

# In-Network Compression for Accelerating IoT Analytics at Scale

Rafael Oliveira<sup>1</sup> and Ada Gavrilovska<sup>1</sup>

<sup>1</sup>Georgia Institute of Technology

**Abstract**—To enable the Internet of Things (IoT) to scale at the level of next generation smart cities and grids, there is a need for cost-effective infrastructure for hosting IoT analytics applications. Offload and acceleration via SmartNICs have been shown to provide benefits to these workloads. However, even with offload, long-term analysis on IoT data still needs to operate on massive number of device updates, often in the form of small messages. Despite offloading, the ingestion of these updates continues to present server bottlenecks. In this paper, we present domain-specific compression and batching engines, that leverage the unique properties of IoT messages to reduce the load on analytics servers and improve their scalability. Using a prototype system based on the InnovaFlex programmable SmartNICs, and several representative IoT benchmarks, we demonstrate that the combination of these techniques achieves up to  $14.5\times$  improvement in sustained throughput rates compared to a system without SmartNIC offload, and up to  $7\times$  improvement over existing offload approaches.

## I. INTRODUCTION

The number of IoT devices and applications is growing exponentially, spanning diverse industry verticals, from different wearables and smart city applications, distributed agriculture and smart infrastructure, to ultra-low-latency industrial automation and mission-critical control [1]. In response of this trend, commercial cloud (e.g., Amazon AWS [5], Google [15], Microsoft [6]) and network (e.g., Deutsche Telecom, SK Telecom, China Mobile, AT&T) operators, are increasingly offering IoT-based hosting services, ranging from programming APIs to fully-operated infrastructure for hosting third-party IoT applications. The scalability and efficiency of this infrastructure tier, is, thus, relevant to a large number of stakeholders. This is expected to be further exacerbated by advancements in 5G technologies, where ultra-reliable and ultra-low latency capabilities of commercial and private 5G networks are enabling new types of IoT ecosystems with even more stringent end-to-end performance requirements [40].

To understand the requirements for this server infrastructure, we first look at the unique characteristics of IoT. These applications are commonly characterized by massive arrays of heterogeneous sensors, generating periodic updates in the form of small messages. IoT devices are limited in their computational, energy, and communication resources, so applications rely on remote servers, in the cloud or in future edge datacenters [40], to aggregate and process sensor updates. The applications' end-to-end performance requirements include high sustained throughput for the aggregate sensor updates, ability to tolerate unexpected bursts, and low and predictable processing

latency for actuation or notification events in response to online analyses of sensor data.

Offload and acceleration to emerging programmable SmartNICs [19], [31]–[33] are techniques which have been proven effective for delivering low and predictable latency and high throughput for request processing across many domains [13], [20], [24], [28], [29], including for IoT [13], [21]. However, while these prior solutions make it possible to configure the NIC's compute resources for in-network packet processing of IoT messages, they do not consider that IoT applications include long term data aggregation, analysis and modelling. The compute and storage requirements of these operations, coupled with the complexity of their legacy software stacks, implies that this functionality remains executed on general-purpose host CPUs. As a result, even with SmartNIC offload and acceleration, the scalability of the IoT analytics server systems is bottlenecked by the host's packet processing capabilities, due to known limitations of general-purpose CPUs and network stacks to deal with large number of small messages [23] [22].

By analyzing several real-world IoT applications, we make the following observation. First, IoT applications commonly consist of distinct *critical-path* and *long-term analytics* components. By decomposing applications into their two components, it is possible to extract efficiency through *in-network offload and acceleration of the critical-path operations*. Performing the acceleration in-network helps achieve low and predictable latency. Second, IoT applications and their messages are built around the *topic* construct, which provides semantic information regarding the message types and formats to the underlying processing runtime. This further allows for *application-specific batching and compression* to be deployed, but *outside of the critical path*, thereby not impacting critical-path latencies, while scaling the sustainable throughput of the analytics path.

Motivated by these observations, we design **Complex** – a set of domain-specific batching and compression engines specialized for SmartNIC offload and acceleration of IoT. This new topic-aware batching and compression can be integrated with existing SmartNIC offload solutions to address the scalability limitations of the SmartNIC-host application interface, and to provide significant benefits to IoT analytics. The outcome is a server framework that achieves low and predictable latency for accelerated critical-path IoT operations, with increased throughput scalability of the long-term modeling and aggregation tasks executed via general-purpose CPUs.

The engines are integrated with a SmartNIC prototype system

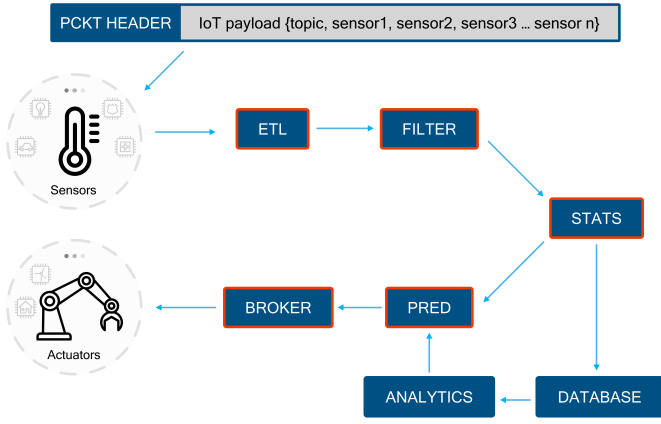


Fig. 1. A Common IoT application dataflow.

based on the Mellanox InnoVaFlex [32] smart NICs, equipped with a Xilinx XCKU060 FPGA. Using four representative IoT benchmarks [3], we show that the use of application-specific batching and compression allow Comprex to improve the stable throughput (messages processed per second) by  $11\times$  compared to just offload approaches, while not degrading latency for critical-path operations. For a fixed load, it reduces the host's per-core CPU utilization by up to  $8\times$ , freeing up those resources to scale the analytics components or to host more applications. The benefits persist even when co-running all four IoT benchmarks to emulate multi-tenant setting.

## II. IoT ACCELERATION OPPORTUNITIES AND CHALLENGES

### A. IoT Workload Characteristics

While the term IoT refers to a broad range of applications, most IoT applications are characterized by ingesting large volumes of sensor data, analyzing it, and delivering updates to one or more subscribers. Sensor data processing involves a common set of steps, shown in Figure 1. These form a critical path which often comes with stringent real-time latency constraints varying from  $250\ \mu\text{s}$  to  $100\ \text{ms}$  [14], and a long-term analytics path, which operates at much longer time scales.

Typical sensors generate small updates of 10s to 1000s of bytes, with existing IoT protocols designed for 20-1600 byte message sizes [45], [49], [53].

In IoT, data message rates range from hundreds per day for air quality sensors, to tens of thousands per second for industrial IoT, with very diverse frequency distributions [3], [38] even within the same application. The massive number of devices, coupled with the real-time latency requirements, impose a big challenge on running these applications at a single location (cloud or edge). For instance, a single natural gas facility houses over 200k devices that monitor critical operations 24/7 [43] and a single manufacturing line can produce as much as 10 million measurements a day [11].

