# rStream: Resilient Peer-to-Peer Streaming with Rateless Codes

Chuan Wu, Baochun Li
Dept. of Electrical and Computer Engineering
University of Toronto
Toronto, ON M5S 3G4, Canada
{chuanwu,bli}@eecg.toronto.edu

## ABSTRACT

The inherent instability and unreliability of peer-to-peer networks introduce several fundamental engineering challenges to multimedia streaming over peer-to-peer networks. First, multimedia streaming sessions need to be resilient to the volatile network dynamics in peer-to-peer networks. Second, they need to take full advantage of the existing bandwidth capacities, by minimizing the delivery of redundant content during streaming. In this paper, we propose to use a recent coding technique, referred to as *rateless codes*, to code the multimedia bitstreams before they are transmitted over peer-to-peer links. The use of rateless codes eliminates the requirements of content reconciliation, as well as the risks of delivering redundant content over the network. It also helps the streaming sessions to adapt to volatile network dynamics. Our preliminary simulation results demonstrate the validity and effectiveness of our new contribution, as compared to traditional solutions with or without erasure codes.

## Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Distributed Applications*; H.3.5 [**Information Storage and Retrieval**]: Online Information Services—*Data sharing*

## General Terms

Design, Performance, Reliability

## Keywords

Peer-To-Peer, Media Streaming, Rateless Codes, Content Reconciliation, Failure Resilience

## 1. INTRODUCTION

Despite its many advantages over traditional streaming that either uses unicast sessions, or relies on IP multicast, peer-to-peer streaming still poses significant technical challenges. Peer-to-peer networks are inherently dynamic and unreliable. Thus the demand for stable streaming bit rates may not be satisfied due to this *network dynamics*. Besides, it is typical in a peer-to-peer streaming session for a peer node to parallelly download from multiple upstream peers. While such mesh streaming topologies improve overall bandwidth availability and resilience to dynamics, there exist fundamental problems with respect to *content redundancy and reconciliation*. As there are always risks that the same contents may be unnecessarily supplied by multiple upstream peers, the peer nodes need to reconcile their differences to minimize such risks.

Among the previous work in peer-to-peer streaming, streaming solutions based on multiple multicast trees [3, 8] have been proposed to address these challenges, which involve high cost of tree maintenance. Other solutions stream over mesh topologies [10, 4], and lie on each peer to reconcile and schedule the content downloading from different upstream peers. A well-known piece of work for content reconciliation is from Byers *et al.* [1], which provides computation-intensive algorithms to reconcile sets of symbols between pairs of collaborating peers.

Our main contribution in this paper is a mesh-based peer-to-peer streaming protocol referred to as *rStream*, which employs a recent coding technique, *rateless fountain codes*, to battle on those challenges. We argue that rateless codes can be readily used in peer-to-peer streaming with substantial advantages. As a class of erasure codes, rateless codes provide natural resilience to losses, and therefore provide the best possible resilience to peer dynamics. Being *rateless*, there is potentially no limit with respect to the number of uniquely encoded "blocks," coded from a set of original data blocks. A sufficient number of encoded blocks from any set of peers may be used to recover the original content. This completely eliminates the needs for content reconciliation, as no redundant contents exist in the network.

The remainder of this paper is organized as follows. We present our network model and rStream protocol in Sec. 2. Simulation results are presented in Sec. 3. We then conclude the paper in Sec. 4.

## 2. NETWORK MODEL AND PROTOCOL

In this paper, we consider a peer-to-peer streaming session with one streaming *source* and multiple participating *receivers*. Each receiver is served by one or more *upstream peers*. Since the upstream peers participate in the same session, they are receivers themselves, with the exception of the streaming source. The objective is to stream live multimedia content, coded to a constant bit rate bitstream with a current generation codec such as H.264/AVC, H.263 or MPEG-4, to all the participating receivers in the session. As each peer may have multiple upstream and downstream peers in the streaming session, a *mesh* overlay topology is established for streaming.

