As this is a maximization problem, we need an upper bound of c^*, and there is an easy one:

$$c^* \leq m$$

where $m = |E|$.

The algorithm is: coloring every node independently with one of the three colors, each with probability $\frac{1}{3}$.

Let random variable

$$X_e = \begin{cases} 1 & \text{edge } e \text{ is satisfied} \\ 0 & \text{otherwise} \end{cases}$$

Then for any given edge e, there are 9 ways to color its two ends, each of which appears with the same probability, and 3 of them are not satisfying.

$$Exp[X_e] = Pr[e \text{ is satisfied}] = \frac{6}{9} = \frac{2}{3}$$

Let Y be the random variable denoting the number of satisfied edges, then by linearity of expectations,

$$Exp[Y] = Exp\left[\sum_{e \in E} X_e\right] = \sum_{e \in E} Exp[X_e] = \frac{2}{3}m \geq \frac{2}{3}c^*$$

[1] ex568.721.313

Number the voters $1, 2, \ldots, 100,000$, where voters 1 through 20000 are the Republican voters. Let X_i be the random variable equal to 0 if i votes for R, and 1 if i votes for D. So $X = \sum_{i=1}^{100000} X_i$.

Now, for $i \leq 20000$, $EX_i = .99 \cdot 0 + .01 \cdot 1 = .01$. For $i > 20000$, $EX_i = .01 \cdot 0 + .99 \cdot 1 = .99$. By linearity of expectation,

$$EX - \sum_{i=1}^{100000} EX_i - 20000 \cdot .01 + 80000 \cdot .99 - 79400.$$

[1] ex734.264.279

(a) Assume that using the described protocol, we get a set S that is not conflict free. Then there must be 2 processes P_i and P_j in the set S that both picked the value 1 and are going to want to share the same resource. But this contradicts the way our protocol was implemented, since we selected processes that picked the value 1 and whose set of conflicting processes all picked the value 0. Thus if P_i and P_j both picked the value 1, neither of them would be selected and so the resulting set S is conflict free. For each process P_i, the probability that it is selected depends on the fact that P_i picks the value 1 and all its d conflicting processes pick the value 0. Thus $P[P_i\text{selected}] = \frac{1}{2} * (\frac{1}{2})^d$. And since there are n processes that pick values independently, the expected size of the set S is $n * (\frac{1}{2})^{d+1}$

(b) Now a process P_i picks the value 1 with probability p and 0 with probability $1 - p$. So the probability that P_i is selected (i.e. P_i picks the value 1 and its d conflicting processes pick the value 0) is $p * (1 - p)^d$. Now we want to maximize the probability that a process is selected. Using calculus, we take the derivative of $p(1 - p)^d$ and set it equal to 0 to solve for the value of p that gives the objective it's maximum value. The derivative of $p(1 - p)^d$ is $(1 - p)^d - dp(1 - p)^{d-1}$. Solving for p, we get $p - \frac{1}{d+1}$. Thus the probability that a process is selected is $\frac{d^d}{(d+1)^{d+1}}$ and the expected size of the set S is $n * \frac{d^d}{(d+1)^{d+1}}$. Note that this is $\frac{n}{d}$ times $(1 - \frac{1}{d+1})^{d\,|\,1}$ and this later term is $\frac{1}{e}$ in the limit and so by changing the probability, we got a fraction of $\frac{n}{d}$ nodes. Note that with $p = 0.5$, we got an exponentially small subset in terms of d.

[1] ex131.386.529

(a) For every node v_k that comes later than v_j, i.e. $k > j$, it has probability $\frac{1}{k-1}$ to link to v_j, since v_k chooses from the $k-1$ existing nodes with equal probabilities. For all the nodes coming before v_j, such probability is obviously zero.

