Báguena Albaladejo, M.; Toh, CK.; Tavares De Araujo Cesariny Calafate, CM.; Cano Escribá, JC.; Manzoni, P. (2013). RCDP: Raptor-based Content Delivery Protocol for unicast communication in wireless networks for ITS. Journal of Communications and Networks. 15(2):198-206. doi:10.1109/JCN.2013.000033.

RCDP: Raptor-based Content Delivery Protocol for unicast communication in wireless networks for ITS

Miguel Báguena, C. K. Toh, Carlos T. Calafate, Juan-Carlos Cano, Pietro Manzoni

Abstract: **Recent advances in Forward Error Correction (FEC) coding techniques were focused on addressing the challenges of multicast and broadcast delivery. However, FEC approaches can also be used for unicast content delivery in order to solve TCP issues found in wireless networks. In this paper, we exploit the error resilient properties of Raptor codes by proposing RCDP - a novel solution for reliable and bidirectional unicast communication in lossy links that can improve content delivery in situations where the wireless network is the bottleneck. RCDP has been designed, validated, optimized, and its performance has been analyzed in terms of throughput and resource efficiency. Experimental results show that RCDP is a highly efficient solution for environments characterized by high delays and packet losses making it very suitable for Intelligent Transport System (ITS) oriented applications since it achieves significant performance improvements when compared to traditional transport layer protocols.**

Index Terms: **Application-layer FEC; Raptor codes; unicast content delivery; testbed.**

I. Introduction

Wireless channels are characterized by low signal levels, multipath interferences, fading signal, etc. that tend to reduce transmission throughput. This has motivated researchers to solve the specific problems that they produce. Most research effort in this area have focused on allowing existing wired network protocols to operate without changes, or on creating new protocols that are compatible with the already existing ones. Therefore, a trade off between compatibility and performance is established where the former usually has the most importance. However, there are scenarios where compatibility is not critical but performance is of capital importance. In these scenarios other ways of action must be explored.

The classic TCP [16] protocol is a perfect example of the performance losses experienced when attempting to use classic wired network protocols and applications over wireless networks. TCP uses packet losses for congestion detection because this is the main reason for packet losses in wired networks. However, in wireless networks, channel problems are the most frequent cause for loss. This characteristic means that TCP is unable to efficiently use the whole bandwidth available in the channel. The research community [14], [9], [6], [10] has proposed different strategies to adapt TCP to wireless networks, being the main trend to avoid retransmissions since such strategies generally does not scale well in large deployments, and the delay introduced by retransmission adversely affects performance.

On the other hand, Application Layer Forward Error Correction (AL-FEC) is a coding technique that allows generating virtually infinite recovery symbols which can be used to recover data at destination even if part of the data is lost. The AL-FEC strategy differs from other FEC approaches in that it operates at the application layer, and so no changes are required at the lower network layers. This technique has been used mostly for broadcasting and multicasting purposes, due to the effectiveness of fountain coded data at recovering missing data with a minimum overhead.

In this paper we propose RCDP, a unicast content delivery protocol based AL-FEC. It uses Raptor codes, a particularly efficient class of AL-FEC codes, to create a solution nearly immune to channel losses. Basically, RCDP relies on packet trains to estimate the end-to-end capacity instead of using a window based rate control like TCP. Data are partitioned into independent blocks, which are protected by as many FEC symbols as necessary for successful block recovery at the receiver. Through a real implementation and testbed tests we validate our solution, proposing optimizations at different levels.

We have organized this paper as follows: in section II we present related work. In section III we explain our protocol attending to its main characteristics. In section IV we propose new improvements for our RCDP protocol. In section V we perform the intensive set of proofs that evaluates the behavior of the protocol. Finally section VI concludes the paper.

II. Related Work

Many proposals to mitigate TCP problems or to expose an alternative protocol to TCP were proposed recently. Those techniques can be classified [2] into five categories: (i) link-layer solutions, (ii) split-connection solutions, (iii) TCP-enhancements, (iv) MANET-specific proposals and (v) FEC based solutions.

In terms of link layer solutions, the AIRMAIL protocol [1] combines both retransmission and error correction to improve performance; the Snoop protocol [3] relies on an agent to detect channel losses; additionally, Tulip [15] enforces retransmission acceleration at the MAC level.

With respect to split-connection approaches, which divide each TCP connection into two separated ones, Mobile TCP [4] has a three-layered structure which routes, reconnects, and controls the transmission rate; Wireless-TCP [20] adopts a different approach, avoiding the use of a window-based flow control.

Concerning those solutions that enhance the original TCP implementation, TCP SACK [14] informs the sender node about packet loss by providing more details than TCP; SMART [9]

M. Báguena is with the Department of Computer Engineering, Universitat Politècnica de València, Camino de Vera S/N, 46022, Spain, email: mibaal@upvnet.upv.es

C. K. Toh is with the National Tsing Hua University, Taiwan, email: ck_away@hotmail.com

C. T. Calafate, J.C. Cano and P. Manzoni are with the Department of Computer Engineering, Universitat Politècnica de València, Camino de Vera S/N, 46022, Spain, emails: {calafate, jucano, pmanzoni}@disca.upv.es

1229-2370/03/$10.00

combines the Go-Back-N approach and the selective ACK; Caceres and Iftode [5] propose a fast retransmission solution specifically focused on mobile communications.

