DerechoDDS: Strongly Consistent Data Distribution for Mission-Critical Applications

Lorenzo Rosa
University of Bologna
Bologna, Italy
lorenzo.rosa@unibo.it

Weijia Song
Cornell University
Ithaca, New York, USA
ws393@cornell.edu

Luca Foschini
University of Bologna
Bologna, Italy
luca.foschini@unibo.it

Antonio Corradi
University of Bologna
Bologna, Italy
antonio.corradi@unibo.it

Ken Birman
Cornell University
Ithaca, New York, USA
ken@cs.cornell.edu

***Abstract*—Mission-critical applications frequently rely on *communication middleware products*, enabling ease of deployment, component integration, and proven dependability. However, existing communication middleware options present limitations such as weak consistency guarantees, reflecting concerns about overheads for strong forms of assurance. The hardware landscape is now evolving: hardware-based kernel bypass technologies like Remote Direct Memory Access (RDMA) offer faster communication with near-perfect reliability. This paper introduces DerechoDDS, an implementation of the OMG Data Distribution Service (DDS) layered over Derecho, an open-source library embodying a new approach to atomic multicast that maps efficiently to RDMA (or TCP emulations of RDMA). We first describe how DerechoDDS maps the standard DDS API on the Derecho library to achieve a zero-copy data path among remote entities. Then, we propose a novel QoS policy to control the level of consistency for data distribution. We demonstrate that DerechoDDS offers comparable or substantially higher performance than today's major DDS implementations, while simultaneously strengthening guarantees. Even when configured for strong consistency, DerechoDDS achieves high performance.**

***Index Terms*—DDS, state machine replication, consistency, QoS**

I. Introduction

Mission-critical applications are often object oriented with a distributed structure, in which components coded by different teams (and perhaps, using multiple languages and libraries) interoperate to solve the larger problem. For example, the command and control (C2) of an airplane or vehicle might have one component to control the motor, another for guidance, yet another for audio dialog between crew members, etc. Safety and correctness often leads to end-to-end quality of service (QoS) expectations in the connectivity technology. For example, although networking hardware may delay, drop or reorder packets, a C2 application may require low delay, high bandwidth, lossless order-based consistency, and so forth. Stronger QoS is presumed costly and used only as needed [1].

Many systems implement this approach by deploying a *middleware layer* between applications and the operating systems, protocol stacks and hardware. Such a layer reduces development time and effort, facilitating integration, reusability, extensibility, and better overall scalability. To ensure that components will interoperate correctly, standards are used such as the OMG Data Distribution Service (DDS) [2].

The OMG DDS defines a data-centric model in which components share typed distributed objects (*topics*), which jointly comprise an abstract global data space (GDS) and can be updated or monitored using the DDS API: the Data-Centric Publish-Subscribe (DCPS) interface. An OMG DDS uses QoS policies to let developers control properties such as reliability, consistency and many others. In particular, DDS policies can be combined to offer two consistency levels. *Weak consistency* is used when low latency is critical but some message loss can be tolerated. Configured for *eventual consistency* subscribers will eventually see an exact copy of the publisher local data store, but there may be long delays.

Our work begins with the observation that some applications need additional guarantees. For instance, when an application needs to replicate data over a set of components (for fault-tolerance, or to share some plan of action among components that will be each be responsible for a distinct aspect), the gold standard is to adopt a *state machine replication model* (SMR). However, the OMG QoS options are not strong enough to directly support SMR [3].

When the standard was defined, this reflected a pragmatic concern: At that time, the available SMR protocols were reputed too costly in terms of higher latency and reduced bandwidth. But the evolution of networking hardware and software calls this conclusion for reconsideration. Kernel-bypassing techniques such as Remote Direct Memory Access (RDMA) enable ultra-efficient high-speed direct data transfers between the virtual memory of processes on remote machines at DMA speeds, achieving high throughput ($>$200 Gbps) and very low latency ($\sim$1-2μs). RDMA emulation enables the same code to run over TCP but at lower speeds.