**Motivation for in-network acceleration.** Handling these message rates and processing latency requirements with general-purpose server systems is not sustainable, particularly given the small message sizes common in IoT. Realizing a scalable

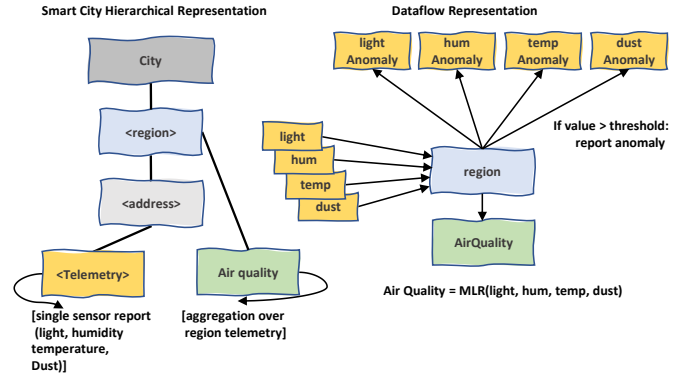


Fig. 2. A hierarchical representation of the City topics. A dataflow representation of /CITY/REGION/ADDRESS/TELEMETRY and /CITY/REGION/ADDRESS/AIR\_QUALITY.

and performant infrastructure for IoT will therefore have to seek efficiency through in-network acceleration.

### B. IoT Modules and Kernels

IoT applications are very modular. They are frequently expressed in a static pipeline of high level compute kernels that: 1) parse data coming from sensors in a byte format to server-side friendly format like JSON (ETL in Figure 1); 2) filter data by range of max-min values, token, device id, etc., so as to eliminate outliers, or to enrich data with filter models such as Kalman Filter, Bloom Filter or interpolation models that can fill in missing values (FILTER); 3) perform statistical analytics such as average, max, min, count, aggregation (STATS); 4) use trained machine learning models to generate actions or to predict future events based on current measurements (PRED); 5) and disseminate these as updates to different types of subscribers (BROKER); 6) train machine learning models to be used in the predictive analytics kernel (ANALYTICS); and 7) store data in persistent memory for future analysis (DATABASE).

These compute kernels are commonly connected in a dataflow format using popular cloud services such as Amazon Kinesis and Lambda functions [5] or Google Cloud functions [15]. The first five functions form a closed-loop flow capable of detecting and responding to events, near real-time, without any human intervention. In addition, the operations represented in these classes of functionality have already been shown amenable to offload and acceleration [13], [21], [24]. The last two functions operate across aggregates of IoT updates over different spatial and temporal scales, and rely on conventional compute and storage infrastructure on general-purpose servers.

### C. Topic, Publisher and Subscriber

**Content of an IoT message.** An IoT device serves as a publisher of messages with sensor values, or as a subscriber to messages with actuation commands. A message has the format shown in Figure 1, and consists of a network header and IoT-specific payload. The IoT payload is structured based on one of the common IoT protocols, such as MQTT [16] and CoAP [51], and

TABLE I  
IoT APPLICATION WORKLOAD AND EVALUATED KERNELS. MLR IS MULTI-VARIABLE LINEAR REGRESSION.

App Name	Topic size [B]	Message size [B]	Number of Topics	Compute Kernels in Critical Path				Number of Fields
CITY	35	123	10,700	Raw, ASCII	Range Filter, Bloom Filter	Kalman, Avg, Max, Min, Count	Decision Tree Air Quality	10
TAXI	47	144	50,000	Raw, ASCII	Range Filter, Bloom Filter	Kalman, Avg, Max, Min, Count	MLR fare amount	27
SmartGRID	30	80	10,217,500	Raw, ASCII	Range Filter, String Match	Kalman, Avg, Max, Min, Count	Threshold Consumption	3
FIT	32	173	10,000	Raw, ASCII	Range Filter, String Match	Kalman, Avg, Max, Min, Count	MLR ecg	17

it includes a number of fields such as device identifiers (device name, user ID), sensor types (e.g., temperature, luminosity, etc., depending on the sensors available on the device) and sensor data (represented in a well-defined format). An application is free to specify how these fields are interpreted: as part of the metadata used to classify and route the message – i.e., its topic – or as the message data payload consisting of (the remaining) multiple fields.

*The topic structure.* Topic is a key abstraction that connects publishers (IoT devices generating data) and subscribers (control processes, alert systems, etc., consuming IoT updates), and is used by the message broker component responsible for distributing data. It is commonly implemented as a string that expresses the hierarchical relationship between the application components. Figure 2.A illustrates how the topic string */smartcity/region/block* expresses the hierarchical relationship of *devices* reporting telemetry at an *address* within a *region* of a *city*. Figure 2.B Illustrates how sensor data is processed separately, to report anomalous behavior, but is aggregated in the parent */city/region* topic in order to compute the air quality of the region.

*Flow and data aggregation.* In IoT, a flow expresses a sequence of execution and logical interactions between devices and compute kernels that process and transform data in the flow, typically implemented as microservices [44]. topics can be used purely to match IoT messages to a flow. For example, the topic “connectedcar/telemetry/vehicleId” in AWS Connected Vehicle solution [48] is used for mapping car sensors to a collection of cloud functions such as DynamoDB, Kinesis Analytics, etc., to perform data aggregation. Data aggregation can happen within and across flows. In City, the average temperature at an address involves aggregation over all the reported temperature updates from a sensor within the same flow, but an air quality of a region invokes a classification function over the aggregated values of multiple different sensors from multiple flows.

### III. EXISTING SMARTNIC ACCELERATION FOR IoT

**In-network acceleration approaches.** Next we consider existing state-of-the-art in-network accelerators. Existing in-network accelerators can be grouped into three different categories: Pipeline-of-Offloads NICs (NICA) [13], Manycore NICs [33], and re-configurable match-action (RMT) NICs (PANIC) [21]. Figure 3 illustrates the main components present in an in-network accelerator. As packets arrive in the accelerator,

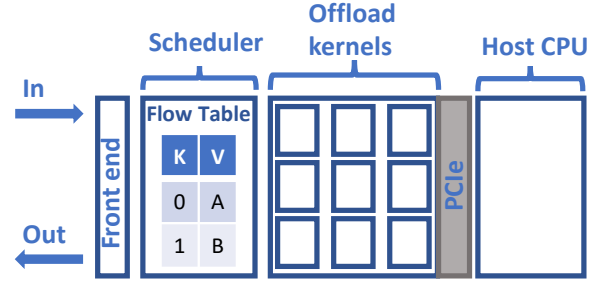


Fig. 3. Key components of a generic representation of in-network accelerators. A flow table maps packets to different combinations of offload kernels.

they are mapped to an entry in the flow table and scheduled across different offload kernels. The flow table is used for storing data regarding how to process the incoming packet. The difference between in-network accelerators can be simplified into how these main components operate in relation to one another. NICA groups offload kernels into a static pipeline that flows can be mapped to. PANIC’s offload kernels are connected via cross-bar with bi-directional communication, allowing pipelines of kernels to be dynamically created.

