Latency and Fairness Trade-Off for Thin Streams using Redundant Data Bundling in TCP

Bendik R. Opstad[1,2], Jonas Markussen[1], Iffat Ahmed[1], Andreas Petlund[1], Carsten Griwodz[1,2], and Palvorsen[1,2]

[1]Simula Research Laboratory, Norway
[2]IFI, University of Oslo, Norway
bendiko@ifi.uio.no, {jonassm,iffat,apetlund,griff,paalh}@simula.no

***Abstract*—Time-dependent applications using TCP often send thin-stream traffic, characterised by small packets and high inter-transmission-times. Retransmissions after packet loss can result in very high delays for such flows as they often cannot trigger fast retransmit. Redundant Data Bundling is a mechanism that preempts the experience of loss for a flow by piggybacking unacknowledged segments with new data as long as the total packet size is lower than the flow maximum segment size. Although successful at reducing retransmission latency, this mechanism had design issues leaving it open for abuse, effectively making it unsuitable for general Internet deployment. In this paper, we have redesigned the RDB mechanism to make it safe for deployment. We improve the trigger for when to apply it and evaluate its fairness towards competing traffic. Extensive experimental results confirm that our proposed modifications allows for inter-flow fairness while maintaining the significant latency reductions from the original RDB mechanism.**

***Index Terms*—Data-bundling, thin streams, TCP, latency, fairness.**

I. Introduction

The main focus of TCP's development has been on maximizing the *throughput*, that is, to achieve the smallest possible time for transferring a large chunk of data correctly from sender to receiver. The focus has been on this *finishing time*, and the TCP streams that transport data in this manner are called *greedy streams*.

However, TCP is also used by time-dependent applications that generate small data segments with high inter-transmission times (ITTs). Such applications include classical *telnet* and *ssh*, but also computer games, high frequency trading, screen sharing applications, and firewall-friendly fallback options for audio and video conferences. Such applications have completely different characteristics compared to greedy streams, and in Table 1, we see the typical small payload sizes (far below the common maximum transmission unit (MTU) of 1500 bytes for IP packets), low bandwidth requirements and high packet ITT statistics for a selection of representative applications. Furthermore, all of these applications depend on small maximum *per-packet* latency to provide a good service rather than high throughput. We call the streams that are generated by these time-dependent applications *thin streams* [1], and we argue that they deserve special consideration because they suffer much more than greedy streams from the effects of packet loss. With traffic patterns that prevent them from utilizing fast retransmit, they suffer from all the penalties associated with timeout retransmission, including head-of-line-blocking (HOLB) in the receiver-side stack, congestion window (CWND) collapse, and exponential backoff in case of repeated packet loss.

Delays due to loss recovery have attracted increasing attention in recent years, and in order to reduce delays, several mechanisms have been proposed that aim at changing how TCP *reacts* to loss by reducing the time before a lost packet is retransmitted, for example faster retransmission and changing timers. Another intuitive idea is to apply *proactive* or predictive techniques that hide packet loss from the usual TCP mechanisms altogether, for instance forward error correction (FEC). However, the reactive mechanisms still provide high maximum latencies, while the proactive are hard to deploy in today's networks because they require both sender and receiver side modifications.

In order to reduce latencies compared to reactive approaches and avoid receiver-side modifications, we have earlier proposed redundant data bundling (RDB) [3]. It is a variant of the proactive idea, but it is a backwards-compatible sender-side only modification to TCP. Unfortunately, the original RDB can be abused since congestion loss is hidden from the congestion control, and the amount of bundling is independent of the delay and loss characteristics of the specific connection. Before a public deployment of RDB, such problems must be solved. In this paper, we address these challenges. The revised RDB mechanism, implemented in the Linux kernel, enables a more aggressive (re)transmission mode for thin streams, and the per-packet latencies can be considerably reduced. RDB maintains compatibility with existing TCP implementations, which allows immediate deployment.

We have imposed a limitation on the mechanism, limiting its activation to when the stream is considered *thin*. We have redefined and evaluated the *thin-stream* classification method used to trigger RDB, and we have experimentally evaluated the new RDB implementation with respect to 1) the sweet-

Application	Payload size (bytes)			Packet inter-arrival time (ms)				Percentile		Avg bandwidth requirement	
	avg	min	max	avg	med	min	max	1	99	(pps)	(bps)
VNC (from client)	8	1	106	34	8	0	5451	0	517	29.412	17K
Skype (2 users)	236	14	1267	34	40	0	1671	4	80	29.412	69K
SSH text session	48	16	752	323	159	0	76610	32	3616	3.096	2825
Anarchy Online	98	8	1333	632	449	7	17032	83	4195	1.582	2168
World of Warcraft	26	6	1228	314	133	0	14855	0	3785	3.185	2046
Age of Conan	80	5	1460	86	57	0	1375	24	386	11.628	12K

Table 1: Examples of thin stream packet statistics based on analysis of TCP packet traces [2].

spot between the best possible latency gain and the level of aggressiveness due to increased packet sizes; 2) the reduction of per-packet latency that is achieved while remaining fair to competing traffic; and 3) the removal of opportunities to abuse RDB for gaining an unfair advantage over competing traffic. Our experimental results show that by proactively retransmitting unacknowledged data segments, RDB is able to significantly reduce the application-layer latency caused by HOLB and retransmission delays even with the new safeguards in place.

The rest of the paper is organized as follows: Section II describes related work in the context of thin streams. Section III gives an overview of how to alleviate the latency issues for thin streams by using the RDB scheme and also presents our improved solution. Our experiments are presented in Section IV, and the results are discussed Section V. Finally, Section VI concludes the findings and suggests future work.

