Store-Edge RippleStream: Versatile Infrastructure for IoT Data Transfer

Madhumita Bharde
Hewlett Packard Enterprise

Annmary Justine K
Hewlett Packard Enterprise

Suparna Bhattacharya
Hewlett Packard Enterprise

Dileep Deepa Shree
Hewlett Packard Enterprise

Abstract

Powerful edge compute frameworks address the issue of latency for IoT data processing at the edge. However, continuous application layer WAN streaming to core for consolidated deep analysis and learning consumes excessive bandwidth and becomes the bottleneck for responsiveness at the core. A state-of-the-art storage or hyperconverged system, on the other hand, advertises compelling in-built features like WAN efficient data protection and delta replication, global unified management, space and bandwidth saving through inline data compression and deduplication. Traditional storage semantics and services, however, are built for data at rest while edge analytics prioritizes responsiveness by processing and moving streaming data at an application layer. In this paper, we propose to enable streaming of IoT data transparently through storage replication. Based on this foundation, we further present light-weight storage plugins to reduce IoT data transfer by detecting and translating semantic redundancies to a deduplication friendly form. Our early results demonstrate that (a) leveraging storage to take responsibility of streaming data in an application-consistent way results in efficient data transfer (b) real-world IoT time-series datasets exhibit a high degree of similarity which can be detected to reduce data transfer from edge (c) video streams for autonomous cars, the transfer of which cannot be reduced enough using traditional video compression or storage deduplication techniques, have significant semantic redundancy. Collectively, advancing research in this direction paves the way to enhance the versatility of state-of-the-art infrastructure for optimized edge computing.

1 Introduction

Traditional Internet of Things (IoT) analytics solutions tradeoff speed and depth - they deliver immediate "time-to-insight" or delayed "depth-of-insight"[8]. Edge computing alleviates this tradeoff - because both speed and depth are desirable, thus driving more compute and storage capabilities closer to the data's origin ("Shift-left"). However, the IoT ecosystem presents multiple challenges for traditional compute and storage infrastructural paradigms: (1) The scale and diversity of deployment scenarios necessitates managing intermittently connected geographically dispersed edge sites with simple interfaces, while allowing flexibility in analytics processing as applications evolve. (2) The high volume and high velocity of data generation requires novel approaches to ***deal with continuous streaming data and manage it based on its semantic value*** since hot/cold are no longer good indicators of value. (3) Analytics can be performed in devices at the edge, in gateways, and in data centers, creating new data and insights along the way to manage, store and transfer efficiently. (4) IoT sensor data is usually time-series, video streams, geo-spatial or asset data. Conventional ***storage de-duplication fails to be effective*** for such data because of the presence of embedded timestamps and statistical variation, resulting in higher storage and bandwidth usage, besides redundant processing.

One approach to address such challenges would be to build dedicated edge solutions comprising a specialized infrastructure stack packaged with frameworks and application components that implement a variety of data reduction, filtering, function distribution, streaming analytics and distributed learning optimization techniques. However, specialization can become limiting for edge platforms which may potentially evolve to run diverse workloads and varied analytics pipelines.What if we could repurpose the existing software defined infrastructure paradigm (with appealing features like unified manageability, storage, WAN optimization) for edge? We believe that the preferred approach to achieve this should embed *generic capabilities in compute and storage platforms so that they can learn about the data; thereby empowering future solutions with enough versatility to handle diversity of IoT data in an application and location transparent way.*

Contributions: First, our approach unifies management of data at rest and data in motion at the storage layer (section 2), by linking streaming platform(s) with asynchronous storage replication for data movement from edge to core. To achieve this, we replace chatty application layer edge to core streaming data transfers by transparently riding on WAN optimized application-consistent storage replication and local streaming transfers. The *processing layer (applications) built with these frameworks remains unchanged*. Second, once the storage layer is plugged in to optimize data transfer, *"insights" about nature of streaming IoT data are incorporated to further optimize the transfer*. This is illustrated by presenting the relative benefits of two similarity detection plugins that identify semantic redundancies in time-series (section 3) and video streams (section 3.2) to leverage storage layer data deduplication and WAN efficient data transfer. Reducing data transfer by identifying semantic redundancies and enabling compute mobility can help ensure that IoT processing (including data analytics, data movement, compute shift and data management) is performed optimally across all edge sites and the core.

