Reinforcement learning algorithms for non-stationary environments

Devika Subramanian
Rice University

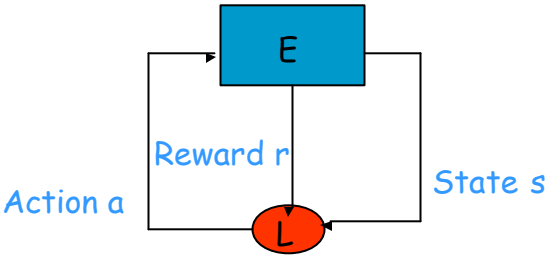
Joint work with Peter Druschel and Johnny Chen of Rice University. Supported by a grant from Southwestern Bell.

Outline

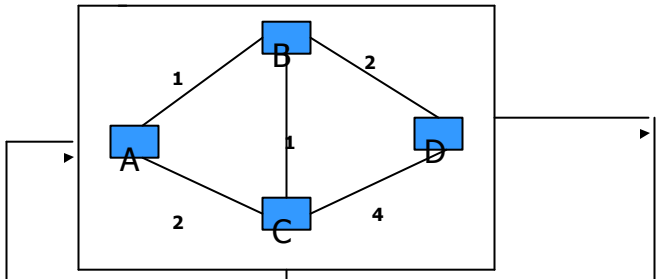
- A short introduction to reinforcement learning
- Modeling routing as a distributed reinforcement learning problem
- The evolution of algorithms for routing in dynamic networks
- Conclusion

Reinforcement learning

- learning optimal policies for sequential decision making tasks by repeated interaction with the environment.

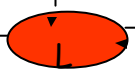


Example: finding shortest paths



Reward: 1 hop cost

Packet location



Action: send packet to node

Routing action choices

e.g., send packet to B

e.g., packet at node A, route to B or C?

Example: continued

- Goal: find shortest paths to node D.
- Interaction with environment:
  - World: packet at node x, choices for routing are nodes y,z...
  - Learner: send packet to y.
  - World: that cost you distance(x,y).
  - World: packet at node y, choices for routing are nodes...
  - Learner: send packet to ...

Example continued

- Reinforcement learner samples paths in the network
  - (A,B,1),(B,D,1)
  - (A,C,2),(C,D,4)
  - (A,C,2),(C,A,2),(A,B,1),(B,D,1)
  - .....
- Reinforcement learner acquires a policy that maps each node to the neighbor it must route a packet to, in order to minimize distance to node D.

Reinforcement learning: basics

- Associate with each node s, a value function V_D which is the cost of the shortest path from s to goal node D.
- Update V_D online as follows, upon observing (s,s',r)

Learning rate

$$V_D(s) = (1-a)V_D(s) + a(r + V_D(s'))$$

Current estimate of distance from s to D.

New estimate of distance from s to D

Why does this equation work?

- It is an online approximation of Bellman's dynamic programming

$$V_D(s) = \min_{s' \in Nbs(s)} [r(s,s') + V_D(s')]$$

- Bellman's equation updates V(s) according to best neighbor of s, while reinforcement learning updates V(s) according to observed neighbor of s in the sampled trajectory. In the limit, they both converge to the same V.

The reinforcement learning algorithm

- Initialize V(s) = 0, for all s in S.
- Repeat until V converges
  - $s = s_0$
  - while (s not a terminal state) do
    - Pick best action a in s, consistent with current V estimates. (can be stochastic)
    - Do a, and transition to state s' with reward r.
    - Update V(s) using (s,s',r)
  - end while

Policies and value functions

- Once the algorithm converges on the optimal value function V_D^*, each node can calculate its best policy as follows:

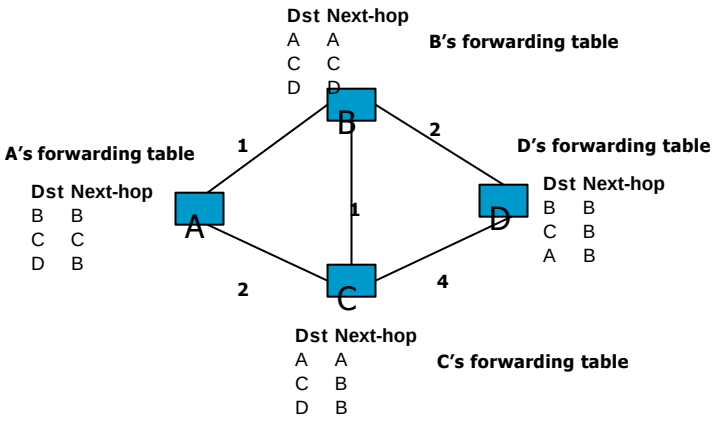
$$p_D^*(s) = \arg\min_{s' \in Nbs(s)} [r(s,s') + V_D^*(s')]$$

  - this is the best neighbor of s to forward a packet headed for node D.

Condition for convergence of RL

- RL learns the value function V by repeated biased random sampling of trajectories.
- Requires that the environment be stationary for convergence:
  - Network topology does not change with time.
  - Network edge costs do not change with time.

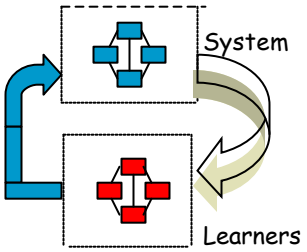
The Internet routing problem



Challenge

- System state is distributed, obtaining global state requires communication.
- System is non-stationary:
  - topology changes with time (link/router failure or recovery),
  - link costs change with time (congestion).

Modeling routing as a distributed reinforcement learning problem



A reinforcement learner at each node of the network

Distributing the value function

- For each destination d, a node maintains V_d : the cost of the cheapest path to that d.

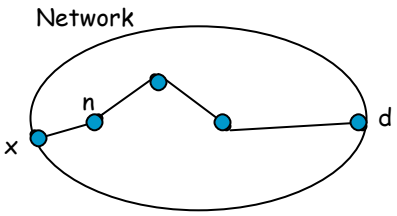
  p_d is the best neighbor to forward packet to for d.
- Question: how do we update these local projections V_d for each node s?

$$V_d(s) = \min_{s' \in Nbs(s)} \left[r(s,s') + V_d(s') \right]$$

A space of RL algorithms for routing

- Information propagation strategy for V estimation
  - Indirect path sampling: get neighbors to supply r(s,s') and $V_d(s')$.
  - Direct path sampling: s sends probes to d and gets path cost from s to d directly.
- Action choice
  - deterministic or stochastic routing.
- Policy computation
  - Indirectly computed from value function.
  - Direct policy update (no value function maintained).

Direct & indirect path sampling



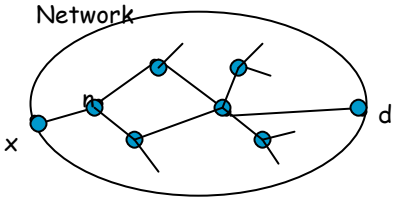
INDIRECT PATH SAMPLING

$V(x,d) = \min_n[r(x,n) + V(n,d)]$, where n ranges over neighbors of x. Neighbors exchange value functions. (classical RL)

DIRECT PATH SAMPLING

Probes travel end-to-end from x to d informing routers in between of cost to x.

Stochastic sampling & flooding for direct path sampling



STOCHASTIC PATH SAMPLING

When a probe reaches node n, it is probabilistically routed to ONE of the neighbors of n.

FLOODING

When a probe reaches node n, it is forwarded to all neighbors of n except for the one that it was received from.

Traditional RL solution: distance vector algorithm

- Information propagation: indirect path sampling
- Action choice: deterministic routing
- Policy computation: indirect via value function update
  - $V(x,d) = \min_n[r(x,n) + V(n,d)]$, where n ranges over neighbors of x.

