RPT: Re-architecting Loss Protection for Content-Aware Networks

Dongsu Han, Ashok Anand†, Aditya Akella†, Srinivasan Seshan
Carnegie Mellon University †*University of Wisconsin-Madison*

Abstract

We revisit the design of redundancy-based loss protection schemes in light of recent advances in content-aware networking. Content-aware networks minimizes the overhead of redundancy, if the redundancy is introduced in a way that the network can understand. With this insight, we propose a new loss protection scheme called redundant packet transmission (RPT). Using redundant video streaming as an example, we show that our approach, unlike FEC in traditional networks, provides low latency with high robustness and is insensitive to parameter selection. We tackle practical issues such as minimizing the impact on other traffic and the network. We show that RPT provides a simple and general mechanism for application-specific control and flow prioritization.

1 Introduction

A variety of current and future Internet applications require time critical or low latency communication. Example applications include delay-sensitive live/interactive video streams, online games, and video-based calls (e.g., Apple's FaceTime), all of which send real-time data. Studies of real-time systems [36, 48] suggest that the maximum tolerable one-way delay is around 150ms for real-time interaction. Within a data center, many soft real-time applications that interact with users require low latency communication [13]. Certain classes of inter-datacenter transfers, such as mirroring financial data, also require real-time communication [17].

The central challenge in supporting such delay-sensitive real-time applications is protecting them from network loss. One set of conventional approaches—acknowledgment-based retransmission protocols—are not appropriate for real-time communication as retransmissions triggered by timeouts can take several RTTs and violate applications' timing constraints [44, 51]. Another set of approaches—redundancy-based schemes such as Forward Error Correction (FEC)—suffer from a fundamental tension between robustness and the bandwidth overhead [20, 25], making them either difficult to tune or inefficient in practice.

These techniques have been tuned to provide the best performance tradeoffs possible in traditional networks. In contrast, the focus of our paper is to show that better protection against congestion losses may be possible in *content-aware networks*. We use the term content-aware networks to refer to the variety of architectural proposals [14, 32, 37, 42] and devices [2, 4, 10, 35] that cache data and remove duplicates to alleviate congestion (i.e., they perform *content-aware* processing of packets). Content-aware processing is seeing ever-growing adoption in a variety of settings, including mobile and cellular networks [11], data centers [35], cloud computing [9], and enterprise networks [4]. The most popular of such content-aware network devices are the WAN optimizers [5, 8, 10] that are typically placed at branch and main offices or between data-centers to reduce the traffic between them. Market reports indicate that the market for such devices is growing rapidly [4].

Our core assumption is that content-aware network devices will be widely deployed across a variety of links in future networks. Given this setting, we ask: *(i)* How do we re-architect loss protection for delay sensitive applications operating in this new context? *(ii)* Does content-awareness help simplify or further complicate the issues that existing loss protection schemes face? And, why?

We show that taking content-awareness into account challenges the conventional wisdom on the trade-offs of redundancy in protecting against losses in time-critical and delay-sensitive applications. In particular, we show that it is now possible to use redundancy in a simple yet clever fashion to ensure robustness against congestion losses while imposing little or no impact on the network or on other existing applications. Equally importantly, we show that it is now far easier to integrate loss protection with other design constraints such as adhering to tight delay bounds.

We believe that the duplicate suppression actions in content-aware frameworks provide a tremendous opportunity to use redundancy-based protection schemes. However, redundancy must be introduced in the right way to ensure: *(a)* the network can eliminate it optimally to provide the desired efficiency and *(b)* the impact on other applications can be controlled.

We describe Redundant Transmission (RT) – a loss protection scheme that intelligently sends multiple copies of the same data. The basic idea of RT is to expose the redundancy directly to the underlying content-aware network.

The simplest form of RT is to send multiple copies of the same packet. When packets are not lost, the duplicate transmissions in RT are compressed by the underlying network and add little overhead. In contrast, when the network is congested, the loss of a packet prevents the compression of a subsequent transmission. This ensures that the receiver still gets at least one decompressed copy of the original data. In some situations, the fact that packet losses do not directly translate to less bandwidth use may raise the concern that RT streams obtain an unfair share of the network. However, existing congestion control schemes, with some critical adjustments to accommodate RT behavior, can address this concern.

In essence, RT signals the network the relative importance of packet by transmitting multiple copies. RT requires almost no tuning; this stands in stark contrast with the difficulty of fine-tuning FEC-based approaches for traditional networks. Finally, RT decouples redundancy from delay and easily accommodates application timing constraints; in comparison, FEC schemes today closely tie delay guarantees with redundancy encoding since the receiver cannot reconstruct lost packets until the batch is complete. In effect, RT on content-aware networks can effectively support a variety of time-critical applications far better than existing approaches for traditional networks.

To illustrate the benefits of RT concretely, we use as an example RPT, a simple variant of RT for real-time video in a redundancy elimination network. Our evaluation of RPT, using a combination of real-world experiments, network measurements and simulations, shows that RPT decreases the data loss rate by orders of magnitude more than FEC schemes applicable to live communications. As a result, it achieves better video quality than FEC for a given bandwidth budget, or uses up to 20% less bandwidth than FEC schemes to deliver the same video quality.

We make the following contributions in this paper:

1. We highlight the need to reconsider the design of loss protection for content-aware networks. We show that network content-awareness enables vastly simpler and more effective approaches to loss protection (§3).

2. We describe a redundancy scheme, RT, that can provide a high degree of robustness at low overhead, and require minimal tuning (§3 and §4).

3. Through extensive experiments and simulations, we find that RT can improve the robustness of real time media applications with strict timing constraints (§6).

In the remaining sections, we review related work (§2), discuss realistic deployment examples as well as general implementation of RT on other content-aware networks (§5), and finally conclude in §7.

2 Current Loss Recovery Schemes

Packet losses are often inevitable on the Internet, especially across heavily-loaded links, such as cross-country or trans-continental links. Many prior works use *timeout-based retransmission* to recover lost data [18, 27, 44, 49, 51] on traditional networks. However, retransmission causes large delays [51] which are often difficult to hide. Also, the performance depends on correct timeout estimation [44] which is often non-trivial [18, 44]. Because of these intrinsic limitations, more sophisticated enhancements such as selective retransmission [27, 49], play-out buffering [49], and modification to codecs [51] are often required to augment retransmission based loss recovery.

Another option is *redundancy-based recovery*, with FEC being an example framework that is widely used today. While coding provides resilience, the use of FEC is constraining in many ways in practice: *(1)* In FEC, the receiver cannot recover lost packets until the batch is complete. This limits the size of the batch for delay-sensitive applications. For example, at most 5 packets are typically batched in video chat applications such as Skype [56]. *(2)* Small batch size makes FEC more susceptible to bursty loss. For example, adding a single coded FEC packet for every five original data packets is not enough to recover from two consecutive lost packets. Therefore, in practice, the amount of redundancy used is high (e.g., 20% to 50% [20, 25, 56]), which is much higher than the underlying packet loss rate. *(3)* Furthermore, FEC needs to adapt to changing network conditions [19, 57], which makes parameter tuning even more difficult. Many studies [29, 57] have shown that fine tuning FEC parameters within various environments is non-trivial.

More sophisticated redundancy-based recovery schemes such as fountain codes [21], rateless coding with feedback [30], and hybrid ARQ [43] introduce redundancy incrementally. However, fountain codes, rateless coding have been mostly used for bulk data transfer or non-real-time streaming. Hybrid ARQ has been mostly used in local wireless networks where the round-trip time is much smaller compared to real-time delay constraints. When the round-trip time is comparable to real-time delay constraints, incremental redundancy schemes degenerate to FEC. Many other sophisticated schemes such as multi-description coding [50] also use FEC to scale the video quality proportional to the bandwidth. While these schemes relax some of the above limitations of FEC, the fundamental limitation of small batch size is inherent to delay-sensitive applications.

3 Redundant Packet Transmission

We now describe the design of Redundant Packet Transmission (RPT), a simple variant of RT that sends fully redundant packets for delivering interactive video streams. We envision a scenario in which real-time video traffic and other traffic coexist, with no more than 50% of the traffic on a particular link being interactive, real-time traffic. We picked this scenario because it is representative

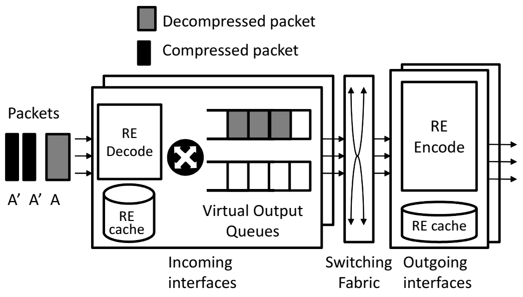


Figure 1: Redundant Packet Transmission in a redundancy elimination router.

