Poster: In-Network Total Order Guarantees supporting State Machine Replication with P4 Programmable Switches

Bochra Boughzala
University of Groningen, The Netherlands

Boris Koldehofe
Technische Universität Ilmenau, Germany

***Abstract*—In this paper we propose P4-based Atomic Multicast (P4mCast), a new *in-network* atomic multicast protocol to support total order guarantees for State Machine Replication (SMR) in cloud-based fault-tolerant and distributed applications. P4mCast builds on *in-network computing*, applying *leader-based consensus* to groups of prominent P4 programmable switches in modern data center networks. P4mCast achieves significantly lower latency overhead in the microseconds scale while increasing the throughput one order of magnitude higher compared to state of the art software-based solutions.**

***Index Terms*—Atomic multicast, consensus, hardware offload, in-network computing.**

I. Introduction

State Machine Replication (SMR) is an important paradigm that enables fault-tolerance in distributed systems. When replicated systems, e.g., a distributed cloud-based database, have a large state to maintain, it becomes more efficient to communicate on updates only [1]. Particularly, to achieve strong consistency between the replicas of the different services, total order atomic multicast, an essential group communication abstraction is often used to ensure the reliable delivery of messages with total order guarantees [1]. By receiving the same set of updates in the same order the states of the individual replicas progress in the same manner. Total order multicast, a generalization of total order broadcast which is an equivalent problem to consensus, relies on complex semantics, e.g., reliable broadcast and total order, which are typically built within a separate communication layer as a middleware service on top of the network. The components of the replicated system are organized in groups of one leader and a set of followers where *leader-based consensus* is executed inside each group. The leaders receive the atomic multicast requests and assign local timestamps to the messages, the followers acknowledge the leader proposals to eventually decide on the final value, i.e., the message global timestamp. The messages are then ordered and delivered to the upper application in an increasing order of their final timestamps.

As SMR-based cloud services are evolving in size and complexity, total order multicast becomes a challenging task which must be as efficient as possible. Recent software-based proposals [2], [3] are adding new optimizations to reduce the latency overhead of atomic multicast operations. For instance, PrimCast [3], a state of the art protocol achieves a total delay of 3 communication steps. However, the latency overhead remains in the scale of tens to hundreds of milliseconds.

979-8-3503-5171-2/24/$31.00 ©2024 IEEE

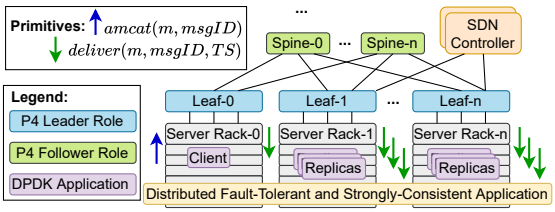

Fig. 1. P4mCast System Architecture.

In this paper, we are interested in improving its performance using hardware assistance. We can leverage Programming Protocol-independent Packet Processors (P4) [5] and corresponding *in-network* P4 programmables switches which have shown significant performance benefits and increased flexibility in describing new protocols and expressing tailor packet processing behavior. We propose P4mCast, a new P4-based atomic multicast protocol supporting the in-network execution of total order multicast. The motivation behind P4mCast is to enable higher message ingestion rate and increase the throughput of ordered messages while reducing the latency overhead. The main challenge of P4mCast is to provide the required robustness and reliability guarantees of total order multicast at the network level. A preliminary evaluation shows that compared to Primcast, P4mCast provides for one order of magnitude higher throughput a significantly lower latency overhead of tens of microseconds.

II. Design Overview

The proposed P4mCast protocol is designed based on the system architecture shown in Fig. 1. A cloud data center network comprised of a set of P4 programmable switches is built according to a multi-tier clos network topology, such as the *leaf-spine* architecture [4]. Distributed and fault-tolerant applications are hosted on commodity rack servers where multiple replicas are spread across the different racks. Each replica runs a sender (to initiate the atomic multicast requests) and receiver (to deliver ordered messages) processes which can be executed as Data Plane Development Kit (DPDK) applications.

The leaf and spine switches implement and execute the total order atomic multicast protocol P4mCast. The leaf switches are assigned the *leader* role. Since they represent the Top-Of-Rack (ToR) switches connecting servers from the same rack, they are the entry point from which the traffic ingresses the network and the earliest stage where the total order multicast requests are received and can be handled by the network. The spine switches are assigned the *follower* role, they depict the network aggregation layer and are essential for enabling the traffic routing between servers from different racks. The switches are organized in groups with a size $N = 2f + 1$ switches of one leader and a set of followers where up to f switches may fail, including the leader. The P4-based leaf and spine switches are managed by a centralized Software-Defined Networking (SDN) Controller which keeps an updated global view of the network. The SDN controller does not participate in the ordering of messages which happens only in the data plane, i.e., *fast path*, in a fully decentralized manner. A leader failure can be detected by the SDN controller, i.e., *slow path*, which would trigger a leader re-election.

A. P4mCast Protocol

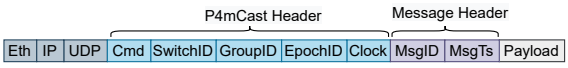

Fig. 2. P4mCast Protocol Header.

