# RC-UDP: On Raptor Coding Over UDP For Reliable High-Bandwidth Data Transport

Abderrahmen Mtibaa*, Charles Good*, Satyajayant Misra*, David G. M. Mitchell†, Bhumika Parikh*
* Department of Computer Science, New Mexico State University
† Klipsch School of Electrical & Computer Engineering, New Mexico State University

***Abstract*—Data-driven and collaborative research has become the trend for today's scientific communities, resulting in large-scale datasets being shared and transported through networks every day. Most of these large data transfers use TCP sockets which are known to be limited in long-distance and high-bandwidth scenarios. UDP, on the other hand, while fast and efficient does not implement any reliability mechanisms. In this paper, we investigate the use of erasure coding techniques, namely fountain codes, on top of UDP to help high speed and reliable data transfer applications to attain high bandwidth in the face of packet losses. We propose RC-UDP, a Raptor Code over UDP framework that enables reliable data transfers for high bandwidth networks. We implement RC-UDP and evaluate its performance using computer simulation (ns-3) and real world testbed experimentations. We compare RC-UDP to HighSpeed and CUBIC TCP. Our results show that RC-UDP, which achieves up to $75\times$ time reduction while incurring minimum overhead, is beneficial when the network is subject to high congestion or packet drop rates.**

## I. Introduction

Scientific data is often extremely large in size and needs to be shared among multiple collaborative researchers, often residing in multiple geographical locations, that results in daily large-scale data transfers over long-distance networks. Such large data transfers create multiple issues and challenges that urged researchers to consider novel architectures, technologies, and transport protocols. The commonly-used Transmission Control Protocol (TCP) has limitations when used for long-distance and/or high-bandwidth networks [1]. While some researchers have focused on improving TCP's congestion control/avoidance algorithms [2], [3], we investigate the use of *rateless coding* [4] to ensure reliability over the "unreliable" User Datagram Protocol (UDP) as a means to increase the throughput in such high bandwidth networks. Thus, we allow as many packets as necessary to be transmitted to successfully recover the source message at the decoder.

A Forward Error Correction (FEC) code typically operates on a fixed block/packet length, with fixed rate, and can correct a number of errors up to a predefined maximum that depends on the encoder and decoder.*Rateless* or *Fountain codes*, however, generate a potentially infinite stream of encoded data, called *symbols*, in order to improve the recovery probability regardless of the error or loss rates applied to the original encoded symbols. Luby Transform (LT) codes [4], one of the first practical rateless coding schemes, incur an encoding/decoding cost of $O(k \log k)$, where $k$ is the original number of encoded symbols (file size). Raptor (rapid Tornado) codes [5] achieve linear time encoding and decoding ($O(k)$) by adding an outer low-density parity-check (LDPC) code.

In this paper, we propose *RC-UDP*, a Raptor Code over UDP framework that enables reliable data transfers for high-bandwidth networks. Our framework is implemented as a middleware that encodes *blocks* of data into an infinite encoded stream and sends them using UDP in order to increase the overall throughput and decrease end-to-end transfer delays. We implement an active acknowledgement mechanism at the decoder, wherein the decoder acknowledges the successful decoding of a received block. This acknowledgement in turn triggers the sender to send another encoded block. RC-UDP implements a block-by-block flow control which limits the flooding rate for a given block $i$ by estimating the number of packets dropped while transporting block $i-1$. We use standalone raptor encoding and decoding modules which we integrate into our RC-UDP protocol implementation. We evaluate RC-UDP using both ns-3 [6] simulation and real world testbed experimentation. We validate our implementation and test a large spectrum of parameters such as the block size, symbol lengths, and encoding matrix size. We also compare RC-UDP versus latest versions of TCP (CUBIC and high speed TCP). In this paper, we investigate the benefits and the drawbacks of using RC-UDP for high bandwidth networks using both the Energy Sciences Network (ESnet) [7] and random network topologies.

Our contributions include: (*i*) proposing RC-UDP, a framework that ensures reliable communication while maximizing the throughput and reducing data delivery latencies. We implement RC-UDP using a raptor code module that encodes all datagram messages; (*ii*) comparison of RC-UDP success rate and transfer delays with HighSpeed TCP performance using both network simulation and testbed experimentation; (*iii*) quantifying the overhead required to recover encoded messages at the decoder node; (*iv*) investigation of the tradeoff between overhead and latency performance of RC-UDP using real world and random topologies and a large spectrum of network parameters; and (*v*) demonstration of RC-UDP in the presence of high network congestion and/or packet drops.