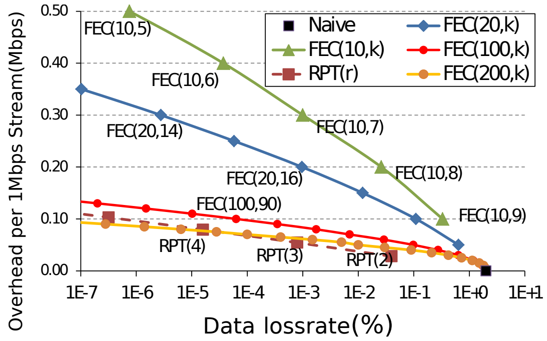


Figure 2: RPT and FEC under 2% random loss.

of forecasts of future network traffic patterns [3].

To simplify exposition, throughout this section, we assume that the RPT flows travel through a network with hop-by-hop Redundancy Elimination (RE) enabled [14]. Later, in §5, we explore RT in content-aware networks of various other forms. As stated earlier, we assume that packet losses happen only due to congestion.

We start by providing background on RE. We then describe our basic idea, followed by a description of key benefits and some comments on our approach.

3.1 Redundancy Elimination Background

In Anand et al's [14] design, RE is deployed across individual ISP links. An upstream router remembers packets sent over the link in a cache (each cache holds a few tens of seconds' worth of data) and compares new packets against cached packets. It encodes new packets on the fly by replacing redundant content (if found) with pointers to the cache. The immediate downstream router maintains an identical packet cache, and decodes the encoded packet. RE is applied in a hop-by-hop fashion.

RE encoding and decoding are deployed on the line cards of the routers as shown in Figure 1. Decoding happens on the input interface before the virtual output queue, and encoding happens on the output interface. The router's buffers (virtual output queues) contain fully decoded packets.

3.2 Basic Idea

As explained earlier, the basic idea of redundant packet transmission (RPT) is to send multiple copies of the same packet. If at least one copy of the packet avoids network loss, the data is received by the receiver. In current network designs, transmitting duplicate packets would incur large overhead. For example, if two duplicates of every original packet are sent, the overhead is 200% and a 1Mbps stream of data would only contain 0.33Mbps of original data. However, in networks with RE, duplicate copies of packets are encoded into small packets, and this overhead would be significantly reduced.

Figure 1 illustrates how RPT works with redundancy elimination. From the input link, three duplicate packets are received. The first packet is the original packet A, and the other two packets A', are encoded packets which have been "compressed" to small packets by the previous hop RE encoder. The compressed packet contains a reference (14 bytes in our implementation) used by the RE decoder of the next hop. At the incoming interface, the packets are fully decoded, generating 3 copies of packet A. They are then queued at the appropriate output queue. The figure illustrates a router that uses virtual output queuing. When congestion occurs, packets are dropped at the virtual output queue. Only packets that survive the loss will go through to the RE encoder on the output interface. When multiple packets survive the network loss, the first packet will be sent as decompressed, but the subsequent redundant packets will again be encoded to small packets by the RE encoder.

In this manner, multiple copies of packets provide robustness to loss and RE in the network reduces bandwidth overhead of additional copies.

3.3 Key Features

Next, we discuss three practically important properties of RPT: *high degree of robustness with low bandwidth overhead*, *ease of use and flexibility for application developers*, and *flow prioritization in the network*.

3.3.1 Low Overhead and High Robustness

As discussed in [14], the packet caches in the RE encoder and decoder are typically designed to hold all packets sent within the last tens of seconds. This is much longer than the timescale in which redundant packets are sent ($\sim$60ms). Thus, all redundant packets sent by the application will be encoded with respect to the original packet. The extra bandwidth cost of each redundant packet is only the size of the encoded packet (43 bytes in our implementation[1].) The overhead of an extra redundant packet, therefore, is less than 3% for 1,500 byte packets, which is 7 to 17 times smaller than the typical FEC overhead for a Skype video call [20, 25].

To compare RPT with FEC, we model RPT and FEC under a 2% uniform random packet loss and analytically

[1]Our implementation does not encode IP and transport layer headers.

derive the data loss of a 1Mbps RPT and FEC streams in an RE network. Figure 2 shows the resulting overhead and data loss rate. All flows operate on a fixed budget but splits its bandwidth between original data and redundancy. The overhead (*y-axis*) is defined as the amount of redundancy in the stream, and the data loss (*x-axis*) as the percentage of data that cannot be recovered. RPT(r) denotes redundant streaming that sends r duplicate packets. FEC flows with various parameters are shown for comparison. FEC(n,k) denotes that k original packets are coded in to n packets. For FEC, we use a systematic coding approach (e.g. Reed-Solomon) that sends k original packets followed by $n-k$ redundant packets. While both schemes introduce redundancy, only the redundancy introduced by RPT gets minimized by the network unlike FEC which does not introduce redundancy in a way that the network understands; thus FEC over RE networks is identical in performance to FEC over traditional networks.

FEC schemes, especially with a small group size (n), incur large overheads, and are much less effective in loss recovery. For example, FEC(10,8), which adds 0.2 Mbps of redundancy, has similar data loss rates as RPT(2), which only adds 0.03Mbps of redundancy. FEC with group size (n=200) performs similar to RPT. However, it takes 2.4 seconds to transmit 200 1,500 byte packets at 1Mbps. This violates timing constraints of real-time communications because a packet loss may only be recovered 2.4 seconds later in the worst case. Thus, in practice, RPT provides high robustness against packet loss at low overhead.

3.3.2 Ease of Use and Control

Application developers can easily tailor RPT to fit their needs. Three unique aspects of RPT help achieve this property:

1) Detailed parameter tuning is not necessary.
2) RPT allows per-packet redundancy control.
3) Delay and redundancy are decoupled.

Ease of parameter selection: With FEC, the sender has to carefully split its bandwidth between original and redundant data in order to maximize the video quality. If the amount of redundancy is larger than the amount of network loss, the stream tolerates loss. However, this comes at the cost of quality because less bandwidth is used for real content. If the amount of redundancy is too low, the effect of loss shows up in the stream and the quality degrades. This trade-off is clear in FEC(10,k)'s performance in Figure 2. Determining the optimal parameters for FEC is difficult and adapting it to changing network conditions is even more so [29].

A unique aspect of RPT is that even though the actual redundancy at the sender is high, the network effectively reduces its cost. Therefore, the sender primarily has to ensure that the amount of redundancy (r) is high enough to tolerate the loss and worry much less about its cost, which makes RPT simple and easy to use. We show in §6.3 that only small amount of redundancy ($r = 3$) is good enough for a wide range of loss rates (1% to 8%), and a sub-optimal overshoot (i.e. unnecessary, extra redundancy) has very little impact on actual video quality.

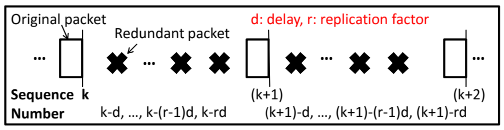
Packet-by-packet redundancy control: RPT introduces redundancy for each packet as opposed to groups of packets, enabling packet-by-packet control of the extent of redundancy. More important packets, e.g., those corresponding to I-frames, could simply be sent more repeatedly than others to increase robustness. In essence, RPT enables fine-grained unequal error protection (UEP) [33]. Thus, RPT is simple to adapt to application-specific needs and data priorities.

Each encoded packet can be viewed as an implicit signal to the network. Importance of the data is encoded in the number of encoded packets, $r-1$. When an original packet gets lost, routers try to resend the original packet when the signal arrives. As such the network tries up to r times until one original copy of the packet goes through.

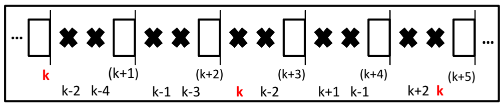
Decoupling of delay and redundancy: Unlike FEC, RPT separates the redundancy decision from delay. FEC schemes closely tie timing with the encoding since the receiver cannot reconstruct lost packets until the batch is complete. In contrast, RPT accommodates timing constraints more easily. For example, sending 3 redundant packets spaced apart by 5 ms is essentially asking every router to retry up to 3 times every 5 ms to deliver the original packet. This mechanism lends itself to application specific control to meet timing constraints. We further discuss the issues in controlling delay in §3.4.

3.3.3 Flow Prioritization

A unique property of RPT-enabled traffic is that it gets preferential treatment over other traffic under lossy conditions. RPT flows do not readily give up bandwidth as quickly as non-RPT flows. This is because for RPT flows packet losses do not directly translate into less bandwidth use due to "deflation" of redundant packets; subsequent redundant packets cause retransmission of the original packet when the original packet is lost. Therefore, *RPT flows are effectively prioritized in congested environments.* As a result, RPT could get more share at the bottleneck link. We believe that this is a desirable property for providing stronger guarantees about the delivery rate of data, and analyze this effect in §6. However, this preferential treatment may not be always desirable. In case where fair bandwidth-sharing is desired, RPT is flexible enough to be used with existing congestion control mechanisms while retraining its core benefits. In §4, we provide an alternative solution that retains other two benefits of RPT except flow prioritization.



