Consistency Fences for Partial Order Delivery to Reduce Latency

Nooshin Eghbal and Paul Lu

University of Alberta, Computing Science Department, Edmonton, Canada
{eghbal, paullu}@ualberta.ca

Abstract. For appropriate workloads, partially ordered message delivery can greatly reduce message latency. For example, updates to screens (e.g., remote desktops, VNC) may not have to be totally ordered with respect to different regions of the screen, but ordered with respect to updates to the same region. Similarly, updates to disjoint regions of a file (e.g., bulk-data transfer of sensor data) can be applied in any order, as long as updates (or reads) to the same region of the file are ordered in a consistent way, per data consistency models.
Therefore, we introduce the concept of a *consistency fence* (CF), inspired by a memory fence from data consistency models, as a mechanism to control, specify, and reason about partial orders. If messages are lost on a network, partial ordering via CFs provides a framework to tolerate the latency associated with retransmission, for key workloads.
In a set of simple experiments, based on screen update workloads, we show the latency benefits of partial ordering with CFs. We also show how forward error-correction (FEC) can be combined with CFs and partial ordering to reduce cumulative latency (represented as a cumulative distribution function), as compared to total ordering of messages.

Keywords: Partial order delivery · Latency · TCP · Quality of service · Real-time.

1 Introduction

For appropriate workloads (e.g., remote desktops, bulk-data transfers), partially ordered message delivery can greatly reduce message latency. The total ordering of message delivery is more common because total orders are simple and easier to reason about. However, if there is flexibility in the ordering, then retransmissions of lost packets can be overlapped with regular transmissions. For screen updates, different regions can be updated in any order, as long as updates to the same regions have some consistent ordering. For bulk-data transfers, the data for different blocks of the same file can arrive in any order, as long as all updates have arrived by the end of the transfer. The challenge is designing a mechanism and semantics for specifying when ordering matters and when it does not matter.

Therefore, we introduce the concept of a *consistency fence* (CF) as a mechanism to specify when ordering between packets and messages matter. Inspired

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08751-6_35

by memory fences from data consistency models [11], CFs are like meta-data messages inserted into the stream(s) of messages. Messages emitted *between* a CF can be reordered among themselves, but messages can never be reordered *across* a CF. Furthermore, CFs must be totally ordered between themselves. As a mechanism, a CF can be implemented using well-known sequencing techniques. The semantics of CFs have the beneficial property that inserting extra CFs into the stream can never cause erroneous interleaving, although with some potential loss of performance due to a reduced flexibility of interleaving.

A general way to represent ordering constraints is as a directed acyclic graph (DAG) (Figure 1). For example, blocks B_1, B_2, and B_3 have no inter-block dependencies and can be delivered to the destination in any order (among those three blocks), but B_9 must be delivered before B_7. Directed edges show the ordering constraints.

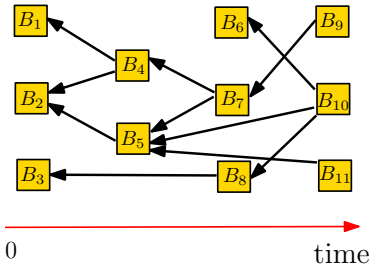


Fig. 1: Dependency graph

Figure 2 compares total order delivery (such as with TCP) vs. partial order delivery for Figure 1, if block B_6 is lost. With total order delivery we cannot deliver B_7-B_9 while we are waiting for the retransmission of B_6 (the green block). This source of latency is known as the Head-Of-Line (HOL) blocking problem. However, with partial order delivery we can first deliver B_7-B_9 after B_5, with lower latency for those blocks, and overlap those deliveries with the retransmission of B_6, resulting in lower total latency for delivering B_1-B_{11}.

Workloads that have periodic updates are candidates for partial ordering mechanisms like CFs. In remote desktop applications or online video games, there are periodic screen updates. In each update, different parts of the screen need to be updated and we (usually) need to finish each screen update before starting the next one. Using partial order delivery, we can apply all messages in each update as soon as we receive them in the destination in any order and do not block and wait for the lost ones. There is no need for the total ordering provided by TCP/IP. For bulk-data transfers, which are not discussed any further, the file is not always consumed until the transfer is complete, therefore different blocks of data can arrive in any order, as long as all data arrives before the file is closed.