## II. Background & Related Work

Many researchers have noted the slowness of TCP and its inability to take advantage of available bandwidth [3]. TCP enhancements [2], [3] have been proposed and various FEC schemes on top of UDP [8], [9], [10] have been considered. Given the limitations of fixed-rate based solutions to adapt to the network characteristics [11] and the inefficiency of random linear network coding [12] for wired unicast application such

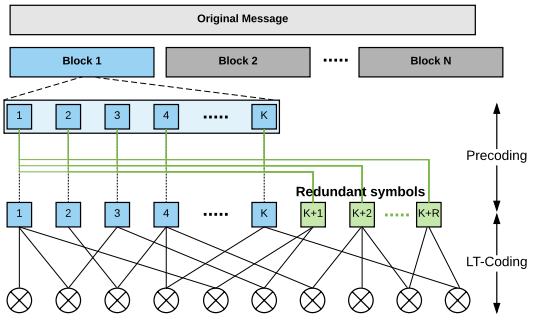



Fig. 1. Raptor coding phases (i) pre-coding, and (ii) LT-coding.

as high-bandwidth data transfers [13], rateless or fountain codes [8] have been largely investigated to ensure reliable communications in lossy networks [14], [15]. Rateless codes have been employed for both the packet erasure channel [10] and general wireless channels [9].

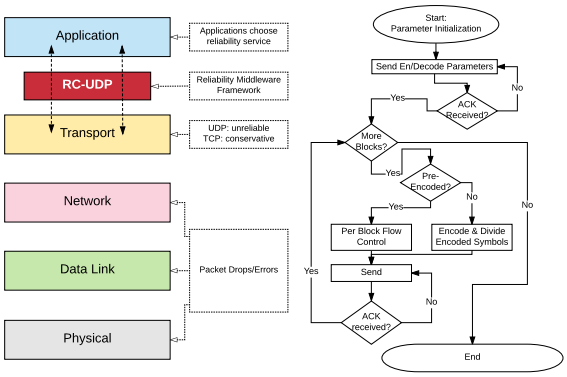
Rateless codes have been used to design new transport protocols using TCP without its congestion control [15], [16], or UDP [17]. Most of the raptor coding transport protocols were designed for small sized packet transfers [17], [14] or lossy wireless communications [17]. Molnar *et al.* [15] proposed DFCP, which uses raptor coding to replace TCP's congestion control, to achieve higher goodput performance under a wide range of packet loss rates and round-trip delay environments. However, TCP's flow control in DFCP uses sliding windows which limits the goodput as mentioned by the authors [15].

Most relevant to our research are (1) RCDP, a raptor-based content delivery protocol was designed for intelligent transport systems wireless communications where messages are small and paths are short [17], and (2) Chong *et al.* [14] proposed a UDP based raptor transport protocol which implements a stop-and-wait protocol making it very slow for high bandwidth network communications. Compared to these approaches, our work implements reliable and fast transfers of large data blocks while maintaining low overhead in the network. It's block-by-block flow control helps a full and quick utilization of the available resources in the network, without incurring performance degradation. To the best of our knowledge, our work is the first to implement, simulate, and experimentally evaluate, using a large spectrum of parameters, raptor coding over UDP as an alternative transport protocol.

## III. Rateless Coding: Preliminaries

For a given vector $(x_1, x_2, \ldots, x_k)$ of $R$ source symbols, a fountain encoder produces a potentially infinite stream of encoded symbols $(y_1, y_2, \ldots)$. When the decoder has received enough symbols/packets (typically slightly larger than the original message size), the message can be recovered. When packet losses occur in the network, the receiver node is able to recover the original message by collecting any sufficiently large subset of the encoded stream.

Let $s$ be a source node (which we also refer to as the encoder) that transmits a message $m$, of size $S_m$, to a destination node $d$ (the decoder). In raptor codes, the original message $m$ is partitioned into a stream of $N$ blocks $(b_1, b_2, \ldots, b_N)$ of




(a) RC-UDP framework (b) RC-UDP process

Fig. 2. Overview of RC-UDP.

fixed size $B$, where $N \times B = S_m$. Each block $b_i$ consists of $K$ symbols of fixed size $L$ bits. The source node encodes each block $b_i$ in two phases as shown in Fig. 1: (*i*) pre-coding, and (*ii*) LT coding. The pre-coding phase consists of encoding the $K$ original symbols by adding parity symbols via a pre-specified (outer) code, usually an LDPC code [18]. This process generates $R$ redundant symbols, called repair symbols. All $K + R$ symbols are then encoded according to the (inner) LT code using an exclusive-or (XOR) operation following a pre-specified distribution [4].

