CS 559: Algorithmic Aspects of Computer Networks **Fall 2007**

Lecture 3 — September 12

Lecturer: John Byers *BOSTON UNIVERSITY* *Scribe: Konstantin Voevodski*

In today's lecture, we discussed multicast protocols. The desirable features of a multicast protocol were described, and two protocols were considered.

3.1 Multicast vs. Unicast

Multicast allows the server to only transmit a message (destined for multiple users) once, instead of establishing an independent connection with each end-point. It outperforms unicast because it saves server load and diminishes network traffic.

However, there are serious problems/challenges with implementing multicast because the paradigm on which networks are built is that the end-points are "smart" while the internal routing devices are "dumb." To reliably and efficiently deliver every packet to every destination is difficult, and requires a lot of feedback and coordination among the routers. Today there is no reliable IP multicast, but P2P approximations exist where the message is first delivered to a number of end-points, which resend it to their neighbors.

3.2 An Ideal Solution

As described in [1], an ideal solution should satisfy the following requirements:

- scalability (solution works just as well when the number of receivers, size of network, size of file increase)
- reliability (message is delivered in its entirety without errors)
- reception-efficiency (the amount of redundant data for the receiver is minimized)
- time-efficiency (the amount of time needed for the receiver to reconstruct the message is minimized)
- time-independence (as far as the receiver is concerned, it does not matter when it wants the file)
- server-independence (the state of the network is irrelevant to the server)
- tolerance (of heterogeneity of network: different bandwidth, latency and loss-rate at different end-points)

3.3 An IDA Implementation on Top of Multicast

Let us consider Rabin's IDA scheme, given in [2], implemented using reliable multicast (assuming we already have a reliable multicast). The server breaks up the file to be sent into k packets, generates the $2k$ packets (dispersions) using the IDA scheme, and multicasts all of them. Users who receive any k distinct packets are done, those which do not must wait for additional packets before they can reconstruct the file.

Thus the server will continuously multicast the $2k$ packets (the so-called "data carousel") and users will log on, wait until they receive k distinct packets (which allows them to reconstruct the original file), and then log off.

Let us think about the advantages and disadvantages of this approach. The solution is reliable because once a receiver has the necessary information, the original file can be reconstructed without error. Also, this scheme is server and time-independent (server and receivers always perform the same actions). The solution is also tolerant in the sense that it will work for receivers with different "abilities," just for some much faster than others. The approach also provides fairly good reception-efficiency because once a receiver has the k packets, it will not receive any more packets (which would be redundant as far as being able to reconstruct the original file). However, the k packets must be DISTINCT, so duplicate packets indeed hurt the reception-efficiency.

The big disadvantage of this solution is that it may take a long time for the user to reconstruct the original message (once the k packets have arrived). The complexity of decoding is $O(k^2)$, so time-efficiency becomes an issue when the number of packets (k) gets large. The scheme thus does not scale well because as the size of the file grows, so does the number of packets, and the time it takes for the user to reconstruct the original file.

3.4 "Fixing" IDA

The scheme described above suffers from poor time-efficiency and less than perfect reception-efficiency. Ideally, we would like to use a code which allows us to reconstruct the original file faster. Also, we would like a longer, if not infinite, data carousel. The reason is that the longer the data carousel, the higher the chances of the user receiving all the necessary packets in one round (without having to wait until the next round of duplicate dispersions and inevitably receiving some duplicate packets it does not need). In the idealized setting, all the packets sent by the server will be distinct (creating an infinite data carousel), and receivers will just wait for enough of them to arrive to reconstruct the original file. Also, the server should not have to do extra work in generating the packet stream (preserving server-independence).