2 Storage Replication for Streaming

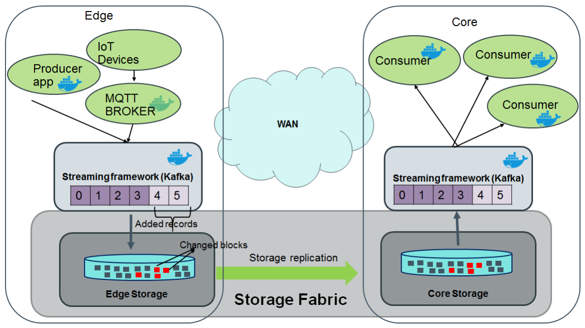


Figure 1: Transparent replication of stored data streams

In a typical edge to core computing pipeline, the data is transferred using application frameworks (e.g. Apache NiFi[3], Kafka[2], Storm[4] etc.) to core. These frameworks also persist the data in storage. We propose to transparently replace streaming of data through application frameworks across WAN with optimized edge to core storage replication. Instead of data streams getting transported as payload in the application layer, changed blocks of the log files used to persist these data streams are replicated at specified intervals by using the asynchronous replication protocol supported by the underlying storage. The replication protocol would stream these delta blocks from the source edges to the destination datacenter (Figure 1). Applications designed on the data streams need not change. Instead of connecting to a server in the remote datacenter, the application connects to a local server to fetch data. A restart of the framework will force it to reevaluate its offsets and recognize the newly added logs. Restart of the framework can be done unobtrusively by restarting the container hosting the framework. Consumer applications communicating with the framework need not restart. Thus the applications are served the same data, though at a maximum lag defined by the replication interval. Application workflows which can work on near real-time data having predictable and consistent amount of latency can take advantage of the WAN optimizations offered by this approach.

Application consistency: The application frameworks used for streaming rely on filesystem fsync for the consistency guarantee of the streams and can be configured with a checkpoint interval for their logs. Storage replication is triggered after the file is checkpointed thereby ensuring that the streams are application consistent. When the filesystem receives fsync system call, all the pending writes are persisted to the disk and further writes are quiesced, the file is snapshotted and compressed unique changed blocks are replicated by taking advantage of built-in deduplication, compression and delta replication features of the underlying storage layer. Further, frameworks such as Kafka incorporate mechanisms such as CRC32 checksums appended to each record of the data streams which can be used to verify the consistency of records at the destination.

Evaluation: We simulated two identical IoT environments consisting of traffic sensors (Refer Figure 4) sending their data to a Kafka broker at edge. This data was forwarded to applications at core which analyzed data from multiple edges. In the storage streaming setup, Kafka logs of processed streams from each edge server were replicated to the log folder of Kafka broker at the core using rsync[12] (a utility for efficiently transferring and synchronizing files across computer systems), whereas in the application streaming setup, streams were remotely consumed at Core from edges using Kafka. Although we have used rsync for our experiments, typically any efficient built-in asynchronous storage replication mechanism could be leveraged. The various factors taken into consideration for characterizing the benefits were payload size, frequency of messages, number of sensors, number of consumers, streaming interval and compression. With very small payload, application layer streaming performs better than the storage streaming, but with increasing number of Kafka consumers sharing same streams, storage streaming fares much better. With increasing workload (Figure 2) (increase in payload, frequency and number of sensors), storage streaming performed significantly better than application streaming. The exact benefits obtained could vary depending on the type of the dataset. A comparison of bandwidth used with and without rsync compression showed significant bandwidth savings due to compression. The results also show that increasing streaming interval decreases the bandwidth needed for storage streaming as larger time

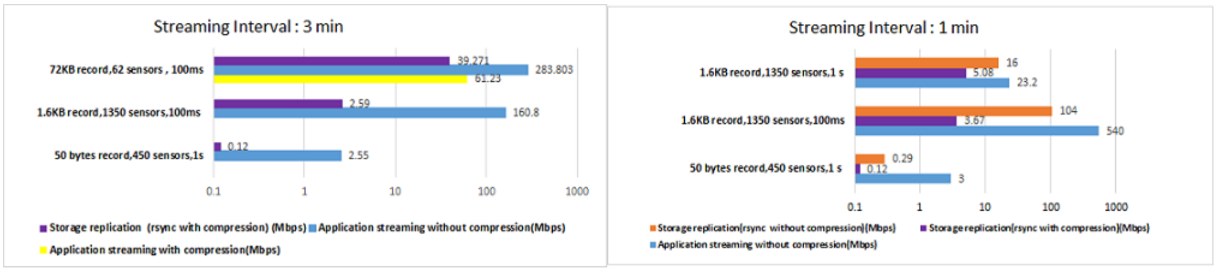


Figure 2: Bandwidth consumption (log scale) of Storage streaming (rsync) vs Application streaming (Kafka) measured using tcpdump [13] and analysed with wireshark[15]

windows result in better compression ratios. Overall, the combination of Kafka compression coupled with Rsync compression is the sweet spot for large amount of data transfer through WAN.

