CPSC 536N: Randomized Algorithms 2014-15 Term 2

Lecture 5

Prof. Nick Harvey *University of British Columbia*

1 Peer-to-peer Systems

Around 10 years ago, there was a lot of interest in "peer-to-peer systems", both academically, and in the real world. A peer-to-peer system is a decentralized approach for organizing computers and data in a distributed system.

On the academic side, there was interesting work giving novel ways to organize data and nodes in a distributed system (Consistent Hashing, Chord, Pastry, Kademlia, etc.). In the real world, many large peer-to-peer systems were developed, primarily for file sharing (Gnutella, Kazaa, BitTorrent, etc.). Several large technology companies were formed based on these technologies (e.g., Akamai and Skype).

The rough design goals of these peer-to-peer systems are as follows.

- Every node should maintain connections to a small number of other nodes. (In other words, the graph of connections between nodes should have small degree.)
- No node knows who all the other nodes in the system are, or how they are connected, or even the number of nodes.
- Any node A should be able to send a message to any other node B by traversing few intermediate nodes. (In other words, the graph of connections between nodes should have small diameter.) Moreover, the node A should be able to efficiently *find* a short path from itself to B.
- We assume that nodes are mostly cooperative: if node A asks node B to take some action, e.g., form a connection, or forward a message onwards, then node B will oblige.

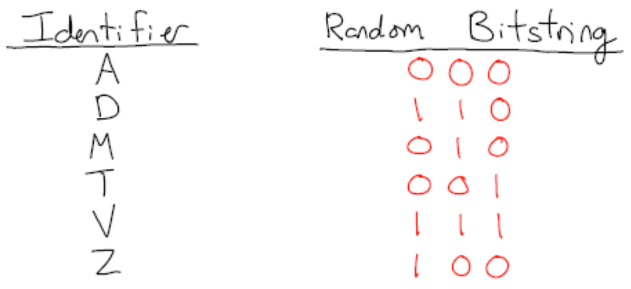
1.1 Example: SkipNet

We will discuss a system SkipNet which meets these design goals. Specifically, in a system with n nodes, every node will maintain connections to $O(\log n)$ other nodes, and any node can send a message to any other node while traversing only $O(\log n)$ intermediate nodes. The system's design is based on ideas from theoretical computer science, in particular the dictionary data structure known as Skip Lists. We discuss this system because its analysis illustrates the usefulness of random variables with the negative binomial distribution.

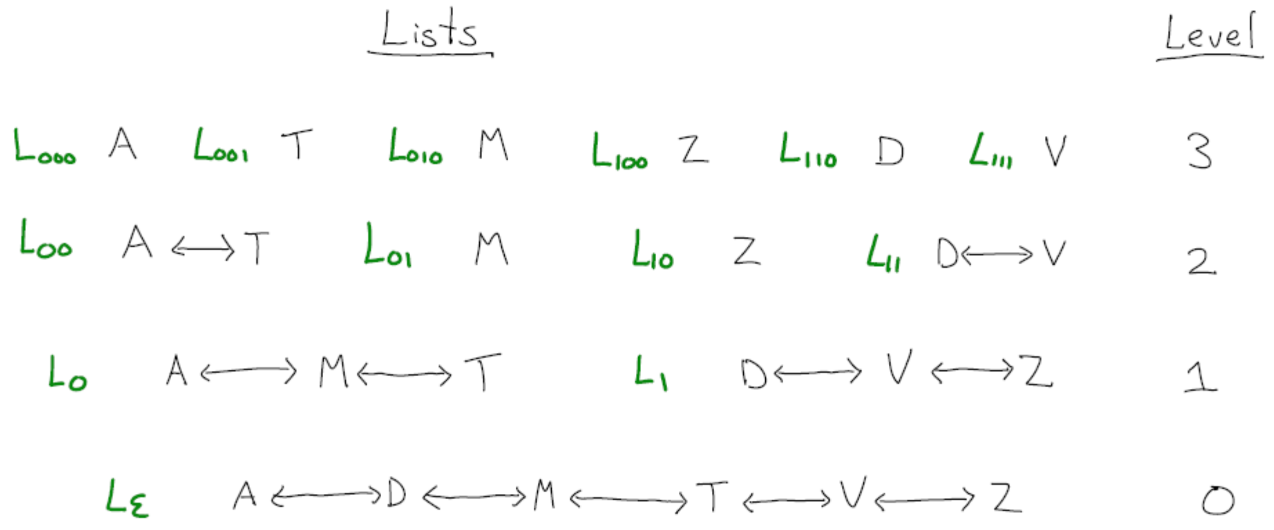
Suppose there are n nodes in the peer-to-peer system. Every node has a string identifier x (e.g., $x =$"www.cs.ubc.ca") and a random bitstring $y \in \{0,1\}^m$, for some parameter m whose value we will choose later.

