Analysis of Probabilistic Temporal Networks

Xiang Fu*, Shangdi Yu*

Abstract

Models and algorithms for routing in networks have been studied in a number of domains, such as transportation systems, computer networks, and telecommunication. In this paper, we propose the Probabilistic Temporal Network (PTN) model, which is a non-deterministic network model that encodes the uncertainty of connections in the network. We show that a stochastic dynamic programming routing strategy is optimal, evaluate the performance of this routing algorithm, and show that the algorithm out-performs two other baseline heuristic algorithms.

I. Introduction

Uncertainties in connections between nodes are common in real-life networks. For example, in a transportation system, the roads between two locations might be temporarily blocked due to maintenance or inclement weather conditions; or in a computer network, the connection between two servers might be unstable; or in a social network, diseases spread depending on the occurrence of the wide spreading regime [5]. Naturally, routing in networks should take these uncertainties in the networks into consideration, just like a driver avoids taking a path consisting of roads that has high probability of being closed. However, considering uncertainty is challenging because then we are unsure about the exact network topology we will face when we make the routing decisions for the future. In this paper, we encode the uncertainty into our network and propose the *probabilistic temporal network* (PTN), where each edge has a probability of being ”available” during each time slot. We consider the routing problem in a PTN, where the goal is to minimize the expected total cost from a given starting node to a given destination node. In Section II, we give the details of the PTN model. Section III includes the optimal routing algorithm in a PTN and Section IV introduces two heuristic routing algorithms. We will prove that in expectation, the proposed optimal policy gives a path with the smallest cost between two nodes. The details of the proof are in the appendix. In section V, we describes the empirical comparisons of the performances of the optimal policy and the two heuristics.

II. Related Work

A key advantage of graph models is its ability to encode interactions between its nodes using the edges. Thus, it is not surprising that the ”reach-ability” between nodes in networks has long been an interest of many researchers. People have been developing different measures for the reach-ability between nodes, evaluating the reach-ability in the real-life networks, and finding routing policies and algorithms between nodes.

For example, Holme [1] studied the network reach-ability of real-world contact sequences on Internet communications and explored how fast can information, or disease, spread across an empirical contact network and how much of the network can be reached from one vertex through a series of contacts. The model we propose in this reaction paper also considers how fast one can reach from one node to another in time-relevant networks. Our problem is different, though, in that the availability of the edges in our model is not deterministic and we are considering a single time-respecting path between two given nodes. In other words, unlike the transmission of information or diseases where the object being transmitted can have several copies, the object of our transmission has only one copy and cannot be split. Analysis of other variants of temporal networks also exists in the literature. Kempe et al. [2] studied two groups of problems on temporal networks: connectivity problems and inference problems. Employing the concept of subdivision and topological minor, the paper specifies a group of graphs in which Menger's Theorem holds and gives a maximum set of disjoint time-respecting paths between node pairs in these graphs. It also introduces an algorithm that reconstructs a partially specified time labeling of a temporal network. Scholtes et al. [4] studied the higher-order measures that better capture the temporal centralities of nodes. Saramäki and Holme [3] studied the correlations between timings of node and link activations in temporal networks using temporal greedy walks to explore the temporal network structure.

In this paper, we focus on the routing in temporal networks and aim to answer the question of how many time slots do we need to reach the destination node d from the starting node s using only available edges in each time slot in a probabilistic temporal network. While such problem can be solved efficiently with Dijkstra's algorithm or Bellman-Ford algorithm in deterministic graphs, they are not suitable for the temporal graphs because of the uncertainties. We will solve the problem using a variant of stochastic dynamic programming.

III. Model Description

The PTN is a probabilistic network model with temporal dynamic. We define the directed underlying graph $G = (V, E)$ and the connection probability function $p : E \times \mathbb{N} \to [0, 1]$, where $p(e, i)$ is the probability of edge e being "available" at time slot i. Then the *probabilistic temporal network*, PTN(G, p) is a sequence of directed graph: $G_1, G_2, ..., G_i, ...$, where each $G_i = (V_i, E_i)$ in time slot i is generated independently by letting $V_i = V$, and for each edge $e \in E$, $e \in E_i$ with probability $p(e, i)$. We say edge e is available at time slot i if $e \in E_i$. A simple example of PTN is given below.

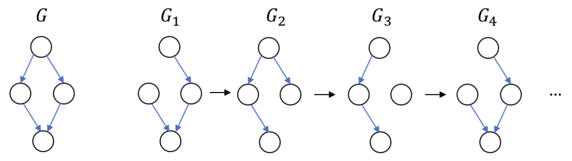

Figure 1: A simple example of the probabilistic temporal network.

We are interested in the routing problem under the following settings: Assuming there exists a path from s to d in the underlying graph. we start from node s and we want to get node d. At each time slot, we are only able to travel through one available edge or stay unmoved. For example, assume now we are at node x in graph G_i, and we have edge $(x, y) \in E_i$. If we decide to move from node x to node y during time slot i, then in time slot $i+1$, we will be at node y in the graph G_{i+1}. In other words, we are only able to move to a subset of nodes S_{xi} in time slot i where $S_{xi} = \{v : (x, v) \in E_i\}$, given we are at node i in time slot i. We are also allowed to stay unmoved at node x. After we make the routing decision, there is a **traveling cost** $c_{vw} \geq 0$ incurred if we travel on the edge (v, w) and a **stalling cost** $c_{vv} \geq 0$ incurred if we do not move and stay at the node v for one time slot. Another way to think of stalling is by adding a self loop edge (v, v) for each node $v \neq d$ that shows up at every time slot and the cost for that edge is c_{vv}. The traveling/stalling costs are incurred in each time slot and add up cumulatively as we travel through the networks.

With $p_{det}(e, i) = 1, \forall e \in E, i \in \mathbb{N}$, the PTN is just a deterministic directed graph with edge costs. In this case, the optimal routing problem is just the shortest path problem in a directed graph, which can be solved efficiently with Dijkstra's algorithm or Bellman-Ford algorithm. The routing problem becomes much more complicated when $p \neq p_{det}$.

Assuming in the underlying graph $G = (V, E)$, there exists a path from s to d. Then in the probabilistic temporal network, we can get d from s for some **total cost** T. Consider the shortest $s-d$ path P in the underlying graph G given by Dijkstra, we will have in the probabilistic temporal network, $T \geq \sum_{e \in P} c_e$. Let $T^* = \sum_{e \in P} c_e$, which is actually the minimum total traveling cost from s to d. Note that d includes not only the traveling costs, but also the stalling costs. Given some routing policy A, we can compute the expected total cost T_A under this policy. We define the **delay ratio** as $\frac{T_A}{T}$.

For routing in a PTN, we might not be able to take the shortest path in every time slot as the graph is changing in each time slot and the edge for the shortest path may not be available in the time slot we need to take it. We can either stall and pay the stalling cost c_{vv} or we can change our route to some path that is not the shortest. If we change our route and deviate from the optimal path in the underlying graph, we will end up taking a longer path and pay extra traveling costs; however, if we stick to the shortest path in the underlying graph and wait for the edges in the path to be available, we need to pay the stalling cost. There is no way to know in advance which edges will be available in the future time slots, so we have to face this trade off. Since we do not know the exact total cost from s to d before we finish our route, we can only estimate the cost of different routes. We are interested in finding good routing policies in order to minimize the expected total cost for getting node d from node s.

IV. Optimal Routing Algorithm

A. Algorithm Description

The optimal routing policy minimize the expected total cost to travel from node s to node d. We define $w(v)$ to be the expected total cost under the optimal routing policy to travel from node v to node d.

The optimal routing algorithm is optimal based on following assumption:

Table I: Notation

NOTATION	MEANING
s	STARTING NODE
d	DESTINATION NODE
T_A	TOTAL COST OF GETTING FROM s TO d IN A PTN USING POLICY A
T^*	MINIMUM TOTAL COST OF GETTING FROM s TO d IN A PTN
$G(V, E)$	GRAPH WITH NODES V AND EDGES E
$p(e, i)$	PROBABILITY OF EDGE e BEING "AVAILABLE" AT TIME SLOT i.
$S_{xi} = \{v : (x, v) \in E_i\}$	THE SET OF REACHABLE NEIGHBORS OF x AT TIME SLOT i.
c_{vw}	COST OF TAKING EDGE (v, w) .
c_{vv}	COST OF STAYING AT NODE v FOR A TIME SLOT.
P	PATH
$N(v)$	NEIGHBORS OF NODE v
$w(v)$	THE EXPECTED TOTAL COST UNDER THE OPTIMAL ROUTING POLICY TO TRAVEL FROM NODE v TO THE DESTINATION NODE d.
$P(x)$	THE PROBABILITY THAT WE WILL GO FROM CURRENT NODE TO NODE x IN THE NEXT TIME SLOT.

