Zenoh-based Dataflow Framework for Autonomous Vehicles

Gabriele Baldoni
ADLINK Technology Inc, France

Luca Cominardi
ADLINK Technology Inc, France

Angelo Corsaro
ADLINK Technology Inc, France

Yong He
Futurewei Technologies, Inc, USA

Abstract—Autonomous Vehicle Softwares are complex to implement. They have strong security and safety constraints and are at the crossroad of several domains, thus leading to as many independent components. Plus, as foreseen by the Vehicle to Everything paradigm, Autonomous Vehicles are expected to interact with their surroundings, again increasing their overall complexity.

To cope with these requirements, frameworks based on dataflow programming are emerging. Indeed, dataflow programming is particularly suited in this context as it is already being used in robotics and time-critical applications. However, it was not designed with decentralization in mind, hence leaving all communication-related aspects to application developers.

To address these challenges, we propose to leverage Eclipse Zenoh, an Edge-native technology. For that purpose, we first show how we used Zenoh to enhance ERDOS, a dataflow framework for Autonomous Vehicles. We then motivate and draft our own dataflow framework based on Zenoh: Zenoh Flow.

Index Terms—Dataflow, edge, V2X, automotive, robotics, zenoh

I. INTRODUCTION

Traditionally, car embedded software has been tightly coupled with its hardware counterpart, a prime example being the Electronic Control Units (ECU) [1]. For instance, the ECU used to control an engine is designed based on the engine’s characteristics (e.g., number of cylinders, fuel used, expected operating temperature, etc.) and the associated software only works for that specific design.

Recently, the software industry is shifting towards a service-oriented approach [2] where components are viewed as interchangeable pieces, hence removing any tight coupling. This shift is gaining the automotive industry thanks to new advances made in the software industry, in particular, dataflow programming [6] and the Cloud-to-Thing continuum [7].

Dataflow programming is often considered alongside a service-oriented approach — for instance, in safety-critical applications such as for radars [8], [9] and in robotics [10], [11]. In dataflow programming, applications are decomposed as a graph of components, called operators from now on, that can be executed concurrently and, potentially, on different devices across the infrastructure.

Establishing an infrastructure that supports end-to-end applications that span from the far-Edge (the “things”) to the Cloud is a vision supported by the Cloud-to-Thing continuum. The computing, storage and networking resources are assumed to be fully-distributed and managed to provide the abstraction of a continuous virtualization infrastructure. Eclipse Zenoh [12] is an Edge-native technology that was built specifically for this perspective: one of its prominent features is to provide a unified view of the data regardless of their physical location, effectively abstracting the underlying infrastructure.

Therefore, by combining dataflow programming and the decentralized infrastructure of the Cloud-to-Thing continuum abstracted through an Edge-native technology like Zenoh, automotive application developers can tackle the new requirements and challenges raised by autonomous vehicles without worrying about how their operators would communicate. Operators can then be spawned in the most appropriate location w.r.t. the various requirements on computing, networking or access to the physical world and seamlessly accessed through a unified abstraction.

In this paper, we show (i) how we used Zenoh to enhance the dataflow-based autonomous vehicle framework ERDOS [13] and achieved these properties, and (ii) how, building on this experience, we designed Zenoh Flow – an edge-native Zenoh-

This work was funded by FutureWei Technologies, Inc., with contributions from the H2020 collaborative research project DAEMON (grant no. 101017109).

1Zenoh Project: https://zenoh.io/ [Accessed: 9 Oct 2021]
The remainder of this paper is organized as following: Section II gives the necessary background on Dataflow programming and the Cloud-to-Thing continuum; Section III discusses Zenoh in more details and explains how we integrated it with the ERDOS framework; Section IV gives a high-level overview of Zenoh Flow, our Edge-native framework for autonomous robots and vehicles. Finally, Section V concludes the paper and delineates future perspectives.

II. BACKGROUND

A. Dataflow programming

The dataflow programming model [6] allows to organise programs as directed graphs where nodes are computational units, i.e. the operators, and the arcs between them symbolize the communication channels. Figure 1 shows a representation of a dataflow graph. Arcs arriving to an operator represent the inputs and those departing from it the outputs.