Such a mesh topology can be modeled as a directed graph $G =$

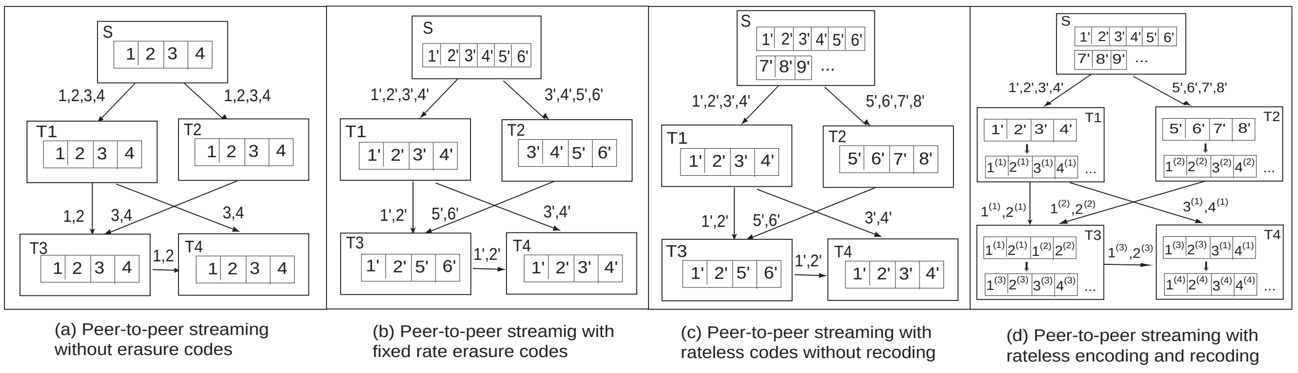


Figure 1: **Peer-to-peer streaming with different coding schemes: a comparison.**

$(N, A)$, where $N$ is the set of vertices (peers) and $A$ is the set of directed arcs (directed overlay links). Let $S$ be the streaming source, and $T$ be the set of receivers in the streaming session. We have $N = S \cup T$. The source $S$ streams a multimedia bitstream $\widetilde{M}$ to the receivers in $T$. Independent of the codec used in $\widetilde{M}$, we treat $\widetilde{M}$ as a stream of symbols, partitioned into consecutive segments $s_1, s_2, \ldots$. A segment typically consists of one media frame, a group of frames (GOF), or simply a period of time (*e.g.*, one second). Each segment $s_i$ is further divided into $k_i$ blocks, and each block has a fixed length of $L$ bits.

## 2.1 Rateless codes

We now motivate the use of rateless codes. As a receiver retrieves media content from multiple upstream peers concurrently, it seeks to minimize the waste of bandwidth due to the delivery of duplicated contents. This problem, commonly referred to as *content reconciliation problem*, is illustrated in an example in Fig. 1 (a). In this example, $S$ transmits the component blocks 1, 2, 3 and 4 of a media segment to $T_1$ and $T_2$ directly, and thus $T_1$ and $T_2$ have the same four blocks. When $T_3$ parallelly streams from $T_1$ and $T_2$, it has to decide which block to retrieve from which upstream peer. This also happens to $T_4$ which parallelly retrieves from $T_1$ and $T_3$.

An opportunity for solving this problem arises from the employment of erasure codes. A $(n, k)$ erasure code, such as Reed-Solomon codes and Tornado codes [2], is a forward error correction code with $k$ as the number of original symbols, and $n$ as the number of generated symbols from the $k$ original symbols. A $(n, k)$ erasure code has the favorable property that if any $k$ (or slightly more than $k$) of the $n$ transmitted symbols are received, the original $k$ symbols can be recovered. Thus an erasure code seamlessly tolerates packet losses and peer dynamics, making it ideal for peer-to-peer parallel transfers.

However, erasure codes do not provide a thorough solution to the reconciliation problem. To illustrate this, consider Fig. 1 (b). With a $(6, 4)$ erasure code, $S$ generates six encoded blocks $1'$, $2'$, ..., $6'$ based on the four original blocks 1, 2, 3 and 4. $T_1$ and $T_2$ both directly retrieve four encoded blocks from $S$ and thus inevitably hold two common blocks. This leads to the need for reconciliation of the parallel retrieval at $T_3$, and later at $T_4$. Even with an erasure code where $n$ is much larger than $k$, the need of content reconciliation may rarely arise, but the problem still may not be completely solved.

