Chernoff Bounds

1 Tail Bounds

Given a random variable X, we are interested in finding strong bounds

- Lower tails: $\mathsf{Pr}[X < a]$
- Upper tails: $\mathsf{Pr}[X \geq a]$

1.1 Chernoff Bounds

Lemma 1.1 (Markov's Inequality) *Let X be a non-negative random variable with finite expectation μ. Then for any $a > 0$: $\mathsf{Pr}[X \geq a] \leq \mu/a$.*

Lemma 1.2 (Chebyshev's Inequality) *Let X be a random variable with finite expectation μ and standard deviation σ. Then for any $a > 0$: $\mathsf{Pr}[X \geq a\,\sigma] \leq 1/a^2$.*

Chebyshev's inequality follows from Markov's inequality for the variable $Y = (X - \mu)^2$ (so $\mathsf{E}[Y] = \mathsf{Var}[X]$) and the fact that $x \mapsto x^2$ is a strictly monotonic function on $\mathbb{R}_0^+$.

$$\begin{aligned}\mathsf{Pr}[|X - \mu| \geq a\,\sigma] &= \mathsf{Pr}[(X-\mu)^2 \geq a^2\sigma^2] \\ &\leq \frac{\mathsf{E}[(X-\mu)^2]}{a^2\sigma^2} = 1/a^2\end{aligned}$$

This makes it tempting to use even higher moments to get better bounds. One way to do this nicely is by considering an exponential of the basic form of e^x: the Taylor expansion of e^x is $\sum x^i/i!$ and contains all the powers x^i.

Let $X_1, X_2, \ldots, X_n$ be independent indicator variables and $X = \sum X_i$, so X is non-negative.

Bernoulli trials: $\mathsf{Pr}[X_i = 1] = p$, so $\mu = \mathsf{E}[X] = np$.

Poisson trials: $\mathsf{Pr}[X_i = 1] = p_i$, so $\mu = \mathsf{E}[X] = \sum p_i$.

Theorem 1.1 (Chernoff Bound)
Lower tail, $0 < \delta \leq 1$:

$$\mathsf{Pr}[X < (1-\delta)\mu] = \left(\frac{e^{-\delta}}{(1-\delta)^{1-\delta}}\right)^{\mu}$$

Upper tail, $0 < \delta$:

$$\mathsf{Pr}[X \geq (1+\delta)\mu] = \left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right)^{\mu}$$

Proof. We only prove the lower tail result, the argument for upper tails is entirely similar. First some auxiliary results.

Digression 1:

$$\mathsf{E}[e^{-tX_i}] = p_i \cdot e^{-t} + (1-p_i)\cdot 1 = 1 - p_i(1-e^{-t}) < e^{p_i(e^{-t}-1)}$$

The inequality uses $1 - z < e^{-z}$ with $z \mapsto p_i(e^{-t}-1)$.

Digression 2:

$$\prod \mathsf{E}[e^{-tX_i}] < \prod e^{p_i(e^{-t}-1)} = e^{\sum p_i(e^{-t}-1)} = e^{(e^{-t}-1)\mu}$$

Digression 3: $e^{-t} + t - \delta t - 1$ has a minimum at $t = -\ln(1-\delta)$.

Differentiation produces $-e^{-t} + 1 - \delta = 0$, so $t = -\ln(1-\delta)$.

2nd derivative: $e^{-t} > 0$.

Main argument: introduce a free parameter $t > 0$ and bind later.

$$\begin{aligned}
\mathsf{Pr}[X < (1-\delta)\mu] &= \mathsf{Pr}[e^{-tX} > e^{-t(1-\delta)\mu}] \\
&\leq \frac{\mathsf{E}[e^{-tX}]}{e^{-t(1-\delta)\mu}} \\
&= \frac{\mathsf{E}[e^{-t\sum X_i}]}{e^{-t(1-\delta)\mu}} \\
&= \frac{\mathsf{E}[\prod e^{-tX_i}]}{e^{-t(1-\delta)\mu}} \\
&= \frac{\prod \mathsf{E}[e^{-tX_i}]}{e^{-t(1-\delta)\mu}} \\
&< e^{(e^{-t}+t-\delta t-1)\mu}
\end{aligned}$$

