Lecture 22:

Erasure (Deletion) Channels & Digital Fountain Codes

November 30, 2005

Erasure Channels

- In many practical communication systems, information is not just corrupted, it is outright lost by the channel.
- For example, on the internet, information is transmitted in packets which often completely disappear.
- As a simple model, imagine that packets are lost with probability f and that each packet has an ID number embedded in it (perhaps using the first few bits), so the receiver knows which packets she received and which were lost.
- We previously studied the Binary Erasure Channel (BEC), which is a very simple example of this when packets have length 1 bit: if the packet was lost we received a "?", otherwise we received the packets correctly.
- We'd like to think about the case when the packets are (much) longer than one bit in length.

Recovering From Erasures

- How can we recover from erasures?
- For the BEC, we saw that low-density parity check (LDPC) codes were very effective at "filling-in" the missing bits and had a very efficient decoding algorithm.
- For slightly longer packets (a few bits to a few tens of bits in length), there is a class of *Reed-Solomon* codes which can correct against erasures and long runs (blocks) of errors.
- For very long packets there is an extremely clever scheme called *Digital Fountain Codes* developed by Mike Luby in 1988.

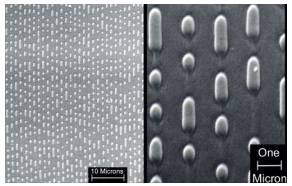
see `www.digitalfountain.com`

Inefficiency of Retransmission

- Why do we need to recover lost packets algorithmically? Why not just either explictly acknowledge each received packet and have the sender retransmit those not acknowledged? Or why not have the receiver explicitly request retransmission of those not received?
- This requires a feedback channel (from receiver $\rightarrow$ sender) which often does not exist, especially in a broadcast situation.
- Furthermore, retransmission is very wasteful of capacity. Either many unecessary acknowledgements are sent (in low-erasure channels) or else many redundant retransmissions are made (in high-erasure channels).
- We would like a scheme which does not require a feedback channel makes optimal use of the capacity of the forward channel, regardless of whether the erasure rate is high or low.

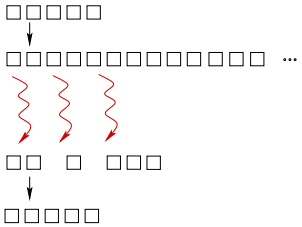
Reed-Solomon Codes

- Think of our message as $K+1$ numbers $s_0, s_1, \ldots, s_K$.
- Consider the polynomial $S(x) = s_0 + s_1 x + s_2 x^2 + \ldots + s_K x^K$.
- Evaluate $S(x)$ at $N > K$ points $x_1, x_2, \ldots, x_N$.
- Transmit the values $S(x_1), S(x_2), \ldots, S(x_N)$ across the channel.
- If *any* $K+1$ or more of the values are received, we fit a polynomial to them and exactly recover the coefficients (the original message).
- This is the basic idea behind Reed-Soloman codes, except that they are implemented with finite precision instead of real numbers, using the mathematics of Galois Fields.
- These codes are used on compact discs, DVDs, HDTV and in communication with Voyager, Gallileo and other space satellites.



Luby Transform (LT) Codes

- The idea of LT codes is that the sender is a fountain that produces an endless supply of encoded packets. (Hence these codes and their variants are often called digital fountain codes.)
- Say the original source file has a size of Kl bits, and each packet contains l encoded bits.
- Anyone who wishes to receive the complete file holds a bucket under the fountain and collects packets until they have collected a little more than K (in practice, this is usually around 5%).
- They can then almost certainly (with prob. $1-\delta$) recover the original file exactly.
- The overhead scales as $\sqrt{K}(\ln(K/\delta))^2$.



LT Codes are Rateless and Universal

- LT codes are *rateless* in the sense that the number of encoded packets that can be generated from the source message is potentially limitless and the number of encoded packets generated can be determined on the fly.
- Regardless of the statistics of the erasure events on the channel, we can send as many encoded packets as are needed in order for the decoder to recover the source data.
- LT codes also have fantastically small encoding and decoding complexities. With probability $1-\delta$, K packets can be communicated with average encoding and decoding costs both of order $K \ln(K/\delta)$ packet operations.
- Luby calls these codes *universal* because they are simultaneously near-optimal for every erasure channel, and they are very efficient as the file length K grows.