In terms of MANET-specific proposals, TCP-F [6] uses RFN and RNN packets to stop and start packet transmission, while Ad-hoc TCP [10] also defines states in the sender; both are examples of research efforts specifically focused on improving TCP performance in MANETs.

Finally, focusing on FEC-based solutions, Luby et al. [11] propose a solution for reliable file delivery over mobile broadcast networks, concentrating on Raptor codes for Multimedia Broadcast and Multicast Services (MBMS) within the scope of the 3GPP specification. Authors emphasize on the goodness of the solution, and consider that Raptor codes are applicable to other scenarios such as video broadcasting over the Internet and peer-to-peer distribution. Overall, authors predict that, with the availability of powerful and low-complexity Raptor codes, many innovative applications and services are enabled in a very efficient and reliable manner. In this paper, we adopt these guidelines, although relying on Raptor codes for unicast content delivery instead.

Our proposal differs from previous solutions by addressing reliable two-way communications following a completely novel approach. In particular, our solution is based on a novel transport protocol which relies on Forward Error Correction to completely avoid retransmissions, along with an end-to-end bandwidth estimation technique to perform rate control. The solution we offer does not require intermediate nodes to actively participate in the process, nor introducing any hardware changes. In terms of implementation, no windowing or retransmission control has to be performed, which simplifies the tasks on both transmitter and receiver sides. To the best of our knowledge, no similar solution has been proposed so far that offer efficient and robust content delivery while supporting reliable and bidirectional communications.

III. The RCDP protocol

Nowadays, there are a lot of application layer protocols used for content delivery purposes, such as HTTP [7], FTP [17], and RTP [18]. When end-to-end reliability is required, most solutions rely on the TCP protocol at the transport layer because it is the most widely used, from operating systems to specific applications. However, when attempting to deliver contents over wireless networks, TCP-based solutions suffer from low performance since TCP is unable to distinguish whether the packet losses detected are due to network congestion or channel-related problems. Thus, efficient content delivery solutions should be sought to optimize performance in wireless environments.

To achieve this goal, we introduce our novel Raptor-based Content Delivery Protocol (RCDP). RCDP is a full-duplex content delivery solution which encompasses sending and receiving processes at both client and server sides. To achieve an error-resilient solution, RCDP combines the use of the UDP protocol at the transport layer with an AL-FEC strategy. The use of an AL-FEC strategy allows the creation of a content delivery solution which is nearly immune to packet losses, and also avoids the well known TCP problems in wireless networks. In particular, RCDP's AL-FEC relies on Raptor codes. This encoding technique allows us to recover the original data even when part of the information is lost. This behavior, mapped into a computer network, makes us able to ignore packet losses completely. Therefore, the original information is recovered through newly incoming packets, not requiring any information to be retransmitted by the sender. Note that such characteristic is common to all fountain codes [13], being Raptor codes [19] a particularly efficient FEC scheme within this group.

Another important feature of RCDP is the use of UDP to implement a rate control strategy that avoids TCP-like window based rate control. Instead, RCDP generates packet trains with a very regular pattern, and uses an end-to-end bandwidth measurement strategy to determine the available bandwidth. This strategy allows the receiver to detect changes on the channel's bandwidth simply by measuring the time differences between consecutive packet arrivals. The receiver can then send this information back to the sender, allowing it to adjust its sending rate to the most appropriate value in order to maximize throughput. Note that this strategy is not applicable to any broadcast/multicast based content delivery scheme previously proposed [11].

Another important difference between RCDP and previous solutions based on Raptor codes proposed for multicast/broadcast information dissemination is the protocol symmetry on both sides in the communication process. In previous proposals, each endpoint only assumes one role: sender or receiver. With RCDP, we provide flexibility by allowing both sides to send and receive data in the same communication process.

In order to follow the standard protocol layering strategy, we split the implementation of RCDP into different sublayers according to the different tasks required, namely FEC coding and rate control. Figure 1 shows the complete structure of the RCDP protocol, highlighting the most important elements and their combination. Below, we discuss the most relevant design issues, and how the different elements of the architecture have been implemented in software.

A. Raptor encoding and decoding process

In RCDP, information is encoded to protect it against packet loss. The selected coding scheme, Raptor codes, uses a fixed block coding strategy where information to be sent must be divided into fixed size blocks, called *source blocks*, which are then encoded separately. From each block of data, smaller pieces are generated; these pieces, called *symbols*, are encapsulated in data packets and delivered to the destination. We use a systematic Raptor coding scheme where the first set of symbols, called *source symbols*, are an exact replica of the content in the source block itself, and so they can be directly obtained before the actual encoding process starts. An unlimited number of *recovery symbols* are then generated through encoding of the source symbols to allow filling-in the information gaps caused by transmission losses.