(a) Sequence of packets sent by RPT(r)



(b) Sequence of packets sent by RPT(3) with d=2

Figure 3: Sequence of packets sent by RPT

3.4 Scheduling Redundant Packets

We now discuss detailed packet sequencing, i.e. how RPT interleaves redundant packets with original packets. Each original packet is transmitted without any delay, but we use two parameters to control the transmission of redundant packets: redundancy (r) and delay (d).

Figure 3(a) shows the packet sequence of an RPT(r) flow. Original packets are sent without any delay, and $r-1$ redundant packets are sent compressed in between two adjacent original packets. Thus, compared to a non-RPT flow of the same bitrate, r times as many packets are sent by a RPT(r) flow.

The delay parameter (d) specifies the number of original packets between two redundant packets that encode the same data. The first redundant packet of sequence number n is sent after the original packet of sequence number $(n+d)$. If the loss is temporally bursty, having a large interval between two redundant packets will help. However, extra delay incurs extra latency in recovering from a loss. So, delay (d) can be adjusted to meet the timing requirements of applications.

Figure 3(b) shows an example with $r = 3$ and $d = 2$. Three copies of packet k is sent, each spaced apart by two original packet transmissions. In §6.3, we evaluate RPT's sensitivity to parameter selection.

3.5 Comments on RT/RPT

Is this link-layer retransmission? Conceptually, RT is similar to hop-by-hop reliability or link-layer retransmission. However, RT fits better with the end-to-end argument-based design of the Internet by giving endpoints an elegant way to control the retransmission behavior inside the network. In contrast, hop-by-hop reliability schemes make it hard for applications to control the delay or to signify the relative importance of data. Similarly, in a naive hop-by-hop retransmission scheme, packets are treated equally and can be delayed longer than the application-specific limit. RT exploits network's content-awareness and provides a signaling mechanism on top of such networks to achieve robustness against packet loss.

Why not make video codecs resilient? In the specific context of video, prior works have proposed making video codecs more resilient to packet loss. Examples include layered video coding [46], H.264 SVC, various loss concealment techniques [55] and codecs such as ChitChat [56]. However, greater loss resilience does not come for free in these designs; these designs typically have lower compression rate than existing schemes or incorporate redundancy (FEC) in order to reconstruct the video with arbitrary loss patterns. Also they are often more computationally complex than existing approaches, which makes them difficult to support on all devices [55].

Our scheme is agnostic to the choice of video codec and the loss concealment schemes used. Of course, the exact video quality gains may differ based on the loss rates, loss patterns and codec used.

How does RT compare to more sophisticated coding? Many sophisticated video coding schemes, such as UEP [33], priority encoding transmission [12], and multiple description coding [23, 50], typically use FEC (or Reed-Solomon codes) as a building block to achieve graceful degradation of video quality. Similarly, we believe that RT can be used as a building block to enable more sophisticated schemes. For example, one can send more important blocks of bits within a stream multiple times. Furthermore, since RE networks also eliminate sub-packet level redundancy, a partially redundant packet may also be used. We leave details of such techniques as future work. In this work instead, we focus on understanding the core properties of RT by comparing a basic form of RT with the most basic use of FEC.

What about wireless errors? We do not yet know if RT/RPT can be extended to protect against categories of losses other than those due to congestion, e.g., partial packet errors due to interference and fading. We do note that there a variety of schemes that aim to provide robust performance in such situations, some with a focus on video (e.g., the schemes in [38, 39, 53] for wireless links). However, RT/RPT's explicit focus on congestion losses means that our approach is complementary to such schemes. In our technical report [31], we discuss how RT can happily coexist with such schemes.

4 RPT with Congestion Control

As explained earlier, RPT flows are effectively prioritized in congested environments[2]. However, in some environments, fair bandwidth sharing may be more desirable. In such cases, the sending rate should adapt to the network conditions to achieve a "fair-share". To meet this goal, we apply TCP friendly rate control [28] (TFRC) to RPT flows. However, this raises surprisingly subtle problems regarding the transmission rate and loss event rate estimation that are germane to TFRC. We describe these challenges and our modifications to TFRC below.

[2]We further verify this later in §6.4.

Packet transmission: In TFRC for RPT, we calculate the byte transmission rate from the equation just as the original TFRC. Note that RPT(r) must send an *original* packet and $r-1$ duplicates. To match the byte sending rate, we adjust the length of the packet so that equal number of bytes are sent by TFRC RPT as the original TFRC in calculating the throughput. Thus, given a computed send rate, TFRC RPT(r) sends r times as many packets. Note that each packet, original or duplicate, carries an individual sequence number and a timestamp for TFRC's rate calculation purposes.

Loss event rate estimation: In the original TFRC, the sending rate is calculated given the loss event rate p, where loss event rate is defined as the inverse of the average number of packets sent between two loss events. A loss event is a collection of packet drops (or congestion signals) within a single RTT-length period.

Ideally, we would want TFRC RPT(r) to have the same loss event rate as the original TFRC, as that would also make TFRC RPT obtain a TCP friendly fair share of bandwidth. However, the observed loss event rate for RPT depends on the underlying packet loss pattern. For ease of exposition, we look at the two extremes of loss patterns: one that is purely random and the other that is strictly temporally correlated.

Purely random packet drops may occur in a link of a very high degree of multiplexing. On the other hand, in a strictly temporal loss pattern, losses occur during specific intervals. One might see such a loss pattern when cross traffic fills up a router queue at certain intervals. In reality, the two patterns appear inter-mixed depending on source traffic sending patterns and the degree of multiplexing.

Next, we discuss how the two loss patterns impact loss event rate estimation and the transmission rate:

- *Uniform random packet loss:* In this setting, TFRC RPT(r) behaves in a TCP friendly manner without any adjustment to loss estimation. This is because the number of packets sent between two loss events does not change even though the packet sending rates change.
- *Temporal packet loss:* In this setting, packets are lost at specific times. During the time between loss, TFRC RPT(r) sends r times as many packets. Thus, the observed loss event rate for TFRC RPT(r) is only $\frac{1}{r}$ of that of the original TFRC. Therefore, TFRC RPT would send more traffic.

Adjusting the loss event rate: As stated earlier, in practice, the two extreme patterns appear inter-mixed. We therefore want to choose an adjustment factor α so that when the loss event rate is adjusted to α times the measured loss event rate p, TFRC RPT(r) is TCP friendly. As seen in the two extreme cases, α has values 1 for uniform random losses and r for temporal losses, respectively. So, in practice, α should be between 1 and r to achieve exact TCP-friendliness. A larger value of α makes the TFRC-RPT react more aggressively to congestion events, and smaller value less aggressive than a TCP flow. This means, even in the worst case, α can be r times off from the value which achieves exact TCP-friendliness. In this case, a TFRC-RPT flow would have performance similar to $\sqrt{r}$ many TFRC flows because the TCP-friendly rate is inversely proportional to $\sqrt{p}$. Therefore, even an incorrect value of α would still make TFRC RPT friendly to a group of TCP connections and still react to congestion events. In practice, we find in §6 that with TFRC-RPT(3), $\alpha = 1.5$ closely approximates the bandwidth share of a single TCP flow under wide range of loss rates and realistic loss patterns.

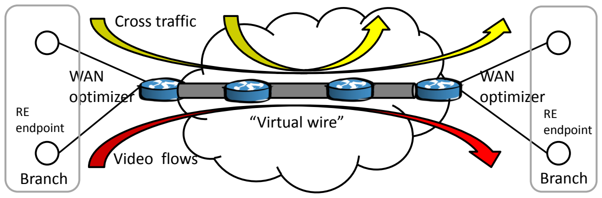


Figure 4: Typical Deployment of WAN optimizers

As such, RT is flexible enough to allow users to adjust the degree of reactivity to congestion events while being highly robust. Regular RPT does not react to congestion events, and can be used to prioritize important flows. TFRC RPT reacts to congestion events and the reaction degree can be controlled by the parameter α.

5 RPT in Various Networks

So far, we have explored RPT on hop-by-hop RE networks as a special case of redundant packet transmission. Here, we look at other deployment scenarios for content-aware devices as well as other content-aware designs.

Corporate networks: WAN optimization is the most popular form of RE deployment in the real world. In a typical deployment, WAN optimizers are placed at branch and main offices or between data-centers to reduce the traffic between them. Example deployments include 58+ customers of Riverbed [10] and Cisco's worldwide deployment to its 200+ offices [8]. While we envision RPT being used in future networks where content-aware devices are widely deployed, RPT can be deployed immediately in such settings.

As shown in Figure 4, these sites have low bandwidth connections using leased line or VPN-enabled "virtual" wires. ISPs offering VPN services typically provide bandwidth and data delivery rate (or packet loss) guarantees as part of their SLA [1, 6]. In practice, their loss rate is often negligible because ISPs provision for bandwidth [24]. [3] Thus, the use of VPN and WAN optimizers effectively creates reliable RE "tunnels" on which RPT can operate. Important, real-time data can be sent with redundancy,

[3]Sprint's MPLS VPN [6] had a packet loss rate of 0.00% within the continental US from Mar 2011 to Feb 2012.