II. Related work

Applications like online games, remote control systems, high-frequency trading and sensor networks provide a hugely increased utility when their per-packet latencies are at their lowest. They have in common that they produce mostly traffic with *thin-stream* characteristics, consisting of small packet payloads and high ITTs. An analysis of latencies in the online game Anarchy Online [1] showed that TCP introduces disproportionately large delays caused by packet loss for these types of streams. The reason identified is that most of the retransmissions are caused by retransmission timeouts (RTOs), which occurs because game traffic has very few packets in flight (PIF) and is unable to trigger a traditional *fast retransmit*. The same observation has been made for other streams with thin-stream characteristics. For example, measurements have shown that around 10% of the connections to Google services experience at least one packet loss, and 77% of these losses were repaired by RTOs [4]. The flows that experience loss take five times longer to complete than the flows without loss.

Such delays as a result of loss recovery have attracted increasing attention in recent years. Whereas early retransmit (ER) [5] and tail loss probe (TLP) [6], [7] are meant to improve stream finishing times that are hindered by losses of the last packets in a stream, others such as linear retransmission timeout (LT) [8], modified fast retransmit (mFR) [8] and RTO Restart (RTOR) [9] focus on the reduction of retransmission delays during the stream's lifetime. They reduce the time before data becomes available to the receiving application, not only for the retransmitted packet, but also for those that experience HOLB due to TCP's ordered delivery. These approaches change TCP's reaction to packet loss with the goal of retransmitting faster. In low latency application scenarios like in Table 1, these *reactive* mechanisms improve the situation, but it still takes at least an RTT to detect the loss, and further improvements are required.

A different approach is to be *proactive* and protect data to avoid retransmissions by transmitting data redundantly even before any loss signals are received. The data segments could simply be transmitted multiple times, which would alleviate sporadic losses as long as one of the packets containing the data segment gets through. An example of this is to transmit every data segment twice [4]. Moreover, FEC can be used to encode redundant data for loss recovery on the client side using TCP FEC [10], [11]. These proactive approaches reduce latency further, but they increase the number of packets sent and require sender- as well as receiver-side modifications.

In summary, the existing reactive techniques still require long time to detect a packet loss, and the proactive solutions suffer from overhead and receiver-side modifications. A proactive, sender-side only mechanism that avoids the extra cost of additional packets is therefore desired.

III. Redundant Data Bundling

RDB is a mechanism that aims to reduce application-layer latency by proactively retransmitting unacknowledged data. Interactive and time-dependent applications commonly send small chunks of data at regular intervals [2]. As seen in Table 1, this rarely leads to segments as large as the Maximum Segment Size (MSS). By using the "free" space available in packets smaller than an MSS, we can retransmit unacknowledged data with every packet that contains new data, thereby hiding possible loss in a proactive manner. Not only does this approach reduce the time until a lost packet is recovered, it eliminates HOLB as well, because the potentially blocked packet itself carries the remedy. So although RDB packets are larger than those of unmodified senders, RDB does not generate any additional packets.

A TCP segment contains one contiguous byte range. The feature that allows the mechanism to work is the fact that every correctly implemented receiver will consume a packet containing new data, even though some of the data had already been received. This is the reason why RDB can be a sender-side only mechanism as any mechanism that allowed redundant sending of arbitrary data segments would require changes to both senders and receivers. Figure 1 shows a short packet sequence timeline, where an RDB-enabled thin stream avoids retransmissions due to redundant data bundling.

A. *The original RDB version*

The RDB implementation presented in [3], which we refer to as *old RDB*, showed great potential on improving the latency for thin streams, but it left certain issues unresolved. When using RDB and a segment is lost, the next packet will contain the lost segment. This leads to data being received in order, meaning that there is no gap in the sequence numbers of the arriving data. The sender will not receive the dupACK that would have normally been produced from this gap, effectively hiding the loss event from the sender. This prevents the congestion control (CC) from adjusting the allowed send rate for the flow accordingly.

The inability to detect loss events properly made the *old RDB* implementation unfair to competing traffic. In addition, *old RDB* imposed no limit on the level of redundancy, which allowed misuse by carefully shaped streams although it was only meant to be used for thin streams. In the next section, we describe how we have extended on *old RDB* to address these issues.

B. *The improved RDB version*

The hiding of loss events prevents the CC from adjusting the stream's send rate in response to loss. We have investigated how to ensure that RDB-enabled streams behave in a TCP-friendly manner towards competing traffic, while still being effective in reducing the application-layer latency for thin streams. It is of utmost importance that RDB-enabled streams react properly to loss caused by congestion.

In this section we present how we have addressed these issues in the new RDB implementation. Firstly, we explain how to detect the hidden loss in the RDB streams. After that, we go into details about the technique we have implemented to limit the use of RDB to only the streams that are *thin*.

1) Loss detection: When a TCP stream experiences packet loss, there are gaps in the received window of segments, resulting in dupACKs when new data is received. A regular TCP sender uses this dupACK information to infer packet loss.