The coding process is divided in two complementary sub processes: the encoding process in the data sender and the decoding process in the data receiver. Additional information must be included in a header by the encoding process in order to make possible the decoding process. The packet header defined at this

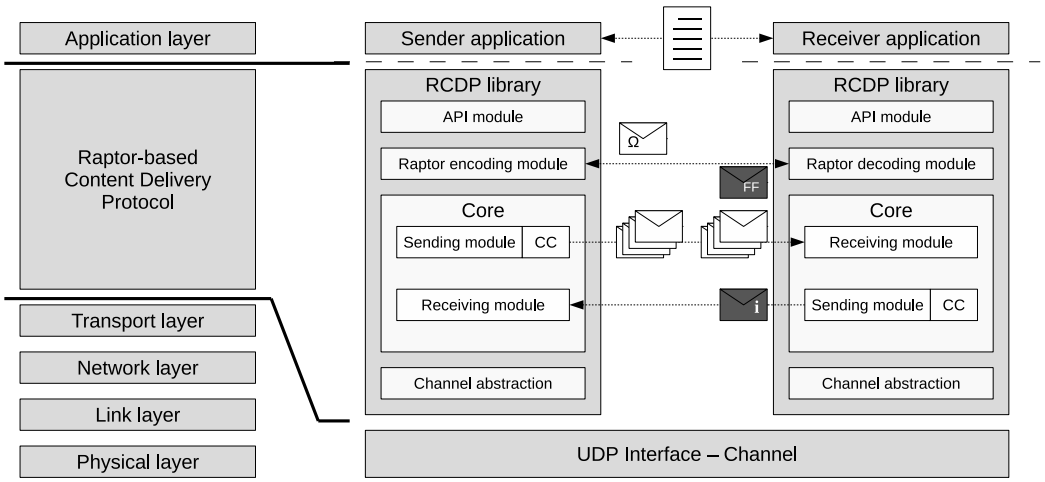


Fig. 1: RCDP implementation diagram.

Algorithm 1 Raptor data processing at the sender.

1. **While** (there is information to send) **do**
   (a) Read next data block from upper sublayer
   (b) Split data block into source symbols
   (c) **For** (all symbols created) **do**
   i. Packetize symbol
   ii. Send packet
   (d) Perform Raptor encoding of data block
   (e) **While not** (received *successfully recovered block* message) **do**
   i. Generate recovery symbol
   ii. Packetize recovery symbol
   iii. Send packet

Algorithm 2 Raptor data processing at the receiver.

1. **While** (information is coming) **do**
   (a) Receive symbol
   (b) **If** (belongs to current block) **do**
   i. Store it in memory
   ii. **If** (received enough symbols to recover the data block) **do**
   A. Recover the data block
   B. Handle it to the upper sublayer
   C. Generate *successfully recovered block* message

level consists of four fields: (i) the number of source symbols in the original message, (ii) the first symbol identifier, (iii) the block identifier, and (iv) the block size.

The sender, as described in algorithm 1, starts by retrieving the information to be sent from the application interface sublayer. It then splits this information into one or more source blocks. Each source block is subdivided into pieces called *source symbols*. The symbols size is typically the maximum size that fits in one packet (header included). We add a header to each symbol to create an RCDP packet, which will be handled by the lower sublayer. Note that, since we rely on systematic Raptor codes, source symbols can be sent immediately, without waiting for the recovery symbols generation process to complete.

These source symbols are used as input for a two stage coding process. In the first stage, pre-coded symbols are generated. Then, through an arithmetic combination of these pre-coded symbols, an infinite number of recovery symbols can be generated.

The Raptor encoding output includes both source and recovery symbols; the latter are also packed and handled to the end-to-end management sublayer for delivery. The latter task goes on uninterruptedly until a successfully recovered block notification is received. The sender repeats this process with the following block until the information flow from the top sublayer ends, or the connection is lost.

With regard to the receiver, it takes the sequence of steps described in algorithm 2. Thus, it is continually awaiting to receive the symbols of a block (both source and recovery symbols), which are stored in memory. When it has enough symbols to recover the source block, it proceeds with the recovery process and sends a *successfully recovered block* notification back to the sender; such notification also serves for flow control purposes. The recovered block is then handed over to the top sublayer, and this entity goes back to the symbol reception state. Note that, in case the channel is lossless, successful decoding takes place immediately after all the source symbols are received, meaning that recovery symbols are not necessary. In those situations the Raptor-related delays are reduced to a minimum, as desirable.

To recover information at the destination, any combination of the original symbols and the recovery symbols allows retrieving of the original information. In fact, for the most recent version of the Raptor libraries [12], the probability of successfully decoding a total of *r* symbols received is shown by equation 1,

$$P_{dec} > 1 - 10^{-2(r-k+1)},\ r \geq k \tag{1}$$

which means that, to recover a source block with symbol size *k*, the probability of a successful decoding is greater than 99% if k encoded symbols are received, greater than 99.99% if k + 1

symbols are received, and greater than 99.9999% if k + 2 symbols are received.

B. The rate control scheme

In wireless networks, channel bandwidth is continuously changing due to variable link quality, variable congestion states, or even variable paths. Therefore, we have to create a rate control system that can easily adapt to highly variable network states, taking advantage of additional bandwidth available or reducing the bandwidth consumption in the presence of other data flows.

The rate control algorithm we proposed is based on channel bandwidth estimations made by the receiver. These estimations are calculated based on packet arrival patterns, and are returned to the sender as soon as they are obtained in order to allow the sender to quickly respond to the bandwidth changes detected.