Assumption 1.

$$c_{vv} \leq c_{xy}, \forall v, x, y \in V, x \neq y$$

That is, all stalling costs are less than or equal to the minimum traveling cost.

Only when this assumption holds, does the graph has a topological order in terms of w. That is, when this assumption holds, in the optimal routing policy, a node v with $w(v)$ and neighbor $N(v)$ will only want to go to nodes with smaller w values. This makes the dynamic program approach feasible: the value of $w(v)$ only depends on the values of w's for other nodes that are smaller than $w(v)$. We also have $w(d) = 0$ as the base case.

In the following analysis, we will assume that 1 always holds. For the simplicity of the analysis, without loss of generality, we assume $p(e, i) = p, \forall e \in E, i \in \mathbb{N}$. That is, all edges have the same probability to be available in all time slots. The optimal routing policy works as the following and we will prove the correctness of this algorithm in the next section.

Say in some time slot i we are at node v, the routing policy will give the next node we will be at in time slot $i + 1$. We will check the possible destination in the graph G_i. We will only want to go to node with smaller (expected total cost to d + cost(v to that node)) than (expected total cost from v to d + cost(stalling at v)) from node v otherwise we would rater stay at node v. And we will always want to go the the node with smallest (expected total cost to d + cost(v to that node)) if that edge is available in G_i. That is, among all the possible destinations for node v in time slot i (always includes node v itself), we will always choose the node x with smallest $w(x) + c_{vx}$ as the node we will be at in time slot $i + 1$. There is an example later that further explain the routing algorithm. Now we can formulate the following stochastic dynamic programming algorithm 1 to calculate $w(v)$ for all $v \in V$ with the input to be the underlying graph G. For the simplicity of our algorithm, we define the following notations:

- Define $N(v) = \{x : (v, x) \in \mathcal{E} \; and \; x \neq v\}$: $x \in N(v)$ if $x \neq v$ and there is a directed edge (v, x) in $\mathcal{E}$. $N(v)$ is just the possible destinations for node v (other than staying at v).
- Say we are at node v in the current time slot and we are moving under the optimal routing policy, denote $P(x)$ as the probability that v will go to node x. Note that the probability that we will stay at v is just $P(v) = 1 - \sum_{x \in N(v), c_{vx} + w[x] < c_{vv} + w[v]} P(x)$.

For example, for the underlying graph G in the following figure 2, if we are currently at the node s and we are trying to go to node d, we will have $N(s) = \{a, b, c\}$. Also we can stay at node s. For all possible destinations for s, sort all the nodes according to (the expected total cost from that node to d + cost(s to that node)), we will have $a < b < s < c$. So when we are at node s, we will never want to go to node c. If the edge (s, a) is available, we will always want to go to node a. if the edge (s, a) is not available and the edge (s, b) is available, we will always want to go to node b. If both edge (s, a) and edge (s, b) are not available, we will just stay at node s as we will never want to go to node c.
In this case, we have:

$$P(a) = \mathbb{P}(\text{edge (s,a) available}) = p$$

$$\begin{aligned} P(b) &= \mathbb{P}(\text{edge (s,a) not available and edge (s,b) available}) \\ &= \mathbb{P}(\text{(s,a) not available})\mathbb{P}(\text{(s,b) available}) = (1-p)p \end{aligned}$$

$$P(v) = \mathbb{P}(\text{stay at node } v) = 1 - \sum_{x \in N(v), c_{vx} + w[x] < c_{vv} + w[v]} P(x) = 1 - p - p(1-p)$$

So according to our routing policy, we can get the expected total cost from node s to node d, $w(s)$, by solving the equation:

$$w(s) = p(w(a) + 1) + p(1-p)(w(b) + 2) + (1 - p - p(1-p))(w(s) + 3)$$

In general for any node v, we can solve for $w(v)$ by solving the equation:

$$w(v) = \sum_{x \in N(v), c_{vx} + w(x) < c_{vv} + w(v)} (c_{vx} + w(x))P(x) + (1 - \sum_{x \in N(v), c_{vx} + w(x) < c_{vv} + w(v)} P(x))(c_{vv} + w(v))$$

which gives:

$$w(v) = \frac{\sum_{x \in N(v), c_{vx} + w(x) < c_{vv} + w(v)} (c_{vx} + w(x))P(x) + (1 - \sum_{x \in N(v), c_{vx} + w(x) < c_{vv} + w(v)} P(x))c_{vv}}{\sum_{x \in N(v), c_{vx} + w(x) < c_{vv} + w(v)} P(x)}$$

And we will be able to solve for $w(a)$ and $w(b)$ following the similar procedure. Note the expected total cost from node d to node d is just 0. That is, $w(d) = 0$.
After we calculate $w(v)$ for all node $v \in V$, we can make routing decision according to the policy stated above: Among all the available destinations(including the current node itself), we always go to the node with the smallest expected total cost(w value).

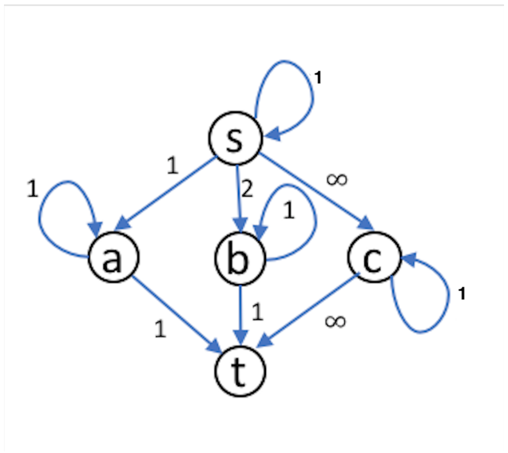

Figure 2: An example underlying graph. The traveling cost/stalling cost is marked along each edge.

It's not hard to notice that, in order to calculate $w(v)$ correctly for some node v, we will need to know the set: $\{x \in N(v), c_vx + w(x) < c_vv + w(v)\}$. How can we know this set before we know $w(v)$? In order to solve this problem we will use the dynamic programming algorithm 1, in which we initialize $w(d) = 0$ and $w(v) = \infty$ for all $v \in V, v \neq d$, and then keep updating w for every node according to the formula given above. If $w(v)$ doesn't change for all nodes $v \in V$, we have w converged. We then stop the algorithm and output w, which is the correct expected total cost under optimal policy for each $v \in V$. We will prove that this algorithm will converge within finite iterations and prove the correctness of this algorithm in the next section.

Algorithm 1 Generating Optimal Routing Table

```
1:  Set w[d] = 0, set w[v] = $\infty$ for all $v \in V, v \neq t$
2:  $flag = 1$
3:  while $flag$ do
4:      $flag = 0$
5:      for all $v \in V$ do
6:          $w'[v] \leftarrow \frac{\sum_{x\in N(v), c_{vx}+w[x]<c_{vv}+w[v]}(c_{vx}+w[x])P(x)+(1-\sum_{x\in N(v), c_{vx}+w[x]<c_{vv}+w[v]} P(x))c_{vv}}{\sum_{x\in N(v), c_{vx}+w[x]<c_{vv}+w[v]} P(x)}$
7:          if $w'[v] < w[v]$ then
8:              $flag = 1$
9:              $w[v] \leftarrow w'[v]$
10: return $w$
```

B. Algorithm Correctness

We give the full details of proofs in the appendix, and only sketch the construction here.