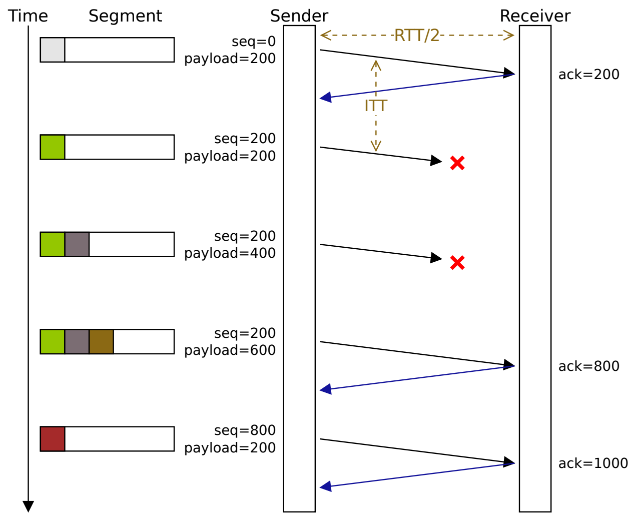


Fig. 1: Packet timeline of a TCP thin stream using RDB

With RDB, however, the lost data is included with the next packet, which does not result in a dupACK when a single packet is lost. The number of previous segments bundled with each packet determines the number of consecutive packets that have to be lost for a gap to occur. For example, if each RDB packet includes bundled data of four previously sent packets, the loss of five consecutive packets is required for a gap in the data to appear on the receiver side, resulting in a dupACK to be produced.

A standard TCP sender must consider that a receiver uses delayed ACKs, which allows the receiver to delay sending ACKs on incoming data. However, delayed ACKs must be switched off when the sequence number of the incoming TCP packet differs from the sequence number that is *expected* to come next. When receiving an RDB packet containing both old and new data, the sequence number of the first byte does not match the expected sequence number, which means that delayed ACKs are disabled as soon as an RDB packet is received.

Consequently, an RDB sender can expect to receive one ACK per RDB packet sent, and every incoming ACK that acknowledges multiple packets is a loss indicator. Thus, we have implemented this method for detecting packet loss in the new RDB implementation.

This loss detection method produces some false positives, such as when reordering has occurred. When loss has been detected by the detection method, TCP enters CWND reduction state. These false positives cause the CC to be more restrictive towards RDB-enabled streams, which can only affect competing traffic positively.

Duplicate selective acknowledgement (DSACK) enables the receiver to inform the sender of duplicate received data segments. When DSACK is enabled on an RDB-enabled stream, the redundant data bundled with newer packets causes the receiver to populate a TCP option field with the range of already received data. It is possible to use this information to accurately identify packet loss, because a packet loss is represented by the lack of a DSACK range in the incoming ACK.

2) PIF-based thin-stream classification: Applications that generate thin streams are very diverse, and so is their traffic, as exemplified by Table 1. Identifying streams that are thin is not a simple task. In a study [12], several indicators were established to identify and classify thin streams: PIFs, ITT, payload size, and the stream duration.

Traditionally, streams that do not send enough data to trigger fast retransmit have been considered thin. Mechanisms like ER [5] and RTOR [7] are only enabled when the number of PIFs is less than four, as such streams will not produce the three dupACKs required to trigger fast retransmit. We call this hard-coded limit Static PIF limit (SPIFL).

Such an SPIFL can hardly ever be exceeded in the low-latency environment of a data center, while even large limits are easily exceeded on high-latency satellite links.

By establishing the minimum ITT as a boundary for distinguishing streams that are thin from those that are not, it is easier to distinguish the applications that deserve to benefit from thin-stream mechanisms from those that do not. Therefore, we

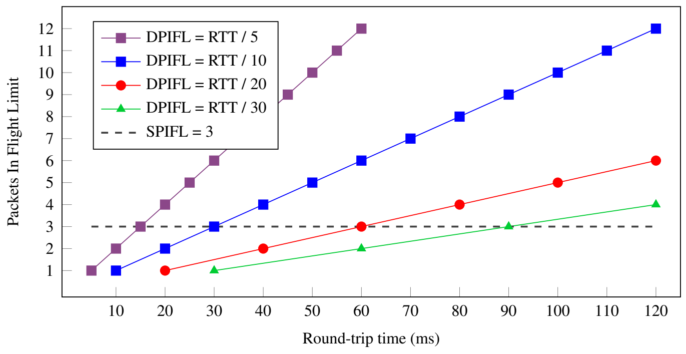


Fig. 2: Example of DPIFL for different minimum ITT values

propose a method that utilises a Dynamic PIF limit (DPIFL), which is dynamically calculated based on a the application's sending rate and the network RTT.

An RDB implementation must first select a fixed lower ITT limit ITT_{min}. Then, the DPIFL is calculated dynamically as a function of the flow's RTT, which ensures that streams are treated similarly in differing network environments. The DPIFL-based thin-stream detection method calculates the DPIFL as $\frac{RTT_{min}}{ITT_{min}}$, where RTT_{min} is the minimum RTT value observed over the lifetime of the connection, and ITT_{min} is a static limit defining the minimum allowed ITT that still permits a stream to use RDB. In other words, ITT_{min} is the limit for when a stream is identified as thin.

Figure 2 illustrates the DPIFL calculated using a few selected ITT_{min} values. How to choose the minimum ITT value is still an open question that would benefit from extensive evaluations over a large number of Internet hosts with a wide range of different RTTs. In our experiments, we have chosen to evaluate DPIFL$_{10}$ and DPIFL$_{20}$. The choice of 10 ms as our lowest evaluated ITT_{min} value was derived from the justifications in RFC 4828 [13], where such a limit is imposed to distinguish semi-greedy flows from small-packet flows. An important consideration is to allow the RDB mechanism to be active for higher RTT connections where having to retransmit would hurt the most. Setting a hard limit of less than four packets in flight would effectively prevent that. We also want to make sure that we don't use a too high ITT_{min} as that would turn the mechanism off for many target interactive applications as we can see from table 1 in section I.

IV. Experimental Evaluation

A. Evaluation metrics

We define the following metrics to measure the effect of RDB: *ACK latency* and *TCP friendliness*. The *ACK latency* quantifies the reduction in latency from the viewpoint of the sender. The *TCP friendliness* quantifies the penalty incurred on competing traffic by the RDB mechanism.