Hence by claim 3:

$$\begin{aligned}
\text{LHS} &< e^{(e^{\ln(1-\delta)}-\ln(1-\delta)+\delta\ln(1-\delta)-1)\mu} \\
&= e^{(-\delta-(1-\delta)\ln(1-\delta))\mu} \\
&= \left(\frac{e^{-\delta}}{(1-\delta)^{1-\delta}}\right)^{\mu} = \text{RHS}
\end{aligned}$$

Done. □

As stated, the bounds are awkward to compute symbolically (numerically one would use a computer, anyhow). It often suffices to use weaker bounds that are easier to evaluate.

Lemma 1.3 (Simplified Chernoff Bounds)

Lower tail:

$$\mathsf{Pr}[X < (1-\delta)\mu] < e^{-\mu\delta^2/2} \qquad 0 < \delta \leq 1$$

Upper tail:

$$\mathsf{Pr}[X \geq (1+\delta)\mu] < e^{-\mu\delta^2/3} \qquad 0 < \delta \leq 1$$

$$\mathsf{Pr}[X \geq (1+\delta)\mu] < e^{-(1+\delta)\mu} \qquad 2e-1 < \delta$$

Proof.

For the lower tail let $D = (1-\delta)^{1-\delta}$. By Taylor expansion we get

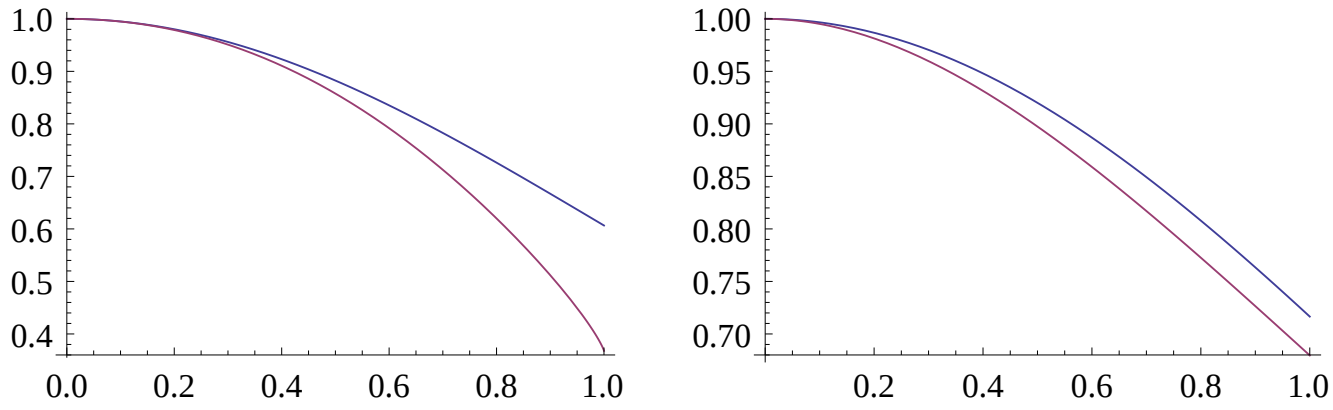
$$\begin{aligned} \ln D &= (1-\delta)\ln(1-\delta) \\ &= (1-\delta)(-\delta - \delta^2/2 - \delta^3/3 - \ldots) \\ &= -\delta + \delta^2/2 + \delta^3/6 + \ldots \\ &> -\delta + \delta^2/2 \end{aligned}$$