Lemma 1. *Given the policy: In any time slot i and node v, among all the available destinations ($\{v, x : (v, x) \in \mathcal{E}_i\} \cap (N(v) \cup \{v\})$), we select the node x with the smallest sum of edge cost and w value to be the node we will* *be at in time slot $(i+1)$. We claim that our algorithm calculates the expected value under this policy and we will not move to any of the nodes $\{x_{large} : w(x_{large}) + c_{vx_{large}} \geq w(v) + c_{vv}\}$ in the next time slot under this policy.*

Lemma 2. *We say $w(v)$ correct if it is indeed the expected cost under the policy we described above. We claim our algorithm gives the $(i+1)^{th}$ smallest $w(v)$ to node v in i^{th} iteration(starting from iteration 0). And if we fail to calculate the correct w value for some node v, we must have overestimated it.*

Lemma 3. *We won't change the w label of a node if we have labeled it correctly.*

Lemma 4. *Calculate the expected cost from each node to d under this policy gives $w(v)$ for all node $v \in V, v \neq d$: the smallest expected total cost from each node $v \in V$ to the destination node d. (Optimality of our routing algorithm.)*

Theorem 1. *The optimal routing algorithm gives the routing policy with smallest expected total cost and finishes in $O(n)$ steps.*

Proof. We show if the algorithm is correct, the algorithm terminates in finite time. By Lemma 2, we will get one node correct at each step. By Lemma 4, we won't change the label of a node if the node has been labeled with the true $w(v)$. As a result, in each step, the number of nodes labeled correctly will increase by at least 1. Since we have n nodes, in at most n steps, we will have correctly labeled all nodes.
Correspondingly in Algorithm 1 line 7-9, we only change flag to 1 if we update the label of a node. After we correctly labeled all nodes, in the next iteration, the condition in line 3 "while flag = 1" does not hold. The algorithm converges and thus stops. At this time, all nodes are labeled with the true $w(v)$. By definition, all nodes are labeled with the smallest expected cost to go from this node to the destination node d. Then we can prove by induction that this routing algorithm is indeed optimal. □

V. Heuristic Routing Algorithms

For a routing policy A, we say the expected total cost using the routing policy A is T_A. And we say the *Delay* of policy A is $T_A - T^*$. So even the optimal routing policy for the problem will have positive delay if $p < 1$. We start by analyzing the following heuristic policies to investigate the routing behavior of the probabilistic temporal network. We define $l(v)$ to be the total cost(length) of the shortest path from v to d and $l(v) = \infty$ if there doesn't exist a path from v to d. Say at time slot i, we are at node x, and we have the available destinations $S_{xd} = \{v : (a, v) \in E_i\}$ to move to or we can stall.

A. *Take Available Shortest Path policy (TASP)*

Under the Take Available Shortest Path policy, we will stall only when $S_{xd} = \emptyset$ or $\forall v \in S_{xd}$, d is not reachable by v. We will move to node v^* where:

$$v^* = \begin{cases} argmin_{v \in S_{xd}} l(v) + c_{xv} & \text{if } \exists v \in E_i \text{ such that } l(v) + c_{xv} \text{ is finite} \\ x & \text{otherwise} \end{cases}$$

Under this policy, we try our best to avoid stalling.

B. *Always Wait policy (AW)*

Under the Always Wait policy, we will always stall when we are not able to take a shortest path in the underlying graph G. The next node v^* to go to ("go to" the current node itself if stall) is given by the expression:

$$v^* = \begin{cases} v' & \text{if } \exists v' \in E_i \text{ such that } c_{xv} + l(v') = l(x)) \\ x & \text{otherwise} \end{cases}$$

Under this policy, we will always keep stalling if we are not able to take a shortest path in the underlying graph. If there exists unique shortest path in the underlying graph, we will have the number of stall under this policy follows a negative binomial distribution: say the there are q edges on the shortest path form s to d, number of stall $= NB(q, 1-p)$. So the expected number of stall is just $\frac{(1-p)q}{p}$. So the expected delay is $\frac{(1-p)q\xi}{p}$.

VI. Simulation Result

We simulated the two heuristics(AW and TSAP) and the optimal algorithm on random k-degree graphs with different parameters. The result of simulation shows that our optimal algorithm consistently performs well in different graphs. Simulation also shows that when assumption 1 does not hold, our policy still serves as a good heuristic.

Though the AW policy performs well in the random k-degree graphs, this is not necessarily the case in other graphs. We will show below by giving an example that the AW policy will perform extremely bad in some graphs.

For all simulations, we test each algorithm on 500 random graphs. Let D be the cost of the shortest path in the underlying graph, and D' be the actual cost to go from s to t in one simulation iteration. We define the delay rate as $d = (D' - D)/D$. The x-axis of the graphs is $\sqrt{d}$. The y-axis are the frequency of the delay rates out of 500 iterations per policy.

In our simulations on k-degree random graphs, the AW performs really well. We think its due the following reasons:

- According to the assumption we make, the "stalling cost" is relatively small. The AW policy takes advantage from this assumption as AW policy only pay extra stalling cost and never pay extra traveling cost.
- The k-degree random graphs have small-world property, so there tends to be a very short path from the source node d the destination node. As we can observe in a lot of cases OPT even makes the same decisions as the AW policy.

- OPT guarantees smallest expected cost, but the variance can be very high according to simulation. So when the cost of self loops are very small, intuitively AW is a competitive policy A has very low variance, this is the same result we get in simulation. The delay rate is bounded below by 0. Though the variance is large, for OPT we cannot extrapolate to negative. So we observe that in some simulations AW policy even have slightly better numbers than OPT.

OPT guarantees smallest expected cost, but the variance can be very high according to simulation. So when the cost of self loops are very small, intuitively AW is a competitive policy and has very low variance, this is the same result we get in simulation. The the cost of self loop is small, the OPT policy will have similar or even worse cost. The delay rate is bounded below by 0. Though the variance is large, we cannot extrapolate to negative.

A. Algorithm Performance

In this section, we show that the performance of TASP in 3-degree random graphs is significantly worse than the AW policy and the OPT policy. Let p denote the probability that an edge is available in a certain timeslot. For each $p = 0.1, 0.2, \ldots, 0.8, 0.9$, we run each of the three algorithms, TASP, AW, and OPT, on 500 randomly generated 3-degree graphs.
From the result of the simulation, we can see that when p is mall, TASP's delay rates are significantly larger. As p goes to 1, the distribution of the delay rates of TASP becomes more and more like that of the other two policies. This means that when the edges are likely to be available, TASP will make similar decision as the other two policies. This is what we expected because when p is large, the shortest path in the underlying graph is more likely to be available; notice that AW policy will only take paths that are shortest in the underlying graph. (Figure 3)

Another noticeable pattern in Figure 3 is that the delay rates of the OPT and AW policy Are very similar. We will show later that this is not always the case - in some graphs, AW policy will make very different decisions as the OPT policy. Here, according to our simulation results, AW policy produces very similar delay rates as OPT does in k-degree random graphs. We discussed the reason at the beginning of this section. In fact, it is even possible that under some seeds, the AW will make exactly the same decision as OPT in all 500 randomly generated temporal graphs. One example is Figure 4. The red(OPT) and blue(AW) areas overlap and show a color of purple.

Figure 5 are graphs that contain only AW and OPT policy. We exclude the TASP because we showed in 3 that TASP has significantly larger delay rates. Removing TASP from the plots allows us to plot the delay rates of AW and OPT more compactly. The plots in Figure 5 again show that AW has similar performance as OPT in k-degree random graphs when our assumption 1 holds.