In Section 2, we summarize some of the related papers, and then we describe our CF mechanism in Section 3. Then, we explain why adding a forward error

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08751-6_35

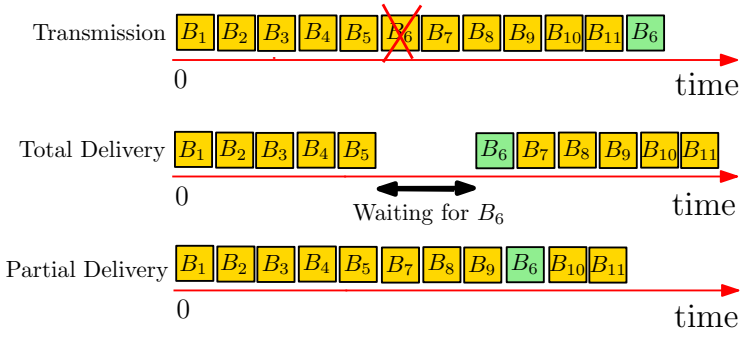


Fig. 2: Total vs. partial order delivery

correction (FEC) technique to CFs can help even more reduction in the message blocking time and describe 2D XOR FEC that we use for the results in this paper in Section 4. In Section 5 we present our emulated results to compare partial ordering through CFs and total ordering over UDT under different packet loss rates and Round Trip Times (RTTs) with and without FEC. Section 6 has a summary and some future extensions of the paper.

2 Related Work

The previous work on supporting partial order delivery used some notion of a dependency graph which needs to be sent to the receiver [2][13]. Therefore, the receiver would know in which order the data blocks need to be delivered to the application layer. Using dependency graphs, we can express the ordering precisely but implementing a networking system based on these graphs to support partial ordering is complicated and needs the dependency knowledge in advance.

Also, there are some projects that release the total ordering manner of TCP to solve the HOL blocking problem such as Stream Control Transmission Protocol (SCTP) [15] and Quick UDP Internet Connections (QUIC) [9]. They use multi-streaming over a single connection such that only the data blocks within each stream need to be delivered in total order but there is no ordering between the data blocks of different streams. However, there is no strategy for supporting partial ordering over SCTP or QUIC. For example if we want to send the eleven data blocks of Figure 1 over them, we must use only one stream for the whole graph because we cannot count on the ordering of data of different streams.

UDP-based Data Transfer (UDT) protocol [6] is a user-level, reliable protocol designed for large data transfers over wide-area networks (WANs). UDT has a built-in loss-based Congestion Control Algorithm (CCA) which provides better throughput than TCP CUBIC under WAN settings. UDT has two modes: 1) stream mode and 2) message mode. The stream mode is like TCP and supports total ordering only but in the message mode we can set an order flag for each message to be delivered in total order or arrival order. However, there is no

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08751-6_35

mechanism to express any partial ordering setup inside UDT. As detailed in the next section, our prototype of CFs is layered on top of UDT, using message mode, where the flag of all messages is set to arrival order delivery.

Resource prioritization for HTTP workloads to reduce web page load time has a long history in the literature [10]. The idea is that scheduling web resources based on the file types plays an important role in minimizing the overall page load time. For example, some of the resource files such as CSS and JavaScript need to be downloaded completely before getting used while image files can be rendered incrementally. To communicate the web resource priorities between HTTP client and server, dependency graphs were used for years over different versions of HTTP. However, managing these graphs was complicated and challenging like the concept of dependency graphs in partial ordering setup. Recently a more practical approach was chosen by IETF for further development in HTTP/3, which is based on labeling resources via urgency vs. incremental by the HTTP client [12]. We also aim to introduce a more practical way of handling partial ordering requirements through our consistency fences mechanism.

3 Consistency Fences

We propose the *consistency fence* (CF) mechanism, in which the sender transmits the independent data blocks (or messages) in parallel and inserts a fence before sending any data block that has dependencies to the previously sent blocks. This way, the receiver would know to deliver all the blocks before each fence and then start delivering the blocks after the fence. The CF is analogous to the memory fence/barrier instructions used by CPUs to enforce an ordering constraint on memory operations issued before and after the fence instruction [11].

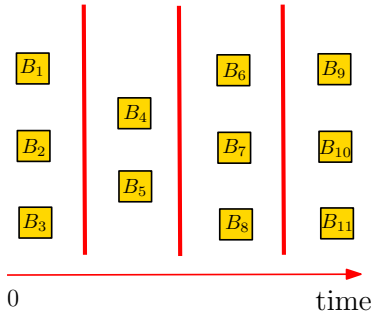


Fig. 3: Consistency fences

In Figure 3, we show how the dependency graph in Figure 1 can be sent using CFs (e.g., vertical red lines). The sender transmits the first three blocks (B_1 to B_3) and then inserts a fence because both B_4 and B_5 are dependent on at least one of the first three blocks, although with no inter-dependencies between B_4

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08751-6_35

and B_5 themselves. The same would happen for the third and fourth groups of blocks which are B_6-B_8 and B_9-B_{11}, respectively.