Routing with ants (IJCAI)

- Information propagation:
  - direct sampling
  - stochastic
- Action choice: stochastic routing policy at every router
  - Probabilistic forwarding table entry at x: $(s, (y_1,p_1), \ldots, (y_n,p_n))$
- Policy computation: direct policy update
  - routing probabilities are modified using path costs announced by ants.

Routing with ants: the details (1)

- Each node s periodically sends an ant [s,d,C].
- When node x receives ant [s,d,C] from neighbor y_i,
  - x adds to C the cost of link (x,y_i).
  - x updates its probabilistic forwarding table entry for node s:

$$p_i = \frac{p_i + \Delta p}{1 + \Delta p}$$

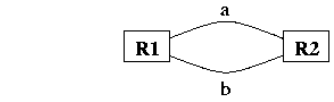
$$p_j = \frac{p_j}{1 + \Delta p}, \text{for } 1 \le j \le n, i \ne j$$

$$\Delta p = \frac{k}{C}, k > 0$$

Routing with ants: the details (2)

- Regular ants
  - x forwards ant [s,d,C'] to a neighbor according to its data packet forwarding policy for node d (probabilistic policy).
  - Ant [s,d,C'] is destroyed when it reaches node d.
- Uniform ants
  - x forwards ant [s,d,C'] to one of its neighbors with equal probability.
  - Ant [s,d,C'] destroyed when C' exceeds a threshold.

Regular ants (theoretical analysis)



dest	a	b
R2	p(t)	1 - p(t)

R1's forwarding table

dest	a	b
R1	q(t)	1 - q(t)

R2's forwarding table

$$p(t+1) = \frac{p(t)+\Delta p_a}{1+\Delta p_a} \text{ with prob } q(t)$$

$$= \frac{p(t)}{1+\Delta p_b} \text{ with prob } 1-q(t)$$

$$q(t+1) = \frac{q(t)+\Delta p_b}{1+\Delta p_b} \text{ with prob } p(t)$$

$$= \frac{q(t)}{1+\Delta p_a} \text{ with prob } 1-p(t)$$

$\Delta p_a = k/C_a$

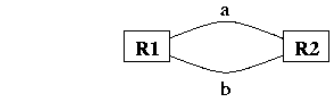
$\Delta p_b = k/C_b$

If $C_a < C_b$ then $p(t) \rightarrow 1$ and $q(t) \rightarrow 1$ as $t \rightarrow$ infinity

Properties of regular ants

- Bad news travels fast.
- Good news travels slow.

Uniform ants (theoretical analysis)



dest	a	b
R2	p(t)	1 - p(t)

R1's forwarding table

dest	a	b
R1	q(t)	1 - q(t)

R2's forwarding table

$$p(t+1) = \frac{p(t)+\Delta p_a}{1+\Delta p_a} \text{ with prob } 0.5$$

$$= \frac{p(t)}{1+\Delta p_b} \text{ with prob } 0.5$$

$$q(t+1) = \frac{q(t)+\Delta p_b}{1+\Delta p_b} \text{ with prob } 0.5$$

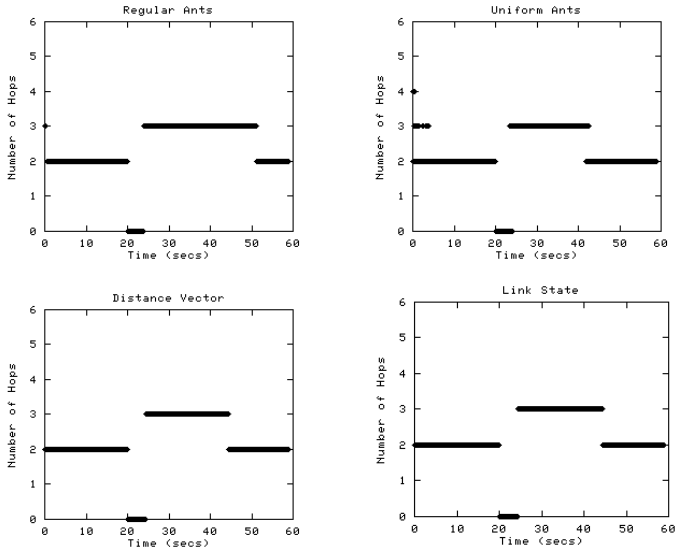
$$= \frac{q(t)}{1+\Delta p_a} \text{ with prob } 0.5$$

Traffic is divided in inverse proportion to the link costs.
Multi-path routing!!