Hence

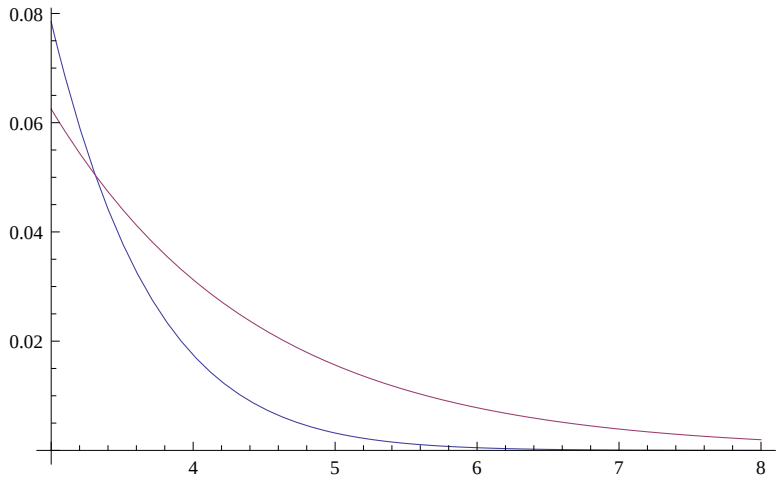
$$\left(\frac{e^{-\delta}}{(1-\delta)^{1-\delta}}\right)^{\mu} = \left(\frac{e^{-\delta}}{e^{\ln D}}\right)^{\mu} < \left(\frac{e^{-\delta}}{e^{-\delta+\delta^2/2}}\right)^{\mu} = e^{-\mu\delta^2/2}$$

The argument for the upper tail is left as an exercise. □

In all the comparisons below we are using $\mu = 1$. For $0 < \delta \leq 1$ the next picture shows lower tails on the left, upper tails on the right. Red is the "real" bound, blue the simplified one.



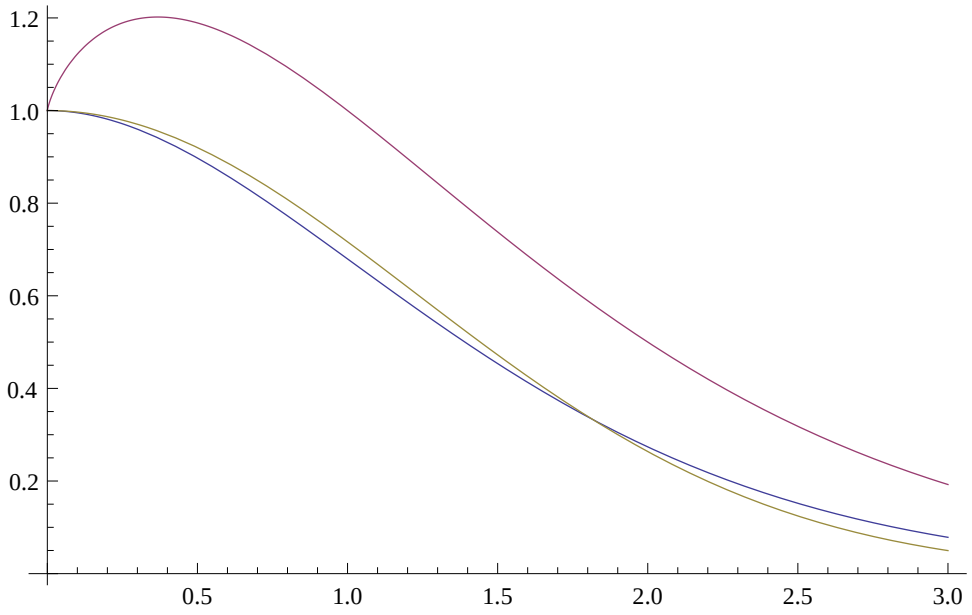
Here is the upper tail bound that is easily seen to be valid for $\delta > 2e-1 \approx 4.44$. Note that it actually holds for $\delta > 3.31$ (but solving the corresponding equation is quite hard).



If you are worried about the gap in admissible δ values between the two simplified bounds for upper tails, one can show that for $\delta < 2e - 1$ a simplified bound of $e^{-\mu\delta^2/5}$ works. Yet another useful simplification of the upper tail bound for $0 \leq \delta$ is

$$\mathsf{Pr}[X \geq (1+\delta)\mu] < e^{-\mu\delta \ln \delta/2}$$

As the following plot shows, this bound is mostly interesting for $\delta > 1.81696$ at which point the other simplification fails.



