exceeds $L\sqrt{k}$, start the next episode



Performance of RL-QN

Theorem 1

For any $0 < \delta < 1$, there exists $k^* < \infty$ such that our algorithm learns $\tilde{\pi}^*$ (i.e. $\tilde{\pi}_k = \tilde{\pi}^*$) within k^* episodes with probability at least $1 - \delta$

The optimal policy for the bounded system

Theorem 2

Under our algorithm, the asymptotic episodic average cost is upper bounded as

$\alpha, \gamma > 0$

$$\lim_{T \to \infty} \frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E}\left[\sum_{ik} Q_{ik}(t)\right] = \rho^* + \mathcal{O}\left(\frac{U^{1+\max\{2\alpha,\gamma\}}}{\exp(U)}\right)$$

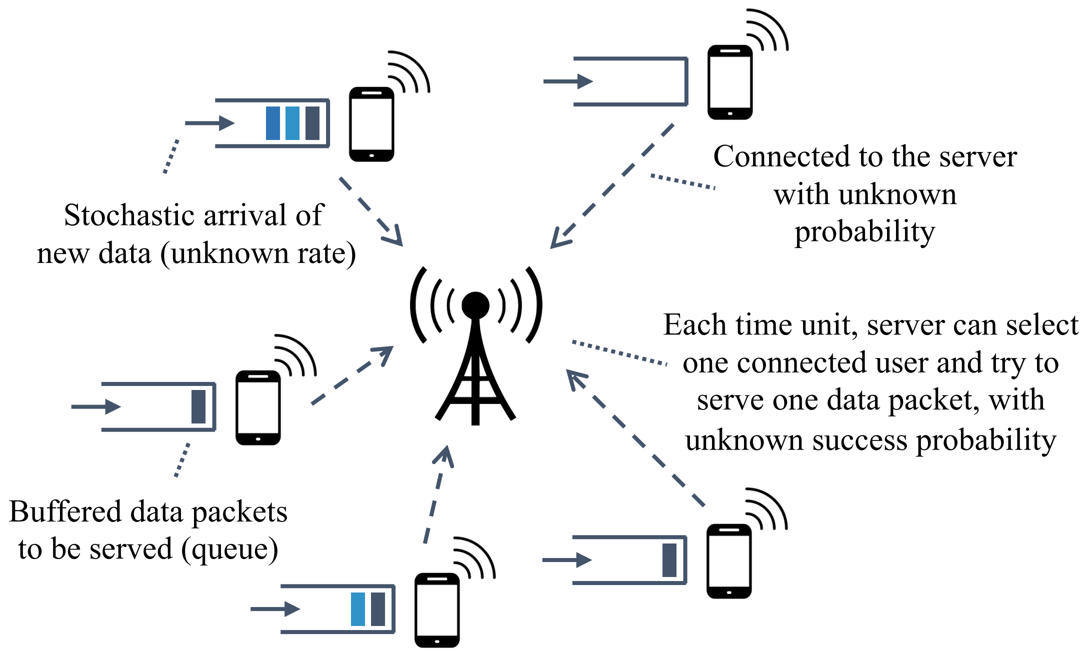
Optimal result

Buffer size of the bounded system $\widetilde{M}$

- We could get arbitrarily close to optimum by increasing U
- But larger U brings heavier computational burden
- Key intuition: in stable system the probability of the queue exceeding U decays exponentially fast in U

Simulation

Model



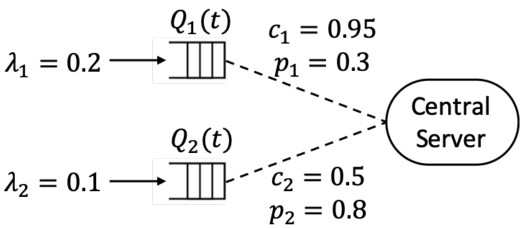
Which user to serve to minimize the average total queue length?

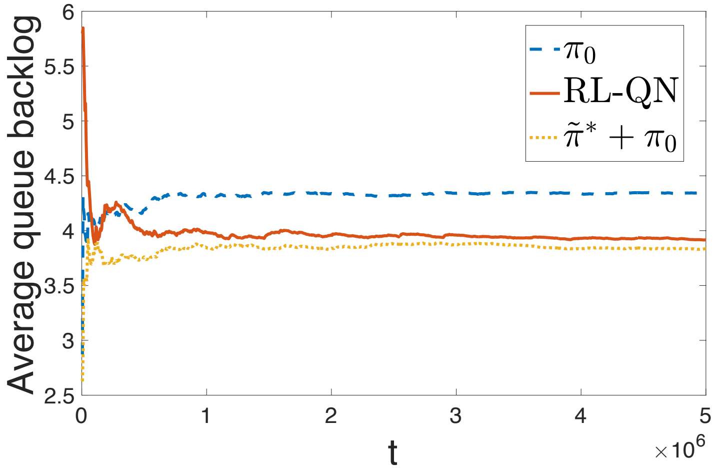
- **Stabilizing policy**: serve the longest connected queue (LCQ) , can bound the queue length [Tassiulas et al., 1993]
- **Minimizing policy**: open problem (except for symmetric cases)
- Reinforcement learning methods might work!

Simulation

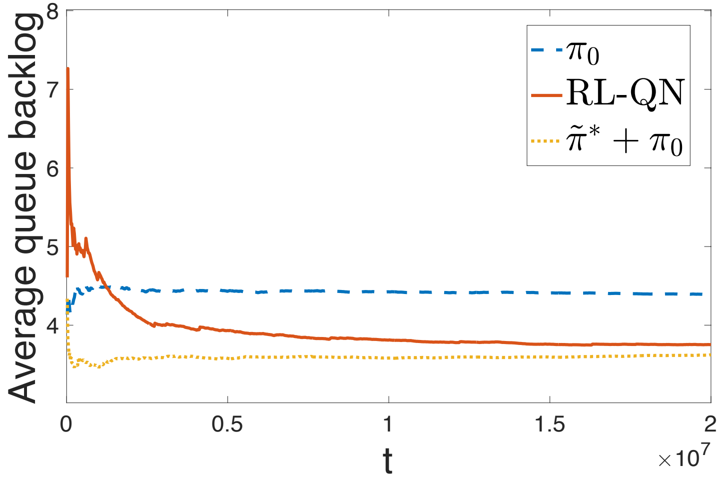
Average queue backlog evolution

- π_0: serve the longest connected queue (LCQ)
- $\tilde{\pi}^* + \pi_0$: the result our algorithm converges to





Results when $U = 5$



Results when $U = 10$

Summary

- Network control schemes typically assume known and controllable dynamics
- Unknown and/or uncontrollable dynamics give rise to the need for ”learning”
- “Learning” for achieving stability is relatively easy and can be accomplished via the queue dynamics
  - Even traditional backpressure learns the optimal policy via the queue dynamics (primal dual interpretation of optimization problem)
- Learning for optimizing network performance is more challenging because control action affect network state
  - Gradient sampling MaxWeight approach for network utility maximization
  - **RL-QN**: optimizing performance for queueing networks with unbounded state-space