A dependency graph (Figure 1) can be more precise than consistency fences to express ordering among data blocks and inserting a fence may add more, implied ordering dependencies. For example, in Figure 3, both B_4 and B_5 seem to be dependent on B_1-B_3, while in the original graph in Figure 1, B_4 is dependent on B_1 and $B2$, and B_5 is only dependent on B_2. However, communicating the ordering requirements through dependency graphs is complicated and in many situations the sender does not have the complete dependency graph ahead of time. Therefore, we believe that having a more practical mechanism like consistency fences could help encourage application-layer programmers to take advantage of partial order delivery.

4 Combining Partial Order Delivery with Forward Error Correction

Forward Error Correction (FEC) has been discussed in the literature as an effective method to speed up packet loss recovery [1][8]. Using FEC, the sender transmits some redundant packets along with the original packets so the receiver will be able to recover lost packets sooner than the time it needs to wait for receiving the retransmissions especially in the networks with high RTTs like WANs. In our previous work, we showed the benefit of using a specific FEC method called two dimensional XOR-based (2D XOR) FEC on reducing the latency and improving the throughput of data transfers over WANs compared to the state-of-the-art WAN protocols in total order delivery setup [3]. However, we believe that FEC can help reduce latency in partial order delivery in the scenario that we have received some data but we cannot deliver it to the destination because it depends on some lost data and we are waiting for its retransmission (e.g. we have received B_4 and B_5 but B_2 is lost). In this scenario, losing the retransmission of the lost data can make the blocking time even worse.

In this section, we briefly explain the 2D XOR FEC and we evaluate the benefit of combining it with consistency fences in the next section. More details on the the 2D XOR FEC implementation and results can be found elsewhere [3].

In the 2D XOR FEC method, we build 2D matrices of the original packets and send the XOR of all rows and all columns as redundant packets to the destination. In our implementation of 2D XOR over UDT, the user sets the number of rows and columns. For example, if the number of rows is set to 20 and the number of columns is set to 40, we protect each group of 800 packets by building a matrix of packets with 20 rows and 40 columns and send 20 row XORs and 40 column XORs. Therefore, the bandwidth overhead of the mentioned settings would be $\frac{20+40}{800}$ or $\frac{3}{40}$.

The interesting thing about 2D XOR FEC is that we can recover bursty packet loss patterns with the help of column recovery, whereas in 1D XOR FEC we can recover only one packet loss in each row [4][5].

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08751-6_35

5 Evaluation

We have implemented CFs as a separate mode inside the UDT version 4 system. The user can enter this partial ordering mode by inserting fences using an API call, `UDT::sendfence(client_socket)`, while sending messages. There are two kinds of packets in UDT: 1) data packets, and 2) control packets. UDT allows us to define user-defined control packets. In our implementation, each fence is a control packet which contains three numbers: 1) sequence number, 2) the number of messages from the previous fence to this fence (or the start of the transmission if this is the first fence), and 3) the sequence number of the last message before this fence. Therefore, at the destination we can check if we can deliver each message or we should wait to receive all the messages before the fence to be able to deliver the messages after the fence.

In addition to the default loss-based CCA, UDT has a non-loss-based CCA called UDPBlast which simply gets a fixed sending rate from the user and will not change it in the case of packet loss. We used UDPBlast for all our experiments to be able to better focus and analyse the benefit of supporting partial ordering. Additional experiments with a loss-based CCA remains as a future work.

We evaluate the latency of our CFs implementation over UDT compared to the total ordering setup over UDT in an emulated testbed. We use the Netem-tc Linux tool [7] to set the loss rate and RTT between two nodes with Linux kernel 4.13.0, 2.30 GHz Intel(R) Core(TM) i3-6100U CPU, 2-cores in total, 32 GB of memory, and 1 Gbps bandwidth.