Upon receiving its inputs, an operator can trigger its computation. Depending on the dataflow model, an operator either waits for all of its inputs [15] or for a specific subset, called a firing set or a firing rule [16]. Firing rules offer more flexibility to application developers, as they can express more conditions: for example, if an input is optional, the computation can still be triggered when it is missing.

As operators are independent — i.e. they only wait on their inputs — they can be simultaneously triggered in parallel. This property makes them especially suitable in environments where many computational devices are available.

B. Cloud-to-Thing continuum

The past decade has seen the rise and reign of the Cloud-computing paradigm. A wide-range of applications rely on its constant availability and its powerful capabilities.

However powerful, the Cloud-computing is not a “one size fits all” solution: for instance, latency-sensitive applications or applications in remote locations with poor connectivity cannot rely on it.

Edge computing has emerged as one of the ways to mitigate those challenges: it aims to provide computing, networking and storage fabrics at the border of the network, closer to the users, while providing the same elasticity as Cloud-computing. Other initiatives include: (i) Hybrid-Cloud [17], from the IT industry that aims at unifying the management of on-premises infrastructure from the Cloud operators; (ii) Fog computing [18], from the manufacturing industry, as an extension of the Industrial Internet of Things (IIoT); and (iii) Multi-Access Edge Computing (MEC) [19] as an extension of the Network Function Virtualization (NFV) [20] paradigm.

Hence, as illustrated in Figure 2, by regrouping the Edge and Cloud domains it is possible to create a continuum, spanning from the Things (e.g., end-user devices, smart-meters, cars, etc.) in the far-Edge, through ad-hoc deployments and small data-centers in the near-Edge, up to large data-centers in the Cloud.

From a top-down view, i.e. from the Cloud down to the Edge, it can be seen how the infrastructure evolves across the continuum: the Cloud comprises large homogeneous data-centers characterized by redundant and high-bandwidth networks, the near-Edge is decentralized and heterogeneous, in both connectivity and computing resources. For example, in the near-Edge, small and medium sized data-centers are usually located within the telecommunication operators networks and connected to a metropolitan fiber ring. By crossing the boundary between the operators and users networks, the infrastructure becomes even more heterogeneous in the far-Edge, with different sort of devices connected via (un)reliable wireless technologies and without any fixed topology.

Similarly, the workload evolves across the continuum: while in the Cloud and near-Edge it is possible to leverage the infrastructure to run heavy workloads, such as Artificial Intelligence or Machine Learning, this cannot be guaranteed in the far-Edge where the infrastructure may not support heavy loads on constrained or low power devices.

III. UNIFYING THE DATA-FABRIC ACROSS THE CONTINUUM

A. Zenoh

Zenoh is the combination of traditional publish/subscribe technologies, geo-distributed storage and computations in a unified and location transparent API. Zenoh was designed from the ground-up to deal with the heterogeneity of the Edge, ensuring time and space efficiency and robustness with respect
to asymmetric systems and different interaction models, such as battery-powered devices going into sleep mode.

More specifically, Zenoh provides: (i) efficient publish/subscribe with dynamic discovery, wire-level batching and various levels of reliability; (ii) geo-distributed storage for automatically storing and retrieving data, and (iii) well-defined semantics for querying and aggregating tasks.

One of the key concepts in Zenoh is that data is represented as \((\text{name}, \text{value})\) pairs: e.g., \((/\text{home/sensor/temperature}, 30)\). This \text{name} acts as both the entry point and the routing information needed to retrieve or store the associated value. Indeed, other than this \text{name}, no prior knowledge of the underlying infrastructure is required in order to interact with data: Zenoh takes care of delivering the content to the right place, as illustrated in Figure 3.

Zenoh also accepts registering computations that can be triggered via queries — called a \text{Queryable}. Enabling this behavior allows for an easy implementation of patterns such as \text{Remote Procedure Call} (RPC) and \text{Map-Reduce} operations.