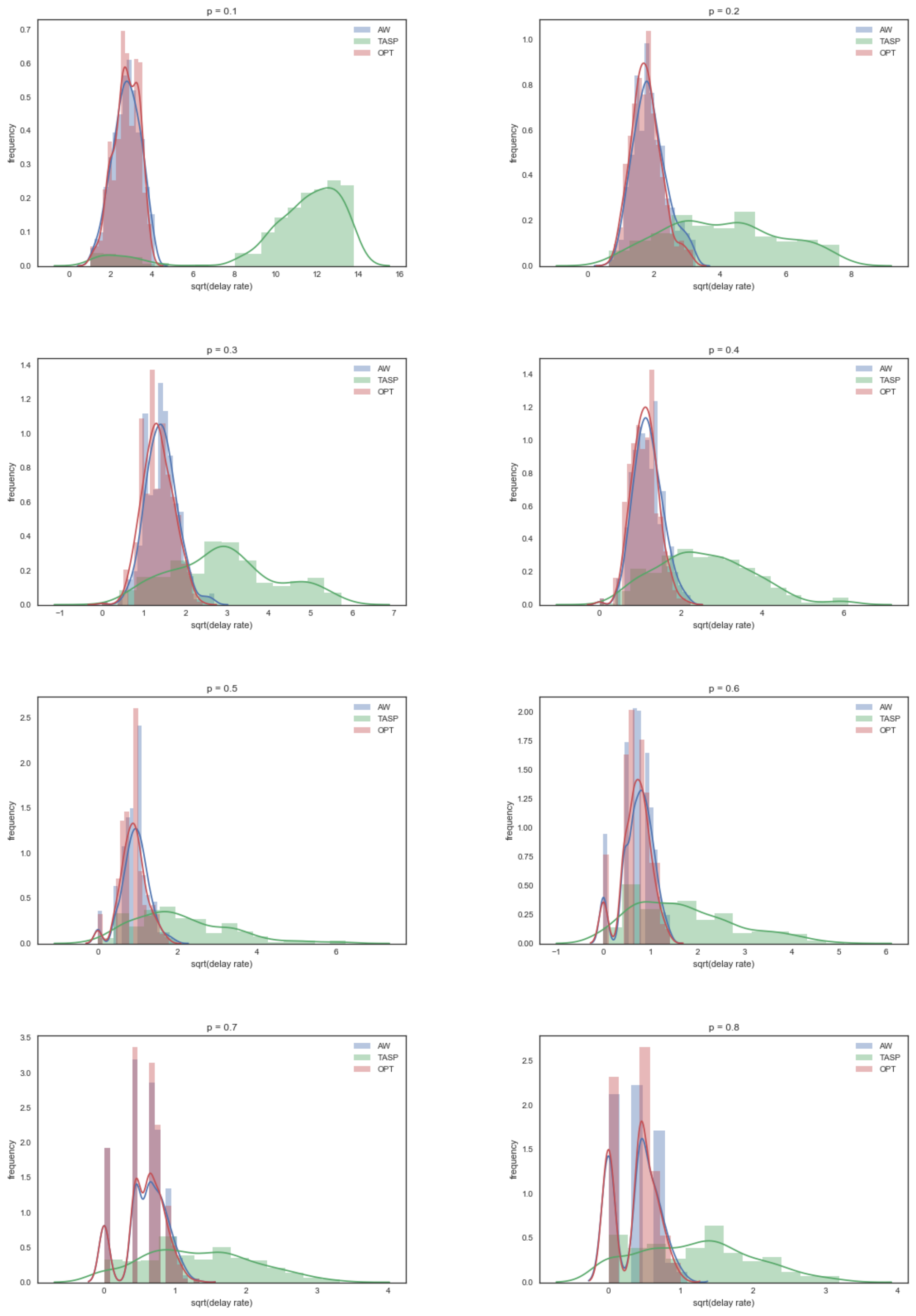

Figure 3: AW, TASP, and OPT run on 500 3-degree random graphs. The number of nodes n is 100 and all edge costs are 1. The probability of each edge $p \in [0.1, 0.8]$

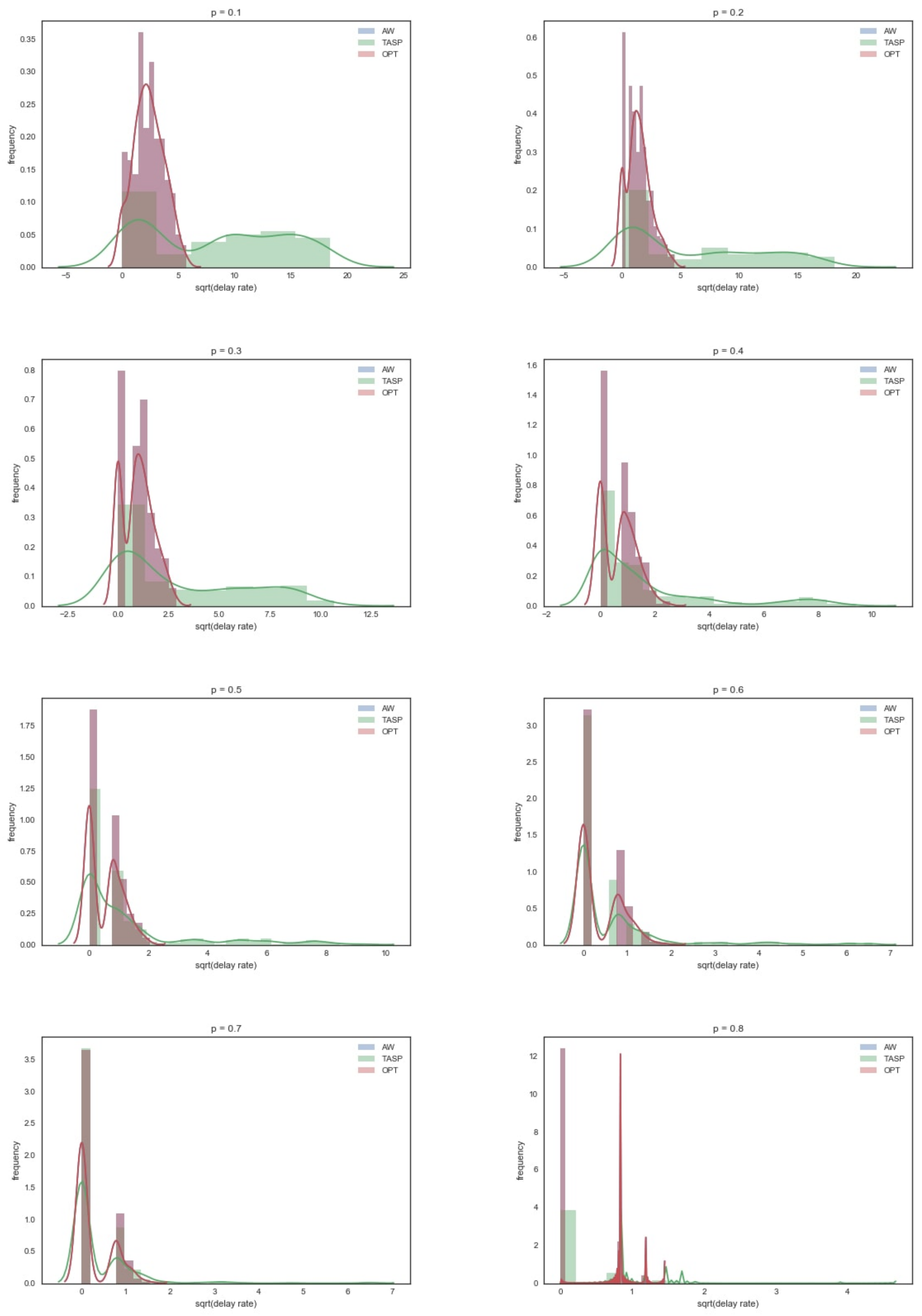

Figure 4: AW, TASP, and OPT run on 500 3-degree random graphs. The cost of each edge(except self-loops) is a random number between 1 and 2. The cost of self-loops are between 1 and the minimum non-self-loop edge cost. The number of nodes n is 500. The probability of each edge $p \in [0.1, 0.8]$ seed = 666.