This paper introduces *DerechoDDS*, an OMG DDS platform that leverages RDMA to offer stronger consistency guarantees without performance penalties. DerechoDDS maps the DCPS interface to Derecho [4], a mature, open-source C++ library that offers multicast and point-to-point communication, supporting total ordering, failure atomicity, and optional durable message logging. Small-messages are sent with one-sided RDMA writes, and large ones using two-sided RDMA transfers over a binomial pipeline [5]. An efficient RDMA hardware mapping enables Derecho to break past performance records. This, in turn, allows us to offer new DDS QoS options supporting SMR while sustaining exceptionally high speeds and low latencies. DerechoDDS is in use by AFRL Wright-Patterson, as part of a middleware layer supporting a new generation of avionic architectures.

In the remainder of the paper, we briefly describe the architecture of DerechoDDS and motivate the need for a QoS policy that controls the consistency of the data distribution. We demonstrate that DerechoDDS performs at least as well as today DDS products, and outperforms them in some configurations - even as it offers higher reliability and consistency. Finally, we show that our higher consistency guarantees can better serve critical traffic under real-world traffic conditions.

II. Background

A. Relevant DDS QoS policies

We start with a short overview of the DDS QoS policies. DDS QoS covers a wide set of non-functional properties, expressing a contract between the local data stores of publishers and subscribers. DDS subscription-matching uses a request-vs-offered approach: a subscriber can subscribe to a topic (can "see" a shared object) only if the requested level of QoS is compatible with (not more restrictive than) the QoS offered by the publisher. This ensures that QoS invariants between publishers and subscribers are preserved [2].

The *Durability* QoS controls the lifetime of data written to the GDS. It supports four values: (1) *Volatile*, if data should be discarded immediately after delivery; (2) *Transient Local*, if data should be stored in the local cache of publishers and subscribers, thus allowing late joiners to catch up; (3) *Transient*, which ensures that data are kept even beyond the lifetime of single publishers or subscribers; (4) *Persistent*, which stores those data in persistent memory to make them survive system failures. The number of data samples (i.e., subsequent writes of the same topic) that must be stored in local cache is determined by the *History* QoS policy, that can assume values "all" or "last n". Together, Durability and History determine the dataset and the time period for which consistency should be maintained.

Three additional policies determine the kind of consistency required. The *Reliability* QoS controls the reliability associated with the transport-level protocol (by default, UDP). The two possible values are *best-effort* and *reliable*. The latter corresponds to the eventual consistency property discussed above: if a publication is somehow lost or delayed, the middleware will arrange for it to be retransmitted, even if this might delay other publications. The *Lifespan* QoS determines the time interval during which a data sample is to be considered valid (either infinite, or a time period). Finally, the *Destination Order* policy controls the delivery to the subscriber of the different updates according to source or destination timestamps.

With respect to the amount of data defined by the Durability and History policies, DDS applies an *eventual consistency* model when Reliability is *reliable*, Lifespan is *infinite*, and Destination Order is *by source timestamp*. Otherwise, consistency is weak as defined in section I. Recall that all DDS policies are enforced for publisher-subscriber pairs, hence there is no option that will impose consistency constraints on groups of subscribers, where multiple system components are tracking the same data. For example, no QoS property will ensure that updates to some topic will be delivered in the same order to all subscribers, or not delivered at all (the core requirement of SMR, and implemented by atomic multicast).

Our work is motivated by this omission: the Air Force applications in which DerechoDDS will be used may include mission-critical tasks, for which adherence to an SMR model can enhance safety and predictability. SMR also enables multi-component coordination paradigms that could be particularly powerful in federated ML settings, where a graphical structure of ML components is deployed as an adjunct to more classical flight-control elements. SMR also lends itself to formal verification and formal reasoning about higher level system properties. Accordingly, DerechoDDS offers a novel SMR-based QoS option.

B. The Derecho library

As noted in the introduction, DerechoDDS layers the DCPS API over Derecho [4]. The Derecho model organizes a group of processes (*nodes*) as a distributed service that can execute either on physical servers or in virtual environments. The membership of this top-level group evolves as processes join, terminate or crash through a series of *views* using partition-free state machine replication. To build a distributed service, developers define"replicated objects," each consisting of a state and a set of operations that operate on it. Processes holding replicas of such an object would be a subgroup of the top-level group membership. Each state update can then be forwarded as a multicast to all the subgroup members and performed by all replicas. Although Derecho supports a weakly reliable multicast, the workhorse of the system is a novel implementation of atomic multicast, which has been proved to correctly implement the SMR model. On an RDMA network, Derecho offers a zero-copy, lock-free critical path among remote applications, leading to exceptionally high bandwidth utilization and very low latency. A more detailed discussion about the properties of RDMA networks and how DerechoDDS leverages them to obtain high network performance is out of the scope of this paper, but we extensively discuss this topic in a previous work [6].