Additionally, such capabilities do not come at the cost of wire-efficiency: Zenoh was designed to have a minimal wire overhead of 5 bytes [21]. This enables the support of constrained transport mechanisms, such as Low-Power WANs (LPWANs), Low-Power Wireless Personal Area Networks (LoWPANs), and Bluetooth Low Energy (BLE), as well as extremely resource-constrained devices, such as 8-bit microcontrollers.

Therefore, leveraging Zenoh, developers can focus on which data should be processed and how to process it, rather than having to worry on where they are located or how to retrieve them. This makes Zenoh the perfect data-fabric for the \text{Cloud-to-thing continuum} as it provides unified abstractions for data-in-use, data-at-rest, data-in-motion and computation.

B. Enhancing ERDOS with Zenoh

As ERDOS was limited to running on a single node, we investigated in [22] the integration of Zenoh as a transport to allow for ERDOS dataflows to be deployed on distributed systems – all of this, ideally without introducing performance degradation and if possible actually improving performance by leveraging Zenoh’s wire efficiency and incredible performance [23].

For the purpose of brevity, we summarize our key findings in what follows. In ERDOS, a computational unit is called a \text{node}. Nodes interact via two separate TCP connections, one for control messages and the other for data. By design, all
nodes in ERDOS must be interconnected, effectively creating two full-mesh networks.

This strategy has two main downsides: (i) the location (here, the IP address) of all nodes must be known before deploying the dataflow graph, and (ii) as not all connections are necessary, the resource allocation is not optimal.

Leveraging Zenoh, both downsides can be easily eliminated: Zenoh is location transparent and its publish-subscribe model allows for nodes to only create (through subscription) the connections actually needed. Additionally, as Zenoh supports several transport protocols, ERDOS nodes would no longer be limited to TCP connections, hence widening the number of potential deployment targets.

Figure 4 shows the difference in throughput between baseline ERDOS and the Zenoh-backed version. As we can see, the sustained throughput is, on average, doubled — the high variation in ERDOS’ results is due to the lack of any batching mechanism over TCP. While these results are promising, we could not exploit Zenoh’s full potential because of some fundamental design decisions taken in ERDOS.

Building on this experience, we set out to design an edge- and Zenoh-native dataflow framework: Zenoh Flow.

IV. ZENOH FLOW

The main objective of Zenoh Flow is to ease the development of Autonomous Robots and Vehicles applications and in general any application that requires cloud-to-thing data flows. As such, Zenoh Flow is designed to deliver the performance and efficiency required by control-oriented applications whilst supporting higher level abstractions required by some machine learning and AI data flows. An initial implementation of Zenoh Flow is published as open source.

Cloud-to-Thing compatible. Given the communication scenarios envisioned in V2X use cases [24], [25], having a framework that is designed to facilitate communication along the Cloud-to-Thing continuum is critical. Zenoh Flow abstracts these interactions through Zenoh’s unified API.

Consequently, Zenoh Flow is also able to seamlessly run a dataflow graph where the computational nodes are running on different machines. Similarly, migrating a computational node, performing load-balancing or adding redundancy become transparent procedures for an application developer.

Feature-rich. Considering the numerous operators and interactions that comprise an Autonomous Vehicle, Zenoh Flow offers a rich set of features to facilitate their creation and management.

For time-sensitive applications, developers can leverage automatic timestamps and end-to-end deadlines. Whenever data “arrive” on a Zenoh Flow dataflow, a timestamp is attached to them. This timestamp is accessible on each computational node and propagated throughout the graph. Developers can also define deadlines between computational nodes. If the deadline is missed, the last computational node at the end of the deadline is notified so that appropriate action be taken.

To provide more flexibility when designing a computational node, Zenoh Flow accepts input rules — its own adaptation of the firing rules. With input rules, a developer can specify under which conditions a computation can be triggered and what to do with the inputs. For instance, it is possible to trigger a computation with only a subset of inputs and keeping them for the next computation.

Zenoh Flow also supports loops in the dataflow graph. Loops are detrimental for all “feedback-based” algorithms and applications, and Autonomous Vehicles fall directly in that category with operators that rely on Machine Learning and AI to perform analysis or predictions.