The proposed strategy consists of grouping data packets in packet trains, and sending them at a rate higher than the one estimated by the receiver as being supported by the end-to-end path. This way, when there is more bandwidth on the channel than the one previously estimated, packets arrive at the receiver faster than expected. In such case, the receiver sends back a new bandwidth estimation reporting that transmission conditions have improved; this allows the sender to increase the sending rate to take advantage of that situation. Otherwise, if the channel conditions have become worse, packets will arrive at a rate lower than expected because of the higher delays experienced. Likewise, the receiver sends back a report so that the sender proceeds to decrease the sending rate accordingly.

To implement this idea, we have devised the following algorithm: while the sender is injecting packets, the receiver is continuously doing bandwidth estimations, one for each packet train, and sending back bandwidth reports (C_i). These bandwidth reports are used by the sender as a reference for its rate adjustments. It applies them a correction parameter (β), as shown in equation 2, to obtain a target data rate R_i . Parameter β varies between 0 and 1, and its purpose is to slightly reduce the target data rate to avoid saturating the channel, thus offering available channel room for other best-effort traffic. The target data rate will be the one we expect to measure at the receiver side. A correction factor (α) allows the determination of train rate (Ω_i) from the target data rate (see equation 3), where α is a value between 0 and 1. The train rate will be the actual rate used to send the packets of a train.

$$R_i = \beta \times C_i \tag{2}$$

$$\Omega_i = \frac{1}{\alpha} \times R_i \tag{3}$$

When the packet train is sent, it is followed by a pause period (inter-train gap) so that the data rate over one period (T_k) matches with the target data rate (R_i), which is the data rate we expect to find in the channel on the long term. Thus, T_k is calculated based on the target data rate, the number of packets in a train (N), and the packet size expressed in bytes (P_{size}) as shown in equation 4.

$$T_k = \frac{8 \times N \times P_{size}}{R_i} \tag{4}$$

Note that the Raptor encoder generates symbols of the same size, which must be initially defined. In our solution, P_{size} is optimized according to the layer-2 MTU, similarly to the approach followed by most TCP implementations.

C. Implementation details

To accelerate the development of the proposed RCDP protocol, we relied on the UDT library [8], which is a communications library written in C/C++ that is available for Linux, Solaris and Windows platforms. This library offers all the features required to implement RCDP, including socket creation and configuration for communication with applications, connection startup and closing, information delivery and reception, etc. Thus, we took advantage of the support code of the UDT library as a starting point to develop RCDP.

Concerning the Raptor modules, they were developed using the libraries provided by Digital Fountain Inc.[1], released under an academic research agreement. In particular, we relied on version 11 of the Raptor libraries for Linux to perform coding/decoding tasks.

We have implemented our RCDP solution in a four-layer approach, following the architecture shown in figure 1. Our implementation combines a multilayered approach with a multithreaded approach, where different modules are combined and different threads cooperate to achieve an efficient and robust solution.

At the top, we have the application interface sublayer, which offers the typical sockets interface, thus allowing the developer to easily update any application. It also simplifies the development of new applications due to the use of a standard interface. It encompasses the API module, which acts as an interface between top level applications and the services offered by the library. At the sender side, it receives and stores the data to be sent, making these data available to lower layers. At the receiver side, it supplies incoming information to the application.

The second layer is the Raptor sublayer. It encompasses both data encoding and decoding modules, and offers encoding/decoding services to upper layers. These modules rely on Raptor codes to generate a virtually infinite flow of symbols which can be used to fully recover sent data blocks, if needed; this will be the actual data sent to the destination. Raptor encoding introduces loss resilience by shielding the transmission against packet corruption or loss. At the sender side, the encoding module encodes the information received by the application interface sublayer, handing packets over to lower sublayer buffers. At the receiver side, the decoding module is responsible for decoding the information received by the lower sublayer, and for handling it to the application sublayer.

The third layer is the end-to-end bandwidth management sublayer. It encompasses both sending and receiving modules, which are responsible for rate control purposes, determining the end-to-end available bandwidth and tuning the transmission rate accordingly. Such mechanisms allow obtaining feedback

[1] Licensed by Qualcomm Inc.

about the network state, a strategy which strongly differs from the TCP's approach, which is based on packet loss detection.

Finally, the bottom layer is the channel abstraction sublayer, which includes a channel abstraction module that simply sends and receives packets to and from an UDP pipe.

Note that all these modules are executed at the user space, being the interaction with the kernel limited to UDP exchanges.

IV. Protocol improvements

RCDP uses Raptor codes (which incur a linear cost in the coding algorithms) that will require efficient implementation. Moreover, since we adopt a user-level development approach, there are additional delays associated with the switching between kernel and user modes that do not appear in kernel level approaches, and whose effects should be mitigated.

A. Baseline optimizations

To optimize the performance of the solution presented above we have to tackle several issues. Below we describe three independent improvements that are able to boost performance.

A.1 Encoding module optimizations

An in-depth analysis of the encoding tasks reveals the following sequence of actions: first, the pre-coding task is performed. Afterward, source codes are delivered to the next module. Then, the coding task takes place and recovery symbols are generated, being fed to the sending module's buffer. At the receiver side, symbols are received and are also stored in a buffer. When the necessary number of symbols is received, the decoding process begins. When a block is successfully recovered, a *successfully recovered block* notification is embedded in each packet train report that is returned to the sender, telling the latter to switch to the next block upon receiving it, in order to establish the flow control.