## IV. The Raptor Coding over UDP Framework

### A. Overview

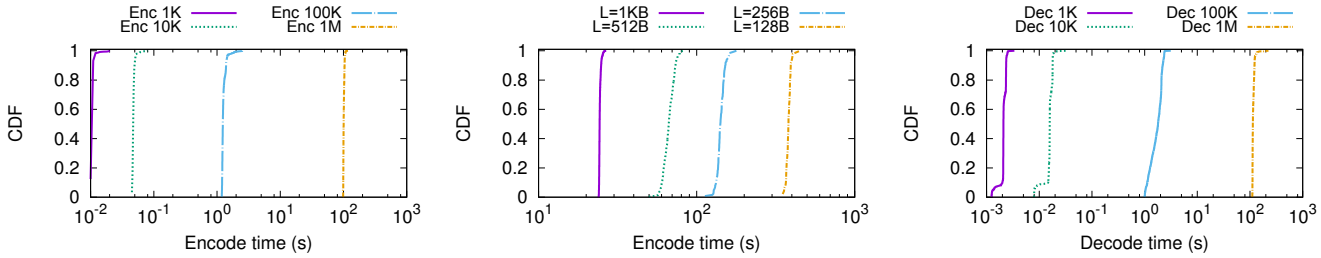
While TCP, and its enhancements for high-bandwidth networks, such as CUBIC [2] and HighSpeed TCP [3], implement multiple features to ensure reliable end-to-end communication between end hosts, their performance degrades in environments where packet drops are transient or a bufferbloat [19] occurs in the network of fast pipes and large queuing delays that overwhelm TCP's congestion control.

RC-UDP resides as a middleware between the application and the transport layers as shown in Fig. 2(a). RC-UDP is designed for efficient and reliable large-scale data transfers such as scientific data collaborative sharing applications. The coding scheme ensures end-to-end reliability using an unreliable but efficient UDP transport protocol.

### B. RC-UDP Protocol Design

RC-UDP implements a rateless coding which consists of sending a stream of encoded symbols to the destination node until it receives an explicit acknowledgement (ACK) indicating successful reception and decoding.

*a) RC-UDP Framework:* Fig. 2(b) details the RC-UDP encoding process. As soon as the application obtains data to transmit, it chunks it into blocks and starts the encoding/sending process block-by-block. The encoder then sends the set of parameters needed for the encoding/decoding procedure. These parameters consist of the total data length $S_m$, number of source blocks $N$, number of symbols per block $K$, block size $B$, and symbol length $L$. When the decoder acknowledges the reception of the parameters, the encoder




(a) Encoding time (1024 bytes symbol size) (b) Encoding time (500 KB block size) (c) Decoding time (1024 bytes symbol size)

Fig. 3. Encoding and decoding time for: (a) and (c) different file sizes, and (b) different block sizes.

starts encoding and sending block-by-block until all data is received and acknowledged by the decoder.

For any given block, RC-UDP generates a list of encoded symbols that are partitioned into a set of $r$ symbols per packet, where $r$ is set according to the maximum transmission unit (MTU). When the decoder receives $(1+\epsilon)K$ symbols from a given block, it acknowledges (ACK) the reception of the block and decodes it. When the encoder receives an acknowledgement of the successful reception of a given block, it checks if there are more blocks to encode/send. If so, it verifies if the next block is already pre-encoded and the process continues as shown in Fig. 2(b).

*b) Block-by-block Flow Control:* The encoder implements a block-by-block flow control consisting of estimating a waiting delay between blocks. If no pre-encoded blocks exist then the encoder waits until the next block is encoded and sends it with no delay. If a block is already pre-encoded at the time it receives an ACK (acknowledging the reception of the previous block) then the encoder computes a waiting time $T$ which aims to reduce the flood of UDP messages if the network is congested. $T$ is computed based on an estimation of the number of packets dropped during the transfer of the previous block, $\Delta P$; upon reception of ACK, the source compares the number of packets transmitted, $P_s$, and the number of packets received by the destination $(1+\epsilon)K/r$; therefore, $\Delta P = P_s - (1+\epsilon)K/r$. We use $T = \Delta P \cdot \delta t$, where $\delta t$ is a waiting delay per dropped packet. In this paper, we set $\delta t = 0.3$ms. Tuning $\delta t$ is not investigated as it is beyond the scope of this paper.

### C. Encoder & Decoder Modules

The rateless encoding and decoding scheme affects the efficiency of RC-UDP. Therefore, we first deploy and test a standalone rateless raptor encoding and decoding python modules in [14]. These modules are integrated into our RC-UDP algorithm as described earlier.