III. Related Work

Many DDS vendors recognize the high cost of unnecessary copying. The usual response is to offer a zero-copy data path that leverages shared-memory, so that when an object is shared by a publisher and subscriber *on the same machine*, they have direct access. However, our work is the first to extend this style of sharing to work across a network, with automated failure handling based on the heavily-studied and formalized SMR model. We believe that this sort of configuration will be of growing importance as new memory models and smart NIC models become standard, and as operating systems increasingly support kernel-bypassing network hardware.

Although our work expands the standard DDS QoS options, we are not alone in doing so. For example, RTI Connext offers an *Availability* QoS [7]. This property allows subscribers to reconstruct an agreed ordering of updates when a shared object is updated by multiple publishers. Availability can also be used

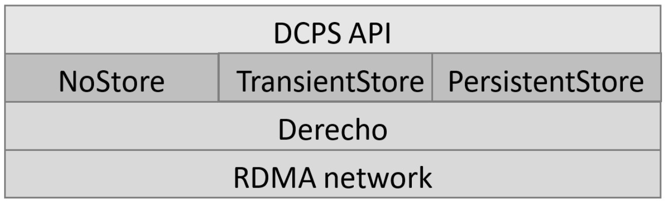


Fig. 1: Architecture of DerechoDDS.

by publishers to define a statically-defined group of *required subscribers* to their updates. However, whereas DerechoDDS has been formally shown to implement a fault-tolerant SMR model, the RTI availability QoS level is not strong enough to support a true atomic multicast in a dynamic group processes that may experience hardware and software crashes.

Our focus on the formal SMR model centers on the opportunity to leverage the rich body of research associated with it. For example, there has been a great deal of recent work on software verification tools for SMR-based protocols and applications. In settings like avionics components for fly-by-wire aircrafts, or other mission-critical control tasks, such tools can strengthen assurances that the platform will operate in a safe and correct manner. Moreover, DerechoDDS itself can be proved correct and verified (as was done in [4] and follow-on work). We are not the first to make this observation: Leslie Lamport's Paxos SMR protocols were designed with proofs in mind, and Lorch *et. al.* implemented a fully proved Paxos-based atomic multicast called IronFleet [8], although without supporting a DDS API or leveraging RDMA.

The evaluation portion of this paper compares four DDS implementations under a variety of metrics. The last general comparison of this kind occurred many years ago [9]–[11], for an earlier generation of hardware and DDS products. A more recent comparison was included in a study of publish/subscribe options for Industrial IoT, which included DDS but also looked at other publish-subscribe models, such as message-queuing middleware [12]. The authors of this more recent study noted that its extensive set of QoS policies make DDS customizable for various scenarios that require a high level of reliability.

We believe that the stronger consistency option proposed in this paper could take such an argument one-step further by enabling the mission-critical community to leverage proof tools and formal methods in the context of DDS-based applications. From a performance perspective, in this same work, the authors stress the importance of batching techniques to improve throughput performance. In work that will be published elsewhere, we show how valuable batching can be in DerechoDDS. The study also recommends enabling (when available) auto-throttling to slow publishers down when the network is congested. As we see in section V, this is indeed important for other DDS products, but not for DerechoDDS, which leverages rate control into RDMA to ensure that data will not be lost in transmission.

IV. DerechoDDS

We now have the context to dive somewhat deeper into DerechoDDS, which implements the OMG Data-Centric Publish-Subscribe (DCPS) API [2]. DerechoDDS is coded in C++ 17, enabling an exceptionally lightweight mapping from the DCPS API to the APIs already present within Derecho, which is also coded in C++ [4]. Given a C++ application the compiler is able to eliminate overheads for this layering at compile time. As a result, DerechoDDS enables the user application to transparently benefit from kernel-bypassing RDMA. The publisher simply operates directly on the shared object, with the effect that the published message is created "in place," i.e., directly in the memory region that Derecho will copy to remote peers via RDMA. On RDMA networks, where even a single message copy or a single lock can sharply degrade performance, this avoids significant overheads. DerechoDDS can also support applications in other languages, such as Python and Java, but at some loss of efficiency: their object representation formats, automated memory-management, and garbage collection features inject unavoidable locking and copying. Here we evaluate only the C++ case.