As the workload, we sent 32000 messages with size 1400 bytes. We set the loss rate to $\{0.1\%, 0.5\%, 1\%\}$, the RTT to $\{20\ ms, 70\ ms\}$, and the number of messages in each update to $\{800, 1600\}$. We inserted a fence after each update in partial ordering mode. Although the workload used is not a full-fledged application, the overall goals of this initial evaluation show:

1. For an appropriate workload (e.g., screen updates) with substantial packet loss (e.g., 1% loss), **the partial ordering capabilities of consistency fences are more effective than FEC** in tolerating the cumulative latencies for each update message (e.g., Figure 4c)
2. As the RTT increases (e.g., to 70 ms, Figure 5), FEC can improve on the cumulative latencies for total orderings beyond just partial ordering. However, **partial orderings with consistency fences can also be combined with FEC** (with less FEC overheads) and still have the least cumulative latencies (e.g., Figure 5c, 6c)
3. When packet losses are lower (e.g., 0.1% or 0.5%), the cumulative latencies for partial orderings are consistently lower than for total orderings, without the overheads of FEC (e.g., Figure 4b vs. 4c). But, if desired, **FEC can be combined with partial ordering to further reduce cumulative latencies as a trade-off** for FEC overheads (e.g., Figure 4a vs. 4b).

Figure 4, 5, and 6 show the Cumulative distribution function (CDF) of message blocking time using total vs. partial order delivery with and without FEC

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08751-6_35

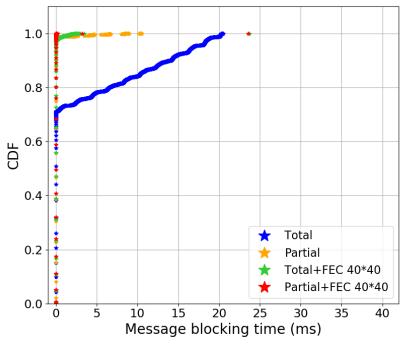
for three different packet loss rates. We also report (the number of recovered messages)/(the number of lost messages) for FEC settings in the captions. Lines which are closer to the top-left corner (x=0 ms, y = 1.0 CDF) show better latency performance (i.e., more messages have lower latency due to blocking). In general, the "Partial" ordering performance lines are better than the "Total" ordering performance lines

The blocking time is the whole time a message was waiting in the receiver buffer (i.e. the time from receiving it from the network to the time being delivered to the application layer). For example, if the RTT is 70 ms, the time from sending a message at the sender application to the time of delivering it to the receiver application (i.e. latency) would be 35 ms plus the blocking time. Therefore, in total order delivery, in the case of losing a message, the latency of the next messages will be increased because we need to block them in the receiver buffer until we receive the retransmission or be able to recover the lost one with FEC. The difference between theses three figures is the RTT and the number of messages in each update so we can study the impact of these two factors on the effectiveness of partial order delivery.

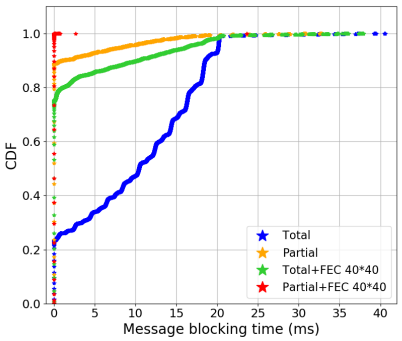
In Figure 4 we set the RTT to 20 ms and the number of independent messages in each update to 1600. In total ordering setup, all the messages after the lost ones need to wait in the receiver buffer until we receive the retransmissions, which will take about an RTT. Also, if we lose any retransmission, the blocking time will increase by another RTT. However, in partial order delivery, we can still deliver all the messages in each update without waiting for the retransmission of the lost ones. Then, to start delivering the next update's messages we need to wait for receiving the retransmissions.

As we increase the packet loss rate from 0.1% in Figure 4a to 0.5% in Figure 4b to 1% in Figure 4c, the blocking time of the total order setup (i.e. blue) gets worse because for each lost packet we need to wait an RTT to receive the retransmissions and during this time we block all the next messages in the buffer. However, adding FEC to total order setup (i.e. Total+FEC in green and violet) help reduce the blocking time but still cannot reach the performance of partial order delivery (i.e. orange) in Figure 4b and 4c where we have moderate and high packet loss rates. The reason is that using 2D FEC with 40 rows and 40 columns with total order delivery in Figure 4b (i.e. green), although we could recover 133 lost messages, we fail recovering 32 messages which result blocking the next messages for an RTT. It gets worse in Figure 4c when we have a higher loss rate so adding FEC 40*40 to total order delivery we can recover 217 lost messages and cannot recover 72 lost messages therefore getting higher blocking time. Also, increasing the number of redundant packets and bandwidth overhead by using FEC 20*20 with total ordering setup (i.e. violet) reduces the blocking time compared to total+FEC 40*40 but since we still worse than partial ordering setup even without FEC (i.e. orange). However, the combination of FEC 40*40 and partial ordering (i.e. red) results close to zero blocking times except for a few number of messages even in Figure 4c with 1% packet loss rate.