Encoding for LT Codes

- Each encoded packet t_n is produced from the source $s_1 \ldots s_K$ as follows:
  1. Randomly choose the "degree" d_n of the packet t_n from a degree distribution $\rho(d)$. (The appropriate choice of ρ depends on the source file size K, as we'll discuss later.)
  2. Choose, uniformly at random, d_n distinct input packets, and set t_n equal to the *bitwise sum* (modulo 2) of those d_n packets. (Computed by successively XOR-ing the packets together.)
- This encoding operation defines a graph connecting encoded packets to source packets. If the mean degree $\bar{d}$ is significantly smaller than K then the graph is sparse.
- We can think of the resulting code as an irregular LDPC code.

Decoding LT Codes

- Decoding LT codes is acheived with a message passing algorithm very similar to the one we used for LDPC codes on the BEC.
- Think of the encoded packets $\{t_n\}$ as check nodes.
- To start with, set all the source packets s_k to be "?" (unknown).
- Repeatedly find a check node t_n that is connected to *only one* unknown source packet s_k. (If there is no such check node, decoding halts fails to recover all the source packets.)
- Fill in the unknown source packet using the XOR of the known packets (if any) and the encoded packet at the check node.

Sending the Connectivity Pattern (Graph)

- The decoder needs to know the degree of each packet that is received, and which source packets it is connected to in the graph.
- This information can be communicated in various ways.
- For example, if the sender and receiver have synchronized clocks, they could use identical pseudo-random number generators, seeded by the clock, to choose each random degree and each set of connections.
- Alternatively, the sender could pick a random key, κ_n, given which the degree and the connections are determined by a pseudo-random process, and send that key in the header of the packet. As long as the packet size l is much bigger than the key size (which need only be 32 bits or so), this key introduces only a small overhead cost.

Designing the degree distribution

- The probability distribution $\rho(d)$ of the degree is a critical part of the design: occasional encoded packets must have high degree (i.e. d similar to K) in order to ensure that there are not some source packets that are connected to no-one.
- Many packets must have low degree, so the decoding process can get started, and keep going, and so the total number of addition operations involved in the encoding and decoding is kept small.
- Ideally, to avoid redundancy, we'd like the received graph to have the property that just one check node has degree one at each iteration. At each iteration, when this check node is processed, the degrees in the graph are reduced in such a way that one new degree-one check node appears.

The Soliton Distribution

- *In expectation*, this ideal behaviour is achieved by the *ideal soliton distribution*,

$$\begin{aligned} \rho(1) &= 1/K \\ \rho(d) &= \tfrac{1}{d(d-1)} \quad \text{for } d = 2, 3, \ldots, K. \end{aligned}$$

- The expected degree under this distribution is roughly $\ln K$.
- Unfortunately, this degree distribution works poorly in practice, because fluctuations around the expected behaviour make it very likely that at some point in the decoding process there will be no degree-one check nodes; and, furthermore, a few source nodes will receive no connections at all. A small modification, slightly increasing the bias towards small degrees, fixes these problems.

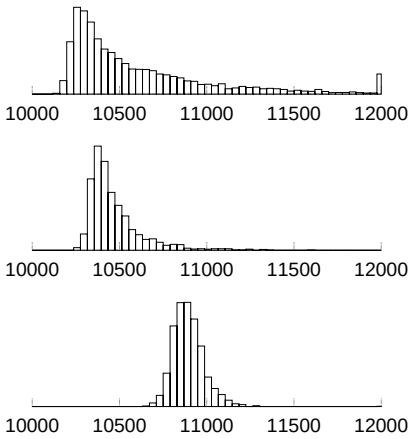
The Robust Soliton Distribution

- The *robust soliton distribution* has two extra parameters, c and δ; it is designed to ensure that the expected number of degree-one checks is about
$$R \equiv c \ln(K/\delta)\sqrt{K},$$
rather than 1, throughout the decoding process.
- The parameter δ is a bound on the probability that the decoding fails to run to completion after a certain number K' of packets have been received; c is a free parameter, with a value somewhat smaller than 1 giving good results in practice.
- Luby's key result is that (for an appropriate value of the constant c) receiving $K' = K + 2\ln(R/\delta)R$ checks ensures that all packets can be recovered with probability at least $1 - \delta$.

Example: 10000 packets

- In practice, LT codes can be tuned so that a file of original size $K \simeq 10\,000$ packets is recovered with an overhead of about 5%.