This encoding process can be further optimized. Since we are using systematic Raptor codes [19], the first output symbols from the encoder are the source symbols themselves, and so no pre-processing for this first set of symbols is required, meaning that they can be sent without actually requiring any encoding to take place. However, the Raptor encoding process is mandatory to create the recovery symbols. We can see this series of tasks in a sequential way, as shown in figure 2a. Following this data flow, a delay between the first (source) and the second (recovery) set of symbols is introduced. To optimize this sequence of tasks, we reconfigure the encoding process so as to eliminate the delay between the two sets of symbols, thereby avoiding additional periods when no packets are sent. In figure 2b, we schematize the target parallelized solution, where partitions indicate that both sending and coding processes are performed in parallel.

There are several ways to implement this optimization. The most intuitive one is to execute them using different threads. However, this approach increases the software overhead, adding function calls that may introduce thread creation and wake up delays. Therefore, we would go with other optimizations that do not imply adding extra software overhead.

Another one could be splitting the coding process into smaller slices, interleaving them with the delivery of source symbols. This technique is possible since we use a Raptor Coding library that allows defining the amount of coding work to perform step by step. By applying this enhancement, we avoid the need to perform encoding only after the sending of source symbols is complete, introducing a pseudo-parallel processing without the need of using threads. However, this approach could introduce additional problems related to the regularity with which the system is able to deliver the packets of a train due to the high CPU usage and low granularity level at this point. As explained in section III, packet trains play an essential role in our solution to assess bandwidth availability in an end-to-end basis, and so their regularity is critical.

The last approach that we have explored is to optimize the buffer's size. If we consider the encoding process as an irregular injection of source and recovery symbols to be sent, instead of two periods of symbol generation followed by an intermediate pause, we can use a buffer to regulate symbol generation to the lower layers. In this case, the only parameter that must be correctly tuned is the buffer size. We must ensure that the transmission time of buffered packets will be greater than the coding time to avoid starvation at the queue level. The buffer size could be estimated empirically or analytically, using equation 5.

$$B_s \geq \frac{T_c \cdot BW}{P_s} \tag{5}$$

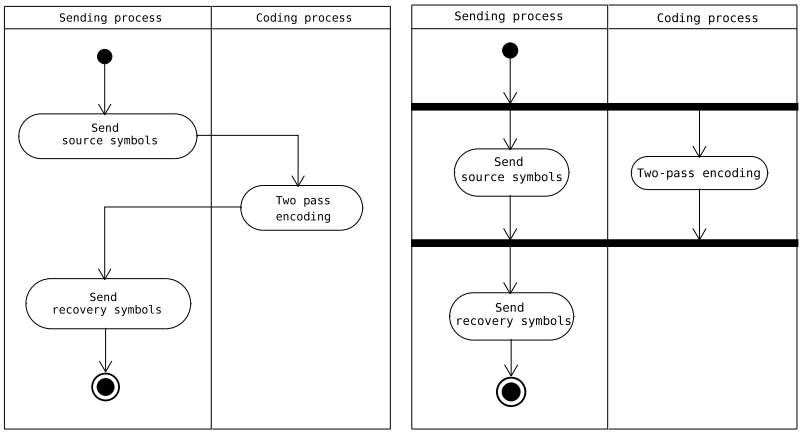
where B_s is the minimum size that the queue buffer should have, in number of packets, T_c is the block coding time, BW is the maximum bandwidth that the channel can achieve (in bits per second), and P_s is the packet size (in bits).

If the buffer size is chosen according to equation 5, the impact of the coding process could be completely mitigated, and the sender may operate without delays, as if the stream of symbols was continuous.

A.2 Packet generation time accuracy

A second issue that must be considered is related to the timing accuracy for the packet generation process. The most precise way to send packets at regular intervals, as required to create a packet train, is to implement a *busy waiting* scheme until the time when a packet must be sent. However, this technique would involve an inadmissible CPU overhead. An alternative is introducing a lower CPU overhead to put the sender thread to sleep between two consecutive packet generation events. That is, if packets of a same train are to be sent every *t* milliseconds, the thread sends a packet, sleeps for t milliseconds, and then wakes up to send another packet. The main drawback of this approach is that there is an additional delay in this sleeping and awakening procedure caused by non instantaneous awakening. Therefore, a delay between the programmed awakening time and the real awakening time takes place, negatively affecting the precision of this procedure.

The proposed solution to this problem relies on a two phase approach, where, in the first phase, a timed waiting period corresponding to a fraction of the sleeping time takes place. In a second phase, the last part of the idle period is a busy waiting. By varying the ratio between both periods we are able to achieve different trade-offs between packet injection time accuracy, and CPU load.



(a) Initial sequential process. (b) Parallel-like process.

Fig. 2: Proposed coding process enhancements.