The standalone raptor encoder module uses encoding parameters that are set by users (or applications), such as the length of the source symbols and the number of overall symbols, to generate a set of encoded symbols, and employs a randomly generated degree distribution used by the LT encoder. The decoder module converts the sparse LDPC matrix into a (potentially) dense generator matrix using Gaussian Elimination in order to determine the repair symbols.

A set of preliminary experiments were run to measure the encoding and decoding times using an Intel Core i7-6700K CPU @ 4.00GHz PC with 64GB of RAM, running Ubuntu 16.04 LTS OS. Note that our results do not evaluate raptor coding performance in general, but rather test its functionality under a given set of parameters. In addition, time requirements can vary drastically depending on the computation resources and could be significantly improved using GPU or a hardware implementation of raptor coding.

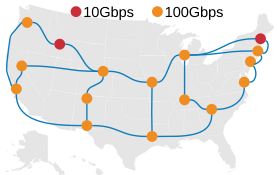



Fig. 4. ESnet topology.

We vary the encoded file size and the symbol size to plot a distribution of 1000 different encoding and decoding times in Fig. 3. First we fix the symbol size $L = 1024$B, and plot in Fig. 3(a) the encoding time distribution for file sizes $S_m =$ 1KB, 10KB, 100KB, 1MB, and 10MB. While in theory raptor codes achieve linear time encoding and decoding, we found that the time to encode is empirically "not linear" using this module implementation. Moreover, when we fix the file size and vary the symbol size $L$ from 128 bytes to 1KB (Fig. 3(b)), we found that the time is inversely proportional to $L$. Indeed, the larger $L$, the less symbols are available to encode (LDPC and LT encoding). Note that we choose not to consider larger symbol sizes in order not to exceed the MTU.

In Fig. 3(c), we plot the decoding time distribution of 1000 trials with 0 to 10% random symbol drop ratio. Unexpectedly, we found that decoding is slightly faster than encoding on average, but decoding of large files can reach $2\times$ the encoding time of the same file. We also note that block size $B = 1$MB and symbol size $L = 1$KB appear to be the best performing block and symbol sizes for both encoding and decoding time. We therefore fix them as constant for the rest of this paper and evaluate our RC-UDP using both simulation (Section V) and real world testbed implementation (Section VI).

## V. Simulation Results

First, we evaluate RC-UDP via computer simulation using the network simulator, ns-3 [6]. We implement RC-UDP in ns-3 as described in the previous section. In this simulation experiment, we did not implement the encoding and decoding modules, but used the time distributions in Fig. 3 as waiting delays to perform such operations. Note that we discuss a real world experimental study of RC-UDP in the next section.

TABLE I
NETWORK TOPOLOGIES.

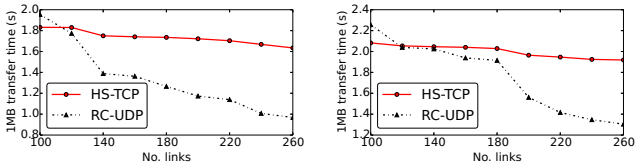
| Topology | # Nodes | # Links | Capacity |
|---|---|---|---|
| ESnet | 16 | 20 | [0.5,5] Mbps |
| Random (ER) | 50 | $[100, 120, \cdots, 280]$ | [10,100] Gbps |

### A. Metrics & Parameters

We consider two types of network topologies to evaluate RC-UDP performance: ESnet and random graph (ER) networks, as summarized in Table I. We emulate the ESnet as shown in Fig. 4, with 16 nodes and 20 links spanning the USA. We assign a capacity of 100Gbps for most of the links except the links connected to the Boston and Boise nodes (highlighted), which have a capacity of 10Gbps [7]. The second topology considered consists of a network of 50 nodes connected randomly, with the number of links from 100 to 280. This variation allows us to investigate the impact of density and congestion on the RC-UDP performance. For each topology, we use HighSpeed-TCP (HS-TCP) as a benchmark method for comparison to quantify the gain achieved with RC-UDP. HS-TCP is designed to overcome TCP's limitations for high-bandwidth file transfer applications [3]. Note that we use CUBIC TCP for our real world experimentation in the next section. In each trial, we consider three simultaneously communicating source/destination pairs where sources send files of size $S_m$ chunked into blocks of size $B$. We vary the size of the message sent by the application while fixing the data payload to 1 symbol of size $L = 1024$ bytes per packet. We set the queue sizes at the nodes to 100 packets in ER and 10000 in ESnet, where packets will be dropped when arriving at nodes with a full queue. We consider $\epsilon = 0.01$ which has been shown more than 99.9% success decoding ratio [8].