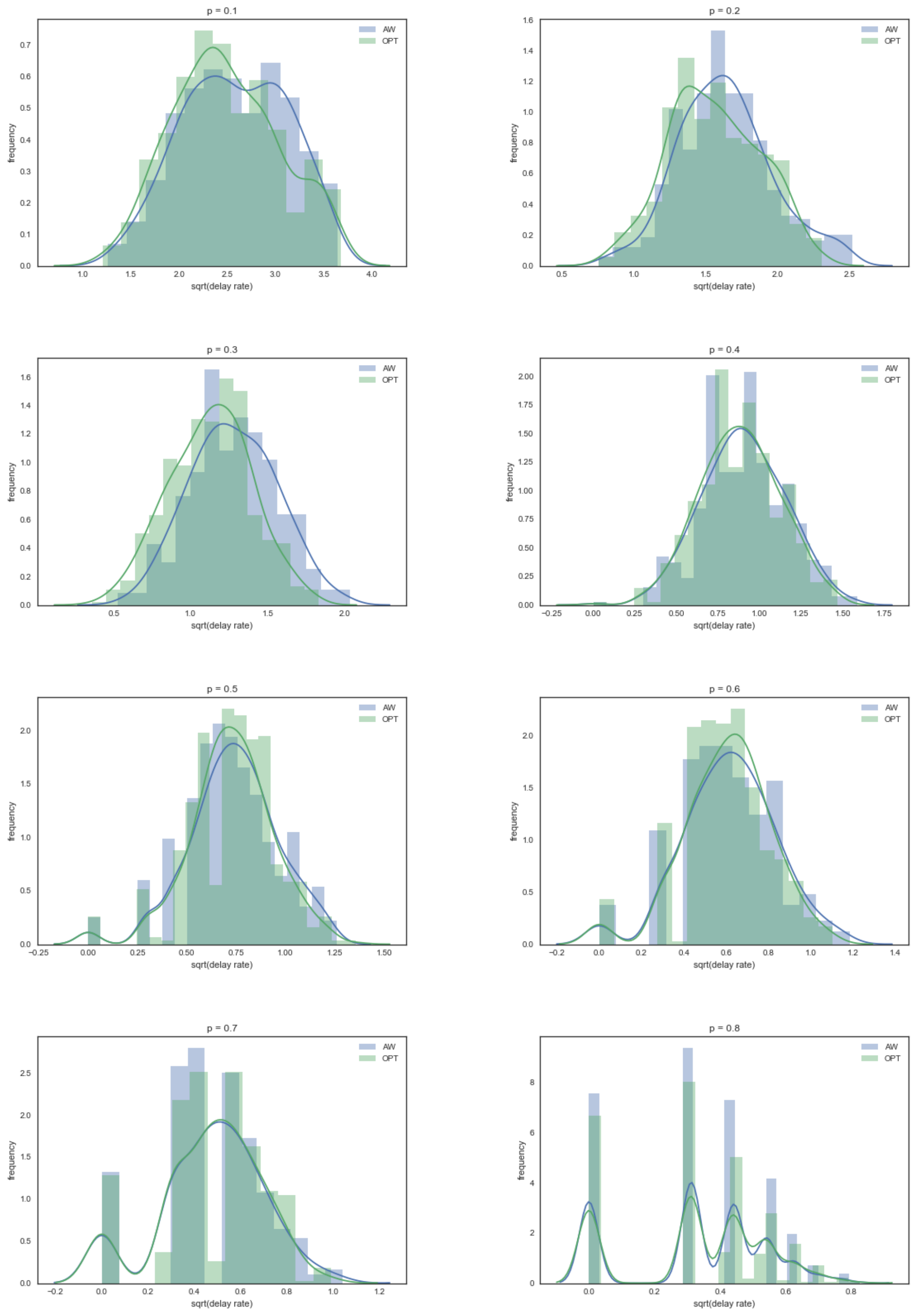

Figure 5: AW and OPT run on 500 2-degree random graphs. The cost of each edge(except self-loops) is a random number between 1 and 2. The cost of self-loops are between 1 and the minimum non-self-loop edge cost. The number of nodes n is 100. The probability of each edge $p \in [0.1, 0.8]$ seed = 5.

B. PTN Instance that AW is bad

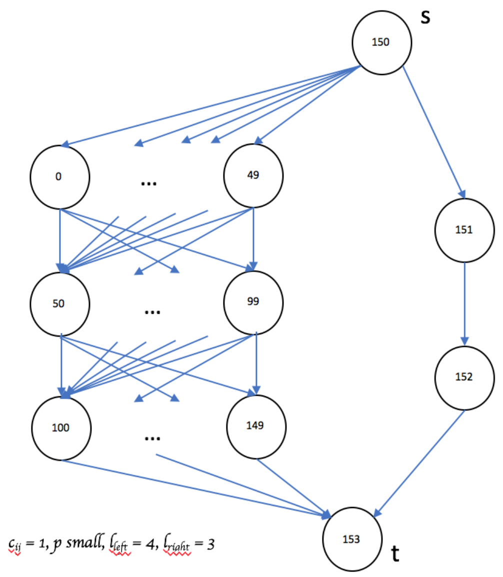

Figure 6: PTN Instance that AW Policy performs badly. All traveling/stalling cost = 1.

For a PTN with underlying graph as above. The AW policy, which performs pretty well in random k-degree graph will perform really badly when p is small. Here in the graph there is a ”broad way” on the left with distance 4 to the destination node d and a ”narrow way” of length 3 on the right to the destination node d. The AW policy will always choose to take the left ”narrow way”.

Here, the ”broad way” contains 3 layers of nodes, each contains 50 nodes, all connected to every node in the next layer; s can go to all 50 nodes in the first layer and ultimately the third layer can all go to the destination node. On the ”borad way” the probability of stalling in each step is $(1-p)^{50}$, which is very small. While the ”narrow way” is just one way, where the probability of stalling in each step is $(1-p)$. So we can expect a lot of stalls on the narrow way when p is small. Therefore, we can expect the AW policy performs badly in this PTN instance when p is small. We have the following simulation results at Figure 7.

As we expected, when p is small, the AW policy has much larger delay rate than the optimal policy and the TASP policy. As p increase, the gap between the AW policy and the other two policies narrows. Since the assumption 1 is satisfied, the optimal algorithm always performs well in this instance, no matter what p is.

C. OPT as a Heuristic when ”Self Loop Minimum” Assumption Fails

Our OPT algorithm requires that the ”stalling costs” cannot be larger than any ”traveling cost” as this requirement is necessary for us to find a topological order and perform the stochastic dynamic programming algorithm. Neverthe-

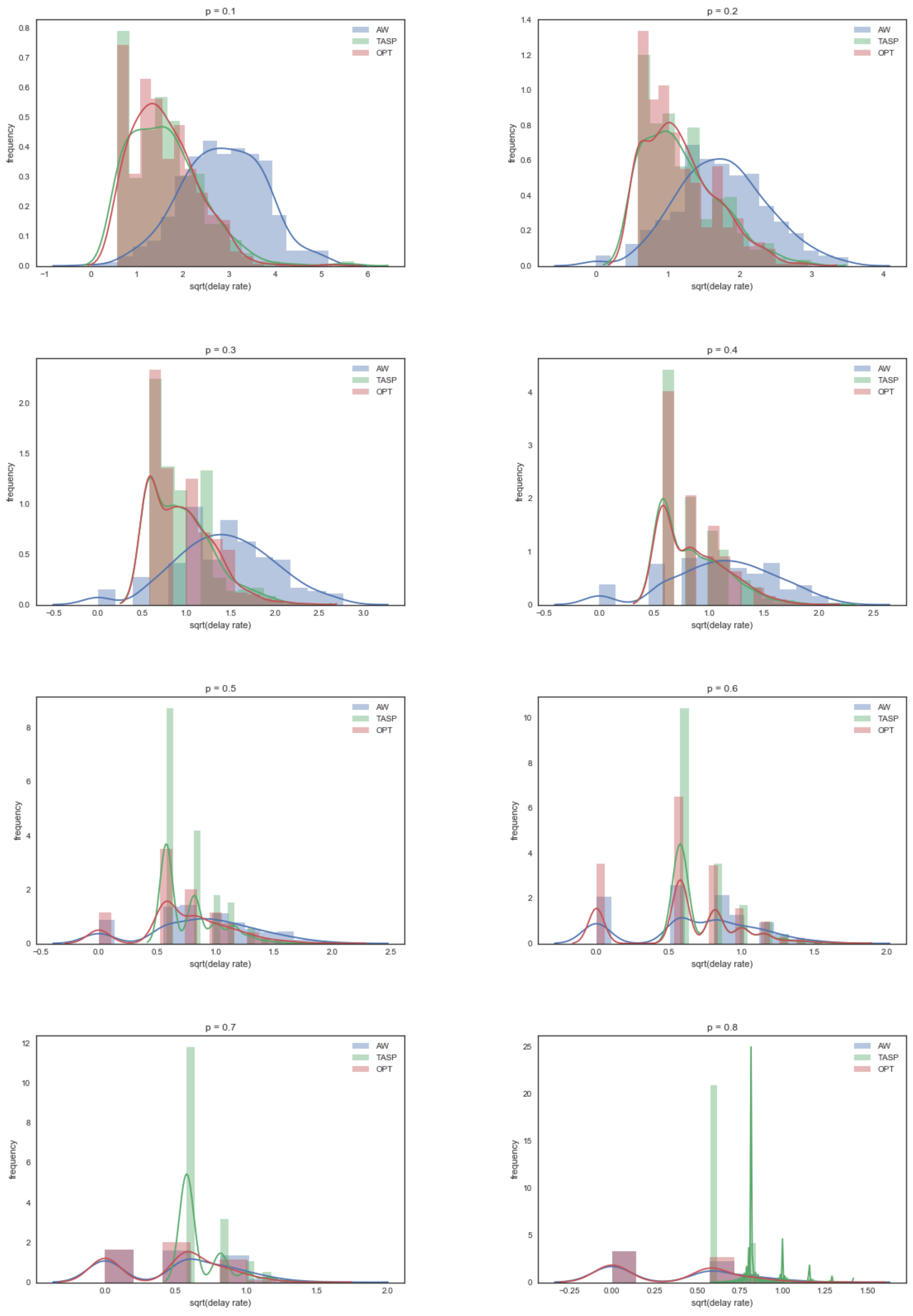

Figure 7: PTN Instance that AW Policy performs badly, graph as shown above. AW policy performs badly when p is small.

less, the result of our simulation shows that our OPT algorithm still performs relatively well when the assumption is violated. As shown in Figure 8, our OPT algorithm produce smaller delay rates in general under all levels of p.