Discussion and Related Work: Kafka Mirror Maker[9] helps to maintain a replica of an existing Kafka cluster in a different datacenter. This would solve the problem of multiple consumers consuming the same records, resulting in fetching the stream multiple times across the WAN. WAN optimizers are dedicated appliances placed at WAN endpoints behind the router. Both these solutions introduce added complexity and cost to the overall ecosystem. We leverage the in-built strengths in storage systems like deduplication and compression. Any streaming framework that uses storage to persist/log state of its consumers/producers and maintains a logical organization (e.g. Kafka has topic-wise directories in log folder, Apache Bookkeeper[1] has separate index files managing data for each ledger) can use replication as a way of streaming data. Streaming through storage, however, poses challenges in application consistency and seamless consumption of the replicated streams by the framework. We take care of application consistency by relying on the periodic flushing of the frameworks to the filesystem at the source and by the consistency checks done by the frameworks at the destination. We are also investigating a possibility of thin framework plugin to make replicated records immediately consumable at the destination without requiring a broker restart.

3 Similarity Awareness for Store-Edge

There are two kinds of applications at the core: (a) applications that periodically derive summary insights from incoming batches of input data (b) deep learning applications that learn updated models/rules to be communicated to edge. We detect similarities between streams of data using similarity service plugin at the edge for these applications. Similarity detection happens in a transparent manner without impacting the analytics workflow. The light-weight similarity service plugin depends on the type of data and augments conventional block deduplication to dedupe semantically similar data.

3.1 Time-series Data

Typical time-series IoT data often contain similar trend and cyclical components that can generate semantically

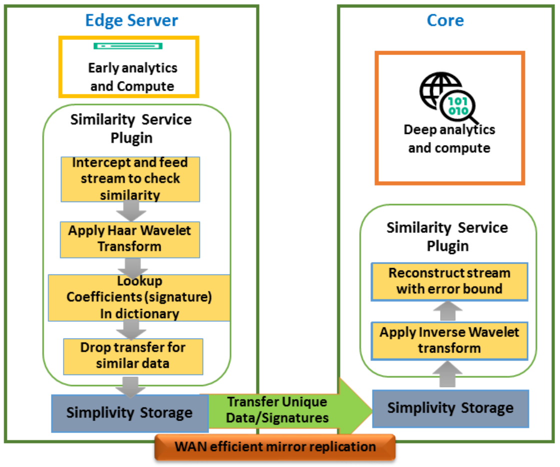

Edge Server
Early analytics and Compute
Similarity Service Plugin
Intercept and feed stream to check similarity
Apply Haar Wavelet Transform
Lookup Coefficients (signature) In dictionary
Drop transfer for similar data
Simplivity Storage
Transfer Unique Data/Signatures
WAN efficient mirror replication
Core
Deep analytics and compute
Similarity Service Plugin
Reconstruct stream with error bound
Apply Inverse Wavelet transform
Simplivity Storage

Figure 3: Similarity Plugin for Time-series datasets

similar analytical outcomes. Traditional storage deduplication does not detect such duplicates because it compares raw bytes. We leverage the method SEeSAW[20] to detect similarity early in the analytics workflow, saving both edge storage and edge to core data transfers.

Figure 3 shows how we extend traditional infrastructure to detect similarities between streams using a similarity service plugin at the edge. We use Discrete Wavelet Transform (DWT) to convert time-series data to a smaller set of representative coefficients (signatures) for each micro-batch ("streamlet"). DWT signatures are stored in an efficient index (e.g. by reusing storage layer de-duplication hash). When an incoming streamlet resembles a previously seen streamlet (signature match), it gets linked to the previous streamlet and its transfer to core is dropped (or delayed). This is achieved by riding on mechanisms already in place for de-duplication of storage thus avoiding duplicate data transfer. Coupled with storage streaming (section 2), this approach provides extra benefit by sending only semantically unique data across WAN to the core (as relevant for analytics results). We also plan to explore the possibility of using a transformation plugin that distributes only the signature instead of the raw stream from edge to core. The similarity transformation service plugin at the core can reconstruct the stream (with a tolerable error bound) by performing an inverse transform (IDWT) on a smaller subset of these signatures.