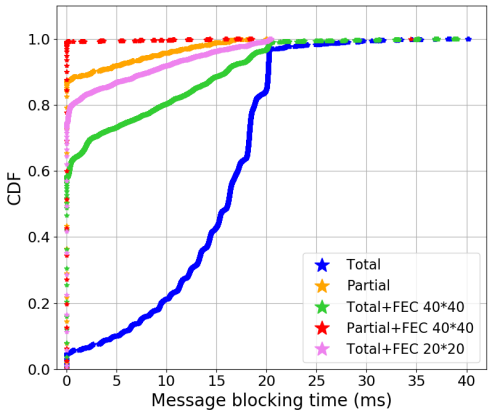
ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08751-6_35



(a) 0.1% loss. Total+FEC 40*40: 34/0. Partial+FEC 40*40: 30/0.



(b) 0.5% loss. Total+FEC 40*40: 133/32. Partial+FEC 40*40: 151/33.



(c) 1% loss. Total+FEC 40*40: 217/72. Partial+FEC 40*40: 237/74. Total+FEC 20*20: 299/22.

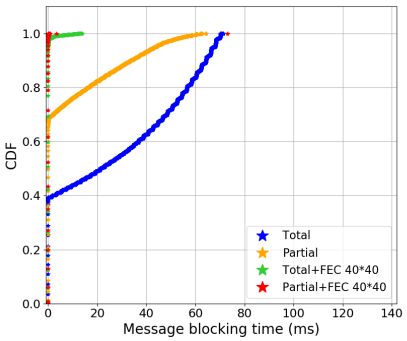
Fig. 4: The CDF of message blocking time (ms) for different amount of packet loss when RTT was **20**ms. The number of messages in each update was **1600**.

As a summary of the results presented in Figure 4, for all loss rates, partial ordering helps reducing the blocking time compared to total order delivery. Also, adding the same rate of redundant packets with FEC 40*40, partial order delivery can get closer to the loss-free scenario compared to total order delivery.

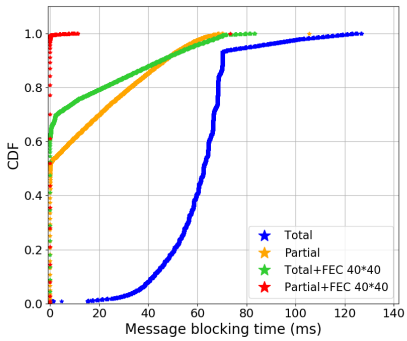
The reason behind choosing 40 for the number of rows and columns in 2D FEC was that we have 1600 messages in each update, so each 2D matrix can include all the messages in each update. Therefore, we can recover the lost mes-

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08751-6_35

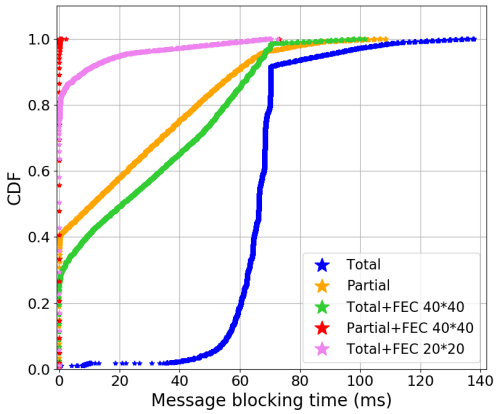
sages of each update without needing any messages of the next update, which is important for partial ordering setup.



(a) 0.1% loss. Total+FEC 40*40: 34/0. Partial+FEC 40*40: 29/0.



(b) 0.5% loss. Total+FEC 40*40: 146/14. Partial+FEC 40*40: 149/7.



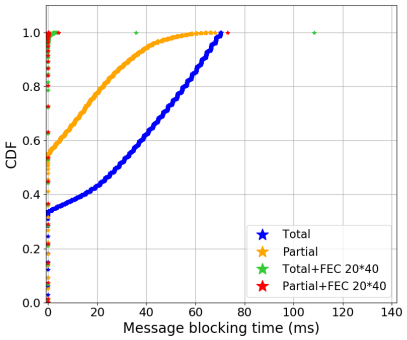
(c) 1% loss. Total+FEC 40*40: 275/45. Partial+FEC 40*40: 304/45. Total+FEC 20*20: 285/2.

Fig. 5: The CDF of message blocking time (ms) for different amount of packet loss when RTT was **70**ms. The number of messages in each update was **1600**.