We consider two evaluation scenarios (with and without packet drops): *a) Scenario 1 (No Drops):* In this scenario we operate our networks in "low" congestion mode where link capacities are adequate to send small bursts. We set all ER link capacities to 5Mbps, thereby providing enough bandwidth for the considered $B = 1$MB blocks sent by all RC-UDP nodes. We note that in the case of ESnet, links are pre-configured and remain unchanged as shown in Fig. 4. *b) Scenario 2 (Drops):* We introduce fixed rate packet drops to the previous scenario to evaluate the response of both RC-UDP and HS-TCP. The `rate error model` in ns-3 is applied to random wired point-to-point links in the network to create packet drops. The packet drop rate varies from 1, 3, 5, to 10%. Results for the 3% packet drops are selected as an exemplary sample, but similar behavior was registered for the other drop rates with increased RC-UDP gains when the packet drop rate increases.

We perform ten different simulation runs for each parameter set and present the average results. We measure; (1) the *average block transfer time*, defined as the time between sending the first byte by the encoder to the time required to collect enough data to start the decoding process of a given block ($(1+\epsilon)K$ symbols). We choose to normalize the transfer time per $B = 1$MB (i.e., block size) to highlight the impact of message size on the time to transfer the same block size; and (2) the *overhead* transmitted by RC-UDP into the network.




(a) $S_m = 1$MB (b) $S_m = 30$MB

Fig. 5. Avg. transfer time per block of $B = 1$MB (ER: Scenario 1, 5Mbps Links no drops).

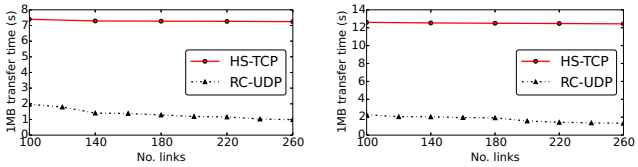
This is measured as the additional data sent by the encoder node in order to receive $(1+\epsilon)K \cdot N$ symbols by the decoder.

### B. Results

*Transfer Time (ER Network):* Fig. 5 compares the average end-to-end transfer time of $B = 1$MB blocks between source and destination pairs as a function of the number of links in the networks using Scenario 1 (5Mbps links). We vary the total data size, and present $S_m = \{1, 30\}$MB as shown in Fig. 5(a) and 5(b), respectively. We find that, while transfer delays decrease when the number of links increases (creating shortcuts and faster routes for each transmission flow), RC-UDP achieves higher transfer delay gain ($\approx 35\%$ reduction) when the number of links in the network increases. Indeed, when the paths become shorter the average number of concurrent flows sharing links decreases, which makes RC-UDP send bursts faster with minimum errors/drops. While RC-UDP transfers data faster than HS-TCP for most of the considered data sizes, HS-TCP seems to outperform RC-UDP in sparse networks (small number of links) due to its active flow and congestion control in the presence of concurrent flows.

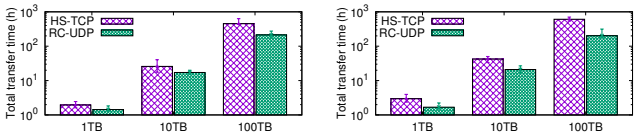
Fig. 6 plots the transfer time for RC-UDP and HS-TCP when we apply 3% packet drops in the ER network (Scenario 2). We find that the gain with RC-UDP is considerable, reaching up to 90% delay reduction compared to HS-TCP. In fact, HS-TCP suffers significantly from such packet drops, whereby its window remains very small resulting in a reduction of the throughput. However, RC-UDP continues sending packets at the same rate while the receiver employs FEC to recover the original message with little or no delay when compared to the previous scenario with no drops. Note that further gains are observed when the drop rate increases.

*Transfer Time (ESnet):* We pre-encode data and emulate blocks of encoded data of size $B = 100$GB. In this experiment, we do not perform decoding, instead we assume that the receiver will successfully decode the original block if it receives $(1+\epsilon) \cdot 100$GB. We plot, in Fig. 7, the average (using 4 runs) total time to transfer different file sizes $S_m = 1, 10,$ and 100TB using (a) Scenario 1, as depicted in Fig. 7(a), and (b) Scenario




(a) $S_m = 1$MB (b) $S_m = 30$MB

Fig. 6. Avg. transfer time per block of $B = 1$MB (ER: Scenario 2, 5Mbps Links, 3% pkt drops).




(a) scenario 1 (no drop) (b) scenario 2 (3% drops)

Fig. 7. ESnet average transfer delays of 1, 10, and 100TB files ($B = 100$GB).