So the expected number of incoming links to node v_j is

$$\begin{aligned}\sum_{k=j+1}^{n} \frac{1}{k-1} &= \sum_{k=1}^{n-1} \frac{1}{k} - \sum_{k=1}^{j-1} \frac{1}{k} \\ &= H(n-1) - H(k-1) \\ &= \Theta(\ln n) - \Theta(\ln k) \\ &= \Theta(\ln \frac{n}{k})\end{aligned}$$

(b) Consider a node v_j, every node v_k with $k > j$ has probability $1 - \frac{1}{k-1}$ not to link to v_j. So if we have random variable X_j s.t.

$$X_j = \begin{cases} 1 & \text{node } v_j \text{ has no in-coming links} \\ 0 & \text{otherwise} \end{cases}$$

then

$$\begin{aligned}Exp[X_j] &= Pr[\text{no nodes links to } v_j] \\ &= \prod_{k=j+1}^{n} \left(1 - \frac{1}{k-1}\right) \\ &= \frac{j-1}{j} \cdot \frac{j}{j+1} \cdot \frac{j+1}{j+2} \cdots \frac{n-2}{n-1} \\ &= \frac{j-1}{n-1}\end{aligned}$$

Therefore, by linearity of expectations, we get the expected number of nodes without in-coming links

$$\sum_{j=1}^{n} Exp[X_j] = \sum_{j=1}^{n} \frac{j-1}{n-1} = \frac{1}{n-1} \sum_{j=1}^{n} (j-1) = \frac{1}{n-1} \cdot \frac{n(n-1)}{2} = \frac{n}{2}$$

[1] ex976.627.720

(a) Consider a clause C_i with n variables. The probability that the clause is not satisfied is $\frac{1}{2^m}$ and so the probability that it is satisfied is 1 less this quantity. The worst case is when C_i has just one variable, i.e. $n = 1$, in which case the probability of the clause being satisfied is $\frac{1}{2}$. Since there are k clauses, the expected number of clauses being satisfied is at least $\frac{k}{2}$. Consider the two clauses x_1 and $\overline{x_1}$. Clearly only one of these can be satisfied.

(b) For variables that occur in single variable clauses, let the probability of setting the variable so as to satisfy the clause be $p \geq \frac{1}{2}$. For all other variables, let the probabilities be $\frac{1}{2}$ as before. Now for a clause C_i with n variables, $n \geq 2$, the probability of satisfying it is at worst $(1 - \frac{1}{2^n}) \geq (1 - p^2)$ since $p \geq \frac{1}{2}$. Now to solve for p, we want to satisfy all clauses, so solve $p = 1 - p^2$ to get $p \approx 0.62$. And hence the expected number of satisfied clauses is $0.62n$.

(c) Let the total number of clauses be k. For each pair of single variable conflicting clauses, i.e. x_i and $\overline{x_i}$, remove one of them from the set of clauses. Assume we have removed m clauses. Then the maximum number of clauses we could satisfy is $k - m$. Now apply the algorithm described in the previous part of the problem to the $k - 2m$ clauses that had no conflict to begin with. The expected number of clauses we satisfy this way is $0.62 * (k - 2m)$. In addition to this we can also satisfy m of the $2m$ conflicting clauses and so we satisfy $0.62 * (k - 2m) + m \geq 0.62 * (k - m)$ clauses which is our desired target. Note that this algorithm is polynomial in the number of variables and clauses since we look at each clause once.

[1] ex633.413.669

We interpret the constraint (μ_i, μ_j, μ_k) to mean that we require one of the subsequences $\ldots, \mu_i, \ldots, \mu_j, \ldots, \mu_k, \ldots$ or $\ldots, \mu_k, \ldots, \mu_j, \ldots, \mu_i, \ldots$ to occur in the ordering of the markers. (One could also interpret it to mean that just the first of these subsequences occurs; this will affect the analysis below by a factor of 2.)