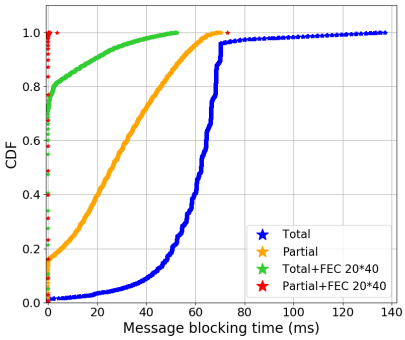
Figure 5 is very similar to Figure 4 except that we have a higher RTT of 70 ms to study the performance of our CFs to support partial ordering in the networks with higher RTTs like WANs. For all three packet loss rates, again partial ordering setup (i.e. orange) helps reduce the blocking time compared to total ordering setup (i.e. blue). However, having a higher RTT compared to Figure 4 the partial ordering results get worse and move to the right. The reason

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08751-6_35

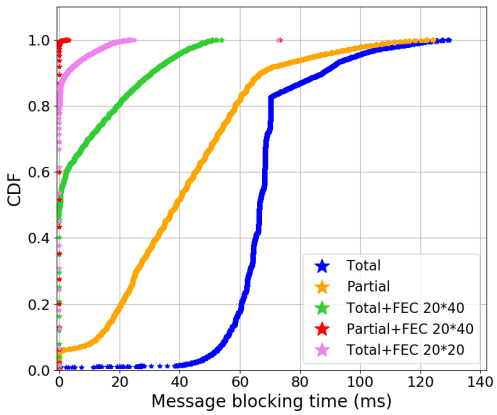
is that having a higher RTT there would be more number of lost messages for which we have not received a retransmission at the time of receiving the messages of the next update so we need to block them in the buffer in the meantime. In Figure 5b and 5c we can get close to partial ordering results when we add FEC 40*40 to total ordering (i.e. green). Also, adding more overhead and using FEC 20*20 with total ordering (violet) in Figure 5c we can reduce the blocking time more than partial ordering setup. However, using FEC 40*40 with partial ordering (i.e. red) is still the best for all loss rates.



(a) 0.1% loss. Total+FEC 20*40: 33/0. Partial+FEC 20*40: 72/0.



(b) 0.5% loss. Total+FEC 20*40: 158/0. Partial+FEC 20*40: 204/0.



(c) 1% loss. Total+FEC 20*40: 302/0. Partial+FEC 20*40: 304/0. Total+FEC 20*20: 302/0.

Fig. 6: The CDF of message blocking time (ms) for different amount of packet loss when RTT was **70**ms. The number of messages in each update was **800**.

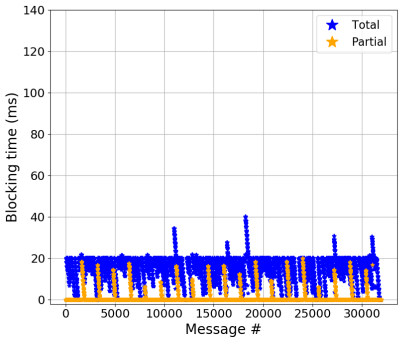
ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08751-6_35

Figure 6 is very similar to Figure 5 except that we reduce the number of messages in each update to 800. Having fewer number of messages in each update means having a workload with more ordering requirements. That is why the partial ordering results (i.e. orange) get closer to total ordering results (i.e. blue) but still better. We set 20 as the number of rows and 40 as the number of columns for FEC because 20*40 gets the number of messages in each update. Addig FEC 20*40 to both total and partial ordering setups is beneficial for all packet loss rates but still partial order delivery can get closer to loss-free scenario compared to total ordering even with 20*20 FEC in Figure 6c (i.e. violet). The reason in that although we can recover all lost messages using FEC 20*40 for all three packet loss rates, as we increase packet loss rate we need more column recovery because of having more than one lost message in each row. Therefore, we need to wait for receiving all the matrix (i.e. 800 messages) and also column XORs to start column recovery. In total order delivery, during this waiting time to receive the whole matrix, we need to block all the messages after the lost ones. That is why when we use FEC 20*20 in 6c for total ordering (i.e. violet), the blocking time is reduced compared to total+FEC 20*40 (i.e. green).