Thanks to Zenoh’s geo-distributed storage, logging and replaying is effortlessly achieved. A storage with the desired database only has to subscribe to the correct resources — an information provided automatically by Zenoh Flow — in order to log an execution. To replay it, the storage simply provides the messages to the dataflow instead of the original source.

Reusable. One of the benefits deriving from Zenoh Flow’s
capability of abstracting communication is the possibility of developing operators that are independent of any underlying infrastructure or dataflow graph. As a result, operators can be composed and reused in any Zenoh Flow dataflow graph.

Attaining reusable operators offers many advantages: users could publish them in a library, this library could then be leveraged to develop applications faster and in turn increase user adoption.

**Declarative.** To target an even broader audience, a dataflow graph in Zenoh Flow is explicitly declared in a YAML file, as opposed to being implicit (such as with ROS or ROS2) \(^2\) or specified directly in a program.

Indeed, an explicit declaration in a human-readable format requires no knowledge of Zenoh Flow’s internals nor a strong programming experience. When combined with a library of reusable operators, a user can rapidly build a working dataflow, by connecting the operators. An example of this approach is given by Node-RED \(^3\).

Figure 5 illustrates an example of a dataflow graph implementing a computer vision application e.g. for obstacle detection. The graph comprehends a source, labeled as camera, three computing operators, and two sinks, labeled as steering wheel and remote log. One of the operators, labeled as tensor, is in fact a composition of multiple operators into an embedded dataflow graph. The other two operators, both labeled as opencv, are interconnected forming a loop for e.g. implementing an iterative object detection. We also included an end-to-end deadline between the tensor operator and the steering wheel sink, representing the maximum expected time to process any incoming video frame through the sub-graph.

To better visualize this example, we could consider Zenoh Flow to have deployed both the camera and tensor operators on a 3D camera residing in an autonomous vehicle and equipped with a Tensor Processing Unit (TPU). The opencv operators could be deployed on the on-board computing unit (OCU), which is equipped with a GPU to perform the necessary object detection. The steering wheel sink could run on a dedicated OCU for real-time mechanical control. The second sink, the remote log, could run on both the OCU and at the edge for remote logging or troubleshooting.

Hence, through this example, we can see how Zenoh Flow effectively abstracts the underlying communication along the Cloud-to-Thing continuum. This allows operators to run at the most suitable location without requiring any modifications, thus drastically increasing their reusability and composability.

V. CONCLUSIONS

Autonomous vehicles software is expected to significantly grow in complexity to deal with advanced V2X use cases and scenarios enabled by edge computing. In order to reduce this complexity in the software control logic, it becomes crucial to delegate communication related aspects to dedicated frameworks. By doing so, developers can solely focus on designing and implementing their application logic, e.g. autonomous driving based on AI.

With this goal in mind, in this article we introduced Zenoh Flow, a dataflow programming model based on Zenoh, an edge-native data-fabric. Zenoh Flow is a decentralized dataflow framework that is designed from the ground up to cope with requirements from end-to-end application across the Cloud-to-Thing continuum.

The capabilities provided by Zenoh Flow includes: (i) declarative applications, to ease development by explicitly defining the application graph, (ii) reusable operator, allowing operators sharing and encouraging code reuse, and (iii) distributed dataflow, enabling transparent communication between operators running within the continuum.

In future works, we plan to undergo a comprehensive experimental evaluation of Zenoh Flow’s performance as well as testing of automotive and robotics use cases.

---

\(^1\) Node-RED: https://nodered.org/

\(^2\) Host 1

\(^3\) Host 2

\(^4\) Host 3

\(^5\) Host 4

Fig. 5: Example of Zenoh Flow deployment
REFERENCES

[1] AUTOSAR, “Specification of rte software,” AUTOSAR Consortium, Specification Document Identification No 84, Release R19-11, 2019.

[2] ——, “Specification of communication management,” AUTOSAR Consortium, Specification Document Identification No 717, Release R19-11, 2019.

[3] OPEN Alliance, “100base-t1 system implementation specification,” OPEN Alliance Special Interest Group, Specification, 2017.

[4] Linux Foundation, “The automotive grade linux software defined connected car architecture,” Linux Foundation, Specification, 2018.