Suppose that we choose an order for the n markers uniformly at random. Let X_t denote the random variable whose value is 1 if the t^{th} constraint (μ_i, μ_j, μ_k) is satisfied, and 0 otherwise. The six possible subsequences of $\{\mu_i, \mu_j, \mu_k\}$ occur with equal probability, and two of them satisfy the constraint; thus $EX_t = \frac{1}{3}$. Hence if $X = \sum_t X_t$ gives the total number of constraints satisfied, we have $EX = \frac{1}{3}k$.

So if our random ordering satisfies a number of constraints that is at least the expectation, we have satisfied at least $\frac{1}{3}$ of all constraints, and hence at least $\frac{1}{3}$ of the maximum number of constraints that can be simultaneously satisfied.

We can extend this to construct an algorithm that *only* produces solutions within a factor of $\frac{1}{3}$ of optimal: We simply repeatedly generate random orderings until $\frac{1}{3}k$ of the constraints are satisfied. To bound the expected running time of this algorithm, we must give a lower bound on the probability p^+ that a single random ordering will satisfy at least the expected number of constraints; the expected running time will then be at most $1/p^+$ times the cost of a single iteration.

First note that k is at most n^3, and define $k' = \frac{1}{3}k$. Let k'' denote the greatest integer strictly less than k'. Let p_j denote the probability that we satisfy j of the constraints. Thus $p^+ - \sum_{j \geq k'} p_j$; we define $p^- - \sum_{j < k'} p_j - 1 - p^+$. Then we have

$$\begin{aligned} k' &= \sum_j j p_j \\ &= \sum_{j<k'} j p_j + \sum_{j \geq k'} j p_j \\ &\leq \sum_{j<k'} k'' p_j + \sum_{j \geq k'} n^3 p_j \\ &= k''(1 - p^+) + n^3 p^+ \end{aligned}$$

from which it follows that

$$(k'' + n^3)p^+ \geq k' - k'' \geq \frac{1}{3}.$$

Since $k'' \leq n^3$, we have $p^+ \geq \frac{1}{6n^3}$, and so we are done.

[1] ex449.507.100

First we give an algorithm that produces a subgraph whose expected number of edges has the desired value. For this, we simply choose k nodes uniformly at random from G. Now, for $i < j$, let X_{ij} be a random variable equal to 1 if there is an edge between our i^{th} and j^{th} node choices, and equal to 0 otherwise.

Of the $n(n-1)$ choices for i and j, there are $2m$ that yield an edge (since an edge (u, v) can be chosen either by picking u in position i and v in position j, or by picking v in position i and u in position j). Thus $E[X_{ij}] = \frac{2m}{n(n-1)}$.

The expected number of edges we get in total is

$$\sum_{i<j} E[X_{ij}] = \binom{k}{2} \cdot \frac{2m}{n(n-1)} = \frac{mk(k-1)}{n(n-1)}.$$

We now want to turn this into an algorithm with expected polynomial running time, which always produces a subgraph with at least this many edges. The analogous issue came up with MAX 3-SAT, and we use the same idea here: For this we use the same idea as in the analogous MAX 3-SAT: we run the above randomized algorithm repeatedly until it produces a subgraph with at least the desired number of edges.

Let p^+ be the probability that one iteration of this succeeds; our overall running time will be the (polynomial) time for one iteration, times $1/p^+$. First note that the maximum number of edges we can find is $e = \frac{k(k-1)}{2}$, and we're seeking $e' = e \cdot \frac{2m}{n(n-1)}$. Let e'' denote the greatest integer strictly less than e'. Let p_j denote the probability that we find a subgraph with exactly j edges. Thus $p^+ = \sum_{j>e'} p_j$; we define $p^- = \sum_{j<e'} p_j = 1 - p^+$. Then we have

$$\begin{aligned} e' &= \sum_j j p_j \\ &= \sum_{j<e'} j p_j + \sum_{j \geq e'} j p_j \\ &\leq \sum_{j<e'} e'' p_j + \sum_{j \geq e'} e p_j \\ &= e''(1-p^+) + \binom{k}{2} p^+ \end{aligned}$$

from which it follows that

$$(e'' + \binom{k}{2}) p^+ \geq e' - e'' \geq \frac{1}{n(n-1)}.$$

Since $e'' \leq \binom{k}{2}$, we have $p^+ \geq \frac{1}{k(k-1)n(n-1)}$, and so we are done.