A.3 Early decoding feedback

A third element prone to optimization is the instant when the source is warned about the correct decoding of the current block. By default, this occurs only when the block decoding procedure is successfully completed. However, since the Raptor libraries provide feedback about the viability of the decoding process even before this process starts, the receiver can warn the sender about it much earlier, thus avoiding wasting time and network resources by preventing the generation of additional recovery symbols when they are no longer required.

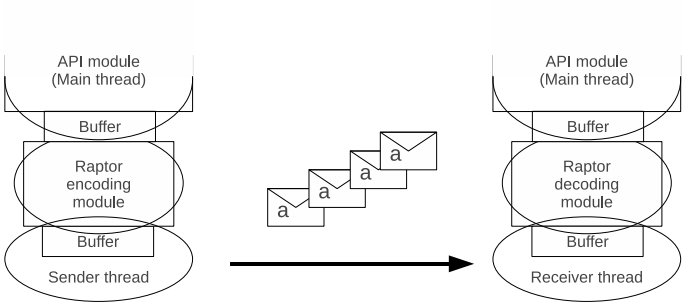
B. Multithreading support

Since hardware architecture trends clearly follow the multiprocessor path, we therefore take advantage of this feature. In our original implementation, blocks are sequentially loaded for encoding. Some library initialization procedures must be performed for every block, thus introducing a startup overhead to the encoding process. In this section we propose a performance improvement strategy that exploits parallel processing to avoid this problem.

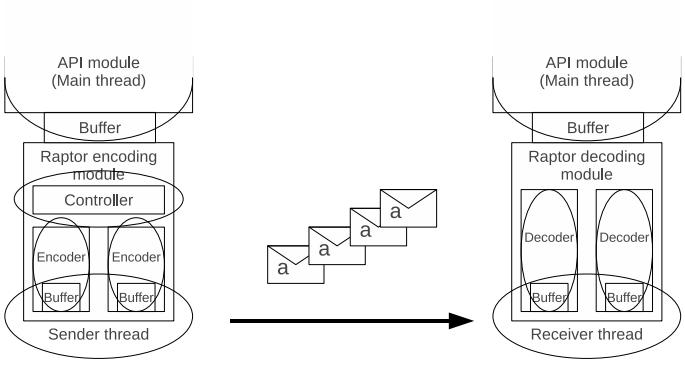
Figure 3a shows the original design for the Raptor coding module in RCDP, where a sequential processing design is adopted. To take advantage of multithreading capabilities, an alternative design is proposed. Figure 3b shows the alternative design when adopting multithreaded coding for both sender and receiver. In both schemes, threads are represented as circles, and data structures as rectangles.

Concerning parallel encoding, it uses two independent threads to encode data blocks in parallel. We will use this new feature to overlap two different encoding processes, thereby avoiding idle periods in the network. To achieve this goal, when a block is being sent, the following block starts being encoded. Due to this preloading of the next block, the sending process will be improved by parallelizing all the management structures.

Parallel decoding adds to the previous one the ability to decode up to two blocks in parallel. This can be used, as parallel coding, to get a decoder ready to work without delay while the previous block is being decoded.



(a) Sequential coding design.



(b) Multithreaded coding design.

Fig. 3: Coding design options in RCDP.

When several threads are working cooperatively, as in the aforementioned cases, processing overhead associated with thread management can become a problem. As occurs in all processes whose complexity is incremented, additional software overhead must be considered. Therefore, these additional processing delays, which could downgrade performance compared to simpler, sequential implementations. It is important to determine the optimal trade-off between parallelization and overhead to achieve the maximum performance.

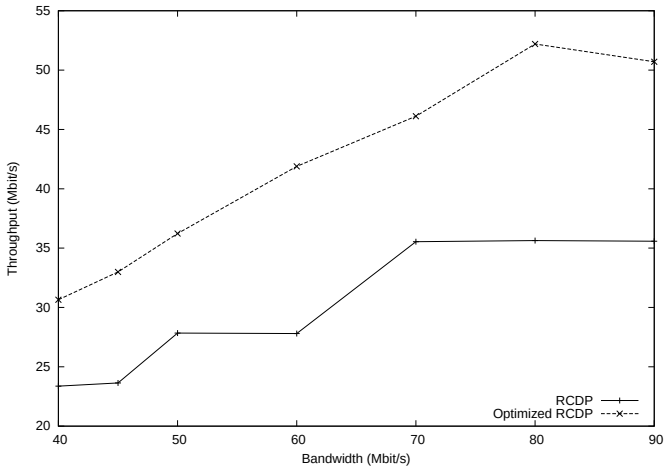


Fig. 4: Throughput vs. available bandwidth in a network with a 10 ms end-to-end delay (null error rate).

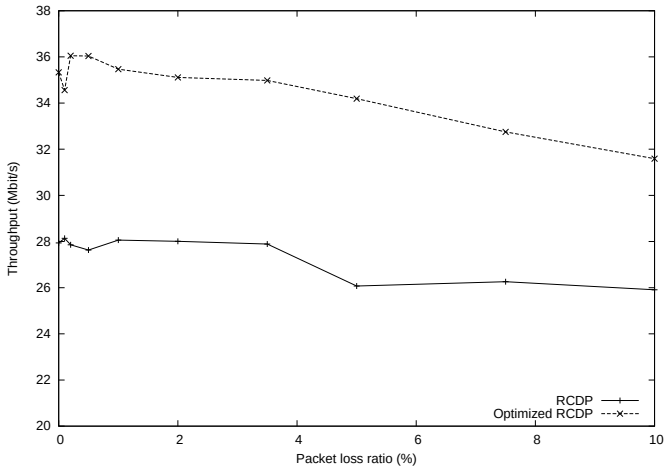


Fig. 5: Throughput vs. packet loss rate in a 50 Mbps channel with a 10 ms delay.