[5] H. Zhou, W. Xu, J. Chen, and W. Wang, “Evolutionary v2x technologies toward the internet of vehicles: Challenges and opportunities,” Proceedings of the IEEE, vol. 108, no. 2, pp. 308–323, 2020.

[6] W. M. Johnston, J. R. P. Hanna, and R. J. Millar, “Advances in dataflow programming languages,” ACM Comput. Surv., vol. 36, no. 1, pp. 1–34, 2004. [Online]. Available: https://doi.org/10.1145/1013208.1013209

[7] L. Cominardi, R. Andres, K. Hopkins, and F. Desbiens, “From devops to edgeops: A vision for edge computing,” Eclipse Foundation, Edge Native Working Group, Tech. Rep., April 2021.

[8] Z. Ul-Abdin and M. Yangt, “Dataflow programming of real-time radar signal processing on manycores,” in 2014 IEEE Global Conference on Signal and Information Processing (GlobalSIP), 2014, pp. 15–19.

[9] Z. Mingxin, T. Jun, and P. Yingning, “A dataflow mechanism for high speed radar signal processing,” in 2002 Region 10 Conference on Computers, Communications, Control and Power Engineering, TENCOM 02. Proceedings., vol. 1, 2002, pp. 359–362 vol.1.

[10] D. G. Murray, F. McSherry, R. Isaacs, M. Isard, P. Barham, and M. Abadi, “Naiad: A timely dataflow system,” in Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles, ser. SOSP ’13. New York, NY, USA: Association for Computing Machinery, 2013, p. 439–455. [Online]. Available: https://doi.org/10.1145/2517349.2522738

[11] R. Smits and H. Bruyninckx, “Composition of complex robot applications via data flow integration,” in 2011 IEEE International Conference on Robotics and Automation, 2011, pp. 5576–5580.

[12] Zenoh Project, “Zenoh overhead: a story from our community,” Eclipse Foundation, Website, 2021. [Online]. Available: https://zenoh.io/blog/2020-12-30-zenoh-overhead/

[13] Zenoh Project, “Zenoh performance: a stroll in rust async wonderland,” Eclipse Foundation, Website, 2021. [Online]. Available: https://zenoh.io/blog/2021-07-13-zenoh-performance-async/

[14] 5GAA, “C-v2x use cases and service level requirements volume i,” 5G Automotive Association, Website, 2020. [Online]. Available: https://5gaa.org/news/c-v2x-use-cases-and-service-level-requirements-volume-i/

[15] ——, “C-v2x use cases and service level requirements volume ii,” 5G Automotive Association, Website, 2020. [Online]. Available: https://5gaa.org/news/c-v2x-use-cases-and-service-level-requirements-volume-ii/

[16] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, “Ros: an open-source robot operating system,” ICRA workshop on open source software, vol. 3, no. 3.2, p. 5, 2009.

[17] Linux Foundation, “The automotive grade linux software defined connected car architecture,” Linux Foundation, Specification, 2018.

[18] OpenFog, “Fog Computing,” Institute of Electrical and Electronics Engineers (IEEE), Fog Computing and Networking Architecture Framework 1934, 2014.

[19] ETSI, “Multi-access Edge Computing (MEC); Terminology,” European Telecommunications Standards Institute (ETSI), Group Specification (GS) 001 v2.1.1, 2019.

[20] ——, “Network Functions Virtualisation (NFV); Use Cases,” European Telecommunications Standards Institute (ETSI), Group Report (GR) NFV 001 v1.2.1, 2017.

[21] 5G Automotive Association, “C-v2x use cases and service level requirements volume ii,” 5G Automotive Association, Website, 2020. [Online]. Available: https://5gaa.org/news/c-v2x-use-cases-and-service-level-requirements-volume-ii/

[22] G. Baldoni, G. Louhet, L. Cominardi, A. Corsaro, and Y. He, “Facilitating distributed data-flow programming with eclipse zenoh: The erdos case,” in Proceedings of the 1st Workshop on Serverless Mobile Networking for 6G Communications, ser. MobileServerless’21. New York, NY, USA: Association for Computing Machinery, 2021, p. 13–18. [Online]. Available: https://doi.org/10.1145/3469263.3469858