To address the challenges from content reconciliation, we propose to use a recently developed category of coding schemes, *rateless codes*. Typical rateless codes include LT codes [5], Raptor codes [9] and online codes [6]. With rateless codes, the number of encoded symbols that can be generated from $k$ original symbols is potentially unlimited. The basic idea that underlines rateless codes is simple. Given $k$ original symbols, a rateless code encoder generates encoded symbols on the fly by performing the exclusive-or operation on a *subset* of the original symbols, which is randomly chosen based on a special degree distribution. At the decoder, a decoding graph, which connects encoded symbols to original symbols, is constructed based on the following information: the number of original symbols each encoded symbol is generated from (the degree) and the indices of these original symbols (neighbor indices). We illustrate the decoding process by a simple example shown in Fig. 2. Based on the decoding graph, the decoder finds that the first encoded symbol is connected to only one original symbol $S_1$, so it sets $S_1 = 1$. Then the decoder performs the exclusive-or operation between $S_1$ and all encoded symbols that are connected to $S_1$ in the graph, removing all the related edges. It next finds another encoded symbol with degree 1, and repeats the above process until all original symbols are recovered.

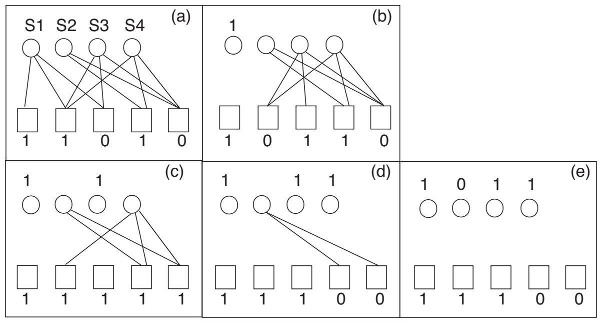


**Figure 2: Decoding with rateless codes: an example.**

With an appropriate choice of degree distribution, the encoded symbols generated in this manner are potentially unique, and any $(1+\epsilon)k$ symbols can be used to recover the original symbols with a high probability by the decoding process. Also since both encoding and decoding only involve exclusive-or operations, rateless codes are very computationally efficient.

Rateless codes are useful towards finding a solution to the content reconciliation problem. In the example shown in Fig. 1(c), from the four original blocks 1, 2, 3 and 4, $S$ generates a unlimited number of encoded blocks $1'$, $2'$, ..., with a rateless code encoder. $T_1$ and $T_2$ are able to each obtain four unique encoded blocks from $S$, thus reconciliation is not required at $T_3$. Unfortunately, content reconciliation may still be required. In our example, $T_1$ and $T_3$ share $1'$ and $2'$, thus $T_4$ still needs to reconcile its parallel retrieval from them.

## 2.2 Recoding with rateless codes

In order to completely eliminate the need for content reconciliation, we explore another desirable property of rateless codes. With rateless codes, the receiver may decode from encoded symbols generated by different rateless-code encoders, as long as they operate on the same set of original symbols [9]. Based on this favorable property, we propose a recoding scheme to be carried out by each peer to guarantee that the received blocks from any upstream peers are unique and useful.

In our protocol, the streaming source encodes the blocks of each media segment by an LT code based on the robust Soliton distribution [5], and streams the encoded blocks. After a peer retrieves the $K_i$ encoded blocks for the segment $s_i$ of media $\widetilde{M}$, where $K_i = (1+\epsilon)k_i$, it decodes the $K_i$ encoded blocks and obtains the $k_i$ original blocks. Upon retrieving requests from other peers, it generates new encoded blocks from these original blocks by a LT encoder based on the same robust Soliton distribution, and delivers these new encoded blocks to other receivers. The *rStream* protocol based on this *rateless-code recoding* is summarized in Table 1.