2, as depicted in Fig. 7(b), where we introduce 3% packet drops. While RC-UDP outperforms HS-TCP in both scenarios, we see that when packet drops occur, HS-TCP performance drops considerably resulting in bigger gains (up to 3.5×) using RC-UDP. Note that the gain of using RC-UDP is up-to 6.7 days (for 100TB transfers with 3% drops) which represents a major improvement for large-scale high bandwidth scientific data transfers. Similar to the ER network results, we find that the RC-UDP performance does not decrease as much when packet drops occur, unlike HS-TCP.

*Overhead:* As described in Section 4, an explicit acknowledgment mechanism was implemented to notify the encoder of successful block decoding and to reduce the overhead resulting from the infinite stream of data. Fig. 8 quantifies the exact overhead (measured as a percentage of the block size) introduced to the network. We find that Scenario 1 uses, on average, only 3.85% overhead, often due to minor packet drops and delays to receive the ACK message. Notably, Scenario 2 exhibits similar overhead as Scenario 1. This indicates that the packet drops introduced in Scenario 2 do not noticeably impact the overhead as they are transient. HS-TCP, however, treats these transient packet drops as a sign of congestion and reduces its throughput as shown in Fig. 6.

## VI. Testbed Experiments

### A. Experimental Setup

We implement our proof-of-concept RC-UDP prototype as a client-server application and evaluate its performance using a real world network testbed consisting of a sender node (encoder), a receiver node (decoder), and a traffic shaper connecting both entities as shown in Fig. 9. A real testbed permits full control over the network parameters which we can decouple and test their impact on RC-UDP performance. We implement the traffic shaper using the Linux command `qdisc` which directs the kernel to enqueue any received packet before forwarding it. We implement `pfifo` (queuing discipline), `tbf` (Token Bucket Filter), and `netem` (network emulation) to control traffic at the shaper node.

We use three Dell laptops with Intel Core 2 Duo 1.66 GHz Dual Core Processor and 2GB RAM running Ubuntu 14.04 and connected via a LAN switch as shown in Fig. 9. We then

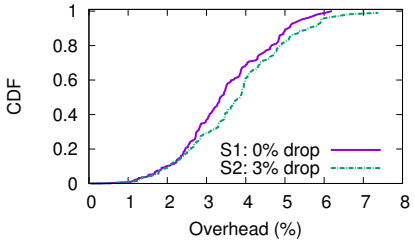



Fig. 8. RC-UDP overhead results using Scenarios 1 & 2 (S1 and S2).

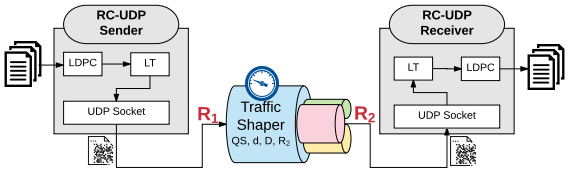



Fig. 9. Schematic diagram of the testbed set-up.

set static routing to force all traffic between the sender and receiver to pass through the traffic shaper. While we do not control the sending rate ($R_1$), we set a delay $D$, packet drops $d$, queuing $QS$, and rate control ($R_2$) at the traffic shaper node as shown in Table II. We set the maximum burst (the TBF bucket size) to 10KB.

### B. Experimental Results

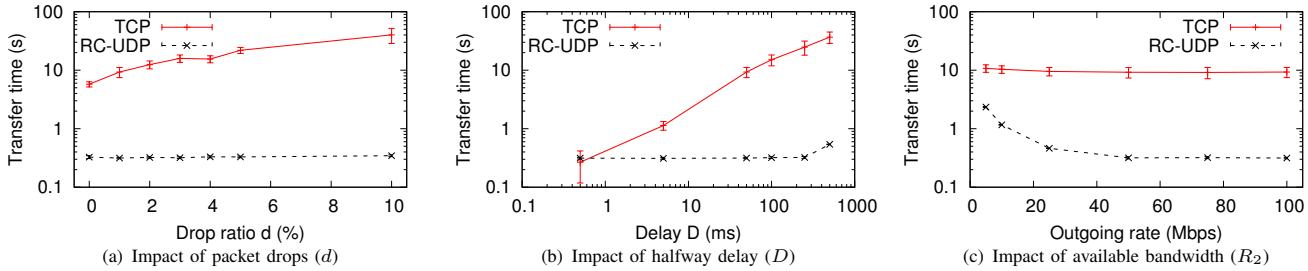
We perform multiple runs (with a two second waiting delay between each run to clear out the packets of the last run from the network) and measure the average total transfer time of $S_m = 1$ and 10MB files using (1) CUBIC TCP (which we will refer to as TCP), and (2) RC-UDP without considering the encoding and decoding times. We note that for large data transfers (e.g., those for collaborative science) it is normal to assume that the data is pre-encoded and decoding happens using fast machines at either ends. The encoding and decoding processes can be easily implemented in GPUs and can be highly parallelized and sped-up.