Lastly, note that one can combine the simplified upper and lower tails for $0 < \delta \leq 1$ to

$$\mathsf{Pr}[|X - \mu| \geq \delta\mu] < 2e^{-\mu\delta^2/3}$$

1.2 Examples

Comparing Strong and Weak Forms

Toss fair coin n times, X sum of results; let $m < n/2 = \mathsf{E}[X] = \mu$.

$$\mathsf{Pr}[X < m] = \mathsf{Pr}[X < (1-\delta)\mu]$$

where $\delta = 1 - 2m/n = 1 - m/\mu$.

Say $m = n/5$, so $\delta = 3/5$.

n	weak form	strong form
100	0.00012341	8.51×10^{-6}
1000	8.19×10^{-40}	1.99×10^{-51}
10000	1.36×10^{-391}	9.92×10^{-508}

Balls and Bins

As usual, we throw n balls into n bins, independently and uniformly at random. Let X be the random variable: number of balls in bin #1. Define indicator variables $X_i = 1$ iff ball i lands in bin #1. So $X = \sum X_i$, $\mathsf{Pr}[X_i = 1] = 1/n$ and $\mu = \mathsf{E}[X] = 1$. Also $\mathsf{Var}[X_i] = 1/n(1-1/n)$ and $\mathsf{Var}[X] = \sum \mathsf{Var}[X_i] = 1 - 1/n$.

Bounding $\mathsf{Pr}[X \geq 7]$

Markov

$$\mathsf{Pr}[X \geq 7] \leq 1/7 \approx 0.142857.$$

Chebyshev

$$\mathsf{Pr}[X \geq 7] = \mathsf{Pr}[|X-1| \geq 6] \leq \frac{1-1/n}{36} \leq 1/36 \approx 0.0277778$$

Chernoff

$$\mathsf{Pr}[X \geq 7] = \mathsf{Pr}[X \geq (1+6)\cdot 1] \leq 0.00048987$$

This is the result for the full bound; the simplified bound still produces 0.000911882.

Bounding $\mathsf{Pr}[X \geq 1 + 10\ln n]$

This is a more interesting situation.

Markov

$$\mathsf{Pr}[X \geq 1 + 10\ln n] \leq 1/(1+10\ln n)$$

Chebyshev

$$\mathsf{Pr}[X \geq 1 + 10\ln n] = \mathsf{Pr}[|X-1| \geq 10\ln n] \leq \frac{1-1/n}{10^2 \ln^2 n} \leq \frac{1}{100\ln^2 n}$$

Chernoff

$$\mathsf{Pr}[X \geq 1 + 10\ln n] = \mathsf{Pr}[X \geq (1+10\ln n)\cdot 1] \leq 2^{-(1+10\ln n)} \approx \frac{1}{2\,n^{6.931}}$$

using the simplified bound for the upper tail, $n \geq 2$. If we apply the second simplification involving ln we even get a bound of $1/n^{10}$.

1.3 Symmetric Variables

The Chernoff bounds we have seen so far only apply to independent Poisson trials, but a very similar approach carries over to independent symmetric variables with $\mathsf{Pr}[X_i = \pm 1] = 1/2$.

Lemma 1.4 *For all $a > 0$:*

$$\mathsf{Pr}[X \geq a] = \mathsf{Pr}[X \leq -a] \leq e^{-a^2/(2n)}.$$

Proof. We essentially repeat the argument from above: establish some auxiliary bounds, then apply Markov. First we can use the standard Taylor expansion of e^z to show

$$\mathsf{E}[e^{tX_i}] = 1/2 \cdot e^t + 1/2 \cdot e^{-t}$$

$$= \sum_{i \geq 0} \frac{t^{2i}}{(2i)!}$$

$$\leq \sum_{i \geq 0} \frac{(t^2/2)^i}{i!} = e^{t^2/2}$$

But then by independence

$$\mathsf{E}[e^{tX}] = \mathsf{E}[e^{t\sum X_i}] = \mathsf{E}[\prod e^{tX_i}] = \prod \mathsf{E}[e^{tX_i}] \leq e^{n\,t^2/2}$$