	RPT(3)	FEC(6,5)
Overhead	9%	22%
Data loss rate	8.0 x 10^{-6}	1.9 x 10^{-3}

Table 1: Comparison of FEC and RPT over CCN

and compete with other traffic when entering this tunnel. Packets will be lost when the total demand exceeds the capacity of the tunnel, but RPT flows will have protection against such loss. We evaluate this scenario in §6.2.

An alternative is to use traditional QoS schemes such as priority queuing. However, this typically involves deploying extra functionalities including dynamic resource allocation and admission control. For businesses not willing to maintain such an infrastructure, using RPT on and existing RE-enabled VPN would be an excellent option for delivering important, time-sensitive data.

Partial deployment: Not all routers in a network have to be content-aware to use RPT. The requirement for "RPT-safety" is that RE is deployed across bandwidth-constrained links [4], and non-RE links are well provisioned. This is because non-RE links end up carrying several duplicate packets. When such links are of much higher capacity, RPT causes no harm. Otherwise, it impacts network utilization and harms other traffic. In §6.2, we explore both cases through examples, and show how the network utilization and the other traffic on the network are impacted when RPT is used in an "unsafe" environment.

To ensure safe operation of RPT, one can detect the presence of RE on bandwidth-constrained links, and use RPT only when it would not harm other traffic. In this section, we outline two possible approaches for this, but leave details as a future work. One approach is to use end-point based measurement: for example, Pathneck [34] allows detection of bottlenecks based on available bandwidth. It sends traceroute packets in between load packets and infers (multiple) bottleneck location(s) from the time gap between returned ICMP packets. Similar to this, we can send two separate packet trains: one with no redundancy and the other with redundancy r but with the same bitrate. If all bandwidth constrained links are RE-enabled and RPT is safe to use on other links, the packet gap would not inflate on previously detected bottlenecks and the redundant packet trains would not report different bottleneck links. Another way is to use systems, such as iPlane [45] and I-path [47], which expose path attributes (e.g. available bandwidth) to end-hosts. These systems can easily provide additional information such as RE-functionality for end-hosts to check for RPT safety.

RPT over CCN: RPT also can be integrated with a broad class of content-aware networks, including CCN [37] and SmartRE [15]. In our technical report [31], we explore discuss how RPT can work atop SmartRE and wireless networks. Here, we focus on applying RPT to CCN.

[4]This matches the common deployment scenario for RE [14, 54].

Quality	Excellent	Good	Fair	Poor	Bad
PSNR (dB)	> 37	$31 \sim 37$	$25 \sim 31$	$20 \sim 25$	< 20

Table 2: User perception versus PSNR

In CCN, data consumers send "Interest" packets, and the network responds with at most one "Data" packet for each Interest. Inside the network, each router caches content and eliminates duplicate transfers of the same content over any link. CCN is designed to operate on top of unreliable packet delivery service, and thus Interest and Data packets may be lost [37].

We now compare RPT and FEC in CCN. Suppose real-time data is generated continuously, say k packets every 100 ms, and RTT is large. In an FEC-equivalent scheme for CCN, the content publisher would encode k data packets and add $(n-k)$ coded data packets, where $n > k$. The data consumer would then generate n Interest packets for loss protection. The receiver will be able to fully decode the data when more than k Interest and Data packets go through. However, up to n Interest/Data pairs will go through the network when there is no loss. In contrast, RPT does not code data packets, but generates redundant Interest packets. This obviously provides robustness against Interest packet loss. Moreover, when a Data packet is lost, subsequent redundant Interest packet will re-initiate the Data transfer. Since Interest packets are small compared to Data and duplicate Interests do not result in duplicate transfers of the Data, the bandwidth cost of redundancy is minimal. In RPT, at most k Data packets will be transferred instead of n in the FEC scheme.

To demonstrate the benefit more concretely, we take the Web page example from CCN and compare RPT and FEC over CCN. In the CCN-over-jumbo-UDP protocol case [37], a client generates three Interest packets (325 bytes) and receives five 1500-byte packets (6873 bytes) to fetch a Web page [37]. To compare RPT and FEC, we assume in RPT a redundancy parameter of 3 is used and in FEC the server adds one packet to the original data. Table 1 shows the overhead and data loss rate of each scheme at the underlying loss rate 2%. The data loss rates is the amount of data that could not be recovered. Even though the overhead of RPT is only 41% of that of FEC, it's data loss rate is 240 times better. To achieve equal or greater level of robustness than RPT($r = 3$), FEC has to introduce 11 times the overhead of RPT($r = 3$).

6 Evaluation

In this section, we answer four specific questions through extensive evaluation:

(i) **Does RPT deliver better video quality?** How well does it work in practice?

In §6.2, we show that RPT provides high robustness and low bandwidth overhead, which translate to higher quality for video applications.

(*ii*) **Is RPT sensitive to its parameter setting, or does it require fine tuning of parameters**?

In §6.3, we show that, unlike FEC, RPT is easy to use since careful parameter tuning is not necessary, and delay can be independently controlled with the delay parameter.

(*iii*) **How do RT flows affect other flows and the overall network behavior**?

In §6.4, we demonstrate that RT flows are effectively prioritized over non-RT flows on congested links and may occupy more bandwidth than their fair-share.

(*iv*) **Can we make RT flows adapt to network conditions and be TCP-friendly**?

We show in §6.5 that RT can also be made TCP-friendly, while retaining the key benefits.

6.1 Evaluation Framework

We use a combination of real-world experiments, network measurements and simulations. We implemented an RE encoder and decoder using Click [41], and created a router similar to that of Figure 1. Using this implementation, we create a hop-by-hop RE network in our lab as well as in Emulab. These serve as our evaluation framework.

We use implementation-based evaluation to show the overall end-to-end performance of RPT, and simulations to unravel the details and observe how it interacts with other cross traffic. To obtain realistic packet traces and loss patterns from highly multiplexed networks, we performed active measurements to collect real-world Internet packet traces. We also created background traffic and simulated RPT and FEC flows in a hop-by-hop RE network using the ns-2 simulator. These video flow packet traces are then fed into *evalid* video performance evaluation tool [40] to obtain the received video sequence with loss. For video, we used the *football* video sequence in CIF format, taken from a well-known library [7]. We used H.264 encoding with 30 frames per second. I-frames were inserted every second and only I- and P-frames were used to model live streams.

RE implementation: We implemented the Max-Match algorithm described in [14]. We further modified it to only store non-redundant packets in the packet cache. Therefore, sending redundant packets does not interfere with other cached content. We use a small cache of 4MB. The implementation of the encoder encodes a 1500 byte fully redundant packet to a 43 byte packet[5]. We also implemented RE in ns-2.

Evaluation metric: We use the standard Peak-to-Signal-to-Noise Ratio (PSNR) [52] as the metric for the video quality. PSNR is defined using a logarithmic unit of dB, and therefore a small difference in PSNR results in visually noticeable difference in the video. The MPEG committee reportedly uses a threshold of PSNR = 0.5dB to test the significance of the quality improvement [52]. Typical values for PSNR for encoded video are between 30 and 50 dB. Table 2 maps the PSNR value to a user perceived video quality [40].

[5]We do not compress network and transport layer headers. Thus, the packet may be compressed even further in practice.

6.2 End-to-end Video Performance

In this section, we evaluate the end-to-end performance of RPT and examine key characteristics.

Experimental setting: First, we use our testbed based on our hop-by-hop RE implementation, and create a streaming application that uses redundant packet transmission. We create a topology where an RE router in the middle connects two networks, one at 100Mbps and the other at 10Mbps. To create loss, we generate traffic from the well-connected network to the 10Mbps bottleneck link.

We generate a 1Mbps UDP video stream and long-running TCP flows as background traffic. We adjust the background traffic load to create a 2% loss on the video flow. We then compare the video quality achieved by RPT, Naive, and FEC that use the same amount of bandwidth. We use RPT that has 6% overhead ($r = 3, d = 2$), and FEC(10,9) with 10% overhead, which closely match in latency constraints with comparable overhead. Naive uses UDP without any protection.

Figure 5 shows the sending rate and the received data rate after the loss. The RPT and FEC senders respectively use about 6% and 10% of their bandwidth towards redundancy, while the Naive sender fully uses 1Mbps to send original data. The sending rates of the three senders are the same, within a small margin of error (1%). The Naive receiver loses 2% of the data and receives 0.98Mbps because of the loss. The FEC receiver only recovers about 66% of the lost data due to the bursty loss pattern. On the other hand, the RPT receiver receives virtually all original data sent. Note that only the amount of redundancy has slightly decreased. This is because when an original packet is lost, a subsequent redundant packet is naturally expanded inside the network.