DerechoDDS stores. The first step to build DerechoDDS was to model the concept of DDS Global Data Space (GDS) introduced in section I. We addressed this by defining a set of distributed key-value stores, one for each *durability* option: *NoStore* for volatile, *TransientStore* for transient local, and *PersistentStore* for persistent (Fig. 1). As a consequence, we can represent DDS Topics as objects of a user-defined type that live in one of these three K/V stores: The object key serves as a topic name (for *keyless* topics [2]) or the name plus a set of fields of the corresponding type (for *keyed* topics). Once we defined this mapping, we implemented these three stores as Derecho replicated objects. First, we defined a basic set of operations to access the store: The most important one is `put`, which inserts a new object in the store, if the corresponding key does not exist, or updates its value if it already exists. Then we modeled the state on which such operations act, which depends on the desired durability level:

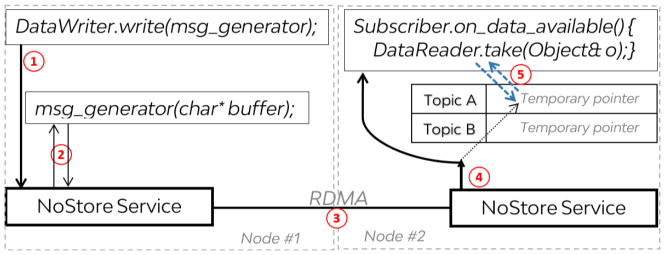
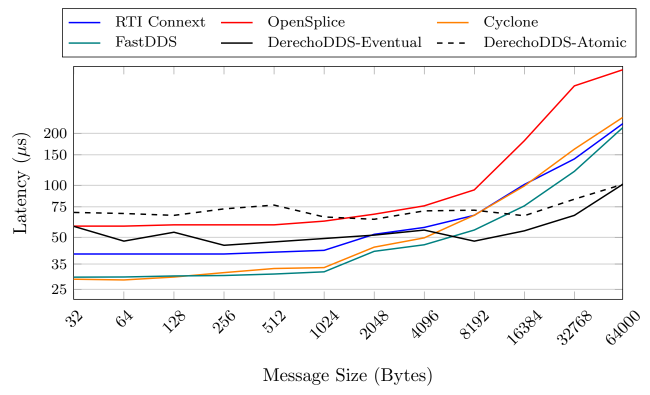

DataWriter.write(msg_generator);
1
msg_generator(char* buffer);
2
NoStore Service
RDMA
3
Node #1
Node #2
Subscriber.on_data_available(){
DataReader.take(Object& o);}
5
Topic A
Temporary pointer
Topic B
Temporary pointer
4
NoStore Service

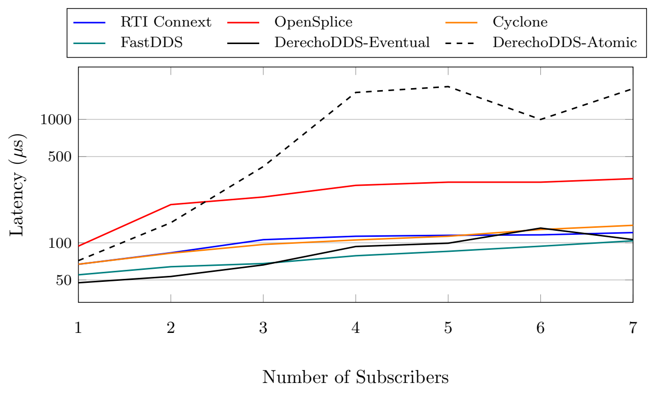
Fig. 2: Zero-Copy data path of DerechoDDS for Volatile durability.

Durability QoS	Consistency QoS	Derecho Service
Volatile	Eventual Atomic	NoStore, Unordered subgroup NoStore, Ordered subgroup
TransientLocal	Eventual Atomic	TransientStore, Unordered subgroup TransientStore, Ordered subgroup
Persistent	Eventual Atomic	PersistentStore, Unordered subgroup PersistentStore, Ordered subgroup

TABLE I: Mapping of DDS *Durability* QoS on Derecho.