Evaluation and Correctness: We created a simulated IoT environment for CityPulse traffic dataset [6] for eval-

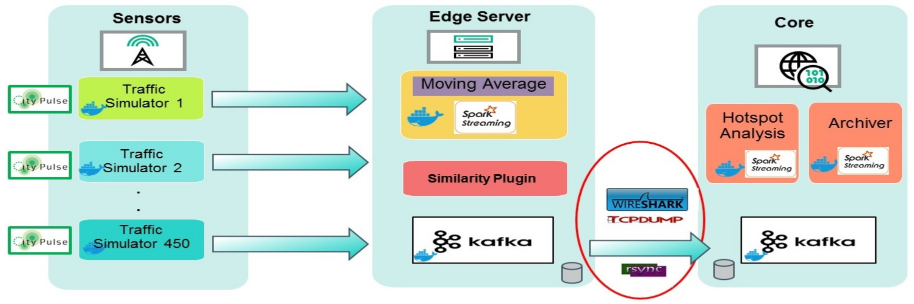


Figure 4: Example Experimental Setup

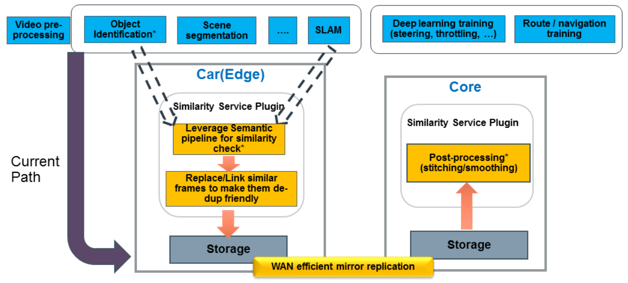
uating the correctness of DWT based similarity detection for time-series datasets. As shown in Figure 4, we use a containerized simulator for traffic dataset for all 450 sensors and replay it at 1 second interval. These producers communicate to the Kafka broker at the edge, where moving average streams of vehicle count for each traffic signal are generated. The similarity plugin achieves savings in storage and transfer by detecting similarity between streamlets before they are stored and replicated to the core. At the core, we run a Hotspot Identification Spark streaming application which operates on incoming streamlets and identifies the top 5 busiest signals for each 3-minute period. We observe 95% reduction in overall data transferred from edge to the core in this setup. To verify correctness, two identical synchronized setups were used with and without the similarity plugin. The top 5 busiest signals identified by the hotspot application on both setups were the same in every iteration. Table 1 shows ***significant data reduction potential in real world IoT datasets*** when using first 4 coefficients of DWT as the signature for similarity detection.

Dataset	Dataset Size	Saving
GREEND[7]	10GB	84.05%
CityPulse Pollution[6]	570 MB	8.28%
CityPulse Traffic[6]	1.35GB	97.71%

Table 1: Similarity Detection for time-series datasets

Discussion and Related Work: Papageorgiou et. al.[21] automate the switching between different data handling algorithms at the network edge, including an analysis of adjusted data reduction methods primarily for time series data. Managing the tradeoff between data reduction and accuracy across different scenarios is an interesting challenge. This may be approached in several ways: (1) adjusting the metrics (streamlet window size, number of DWT coefficients, distance threshold, desired accuracy) via application guidance or a core-to-edge feedback driven adaptive configuration mechanism as proposed in SEeSAW[20] (2) maintaining both exact and semantically de-duplicated views at different priority levels by differentiated storage and transfer treatment to exact and semantically similar data (3) shifting analytics stages that are sensitive to exact data (e.g. anomaly detection) to the edge before semantic redundancy detection (4) expending extra computation at the core to apply needed corrective adjustments.