[1] ex553.136.7

The strategy is as follows. The seller watches the first $n/2$ bids without accepting any of them. Let b^* be the highest bid among these. Then, in the final $n/2$ bids, the seller accepts any bid that is larger than b^*. (If there is no such bid, the seller simply accepts the final bid.)

Let b_i denote the highest bid, and b_j denote the second highest bid. Let S denote the underlying sample space, consisting of all permutations of the bids (since they can arrive in any order.) So $|S| = n!$. Let E denote the event that b_j occurs among the first $n/2$ bids, and b_i occurs among the final $n/2$ bids.

What is $|E|$? We can place b_j anywhere among the first $n/2$ bids ($n/2$ choices); then we can place b_i anywhere among the final $n/2$ bids ($n/2$ choices); and then we can order the remaining bids arbitrarily ($(n-2)!$ choices). Thus $|E| = \frac{1}{4}n^2(n-2)!$, and so

$$P[E] = \frac{n^2(n-2)!}{4n!} = \frac{n}{4(n-1)} \geq \frac{1}{4}.$$

Finally, if event E happens, then the strategy will accept the highest bid; so the highest bid is accepted with probability at least 1/4.

[1] ex437.89.251

Let X be a random variable equal to the number of times that b^* is updated. We write $X = X_1 + X_2 + \cdots + X_n$, where $X_i = 1$ if the i^{th} bid in order causes b^* to be updated, and $X_i = 0$ otherwise.

So $X_i = 1$ if and only if, focusing just on the sequence of the first i bids, the largest one comes at the end. But the largest value among the first i bids is equally likely to be anywhere, and hence $EX_i = 1/i$.

Alternately, the number of permutations in which the number at position i is larger than any of the numbers before it can be computed as follows. We can choose the first i numbers in $\binom{n}{i}$ ways, put the largest in position i, order the remainder in $(i-1)!$ ways, and order the subsequent $(n-i)$ numbers in $(n-i)!$ ways. Multiplying this together, we have $\binom{n}{i}(i-1)!(n-i)! = n!/i$. Dividing by $n!$, we get $EX_i = 1/i$.

Now, by linearity of expectation, we have $EX = \sum_{i=1}^{n} EX_i = \sum_{i=1}^{n} 1/i = H_n = \Theta(\log n)$.

[1] ex547.67.324

(a) Let's look at a given machine p. In order for it to have no job, every job must be assigned to a different machine. As the jobs are assigned randomly and uniformly, the probability that a given job j is not assigned to p is $(1 - \frac{1}{k})$ and therefore the probability that p doesn't get any job is $(1 - \frac{1}{k})^k$. Therefore the expected number of machines with no jobs is $N(k) = k(1 - \frac{1}{k})^k$.

Finally $N(k)/k = (1 - \frac{1}{k})^k$, which goes to $1/e$ as k goes to infinity. Also notice that in the limit the number of machines with no jobs is k/e.

(b) There is a very simple solution to this problem. We notice that the number of rejected jobs (denote it by N_{rej}) is the number of total jobs k minus the number of accepted jobs N_{acc} ($N_{rej} = k - N_{acc}$). The number of jobs accepted is the k minus the number of machines with no jobs N_{nojob} (since the rest of the people do exactly 1 job). Therefore $N_{rej} = k - N_{acc} = k - (k - N_{nojob}) = N_{nojob}$. Therefore the answer to part **(b)** is the same as the answer to part **(a)**.