**Table 1: The *rStream* Protocol**

```
The streaming source: In the interval $[t_{i-1}, t_i]$ for broadcasting segment $s_i$ in the multimedia bitstream $\widetilde{M}$ of rate $r$
    For a peer $p$ streaming from the source at rate $x$:
    From $j = 1$ to $\frac{x}{r}K_i$
    1. Generate encoded block $B_j^p$ from the original blocks $b_1, b_2, \ldots, b_{k_i}$ by
        (1.a) Randomly choose the degree $d_j^p$ from the robust Soliton distribution;
        (1.b) Choose $d_j^p$ distinct original blocks uniformly at random and set $B_j^p$ to be exclusive-or of these blocks.
    2. Packetize $B_j^p$ into a packet with the header indicating the degree $d_j^p$ and neighbor indices.
    3. Deliver the packet to neighbor $p$.

A receiver: After receiving $K_i$ packets for segment $s_i$
    Decode to obtain the $k_i$ original blocks.
    To serve another receiver $q$ at rate $y$:
    From $j = 1$ to $\frac{y}{r}K_i$
    1. Generate encoded block $B_j^q$ from the original blocks $b_1, b_2, \ldots, b_{k_i}$ by
        (1.a) Randomly choose the degree $d_j^q$ from the robust Soliton distribution;
        (1.b) Choose $d_j^q$ distinct original blocks uniformly at random and set $B_j^q$ to be exclusive-or of these blocks.
    2. Packetize $B_j^q$ into a packet with the header indicating the degree $d_j^q$ and neighbor indices.
    3. Deliver the packet to $q$.
```

The following proposition proves the correctness of our recoding protocol.

**Proposition.** *The $k_i$ original blocks of segment $s_i$ in $\widetilde{M}$ can be recovered from any set of $(1+\epsilon)k_i$ encoded blocks with high probability, in a peer-to-peer streaming session implementing the protocol in Table 1.*

*Proof:* We present a brief outline of the proof. The encoded blocks a receiver receives for recovering segment $s_i$ are either encoded by the streaming source or recoded by an upstream peer, both from the same set of $k_i$ original blocks of $s_i$. Since all the encoders follow the same encoding steps and generate each block independently from any other one based on the same robust Soliton distribution, the encoded blocks are all unique as if they are produced by a same encoder. Thus, after collecting $(1+\epsilon)k_i$ encoded blocks from any upstream peers, a receiver can recover the $k_i$ original blocks with high probability. $\square$

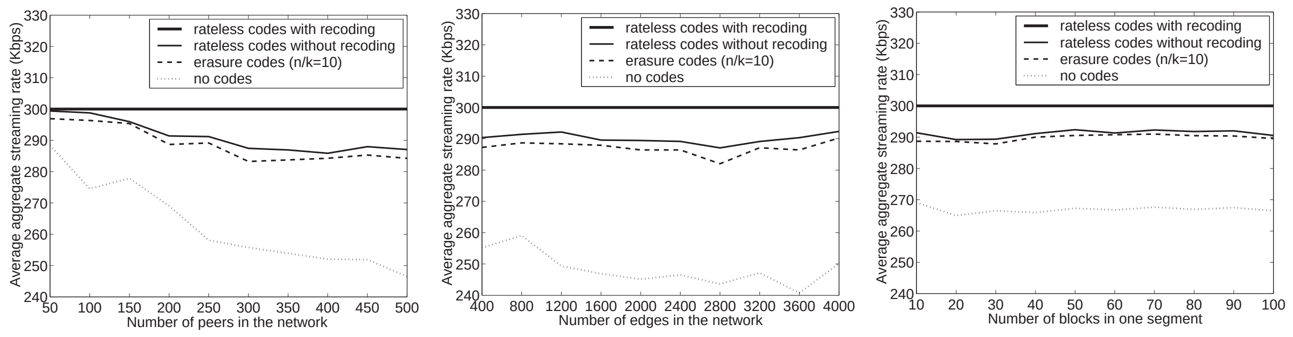
By guaranteeing the uniqueness of all the encoded blocks in the session, *rStream* successfully eliminates the need for content reconciliation. In Fig. 1 (d), for example, $S$ generates a potentially unlimited number of blocks $1'$, $2'$, $\ldots$ from the original blocks 1, 2, 3 and 4. The difference between Fig. 1(d) and (c) is that, rather than simply relaying received blocks, all peers recode the recovered original blocks and deliver the freshly encoded blocks. After receiving blocks $1'$, $2'$, $3'$ and $4'$ from $S$, $T_1$ decodes them to derive 1, 2, 3 and 4, then it encodes them again into $1^{(1)}$, $2^{(1)}$, $3^{(1)}$, $4^{(1)}$, $\ldots$, upon requests from $T_3$ and $T_4$. Similarly, $T_2$ recovers the four original blocks from $5'$, $6'$, $7'$ and $8'$, and recodes to obtain $1^{(2)}$, $2^{(2)}$, $3^{(2)}$, $4^{(2)}$, $\ldots$. Thus $T_3$ can safely retrieve unique encoded blocks $1^{(1)}$, $2^{(1)}$, $1^{(2)}$, $2^{(2)}$ from $T_1$ and $T_2$. After decoding, $T_3$ further recodes to obtain unique blocks for delivery: $1^{(3)}$, $2^{(3)}$, $3^{(3)}$, $4^{(3)}$, $\ldots$. Therefore, $T4$ is able to parallelly retrieve blocks $1^{(3)}$, $2^{(3)}$, $3^{(1)}$, $4^{(1)}$ without reconciliation between $T_1$ and $T_3$.