Applying Markov we get

$$\mathsf{Pr}[X \geq a] = \mathsf{Pr}[e^{tX} \geq e^{ta}] \leq \frac{\mathsf{E}[e^{tX}]}{e^{ta}} \leq e^{nt^2/2 - ta}$$

Now set $t = a/n$:

$$\mathsf{Pr}[X \geq a] \leq e^{-a^2/(2n)}$$

□

2 Application: Permutation Routing

2.1 Routing

Informally, the Routing Problem is the following. We are given n processors connected according to some topology. For each processor u we wish to send a message (packet) M_u to some other processor $\mathsf{D}(u)$, the *destination* of M_u. Each processor has a few immediate neighbors that it can send a packet to directly in one hop. To send a message to another processor several hops are required. Naturally, the routing process proceeds in rounds: at each round, a packet may move from its current location to an immediate neighbor. We assume that there is a global clock, all packets move synchronously.

Here is the critical constraint: in each round, at most one packet can move from a processor to single adjacent one. All others are placed into a local container and have to wait.

The obvious model for the topology is an undirected graph $G = \langle V, E \rangle$. Clearly, for each u we have to choose a path in G:

$$\mathsf{rt}(u) : u = u_0, u_1, \ldots, u_r = \mathsf{D}(u)$$

We will only consider a method that selects a *shortest path* though it is not hard to construct artificial networks where shortest paths cause poor performance. Write $dist(u)$ for the length of a shortest path from u to $\mathsf{D}(u)$. Let $\mathsf{T}(u)$ be the round when packet M_u arrives at $\mathsf{D}(u)$. The *delay* of packet M_u is $\mathsf{T}(u) - dist(u)$.

Our goal is to minimize the maximum of all the delays.

Intuitively, we have to avoid having many packets pile up at some processor. The details of the pileup depend on the queuing policy used. Here is a more fastidious explanation of the delay caused by colliding packets. For every vertex u we fix a route $\mathsf{rt}(u) = u_0, u_1, \ldots, u_d$ where $d = \mathsf{dist}(u)$ is the distance between u and $\mathsf{D}(u)$ in G. For every edge e we have a container C_e that stores packets wanting to traverse the edge (because e is on their chosen route).

- Initialization

  For all u pick a route $\mathsf{rt}(u)$.

  Place all M_u such that $\mathsf{D}(u) \neq u$ into C_e where e is the first edge on $\mathsf{rt}(u)$.

- Main Loop

  $r = 0$

  while(there is a packet not at its destination)

  in parallel, for all edges $e = (x, y)$ do

  if C_e is not empty, pick the highest priority packet in C_e

  in parallel, move all these packets to y

  if (a moved packet is not at its destination)

  place it in $C_{e'}$ according to the next edge on its route

  $r = r + 1$

Some restrictions:

Permutation Routing We assume that $\mathsf{D} : V \to V$ is a permutation. Clearly if all processors wanted to send to a single one there would be a problem.

Queuing Discipline We need to determine how to remove packets from the queue at each processor. For example, we could give preference to a packet that has furthest to go. Or we could use LIFO or FIFO plus a method to resolve ties when packets arrive at the same time.

Oblivious Routing Ideally all the paths would be planned by some central authority. Alas, this is computationally not feasible, routing has to be cheap. Hence we assume that the path for u depends only on $\mathsf{D}(u)$ — but not on any of the other paths.

Deterministic oblivious algorithms have serious limitations.

Theorem 2.1 *Suppose G has degree d. Any deterministic oblivious permutation routing scheme requires $\Omega(\sqrt{n/d})$ steps for some permutations.*

In any practical situation d is small compared to n, something like $\log n$, so this result represents a serious problem. We want performance closer to the diameter of G (diameter: length of the longest shortest path).