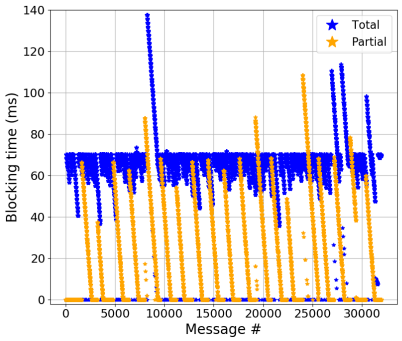
Figure 7 shows the message blocking time (ms) (not the CDF) of the three previous figures but only for 1% loss rate setting. In this figure we can better observe the impact of increasing RTT and decreasing the update size on the message blocking time of partial ordering setup. As we increase RTT from 20 ms in Figure 7a to 70 ms in Figure 7b the length of orange vertical lines increase, which means that more number of messages get blocked and for a longer time at the beginning of each update because we are still waiting for receiving the retransmission of the lost messages in the previous update. Also, as we decrease the number messages in each update from 1600 in Figure 7b to 800 in Figure 7c, the length of orange vertical lines get shorten because we expect to have about 8 lost messages in each update since the loss rate is 1% while it is 16 lost messages for 1600 messages in each update. Therefore, it would take less time to receive the retransmissions for all lost messages in each update but we have more frequent orange vertical lines since having 32000 messages in total and 800 messages in each update, we have 40 updates where we have 20 updates for 1600 messages in each update. As a summary, both increasing RTT and decreasing the update size would increase the number of blocked messages and the amount of blocking time in partial ordering setup. However, for workloads with periodic updates when there is a pause between updates at the sender side (e.g. remote desktop or online games), that would give the receiver more time after each update to receive the retransmissions and result in fewer blocked messages.

Figure 8 shows the message blocking time (ms) of the combination of either total or partial ordering setups with FEC when the packet loss is 1%. For all three settings, the message blocking time of partial+FEC 40*40 (i.e. red) is close to zero except for a few data points as we presented in the CDF graphs. However, for the same rate of redundant packets for total+FEC 40*40 in Figure 8a and Figure 8b, the vertical green lines show the lost messages that could not be recovered with FEC and we need to wait for the retransmissions. The

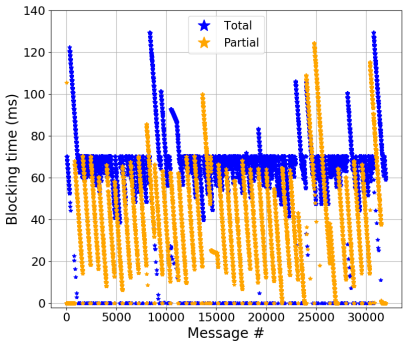
ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08751-6_35



(a) Update size: **1600** messages.
RTT: **20ms**.



(b) Update size: **1600** messages.
RTT: **70ms**.

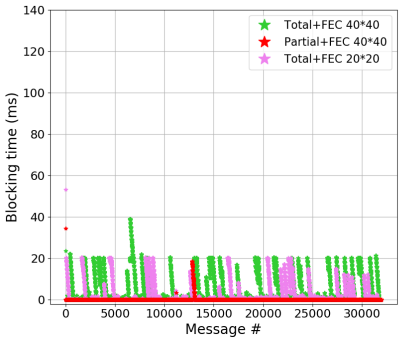


(c) Update size: **800** messages.
RTT: **70ms**.

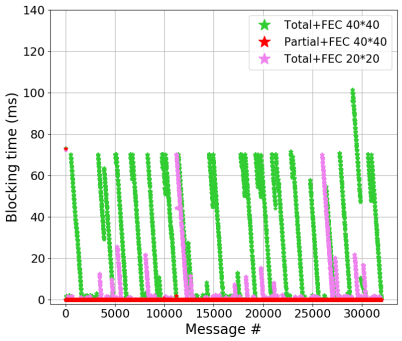
Fig. 7: The message blocking time (ms) for **1%** packet loss.

reason that not-recovered lost messages with FEC do not have the same impact on partial ordering is that with partial order delivery we do not wait for the lost messages in the **middle** of each update and having 1600 messages in each update there is a good chance of getting the retransmissions by the end of each update. Total+FEC 20*20 has fewer and shorter vertical violet lines compared to 40*40 in green because having more redundant packets we could recover more lost messages and at the same time having smaller matrices we could recover them faster. The interesting and different thing about Figure 8c is that we can compare the blocking time of total+FEC 20*40 and total+FEC 20*20 when we could recover all lost messages in both redundant rates that is why there is not green or violet points around 70 ms. Therefore, the difference between total+FEC 20*40 and total+FEC 20*20 in Figure 8c is that having smaller matrices in 20*20 setting makes the loss recovery faster.

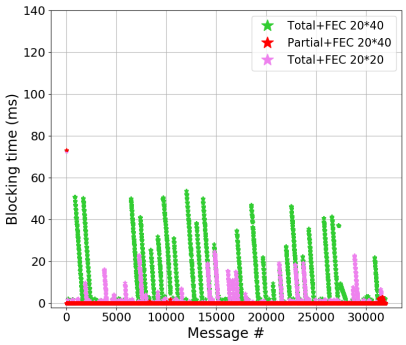
ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08751-6_35



(a) Update size: **1600** messages.
RTT: **20ms**.