We now discuss the efficiency of the *rStream* protocol. As mentioned earlier, rateless codes are highly efficient. For the LT codes we employ, it takes on average $O(\ln(k/\delta))$ block exclusive-or operations to generate an encoded block from $k$ original blocks, and $O(k\ln(k/\delta))$ block exclusive-or operations to recover the $k$ original blocks from any $k + O(\sqrt{k}\ln^2(k/\delta))$ encoded blocks with probability $1-\delta$. Each block exclusive-or operation includes $L$ bitwise exclusive-or operations. For LT codes, it is more efficient in practice to have larger values of $L$ due to less overhead with bookkeeping operations. In our protocol, we choose $L$ to be a little less than the length of the packet payload. It is also not necessary for the rateless-code encoder and decoder to keep additional information of the coding method in use, as the traditional erasure codes require. A rateless-code encoder can generate the encoded blocks on the fly with the transmission process. To summarize, in *rStream*, recoding at each peer does not introduce much delay and computation overhead into the streaming session, but saves the high overhead for content reconciliation needed for every parallel retrieval.

The use of rateless codes in *rStream* protocol completely eliminates the need for content reconciliation when communicating with multiple upstream peers, while retaining all the advantages of traditional erasure codes such as Tornado codes, including loss resilience and decoding efficiency. We argue that the rateless recoding protocol further provides better failure tolerance for the streaming session. When a peer detects the failure of an upstream peer, it attempts to acquire more upload capacities from its remaining upstream peers. Since our rateless recoding protocol guarantees the uniqueness and usefulness of each delivered block in the system, the peer can actually make full use of the additional upload bandwidth to compensate for its missing streaming rate.

## 3. PERFORMANCE EVALUATION

We have carried out simulations to investigate the benefits of rStream in solving the reconciliation problem and maximizing bandwidth utilization. To this end, we compare four different coding schemes: no coding, fixed-rate erasure codes, rateless codes without recoding, and rateless codes with recoding. In our general setting, we simulate a live streaming session of a high-quality 300 Kbps multimedia bitstream from a streaming source with 10 Mbps of upload capacity. A realistic random network topology is generated with the BRITE topology generator [7] with power-law degree distributions. There are two classes of receivers in the network: ADSL/cable modem peers and Ethernet peers. ADSL/cable modem peers take $70\%$ of the total population with $1.5-4.5$ Mbps of download capacity and $0.6-0.9$ Mbps of upload capacity, and Ethernet peers take the other $30\%$ with both upload and download capacities of $8-12$ Mbps. The multimedia bitstream is partitioned



(a) Comparison in random networks of different network sizes (N) and fixed edge density (4N)

(b) Comparison in random networks of a fixed network size (200 peers) and different edge densities

(c) Comparison among different block numbers in a random network of 200 peers and 800 edges

**Figure 3: Comparison of average aggregate streaming rates in the system among $4$ different coding schemes: no codes, erasure codes with rate $n/k = 10$, rateless codes without recoding, and rateless codes with recoding.**

into fixed-length segments, each consisting of $50$ blocks. In each experiment with one coding scheme, we stream the media without content reconciliation among peers. We then eliminate the duplicated blocks obtained from different upstream peers and calculate the actual aggregate streaming rate at each receiver.

