E.R.R.A. protocol: design, implementation and analysis

Carraro Enrico, Cioccarelli Marco, Sorato Giacomo

March 13, 2014

1 Introduction

Exotic Random Routing Algorithm (ERRA) is a pseudo peer-to-peer application-layer routing algorithm, in which some nodes, defined as bootstrap, are supposed to be known by the entire network. To transfer a file, the sender partitions it in a suitable number of packets that are then sent to the destination, each following a different path that crosses all the active nodes in a random order.
There are various possible design choices to define the protocol.

2 Protocol general choices

We have considered an approach where the bootstrap is the only node that knows the node list, while generic nodes have to contact the bootstrap to get access to it. Therefore each node that joins or disconnects from the network has to communicate its address, that we decided to be the IP address of the host, to the bootstrap, which stores it in the list. When a node wants to send a file to another node, it has to ask to the bootstrap the node list, then it decides the route that the packet has to follow and stores it in the packet's header.
The advantage is that we have low "service" traffic in case of frequent connections and disconnections compared to the approach where all the nodes store the node list in local memory, which would have to be kept updated each time a node joins or disconnects from the network.
In case the number of connections and disconnections were relativity low, we could adopt the alternative above-mentioned approach which would permit direct transfer of files between nodes, without requesting node list to the bootstrap.
Assuming frequent connections and disconnections and relatively few file transfers, we chose the first approach.
The name of each file at the source is the IP address of the destination node, while at the destination each file is stored with the name of the source plus an incremental index, since each node can receive more than one file from the same source. Finally, all connections are established by using the TCP protocol.

3 Protocol design

In addition to data packets, the protocol employs signaling packets, distinguished from each other by the first byte of the header: connection request, disconnection request, node list request, node list response, error no destination found. The protocol is so structured:

Initializing the network: An active bootstrap is required and its IP address is assumed known by every node.

Joining/disconnecting from the network: A new node establishes a connection (sending a connection/disconnection request packet) with the bootstrap, that stores/removes the IP address of the node in the node list.

Sending a file: To send a file of dimension F:

- the source node sends a node list request packet to the bootstrap
- the addresses list is received from the bootstrap

- the source splits the file in F/N portions, where N is the portion size
- for each portion, a different route is generated by randomly shuffling the addresses list and inserting the destination address at the end of it
- each portion is encapsulated in a data packet having an header defined in this way:

1 BYTE [0]	:	Number of addresses in the addresses list
4 BYTES [1-4]	:	IP address of the sender
1 BYTE [5]	:	Number of files sent by the source up to this moment
1 BYTE [6]	:	Packet number, denoting the i-th packet
8 BYTES [7-14]	:	File length
4 BYTES * number of addresses	:	Addresses list

  Byte [0] is used to define the type of the packet (data or the different types of signaling) and in case of data packet assumes integer values in (0-250), representing the number of addresses in the addresses list. Each one of the values in (251-255) is reserved for a different type of signaling packet.
  Bytes [1-5] are used to identify uniquely each file. Byte [5] is necessary since, in a certain moment, the destination could be receiving two or more different files from the same source, and therefore it needs to be able to distinguish them.
  Byte [6] (Packet number) is used by the destination node to reorder the packets that compose a file.
  Bytes starting from 15 (Addresses list) contain the IP addresses of the nodes that the packet has yet to visit. At the time of sending it is composed by all the nodes addresses except those of source and of the immediately successive node. So the first address in the list contained in the packet received by the second node of the route from the source is the address of the third node.
- Once all the portions of a file have been sent, the variable representing the number of files sent by the source up to this moment is incremented by 1.

Forwarding: When a node receives a packet, it checks the value of byte [0]: if it equals 0, the addresses list is empty, therefore this node is the destination and the payload is saved. Otherwise byte [0] is decremented by 1, the address of the next node in the route (bytes [15-18] of the header) is read and removed from the addresses list in the header and then the packet is forwarded to that node.

Saving a file: For each file it is receiving, the destination keeps a counter of the remaining portions. When it has received all the portions of a file, it orders them using the packet number (byte [6] of the header) and merges them in a single file.

Node not reachable: A node N in the route could be not reachable either because that the node has suddenly disconnected without sending a disconnection packet to the bootstrap or because the addresses list in a packet's header still contains the address of a disconnected node. A node M, after failing to connect to N, sends a disconnection packet to the bootstrap signaling the departure of N, deletes the address of N from the addresses list in the header and forwards the data packet to the next node in the list.

Bootstrap: The bootstrap node behaves as a generic node and, moreover, manages the node list, adding or removing addresses and sending it to the nodes requesting it.

We implemented the protocol in Java code realizing two classes, one for the generic node and one for the bootstrap.