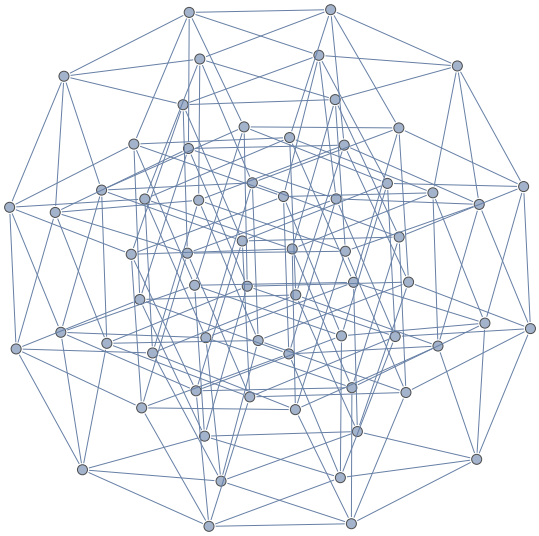
Two packets M and M' potentially collide with each other if their routes share at least one edge. Our goal is then to devise a randomized algorithm that does not produce too many actual collisions: the delay should be a small multiple of the diameter, with high probability.

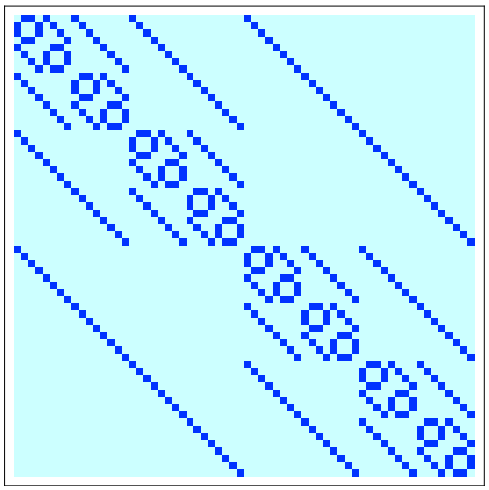
2.2 Hypercubes

Let's restrict the topology of the processor network to just one type: k-dimensional hypercubes where $V = \mathbf{2}^k$, so $n = 2^k$. Write $H(x, y)$ for the Hamming distance between x and y, so $H(x, y) = \#i : x_i \neq y_i$ and $0 \leq H(x, y) \leq k$. In a hypercube, there is an edge from x to y iff the Hamming distance of x and y is 1. Historical note: the Hillis Connection Machine from the 1980's had a hypercube architecture with $k = 16$.

In this type of network there is a natural routing algorithm for single packets.

Below is the 6-dimensional hypercube and its adjacency matrix.





Bit-Fixing

Suppose we have a source node x and a target node y at Hamming distance d. Note that there are $d!$ shortest paths from x to y: let $1 \leq e_1 < e_2 < \ldots < e_d \leq k$ be the positions where x and y differ. If we change the bits in x in position e_i, $i = 1, \ldots, d$, we obtain a shortest path. This is called (left-to-right) *bit-fixing*. All shortest path are obtained by permuting these d positions (so there are $d!$ shortest paths). From now on assume we determine routes by bit-fixing. Note that there is no need to actually compute and store a path: given a current node x and a target $y \neq x$ we can directly compute the next edge on the bit-fixing path.

Obviously, the hypercube has diameter k.

Proposition 2.1 (No-Diamonds) *Two paths constructed by bit-fixing can join and later separate; after that they can never join again.*

Of course, there may be collisions when we are trying to route multiple packets. For example, assume $k = 2m$ and let $\mathsf{D}(xy) = yx$ where $x, y \in \mathbf{2}^m$. Then $2^m = \sqrt{n}$ packets want to pass through y for all $y \in \mathbf{2}^m$.

Here is the randomized algorithm that we will analyze in the following.