Oil Anomaly Detection	In-network accell	CPU
Clock Frequency	216Mhz	2.9GHz
Throughput	38.2M m/s	433K m/s
Latency	64us	840us

TABLE II  
MICRO-BENCHMARK THROUGHPUT COMPARISON BETWEEN HARDWARE AND SOFTWARE OF A SIMPLE OIL ANOMALY DETECTION IoT DATAFLOW.

**Quantifying the opportunity.** The previous section suggests the opportunities to achieve low and predictable latency via *in-network offload and acceleration of the critical-path operations*. Table II illustrates the performance comparison for the critical path part of a vehicle oil temperature anomaly detection. In this simple IoT application, vehicles report their oil temperature in a 6-byte string attached to a 64 byte UDP packet. The micro-benchmark uses several kernels: string-to-float, threshold filter, Kalman Filter, average and threshold control that generates a notification every time a measurement is a certain value above average. We implement and synthesize the compute

kernels using Vivado HLS pipeline directive and deploy it on a Mellanox® InnoVa™ Flex 4 Lx EN (1st gen.) SmartNIC, equipped with a Xilinx XCKU060 FPGA that functions as a bump-in-the-wire accelerator. The software implementation of the kernels uses the same C++ code used in Vivado HLS except for the HLS directives. The CPU is an Intel i5 (single core). On the client side we used an AMD threadripper to generate the IoT packets. The in-network accelerator has a throughput 88x higher and a round-trip latency 13x smaller than a single CPU core, making in-network acceleration a compelling approach for handling the *critical-path* of IoT applications.

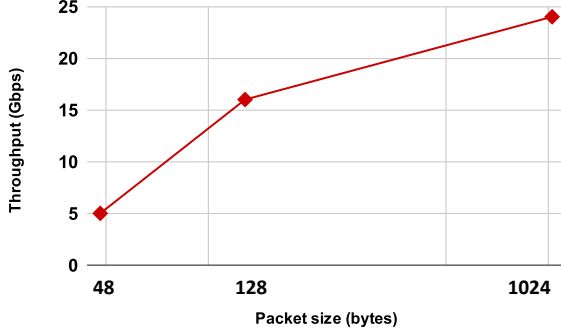


Fig. 4. Throughput vs. message size on a no-touch flow. UDP packets received by the SmartNIC are transferred to the host CPU with no modification to the message.

**Limitation.** The SmartNIC compute elements can be used to execute at line rates the offloaded kernels for each IoT message, process the critical path, and with low latency send out to subscribers messages like the anomaly notifications shown in Figure 1. However IoT messages must also be ingested by the host-side analytics path, for long(er) term modeling and optimizations. Figure 4 exposes the problem of processing high message rates of small-sized packets. In this microbenchmark, all 5 CPU cores were at 100% utilization. We notice that, with the same number of cores, the throughput can be increased by almost 5x over the minimum size packet. Above 1024 bytes the number of ingested messages per second, not the message size, becomes the bottleneck. Current SmartNIC acceleration solutions provide scalability to the host-side analytics by freeing up host resources via offload but does not explicitly consider the scaling limitations from the large number of IoT messages that still need to be delivered to and processed in the analytics data path.

#### IV. ACCELERATING THE IOT ANALYTICS PATH

Given the richness of the IoT space, the operations in the analytics path vary significantly, ranging from database load/store and queries, to machine learning training. As such, we rely on the host-side runtime CPU for their execution. We use two insights to scale up the analytics path on the host-side runtime CPU. First, at high message loads, the host-side runtime CPU becomes a bottleneck for even the basic processing needed to extract individual IoT messages from network packets,

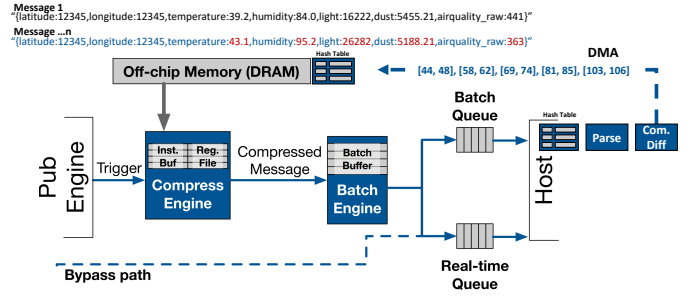


Fig. 5. Compress Engine and Batch Engine used for accelerating the transfer of IoT messages from the network card to the host CPU.

underutilizing both the available network bandwidth and PCIe bandwidth. Second, the analytics path is throughput oriented and does not have strict requirements on latency, unlike the critical path. Leveraging the two insights, we propose Compress, a NIC-side engine that compresses and batches together a large number of IoT messages to fill out pre-allocated (MTU-sized) buffers that are then passed to the host-side CPU runtime. The current implementation re-uses the functionality for the network interface packet buffer management on our in-network accelerator SmartNIC, and thus uses maximum size 1500 byte UDP-over-Ethernet network packet buffers for the SmartNIC-host-side runtime communication.

**Compression and Batching of IoT messages.** Figure 5 depicts the design of the Compress Engine and the Batch Engine. The Compress Engine receives its trigger to start execution when an IoT message is processed by the pipeline of engines that accelerate the critical path. IoT messages are too small to benefit from generic compression techniques like Snappy [17] and LZ4 [50] (see Table I). In fact, as shown in Figure 7, both LZ4 and Snappy *increase* the overall data volume. However, we observe that a large portion of IoT messages remain unchanged across the two endpoints – the sensors publishing to a certain topic and the application server. For instance, a geolocation of a fixed sensor in CITY, or of a taxi waiting for a client in TAXI, user and device ids in FIT and GRID etc., are common examples of parameters that can remain unchanged across messages.