1) ACK latency: As the main goal of RDB is to improve the latency for thin streams, the main metric we use for our evaluations is latency. The end goal is to reduce the *application-layer latency*: the time interval between the pushing of a segment of data to the network stack by the application on the sender-side, until the application in the receiving end is able to read the data. However, as we currently do not have a good way of measuring the application-layer latency directly, we measure the ACK latency. The ACK latency is the time interval between the sending of a TCP segment onto the network by the sender, until an ACK covering the segment is received by the sender. This way of measuring latency, however, hides the sojourn time caused by the sender buffering segments before it is able to send it out on the network, as we explain in detail in section V-B.

2) TCP friendliness: The TCP friendliness metric expresses how well the RDB modifications play with regular (non-RDB) TCP flows. Because RDB bundles previously sent segments that are unacknowledged, it sends more data per segment. Therefore we consider the effect this has on competing traffic.

The traditional definition of a TCP friendly network stream is one that does not exceed its fair share of the bandwidth when competing with other flows for the resources of a common bottleneck. This paradigm, however, does not consider the case where bandwidth is not the limiting factor for a stream, as is the case for thin streams [14]. We have included the impact that RDB has on the latency of competing thin streams in our evaluations, in addition to the impact on the throughput of competing traffic, both thin streams and cross traffic.

B. Testbed setup

All the experiments were performed in a lab testbed using Linux Debian 7 machines. The testbed was set up with three senders and one receiver, as well as a dedicated router and a network emulator as depicted in Figure 3. The outgoing link of the router has reduced capacity by using a software rate control, in order to make it a shared bottleneck between the senders.

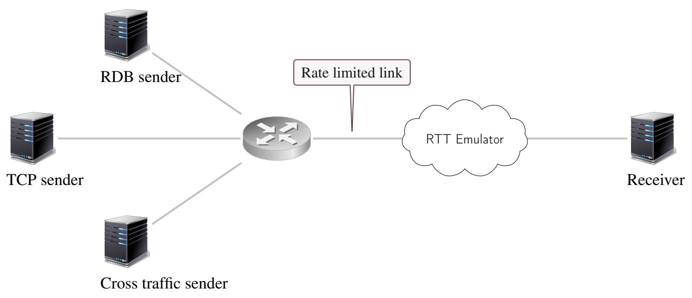


Fig. 3: Testbed network topology

One of the sender hosts is used to generate greedy cross traffic to create congestion over the bottleneck. The two thin-stream senders are running Linux kernel version 3.16, where one is using our RDB implementation, and the other is using Linux TCP NewReno with recommended low-latency options[1] for comparison. TCP NewReno is used instead of the default TCP CUBIC, as NewReno has been shown to be the best choice of CC for thin streams [1]. The two thin-stream senders are also modified to use functionality from before version 3.16 of the Linux kernel to determine when the CWND should be increased. Recent changes to the Linux kernel [15] have made matters worse for thin streams by making application-limited

[1]Nagle's algorithm is disabled, TLP and ER are enabled on the senders.

streams unable to increase their CWND after loss recovery, and consequently remain in a constant congestion-limited state.

Rate control is implemented using `tc-htb` on the router, and the RTT emulation is accomplished by configuring `netem` on a dedicated host. We also use this `netem` instance to cause random loss events in our first experiment. We have followed a proposed set of guidelines for designing a testbed [16], including disabling any offloading mechanisms on the network interfaces.

C. Experiments

We have performed three experiments with different configurations, where we measure the reduction in latency by calculating the ACK latency, and evaluating the TCP-friendliness by comparing the achieved goodput of other streams when competing with RDB.

1) Latency tests with uniform loss: We designed a set of tests in order to evaluate the effect RDB has on the latency of each data segment, and especially how it alleviates the HOLB issue.

2) Redundancy level tests: The second experiment is designed to test the effects of RDB in a more realistic scenario. We use two thin-stream senders in order to compare RDB-enabled thin streams to thin streams without RDB. We use the third sender to generate greedy cross traffic through the same network path, in order to cause congestion on the shared bottlenecked link.

This experiment evaluates how the number of outstanding segments RDB is allowed to bundle affects the performance of both RDB-enabled streams and competing streams. It is a form of fairness test to show that a limitation on the level of redundancy is needed to avoid excessive bundling of redundant data.

3) TCP fairness tests: The final experiment is designed to test how *TCP friendly* RDB is. This test is a form of fairness test where we attempt to misuse the RDB mechanism in order to get an unfair share of the bandwidth. This is done by using a transmission pattern that tries to increase the throughput by bundling redundant data on streams that are not thin.

V. Evaluation Results

An extensive collection of tests were performed in our lab testbed. Table 2 shows the range of parameters we have used for these tests. Because of space limitations, we only present a selection of the entire set of results. In this paper, we present the tests with an ITT of 30 ms, as this ITT corresponds with Skype, as shown in Table 1, as well as many other VoIP applications [2].

The payload size is 120 bytes for all our tests. We argue that the choice of packet size is insignificant as entire *packets* are lost and as long as the payload size is small enough to bundle previous segments.