The nodes are organized into several doubly-linked lists. Each node is a member of several of these lists, but not all of them. Each of these lists is sorted using the nodes' string identifiers. Every pair of nodes that are adjacent in these lists forms a bidirectional network connection.

Formally, for every bitstring z of length at most m, there is a list L_z. We denote the length of bitstring z by $|z|$. The list L_z contains all nodes for which z is a prefix of their random bitstring. We say that the list L_z has "level" equal to $|z|$. For example, for the empty bitstring $z = \varepsilon$, there is a list L_ε which contains *all* nodes, sorted by their identifiers. This list has level 0. At level 1 there are two lists, L_0 and L_1, each of which contain a subset of the nodes: the list L_0 contains all nodes whose random bitstrings start with a 0, and L_1 contains all nodes whose random bitstrings start with a 1. So, for each level, every node belongs to exactly one list at that level.



Identifier	Random Bitstring
A	0 0 0
D	1 1 0
M	0 1 0
T	0 0 1
V	1 1 1
Z	1 0 0



Lists
Level
L_{000} A L_{001} T L_{010} M L_{100} Z L_{110} D L_{111} V 3
L_{00} A ⟷ T L_{01} M L_{10} Z L_{11} D ⟷ V 2
L_0 A ⟷ M ⟷ T L_1 D ⟷ V ⟷ Z 1
L_ε A ⟷ D ⟷ M ⟷ T ⟷ V ⟷ Z 0

Claim 1 *Let $z \in \{0,1\}^*$ be a bitstring of length $|z| = k$. Then the expected size of the list L_z is $n/2^k$.*

PROOF: Consider any node and let its random bitstring be y. The probability that z is a prefix of y is exactly $1/2^k$. □

This suggests that for most bitstrings z with $|z| \gg \log n$ the list L_z is empty. The following claim makes this more precise.

Claim 2 *Let $k = 3\log_2 n$. With probability at least $1 - 1/n$, every list L_z with $|z| \geq k$ contains at most one node.*

PROOF: Note that if two nodes belong to different lists at level i then they also belong to different lists at every level $j \geq i$. So it suffices to prove the claim for the case $|z| = k$.

Consider two nodes with random bitstrings y_1 and y_2. These nodes belong to the same list at level k if and only if y_1 and y_2 have equal prefixes of length k, i.e., the first k bits of y_1 equals the first k bits of

y_2. So the probability that these two nodes belong to the same list at level k is 2^{-k}. By a union bound,

$$\Pr[\,\text{any two nodes belong to the same list at level } k\,] \;<\; n^2 2^{-k} \;=\; 1/n.$$

□

Recall that we only create network connections between pairs of nodes that are adjacent in any list. So the previous claim has two implications. First of all, we should choose $m = 3\log_2 n$. There is no point in creating lists at any level higher than $3\log_2 n$ because they will almost certainly contain at most 1 node, and therefore will not be used to create any network connections between nodes. Second of all, since every node belongs to m lists, it has only $O(m)$ neighbors in total, and therefore participates in only $O(\log n)$ network connections. This satisfies our first goal in the design of the peer-to-peer system.

Sending Messages. It remains to discuss how a node can send a message to any another node. Recall that this will happen by a sequence of intermediate nodes forwarding the message towards the final destination. The simplest way to do this would be to use the list L_ε at level 0. Since every node belongs to this list, a node can just send the message to his neighbor until the message arrives at the destination. Note that this process does not involve "flooding": in our example above, if D wants to send a message to T then the message would traverse $D \to M \to T$ and would not be sent to nodes A, V or Z.

