Answers to Homework #2, CS 239, Section 1, Spring 2007

1.

a). (6 points) A purely probabilistic model generates the different operations randomly with the probabilities observed in the generating data. That is,

Operation	Probability
Metadata	11.4%
Small File	57.1%
Seq. Large File	25.7%
Non-Seq. Large File	5.7%

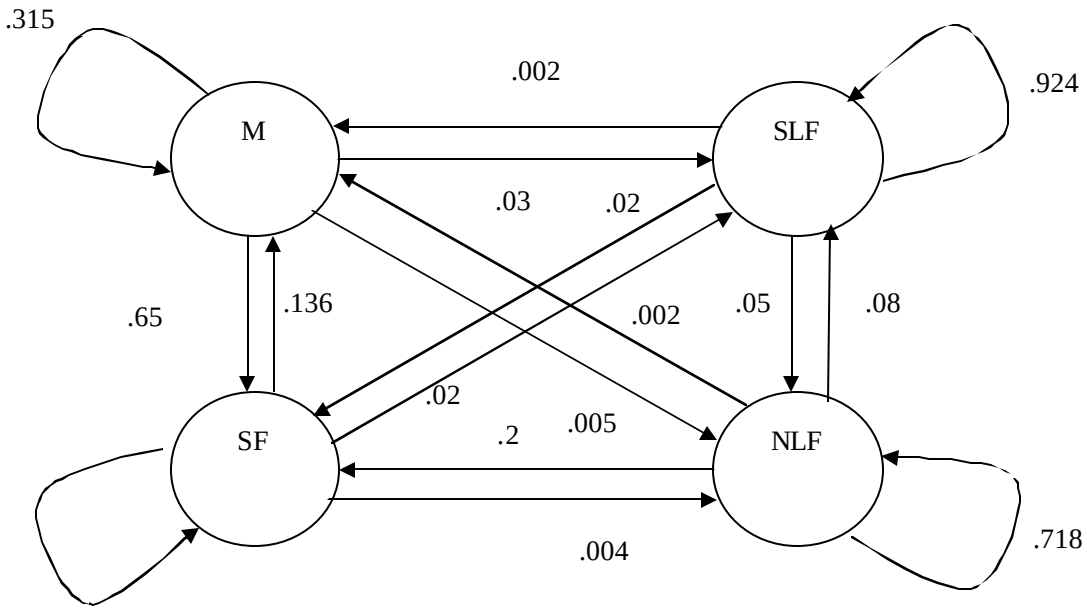
b). (10 points) To build a Markov model, one first calculates the probabilities of each operation following the others, from the generating data. First, the raw numbers of operations and the sums of them:

	M	SF	SLF	NLF	sum	
M	31500	65000	3000	500	100000	
SF	68000	419000	10000	2000	499000	
SLF	400	5000	208000	11600	225000	
NLF	99	10000	4000	35900	49999	
					873999	Total operations

Next, the probabilities:

	M	SF	SLF	NLF
M	0.315	0.65	0.03	0.005
SF	0.136273	0.839679	0.02004	0.004008
SLF	0.001778	0.022222	0.924444	0.051556
NLF	0.00198	0.200004	0.080002	0.718014

The model is most commonly expressed not as this kind of table, but as a diagram, as below:



c. (10 points) It's not entirely clear how one would handle sequential large file accesses without some kind of history-driven model. Even with one, there is the issue of when one stops accessing one large file and starts accessing another. A reasonable choice would be to assume that a metadata access after a large file access indicates an open of a different large file, but there are other possibilities. It might be nice to know what each kind of metadata access really is: an open, a close, an attribute lookup, etc.

That issue aside, a Markov model is likely to be a better choice because of the issue of sequential accesses to large files. As the model shows, the vast majority of the time, the next operation after a sequential access to a large file is another such access. Since Conquest's large file system is optimized for just such behavior, and presumably will achieve high disk bandwidth and good speed when one such access is instantly followed by a string of others, a model that blindly generates one access in four to such a file will have a very different performance than the Markov model. Order probably matters less for the other forms of access. Unless one is carefully modeling which bytes of which small files and metadata are accessed, one won't capture L2 caching effects (if any) for small file and metadata accesses based on operation order. But given you're using a history-driven method for at least the large sequential file accesses, it makes sense to use Markov methods uniformly.