4 Performance analysis: best packet length choice

An important parameter for the message transfer time is the length of the portion in which a file is split at the source.
To find the optimal portion length N^* that minimizes the transfer time, we adopted an experimental approach: we tested ERRA for incremental values of N in a 6-node network and in a 12-node one, transferring a 1.3MBytes file, and in a 6-node network with a 500kBytes file (Figure 1).
All curves have the same concave trend where, with a file of size X:

- As N tends to small values, the number of packets X/N in which a file is split increases, implying a high number of ERRA headers that reduces the goodput. On the other hand each node can send a portion of the file simultaneously, parallelizing the file transfer.
- As N tends to the file length, X/N decreases up to the limit case in which $X = N$: the file is sent a single packet and the overhead due to ERRA headers becomes negligible, but the packet can cross a single node per time, while the other nodes are idle, implying a high transfer time.

From the data collected we can see that the optimal N^* belongs to the interval (40000-50000) bytes. Therefore, we chose the value 40000 for the portion size.

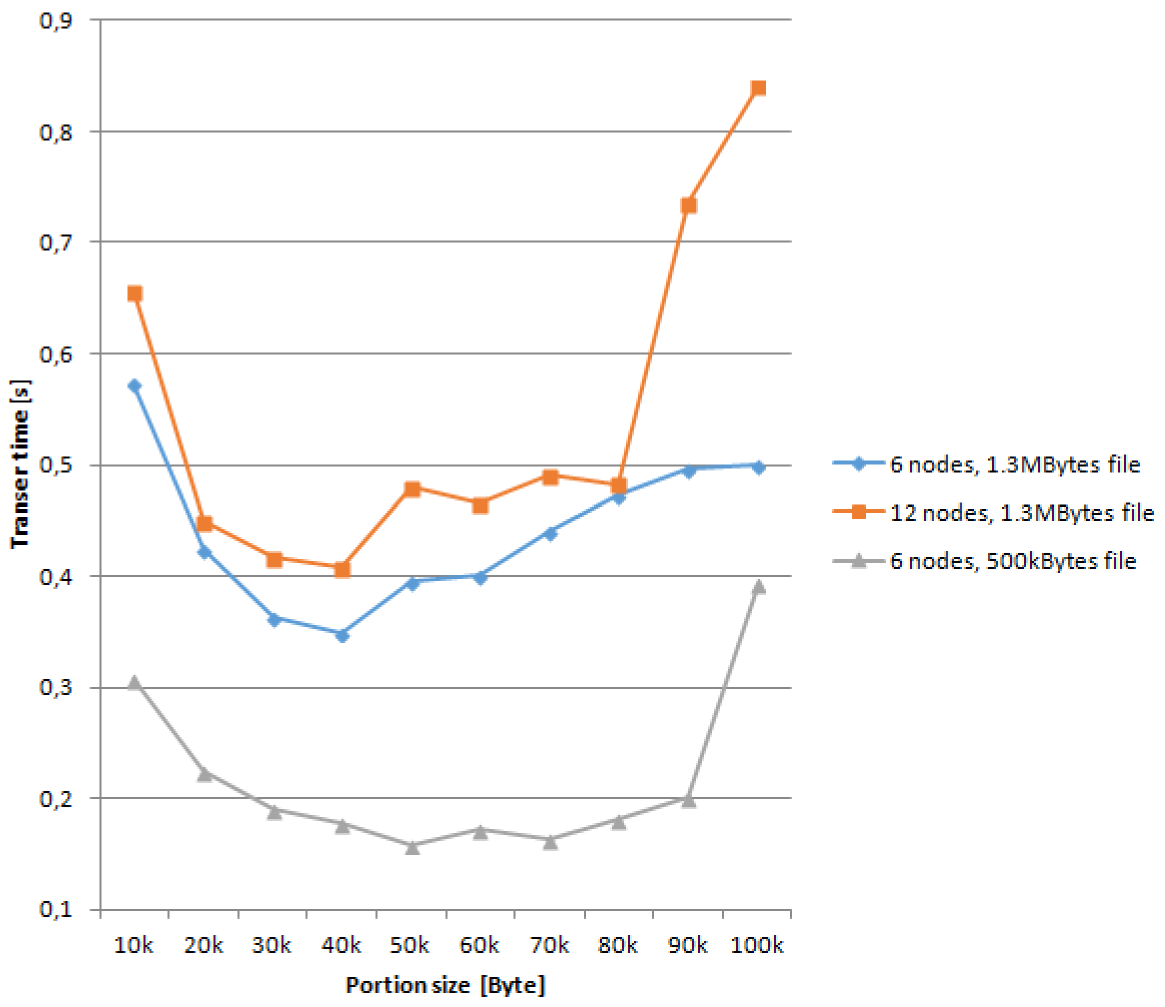

Figure 1: Transfer time of a file for increasing values of the portion length

5 Conclusions

In our version of the ERRA protocol a source routing approach is used and the packets transmissions are supported at transport layer by TCP protocol. A key protocol parameter is the length of portions in which a file is split: through an empirical approach we estimated an optimal portion length in order to minimize the transfer time of a file. The way in which a file is transferred using ERRA can be useful to evenly distribute the traffic among the hosts of a network.
Future improvements of the protocol could be an application layer loss recovery strategy and an optimal portion length chosen at sending time depending also on the network size.