D. *Distribution of* $\sqrt{delay\ rate}$

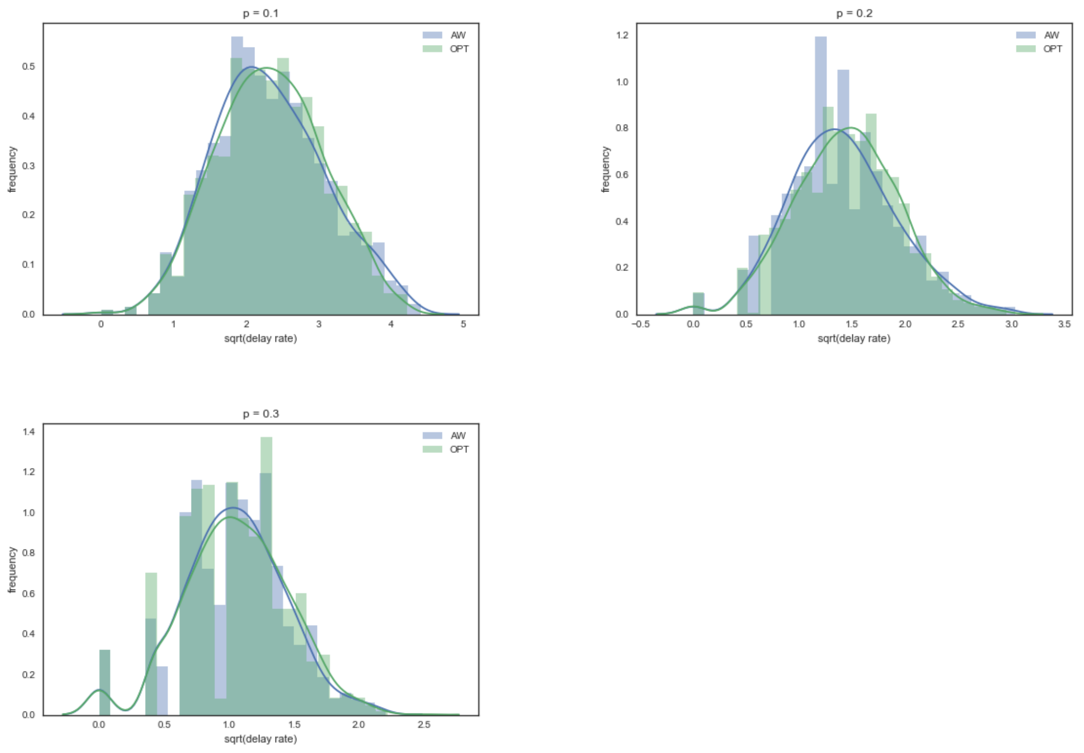

Figure 9: AW and OPT run on 2000 3-degree 25-node random graphs. The cost of each edge(except self-loops) is a random number between 1 and 2. The cost of self-loops are between 1 and the minimum non-self-loop edge cost. The probability of each edge $p \in [0.1, 0.3]$.

According to our simulation the result, the distribution of the square root of the delay rate on a k-degree random graph is approximately normal when p is small.

VII. Discussion

In this paper, we propose the PTN model, where in different time slots, edges are not always available. Each edge is available in each time slot with a probability. This model captures the uncertainty in a lot of real-life networks: such like ride-sharing network, information propagation network, and transportation network, etc. We investigate the routing problem in PTNs. We propose two heuristic routing policies and an optimal routing policy. We prove the optimality of the optimal routing policy when the assumption 1 holds. We run simulations on k-degree random graphs and some special graphs with these 3 routing policies and get some insight into routing in PTNs.

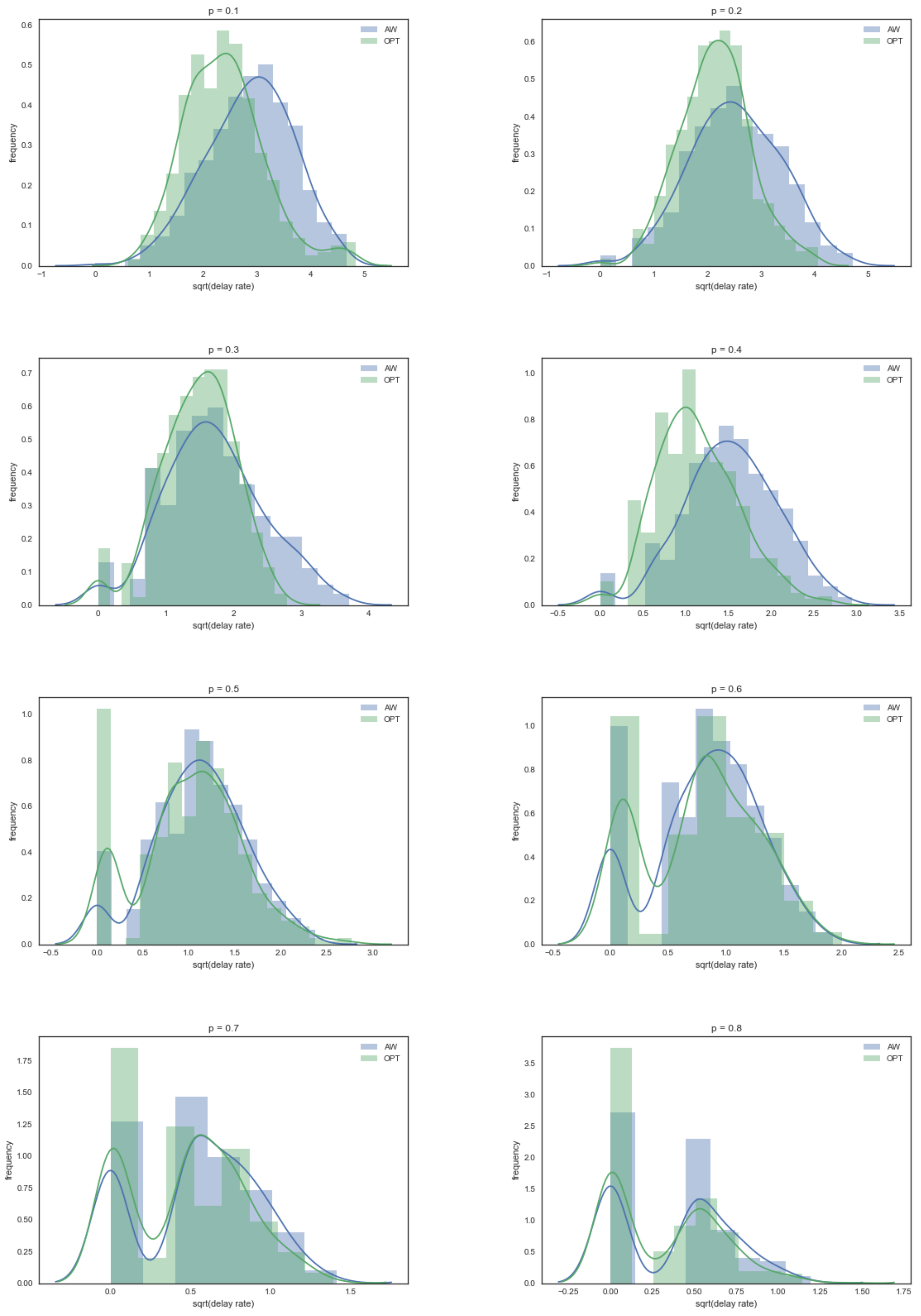

Figure 8: AW and OPT run on 500 3-degree random graphs. The cost of each edge(including self-loops) is a random number between 1 and 5. The number of nodes n is 30. The probability of each edge $p \in [0.1, 0.8]$. Note that assumption 1 is violated.

Some future directions that extends our current work include: (1) running simulations on random graphs that do not have the small world property; (2) developing an algorithm that finds the smallest expected cost of going from one node to another when the ”stalling cost smallest” assumption does not hold; (3) investigating other problems on probabilistic temporal networks, such as cascading problem. (4) Different probability models; for example, the adjacency matrix can be modeled as a Markov Chain.

Acknowledgment

The authors would like to thank Professor Chris De Sa for suggesting stochastic dynamic programming for the routing algorithm.

References

[1] Petter Holme. Network reachability of real-world contact sequences. *Phys. Rev. E*, 71:046119, Apr 2005. doi: 10.1103/PhysRevE.71.046119. URL https://link.aps.org/doi/10.1103/PhysRevE.71.046119.

[2] David Kempe, Jon Kleinberg, and Amit Kumar. Connectivity and inference problems for temporal networks. *Journal of Computer and System Sciences*, 64(4):820 – 842, 2002. ISSN 0022-0000. doi: https://doi.org/10.1006/jcss.2002.1829. URL http://www.sciencedirect.com/science/article/pii/S0022000002918295.

[3] Jari Saramäki and Petter Holme. Exploring temporal networks with greedy walks. *The European Physical Journal B*, 88(12):334, Dec 2015. ISSN 1434-6036. doi: 10.1140/epjb/e2015-60660-9. URL https://doi.org/10.1140/epjb/e2015-60660-9.