(c) This part will involve slight calculations. We know that the number of machines with no jobs is k/e (from the first part). We first calculate the number of machines with exactly one job. Again look at a machine p. The probability that only 1 job is assigned to that machine is $k\frac{1}{k}(1 - \frac{1}{k})^{k-1}$. (The chance of a given job j being assigned to p is $1/k$ and the probability that the remaining jobs will not be assigned to p is $(1 - \frac{1}{k})^{k-1}$. Finally there are k choices of the "given" job j which puts the coefficient k in the beginning). Notice that this also in the limit $1/e$ therefore the number of machines with exactly 1 jobs is also k/e.

Finally the remaining machines regardless of how many jobs they were assigned will perform exactly two jobs. There are $k - \frac{2k}{e}$ of these.

The final tally is k/e machines with one job and $k - \frac{2k}{e}$ people with two jobs. Subtracting this from k (the total number of jobs) we get that $\frac{k(3-e)}{e}$ jobs are rejected, which is approximately 11%.

[1] ex16.34.694

Consider a graph G with nodes s and t, and $n-2$ other nodes $v_1, \ldots, v_{n-2}$. There are two parallel edges from s to each v_i, and one edge from v_i to t. The minimum s-t cut is to separate t by itself.

If we run the version of the contraction algorithm described in the problem, it will independently contract each of the length-2 paths from s to t in some order. In order for it to find the minimum s-t cut, it must contract each v_i into s, not into t. There is a $2/3$ chance of this happening for each i, so the probability that the minimum s-t cut is found is $(2/3)^{n-2}$, an exponentially small quantity.

(Note that this example poses no problem for the global minimum cut, which consists of any of the nodes v_i on its own.)

[1] ex242.186.32

The mean for X_2 is n, so in order to have $X_1 - X_2 > c\sqrt{n}$, we need $X_2 < E[X_2] - \frac{c}{2}\sqrt{n} = (1-\delta)E[X_2]$ for $\delta = \frac{c}{2\sqrt{n}}$. Plugging this into the Chernoff lower bound, the probability this happens is

$$e_{-\frac{1}{2}\delta^2 E[X_2]} = e^{-c^2/4}.$$

This can be made smaller than a constant ε by choosing the undetermined constant c large enough.

[1] ex646.944.578

(a) Let n be odd, $k = n^2$, and represent the set of basic processes as the disjoint union of n sets $X_1, \ldots, X_n$ of cardinality n each. The set of processes P_i associated with job J_i will be equal to $X_i \cup X_{i+1}$, addition taken modulo n.

We claim there is no perfectly balanced assignment of processes to machines. For suppose there were, and let Δ_i denote the number of processes in X_i assigned to machine M_1 minus the number of processes in X_i assigned to machine M_2. By the perfect balance property, we have $\Delta_{i+1} = -\Delta_i$ for each i; applying these equalities transitively, we obtain $\Delta_i = -\Delta_i$, and hence $\Delta_i = 0$, for each i. But this is not possible since n is odd.

(b) Consider independently assigning each process i a *label* L_i equal to either 0 or 1, chosen uniformly at random. Thus we may view the label L_i as a 0-1 random variable. Now for any job J_i, we assign each process in P_i to machine M_1 if its label is 0, and machine M_2 if its label is 1.

Consider the event E_i, that more than $\frac{4}{3}n$ of the processes associated with J_i end up on the same machine. The assignment will be nearly balanced if none of the E_i happen. E_i is precisely the event that $\sum_{t \in J_i} L_t$ either exceeds $\frac{4}{3}$ times its mean (equal to n), or that it falls below $\frac{2}{3}$ times its mean. Thus, we may upper-bound the probability of E_i as follows.

$$\begin{aligned}
\Pr[E_i] &\leq \Pr[\sum_{t \in J_i} L_t < \frac{2}{3}n] + \Pr[\sum_{t \in J_i} L_t > \frac{4}{3}n] \\
&\leq \left(e^{-\frac{1}{2}(\frac{1}{3})^2}\right)^n + \left(\frac{e^{\frac{1}{3}}}{\left(\frac{4}{3}\right)^{\frac{4}{3}}}\right)^n \\
&\leq 2 \cdot .96^n.
\end{aligned}$$

Thus, by the union bound, the probability that any of the events E_i happens is at most $2n \cdot .96^n$, which is at most .06 for $n \geq 200$.