V. Performance evaluation

In this section, we will quantify the difference between the original implementation, described in section III, and the optimized one described in section IV.

Our experiments were made with real working software on the GNU/Linux platform. To ensure that the test sequences were reproducible, we created a controlled environment for testing using channel emulation. We created a network black box over an Ethernet connection that emulates a configurable point-to-point connection between two wireless hosts. It can be configured by setting parameters such as the packet loss rate, the available bandwidth or the end-to-end delay in order to emulate different wireless channel conditions. The proposed network black box was initially validated to make sure that test results were reliable, and that all the comparisons we made were fair.

A. Performance evaluation under different conditions

In this section, we study the performance of both RCDP versions when varying the available channel bandwidth, the end-to-end packet loss ratio, and the size of the contents to be delivered.

Figure 4 shows the throughput achieved when varying the available bandwidth. We observe how, in general, the different RCDP versions tested behave as expected, experiencing an almost linear throughput increase as the available bandwidth increases. We find that the optimized version is able to achieve a higher degree of productivity compared to the original one. In particular, when compared to the original RCDP implementation, combining baseline optimizations with parallel coding allows increasing throughput by about 45% in the best case, which is a very substantial improvement. Besides throughput enhancements, the block preload technique described in section IV-B, along with the buffer size optimization described in section IV-A, enables the generation of a continuous symbol flow which contributes to a more regular transmission rate compared to the original RCDP.

In figure 5, we can see the throughput performance when varying the packet loss rate in the network. The desired behavior would show a linear throughput decrease for increasing packet loss ratios. We find that, in general, both RCDP versions approximately follow this trend. Results show that error immunity remains similar to the original RCDP implementation in the optimized one, although actual throughput values basically depend on the different enhancements proposed, as explained above.

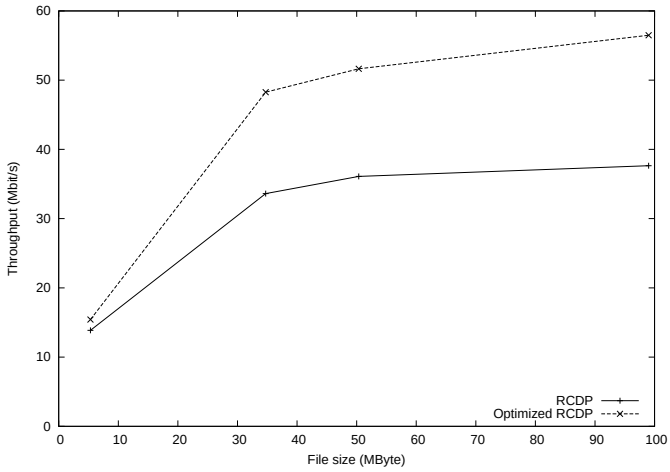


Fig. 6: Throughput vs. file size in a network environment with 50 Mbps bandwidth and a 10 ms delay (null error rate).

In figure 6, we can see the throughput performance when varying the size of the delivered content. Note that, when the file size is small, the average throughput is low because the initial startup time overhead is similar to the transmission time. A similar effect occurs with TCP as well. For greater files sizes, the impact of the startup times on throughput become negligible. Once again, actual throughput values depend on the different enhancements proposed.

B. Resource consumption analysis

We now focus on the computational resources required by RCDP and our proposed optimizations. The results were ob-

Table 1: Resource consumption

	CPU utilization		RAM Consumed	
Version	Client	Server	Client	Server
RCDP	13 %	57 %	10 MB	77 MB
Optimized RCDP	23 %	60 %	58 MB	120 MB

tained using the Linux "ps" tool as a background process, which periodically measure the CPU utilization (CPU time used divided by the time the process has been running) and RAM utilization at the both client and server. Note that, despite similar software is running on both client and server, and despite communication is bidirectional, data is being transferred mostly from server to client. The server will mostly be performing data encoding tasks, while the client will be performing decoding tasks instead. Thus, for our baseline RCDP implementation, the server CPU load is about 40% greater than the load at the client.

Table 1 shows the additional CPU overhead of both original and optimized RCDP. We can see how CPU utilization is incremented in both client and server. However, it is a little increment if compared to the throughput increment shown in previous section. Focusing on RAM usage, RAM increment is bigger due to structure duplication needed to perform optimized tasks, but it is maintained at very low values compared to current RAM availability in most devices.

Overall, results show that, despite both client and server share the same protocol architecture, the emphasis on either encoding or decoding tasks results in different behaviors. In both cases, the requirements and complexity of Raptor encoding impose more overhead on the server compared to the client: about 40% in terms of CPU, and around 60 MB in terms of RAM.