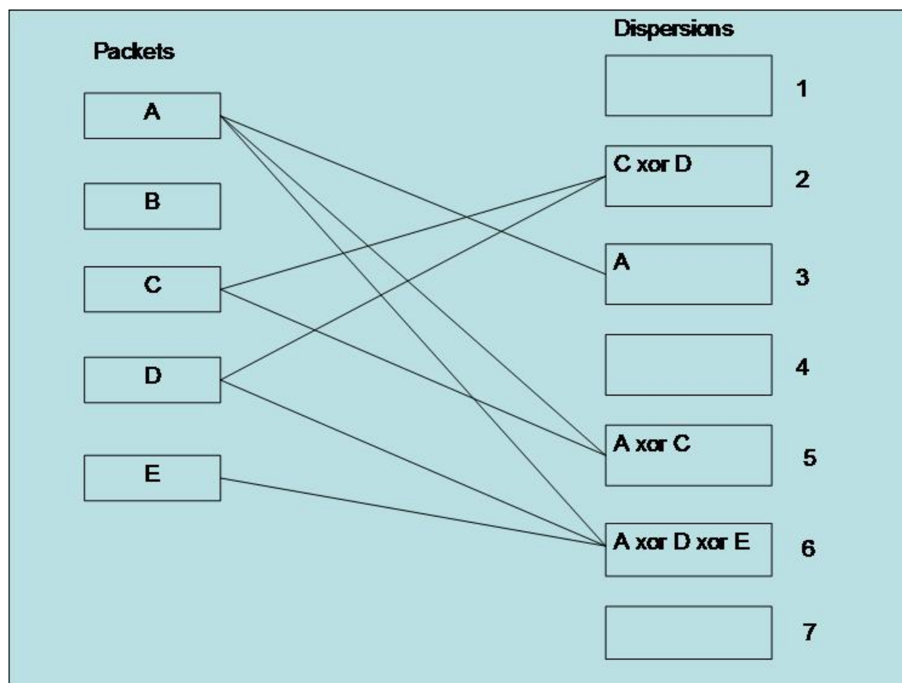
One way to think about decoding the original file is to picture a bipartite graph where one side contains the packets of the original file and the other side contains the dispersions. There is an edge between a packet and a dispersion if that dispersion depends on that packet. The complexity of decoding the file is then proportional to the number of edges in this graph. In the Rabin IDA scheme the number of edges is $O(k^2)$ because each dispersion depends on

every packet of the original message.

The complexity of decoding the file can thus be reduced if the dependency graph between packets and dispersions can be sparsified. A way to do this with the Rabin scheme is to introduce zeroes into the transformation matrix A, so that each dispersion is not dependent on all the packets. This will speed up the time-efficiency, but hurt the reception-efficiency because the user may have to wait a long time for the last few dispersions needed to decode the remaining packets, throwing away all other packets it receives.

3.5 Tornado Codes

The idea behind Tornado codes, described in [1], is to make the dependency graph sparse on average, but include "high-degree" dispersions which occur often enough so that the receiver does not wait long to receive all the dispersions needed to reconstruct the whole file. Dispersions are generated by picking a subset of packets at random, and taking their XOR. Decoding is performed in a bottom up manner by starting from the simplest dispersions (ones which originate from just one packet), and proceeding to the more complicated ones. Any dispersion which has one or fewer "unknowns" can be decoded. As an example, see the figure below:



First, we can recover packet A from the 3rd dispersion. Now that we know A we can determine C from the 5th dispersion. Then we can recover D from the 2nd dispersion. Finally, we can recover E from the 6th dispersion. However, we still do not know B, so we must wait for more dispersion packets to arrive until we can recover it. Clearly, Tornado codes are very efficient to decode, the hard part is generating the dispersions in such a manner

to prove that the receiver will not wait long on average before having enough dispersions to reconstruct the entire original file.

3.6 Bibliography

[1] J. W. Byers, M. Luby, M. Mitzenmacher, *A Digital Fountain Approach to Asynchronous Reliable Multicast,* IEEE Journal on Selected Areas in Communication, 2002.

[2] M. Rabin. *Efficient Dispersal of Information for Security, Load Balancing and Fault Tolerance,* Journal of the ACM 38, pp. 335-348, 1989.