Thus, our randomized algorithm is as follows. We perform a random allocation of each process to a machine as above, check if the resulting assignment is perfectly balanced, and repeat this process if it isn't. Each iteration takes polynomial time, and the expected number of iterations is simply the expected waiting time for an event of probability $1 - .06 - .94$, which is $1/.94 < 2$. Thus the expected running time is polynomial.

This analysis also proves the *existence* of a nearly balanced allocation for any set of jobs.

(Note that the algorithm can run forever, with probability 0. This doesn't cause a problem for the expectation, but we can deterministically guarantee termination without hurting the running time very much as follows. We first run k iterations of the randomized algorithm; if it still hasn't halted, we now find the nearly balanced assignment that is guaranteed to exist by trying all 2^k possible allocations of processes to machines, in time $O(n^2 \cdot 2^k)$. Since this brute-force step occurs with probability at most $.06^k$, it adds at most $O(n^2 \cdot .12^k) = O(n^2 \cdot .12^n) = o(1)$ to the expected running time.)

[1] ex41.971.873

We imagine dividing the set S into 20 *quantiles* $Q_1, \ldots, Q_{20}$, where Q_i consists of all elements that have at least $.05(i-1)n$ elements less than them, and at least $.05(20-i)n$ elements greater than them. Choosing the sample S' is like throwing a set of numbers at random into bins labeled with $Q_1, \ldots, Q_{20}$.

Suppose we choose $|S'| = 40,000$ and sample with replacement. Consider the event $\mathcal{E}$ that $|S' \cap Q_i|$ is between 1800 and 2200 for each i. If $\mathcal{E}$ occurs, then the first nine quantiles contain at most $19,800$ elements of S', and the last nine quantiles do as well. Hence the median of S' will belong to $Q_{10} \cup Q_{11}$, and thus will be a (.05)-approximate median of S.

The probability that a given Q_i contains more than 2200 elements can be computed using the Chernoff bound (4.1), with $\mu = 2000$ and $\delta = .1$; it is less than

$$\left[\frac{e^{.05}}{(1.05)^{(1.05)}}\right]^{10000} < .0001.$$

The probability that a given Q_i contains fewer than 1800 elements can be computed using the Chernoff bound (4.2), with $\mu = 2000$ and $\delta = .1$; it is less than

$$e^{-(.5)(.1)(.1)2000} < .0001.$$

Applying the Union Bound over the 20 choices of i, the probability that $\mathcal{E}$ does not occur is at most $(40)(.0001) = .004 < .01$.

[1] ex835.763.619

One algorithm is the following.

```
For i = 1, 2, ..., n
  Receiver j computes β_ij = f(β*_1 ··· β*_{i-1}, α_i^⟨j⟩).
  β*_i is set to the majority value of β_ij, for j = 1, ..., k.
  End for
Output β*
```

We'll make sure to choose an odd value of k to prevent ties.

Let $X_{ij} = 1$ if $\alpha_i^{\langle j \rangle}$ was corrupted, and 0 otherwise. If a majority of the bits in $\{\alpha_i^{\langle j \rangle} : j = 1, 2, \ldots, k\}$ are corrupted, then $X_i = \sum_j X_{ij} > k/2$. Now, since each bit is corrupted with probability $\frac{1}{4}$, $\mu = \sum_j EX_{ij} = k/4$. Thus, by the Chernoff bound, we have

$$\begin{aligned} \Pr[X_i > k/2] &- \Pr[X_i > 2\mu] \\ &< \left(\frac{e}{4}\right)^{k/4} \\ &\leq (.91)^k. \end{aligned}$$

Now, if

$$k \geq 11 \ln n > \frac{\ln n - \ln .1}{\ln(1/.91)},$$

then

$$\Pr[X_i > k/2] < .1/n.$$

(So it is enough to choose k to be the smallest odd integer greater than $11 \ln n$.) Thus, by the union bound, the probability that *any* of the sets $\{\alpha_i^{\langle j \rangle} : j = 1, 2, \ldots, k\}$ have a majority of corruptions is at most .1.