VI. Conclusions and future work

TCP has the gift and curse of being the most widely used transport protocol in the world. It was created when wired networks were dominant and it was designed primarily for that environment. However, as we move towards the wireless era, we find that TCP is not highly efficient under certain wireless channel conditions. In fact, to maximize performance in an environment characterized by high delays and high packet loss rates such as those where ITS are deployed, other alternatives must be sought. In this article, we present RCDP - a content delivery solution designed to exhibit a higher throughput than TCP under those conditions.

RCDP is a full-duplex content delivery protocol which introduces a bidirectional communication scheme following a completely novel approach. It encompasses the sending and receiving processes at both client and server, using an AL-FEC strategy to provide a reliable transmission without any retransmission requirements.

Experimental results in a realistic testbed show that even the most basic implementation of RCDP achieves a high degree of throughput. RCDP also shows a near ideal throughput curve when varying the packet loss rate. When applying different enhancements proposed in the paper, we find that throughput levels are further improved. Overall, we consider RCDP as an useful alternative to TCP in wireless environments such as vehicular and ad hoc networks while offering performance levels similar to TCP in low loss environments. Therefore, it represents a good alternative to apply this protocol in intelligent transport systems.

Acknowledgments

This work was partially supported by the *Ministerio de Ciencia e Innovación*, Spain, under Grant TIN2011-27543-C03-01 and by the *Ministerio de Educación*, Spain, under the FPU program, AP2010-4397.

REFERENCES

[1] E. Ayanoglu, S. Paul, T.F. LaPorta, K.K. Sabnani, and R.D. Gitlin. AIRMAIL: A link-layer protocol for wireless networks. *Wireless Networks*, 1(1):47–60, 1995.

[2] H. Balakrishnan, V.N. Padmanabhan, S. Seshan, and R.H. Katz. A comparison of mechanisms for improving TCP performance over wireless links. *Networking, IEEE/ACM Transactions on*, 5(6):756–769, 1997.

[3] H. Balakrishnan, S. Seshan, E. Amir, and R.H. Katz. Improving TCP/IP performance over wireless networks. In *Proceedings of the 1st annual international conference on Mobile computing and networking*, pages 2–11. ACM, 1995.

[4] K. Brown and S. Singh. M-TCP: TCP for mobile cellular networks. *ACM SIGCOMM Computer Communication Review*, 27(5):19–43, 1997.

[5] R. Caceres and L. Iftode. Improving the performance of reliable transport protocols in mobile computing environments. *Selected Areas in Communications, IEEE Journal on*, 13(5):850–857, 2002.

[6] K. Chandran, S. Raghunathan, S. Venkatesan, and R. Prakash. A feedback-based scheme for improving tcp performance in ad hoc wireless networks. *Personal Communications, IEEE*, 8(1):34–39, 2001.

[7] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Rfc2616: Hypertext transfer protocol–http/1.1. RFC Editor United States, 1999.

[8] Y. Gu and R.L. Grossman. Supporting configurable congestion control in data transport services. In *Supercomputing, 2005. Proceedings of the ACM/IEEE SC 2005 Conference*, pages 31–31. IEEE, 2005.

[9] S. Keshav and SP Morgan. SMART retransmission: Performance with overload and random losses. In *INFOCOM'97. Sixteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, volume 3, pages 1131–1138. IEEE, 1997.

[10] J. Liu and S. Singh. ATCP: TCP for mobile ad hoc networks. *Selected Areas in Communications, IEEE Journal on*, 19(7):1300–1315, 2002.

[11] M. Luby, M. Watson, T. Gasiba, T. Stockhammer, and Wen Xu. Raptor codes for reliable download delivery in wireless broadcast systems. In *Consumer Communications and Networking Conference, 2006. CCNC 2006. 3rd IEEE*, volume 1, pages 192 – 197, January 2006.

[12] M. Luby, A. Shokrollahi, M. Watson, and T. Stockhammer. RaptorQ Forward Error Correction Scheme for Object Delivery. Internet Engineering Task Force, Internet Draft draft-ietf-rmt-bb-fec-raptorq-00, Work in progress, January 2010.

[13] D.J.C. MacKay. Fountain codes. In *Communications, IEE Proceedings-*, volume 152, pages 1062–1068. IET, 2005.

[14] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. RFC2018: TCP Selective Acknowledgement Options. *RFC Editor United States*, 1996.

[15] C. Parsa. TULIP: A link-level protocol for improving TCP over wireless links. In *Wireless Communications and Networking Conference, 1999. WCNC. 1999 IEEE*, pages 1253–1257. IEEE, 2002.

[16] J. Postel. Rfc793: Transmission control protocol (tcp). Internet Engineering Task Force, September, 1981.

[17] J. Postel and J. Reynolds. Rfc 959: File transfer protocol (ftp). InterNet Network Working Group, 1985.

[18] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A Transport Protocol for Real-Time Applications. RFC 3550, 2003.

[19] A. Shokrollahi. Raptor codes. *Information Theory, IEEE Transactions on*, 52(6):2551–2567, 2006.

[20] P. Sinha, T. Nandagopal, N. Venkitaraman, R. Sivakumar, and V. Bharghavan. WTCP: A reliable transport protocol for wireless wide-area networks. *Wireless Networks*, 8(2/3):301–316, 2002.