The link speed of the bottlenecked link was configured to 5 Mb/s and the network emulator was configured to create an RTT of 150 ms. A FIFO queuing mechanism was used on the router with a size limit corresponding to one bandwidth–delay

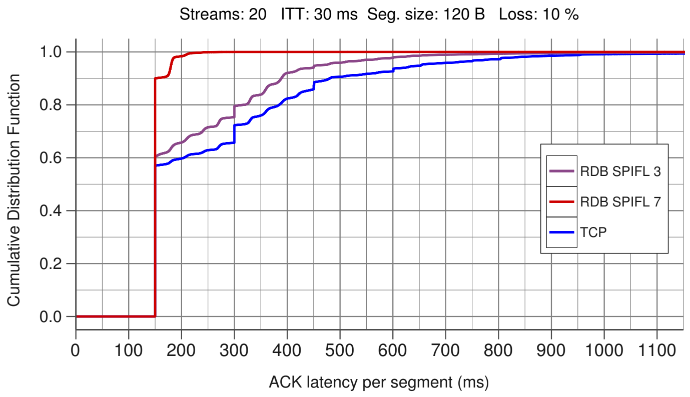


Fig. 4: Excerpt from the latency tests with uniform loss. RDB-enabled thin streams with different SPIFLs compared to thin streams using unmodified TCP.

product, 5×10^6 bits $\times 150 \times 10^{-3}$ seconds $= 93,750$ bytes $\approx$ 63 MTU-sized packets. The duration of all the tests was set to 5 minutes.

The measurements were taken by running a packet sniffer on all the sender and receiver hosts. We also calculated the exact packet loss by comparing the number of TCP segments captured in the sender-side filter with the number of segments captured in the receiver-side filter. Throughput was calculated by counting the number of bytes seen on the receiver-side packet trace for each one-second interval. Goodput was calculated in the same way, except headers were stripped and only new data was counted.

A. Latency tests with uniform loss

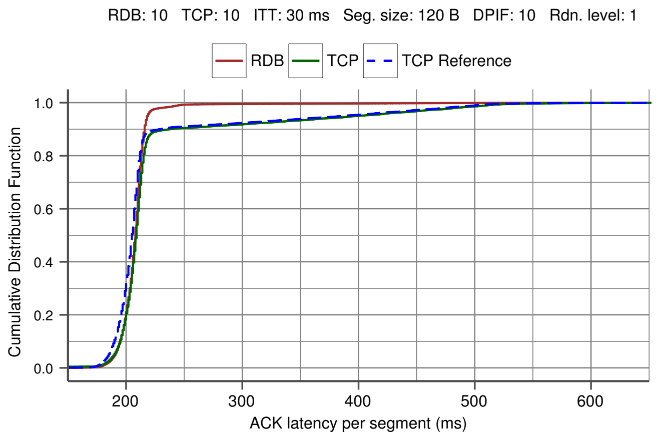
In the first experiment, a single user was configured to generate 20 thin streams with different ITTs and segment sizes. In order to avoid the unpredictability of loss caused by network congestion, we configured the network emulator `netem` to induce different uniformly distributed loss rates. The RDB mechanism was tested with different PIF limits, both SPIFL and DPIFL. The performance of RDB was further compared with the regular TCP thin streams.

The ACK latency results presented in Figure 4 show the cumulative distribution function (CDF) of ACK-latencies of two selected tests from this experiment. The test scenario is as follows: 20 thin streams producing a segment of 120 bytes every 30 ms. In order to avoid *ITT synchronization* [17], a randomly chosen variable from a normal distribution was added to the ITT for each individual stream. In the two tests, we used RDB with SPIFL$_3$ and SPIFL$_7$, respectively. We also included a regular TCP thin stream as a baseline reference. In this selected scenario, the loss rate was configured to 10% with uniform distribution. This high loss rate was chosen to emphasise the latency benefits of RDB, which become more apparent with higher loss rates. Periods of such high loss may frequently appear, for example in wireless networks experiencing noise or media contention.

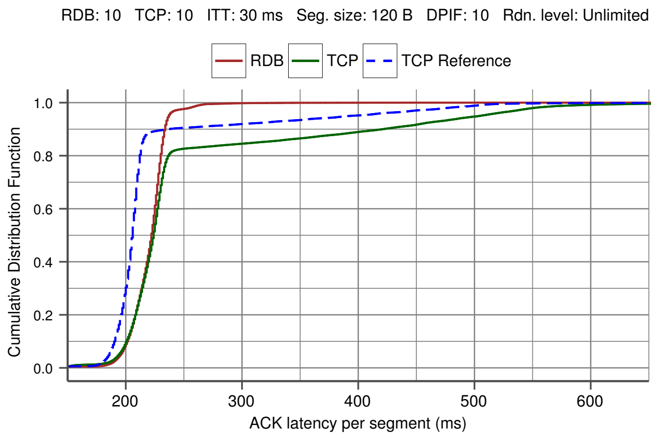
Figure 4 illustrates how TCP thin streams suffer from HOLB, where lost segments impact the following segments causing the "staircase" pattern seen in the CDF. Each step aligns with an ITT interval because the n-th segment following a lost segment

	Latency tests	Redundancy level tests	TCP fairness tests
Loss rate (%)	0.5, 2, 5, 10	-	-
Cross traffic streams (#)	-	5	3, 5, 7, 10, 13
TCP Reference streams (#)	20	5, 10, 16	3, 5, 7, 10, 13
RDB streams (#)	20	5, 10, 16	3, 5, 7, 10, 13
ITT (ms)	30, 50, 75, 100	10, 30, 50, 75	5, 15, 30
Payload (B)	120	120	400
PIF limit	SPIFL_3, SPIFL_7	DPIFL_{10}, DPIFL_{20}	DPIFL_{10}
RDB_{lim}	∞	1, ∞	∞

Table 2: Parameters for the testbed experiments. For the latency tests a uniformly random loss rate where configured, whereas for the redundancy level tests and fairness tests loss were caused by competing greedy cross traffic streams.