As a result, the RPT flow gives much higher video quality. Figure 6 shows a snapshot of the video for RPT and Naive flows. Table 3 shows the video quality of an encoded video and the received video. The encoded video column shows the quality of video generated at the sender before packet loss. When RPT and FEC are used, the encoded video quality is slightly lower because of the bandwidth used towards redundancy. However, **because the RPT flow is highly robust against loss, it provides the best video streaming quality** (1.8 dB better than FEC and almost 6dB better than Naive).

RE-enabled Corporate VPN of §5 is the most common deployment scenario of RT in today's networks. To demonstrate the feasibility of this scenario, we set up a network of four routers in Emulab [26] and created an RE-enabled VPN tunnel that isolates the traffic be-

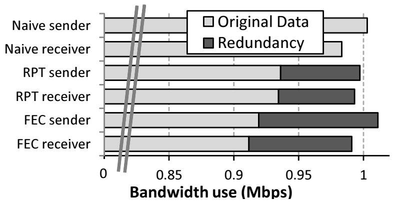
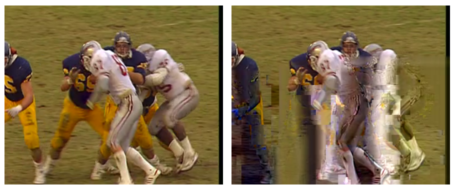


Figure 5: Bandwidth use



(a) RPT flow (b) Naive flow

Figure 6: Snapshot of the video

	Encoded	Received
RPT	37.3 dB	**37.1 dB**
FEC	36.9 dB	**35.3 dB**
Naive	37.5 dB	**31.4 dB**

Table 3: Average video quality (PSNR)

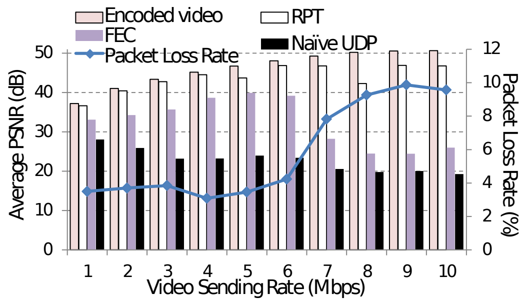


Figure 7: Video quality and loss rate for real traces

tween two remote offices similar to that of Figure 4. The physical links between two remote offices have 100Mbps capacity, and carries traffic from other customers. We generate cross traffic over the physical links that carries the VPN traffic so that the physical links experience congestive loss. We allocate 5Mbps of bandwidth to the VPN-enabled "virtual" wire, which is emulated using the priority queuing discipline from the Linux kernel's traffic control module. We introduced a 1Mbps video traffic and 5 TCP connections between the two remote offices, and compare RPT(3) and FEC(10,9) whose bandwidth overhead best matches to that of RPT(3), while adhering to the latency constraint. The video stream experiences a loss rate of around 2.7% and the tunnel's link utilization was nearly 100% in both cases. The resulting PSNR of the RT flow and FEC were 37.1dB and 34.1dB respectively. This result shows that RT also works well on the most common form of today's content-aware networks.

Real traces: To study the performance of RPT in a realistic wide-area setting, we collected a real-world packet trace. We generated UDP packets from a university in Korea to a wired host connected to a residential ISP in Pittsburgh, PA. The sending rate was varied from 1Mbps to 10Mbps, each run lasting at least 30 seconds. The round-trip-time was around 250ms, which indicates that retransmissions would violate the timing constraint of an interactive stream.

Assuming that the packet loss rates do not change significantly with RPT[6], we apply the loss pattern obtained from the measurement to an RPT flow, an FEC flow and a Naive UDP flow. For RPT, we use a redundancy parameter of $r = 3$ and a delay parameter $d = 2$. For FEC, we choose the parameters so that the overhead matches closest to that of RPT, while the additional latency incurred by FEC at the receiver does not exceed 150ms, which results in different parameters for different sending rates. Figure 7 shows the video quality for each scheme. The solid line shows the packet loss rate. The `Encoded video` bar shows the ideal PSNR without any loss when all the bandwidth is used towards sending original data, presented as a reference. As the sending rate increases, the quality of the encoded video increases. However, the loss rate from Korea to U.S. was 3.5% at 1Mbps but increased to 9.8% at 10Mbps as the sending rate increases. Because of the high loss rate, the naive UDP sender performs poorly (PSNR well under 30dB). FEC achieves better performance than naive, but much worse than RPT especially under high loss rates. In contrast, **RPT gives the best performance, closely following the quality of the encoded video until the loss rate is about 8%**. Even at the higher loss rates, the impact on quality is much less than the FEC scheme. This is because RPT gives much better protection against loss than FEC at similar overhead. [7]

6.3 Parameter Selection and Sensitivity

In this section, we provide an in-depth performance evaluation of RPT. In particular, we compare RPT and FEC's parameter sensitivity using simulations that produce packet loss patterns of highly multiplexed networks with realistic cross traffic.

Simulated RE Network: We use the ns-2 simulator to create a realistic loss pattern by generating a mix of HTTP and long-running TCP cross traffic. We use a dumbbell topology with the RE-enabled bottleneck link capacity set to 100Mbps, and simulate a hop-by-hop RE network and RPT flows. We generate 100 long-running TCP flows and 100 HTTP requests per second. We used the packmime [22] module to generate representative HTTP traffic patterns. We also generate ten video flows each having

[6]We later verify this and see how RPT affects the loss rate in §6.4.

[7]Large drop in PSNR at 8 Mbps is an artifact of the video's resolution being too small compared to its encoding rate and a particular pattern of bursty data loss. When the video compression gets nearly lossless, even a small data loss causes PSNR to drop sharply. In addition, two original packets that are close together in sequence were lost by coincidence in 8Mbps RPT. This had a more detrimental effect on the PSNR value because the lost data belonged to the same video frame. The actual data loss rate of the 8Mbps RPT flow was 0.165%, which is less than 0.174% of the 9Mbps RPT flow.

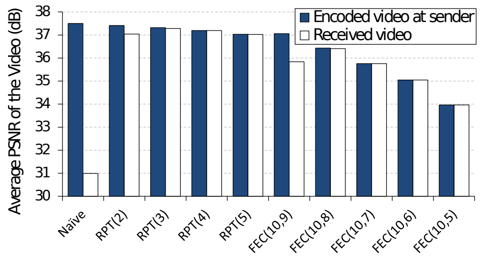


Figure 8: RPT's performance is much less sensitive to its parameter setting.

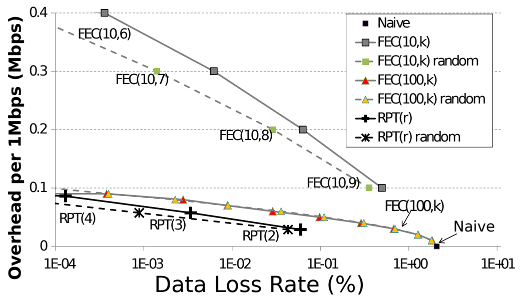


Figure 9: Percent data loss rate and overhead: RPT greatly outperforms FEC with small group size.

1Mbps budget regardless of the loss protection scheme it uses. The results presented are averages of ten runs with each simulating five minutes of traffic. We first look at the final video quality seen by the end receiver under different parameter settings, and then analyze the underlying loss rate and overhead.

How do RPT flows and FEC flows perform with different redundancy parameters? For RPT, we vary the redundancy parameter r from 2 to 5, while fixing the delay parameter d to 2. For FEC, we use a group size $n = 10$ to meet the latency constraints and vary the number of original data packets k from 5 to 9.

Figure 8 shows the quality of the video seen by the receiver compared to the encoded quality at the sender. The result shows that **RPT performs better than FEC's best, and its performance is stable across different parameter settings.** In contrast, FEC's performance is highly sensitive to the parameter selection. Therefore, with FEC, the sender has to carefully tune the parameter to balance the amount of redundancy and encoding rate.

Figure 9 shows the underlying data loss rate and overhead of the video flows. The x-axis shows the data loss rate in log-scale, and the y-axis shows the amount of overhead. All video flows experience $\sim$2% packet loss. For comparison, the performance of RPT and two FEC families (n=10, 100) under uniform random loss (dotted lines) are also shown. RPT(4)'s data loss rate is several orders of magnitude lower than the loss rate of FEC(10,9) whose overhead is similar. RPT(4) even performs better than FECs with large group size, such as FEC(100,91), whose latency exceeds the real-time constraint. FEC(10,7) achieve similar data loss rate to RPT(3) but has 6 times the overhead, which translated to 2dB difference in PSNR.

The gap between the uniform loss and actual loss lines in Figure 9 represents the effect of bursty loss performance. FEC(10,k) and RPT show a relatively large gap between the two lines. Analyzing the underlying loss pattern, we observe that within a group of 10 packets, losses of 2 to 4 packets appear more frequently in the actual loss pattern. This shows that the underlying traffic is bursty. On the other hand, loss bursts of more than 5 occur less frequently in the actual pattern because TCP congestion control eventually kicks in.

We now show how the parameter should be set in RPT.