[4] Ingo Scholtes, Nicolas Wider, and Antonios Garas. Higher-order aggregate networks in the analysis of temporal networks: path structures and centralities. *The European Physical Journal B*, 89(3):61, Mar 2016. ISSN 1434-6036. doi: 10.1140/epjb/e2016-60663-0. URL https://doi.org/10.1140/epjb/e2016-60663-0.

[5] Eugenio Valdano, Chiara Poletto, and Vittoria Colizza. Infection propagator approach to compute epidemic thresholds on temporal networks: impact of immunity and of limited temporal resolution. *The European Physical Journal B*, 88(12):341, Dec 2015. ISSN 1434-6036. doi: 10.1140/epjb/e2015-60620-5. URL https://doi.org/10.1140/epjb/e2015-60620-5.

Appendix A
Proof

Lemma 1. *Given the policy: In any time slot i and node v, among all the available destinations ($\{v, x : (v, x) \in \mathcal{E}_i\} \cap (N(v) \cup \{v\})$), we select the node x with the smallest sum of edge cost and w value to be the node we will be at in time slot $(i+1)$. We claim that our algorithm calculates the expected value under this policy and we will not move to any of the nodes $\{x_{large} : w(x_{large}) + c_{vx_{large}} \geq w(v) + c_{vv}\}$ in the next time slot under this policy.*

Proof. First, we define

$N_{large}(v) = \{x_{large} : w(x_{large}) + c_{vx_{large}} \geq w(v) + c_{vv}, (v, x_{large}) \in \mathcal{E}\}$,

$N_{small}(v) = \{x_{small} : w(x_{small}) + c_{vx_{small}} < w(v) + c_{vv}, (v, x_{small}) \in \mathcal{E}\}$.

E_{small} := the set of events that at least one of the edges $\{(v, x_{small})\}$ is available.

E_{large} := the set of events that none of those edges is available.

$A = E_{small} \cup E_{large}$. Notice that P(A)=1 since the two events are the complement of each other.

Let $\mathcal{E}_a$ be the edges available in event $a \in A$.

$$\begin{aligned} w(v) &= \sum_{a \in A} P(a) * min_{(v,x) \in \mathcal{E}_a} w(x) + c_{vx} \\ &= \sum_{a \in E_{small}} P(a) * min_{(v,x) \in \mathcal{E}_a} c_{vx} + w(x) + \sum_{a \in E_{large}} P(a) * min_{(v,x) \in \mathcal{E}_a} c_{vx} + w(x) \end{aligned}$$

Claim. For each node $v \in V$, we have

$$\frac{\sum_{a \in E_{small}} P(a) * min_{(v,x)+c_{vx}) \in \mathcal{E}_a} w(x) + (1 - \sum_{a \in E_{small}} P(a)) * c_{vv}}{\sum_{a \in E_{small}} P(a)}$$

Note that according to Assumption 1, we have the inequality $w(x_{large}) + c_{vx_{large}} \geq w(v) + c_{vv}$ always holds. and under our policy we always go to the available node with the smallest sum of cost and w value. So according to the inequality above, when events in E_{large} happens, we always stay at the current node. Then we have:

$$\begin{aligned} w(v) &= \sum_{a \in E_{small}} P(a) * min_{(v,x) \in \mathcal{E}_a} (c_{vx} + w(x)) + \sum_{a \in E_{large}} P(a) * (c_{vv} + w(v)) \\ &= \sum_{a \in E_{small}} P(a) * min_{(v,x) \in \mathcal{E}_a} (c_{vx} + w(x)) + (1 - \sum_{a \in E_{small}} P(a)) * (c_{vv} + w(v)) \\ &= \sum_{a \in E_{small}} P(a) * (min_{(v,x)) \in \mathcal{E}_a} (w(x) + c_{vx}) + (1 - \sum_{a \in E_{small}} P(a)) * (w(v) + c_{vv}) \\ &= \frac{\sum_{a \in E_{small}} P(a) * min_{(v,x)+c_{vx}) \in \mathcal{E}_a} w(x) + (1 - \sum_{a \in E_{small}} P(a)) * c_{vv}}{\sum_{a \in E_{small}} P(a)} \end{aligned}$$

This is exactly the recurrence we used in our algorithm.

Since $w(x_{small}) + c_{vx_{small}} < w(v) + c_{vv} \leq w(x_{large}) + c_{vx_{large}}$, we know $min_{(v,x) \in \mathcal{E}_a} w(x) + c_{vx}) \neq w(x_{large}) + c_{vx_{large}}\ \forall x_{large}$.

So when an edge to x_{small} is available, we won't take any of x_{large}. And when none of the edges to x_{small} is available, we will take the self loop. Thus, our policy will only go to nodes that has has smaller minimum expected value. So under this policy, we will not move to any of the nodes $\{x_{large} : w(x_{large}) + c_{vx_{large}} \geq w(v) + c_{vv}\}$ in the next time slot. □

Lemma 2. *We say $w(v)$ correct if it is indeed the expected cost under the policy we described above. We claim our algorithm gives the $(i+1)^{th}$ smallest $w(v)$ to node v in i^{th} iteration(starting from iteration 0). And if we fail to calculate the correct w value for some node v, we must have overestimated it.*

Proof. By construction of our algorithm, the w values either decrease or stay the same at each iteration. By Lemma 1, the recurrence equation we used in the algorithm calculates the correct $w(v)$ value for node v at an iteration if all w(x), where $x : w(x) + c(v,x) < w(v) + c(v,v)$, have been calculated correctly at this iteration.

□

Lemma 3. *We say $w(v)$ correct if it is indeed the expected cost under the "going to smallest" policy we described above. We claim our algorithm gives the $(i+1)^{th}$ smallest $w(v)$ to node v in i^{th} iteration(starting from iteration 0). And if we fail to calculate the correct w value for some node v, we must have overestimated it.*

Proof. We prove the correctness of this Lemma by showing the statement: $P(i)$ = "The $(i+1)^{th}$ smallest $w(v)$ is given to node v in i^{th} iteration, where that value for a node x is:

$$w(x) = \frac{\sum_{a \in E_{small}} P(a) * min_{(v,x)+c_{vx}) \in \mathcal{E}_a} w(x) + (1 - \sum_{a \in E_{small}} P(a)) * c_{vv}}{\sum_{a \in E_{small}} P(a)}$$

. And if we fail to calculate the correct w value for some node v in i^{th} iteration, we must have overestimated it." is true for all $i = 0..n$. We prove this by strong induction.

Inductive Hypothesis:

$P(i)$ = "The $(i+1)^{th}$ smallest $w(v)$ is given to node v in i^{th} iteration. If we fail to calculate the correct w value for some node v in i^{th} iteration, we must have overestimated it." is true for all $i = 0..k$.

Base Case:

The base case when $k = 0$ is true because the 1^{st} smallest $w(v)$ is 0 where v is the destination node d. When we reached destination node, we don't need to pay the stalling cost anymore. In the initialization, we set $w(d) = 0$ explicitly. So in the 0^{th} iteration, we have given the 1^{st} smallest $w(v) = 0$ for $v = d$.

We initialize all other nodes' w values to ∞ in iteration 0. So we must have either (1) correctly initialized w when w is indeed ∞ or, (2) overestimated all the w values in iteration 0.

Case $i = k+1$:

Suppose we are at node v in the current iteration. We sort all nodes according to $w(v_i) + c(x, v_i)$, from the smallest to the largest and get the sequence $v_1, v_2, ..., v_n$.

By inductive hypothesis, we know that in k^{th} iteration, we have labeled nodes $v_1, v_2, ..., v_{k+1}$ correctly, where $w(v_1) \leq w(v_2) \leq ...w(v_k) \leq w(v_{k+1}) \leq w(v_{k+1})$. By Lemma 1, our routing algorithm will not move to any of the nodes $\{x : c(v_{k+1}, x) + w(x) \geq c(v_{k+1}, v_{k+1}) + w(v_{k+1})\}$. This is equivalent of saying the value $w(v_{k+1})$ only depends on the nodes $v_1, v_2, ..., v_k$ and indeed our algorithm only uses the w values of these nodes when calculating $w(v_{k+1})$. So the $(k+1)^{th}$ smallest $w(v)$ is calculated from the $1^{st}...(k)^{th}$ smallest w values, which by inductive hypothesis are correct at this iteration. As a result, the $w(v_{k+1})$ is correctly calculated at this iteration.