Fig. 10 plots the average over 50 different runs for the total transfer time of a 1MB file (a single block) along with the standard deviation (as error bars). Varying drop rates, delay, and $R_2$ are shown in Figs. 10(a), 10(b), and 10(c), respectively. We found that RC-UDP outperforms TCP in most of the experiments. Our testbed experiment allowed us to isolate parameters to study the impact of each one on the performance. Similar to the simulation results, when packet drop rates increase, RC-UDP's performance is not impacted compared to that of TCP, which gets slower by a factor of 75× with 10% drop rate. Similar behavior is registered when the network delay $D$ (RTT) increases (see Fig. 10(b)). The TCP transfer time increases exponentially due to its RTO estimation [20], while RC-UDP recovers 40× faster when $D = 500$ms. TCP outperforms RC-UDP by only 0.01ms when $D = 0.5$ms (representing a fast link(s) between source and destination pairs or a LAN connecting the pairs).

In Fig. 10(c), we show that TCP and RC-UDP transfer times decrease when $R_2$ increases beyond 50Mbps (50% of $R_1$). For $R_2 > 50$Mbps, the large queue size will allow all packets to be delivered with slight delays. Here, RC-UDP outperforms TCP in all considered rates. In fact, while TCP reduces its congestion window when $R_2$ decreases, resulting in longer transfer times, RC-UDP will register more drops resulting in

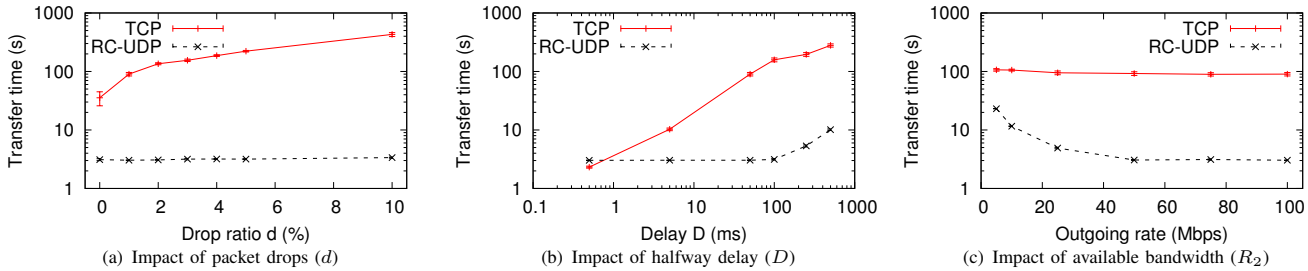
TABLE II
Experimentation Parameters.

| Parameter | Label | Range | Nominal Value |
|---|---|---|---|
| Bandwidth (Mbps) | $R_2$ | [5,10,25,50,75,100] | 100 |
| Drop ratio (%) | $d$ | $[0, 1, \ldots, 5, 10]$ | 1 |
| Delay (ms) | $D$ | [0.5,5,50,100,250,500] | 50 |
| Queue size (KB) | $QS$ | 130 | 130 (≈ 10pkt) |




(a) Impact of packet drops ($d$) (b) Impact of halfway delay ($D$) (c) Impact of available bandwidth ($R_2$)

Fig. 10. 1MB file transfer experimental results; all y-axis are in log-scale; log/log-scale in (b).




(a) Impact of packet drops ($d$) (b) Impact of halfway delay ($D$) (c) Impact of available bandwidth ($R_2$)

Fig. 11. 10MB file transfer experimental results; all y-axis are in log-scale; log/log-scale in (b).

longer transmissions and more overhead to recover. However, RC-UDP recovers faster and achieves $5\times$ reduction in latency at $R_2 = 5$Mbps. Moreover, we can see that RC-UDP registers more consistent performance compared to TCP (registering higher standard deviation values in most of the experiments).

We perform 10 runs, and measure the average transfer times of 10MB files using TCP and RC-UDP in Fig.11. We notice similar performance results as shown in Fig. 10 with slightly slower transfer delays using TCP under $D = 0.5$ms.