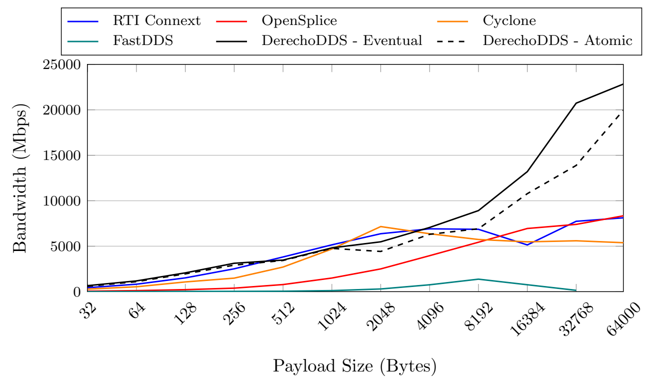


(a) Increasing payload sizes with 1 publisher and 1 subscriber

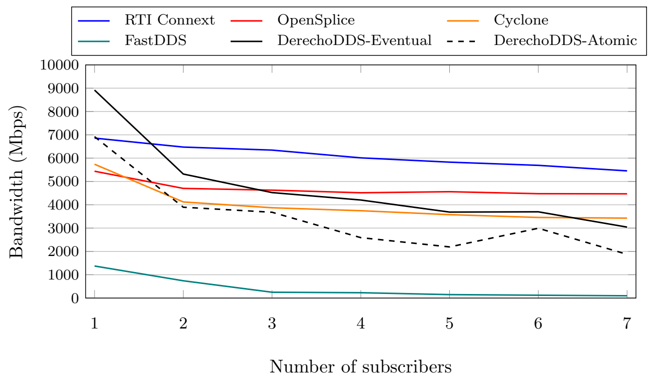


(b) 8KB payload size and an increasing number of subscribers

Fig. 3: Median Round-Trip Time for different payload sizes and numbers of subscribers.



(a) Increasing payload sizes with 1 publisher and 1 subscriber



(b) 8KB payload size and an increasing number of subscribers

Fig. 4: Average throughput for different payload sizes and number of subscribers.

the *NoStore* store does not have an actual state, as updates fade after application delivery. The *Transient* store keeps the last n received updates in main memory, as well as the *Persistent* store which additionally backs them up in persistent storage. For reasons of brevity,we will focus on the Volatile case.

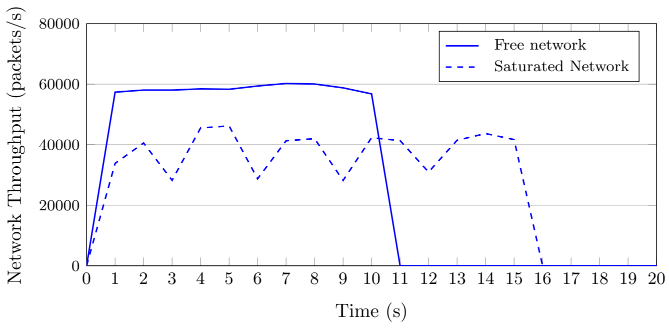
Zero-copy data path. Once the stores were defined, we mapped the standard DDS interface onto them as seen in Fig. 2. As a first step the publisher registers the generating function by calling `DataWriter.write`. Internally, DerechoDDS calls the `put` operation to update the topic value on the corresponding store, which in turn will ask Derecho a free memory buffer. When the buffer is available, Derecho requests that the user-supplied message generation logic build the message (step 2), after which it can push the message remotely via Derecho multicast (step 3). On reception, the Derecho core notifies DerechoDDS of the update. The middleware places in a topic-specific variable a pointer to the received message, and then invokes the subscriber's listener (step 4). Within the listener, the user-provided logic retrieves the value (step 5).

Consistency QoS. Our new QoS policy, *Consistency*, allows users to enhance the maximum DDS consistency level from *eventual* to *atomic*. Consistency offers two possible values. The *eventual* setting selects for the standard OMG behavior, while the *atomic* option selects for SMR guarantees. Recall that Derecho itself has two forms of multicast: a weakly reliable one, and an atomic option. Accordingly, it suffices for DerechoDDS to employ the appropriate primitive, as seen in Table I. If *eventual* consistency is selected, DerechoDDS will deliver any update to the relevant subscribers as soon as it is available. In contrast, for *atomic* consistency, DerechoDDS selects the Derecho atomic multicast, which will delay delivery until the SMR obligations of totally ordered, fault-tolerant delivery can be assured. As such, the latency costs of the atomic option are of particular interest, and we evaluate them carefully in the next section.