How should we choose parameters in RPT? To answer this question, we study how loss rate and burstiness of loss affect the performance of RPT. First, we use the random loss pattern and vary the packet loss rate from 1% to 8%. For RPT, we vary the redundancy parameter from 2 to 4, but fix the delay parameter at 2. For each loss rate, the average PSNR of a naive sender and an RPT sender is shown in Figure 10. It shows that video quality of RPT(3) is virtually immune to a wide range of packet losses; the average PSNR for RPT(3) under 8% loss only decreased by 0.25dB compared to the zero-loss case. We, therefore, use $r = 3$ in the rest of our evaluation.

Second, we look at the role of the delay parameter under bursty loss. For reference, we generate a 2% random loss, which on average has 1 lost packet every 50 packets. We then create bursty loss patterns by reducing the number of packets between losses by up to 15 and 35, while keeping the average loss rate the same. The three cases are named as `Uniform random`, `Burst+`, and `Burst++` respectively. Figure 11 shows the data loss rate of RPT with different delay parameters under the three loss conditions. An increase in burstiness negatively impacts the data loss rate, but as delay is increased

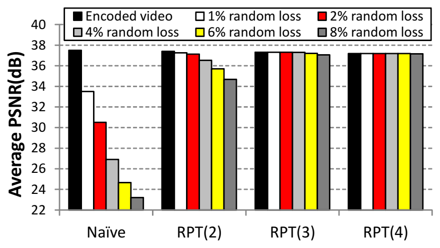


Figure 10: Video quality under 1 to 8% loss. RPT(3) steadily delivers high quality even under high loss.

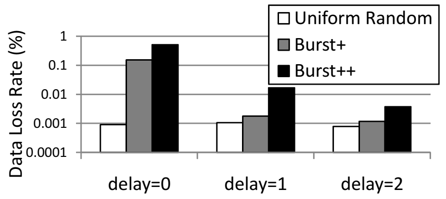


Figure 11: Bursty loss increases the data loss especially when the delay parameter is small.

	FEC(10,6)	FEC(100,92)	RPT(3)
No sender buffering	240ms	2400ms	60ms
Sender buffering	180ms	1300ms	-

Table 4: Maximum one-way delay of RT and FEC

the negative impact is decreased. In general, a large delay parameter gives more protection against bursty loss, but incurs additional latency. **We use $d = 2$ and $r = 3$ for RPT because of its superior performance in wide range of loss conditions**.

How much latency is caused by RPT and FEC flows? A loss might be recovered by subsequent redundant packets; here we quantify the delay in this. In RPT(r) with delay d, the receiver buffer must hold $d \cdot r$ packets. So the delay in RPT is $d \cdot r \cdot intv$, where $intv$ is the interval between packets. The RPT sender needs no additional buffering, as it transmits the packet as soon as a packet is generated from the encoder. In FEC, two alternatives exist where one does sender buffering to pace packets evenly and the other doesn't but further delays the transmission of redundant coded packets [31]. Table 4 shows the delay caused by RPT and FEC for 1Mbps RPT(3) with $d = 2$ and FEC streams that exhibit similar data loss rate from Figure 9. We see that **RPT gives a significantly lower delay than FEC schemes of similar strength in loss protection**, and FEC(100,k) is not suitable for delay-sensitive communication.

How do RPT and FEC flows perform under extreme load? One might think that in a highly congested link with a high fraction of RPT traffic, RPT flows would constantly overflow the buffer and the performance would drop. To create such a scenario with increased traffic load, we vary the fraction of video traffic in the link from 10% to 90%, while keeping the number of background TCP connections and bottleneck bandwidth the same. Detailed evaluation is provided in [31]. In summary, we find that **RPT flows achieve close-to-ideal video quality, and better quality compared to the best FEC scheme** in all cases (10% to 80%) except for one very extreme case (90%) with very heavy cross traffic creating loss rate $> 10\%$. The extreme case we portray in our experiment is unlikely to occur in practice for two reasons: 1) The loss rates in practice are likely to be much lower. 2) Even the aggressive estimate suggests that no more than 15% of traffic in future will be real-time in nature [3].

6.4 Impact of RPT on the Network

We now examine the effect of RPT flows on other cross traffic and the network. For this evaluation, we use the same simulation setup and topology described in §6.3.

How do other TCP flows perform? We look at the impact on two different types of TCP flows: long-running TCP flows and HTTP-like short TCP flows.

To evaluate the *impact on long-running TCP*, we send

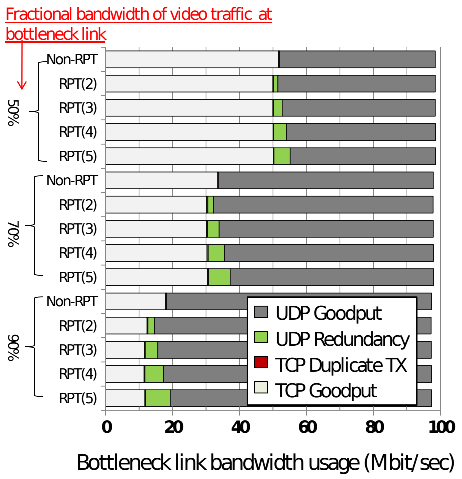


Figure 12: Breakdown of bottleneck link utilization: RPT flows do not impact the link utilization. RPT flows are prioritized over competing TCP flows.

100 long-running TCP flows and a varying number of video flows to vary the fraction of video traffic on the bottleneck link (from 10 to 90%). We also vary the redundancy parameter from 0 (Non-RPT) to 5. We use a small router buffer size of $\frac{2 \cdot B \cdot RTT}{\sqrt{100}}$ [16] to maximize the negative impact.

Figure 12 shows the bottleneck link traffic decomposition in four categories[8]: UDP goodput, UDP redundancy, TCP duplicate, TCP goodput. UDP goodput is the bandwidth occupied by the original packet and the UDP redundancy represents the bandwidth occupied by the compressed packets. TCP goodput represents packets contributing to application throughput, and TCP duplicate Tx shows the amount of duplicate TCP packets received.

We observe that **1) the bandwidth utilization is not affected by RPT, and 2) RPT flows are effectively prioritized over non-RPT TCP flows.** In all cases the bottleneck bandwidth utilization was over 97.5%; TCP fills up the bottleneck even if the router queue is occupied by many decompressed redundant packets. TCP throughput, on the other hand, is impacted by the RPT flow especially when the network is highly congested; e.g. in the 90% video traffic case (bottom-most bars), TCP goodput (white region) decreases when RPT is used. This is because when RPT and non-RPT cross traffic competes, even though they experience the same underlying packet loss rates, for RPT flows packet loss do not directly translate into throughput loss. With redundant transmissions, the network recovers the loss through subsequent uncompressed redundant packets, effectively prioritizing the RPT flows.

To see the *impact on HTTP-type short flows*, we look at how RT changes the response time of short flows under

[8]Only a subset of results (video traffic occupying 50% to 90% of bottleneck) shown for brevity, but all cases confirm the same results.

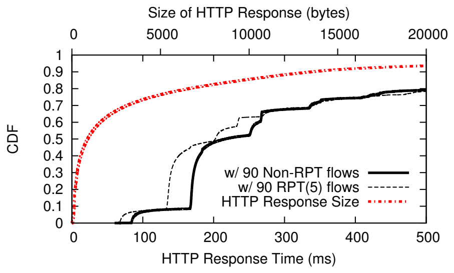


Figure 13: Response time and size for short HTTP flows (long flows omitted for clarity).

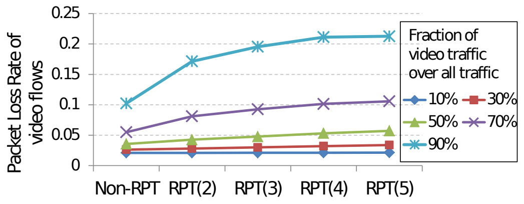


Figure 14: Impact on loss rate due to RPT flows.

the setup described in §6.3. Figure 13 shows the CDFs of the size of the response messages, and the response times. To highlight the difference, we only show response times when 90% of the traffic is video and RPT(5) is used, but the trend is visible across all cases.

The response time for short flows decreases in the presence of RPT flows. Since redundant packets in the queue are compressed when they are sent out, the service rate of the queue increases with RT. Therefore the queuing delay is reduced, which results in a decrease in the response time. However, for larger flows (tail end of the figure) the response time actually increases, as they behave more like long-running TCP flows, which obtain less throughput under congestion (Figure 12).

How does network behavior change with RPT flows? There are subtle changes in loss rate and queuing delay. *Loss rate:* Figure 14 shows the packet loss rate of UDP video flows at the bottleneck router with varying amount of RPT traffic. **When the fraction of video traffic is moderate, adding more redundancy has little impact on the underlying packet loss of the video flow.** This is because while redundant packets increase the load, they also increase the service rate of the link. However, we observe that when RPT flows dominate the bottleneck link, the underlying loss rate for video flows goes up as redundancy increases. The underlying reason for increased loss is that under congestion RPT flows compete with each other for bandwidth when most of the traffic is from RPT flows. However, in §6.3, we saw that even under such extreme conditions RPT performs better than FEC. *Queuing Delay:* Figure 15 shows the average queuing delay with varying redundancy parameters and varying number of RPT flows. **Redundant packets decrease queuing delay.** This is because redundant packets appear as decompressed at the router queue, but are sent out compressed at the bottleneck link. Therefore, as the number of redundant packets increase, service rate becomes faster.