Assuming that a majority of the bits in each of these sets are not corrupted, which happens with probability at least .9, one can prove by induction on i that all the bits in the reconstructed message β^* will be correct.

[1] ex482.918.336

Let Y denote the number of steps in which your net profit is positive. Then $Y = Y_1 + Y_2 + \cdots + Y_n$, where $Y_k = 1$ if your net profit is positive at step k, and 0 otherwise.

Now, consider a particular step k. $Y_k = 1$ if and only if you have had more than $k/2$ steps in which your profit increased. Since the expected number of steps in which your profit increased is $k/3$, we can apply the Chernoff bound (4.1) with $\mu = k/3$ and $1 + \delta = 3/2$ to conclude that EY_k is bounded by

$$\left[\frac{e^{1/2}}{(3/2)^{(3/2)}}\right]^{(k/3)} < (.97)^k.$$

Thus,

$$EY = \sum_{k=1}^{n} EY_k < \sum_{k=1}^{n} (.97)^k < \frac{1}{1-(.97)} < 34,$$

which is a constant independent of n.

[1] ex251.139.906

(a) False. A bad example can consist of a single edge $e = (u, v)$. Assume the cost of u is 1 while the cost of v is more than $2c$. The minimum cost of a vertex cover is 1, while the algorithm selects node v with probability $1/2$, and hence has expected cost more than c. Alternately we could have u at most 0 and v at most 1. Now the algorithm's expected cost is $1/2$, while the optimum is 0.

(b) This is true. Let p_e be the probability that edge e is selected by the algorithm. Note that the algorithm, as given by the problem set, does not specify the selection rule of edges. You may select uncovered edges at random, or by smallest index, etc. The probability p_e will of course depend on what selection rule was used. But any selection rule gives rise to such probabilities. Now we need to notice two facts. First that $\sum_{e \in E} p_e$ is exactly, the expected number of nodes selected by the algorithm. This is true, as every time we select an edge e we add one node to the vertex cover.

Next we consider the sum of the probabilities p_e for edges adjacent to a vertex v. Let $\delta(v)$ denote the set of edges adjacent to vertex v, and consider $\sum_{e \in \delta(v)} p_e$. Note that this is exactly the expected number of edges selected that are adjacent to node v. Let $S(v)$ be the random variable indicating the selected edges adjacent to v. We have that $Exp(|S(v)|) = \sum_{e \in \delta(v)} p_e$. We claim that this expectation is at most 2. This is true as each time an edge in $\delta(v)$ is selected, with $1/2$ probability, we use node v cover edge e, and then all edges in $\delta(v)$ are covered, and no more edges in this set will be selected selected. To make this argument precise, let E_i denote the event that at least i edges are selected adjacent to i. Now we have the following inequality for the expected number of edges selected.

$$Exp(|S(v)|) - \sum_i iProb(E_i - E_{i+1}) - \sum_i Prob(E_i) \leq 1 + \sum_{i>1} 2^{i-1} \leq 2,$$

where the inequality $Prob(E_i) \leq 2^{i-1}$ follows for $I > 1$ as after each edge selected adjacent to v we add v to the vertex cover with probability $1/2$.

Now we are ready to bound the expected size of the vertex cover compared to the optimum. Let S^* be an optimum vertex cover.

$$\sum_e p_e \leq \sum_{v \in S^*} \sum_{e \in \delta(v)} p_e \leq \sum_{v \in S^*} 2 = 2|S^*|,$$

where the first inequality follows as S^* is a vertex cover, and so the second sum must cover each edge e.

[1] ex593.991.129