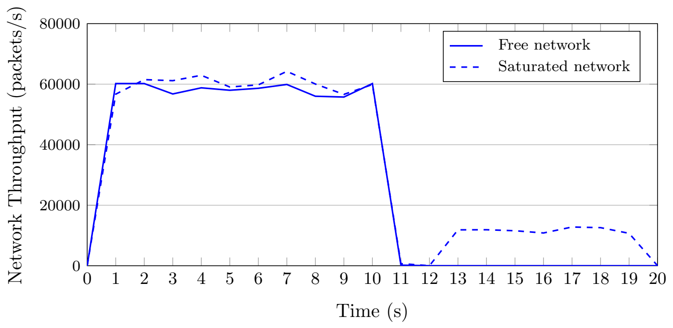
V. Experimental Evaluation

We now assess the performance of DerechoDDS. Our goal is to demonstrate that not only do the added consistency guarantees not harm performance when compared with weak consistency, but also that DerechoDDS can match the performance limits of the hardware in the atomic mode, and can ride out periods when the network briefly becomes fully saturated. We believe that this opens the door to use of SMR even in today's most demanding mission-critical scenarios.

All our tests were performed on the CloudLab platform [13], which offers on-demand bare metal nodes. We used 8 physical hosts, each equipped with a ten-core Intel E5-2640v4 2.4 GHz processor, 64GB memory, and a dual-port Mellanox ConnectX-4 25 Gbps NIC.

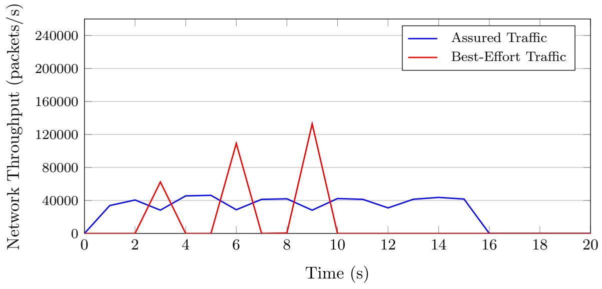


(a) DerechoDDS over RDMA

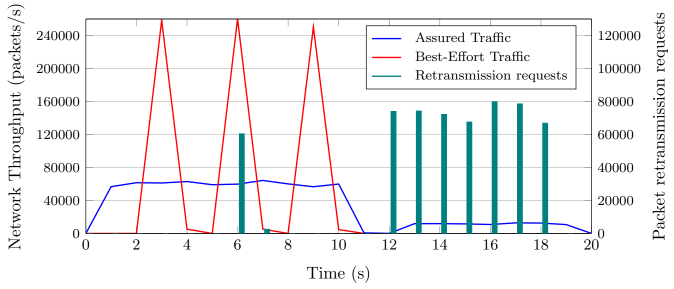


(b) RTI Connext over UDP multicast

Fig. 5: Throughput of the critical traffic flow under different network conditions



(a) DerechoDDS over RDMA



(b) RTI Connext over *reliable* UDP multicast

Fig. 6: Impact of bursts of low-importance traffic on a critical traffic flow.

A. Latency and Throughput

We begin with a performance comparison of DerechoDDS against four DDS implementations that we selected for their widespread adoption in the community. Two are mature products: RTI Connext 6.0 [7] (the only non-open-source option) and Adlink Opensplice Community 6.9 [14]. The other two are emerging implementations, Eclipse Cyclone DDS 0.7 [15] and eProsima FastDDS 2.3 [16]. Our comparison includes a latency and a throughput test, each repeated at least 5 times for different payload sizes and for different numbers of subscribers. The single publisher and the subscribers are all on different nodes to stress the network performance. We used *volatile* durability, *reliable* reliability, and UDP multicast as the transport protocol to obtain a consistency level equivalent to the *eventual* consistency of DerechoDDS on RDMA.