Histograms of the actual number of packets required for a couple of settings of the parameters, achieving mean overheads Z smaller than 5% and 10% respectively.



top: $c = 0.01$, $\delta = 0.5$
($R = 10$, $K/R = 1010$, $Z \simeq 1.01$)
middle: $c = 0.03$, $\delta = 0.5$
($R = 30$, $K/R = 337$, $Z \simeq 1.03$)
bottom: $c = 0.1$, $\delta = 0.5$
($R = 99$, $K/R = 101$, $Z \simeq 1.1$)

Application: Storage

- You wish to make a backup of a large file, but you are aware that your magnetic tapes and hard drives are all unreliable in the sense that catastrophic failures, in which some stored packets are permanently lost within one device, occur at a rate of something like 10^{-3} per day. How should you store your file?
- A digital fountain can be used to spray encoded packets all over the place, on every storage device available. Then to recover the backup file, whose size was K packets, one simply needs to find $K' \simeq K$ packets from anywhere. Corrupted packets do not matter; we simply skip over them and find more packets elsewhere.

Digital Fountain RAID

- This method of storage also has advantages in terms of *speed* of file recovery. In a hard drive, it is standard practice to store a file in successive sectors of a hard drive, to allow rapid reading of the file; but if, as occasionally happens, a packet is lost (owing to the reading head being off track for a moment, giving a burst of errors that cannot be corrected by the packet's error-correcting code), a whole revolution of the drive must be performed to bring back the packet to the head for a second read. The time taken for one revolution produces an undesirable delay in the file system.
- If files were instead stored using the digital fountain principle, with the packets stored in one or more consecutive sectors on the drive, then one would never need to endure the delay of re-reading a packet; packet loss would become less important, and the hard drive could consequently be operated faster, with higher noise level, and with fewer resources devoted to noisy-channel coding.

Application: Brodcast

- Imagine that ten thousand subscribers in an area wish to receive a digital movie from a broadcaster. The broadcaster can send the movie in packets over a broadcast network – for example, by a wide-bandwidth phone line, or by satellite.
- Imagine that not all packets are received at all the houses. Let's say $f = 0.1\%$ of them are lost at each house. In a standard approach in which the file is transmitted as a plain sequence of packets with no encoding, each house would have to notify the broadcaster of the fK missing packets, and request that they be retransmitted. And with ten thousand subscribers all requesting such retransmissions, there would be a retransmission request for almost every packet.
- Thus the broadcaster would have to repeat the entire broadcast twice in order to ensure that most subscribers have received the whole movie, and most users would have to wait roughly twice as long as the ideal time before the download was complete.

Fountain Broadcast

- If the broadcaster uses a digital fountain to encode the movie, each subscriber can recover the movie from *any* $K' \simeq K$ packets. So the broadcast needs to last for only, say, $1.1K$ packets, and every house is very likely to have successfully recovered the whole file.
- Another application is broadcasting data to cars. Imagine that we want to send updates to in-car navigation databases by satellite.
- There are hundreds of thousands of vehicles, and they can receive data only when they are out on the open road; there are no feedback channels. A standard method for sending the data is to put it in a *carousel*, broadcasting the packets in a fixed periodic sequence. 'Yes, a car may go through a tunnel, and miss out on a few hundred packets, but it will be able to collect those missed packets an hour later when the carousel has gone through a full revolution (we hope); or maybe the following day...'
- If instead the satellite uses a digital fountain, each car needs to receive only an amount of data equal to the original file size (+5%).

Details of the Robust Soliton

- We define a positive function

$$\tau(d) = \begin{cases} \frac{R}{K}\frac{1}{d} & \text{for } d = 1, 2, \ldots (K/R) - 1 \\ \frac{R}{K}\ln(R/\delta) & \text{for } d = K/R \\ 0 & \text{for } d > K/R \end{cases}$$

  then add the ideal soliton distribution ρ to τ and renormalize to obtain the robust soliton distribution, μ.
- The number of encoded packets required at the receiving end to ensure that the decoding can run to completion, with probability at least $1 - \delta$, is $K' = K[\sum_d \rho(d) + \tau(d)]$.
- Luby's analysis explains how the small-d end of τ has the role of ensuring that the decoding process gets started, and the spike in τ at $d = K/R$ is included to ensure that every source packet is likely to be connected to a check at least once.