## VII. Concluding Remarks

This paper investigated the benefits (latency gain) and the costs (overhead) of RC-UDP as a novel raptor code based middleware proposed for reliable data transfers for large amounts of data over high speed networks. The performance was evaluated with/without congestion and in the presence or absence of packet drops. We showed that although, RC-UDP does not achieve good overhead-latency performance when abundant bandwidth is available (compared to HighSpeed TCP), it results in significant latency gains with minimum overhead costs when the network is subject to congestion and/or packet drops. Our study represents the first step towards implementing a real-world platform for designing and testing rateless coding schemes for data transfer applications.

## Acknowledgments

This material is based upon work supported in part by the National Science Foundation under Grant Nos. CCSS-1710920, 1345232, 1248109, 1719342, and in part by NASA Training Grant NNX15AL51H. The information reported here does not reflect the position or the policy of the federal government.

## References

[1] V. Jacobson, R. Braden, and D. Borman, "TCP extensions for high performance," 1992.

[2] S. Ha, I. Rhee, and L. Xu, "Cubic: a new TCP-friendly high-speed TCP variant," *ACM SIGOPS Operating Systems Review*, vol. 42, no. 5, pp. 64–74, 2008.

[3] S. Floyd, "Highspeed TCP for large congestion windows," 2003.

[4] M. Luby, "LT codes," in *The 43rd Annual IEEE Symposium on Foundations of Computer Science*, pp. 271–280, IEEE, 2002.

[5] A. Shokrollahi, "Raptor codes," *IEEE Transactions on Information Theory*, vol. 52, no. 6, pp. 2551–2567, 2006.

[6] G. F. Riley and T. R. Henderson, "The ns-3 network simulator," *Modeling and tools for network simulation*, pp. 15–34, 2010.

[7] W. Johnston, "Esnet: Advanced networking for science," *SciDAC Review*, vol. 4, p. 48, 2007.

[8] D. J. MacKay, "Fountain codes," *IEE Proceedings-Communications*, vol. 152, no. 6, pp. 1062–1068, 2005.

[9] M. Luby, T. Gasiba, T. Stockhammer, and M. Watson, "Reliable multimedia download delivery in cellular broadcast networks," *IEEE Transactions on Broadcasting*, vol. 53, no. 1, pp. 235–246, 2007.

[10] Z. Xu, C. Yang, Z. Tan, and Z. Sheng, "Raptor code-enabled reliable data transmission for in-vehicle power line communication systems with impulsive noise," *IEEE Communications Letters*, vol. PP, no. 99, pp. 1–4, 2017.

[11] G.-H. Gho, L. Klak, and J. M. Kahn, "Rate-adaptive coding for optical fiber transmission systems," *Journal of Lightwave Technology*, vol. 29, no. 2, pp. 222–233, 2011.

[12] P. Wu and N. Jindal, "Coding versus ARQ in fading channels: How reliable should the phy be?," *IEEE Transactions on Communications*, vol. 59, no. 12, pp. 3363–3374, 2011.

[13] C. Fragouli, J.-Y. Le Boudec, and J. Widmer, "Network coding: an instant primer," *ACM SIGCOMM Computer Communication Review*, vol. 36, no. 1, pp. 63–68, 2006.

[14] Z.-K. Chong, H. Ohsaki, B. Ng, B.-M. Goi, H.-T. Ewe, and S.-R. Chong, "Improving reliable transmission throughput with systematic random code," in *41st Conference on Local Computer Networks (LCN)*, pp. 539–542, IEEE, 2016.

[15] S. Molnár, Z. Móczár, A. Temesváry, B. Sonkoly, S. Solymos, and T. Csicsics, "Data transfer paradigms for future networks: Fountain coding or congestion control?," in *IFIP Networking Conference, 2013*, pp. 1–9, IEEE, 2013.

[16] Z. Móczár, S. Molnár, and B. Sonkoly, "Multi-platform performance evaluation of digital fountain based transport," in *Science and Information Conference (SAI), 2014*, pp. 690–697, IEEE, 2014.

[17] M. Báguena, C.-K. Toh, C. T. Calafate, J.-C. Cano, and P. Manzoni, "RCDP: Raptor-based content delivery protocol for unicast communication in wireless networks for ITS," *Journal of Communications and Networks*, vol. 15, no. 2, pp. 198–206, 2013.

[18] R. Gallager, "Low-density parity-check codes," *IRE Transactions on Information Theory*, vol. 8, no. 1, pp. 21–28, 1962.

[19] J. Gettys and K. Nichols, "Bufferbloat: Dark buffers in the internet," *Queue*, vol. 9, no. 11, p. 40, 2011.

[20] P. Sarolahti, M. Kojo, and K. Raatikainen, "F-RTO: an enhanced recovery algorithm for TCP retransmission timeouts," *ACM SIGCOMM Computer Communication Review*, vol. 33, no. 2, pp. 51–63, 2003.