The latency test is a simple ping-pong application designed to highlight any overhead in the DDS send and receive pipeline. Performed on a dedicated network, this test measures the round-trip time (RTT) of every sample published on a “ping” topic and received by a remote subscriber which sends it back to the publisher. In case of multiple subscribers, the first “pong” message is considered. We run this test for 60 seconds, disabling sample batching and using a *keep-last-1* history. With small messages in a single subscriber scenario (Fig. 3a), DerechoDDS in *eventual* consistency mode exhibits approximately 50% higher latency than the best UDP-based alternatives, even though it still performs better than some of the products. On average, the strong consistency guarantee adds another 40% performance penalty for those sizes. These numbers could be substantially improved with a careful optimization of our prototypical implementation, as RDMA is incredibly sensitive even to tiny delays. But this pattern only holds up to 4KB. With message sizes higher than 8KB, the situation is reversed: even the *atomic* mode has a 2x lower median latency than the alternatives: This is where the true potential of RDMA emerges, as protocol-induced delays becomes negligible. If we scale the 8KB case to more subscribers (Fig. 3b), we see that the performance advantage is substantially preserved for the eventually consistent case, but that DerechoDDS with its atomic guarantee incurs a delay while waiting to ensure that the SMR properties have been achieved. This delay rises to as much as 1ms, but then stabilizes and remains constant as the number of subscribers is increased from 4 to 7.

The throughput test asks whether each DDS can fully saturate available bandwidth when a publisher continuously updates a topic with one remote subscriber: a crucial capability on high-speed networks. Fig. 4 plots the results. We observe that FastDDS is much slower than the others. This product lacks data batching, putting it at a substantial disadvantage. DerechoDDS on RDMA almost saturates the available bandwidth for bigger message sizes, a result impossible to obtain when using UDP and the *reliable* QoS: the traditional networking stack cannot handle such a high throughput, so many packets are lost and the retransmission cost increases. We also observe that the *atomic* mode in DerechoDDS does not suffer an excessive overhead, and for 64KB payload size it is still 2.3x faster than the best existing DDS implementation.

B. Reliability

This experiment compares DerechoDDS with RTI Connext (fastest among the four DDS products). One publisher writes small (128 byte) critical updates with *reliable* QoS. DerechoDDS always runs in reliable mode, but is additionally configured with *atomic* consistency. We picked a constant data publishing rate of 60K samples per second, which guarantees that no UDP packets are lost by RTI Connext, and publish for 10s. Both DDS products deliver all messages within 11s (solid lines in Fig. 5).

Next, we introduce a second publisher and a second remote subscriber on the same network. These produce and consume low-importance data, expressed using the *best-effort* reliability for RTI Connext and *eventual* consistency for DerechoDDS. In our first experiment, we configure the second producer to generate a steady rate of background traffic designed to fully saturate the network link when both publishers are running at once (dashed lines). We see that DerechoDDS obtains a reduced share of the network, requiring 16 seconds to complete the transmissions, but then is finished. In contrast, RTI exceeds the peak network capacity, causing some packets to be dropped because of congestion. In reliable QoS mode, these must later be retransmitted, so we see a series of retransmission requests (blue bars) and a second wave of deliveries, ending after 20s.

As a final experiment we reconfigure our second publisher to be bursty: it pauses for 2s, then sends rapidly for 1s. Fig. 6 plots the results. RTI Connext sends at full speed regardless of network load, causing a high rate of lost packets, so the subscriber issues many retransmission requests and the total test time once again jumps from 11s to 20s. DerechoDDS runs at a slightly lower bandwidth but with no loss: the RDMA hardware has a built-in mechanism that only transmits data when the receiver is ready for the incoming bytes. Fig. 5a shows that although the data rate of the critical flow drops from 60K to 40K packets per second, the experiment completes in 16 seconds, 20% faster than for RTI.

We should note that RTI connect offers a proprietary API with which the application can explicitly throttle its rate of publications. A knowledgeable user could configure the two DDS applications (critical and background) to prevent loss in this experiment. We did not evaluate this option because it is not automated: the application designer must anticipate the congestion conditions and specify the peak rate of transmission for each topic.