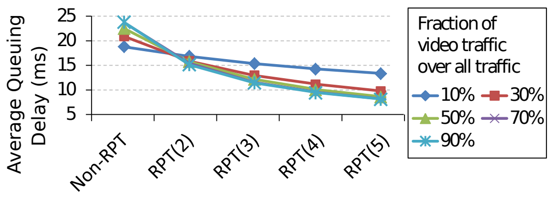


Figure 15: Queuing delay is reduced with RPT flows.

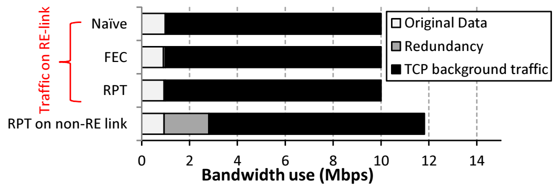


Figure 16: *No Harm:* Bandwidth use on a non-RE link in the no harm case.

What's the impact of partial content-awareness? In §5, we noted that RT may cause harm in partially content-aware networks and should be used only after detecting RT-safety. Here, demonstrate both the *no-harm* and the *harm* case, and quantify the impact using our experimental testbed, which has a 10Mbps and a 100Mbps RE link.

To demonstrate *no-harm*, we disabled the RE encoder on the non-bottleneck 100Mbps links of our testbed. We then generated an RPT flow and TCP background flows through this 100Mbps link and the 10Mbps RE-enabled link. Figure 16 shows the traffic on both links. The RPT flow occupying 1Mbps on an RE-enabled bottleneck link introduces almost 2Mbps of overhead (`redundancy`) on the non-RE 100Mbps link. However, this causes no harm since the 100Mbps link is not bandwidth constrained.

To demonstrate *harm*, we reduced the capacity of the non-RE link to 15Mbps, and introduced four 1Mbps video flows and a TCP flow. Table 5 compares bandwidth use on the 10Mbps RE link when the video flows are sent with and without redundancy. When there's no redundancy (`Non-RPT`), the link utilization of the 10Mbps link is 97%. When RPT(3) is used, the 4Mbps video flows occupy 11.2Mbps on the non-RE link. This shifts the bottleneck to be the 15Mbps non-RE link, which forces the 10Mbps RE-link and the network to be under-utilized at 76%. This verifies that in a partial deployment setting, detecting RT-safety is important as discussed in §5.

6.5 TCP-Friendly RPT

RPT flows do not give up bandwidth as easily under congestion. In §4, we discussed an alternative that makes RPT flows achieve fair bandwidth sharing using TCP-friendly rate control. In particular, we showed that incorporating TFRC requires careful adjustment of loss event rate, and explained how it should be done in two distinct loss

	Non-RPT	RPT(3)
TCP traffic (Mbps)	5.7	3.6
Video traffic (Mbps)	4.0	4.0
Total (Utilization)	9.7 (97%)	7.6 (76%)

Table 5: *Harm:* Bandwidth use on a RE-link.

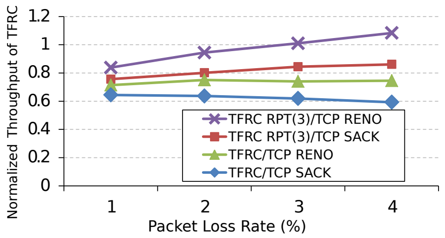


Figure 17: TFRC RPT under random loss.

patterns: *Uniform random* and *Temporal* packet loss.

Is TFRC RPT TCP-friendly? We first evaluate our scheme under the *two extreme loss patterns* created artificially, and evaluate it under a more realistic loss pattern.

Uniform random loss: In this setting, TFRC RPT behaves in a TCP friendly manner without any adjustment in the loss estimation. Figure 17 shows the normalized throughput of TFRC and TFRC RPT(3) with respect to TCP Sack and Reno under 1% to 4% random loss. TFRC RPT(3) performs slightly better than TFRC because multiple packet losses within an RTT are counted as one loss event, and therefore the loss event rate for RPT(3) is slightly lower than that of normal TFRC.

Temporal packet loss: Here, we adjust the loss event rate of TFRC RPT(r) to be r times the observed loss event rate. To validate TCP-friendliness, we evaluated the performance of TFRC RPT(3) and TFRC under the same temporal loss pattern. To create such a pattern, we generated the same cross traffic, but artificially modified the router's queue so that redundant packets do not increase the queue length. Indeed, the performance difference of the two was less than 3% with the adjusted loss event rate.

Realistic environment: The two cases appear in an inter-mixed way in practice. As discussed in §3, an adjustment factor between 1 and r is sufficient for TCP friendliness. To create realistic loss patterns, we ran TFRC with competing TCP flows. The same dumbbell topology with 1 Gbps bottleneck link capacity is used. We vary the number of competing TCP flows from 200 to 2000. Each flow's RTT is randomly selected between 40ms and 200ms. Among the TCP flows, five of them are set to have the same RTT as the TFRC flows. We compare the relative throughput of TFRC flows to the average throughput of TCP flows. Our result shows TFRC RPT(3)'s performance reasonably matches that of TCP when $\alpha = 1.5$ across various loss rates. Figure 18 shows the normalized TFRC RPT(3)'s performance with respect to TCP Reno and TCP Sack. The result show that **TFRC RPT is TCP friendly.**

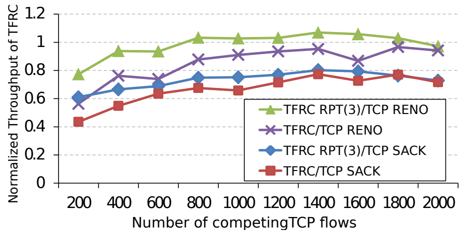


Figure 18: TFRC RPT exhibit TCP friendliness.

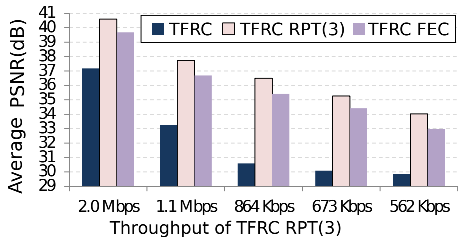


Figure 19: Video quality comparison.

Video quality: We created video streams using TFRC and TFRC RPT; in either case, we output video according to the TFRC's or TFRC RPT's sending rate.[9] We compare the video quality of normal TFRC, normal TFRC with FEC, and TFRC RPT under the same cross traffic. We vary the cross traffic to create TFRC flows whose throughputs range from 562Kbps to 2.0Mbps. For TFRC with FEC, we choose the parameter which gives the best PSNR with delay under 150ms. Figure 19 shows the video quality achieved by the TFRC flows. **We see that TFRC RPT gives the best video quality in all cases.**

7 Conclusion

This paper explores issues arising from the confluence of two trends – growing importance and volume of real-time traffic, and the growing adoption of content-aware networks. We examine a key problem at this intersection, namely that of protecting real-time traffic from data losses in content-aware networks. We show that adding redundancy in a way that network understands reduces the cost and increases the benefits of loss protection quite significantly. We refer to our candidate loss protection approach as redundant transmission (RT). Using Redundant Packet Transmission (RPT) in redundancy-elimination networks [14] as an example, we highlight various features of RT and establish that is a promising candidate to use in several practical content-aware networking scenarios. We show that RT decreases the data loss rate by orders-of-magnitude more than typical FEC schemes applicable in live video communications, and delivers higher quality video than FEC using the same bandwidth budget. RT provides fine-grained control to signal the importance of data and satisfies tight delay constraints. Yet, it is easy to use as its performance is much less sensitive to parameter selection. We show that constant bitrate RT flows

[9]For more details, refer to our technical report [31].

are prioritized over non-RT flows, but can share bandwidth fairly by incorporating TCP friendly rate control into RPT. We also show that RT provides an efficient and cost-effective loss protection mechanism in other general content-aware networks.

Acknowledgments

We thank our shepherd Xiaowei Yang and anonymous reviewers for their feedback. This research was supported in part by the National Science Foundation under awards CNS-1040757, CNS-0905134, CNS-0905277 and CNS-1040801.

References

[1] AT&T Businees Service Guide - AT&T VPN Service. `http://new.serviceguide.att.com/portals/sgportal.portal?_nfpb=true&_pageLabel=avpn_page`, 2011.

[2] Cisco Wide Area Application Services (WAAS) Software. `http://www.cisco.com/en/US/prod/collateral/contnetw/ps5680/ps6870/prod_white_paper0900aecd8051d5b2.html`, 2009.