2. There is not necessarily a uniquely right answer here. Answers that offer a good argument for their choices, even if they don't match the below answers, might receive full or partial credit.

A. Probably a generator is the best choice here. We might expect to see varying behavior from user to user, and being able to tune the workload easily to test different possibilities would be helpful. On the other hand, the entire set of users, likely enough,

uses some fairly common set of tools for their most common operations, so creating a generator to match those few tools would be reasonable. The generator should create the kinds of operations common to the salesmen's normal activities, which are likely to include at least process running, file system activity, and network activity.

B. There's a lot to be said for a trace here. However, the rapidly growing element would make one concerned that the trace of today's activity might not match tomorrow's. If the expectation is merely more users doing the same kind of operations, the trace might be scalable pretty easily. If not, a generator might be necessary. A benchmark is not as good a choice, since the company already knows at least its current load, and this is precisely the sort of thing they care about. A generic benchmark is less likely to behave like their environment. The workload should consist of web requests, probably organized into typical web sessions (which happens automatically with a trace, but must be coded in for a generator).

C. A generator is the best way to go here. There is no benchmark of this sort (and it's not clear there should or could be), traces might not scale for different types of worms, and live testing is right out. The operations the generator should produce are connection creations, either normal or modeling known or postulated worm behavior.

D. Assuming that the issue is handling the kind of traffic experienced today, a trace is a good solution here. Since it's the company's own internal traffic, a trace can easily be gathered. Privacy issues are not likely to be a problem, but, on the other hand, chances are that the router will be neutral to any packet contents beyond the IP header fields, so a trace with all packet contents zeroed out after the header will probably be sufficient.

E. A benchmark sounds good for this one, assuming one can find a benchmark that has characteristics similar to the type of traffic to be carried. Benchmarks allow easy head-to-head comparisons of different system options, with lower costs than generators usually require. The benchmark would need to be able to generate suitably sized file transfers using TCP.

F. Benchmarks that the community agrees are acceptable aren't available for phishing. Generation really isn't suitable, since the performance question hinges on distinguishing phishing sites from non-phishing sites. A trace is a possibility, where here a trace would actually represent a set of web pages that are known to be legitimate and another set known to be phishing sites. While there is something to be said for running the system with live users as test subjects, since that would help determine the efficacy of the color-coding in alerting users, we would not want to run normal live traffic for the test, since we would probably be unable to determine whether visited sites really are legitimate or phishing sites. The workload would consist of visits to a number of web sites, some of which are phishing sites and some of which are legitimate.

G. The workload here is disk contents. Assuming we wish to check that the program can find viruses and determine its performance on infected files, we would need to have some real viruses infecting files on the disk. In live testing, we'd get no assurance of such files being present, and no control over how widely infected the disk is. Traces are not likely to be readily available, and benchmarks don't really exist for this. The best solution is a generator that creates a whole disk's contents with a controllable

amount and character of virus infection. In more detail, the workload is a file system containing some files that should be identified as infected and other files that should not.

H. Generally speaking, traces of emails containing viruses aren't available. Live workloads are very likely to eventually produce some emails that contain a virus, but it isn't very controllable. Benchmarks aren't agreed upon. That leaves a generator. It should create email messages, some of which contain viruses (perhaps in a crippled but recognizable form, since otherwise they are dangerous) and some of which don't.

I. This one is tricky, since AES is used potentially for many purposes, from bulk encryption of large quantities of data to encrypting a trickle of data across a network. Performance for the rival algorithm could differ for the various cases. If one is evaluating the rival for one's own purposes, the workload should match what you are likely to do with it. Using samples of real work of the kind you do would be sensible. One might call that a trace or a benchmark or perhaps even a generator. If one is interested in making wider ranging statements on the comparative performance under many circumstances, the workload must capture all of them. A generator is better for the latter case. On the other hand, cryptographic algorithm performance is widely studied, and, depending on exactly what one wants to learn, one can perhaps find suitable benchmarks.