However, that process of forwarding messages along the list L_ε is not very efficient. If there are n nodes in the system, then the message might need to be forwarded $\Theta(n)$ times before arriving at its destination. (The Skype network has tens of millions of active users, so forwarding each message millions of times would not be very desirable!)

To send messages more efficiently, we will make use of the other lists. The main idea is: a list at a high level has few nodes, and those nodes are essentially uniformly distributed amongst all nodes, so the connections formed by high-level lists allow one to "skip" over many nodes in the L_ε list. To be a bit more precise, consider any bitstring z. Let $z0$ and $z1$ be the bitstrings respectively obtained by appending 0 and 1 to z. Every node in L_z randomly chooses (by choosing its random bitstring) whether to join L_{z0} or L_{z1}, with roughly half of the nodes going to each. If the nodes in L_z who join L_{z0} perfectly interleave the nodes who join L_{z1} then every connection between adjacent nodes in L_{z0} corresponds to a *path of two connections* in L_z. But, due to the randomness, the connections in L_{z0} may not correspond to a path of exactly two connections, it might correspond to just one connection in L_z, or a path of three connections, etc.

This discussion suggests that sending a message using the connections in L_{z0} instead of L_z allows one to make roughly twice as much progress towards the destination. So we'd prefer to send the message using only the high-level lists. Of course, that is not always possible: the source and the destination might belong to different high-level lists, in which case one must use the low-level lists too. So we devise the following rule for routing messages from a node x.

- Send the message through node x's level m list as far as possible *towards* the destination without going *beyond* the destination, arriving at node e_m.
- Send the message from e_m through node x's level $m-1$ list as far as possible *towards* the destination without going *beyond* the destination, arriving at node e_{m-1}.
- Send the message from e_{m-1} through node x's level $m-2$ list as far as possible *towards* the destination without going *beyond* the destination, arriving at node e_{m-2}.
- ...
- Send the message from e_1 through the level 0 list to the destination (which we can also call e_0).

Consider our previous example. Suppose that node Z wants to send a message to node A. Node Z's level 3 list contains only node Z, so it cannot be used to make progress towards the destination. So $e_3 = Z$. The same is true for its level 2 list. So $e_2 = Z$. Node Z's level 1 list has a connection to node V, which is not beyond the destination, so we send the message to V. Node V can continue sending the message through node Z's level 1 list to node D. So $e_1 = D$. Finally, node D can send the message through the level 0 list to A, which is the destination.

The correctness of this routing rule is guaranteed by the last step: sending the message through the level 0 list always reaches the destination because that list contains all nodes. It remains to analyze how efficient the rule is.

2 Analysis of SkipNet Routing

Our main theorem shows that any message traverses only $O(\log n)$ nodes to reach its destination.

Theorem 3 *With probability at least $1 - 2/n$ (over the choice of the nodes' random bitstrings), this routing scheme can send a message from any node to any other node by traversing $O(\log n)$ intermediate nodes.*

PROOF: Let the source node have identifier x and random bits y. Let the destination node have identifier x_D. Let P be the path giving the sequence of nodes traversed when routing a message from the source x to the destination x_D.

Intuitively, we want to show that the path traverses only a *constant* number of connections at each level, and since the number of levels is $m = O(\log n)$, that would complete the proof. Unfortunately, this statement is too strong: there might be a level where we traverse many connections (even $\Omega(\log n)$ connections). But at long as this doesn't happen too often, we can still show that the total path length is $O(\log n)$.

So instead, our analysis is a bit more subtle. The main trick is to figure out a protocol for routing the message *backwards along the same path* P from the destination x_D to the source x. Recall that each node e_i is the node in x's level i list which is closest to the destination x_D (without going beyond it). Since e_i is also contained in x's level $i-1$ list, we can find e_i in the following way: Starting from e_{i-1}, move *backward* through x's level $i-1$ list towards x, until we encounter the first node lying in x's level i list. That node must be e_i.

In other words, the following protocol routes backwards along path P from $x_D = e_0$ to x.