**Static and Dynamic regions** The loop-like process of finding and eliminating unchanged regions of the IoT message is illustrated in Figure 5. The metadata used to compress and decompress a message is grouped by the topic string with the help of a Topic Table. When a host server receives a message, it checks to see if there is an entry corresponding to the message’s topic. If the entry does not exist, a function that computes dynamic region (**Comp.Diff**) marks the entirety of the message as dynamic. As more messages for the same topic arrive, **Comp.Diff** starts to identify regions of the message that have remained unchanged across a given number of messages. The host-side runtime, then, loads the Diff Topic Table on the FPGA NIC with the index and offsets of the dynamic regions. The Compress Engine uses this information to remove the static regions from upcoming messages to the same topic,



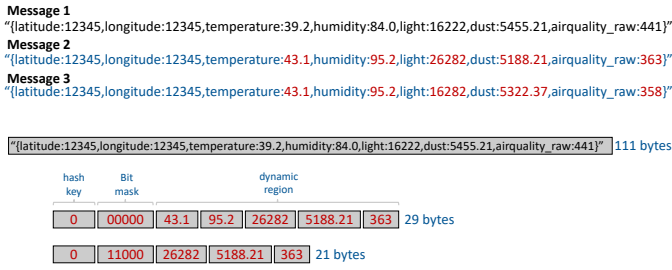


Fig. 6. Example of 3 messages for the same topic arriving in the Compress Engine. 1. the entirety of the message is transferred to host. 2. only the dynamic regions are transferred, alongside the hash key and the bit mask. 3. two dynamic regions are eliminated from the message since they remained unchanged.

sending only the dynamic region to the host. Furthermore, the Compress Engine stores  $n$  number of messages on the same Topic Table entry. If the **Comp.Diff** detects any variation on the static region, it instructs the Compress Engine to dump its stored messages on the upcoming packet, which goes into the Real-Time Queue, and the process starts over.

**Dynamic region elimination.** Even dynamic regions can become static for brief period of time. A stationary vehicle reporting its speed, for example, will report the same speed and geolocation in multiple messages. Figure 6 illustrates how we leverage this eventual duplication of data in the dynamic regions to eliminate it from the message payload. This is accomplished by adding a 16 bit mask to the message payload that allows the runtime to track the dynamic regions that were eliminated in the compression process. Additionally, we observe that the topic string itself accounts for 41% in the FIT message payload and as high as 76% in the Smart Grid, after static regions are removed. Since the topic table is shared between the Vasado SmartNIC and the host-side runtime and control drivers, only a 4 byte hash key is added to the message payload, yielding a higher compression rate.

After compression, the Batch Engine combines multiple compressed IoT messages to construct a single large UDP message. The implementation of the Batch Engine is straightforward. It keeps track of the size of the current load, and checks to see if the next message coming from the Compress Engine fits. If it fits, the compressed message is added to the load, and the load size is updated. Otherwise, the current load is transferred to the host-side runtime and a new load is created.

Once the batch is received by the host-side runtime, it processes the protocol stack once for the entire batch, decompresses the message by adding the deleted regions, and passes the individual messages to the application. No modifications are required in the application in order to ingest and process the messages.

One could argue that Snappy and LZ4 will demonstrate better results if used after batching. However, this would impact the number of messages in the batch and require costly variable-length string manipulations operations [2]. Leveraging the well-formed structure of IoT messages allows Compress to simplify

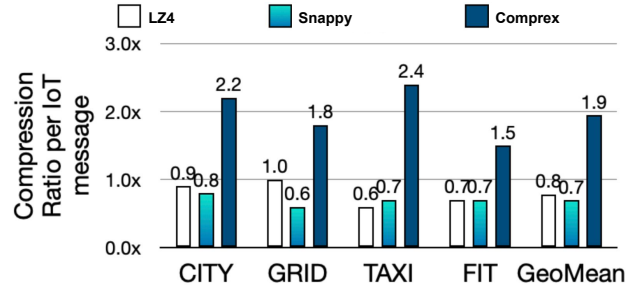


Fig. 7. Compression gain comparison between Compress's IoT specific compression and generic compression techniques.

the compression and decompression process by copying data from fixed offsets. It also opens up opportunities for future optimizations such as to pre-initialize message buffer.

**Software Managed Queues.** The current implementation for our prototype uses two different queues to send processed UDP packets to the host-side runtime CPU: (1) a low-priority queue that holds IoT topics that are either not accelerated or are only partially accelerated on the SmartNIC; and (2) a high-priority queue that holds the combined large compressed and batched messages. The priorities of the two queues are managed by the Compress runtime system to ensure fairness and to avoid starvation for different types of queues.

## V. EVALUATION

### A. Methodology

**Benchmarks.** To evaluate the impact of domain-specific compression and batching on extending the benefits of SmartNIC offload alone, we use four representative IoT applications, summarized in Table I. The applications perform a diverse set of operations in the critical path to process incoming messages from publishers and to generate a response for subscribers. CITY [9] is a crowd-source deployed application aimed at monitoring urban air quality. Smart GRID [4] is an application composed of sensors that report energy consumption to a pilot smart GRID in Ireland. It has only one observation field, a meter id and a timestamp. FIT [34] is a health application that measures body movements and vital signs. TAXI [12] is a smart transportation application that reports information of the current trip.

**Evaluation testbed.** We use two servers equipped with Intel Xeon X3430 processors running at 2.4 GHz and 16 GB of RAM to emulate IoT traffic at scale. The servers are connected using two 40 Gbps Mellanox Innova Flex SmartNICs [32], each comprising a Mellanox ConnectX-4 Lx EN ASIC NIC and a Xilinx Kintex UltraScale (XCKU060) FPGA. The first server emulates IoT publishers and subscribers, while the second server hosts the IoT applications.

**Baselines.** All results for Baseline and Baseline plus Batch use an in-house C-based server application that operates on top of a light-weight UDP-over-Ethernet protocol. We compare this baseline against popular MQTT Mosquitto [16] and CoAP [51]

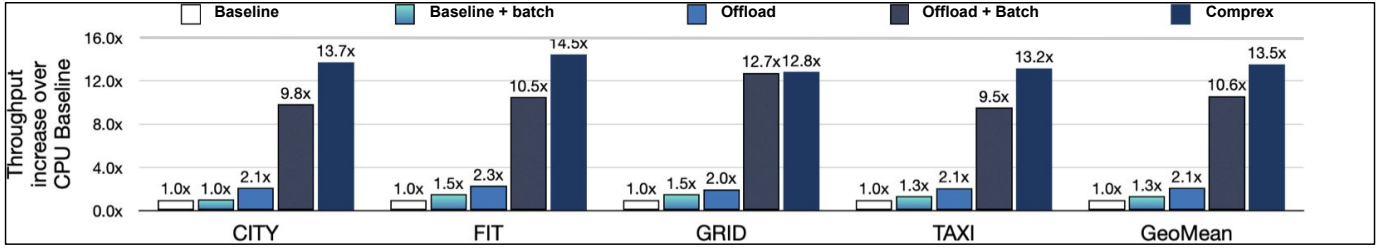


Fig. 8. Throughput comparison.