(a) RDB with a bundling limit of one previous segment ($\text{RDB}_{\text{lim}=1}$)



(b) RDB with no bundling limit ($\text{RDB}_{\text{lim}=\infty}$)

Fig. 5: Excerpt from the redundancy level tests. RDB-enabled thin streams with different levels of redundancy. Note the baseline TCP reference. The network RTT is configured to 150 ms.

will have an ACK latency that is *at least* $\text{RTT} + n \times \text{ITT}$. As many thin streams suffer from the inability to retransmit before an RTO occurs, the duration of the RTO-timer also becomes significant and adds to the delay [2]. If a retransmitted segment is lost, the increase in delay is even more significant due to exponential back-off [2].

In the scenario depicted in Figure 4, the thin streams, having an ITT of 30 ms ($1/5^{th}$ of the RTT), can have as many as five PIFs at any given time. We see that at around 300 ms, twice the RTT, there is a significant "step" in the ACK latency CDF for the baseline TCP thin streams, caused by a fast retransmit being triggered and the ITT aligning with the RTT. While the same "step" occurs for RDB SPIFL_3, it is less prominent. This is because RDB SPIFL_3 has reduced the number of required retransmissions by having already bundled some of the segments that are lost, and thus less segments are affected by HOLB.

When the PIF limit is higher than the actual number of PIFs, RDB is more efficient because it is able to bundle more often. RDB with SPIFL_3, is only allowed to bundle when it has less than three packets in the pipe, which in this scenario only occurs after loss recovery. With SPIFL_7, RDB is allowed to bundle whenever it experiences loss. When it is able to bundle, RDB reduces the effect of HOLB drastically. Where the aggregated 97^{th} percentile for the TCP reference streams is 775 ms, it is 551 ms for RDB streams with SPIFL_3 and only 183 ms for the RDB streams with SPIFL_7.

B. *Redundancy level tests*

In this experiment, the performance of RDB-enabled thin streams was compared to the performance of baseline TCP thin streams. Each thin-stream sender generated a certain number of streams with a fixed ITT and segment size. One sender used RDB and the other unmodified TCP. For all our tests, we also conducted a reference test using *only* unmodified TCP thin streams, labeled *TCP Reference*. To ensure a certain amount of packet loss and contention, we ran five greedy TCP cross-traffic streams over the bottleneck link to cause congestion.

Figure 5 depicts two of the tests from this experiment, showing 10 RDB-enabled thin streams competing with 10 regular TCP thin streams. The reference test is also included in the figure, in order to illustrate how the competing baseline TCP thin streams are affected when RDB bundles more aggressively. Both types of thin streams produce segments of 120 bytes every 30 ms. The RDB streams used DPIFL_{10}. The difference between the two depicted tests is that 5a limited the number of previous segments to be bundled to one ($\text{RDB}_{\text{lim}=1}$), while 5b had no such bundling limitation ($\text{RDB}_{\text{lim}=\infty}$).

We see in Figure 5 and in Table 3, that the effects of $\text{RDB}_{\text{lim}=1}$ on competing traffic is small: The TCP thin streams' ACK latencies and loss rates are almost identical to the ACK latency values and loss rates of the TCP Reference streams, even though the RDB-enabled streams send twice the amount of data as the baseline TCP thin streams. This is because a bundling limitation of a single outstanding segment allows RDB to recover from random loss events, e.g. loss caused by a

Stream type	Segs sent	Loss ratio	Total data	Rdn data	Rdn ratio
TCP Reference	*96.1 k*	*1.25 %*	*12 MB*	*147 kB*	*1.21 %*
TCP vs. $\mathrm{RDB}_{\mathrm{lim}=1}$	96.2 k	1.28 %	12 MB	153 kB	1.24 %
TCP vs. $\mathrm{RDB}_{\mathrm{lim}=\infty}$	87.2 k	2.48 %	12 MB	317 kB	2.54 %
$\mathrm{RDB}_{\mathrm{lim}=1}$	91.8 k	1.34 %	24 MB	12 MB	50.0 %
$\mathrm{RDB}_{\mathrm{lim}=\infty}$	80.3 k	2.58 %	**75 MB**	**63 MB**	**83.9 %**

Table 3: Results depicted in Figure 5: Thin TCP streams competing against RDB with a bundling limit of one previous segment ($\mathrm{RDB}_{\mathrm{lim}=1}$) and RDB with no bundling limit ($\mathrm{RDB}_{\mathrm{lim}=\infty}$)

competing greedy stream actively probing for more bandwidth, while keeping the level of redundancy to a minimum.

When RDB bundles more aggressively, more redundant data is sent, contributing to congestion and considerably increased average delay. This effect is reflected in the increase in the queuing delay, which is illustrated by the less steep curve for both $\mathrm{RDB}_{\mathrm{lim}=\infty}$and TCP, compared to the TCP Reference in Figure 5b.

Further, higher packet loss rates for the TCP thin streams are visible in Table 3. The increase in packet loss affects the ACK latencies, as depicted in Figure 5b. The 90^{th} percentile of the aggregated ACK latencies for the TCP Reference thin streams is 230 ms, and 246 ms for the TCP thin streams competing against streams using $\mathrm{RDB}_{\mathrm{lim}=1}$ and 421 ms for the thin streams competing against streams using $\mathrm{RDB}_{\mathrm{lim}=\infty}$. In addition, CWND will decrease for the streams when experiencing higher loss, leading to additional sojourn times because more segments are held back at the sender before being sent. It is important to note that this extra delay increases the *application-layer latency*, but is *not* reflected in the ACK latency values shown in Figure 5b.