- **Phase 0.** Send the message from e_0 through the level 0 list towards x, until we reach the first node in x's level 1 list. (This node is e_1.)
- **Phase 1.** Send the message from e_1 through x's level 1 list towards x, until we reach the first node in x's level 2 list. (This node is e_2.)
- ...
- **Phase $m-1$.** Send the message from e_{m-1} through x's level $m-1$ list towards x, until we reach the first node in x's level m list. (This node is e_m.)
- **Phase m.** Send the message from e_m through x's level m list until reaching node x.

It is easier to analyze the protocol for routing backward along P rather than forward along P because it allows us to "expose the random bits" in a natural order. To start off, imagine that only node x has chosen its random bitstring y.

In Phase 0, we walk through the level 0 list from e_0 towards node x. At each node, we flip a coin to choose the first random bit of its bitstring. If that coin matches the first bit of y then that node is in x's level 1 list and so that node must be e_1. So the number of steps in Phase 0 is a random variable with the following distribution: the number of fair coin flips needed to see the first head. This is a geometric random variable with probability 1/2. (Actually it is not quite that distribution because we would stop if we were to arrive at x without ever seeing a head. But the number of steps is certainly upper bounded by this geometric random variable.)

We then flip coins to choose *every* node's first bit in their random bitstring, except for the bits that are already determined (i.e., the first bit of y and the bits that we just chose in Phase 0). Now x's level 1 list is completely determined, so we can proceed with Phase 1.

In Phase 1, we walk through x's level 1 list from e_1 towards node x. At each node, we flip a coin to choose the *second* random bit of its bitstring. If that coin matches the second bit of y then that node is in x's level 2 list and so that node must be e_2. As before, the number of steps in Phase 1 is at most the number of fair coin flips needed to see the first head.

This process continues in a similar fashion until the end of Phase $m-1$.

So how long is the path P in total? Our discussion just showed that the number of steps needed to arrive in x's level m list it is upper bounded by the number of fair coin flips needed see m heads, which is a random variable with the negative binomial distribution. But once we arrive in x's level m list, we are done because Claim 2 shows that this list contains only node x (with probability at least $1-1/n$). So it remains to analyze the negative binomial random variable, which we do with our tail bound from Lecture 3.

Claim 4 *Let Y have the negative binomial distribution with parameters k and p. Pick $\delta \in [0,1]$ and set $t = \frac{k}{(1-\delta)p}$. Then $\Pr[Y \geq t] \leq \exp\big(-\delta^2 k/2(1-\delta)\big)$.*

Apply this claim with parameters $k=m$, $p=1/2$ and $\delta=3/4$, so $t = \frac{m}{(1/4)(1/2)} = 8m$. This shows that the probability of Y being larger than t is at most

$$\exp\Big(-\frac{\delta^2 k}{2(1-\delta)}\Big) \;=\; \exp\Big(-\frac{(9/16)m}{2(1/4)}\Big) \;<\; \exp(-m) \;<\; n^{-3}.$$

Taking a union bound over all possible pairs of source and destination nodes, the probability that any pair has Y larger than $8k$ is less than $1/n$. As argued above, the probability than any node's level m list contains multiple nodes is also at most $1/n$. So, with probability at least $1-2/n$, any source node can send a message to any destination node while traversing at most $8k = O(\log n)$ nodes. □

2.1 Handling multiple types of error

In the previous proof, there were two possible bad events. The first bad event $\mathcal{E}_1$ is that the source node x's list at level m might have multiple nodes. The second bad event $\mathcal{E}_2$ is that the path from the destination back to x's list at level m might be too long. How can we show that neither of these happen? Can we condition on the event $\mathcal{E}_1$ not happening, then analyze $\mathcal{E}_2$?

Such an analysis is often difficult. The reason is that our analysis of $\mathcal{E}_2$ was based on performing several independent trials. Those trials would presumably not be independent if we condition on the event $\mathcal{E}_1$ not happening.

Instead, we use a union bound, which avoids conditioning and doesn't require independence. We separately showed that $\Pr[\mathcal{E}_1] \leq 1/n$ and $\Pr[\mathcal{E}_2] \leq 1/n$. So $\Pr[\mathcal{E}_1 \cup \mathcal{E}_2] \leq 2/n$, and the probability of no bad event occuring is at least $1 - 2/n$.