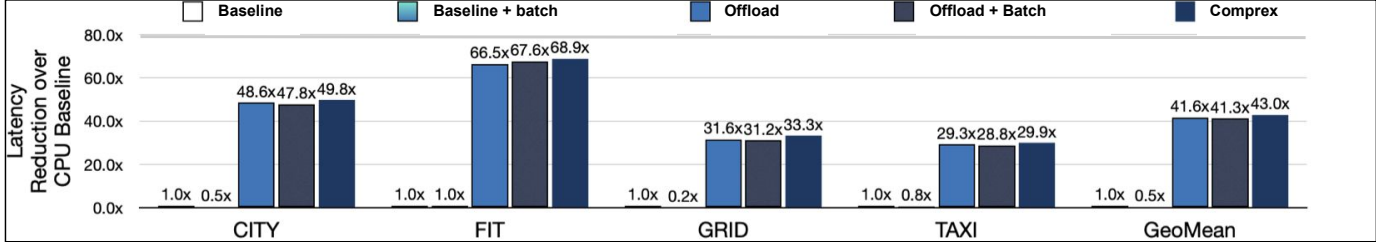


Fig. 9. Latency comparison.

IoT servers and establish that our in-house baseline is an order of magnitude faster than these systems. **Baseline** accepts a single IoT publisher message at a time from the NIC, processes the message to generate and send responses to all subscribers. For **Baseline plus Batch**, we use the SmartNIC to combine multiple IoT messages from different topics into a single 1500 Byte batch. For **Offload**, we use the SmartNIC to completely subsume the operations for the critical path in each topic of the evaluated applications. **Offload plus Batch** further performs batching. Finally **Comprex** performs compression and batching as described in Section IV to fit a larger number of IoT messages in a single batch.

**Latency and throughput metrics.** We measure the latency of the critical path as the time from the publishing of an IoT message to the time when the last IoT notification is received by subscribers. The reported stable throughput corresponds to a level where message loss does not exceed a current threshold of 0.1% for a duration of 10 minutes. More specifically, for each measurement in this section, we increase the throughput until the host-side becomes unstable. As such, throughput measurements are calculated for messages that have the *critical* and *non-critical sides* properly handled by the system.

### B. Experimental Results

**Increase in sustained throughput.** Figure 8 compares the throughput in messages-per-second, across different configurations. **Baseline plus Batch** shows a geomean 1.3 $\times$  increase in throughput from batching multiple IoT messages into a single network packet, at the cost of  $\approx 2\times$  increase in latency, as we show in the Figure 9. Further, hardware acceleration alone does not provide a significant improvement in throughput. **Offload** provides a 2.1 $\times$  improvement in throughput over **Baseline** and a 1.6 $\times$  improvement in throughput over **Baseline plus Batch**.

By combining the benefits from hardware acceleration and offload with optimizations for domain-specific batching and compression of IoT messages on-the-fly, **Offload** and **Comprex** provide a geomean 10.6 $\times$  and 13.5 $\times$  increase in performance over **Baseline**. Unlike the CPU-only **Baseline plus Batch**, neither **Offload plus Batch** nor **Comprex** show a significant increase in latency since the entire critical path is accelerated on the SmartNIC to generate a reply before applying batching and compression. **FIT** sees the highest increase in throughput of 14.5 $\times$  with **Comprex** since the size of the IoT message is highest in that benchmark. **GRID** sees the lowest throughput improvement of around 12.8 $\times$  with **Comprex** since this benchmark does not benefit significantly from compression.

**Scalability of host resources.** We evaluate the end-to-end impact on the scalability of the host resources. For **Baseline**, we measure the CPU-side requirements to execute the critical path operations on the host and to pass the data to the long-term analytics component. In the **Comprex** case, the critical path is executed on the SmartNIC using a pipeline-like accelerator similar to NICA [13]; the host-side runtime component receives the compressed messages and performs decompression before delivering them to the rest of the application. In order to avoid measuring bottlenecks related to our current database or machine learning components, in these measurements the application is an in-memory log. When comparing **Baseline** and **Comprex** with respect to the per-core CPU utilization, at the same level of sustained throughput, **Baseline** utilizes 91-99% CPU, whereas **Comprex** requires only 11-12%. This is an 8 $\times$  improvement in CPU efficiency.

**Decompression performance.** To compare the performance of the decompression part of **Comprex** and **Snappy**, we create two dump files with four million compressed messages from all four IoT applications. Only one CPU core was used on this micro-

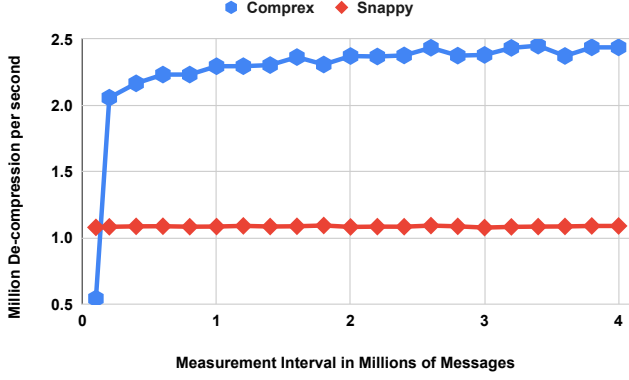


Fig. 10. Decompression throughput comparison between Compress and Snappy. For the first few thousand iterations, Compress is heavily impacted by **Comp.Diff** overhead. As static regions remained unchanged, **Comp.Diff** overhead is greatly reduced. Notice that Snappy throughput is unchanged over time.

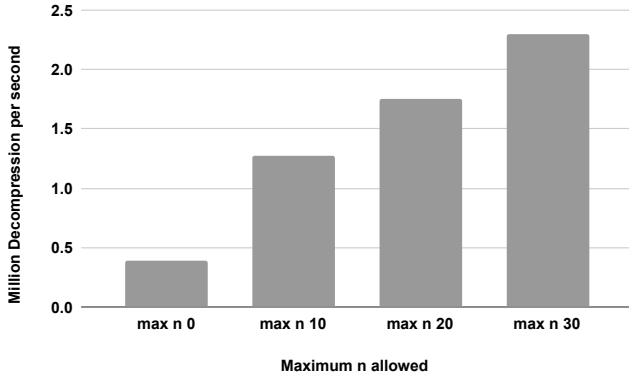


Fig. 11. Performance measurement for different caps on how much **n** (interval in number of messages) can grow before **Comp.Diff** is executed. As static regions remain unchanged, **n** increases, incrementally, until it reaches the **max n allowed**.

benchmark. As illustrated in Figure 10, Compress is heavily penalized by cold-start since it executes **Comp.Diff** for every message. As messages for the same topic start to repeat over time and static regions remain unchanged, calls to **Comp.Diff** is reduced resulting in a  $\approx 2\times$  higher throughput compared to Snappy. The wavy behavior of Compress throughput can be attributed to the overhead of detecting a change in the static region. It not only resets the metadata for the given topic, but forces the SmartNIC to transfer previously stored messages to the host side. Lastly, if a topic does not have static regions that remain unchanged for at least five messages, Compress reduces the frequency in which **Comp.Diff** is executed until the topic is completely removed from consideration.

**Impact of max n on throughput and lossiness.** Figure 11 illustrates the impact of different caps imposed on **n**. If **max n** is 0, **Comp.Diff** is executed on every message which greatly reduces the benefits of Compress. When **max n** is set to 10, Compress can perform over 1M decompression per second.