Two-Phase Randomized Routing

Phase I Pick a random permutation π of n. Route M_u from u to $\pi(u)$ using bit-fixing.

Phase II Pick a random permutation π of n. Route M_u from $\pi(u)$ to $\mathsf{D}(u)$ using bit-fixing.

This may look pretty strange, but think about randomized incremental algorithms or randomized quicksort. Of course, the algorithm as stated is non-adaptive; in reality we would treat some special cases differently (e.g., when D is a transposition).

Our goal is to establish the following theorem.

Theorem 2.2 *The randomized two-phase routing method from above routes all packets in $O(k)$ steps, with probability $1 - O(1/n)$.*

First note that it suffices to consider phase I: the second phase is just another permutation routing problem, since it is equivalent to $u \mapsto \pi^{-1}(\mathsf{D}(u))$.

So, we may assume that D is chosen uniformly at random from all permutations of $\mathbf{2}^k$.

Lemma 2.1 (Delay Lemma) *Let $M = M_u$ be a packet using path P. Set S to be the set of all packets other than u that share an edge with P. Then the delay of M is at most $|S|$.*

Proof.

Proof idea: every packet has one dollar. Someone has to pay M one dollar for each delay. Need to make sure there is enough money.

To see why, let $P : e_0, e_1, \ldots, e_{d-1}$ be the route of packet $M = M_u$. Define the *lag* of a packet at round t at the source of edge e_i at the beginning of round t to be $t - i$. Clearly the lag of M is 0 initially and is its delay when e_{d-1} is traversed.

Now consider the situation when the lag of M increases from ℓ to $\ell + 1$ during round t. We need to find some packet responsible for this increase. Obviously, there must be some M' in S that traverses the edge M is waiting at, and M' must have lag ℓ at the beginning and end of round t.

Now let τ be the last round when some packet M' in S still has lag ℓ. By choice of τ, M' must leave P in round τ. Charge M' for the increase in lag of M. This works since, by the no-diamond proposition, every message in S leaves P at most once. □

We would like to use a Chernoff bound, so we need to find some independent Poisson trials somewhere. This requires a bit of thought, lots of random variables associated with this algorithm are not independent; the right approach is to consider

$$H_v = \begin{cases} 1 & \text{if the route of } v \text{ and } P \text{ share an edge,} \\ 0 & \text{otherwise.} \end{cases}$$

Note that the delay of M is bounded by $H = \sum H_v$, so we need to bound H. To this end introduce another random variable $R(e)$ for all edges e on P:

$$R(e) = \#\text{ routes using edge } e$$

Then $H \leq \sum_{e \text{on} P} R(e)$. Hence

$$\mathsf{E}[H] \leq \mathsf{E}[\sum H_v] \leq \mathsf{E}[\sum R(e)] \leq \sum \mathsf{E}[R(e)].$$

But note that a hypercube has strong symmetry properties: its automorphism group acts transitively on the edge set. Hence the distribution of $R(e)$ is the same as $R(e')$ for all e, e'.

The expected length of a route is $k/2$, so the total number of edges on all routes is expected to be $nk/2$. The total number of edges in a hypercube is nk, so we have $\mathsf{E}[R(e)] = 1/2$.

It follows that $\mathsf{E}[H] \leq k/2$.

Now apply the simplified Chernoff bound where $\delta > 2e - 1$:

$$\mathsf{Pr}[H \geq 6k] < 2^{-6k}$$

This is allowed since $6k = (\delta + 1)\mu$ implies $\delta > 11$.

But then the probability that any packet has a delay larger than $6k$ is at most $2^k \cdot 2^{-6k} = 2^{-5k}$. Hence with probability at least $1 - 2^{-5k}$ all packets arrive at their destination in at most $7k$ steps. □

So this randomized algorithm beats every deterministic one by an exponential amount for some inputs.

Note that the analysis applies to having two separate phases. It is tempting to "just keep going" when a packet M_u has reached its destination $\pi(u)$ during phase I and continue on to $D(u)$. I don't know what this modification would do to performance.