J. "Reasonable performance" on a desktop operating system can be hard to quantify, since it more or less boils down to user satisfaction. There would be something to be said for live testing here. Choose a subset of the company and switch them to Vista as a pilot program. Both measure the observed performance and get the users' feedback. If this approach is not feasible, running benchmarks is probably the next best choice. Creating a generator that matches the workload will be difficult, and gathering sufficiently detailed traces hardly less so. The workload will consist either of the real work of the pilot users or of whatever the benchmarks do.

3. There is not necessarily a uniquely right answer here. Answers that offer a good argument for their choices, even if they don't match the below answers, might receive full or partial credit.

A. The most important parameter is the size of the data flow we are trying to move. We could also consider other characteristics as parameters, such as whether the data flow is constant or varying, but size alone is probably the place to start. A histogram is probably the best choice here. We would like to test some representative flow sizes from the entire range of likely flows, not just those at the mean or relatively close to it. If the multipath algorithm allows adjustments based on number of paths used, testing that as a parameter is also worthwhile. In such a case, chances are the maximum number of allowable or feasible paths isn't too large, in which case one should test them all. The choice of source and destination nodes is another possible parameter, to allow us to investigate different path lengths and possibility of multiple paths. The question's choices for parameter variation don't make much sense for this parameter, since, effectively, this is a categorical parameter. Selecting a set of choices that exercise different parts of the parameter space is perhaps the best way to characterize this parameter.

B. We clearly need to vary the number of hash lookups to be performed, and the number of hash table creations. Both can be reasonably characterized by an average and

a variation from that average. Depending on what we're investigating, varying the number of nodes in the DHT might also make sense. This is not, however, really a workload parameter, though it certainly is an experimental factor to be considered in experiment design.

C. To test this code properly, we need to offer workloads characteristic of both network traffic and file system traffic. The network traffic should probably have parameters that control its direction (send or receive), the number of requests per unit time, and the size of each request. The file system traffic should consist of typical file operations (read, write, create, remove, access metadata), so one parameter should be the mix of such operations, another should be the time between operations, and a third should be size. The network traffic parameter for direction need merely be the average percentage of the types of operation. The requests per unit time and the size can be controlled reasonably by average and variation. For file system parameters, the mix can be represented as an average. Very large file operations are not that uncommon and are likely to put particular stress on a buffer system, so size might be best handled by histogram methods. Timing can reasonably be handled by an average and deviation.

D.

i). The algorithm will use some set of inputs, probably based on timing, to make its decision on spinning down the disk. One could test the battery power savings by generating such events, using a parameter to control the timing of each type of input (file system read/write, mouse click, key press, packet arrival, whatever). An average and variation will capture this well enough.

ii). Different applications are likely to have different tolerances to the disk having to be spun up, particularly if the inputs used are beyond merely observing file system activities. We would need a deeper understanding of the program mix and behavior to provide a proper workload for this purpose. Probably the easiest workload for this purpose would be to run characteristic applications themselves, rather than generating events. The most likely parameters to vary would control the mix of applications to be run and the characteristics of those applications (such as run time, number of remote sites communicated with, number of files written, etc.)

iii). One could use the workload and parameters for testing performance effects to also test battery power savings. If one was interested in measuring both effects, much could be learned merely by using that workload and varying its parameters. However, being able to characterize the battery savings under extremes of conditions might be helpful, and it could be hard to create those conditions by playing with application control parameters. So to the extent that understanding the full range of power implications is important, one should test that workload and the extremes of its parameters. This workload, however, will tell one little about the user experience of using such a machine.

4. 1). Breadth of coverage is vital here. Your article will be read by a wide variety of people whose systems behave in many different ways. You need a workload that will allow you to easily test situations that correspond to many different real world systems. You might, for example, decide to characterize the most common workloads as belonging to N categories and create a separate workload for each. Or you might try to argue that you can generalize the workload for all environments by controlling a few important

parameters, such as number of remote hosts contacted or types of applications used. High applicability is of more importance than accurate representation of any single reality.

2). You care if it works well in your system, not in someone else's. A workload representing an actual trace of your office's activity would be a reasonable candidate, if it were readily available. If it isn't, a benchmark or generator that you believe accurately captures the characteristics of your system is what you want. Parameters would be varied to represent how conditions vary in your office, or to capture likely changes that might happen in the near future, not to describe very different situations that occur elsewhere, but not in your environment. Accurate representation of your reality trumps generality in your workload characterization, here.