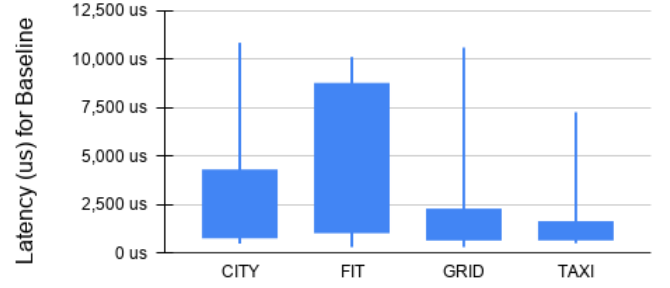


Fig. 12. Latency variation for Baseline. The figure shows minimum, 25<sup>th</sup> percentile, 75<sup>th</sup> percentile, and maximum latency for each benchmarked application. For the same workload Offload + Compress 99<sup>th</sup> percentile latency is less than 60  $\mu s$

Above **max n** 30 the benefits are negligible. As seen in section IV, **n** indicates how many messages need to be stored on the SmartNIC DRAM for Compress to be lossless. For example, if **max n** is set to 10 and **Comp.Diff** detects a change on static region on the 11<sup>th</sup> message received, Compress can still retrieve the original messages. However, being over-cautious and storing 30x more messages than necessary might not be the best case for IoT specific workloads. For instance, it is common to see implementations [3] using interpolation to stipulate missing values. To account for such scenarios, Compress allows user to determine how much **max n** can drift from **max store** before the SmartNIC dumps stored messages at the **Comp.Diff** request. For the microbench in Figure 10, no data was lost with **max n** and **max store** set to 30 and 10 respectively.

**Reduction in average latency.** Figure 9 compares the reduction in the average latency of Baseline, Baseline plus Batch, Offload, Offload plus Batch, and Compress compared to Baseline for each of the benchmarks. As shown in the figure, batching has an adverse impact on the latency for the critical path and increases the critical path latency. Offload, Offload plus Batch, and Compress all use the SmartNIC to yield a significant reduction in latency by completely offloading the operations of the critical path. The FIT benchmark sees the highest reduction in critical path latency from offload since this benchmark has the highest number of parameters used for computations of the critical path. The GRID and TAXI benchmarks see the lowest reduction in latency since these benchmarks have fewer number of parameters that are processed for the critical path. Nevertheless, all benchmarks see more than 28.8 $\times$  reduction in latency. Furthermore, none of the benchmarks show a significant degradation in latency from batching in Offload plus Batch and batching+compression in Compress, compared to Offload, since batching is performed outside of the critical path of the message latency. This illustrates how Compress significantly raises end-to-end stable throughput while not having a negative impact on latency for the critical path.

**Impact on tail latency.** Interestingly, for the offload-based cases, all the benchmarks show  $\approx 60 \mu s$  latency, while latency for the CPU baselines – Baseline and Baseline plus Batch – exhibits high variability, from 600  $\mu s$  to 12,000  $\mu s$ , depending on the workload. Average latency for CPU baselines varies

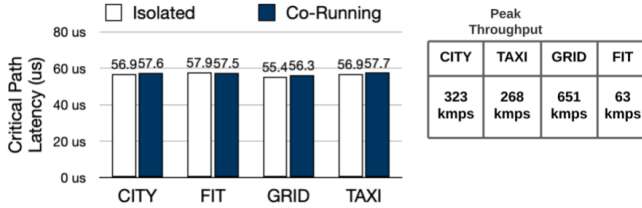


Fig. 13. Latency for critical path when running a single vs. multiple applications.

from and 1,800  $\mu$ s (GRID) to 4,200  $\mu$ s (FIT). Figure 12 summarizes the variation in latency for the CPU Baseline for the benchmarked applications. The CPU-only baseline has a high variation in latency, with tail latency between  $31\times$  (GRID) to  $14\times$  (TAXI) the minimum latency for the critical path. Cloud IoT services often provide SLAs for the 99<sup>th</sup> percentile latency, and limit the number of messages per second that can be processed to not violate the SLAs. Offload plus Comprehex, on the other hand has a 99<sup>th</sup> percentile that is far more stable at higher message rates and lower than the CPU baselines by completely circumventing the CPU and generating a response directly in the FPGA-equipped NIC.

**Co-running multiple applications.** Figure 13 shows the impact of accelerating multiple applications with our system. Isolated execution accelerates only the topics for a single application, shown in the x-axis. Each application (eg. CITY) consists of 5,000 topics, where each topic performs the exact same computations. Co-running execution accelerates multiple applications by randomly interleaving messages from topics in all four benchmarked application. As Figure 13 shows, there is no noticeable degradation in latency when co-running multiple IoT applications. The slight variation in latency reported in the figure stems from the networking stack and software overheads when measuring latency. The actual variation in latency within the SmartNIC measured via simulations, is less than a tenth of a  $\mu$ s ( $\sim 10$  cycles @150MHz) between the application with the most computations FIT, and the application with the least computations TAXI.

**FPGA utilization.** We are able to support all four applications using around 2% LUTs and 3% BRAMs resources available in the Xilinx XCKU060 FPGA on the Mellanox SmartNIC [32].

## VI. RELATED WORK

**IoT Specific Compression and Batching** Several works have demonstrated the benefits of compressing and batching IoT messages at the edge [30] [35] [52]. However, naive adoption of these techniques for accelerated SmartNIC-based IoT applications puts the de-/compression and batching operations in the critical path, making them not suitable for IoT applications with real-time processing requirements.

**SmartNIC acceleration frameworks.** A number of related efforts have explored the benefits of in-network acceleration or developed programming interfaces for computation offload to SmartNICs [13], [18], [22], [24], [26], [27], [29], [36], [39], [41]. NICA [13] provides a framework for general server

application acceleration on FPGA-based SmartNICs. It uses a low-level socket interface for steering traffic to accelerators. sPIN [18] and INCA [41] provide programming models for in-line message processing in HPC systems. INCA uses tag-matching for selecting instructions, analogously to Comprehex topic abstraction. FlexNIC [22] offers a SmartNIC design based on reconfigurable match-action tables, and COPA [24] presents a generic architecture for integration of inline or lookaside accelerator cores with the SmartNIC packet processing path. The work presented in this paper is complementary to those efforts, as it focuses on further amplifying the benefits from SmartNIC offload via more effective compression and batching, as shown via the differences between the Comprehex and offload measurements in Figure 8.

**Accelerators for IoT.** Several works have demonstrated accelerators that may be utilized for individual processing steps of a typical IoT pipeline. Optimus Prime [37] accelerates data format transformation, and NICA [13] includes an IoT cryptographic message authentication engine. Previous work has created sketch and statistics accelerators [25], [46], predictive engines [10], [42], [47], accelerators for AI-based IoT device management [7], or for acceleration of other common operators in the IoT critical path [8]. These engines can be combined with Comprehex as part of an IoT SmartNIC solution.