By inductive hypothesis, all w values that are not calculated correctly in previous iterations are over estimated. So when we calculate $w(x) = \frac{\sum_{a \in E_{small}} P(a) * min_{(v,x)+c_{vx}) \in \mathcal{E}_a} w(x) + (1 - \sum_{a \in E_{small}} P(a)) * c_{vv}}{\sum_{a \in E_{small}} P(a)}$, the w values we used are either correct or overestimated. Since the probabilities are non negative, we either correctly calculate $w(x)$ or also overestimated $w(x)$ in this iteration.

□

Lemma 4. *We won't change the label of a node if we have labeled it correctly.*

Proof. By line 7-9 in the algorithm, we only update the label of a node if the new label is smaller than the current label. As we showed in Lemma 3, we never underestimate the label of any node. So since every update we make decrease the label and we never underestimate the label, we never change the label of a node if we have labeled it correctly. □

Lemma 5. *Recall our routing policy: In some time slot i, say we are currently at node $v \neq t$, among all the available destinations ($\{v, x : (v, x) \in \mathcal{E}_i\} \cap (N(v) \cup \{v\})$), we select the node x with the smallest $w(x) + c_{vx}$ to be the node we will be at in time slot $(i + 1)$.*
Calculate the expected cost from each node to d under this policy gives $w(v)$ for all node $v \in V, v \neq d$: the smallest expected total cost from each node $v \in V$ to the destination node d. (Optimality of our routing algorithm.)

Proof. We prove this claim by strong induction. Say the true smallest expected total cost from node $v \in V$ to node d is $c_s(v)$. We sort all nodes in V according to $c_s(v)$, from the smallest to the largest and get the sequence $v_1, v_2, ..., v_n$, where we actually know $v_1 = d$. Then we just want to show: $w(v) = c_s(v)$ for all $v \in V$.
Base Case: We want to show $w(v_1) = c_s(v_1)$. This is true since we know $v_1 = d$ and we initialize $w(d) = 0$, which is indeed true since if we are already at node d there is no cost to get node d.
Inductive Step: Inductive Hypothesis: $w(v_j) = c_s(v_j)$ for $j = 1, ..., k$. Want to show: $w(v_j) = c_s(v_j)$ for $j = 1, ..., k + 1$.
We have $w(v_j) = c_s(v_j)$ for $j = 1, ..., k$ according to the inductive hypothesis. We are only left showing that $w(v_{k+1}) = c_s(v_{k+1})$.
According to Lemma 1, Lemma 3, when we are currently at node v_{k+1}, we will not go to node $v \in \{v_{k+2}, ..., v_n\}$ in the next time slot, and thus $(w(v_{k+1}))$doesn't depend on these nodes. The value of $w(v_{k+1})$ only depends on $w(v_1), ..., w(v_k)$. Without the loss of generality: Say in some time slot i, there are available destinations: $\{x_1, ..., x_q\} = \{v, x : (v, x) \in \mathcal{E}_i\} \cap (N(v) \cup \{v\})$ where $x_1, ..., x_q$ are sorted according to the $c_v x + w(x)$ value, from the smallest to the largest. Then our policy will choose to go to x_1.
We prove by contradiction that our policy gives the minimum expected cost.
Note that the expected cost from node v to node d $E[v] = \min_{x \in \{availabledestination\}} c(v, x) = c_s(x)$. Say if our policy does not give the minimum expected cost in this case. there exists some destination $x_l \neq x_1$, where $1 < l \leq q$ such that $c_{vx_l} + c_s(x_l) < c_{vx_1} + c_s(x_1)$. According to our inductive hypothesis we have $c_s(x_l) = w(x_l)$ and $c_s(x_1) = w(x_1)$. So we have $c_{vx_l} + w(x_l) < c_{vx_1} + w(x_1)$. But we know that $x_1, ..., x_q$ are sorted according to the $c_v x + w(x)$ value, from the smallest to the largest. So we have a contradiction.
Then since our policy gives the minimum expected cost for every node in every possible temporal graph G_i, and

we also know $w(v_j) = c_s(v_j)$ for $j = 1, ..., k$, we have $w(v_{k+1})$ and the true smallest expected cost form node v_{k+1} to node d is given by:

$$w(v_{k+1}) = c_s(v_{k+1}) = \frac{\sum_{x \in N(v), c_{vx} + < c_{vv} + w(v)} (c_{vx} + w(x))P(x) + (1 - \sum_{x \in N(v), c_{vx} + w(x) < c_{vv} + w(v)} P(x))c_{vv}}{\sum_{x \in N(v), c_{vx} + w(x) < c_{vv} + w(v)} P(x)}$$

. $\square$

Appendix B
Mean and Variance of Simulation

Note: the mean and variance are the mean and variance of delay rate, not the mean and variance of the square-rooted values.

Figure 3:

	Mean			Variance		
p	AW	TASP	OPT	AW	TASP	OPT
0.1	8.09	127	7.94	13.1	2350	10.6
0.2	3.82	19.8	3.36	4.31	224	10.6
0.3	2.17	10.2	1.92	1.29	62.7	3.07
0.4	1.54	8.02	1.36	0.767	46.1	1.01
0.5	1.02	5.81	0.89	0.893	43.4	0.352
0.6	0.647	4.08	0.580	0.191	22.1	0.160
0.7	0.432	2.33	0.406	0.109	5.09	0.0952
0.8	0.240	2.04	0.220	0.0570	4.58	0.0454
0.9	0.106	4.37	0.0976	0.0247	36.2	0.0193

Table II: Simulation results for 500 iterations (mean and variance for delay rate) for k-degree random graph: $n = 100, k = 3$, edge cost 1, seed = 5, p ranges from 0.1 to 0.9.

Figure 4:

	Mean			Variance		
p	AW	TASP	OPT	AW	TASP	OPT
0.1	7.39	105	7.39	47.4	11600	47.4
0.2	3.11	67.1	3.11	14.0	7810	14.0
0.3	1.51	19.9	1.51	2.96	784	2.96
0.4	0.841	8.27	0.841	1.11	317	1.11
0.5	0.596	5.68	0.596	0.619	197	0.619
0.6	0.363	1.78	0.363	0.311	33.9	0.311
0.7	0.240	0.654	0.240	0.180	9.62	0.180
0.8	0.158	0.295	0.158	0.137	1.73	0.137
0.9	0.0703	0.0847	0.0703	0.0564	0.101	0.0564

Table III: Simulation results for 500 iterations (mean and variance for delay rate) for k-degree random graph: $n = 100,\ k = 3$, cost ranges from 1 to 2, seed = 666, p ranges from 0.1 to 0.9.

Figure 7

	Mean			Variance		
p	AW	TASP	OPT	AW	TASP	OPT
0.1	9.04	3.30	3.024	26.9	12.8	7.32
0.2	3.47	1.68	1.624	5.35	2.23	1.87
0.3	2.34	1.02	1.00	2.53	0.632	0.586
0.4	1.51	0.785	0.810	1.09	0.337	0.329
0.5	1.02	0.661	0.692	0.698	0.208	0.361
0.6	0.647	0.587	0.523	0.321	0.166	0.237
0.7	0.421	0.483	0.347	0.189	0.0768	0.113
0.8	0.267	0.424	0.230	0.100	0.0460	0.0694
0.9	0.129	0.365	0.117	0.0468	0.0114	0.0364

Table IV: PTN Instance that AW policy performs badly

Figure 8

	Mean		Variance	
p	AW	OPT	AW	OPT
0.1	8.88	6.34	20.5	16.6
0.2	7.07	5.04	18.8	8.64
0.3	3.48	2.48	7.26	2.40
0.4	2.61	1.52	2.78	1.47
0.5	1.50	1.35	1.26	1.44
0.6	0.917	0.848	0.564	0.627
0.7	0.437	0.354	0.165	0.136
0.8	0.257	0.208	0.0809	0.0741
0.9	0.108	0.0823	0.0428	0.0233

Table V: Simulation results for 500 iterations (mean and variance for delay rate) for k-degree random graph: $n = 30, k = 3$, cost ranges from 1 to 5, seed = 94275, p ranges from 0.1 to 0.9.

Figure 5

	Mean		Variance	
p	AW	OPT	AW	OPT
0.1	7.21	6.72	8.31	8.18
0.2	2.77	2.53	1.25	1.01
0.3	1.71	1.35	0.537	0.372
0.4	0.878	0.836	0.209	0.182
0.5	0.597	0.573	0.107	0.0916
0.6	0.427	0.409	0.0716	0.0570
0.7	0.281	0.274	0.0427	0.0371
0.8	0.156	0.157	0.0182	0.0181
0.9	0.0692	0.0688	0.00727	0.00738

Table VI: Simulation results for 500 iterations (mean and variance for delay rate) for k-degree random graph: $n = 100$, $k = 2$, cost range from 1 to 2, seed = 5, p range from 0.1 to 0.9.