3.2 Autonomous Driving


Video pre-processing
Object Identification*
Scene segmentation
....
SLAM
Deep learning training (steering, throttling, ...)
Route / navigation training
Current Path
Car(Edge)
Similarity Service Plugin
Leverage Semantic pipeline for similarity check*
Replace/Link similar frames to make them de-dup friendly
Storage
WAN efficient mirror replication
Core
Similarity Service Plugin
Post-processing* (stitching/smoothing)
Storage
Figure 5: Similarity Plugin for Autonomous cars

Autonomous cars generate huge amounts of video data[10]. Poor network conditions (e.g. WiFi/4G etc.) make it impractical for all of this data to be sent to core (or cloud) for deep learning and training. Video compression requires temporal context to reduce the data size and is thus ineffective for detecting semantic or contextual similarity spread across time (e.g. consider video streams generated during home to office drive every morning). Storage deduplication works beyond temporal context and can be used for WAN efficient streaming as described in section 2. However, storage deduplication and compression are not effective on already compressed video data formats (e.g. mp4, mpeg).

There is an already existing video processing pipeline at the edge for taking real time driving decisions. This involves techniques like object detection, semantic segmentation, SLAM (simultaneous localization and mapping) etc. We propose to leverage it by introducing a native thin semantic deduplication layer for efficient data transfer from autonomous cars. Similar to section 3, we create a signature for each video frame and maintain a dictionary at the edge for similarity detection. We use a combination of Mobilenet Caffe model[5] for object identification and GIST descriptors[19] to detect semantic similarity between an incoming video frame and previously encountered frames. If a match is found, we replace new frame with the matching frame. With this approach, semantically similar frames are made exactly similar before they are stored; so that block deduplication and incremental transfer is efficient. If the entire video is needed to be replayed at core, similarity plugin for post-processing (stitching or smoothing) is used. In other use-cases (like training/deep learning), the representative video presented to applications at core could be shorter as it captures the most informative bits of video from model building perspective.

Dataset	Size	Saving (Frames)	Gzip-Saving	Saving (Video)
Kitti[17]	6.3GB	42%	1.5%	13%
UMich Downtown[14]	78GB	66%	26%	60%
UMich Ford[14]	119GB	66%	27%	51%
CCSAD*[18]	8MB	38%	17%	23%

Table 2: Similarity Detection for Autonomous Cars (*sample video used for visual verification)

Correctness and Results: We were able to reduce CCSAD sample video[18] to 62% its size. Visual validation showed little difference (we observed minor jerkiness) between original and processed video, even after replacing 38% of the frames. As mentioned before, in use-cases where application at core learns a model, only incremental bits can be moved to core. We tested across night and day trips[11] of the same route and achieved 41% reduction for object identification model at core. We needed small video clips for visual validation, however the reduction usually improves for more amount of data, so bigger datasets also mean better semantic similarity. Table 2 shows how we performed on some real-world datasets for autonomous cars. Early results show the relative advantage and future potential of our approach as compared to tar-gzip based compression.

Discussion and Related Work: Optimizing self-driving car video transfer is an upcoming area of fast-paced innovation. Traditional video compression techniques optimize human perception quality and require decompression of the video frames at the time of use, thus increasing the size in the process [22]. Techniques applied for compressed sensing of surveillance videos [16] can potentially be leveraged for autonomous driving. We complement such techniques by translating semantic redundancies to leverage existing de-duplication and delta replication. The proposed approach provides extra reduction to video streams transferred through storage replication. Most video processing pipelines at edge tend to work on raw camera feed before sending the video in codec format to edge. It is at this stage that we intercept or leverage the pipeline which results in native and organic reduction to encoded video before it is stored or sent. We are also investigating an approach to identify keyframes and transform video snippets to a signature so that replacement and block deduplication happens at a snippet granularity rather than frame granularity. Thus, the resultant video stream after replacing semantically similar frames/snippets is guaranteed to be lower in size than what it originally would have been. As per our tests (Table 3, column 4) , the reduction % in terms of video size comparison is marginally smaller (but still better than gzip saving) than reduction % for input image datasets.

Our experience showed a low overhead on object identification but relatively higher overhead for GIST. Good news is that with two-stage semantic deduplication approach, we don't propose doing GIST comparison over the entire sample space. We narrow down candidates for near-exact comparison (like GIST) by performing a light-weight first stage signature-comparison (object identification). More importantly, these two are just the representatives but the right thing would be to leverage the edge processing pipeline itself to identify candidate analytics stages for semantic deduplication. If this processing is native or inherent to the pipeline, it is not really an overhead.