## VII. CONCLUSION

The rapid increase in the number of IoT devices and the real-time ultra-low latency requirements of new 5G applications challenges IoT service providers. Solutions that combine offload to SmartNICs and acceleration of common latency critical operations present in these workloads, provide significant benefits. This paper presents a solution that further amplifies those benefits through use of an IoT-specific compression and batching engine, that further improves the efficiency and scalability of the IoT analytics server platforms. Our results with 4 real-world applications enable offloads to SmartNIC to reduce critical path latency by  $43\times$  while increasing the stable throughput  $13.5\times$ , and show  $8\times$  increase in CPU efficiency.

The solution also paves the way for integration of other, potentially lossy, compression techniques which leverage the IoT message structure to provide further benefits. In addition, beyond just as part of the SmartNIC-server interface, these techniques can be relevant in general at the network interface of distributed IoT infrastructure. Namely, leading IoT service providers [5], [6], [15] sometimes split the application critical and non-critical parts and run them separately – at the edge and at the data center respectively. Domain-specific compression and batching can be integrated at the network interfaces between the edge and cloud, to reduce network load and improve the infrastructure efficiency.

**Acknowledgement.** We thank the anonymous reviewers for their valuable feedback. Hardik Sharma, Haggai Eran, Hadi Esmaelizadeh and Mark Silberstein helped in various stages of this project. This work was partially supported by NSF awards SPX-1822972 and CNS-2016701, and by the ADA and PRISM centers, via the joint SRC and DARPA JUMP programs.



## REFERENCES

- [1] "Cisco Annual Internet Report," <https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html>, Cisco, 2018.
- [2] B. Abali, B. Blamer, M. Klein, A. Mishra, C. B. Agricola, B. Sendir, A. Buyuktosunoglu, C. Jacobi, W. J. Starke, H. Myneni, and C. Wang, "Data Compression Accelerator on IBM POWER9 and z15 Processors : Industrial Product," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, 2020, pp. 1–14.
- [3] S. Anshu, C. Shilpa, and S. Yogesh, "Riotbench: A real-time iot benchmark for distributed stream processing platforms," *TPCTC*, 2016. [Online]. Available: <https://arxiv.org/pdf/1701.08530.pdf>
- [4] I. A. D. Archive. (2012) Smart grid. [Online]. Available: <https://www.ucd.ie/issda/data/commissionforenergyregulationcer/>
- [5] AWS. (2020) AWS IoT core documentation. [Online]. Available: <https://docs.aws.amazon.com/iot/index.html>
- [6] Azure IoT Hub. (2020) Azure IoT hub. [Online]. Available: <https://azure.microsoft.com/en-us/services/iot-hub/>
- [7] N. T. R. Babu and C. Stewart, "Energy, latency and staleness tradeoffs in AI-driven IoT," in *Proceedings of the 4th ACM/IEEE Symposium on Edge Computing*, ser. SEC '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 425–430. [Online]. Available: <https://doi.org/10.1145/3318216.3363381>
- [8] S. Biookaghazadeh, M. Zhao, and F. Ren, "are fpgas suitable for edge computing?"
- [9] D. Canvas. (2020) Sense your city. [Online]. Available: <http://datacanvas.org/sense-your-city/>
- [10] E. Chung and t. y. v. n. p. J. Fowers et.al, journal=IEEE Micro.
- [11] G. I. Core, "Oden technologies: Optimizing production on the factory floor," Google Cloud Solutions, 2020. [Online]. Available: <https://cloud.google.com/customers/oden-technologies>
- [12] B. Donovan and D. B. Work, "Using coarse GPS data to quantify city-scale transportation system resilience to extreme events," 2015.
- [13] H. Eran, L. Zeno, M. Tork, and M. Silberstein, "NICA: An infrastructure for inline acceleration of network applications," *atc*, 2019. [Online]. Available: [https://haggaie.github.io/files/atc19nica\\_final.pdf](https://haggaie.github.io/files/atc19nica_final.pdf)
- [14] P. S. et al, "Latency critical iot applications in 5g: Perspective on the design of radio interface and network architecture," in *IEEE Communications Magazine*, vol. 55. IEEE Communications Magazine, 2008, pp. 207–220. [Online]. Available: <https://ieeexplore.ieee.org/document/7842415>
- [15] Google Cloud. (2020) Cloud IoT core. [Online]. Available: <https://cloud.google.com/iot-core>
- [16] Google Cloud. (2020) HTTP vs. MQTT: A tale of two IoT protocols. [Online]. Available: <https://cloud.google.com/blog/products/iot-devices/http-vs-mqtt-a-tale-of-two-iot-protocols>
- [17] Google Snappy. (2020) Cloud snappy. [Online]. Available: <https://google.github.io/snappy/>
- [18] T. Hoefler, S. Di Girolamo, K. Taranov, R. E. Grant, and R. Brightwell, "sPIN: High-performance streaming processing in the network," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 2017, p. 59.
- [19] Intel. Accelerate Data from Edge to Cloud, urldate = 2023.
- [20] Z. István, D. Sidler, G. Alonso, and M. Vukolic, "Consensus in a box: Inexpensive coordination in hardware," in *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, 2016.
- [21] B. E. S. Jiaxin Lin, Kiran Patel, "Panic: A high-performance programmable nic for multi-tenant networks," *USENIX*, 2020. [Online]. Available: <https://wiscr.cs.wisc.edu/papers/osdi20-panic.pdf>
- [22] A. Kaufmann, S. Peter, N. K. Sharma, T. Anderson, and A. Krishnamurthy, "High performance packet processing with FlexNIC," in *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 67–81. [Online]. Available: <https://doi.org/10.1145/2872362.2872367>
- [23] J. Korhonen and Y. Wang, "Effect of packet size on loss rate and delay in wireless links," vol. 3, 04 2005, pp. 1608 – 1613 Vol. 3.
- [24] V. Krishnan, O. Serres, and M. Blocksome, "Configurable Network Protocol Accelerator (COPA) - An Integrated Networking/Accelerator Hardware/Software Framework," in *Hot Interconnects'20*, 2020.
- [25] A. Kulkarni, M. Chiosa, T. B. Preußner, K. Kara, D. Sidler, and G. Alonso, "Hyperloglog sketch acceleration on FPGA," *arXiv preprint arXiv:2005.13332*, 2020.
- [26] B. Li, K. Tan, L. L. Luo, Y. Peng, R. Luo, N. Xu, Y. Xiong, P. Cheng, and E. Chen, "ClickNP: Highly flexible and high performance network processing with reconfigurable hardware," in *Proceedings of the 2016 ACM SIGCOMM Conference*, ser. SIGCOMM '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 1–14. [Online]. Available: <https://doi.org/10.1145/2934872.2934897>
- [27] M. Liu, T. Cui, H. Schuh, A. Krishnamurthy, S. Peter, and K. Gupta, "Offloading distributed applications onto SmartNICs using iPipe," in *Proceedings of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 318–333. [Online]. Available: <https://doi.org/10.1145/3341302.3342079>
- [28] M. Liu, L. Luo, J. Nelson, L. Ceze, A. Krishnamurthy, and K. Atreya, "IncBricks: Toward In-Network Computation with an In-Network Cache," in *Architecture Support for Programming Languages and Operating Systems (ASPLOS)*, 2017.
- [29] M. Liu, S. Peter, A. Krishnamurthy, and P. M. Phothilimthana, "E3: Energy-efficient microservices on SmartNIC-accelerated servers," in *2019 USENIX Annual Technical Conference (USENIX ATC 19)*. Renton, WA: USENIX Association, Jul. 2019, pp. 363–378. [Online]. Available: <https://www.usenix.org/conference/atc19/presentation/liu-ming>
- [30] T. Lu, X. Zou, Q. Xia, and W. Xia, "Adaptively compressing iot data on the resource-constrained edge," *hotedge*, 2020. [Online]. Available: [https://www.usenix.org/system/files/hotedge20\\_paper\\_lu.pdf](https://www.usenix.org/system/files/hotedge20_paper_lu.pdf)
- [31] Marvell. Data Processing Units. [Online]. Available: <https://www.marvell.com/products/data-processing-units.html>
- [32] Mellanox Technologies. (2017) Innova Flex 4 Lx EN adapter card product brief. [Online]. Available: [https://www.mellanox.com/related-docs/prod\\_adapter\\_cards/PB\\_Innova\\_Flex4\\_Lx\\_EN.pdf](https://www.mellanox.com/related-docs/prod_adapter_cards/PB_Innova_Flex4_Lx_EN.pdf)
- [33] Mellanox Technologies. (2019) BlueField SmartNIC for Ethernet. [Online]. Available: [https://www.mellanox.com/sites/default/files/related-docs/prod\\_adapter\\_cards/PB\\_BlueField\\_Smart\\_NIC.pdf](https://www.mellanox.com/sites/default/files/related-docs/prod_adapter_cards/PB_BlueField_Smart_NIC.pdf)
- [34] M. Memon, S. R. Wagner, C. F. Pedersen, F. H. A. Beevi, and F. O. Hansen, "Ambient assisted living healthcare frameworks, platforms, standards, and quality attributes," *Sensors*, vol. 14, no. 3, pp. 4312–4341, 2014.
- [35] T. Nguyen gia, Q. L., J. Peña Queralta, Z. Zou, H. Tenhunen, and T. Westerlund, "Lossless compression techniques in edge computing for mission-critical applications in the iot," 11 2019.
- [36] P. M. Phothilimthana, M. Liu, A. Kaufmann, S. Peter, R. Bodik, and T. Anderson, "Floem: A programming system for NIC-accelerated network applications," in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. Carlsbad, CA: USENIX Association, Oct. 2018, pp. 663–679. [Online]. Available: <https://www.usenix.org/conference/osdi18/presentation/phothilimthana>
- [37] A. Pourhabibi, S. Gupta, H. Kassir, M. Sutherland, Z. Tian, M. P. Drumond, B. Falsafi, and C. Koch, "Optimus Prime: Accelerating data transformation in servers," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 1203–1216. [Online]. Available: <https://doi.org/10.1145/3373376.3378501>
- [38] T. P. Raptis, A. Passarella, and M. Conti, "Maximizing industrial iot network lifetime under latency constraints through edge data distribution," *EEE Industrial Cyber-Physical Systems*, 2018. [Online]. Available: <https://ieeexplore.ieee.org/document/8390794>
- [39] T. Rinta-aho, M. Karlstedt, and M. P. Desai, "The Click2NetFPGA toolchain," in *Presented as part of the 2012 USENIX Annual Technical Conference (USENIX ATC 12)*. Boston, MA: USENIX, 2012, pp. 77–88. [Online]. Available: <https://www.usenix.org/conference/atc12/technical-sessions/presentation/rinta-aho>
- [40] M. Satyanarayanan, "The emergence of edge computing," *Computer*, vol. 50, no. 1, pp. 30–39, 2017.
- [41] W. Schonbein, R. E. Grant, M. G. Dosanjh, and D. Arnold, "INCA: in-network compute assistance," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2019, pp. 1–13.
- [42] H. Sharma, J. Park, D. Mahajan, E. Amaro, J. K. Kim, C. Shao, A. Mishra, and H. Esmailzadeh, "From high-level deep neural models to FPGAs," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2016, pp. 1–12.