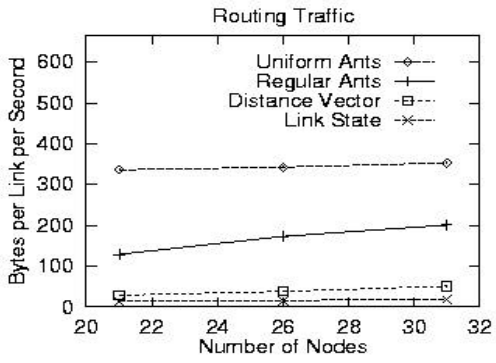
Routing performance study

- We test routing traffic scalability by
  - Fixing the convergence times for the algorithms: uniform and regular ants, and two state-of-the-art routing algorithms: Distance Vector and Link State.
  - Creating a parametric "Internet like" network topology: tree with cross edges.
  - Measuring routing traffic in bytes/sec/link in network as size of network increases.

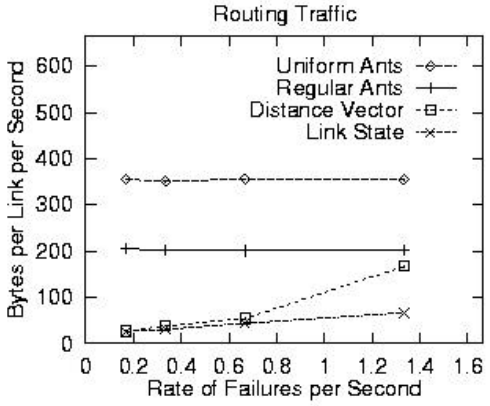
Path convergence times



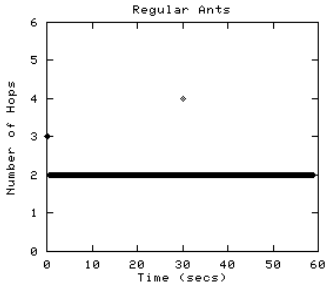
Routing traffic with Internet-like topology

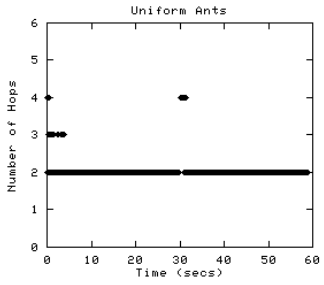


Routing traffic with topology changes



Resilience to state corruption





Both regular and uniform ants perform exceedingly well in the face of state corruption in routers.

Lessons learned

- Direct policy updates slow in responding to topology changes compared to state-of-the-art solution (distance vector).
- Stochastic path sampling (biased random or uniform random) requires an order of magnitude more routing resources than conventional solution.
- Stochastic path sampling offers a degree of resilience unmatched by conventional solution.

The Scout routing algorithm

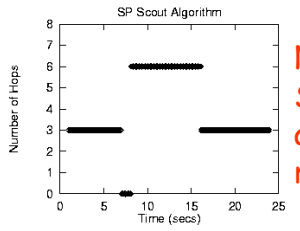
- Information propagation:
  - direct
  - flooding
- Action choice: deterministic routing
- Policy computation: indirect via value function update
  - V(x,d) : cost of path from x to d revised by ants.

Ants with all the stochastic stuff thrown away!