In P4mCast, an atomic multicast request corresponds to a packet. A P4mCast packet is built on top of the User Datagram Protocol (UDP) transport layer. It includes the P4mCast header and message header followed by the payload (Fig. 2). An example of a normal execution of P4mCast is presented in Fig. 3. The sender process (client) initiates an atomic multicast request for a message m identified by a unique message ID $msgID$ by invoking `amcast(m, msgID)` primitive. The message is then encapsulated in a *start* P4mCast packet and sent to the leaders of the different groups belonging to $m.dest$. Subsequently, the leader advances its internal clock ($clock = clock + 1$), assigns a local timestamps $lts = clock$ to the message m and sends a proposal to all followers in $m.dest$ (own group and other groups) via the *prop* command. When receiving a proposal, each follower first updates its local clock ($clock = max(clock, ts)$) then sends an acknowledgment to all members in $m.dest$ via the *ack* command. Upon receiving an acknowledgment, each switch node updates the number of acknowledgments in a persisted register value. When the total count of acknowledgments hits the majority, i.e., $quorum = sizeof(m.dest)/2 + 1$, in each group then the final timestamp is decided as the maximum of all local timestamps. The nodes send the message with its final timestamp in a *deliver* P4mCast packet to corresponding replicas where the messages will be delivered to the upper application in an increasing order of their final timestamps by calling the `deliver(m,msgID,ts)` routine.

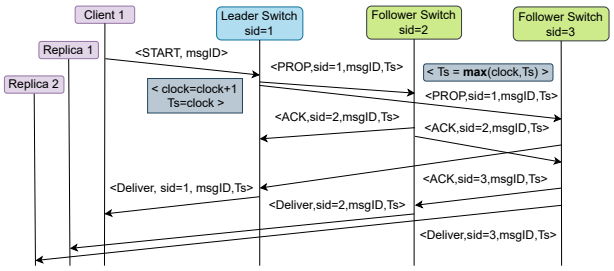

Fig. 3. Normal execution of P4mCast for a request initiated by Client1, simplified by showing the interactions of one group.

III. Preliminary Results

We realized a prototype of P4mCast in $P4_{16}$ programming language [6] for the Intel Tofino switch [7]. In our testing scenario we consider one group of hypothetically three nodes, i.e., one leader and 2 followers where up to one switch failure can be accepted. Concretely, we have two Tofino1 switches, i.e., one leader (leaf) and one follower (spine). Our hardware testbed also comprises two Dell PowerEdge R750 rackserver with Intel(R) Xeon(R) Platinum 8358 (2.60GHz) as CPU. Both servers are equipped of an Intel E810-XXVAM2 and an Intel X710-BM2 Dual-Port Ethernet Network Interface Card (NIC). In our implementation, the P4mCast program deployed on the switches incorporates both the leader and the follower logic. The specific switch role is configured at runtime. To evaluate the performance of P4mCast, we used T-Rex [8], a DPDK-based traffic generator on one of the servers to inject UDP traffic over 10Gb speed link into the leaf switch. The UDP packets were augmented by the message header i.e., *msgID*, and the P4mCast header with the `start` command to initiate the atomic multicast requests sent to the leader switch. We measured the end-to-end network latency of P4mCast, from *start* to *deliver* command and then computed the average value for a varying packet sizes (150 to 1500 Bytes) and increasing throughput. The performance results presented in Fig. 4 show that P4mCast provides a major latency reduction of 12 microseconds for 1500 Bytes packets and one order of magnitude higher throughput compared to Primcast. Fig. 4 also shows the higher the throughput the smaller the latency which is a property of the Tofino pipeline for small packets.

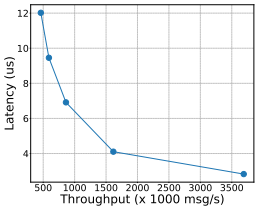

Fig. 4. P4mCast Throughput vs. Latency.

IV. Conclusion

As a next step, we aim at refining our prototype in dealing with message loss and failures of the sender and the leader.

REFERENCES

[1] Birman, K., 2006. Reliable distributed systems: technologies, web services, and applications. Springer Science & Business Media.

[2] Gotsman, A., Lefort, A. and Chockler, G., 2019, June. White-box atomic multicast. In Proceedings of the 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN) (pp. 176-187). IEEE.

[3] Pacheco, L., Coelho, P. and Pedone, F., 2023, November. PrimCast: A Latency-Efficient Atomic Multicast. In Proceedings of the 24th International Middleware Conference (pp. 124-136).

[4] Singh, A., Ong, J., Agarwal, A., Anderson, G., Armistead, A., Bannon, R., Boving, S., Desai, G., Felderman, B., Germano, P. and Kanagala, A., 2015. Jupiter rising: A decade of clos topologies and centralized control in google's datacenter network. ACM SIGCOMM computer communication review, 45(4), pp.183-197.

[5] Bosshart, P., Daly, D., Gibb, G., Izzard, M., McKeown, N., Rexford, J., Schlesinger, C., Talayco, D., Vahdat, A., Varghese, G. and Walker, D., 2014. P4: Programming protocol-independent packet processors. ACM SIGCOMM Computer Communication Review, 44(3), pp.87-95.

[6] $P4_{16}$ Language Specification. (accessed August, 2024) https://p4.org/p4-spec/docs/P4-16-v1.0.0-spec.html

[7] Intel®Tofino. (accessed August, 2024) https://www.intel.com/content/www/us/en/products/details/network-io/intelligent-fabric-processors/tofino.html

[8] TRex. Realistic Traffic Generator. (accessed August, 2024) https://trex-tgn.cisco.com/