(b) Update size: **1600** messages.
RTT: **70ms**.



(c) Update size: **800** messages.
RTT: **70ms**.

Fig. 8: The message blocking time (ms) for **1%** packet loss.

6 Concluding Remarks

There are some latency-sensitive workloads (e.g., remote desktops, bulk-data transfers) that do not require total order delivery supported by stream-based protocols such as TCP. When packets are lost in TCP, unnecessary head-of-line blocking can cause extra per-packet latency and lower overall performance. We propose a practical mechanism, called consistency fences, to support partial order delivery and help reduce the unnecessary blocking time and latency. We also show that combining forward error correction with partial order delivery setup makes a better improvement compared to total order delivery setup with the same rate of redundant data. We study the impact of increasing RTT and decreasing the update size on the effectiveness of CFs and show that for the

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08751-6_35

workloads with a pause between updates we will still get noticeable benefit using CFs under high RTT and low update size scenarios.

Studying the performance of our CFs implementation with loss-based CCAs remains as a future work. Also, we want to expand our evaluation of the combination of CFs and FEC to other more efficient FEC methods like rate-less codes such as Raptor codes [14].

Acknowledgments: Thank you to Steve Sutphen, the Natural Sciences and Engineering Research Council (NSERC) of Canada, and Huawei.

References

1. Balakrishnan, M., Marian, T., Birman, K.P., Weatherspoon, H., Ganesh, L.: Maelstrom: transparent error correction for communication between data centers. IEEE/ACM Transactions On Networking **19**(3), 617–629 (2011)
2. Connolly, T., Amer, P., Conrad, P.: An extension to tcp: Partial order service. Internet RFC1693, Nov (1994)
3. Eghbal, N., Lu, P.: Low-variance latency through forward error correction on wide-area networks. In: 2021 IEEE 46th Conference on Local Computer Networks (LCN). pp. 90–98. IEEE (2021)
4. Ferlin, S., Kucera, S., Claussen, H., Alay, Ö.: Mptcp meets fec: Supporting latency-sensitive applications over heterogeneous networks. IEEE/ACM Transactions on Networking **26**(5), 2005–2018 (2018)
5. Flach, T., Dukkipati, N., Cheng, Y., Raghavan, B.: Tcp instant recovery: Incorporating forward error correction in tcp. Working Draft, IETF Secretariat, Internet-Draft draft-flach-tcpm-fec-00, July (2013)
6. Gu, Y., Grossman, R.L.: Udt: Udp-based data transfer for high-speed wide area networks. Computer Networks **51**(7), 1777–1799 (2007)
7. Hemminger, S., et al.: Network emulation with netem. In: Linux conf au. vol. 5, p. 2005. Citeseer (2005)
8. Kim, M., Cloud, J., ParandehGheibi, A., Urbina, L., Fouli, K., Leith, D., Médard, M.: Network coded tcp (ctcp). arXiv preprint arXiv:1212.2291 (2012)
9. Langley, A., Riddoch, A., Wilk, A., Vicente, A., Krasic, C., Zhang, D., Yang, F., Kouranov, F., Swett, I., Iyengar, J., et al.: The quic transport protocol: Design and internet-scale deployment. In: Proceedings of the conference of the ACM special interest group on data communication. pp. 183–196 (2017)
10. Marx, R., De Decker, T., Quax, P., Lamotte, W.: Resource multiplexing and prioritization in http/2 over tcp versus http/3 over quic. In: International Conference on Web Information Systems and Technologies. pp. 96–126. Springer (2019)
11. Mosberger, D.: Memory consistency models. ACM SIGOPS Operating Systems Review **27**(1), 18–26 (1993)
12. Oku, K., Pardue, L.: Extensible prioritization scheme for http. Work in Progress, Internet-Draft, draft-ietfhttpbis-priority-02 **1** (2020)
13. Pooya, S., Lu, P., MacGregor, M.H.: Structured message transport. In: 2012 IEEE 31st International Performance Computing and Communications Conference (IPCCC). pp. 432–439. IEEE (2012)
14. Shokrollahi, A.: Raptor codes. IEEE transactions on information theory **52**(6), 2551–2567 (2006)
15. Stewart, R., Metz, C.: Sctp: new transport protocol for tcp/ip. IEEE Internet Computing **5**(6), 64–69 (2001)

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08751-6_35