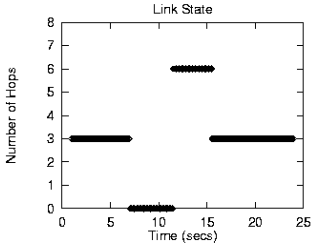
Scout routing algorithm (the details)

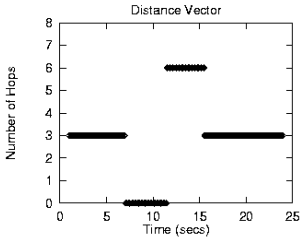
- A node x sends scout [x,0] to each of its neighbors.
- When node y receives scout [x,C] from neighbor z,
  - it updates C to C′ by adding cost of link (y,z) to C.
  - it compares its current estimate of path cost to x to C′; if C′ < x then it updates its next-hop to x to be z.
  - it forwards [x,C′] to every neighbor other than z.

Scout performance on Internet-like topology

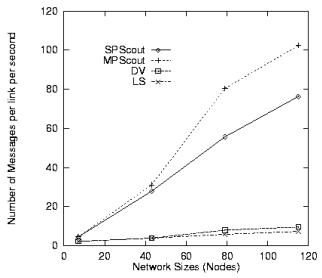


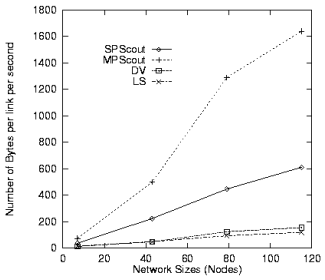
Note Scout's fast convergence! Scout convergence times can be adjusted by varying the sampling rate.





Scout routing costs





Both in terms of messages and bytes, Scout requires 1-2 orders of magnitude more resources for the same routing performance.

Lessons learned

- Direct path sampling is very expensive because route computations are not aggregated.

Adapting to network congestion: traditional solution

- Link costs depend on traffic conditions. Path re-computation triggered when the change in a link's cost exceeds a specified threshold.
- All paths affected by cost change are recalculated.
- Recalculations occur quasi-simultaneously.
- First implemented in early 1980s on the Arpanet; providing 50% improvements in round-trip delay and 18% improvement in throughput.

Problems with traditional solution

- All-pairs and quasi-simultaneous path re-computations cause routing instabilities (e.g., oscillations)
- Routing traffic is data traffic dependent (need thresholds and hold-downs to control routing traffic overhead).

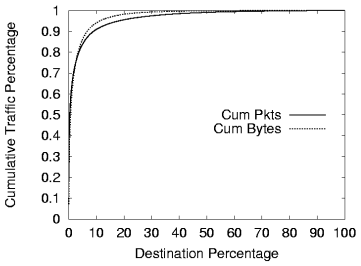
Lessons learned from Scout algorithm

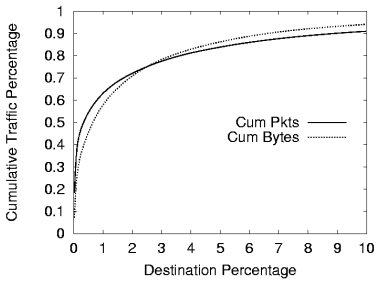
- Direct path sampling allows path re-computations to be uncorrelated across time, so that all nodes in the network do not simultaneously switch paths in response to congestion (route oscillation).
- Therefore, direct path sampling is excellent for adapting paths to network congestion.

Harnessing the power of direct path sampling

- Sample paths to a limited number of "hot" destinations.

Network traffic locality



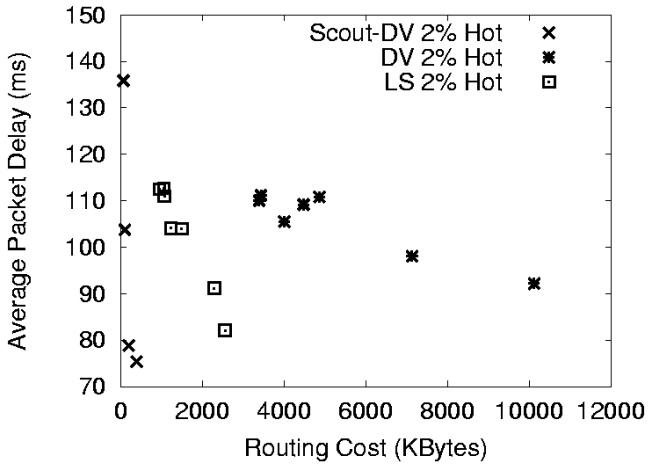


Analyzed traffic traces from DEC and LBL.
Top 10% of destinations receive over 90% of the traffic. Top 1% of destinations receive over 60% of the traffic.