[3] Cisco visual networking index: Forecast and methodology, 20092014. `http://www.cisco.com/`, 2010.

[4] Magic Quadrant for WAN Optimization Controllers. `http://www.gartner.com/technology/media-products/reprints/riverbed/article1/article1.html`, 2010.

[5] Juniper Networks Datasheet. `http://www.juniper.net/us/en/local/pdf/datasheets/1000113-en.pdf`, 2009.

[6] Sprint Network Performance. `https://www.sprint.net/sla_performance.php?network=pip`, 2012.

[7] YUV CIF reference videos. `http://www.tkn.tu-berlin.de/research/evalvid/cif.html`, 2010.

[8] Cisco Internal WAAS Implementation. `http://blogs.cisco.com/ciscoit/cisco_internal_waas_implementation/`, 2010.

[9] Riverbed Cloud Products. `http://www.riverbed.com/us/products/cloud_products/cloud_steelhead.php`, 2011.

[10] Riverbed Customer Stories. `http://www.riverbed.com/us/customers/index.php?filter=bandwidth`, 2011.

[11] Riverbed Steelhead Mobile. `http://www.riverbed.com/us/products/steelhead_appliance/steelhead_mobile/`, 2011.

[12] A. Albanese, J. Blöer, J. Edmonds, M. Luby, and M. Sudan. Priority encoding transmission. *IEEE Transactions on Information Theory*, 42, 1994.

[13] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan. Data center TCP (DCTCP). In *Proc. ACM SIGCOMM*, 2010.

[14] A. Anand, A. Gupta, A. Akella, S. Seshan, and S. Shenker. Packet caches on routers: the implications of universal redundant traffic elimination. In *Proc. ACM SIGCOMM*, 2008.

[15] A. Anand, V. Sekar, and A. Akella. SmartRE: an architecture for coordinated network-wide redundancy elimination. In *Proc. ACM SIGCOMM*, 2009.

[16] G. Appenzeller, I. Keslassy, and N. McKeown. Sizing router buffers. In *Proc. ACM SIGCOMM*, 2004.

[17] M. Balakrishnan, T. Marian, K. Birman, H. Weatherspoon, and E. Vollset. Maelstrom: transparent error correction for lambda networks. In *Proc. USENIX NSDI*, 2008.

[18] A. C. Begen and Y. Altunbasak. Redundancy-controllable adaptive retransmission timeout estimation for packet video. In *Proc. ACM NOSSDAV*, 2006.

[19] J.-C. Bolot, S. Fosse-Parisis, and D. Towsley. Adaptive FEC-based error control for internet telephony. In *Proc. IEEE INFOCOM*, 1999.

[20] O. Boyaci, A. Forte, and H. Schulzrinne. Performance of video-chat applications under congestion. In *Proc. IEEE ISM*, Dec. 2009.

[21] J. W. Byers, M. Luby, M. Mitzenmacher, and A. Rege. A digital fountain approach to reliable distribution of bulk data. In *Proc. ACM SIGCOMM*, 1998.

[22] J. Cao, W. Cleveland, Y. Gao, K. Jeffay, F. Smith, and M. Weigle. Stochastic models for generating synthetic HTTP source traffic. In *Proc. IEEE INFOCOM*, volume 3, 2004.

[23] P. A. Chou, H. J. Wang, and V. N. Padmanabhan. Layered multiple description coding. In *Proc. Packet Video Workshop*, 2003.

[24] Cisco. Deploying guaranteed-bandwith services with mpls. `http://www.cisco.com/warp/public/cc/pd/iosw/prodlit/gurtb_wp.pdf`, 2012.

[25] L. De Cicco, S. Mascolo, and V. Palmisano. Skype video responsiveness to bandwidth variations. In *Proc. ACM NOSSDAV*, 2008.

[26] Emulab. Emulab. `http://www.emulab.net/`.

[27] N. Feamster and H. Balakrishnan. Packet loss recovery for streaming video. In *Proc. International Packet Video Workshop*, 2002.

[28] S. Floyd, M. Handley, J. Padhye, and J. Widmer. Equation-based congestion control for unicast applications. In *Proc. ACM SIGCOMM*, 2000.

[29] P. Frossard and O. Verscheure. Joint source/FEC rate selection for quality-optimal MPEG-2 video delivery. *IEEE Transactions on Image Processing*, 10(12), Dec. 2001.

[30] A. Hagedorn, S. Agarwal, D. Starobinski, and A. Trachtenberg. Rateless coding with feedback. In *Proc. IEEE INFOCOM*, 2009.

[31] D. Han, A. Anand, A. Akella, and S. Seshan. RPT: Re-architecting loss protection for content-aware networks. Technical Report TR-11-117, Carnegie Mellon Univ., 2011.

[32] D. Han, A. Anand, F. Dogar, B. Li, H. Lim, M. Machado, A. Mukundan, W. Wu, A. Akella, D. Andersen, J. Byers, S. Seshan, and P. Steenkiste. XIA: An architecture for an evolvable and trustworthy Internet. In *Proc. USENIX NSDI*, Apr. 2012.

[33] U. Horn, K. Stuhlmller, E. E. Herzogenrath, M. Link, and B. Girod. Robust internet video transmission based on scalable coding and unequal error protection. *Signal Processing: Image Communication*, 1999.

[34] N. Hu, L. E. Li, and Z. M. Mao. Locating Internet bottlenecks: Algorithms, measurements, and implications. In *Proc. ACM SIGCOMM*, 2004.

[35] Infineta. Velocity Dedupe Engine. `http://www.infineta.com/technology/reduce`, 2011.

[36] ITU-T. Recommendation G.114 one-way transmission time.

[37] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard. Networking named content. In *Proc. ACM CoNEXT*, 2009.

[38] S. Jakubczak and D. Katabi. A cross-layer design for scalable mobile video. In *Proc. ACM MobiCom*, 2011.

[39] K. Jamieson and H. Balakrishnan. PPR: Partial packet recovery for wireless networks. In *Proc. ACM SIGCOMM*, Aug. 2007.

[40] J. Klaue, B. Rathke, and A. Wolisz. EvalVid - a framework for video transmission and quality evaluation. In *Proc. Performance TOOLS*, 2003.

[41] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek. The click modular router. *ACM TOCS*, 2000.

[42] T. Koponen, M. Chawla, B.-G. Chun, A. Ermolinskiy, K. H. Kim, S. Shenker, and I. Stoica. A Data-Oriented (and Beyond) Network Architecture. Aug. 2007.

[43] H. Liu and M. El Zarki. Performance of H.263 video transmission over wireless channels using hybrid ARQ. *IEEE JSAC*, 15(9), Dec. 1997.

[44] D. Loguinov and H. Radha. On retransmission schemes for real-time streaming in the Internet. In *Proc. IEEE INFOCOM*, 2001.

[45] H. V. Madhyastha, T. Isdal, M. Piatek, C. Dixon, T. E. Anderson, A. Krishnamurthy, and A. Venkataramani. iPlane: An information plane for distributed services. In *Proc. 7th USENIX OSDI*, Nov. 2006.

[46] S. McCanne, M. Vetterli, and V. Jacobson. Low-complexity video coding for receiver-driven layered multicast. *IEEE JSAC*, 15(6), 1997.

[47] K. Nakauchi and K. Kobayashi. An explicit router feedback framework for high bandwidth-delay product networks. *Comput. Netw.*, 51, May 2007.

[48] L. Pantel and L. C. Wolf. On the impact of delay on real-time multiplayer games. In *Proc. NOSSDAV*, 2002.

[49] C. Papadopoulos. Retransmission-based error control for continuous media applications. In *Proc. NOSSDAV*, 1996.

[50] R. Puri and K. Ramchandran. Multiple description source coding using forward error correction codes. In *Proc. Asilomar conference on signals, systems, and computers*, 1999.

[51] I. Rhee. Error control techniques for interactive low-bit rate video transmission over the internet. In *Proc. ACM SIGCOMM*, 1998.

[52] D. Salomon. *Data Compression; the Complete Reference*. Springer, 2007.

[53] S. Sen, N. K. Madabhushi, and S. Banerjee. Scalable wifi media delivery through adaptive broadcasts. In *Proc. USENIX NSDI*, 2010.

[54] N. T. Spring and D. Wetherall. A protocol-independent technique for eliminating redundant network traffic. In *Proc. ACM SIGCOMM*, 2000.

[55] A. Suissa, J. Mellor, F. Lohier, and P. Garda. A novel video packet loss concealment algorithm & real time implementation. In *Proc. DASIP*, 2008.

[56] J. Wang and D. Katabi. ChitChat: Making video chat robust to packet loss. Technical Report MIT-CSAIL-TR-2010-031, MIT, July 2010.

[57] X. Zhu, R. Pan, N. Dukkipati, V. Subramanian, and F. Bonomi. Layered Internet video engineering (LIVE): Network-assisted bandwidth sharing and transient loss protection for scalable video streaming. In *Proc. IEEE INFOCOM*, 2010.