Since $\mathrm{RDB}_{\mathrm{lim}=\infty}$ uses DPIFL_{10}, it is allowed to bundle as long as there are less than 15 PIFs. However, more redundant data can cause more congestion, and this leads to even more packet loss. As long as there is enough available space in the IP packets, RDB with the loose bundling restrictions can do more harm than good. Table 3 illustrates this, where we see that the redundant data amounts to almost 84% of the total transmitted data by the streams using $\mathrm{RDB}_{\mathrm{lim}=\infty}$. $\mathrm{RDB}_{\mathrm{lim}=\infty}$ is being so redundant that it generates almost six times the amount of network traffic of that of the TCP thin streams without actually getting any more *useful* data through. Nonetheless, it is interesting to note that despite this increased congestion, $\mathrm{RDB}_{\mathrm{lim}=\infty}$ is able to achieve considerably lower latency values than the regular TCP thin streams, as we see in Figure 5b. The values below the 90^{th} percentile are much lower for RDB than for the TCP thin streams.

It is even more interesting that RDB in both tests completely alleviates the extreme latency values experienced by unmodified thin streams, as shown in the Figure 5 as the values above the 90^{th} percentile. These are the delays that severely impacts the latency experienced by the user [1] [2] [8].

C. *TCP fairness tests*

In the final set of experiments, we tested whether or not we are able to use RDB in such a way that we get an unfair advantage over competing traffic. Instead of generating thin streams, the senders produced application-limited streams having a considerably higher data rate than what we would consider a conventional thin stream to have. We call these not-so-thin streams *isochronous streams* (ISOCH), meaning that they are still application limited rather than CWND limited due to their transmission pattern. These ISOCH streams try to achieve higher throughput by sending as much data as possible while still leaving space in each packet for bundling redundant data.

To be able to directly compare the TCP fairness, we include the greedy TCP streams in our comparisons as well. All three senders start an equal number of competing streams, and we compare the achieved throughput and goodput for each type of stream.

Figure 6 shows the results of one of these tests. Five greedy TCP streams competed against five ISOCH TCP streams and five RDB-enabled ISOCH streams. Each ISOCH stream of both types produced segments of 400 bytes every 30 ms. As ISOCH streams are application-limited and therefore send less than their "fair" share, both types of ISOCH streams achieve their optimal goodput. With no bundling limitation and an attempted transmission of 400 bytes, the RDB-enabled ISOCH streams are able to bundle up to two redundant segments with each packet. We see this reflected in the increased throughput.

The impact of RDB-enabled streams on competing traffic is limited, even when there is no bundling limit, because RDB only allows bundling with packets that are already scheduled for transmission. In other words, RDB does not send any additional packets, only bigger packets. When an RDB-enabled stream is limited by the CWND, i.e. the stream exceeds its fair share, the segments are held back before transmission leading to fewer and maximum segment sized packets. This buffering before transmission effectively leads to a temporary disabling of the RDB mechanism, because there is no space left for bundling. Bundling will only resume when the stream is no longer limited by network congestion.

With our testbed setup, tests having ITT intervals lower than 30 ms show that RDB is unable to bundle. Attempting to send larger segments also limits how much RDB is able to bundle, because less space is available.

RDB is only effective as long as the stream is application limited. An inherent property of application-limited streams is that they do not use their fair share of the available bandwidth.

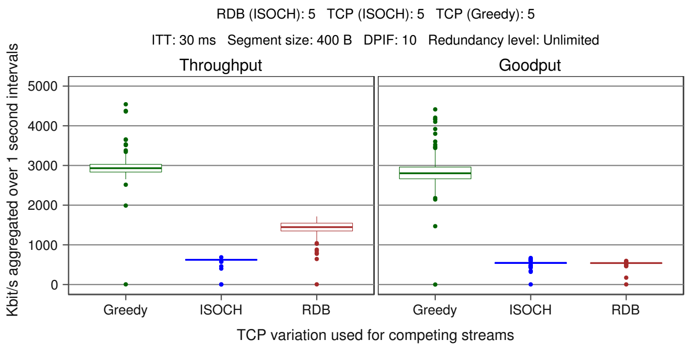


Fig. 6: Excerpt from the TCP fairness tests. Aggregated throughput and goodput for competing streams using different TCP variations.

From this, we argue that RDB can not be misused to gain an unfair advantage over competing traffic. Using RDB will cause more data being sent, which may contribute to higher congestion on low capacity networks or already saturated links. However, this is true for any transport protocol.

VI. Conclusion and Future Work

In this paper, we have presented the RDB mechanism to improve the service that TCP provides for interactive applications that generate only intermittent small data packets. By proactively retransmitting unacknowledged data segments, RDB is able to significantly reduce latencies caused by HOLB and retransmission delays, even in the presence of aggressive cross traffic. We have extended the original RDB idea [3] by implementing loss detection in order for RDB-enabled streams, with existing CC algorithms, to react properly to network congestion in the same way as unmodified TCP.

In addition to implementing loss detection, we discussed the possibility of including a definition of how limited, or how *thin*, a stream must be in order for RDB to be allowed to bundle. Defining what a thin stream is, however, is still an open issue [12]. In our experiments, we used different ITT-to-RTT ratios (DPIFLs) for defining when a stream was *thin enough* for RDB to be allowed to bundle. Other stream characteristics could be used, such as segment sizes or data rate, which would alter how a stream is classified as thin. We argue, however, that as long as the stream is application-limited, it does not contribute to congestion the same way greedy streams do because it does not exceed its fair share of the bandwidth. If the stream experiences loss and consequently becomes limited by the CWND, RDB is effectively disabled due to sender-side transmission buffering. Only after the CWND grows, and segments are no longer held back, will RDB be able to bundle again. Compared to other proactive latency-reducing mechanisms, such proactive double transmission [4] or TCP with FEC [10], [11], RDB transmits redundant data without generating additional packets.