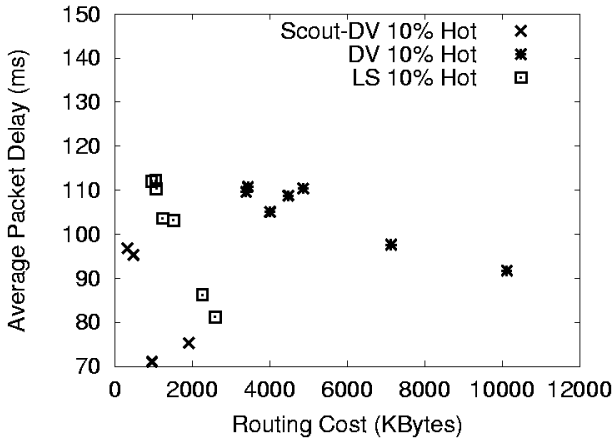
Hybrid Scout (Infocomm)

- Direct path sampling with flooding (Scout) for hot destinations alone.
- Indirect path sampling (Distance vector) for all other nodes.
- Deterministic routing and indirect policy computation via value function update in both cases.

Routing costs vs performance



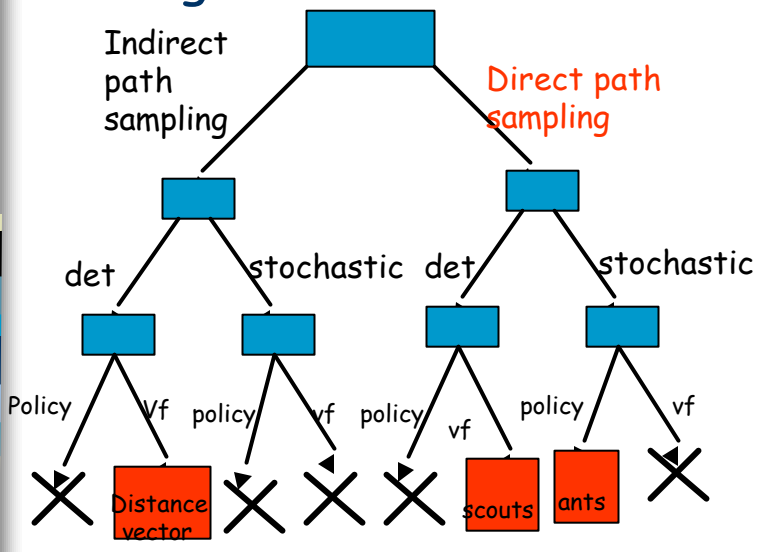
Routing cost vs performance



Lessons learned

- Indirect path sampling performs poorly in non-stationary environments.
  - Route oscillation problems.
  - inability to solve selective portions of the all-pairs-shortest-path problem.
- Direct path sampling overcomes stability problems in non-stationary environments, but performance requirements restrict its use to "important" nodes.

Reinforcement learning & routing



Conclusions

- Reinforcement learning algorithms for non-stationary environments can be built and they can be very effective.
- Blending direct and indirect path sampling is the key to handling non-stationarity. Direct path sampling decouples path re-computations in changing network providing stability and convergence. Indirect path sampling's efficiency hard to beat under stationary conditions!
- Stochastic action choice cannot compete with deterministic action choice in Internet routing.
- Direct policy update is inefficient compared to value function updates.

Open questions

- How do we formulate convergence criteria for non-stationary environments?
- How do we prove convergence of the direct sampling reinforcement learning methods?