VI. Concluding remarks

As smart NIC hardware costs drop, operating systems evolve to support kernel-bypassing techniques, and lock-free zero-copy computing gain adoption, it is time to revisit the OMG DDS QoS options to support stronger consistency and fault-tolerance guarantees that were omitted in past standardization work due to concerns about overheads. DerechoDDS is a new OMG-compliant DDS solution, available in open-source, that addresses this limitation. Our evaluation shows that even in its strongest QoS configuration, DerechoDDS is highly efficient, equaling or exceeding the bandwidth of existing DDS products while also reducing latency. The work reported here should be understood as a snapshot: we are continuing to extend DerechoDDS. Near term goals include support for "external" clients with non-RDMA links to the platform, efficient operation over 5G networking devices using the DPDK standard (perhaps, via the URDMA package). There is also a great deal of interest in Time Sensitive Networking (TSN). In the longer term, we are extremely interested in applications that combine the DDS model with AI and ML code, particularly under tight deadlines and demanding performance conditions.

Acknowledgments The authors wish to acknowledge the support received from AFRL/RY (Wright-Patterson), Microsoft, Siemens and Nvidia corporations. We additionally wish to express our gratitude to the anonymous reviewers for their constructive comments and suggestions.

References

[1] D. Schmidt and F. Kuhns, "An overview of the real-time corba specification," *Computer*, vol. 33, no. 6, pp. 56–63, 2000.

[2] "OMG Data Distribution Standard." [Online]. Available: https://www.dds-foundation.org/omg-dds-standard

[3] F. B. Schneider, "Implementing fault-tolerant services using the state machine approach: A tutorial," *ACM Comput. Surv.*, vol. 22, no. 4, p. 299–319, Dec. 1990.

[4] S. Jha, J. Behrens, T. Gkountouvas, M. Milano, W. Song, E. Tremel, R. V. Renesse, S. Zink, and K. P. Birman, "Derecho: Fast state machine replication for cloud services," *ACM Trans. Comput. Syst.*, vol. 36, no. 2, Apr. 2019.

[5] Mellanox, "RDMA aware networks programming user manual." [Online]. Available: https://www.mellanox.com/related-docs/prod_software/RDMA_Aware_Programming_user_manual.pdf

[6] L. Rosa, S. Jha, and K. Birman, "DerechoDDS: Efficiently leveraging RDMA for fast and consistent data distribution," in *CARS 2021 6th International Workshop on Critical Automotive Applications: Robustness & Safety*, Munich, Germany, 2021.

[7] RTI, "Rti connext dds. core libraries user's manual." [Online]. Available: https://community.rti.com/documentation

[8] C. Hawblitzel, J. Howell, M. Kapritsos, J. R. Lorch, B. Parno, M. L. Roberts, S. Setty, and B. Zill, "Ironfleet: Proving safety and liveness of practical distributed systems," *Commun. ACM*, vol. 60, no. 7, p. 83–92, Jun. 2017.

[9] P. Bellavista, A. Corradi, L. Foschini, and A. Pernafini, "Data distribution service (dds): A performance comparison of opensplice and rti implementations," in *2013 IEEE Symposium on Computers and Communications (ISCC)*, 2013, pp. 377–383.

[10] T. Rizano, L. Abeni, and L. Palopoli, "Experimental evaluation of the real-time performance of publish-subscribe middlewares," in *REACTION*, 2013.

[11] K. Krinkin, A. Filatov, A. Filatov, O. Kurishev, and A. Lyanguzov, "Data distribution services performance evaluation framework," in *2018 22nd Conference of Open Innovations Association (FRUCT)*, 2018, pp. 94–100.

[12] Z. Kang, R. Canady, A. Dubey, A. Gokhale, S. Shekhar, and M. Sedlacek, "A study of publish/subscribe middleware under different iot traffic conditions," in *Proceedings of the International Workshop on Middleware and Applications for the Internet of Things*, ser. M4IoT'20. New York, NY, USA: Association for Computing Machinery, 2020, p. 7–12.

[13] D. Duplyakin *et al.*, "The design and operation of CloudLab," in *Proceedings of the USENIX Annual Technical Conference (ATC)*, Jul. 2019, pp. 1–14.

[14] Adlink, "Vortex opensplice." [Online]. Available: https://www.adlinktech.com/en/vortex-opensplice-data-distribution-service

[15] Eclipse Foundation, "Eclipse cyclone dds." [Online]. Available: https://projects.eclipse.org/projects/iot.cyclonedds

[16] eProsima, "Fastdds." [Online]. Available: https://www.eprosima.com/products-all/eprosima-fast-dds