Extensive evaluations was performed in order to determine the appropriate trade-off between the level of redundancy and the reduced latency experienced by the thin streams using RDB. We show that in congested scenarios, where the combined throughput of the RDB-enabled thin streams constitutes a significant proportion of the total throughput, imposing a limitation on the level of redundancy can be beneficial to avoid further congesting the network. Our findings show that even with a bundling limitation of one unacknowledged segment ($\text{RDB}_{\text{lim}=1}$), RDB still alleviates HOLB significantly. When bundling only one segment, RDB is able to recover from random loss events caused by competing traffic while keeping the redundancy to a minimum. This yields a reasonable trade-off between latency reduction and the redundancy.

A candidate for further research is to include the size of the segments produced by the application and the ITT of the application when considering the redundancy level. Adjusting the redundancy level according to the stream's experienced congestion is another such candidate. We argue that higher redundancy levels can be justified in scenarios with very thin streams or with extreme packet loss, not caused by congestion, as this may still be considered TCP fair.

Server administrators that expect a large amount of thin stream traffic, such as online game providers or web server administrators, must consider the impact the redundant level has on their outgoing gateways. Based on our findings, we recommend using a bundling limitation of one segment as a conservative default setting. For a home user with a small number of RDB-enabled thin streams, RDB will have a negligible effect on congestion.

We argue that our improved RDB mechanism can be considered TCP-friendly and safe to deploy. Being a sender-side-only modification to TCP, RDB can be incrementally deployed. This makes it a suitable candidate for widespread deployment.

Acknowledgements

This work is funded by the European Community under its Seventh Framework Programme through the Reducing Internet Transport Latency (RITE) project (ICT-317700) and the TimeIn project (213265-O70) by the Research Council of Norway.

References

[1] C. Griwodz and P. Halvorsen, "The fun of using TCP for an MMORPG," *International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, pp. 1–7, May 2006.

[2] A. Petlund, "Improving latency for interactive, thin-stream applications over reliable transport," Ph.D. dissertation, University of Oslo, Dec. 2009.

[3] K. Evensen, A. Petlund, C. Griwodz, and P. Halvorsen, "Redundant bundling in TCP to reduce perceived latency for time-dependent thin streams," *Comm. Letters, IEEE*, vol. 12, no. 4, pp. 324–326, April 2008.

[4] T. Flach, N. Dukkipati, A. Terzis, B. Raghavan, N. Cardwell, Y. Cheng, A. Jain, S. Hao, E. Katz-Bassett, and R. Govindan, "Reducing web latency: The virtue of gentle aggression," *SIGCOMM Comput. Comm. Rev.*, vol. 43, no. 4, pp. 159–170, Aug. 2013.

[5] M. Allman, K. Avrachenkov, U. Ayesta, J. Blanton, and P. Hurtig, "Early Retransmit for TCP and Stream Control Transmission Protocol (SCTP)," RFC 5827 (Experimental), IETF, May 2010.

[6] N. Dukkipati, N. Cardwell, Y. Cheng, and M. Mathis. (2012, Jul) TCP loss probe (TLP): An algorithm for fast recovery of tail losses. [Online]. Available: http://tools.ietf.org/html/draft-dukkipati-tcpm-tcp-loss-probe-01

[7] M. Rajiullah, P. Hurtig, A. Brunstrom, A. Petlund, and M. Welzl, "An evaluation of tail loss recovery mechanisms for TCP," *SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 1, pp. 5–11, Jan. 2015.

[8] A. Petlund, K. Evensen, C. Griwodz, and P. Halvorsen, "TCP mechanisms for improving the user experience for time-dependent thin-stream applications," in *Proceedings of the IEEE Conference on Local Computer Networks (LCN), Montreal, Canada*, Oct. 2008, pp. 176–183.

[9] P. Hurtig, A. Brunstrom, A. Petlund, and M. Welzl. (2014, Oct) TCP and SCTP RTO restart. [Online]. Available: https://tools.ietf.org/html/draft-ietf-tcpm-rtorestart-04

[10] H. Lundqvist and G. Karlsson, "TCP with end-to-end FEC," in *Communications, 2004 International Zurich Seminar on*, 2004, pp. 152–155.

[11] L. Baldantoni, H. Lundqvist, and G. Karlsson, "Adaptive end-to-end FEC for improving TCP performance over wireless links," in *Communications, 2004 IEEE International Conference on*, vol. 7, 2004, pp. 4023–4027.

[12] M. S. Fuchs, "Time-dependent thin transport layer streams: Characterization, empirical observation and protocol support," Master's thesis, University of Kaiserslautern, january 2014.

[13] S. Floyd and E. Kohler, "TCP Friendly Rate Control (TFRC): The Small-Packet (SP) Variant," RFC 4828 (Experimental), IETF, Apr. 2007.

[14] B. Briscoe, "Flow rate fairness: Dismantling a religion," *SIGCOMM Comput. Commun. Rev.*, vol. 37, no. 2, pp. 63–74, Mar. 2007.

[15] Git commits, "Changes to CWND growth," Available: http://git.kernel.org/linus/e114a710aa and http://git.kernel.org/linus/ca8a226343.

[16] Bufferbloat-project, "Best practices for benchmarking CoDel and FQ CoDel," May 2014, https://www.bufferbloat.net/projects/codel/wiki/Best_practices_for_benchmarking_Codel_and_FQ_Codel.

[17] B. R. Opstad, "Taming redundant data bundling," Master's thesis, University of Oslo, may 2015.