4 Conclusion and Future Work

We presented a two-fold vision to empower general purpose storage infrastructure to be operated at the edge for efficient data transfer. First, facilitating streaming of IoT data transparently through storage leverages (a) streaming frameworks' ability to separate streams by application determined context (e.g. topics) at the time of persistence (b) storage layers' ability to do selective data replication for application-consistent transfer. This opens up possibilities to build more intelligence in the storage layer. Second, we proposed thin storage plugins at the edge for time-series and video data so that semantic redundancies can be detected before its storage or transfer. These plugins essentially translate semantically similar data to exactly similar so that storage deduplication, compression and replication becomes effective on streams of IoT data. We demonstrated that this translation doesn't affect the overall essence and pattern of the original data; thereby maintaining the accuracy of application at core, with much lesser bandwidth cost. These early findings open up fresh questions about the best way to leverage end to end infrastructure features for an efficient edge-to-core pipeline. How should computation be distributed to enable effective detection of semantic redundancies? How should framework and infrastructure layers be co-designed for maximum versatility? We believe there's a place for general purpose optimizations even in a data-diverse IOT world as we continue investigations towards versatile and all-inclusive approaches to enhance infrastructure at the edge.

Acknowledgements: We thank Peter Corbett for his support and guidance in investigating the notion of unifying management of data at rest and data in motion for hyperconverged data fabric in edge computing context.

References

[1] Apache bookkeeper. https://bookkeeper.apache.org/docs/r4.0.0/bookkeeperOverview.html.

[2] Apache kafka. https://kafka.apache.org/.

[3] Apache nifi. https://nifi.apache.org/.

[4] Apache storm. http://storm.apache.org/index.html.

[5] Caffe implementation of google mobilenet ssd. https://github.com/chuanqi305/MobileNet-SSD.

[6] Citypulse eu fp7 project. http://iot.ee.surrey.ac.uk:8080/datasets.html.

[7] Greend, an energy dataset. https://sourceforge.net/projects/greend/.

[8] Industrial internet consortium. http://blog.iiconsortium.org/2015/07/the-7-principles-of-the-internet-of-things-iot.htm.

[9] Kafka mirror maker. https://docs.confluent.io/current/multi-dc/mirrormaker.html.

[10] Networkworld. https://www.networkworld.com/article/3147892/internet/one-autonomous-car-will-use-4000-gb-of-dataday.html.

[11] Night and day clips. https://www.youtube.com/watch?v=7BjNbkONCFw and https://www.youtube.com/watch?v=ev5nddpQQ9I.

[12] Rsync- linux man page. https://linux.die.net/man/1/rsync.

[13] Tcpdump- linux man page. https://linux.die.net/man/8/tcpdump.

[14] University of michigan ford campus vision and lidar dataset. http://robots.engin.umich.edu/SoftwareData/Ford.

[15] Wireshark. https://www.wireshark.org/.

[16] Chen, C., Cai, J., Lin, W., and Shi, G. Surveillance video coding via low-rank and sparse decomposition. In *Proceedings of the 20th ACM International Conference on Multimedia* (New York, NY, USA, 2012), MM '12, ACM, pp. 713–716.

[17] Geiger Andreas, Philip Lenz, R. U. Are we ready for autonomous driving? the kitti vision benchmark suite. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2012*.

[18] Guzman, R., Hayet, J.-B., and Klette, R. Towards ubiquitous autonomous driving: The ccsad dataset. In *Conference on Computer Analysis of Images and Patterns (CAIP)* (2015).

[19] Ivan Sikiric, Karla Brkic, S. S. Classifying traffic scenes using the gist image descriptor. *CoRR abs/1310.0316* (2013).

[20] Kannan, K., Bhattacharya, S., Kumar, R., Murugan, M., and Voigt, D. Seesaw - similarity exploiting storage for accelerating analytics workflows. In *Proceedings of HotStorage '16* (2016).

[21] Papageorgiou, A., Cheng, B., and Kovacs, E. Real-time data reduction at the network edge of internet-of-things systems. In *Proceedings of the 2015 11th International Conference on Network and Service Management (CNSM)* (Washington, DC, USA, 2015), CNSM '15, IEEE Computer Society, pp. 284–291.

[22] Seif Heiko G., X. H. Autonomous driving in the icityhd maps as a key challenge of the automotive industry. In *Engineering 2, no. 2 (2016): 159-162.*