The comparison results are shown in Fig. 3. We can see in all the comparison scenarios, only rStream's rateless recoding scheme can actually achieve the end-to-end streaming rate of $300$ Kbps at all receivers. In other schemes, the aggregate streaming rates are reduced at different degradation levels, caused by the duplication in the received blocks. This also shows that rStream performs best with respect to bandwidth utilization. In our experiments with fixed-rate erasure codes, we notice that increasing the ratio of $n/k$ helps alleviate the conflicts at receivers. Nevertheless, this improvement is upper bounded by the results of the scheme of rateless encoding without recoding.

In Fig. 3(a), the comparison is made in networks of different numbers of peers (*network sizes*) and a fixed *edge density*, in which the number of edges is about four times the number of peers. We find that, under the other three schemes, the average aggregate streaming rates drop with the increase of network sizes. This is caused by more severe conflicts of blocks held by different peers in larger networks, where the total number of different blocks in each media segment is limited. When the edge density changes in a fixed-size network, the average aggregate streaming rates do not change significantly for each coding scheme, as shown in Fig. 3(b).

We also investigate the impact of the number of blocks per media segment on the block conflicts at the receivers (Fig. 3(c)). Varying the number from $10$ to $100$, we find the average aggregated streaming rates remain approximately the same for each coding scheme. Thus, by varying the number of blocks per segment, we are not able to alleviate the severity of block conflicts much in those coding schemes where content reconciliation is required.

## 4. CONCLUSION

We conclude this paper by reinforcing our strong argument that rateless codes are ideal companions to any peer-to-peer streaming solutions, and are orthogonal to any multimedia codecs, including H.264/AVC. A typical multimedia stream, such as an MPEG-4 or H.264 stream, can be treated as a bitstream demanding a constant bit rate, which can be segmented and treated by rateless codes. Using examples, analysis and simulation results, we have demonstrated that rateless codes represent an excellent solution to provide resilience to dynamics typically found in peer-to-peer networks, and to completely eliminate the need for content reconciliation. In ongoing work, we are working on the optimal peer selection and streaming rate allocation strategies that can be computed on-the-fly in a decentralized manner. We believe the combination of rateless codes and optimal peer selection will provide a complete solution towards winning the battle on all fronts of the peer-to-peer streaming challenge: dynamics, reconciliation, and bandwidth. We are also working towards a full-fledged implementation of peer-to-peer streaming based on rateless codes, as a first step towards large-scale deployment in the Internet, and towards making peer-to-peer streaming a reality.

## 5. REFERENCES

[1] J. Byers, J. Considine, M. Mitzenmacher, and S. Rost. Informed Content Delivery Across Adaptive Overlay Networks. In *Proc. of ACM SIGCOMM 2002*, August 2002.

[2] J. Byers, M. Luby, M. Mitzenmacher, and A. Rege. A Digital Fountain Approach to Reliable Distribution of Bulk Data. In *Proc. of ACM SIGCOMM 1998*, September 1998.

[3] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. SplitStream: High-Bandwidth Multicast in Cooperative Environments. In *Proc. of the 19th ACM Symposium on Operating Systems Principles (SOSP) 2003*, October 2003.

[4] J. Li. PeerStreaming: A Practical Receiver-Driven Peer-to-Peer Media Streaming System. Technical report, Microsoft Research MSR-TR-2004-101, September 2004.

[5] M. Luby. LT Codes. In *Proc. of the 43rd Symposium on Foundations of Computer Science*, November 2002.

[6] P. Maymounkov and D. Mazieres. Rateless Codes and Big Downloads. In *Proc. of the 2nd Int. Workshop Peer-to-Peer Systems (IPTPS)*, February 2003.

[7] A. Medina, A. Lakhina, I. Matta, and J. Byers. BRITE: Boston University Representative Internet Topology Generator. Technical report, http://www.cs.bu.edu/brite, 2000.

[8] V. N. Padmanabhan, H. J. Wang, P. A. Chow, and K. Sripanidkulchai. Distributing Streaming Media Content Using Cooperative Networking. In *Proc. of NOSSDAV 2002*, May 2002.

[9] A. Shokrollahi. Raptor Codes. In *Proc. of the IEEE International Symposium on Information Theory (ISIT) 2004*, June 2004.

[10] X. Zhang, J. Liu, B. Li, and T. P. Yum. Data Driven Overlay Streaming: Design, Implementation, and Experience. In *Proc. of IEEE INFOCOM 2005*, March 2005.