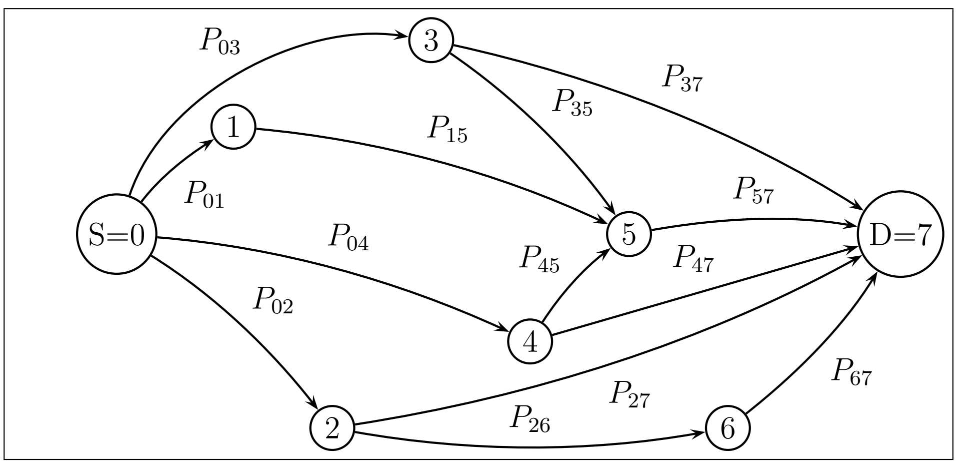
Lecture 8 — September 26

Lecturer: John Byers *BOSTON UNIVERSITY* *Scribe: Kevin Donnelly*

In this lecture we first wrapped up discussion of the MORE paper, which combines network coding with opportunistic routing, and then started to learn about **Bloom filters**, which are compact approximate representations of sets.

8.1 MORE wrap-up

Recall that between a sender S and destination D, there may be several paths that can be taken. It is assumed that between any two nodes, i and j, there is a fixed probability, P_{ij}, that a transmission from node i will be successfully received by node j. The value $\epsilon_{ij} = 1 - P_{ij}$ is the probability that a packet sent by i will not be successfully received by j. The figure below shows an example of a set of paths from S to D, the connections shown are the ones with non-zero success probability.



Recall that the ETX distance from i to j is the expected number of total transmissions (including transmissions by forwarding nodes) required to send a packet from i to j. In the MORE protocol, the delivery of information from the source to the destination uses opportunistic routing, so there is no well-defined next-hop for any particular packet; any router closer to the destination is a potential next-hop. The protocol attemps to continue retransmitting the packet until it is successfully received by *any* forwarder with a smaller ETX distance from the destination. Rather than try to ensure this using acknowledgments, the protocol calculates, a priori, how many times each packet should be forwarded by each router. To do this, routers calculate the following values.

z_j = The expected number of transmissions that router j must issue for a given packet.
L_j = The probability that node j must participate in forwarding any given packet.

These values have the following relationship:

$$
\begin{aligned}
z_j &= \frac{L_j}{1 - \prod_{k \dot{<} j} \epsilon_{jk}} \\
L_j &= \sum_{i \dot{>} j} (z_i (1 - \epsilon_{ij}) \prod_{k \dot{<} j} \epsilon_{ik})
\end{aligned}
$$

where $i \dot{<} j$ means that i is closer to the destination than j according to the ETX metric. These values are calculated incrementally by each node as a packet travels toward the destination. All routers periodically ping each other to figure out all the ϵ_{ij}'s. For each packet received by from an upstream forwarder, each router tries to send $\frac{1}{1-\prod_{k \dot{<} j} \epsilon jk}$ recoded packets.

8.1.1 Bloom Filters

Motivation

A Bloom filter is a concise probabilitic representation of a set of integers. A Bloom filter for a set S is used to answer questions of the form "$x \in S$?"

Often a Bloom filter will be constructed by one system and sent over a network to a recipient. The Bloom filter for a set is much smaller than the set itself, which makes it appropriate for sending over a network (or for storing the filter at a higher level in memory hierarchy when the full set will not fit).

Using the Bloom filter, the recipient can answer questions about set membership. When the Bloom filter says "$x \notin S$", the answer is always correct. When the Bloom filter says "$x \in S$", there is some probability, Pr[false positive], that the element x is not really in the set S.

Construction

Bloom filters are based on randomized binary hashing. The filter itself is a bit array of length m, and a set of k hash functions, $h_1, ..., h_k : I \rightarrow \{0, 1\}$ where I is the universe of elements that may be part of the set, and is usually very large or infinite in applications where Bloom filters are used (for example I may be the set of all URLs or of all strings). When constructing the filter, the bit array is initially set to all 0, and the k hash functions are generated randomly.

- To insert the element x, the constructor sets the bits in positions $h_1(x), ..., h_k(x)$ to 1.
- To check is x is in the set, the querier checks if all of the bits in positions $h_1(x), ..., h_k(x)$ are set to 1. If so, the answer is yes (but this could be a false positive). If any of the bits are 0, the answer is no.

False positives occur due to collisions between each hash of the queried element and *any* hash of *any* element in the set. For example, if a Bloom filter is representing the set $S = \{a, b, c\}$ using 3 hash functions h_1, h_2, h_3 and for $d \notin S$, $h_1(d) = h_2(c)$, $h_2(d) = h_2(a)$ and $h_3(d) = h_1(a)$, then there would be a false positive for d.

In the next lecture we will learn how the parameters of the Bloom filter can be tuned to attempt to minimize both the size of the Bloom filter, m, and Pr[false positive].