- [43] A. I. Solutions, “Think big, prototype small, scale fast,” Amazon IoT Solutions, 2018. [Online]. Available: [https://aws.amazon.com/solutions/case-studies/woodside/#Woodside\\_Case\\_Study.3A\\_Think\\_big\\_Prototype\\_small\\_Scale\\_fast](https://aws.amazon.com/solutions/case-studies/woodside/#Woodside_Case_Study.3A_Think_big_Prototype_small_Scale_fast).
- [44] A. I. Solutions, “What is aws iot things graph?” AWS IoT Solutions, 2021. [Online]. Available: <https://docs.aws.amazon.com/thingsgraph/latest/ug/iot-tg-what-is.html>
- [45] N. Sornin, M. Luis, T. Eirich, T. Kramp, and O. Hersent, “Lorawan specification,” LoRa alliance, 2015.
- [46] D. Tong and V. K. Prasanna, “Sketch acceleration on FPGA and its applications in network anomaly detection,” IEEE Transactions on Parallel and Distributed Systems, vol. 29, no. 4, pp. 929–942, 2018.
- [47] B. Van Essen, C. Macaraeg, M. Gokhale, and R. Prenger, “Accelerating a random forest classifier: Multi-core, GP-GPU, or FPGA?” in 2012 IEEE 20th International Symposium on Field-Programmable Custom Computing Machines, 2012, pp. 232–239.
- [48] A. C. Vehicle, “Aws connected vehicle solution,” Amazon Solutions, 2020. [Online]. Available: <https://docs.aws.amazon.com/solutions/latest/connected-vehicle-solution/connected-vehicle-solution.pdf#welcome>
- [49] Y.-P. E. Wang, X. Lin, A. Adhikary, A. Grovlen, Y. Sui, Y. Blankenship, J. Bergman, and H. S. Razaghi, “A primer on 3gpp narrowband internet of things,” IEEE communications magazine, vol. 55, no. 3, pp. 117–123, 2017.
- [50] Yann Collet. (2011) LZ4. [Online]. Available: <https://lz4.github.io/lz4/>
- [51] K. H. Z. Shelby and C. Bormann. (2020) “constrained application protocol (coap) (core working group). [Online]. Available: <https://datatracker.ietf.org/doc/draft-ietf-core-coap/>
- [52] X. Zenga and S. Zhang, “Cstream: Parallel data stream compression on multicore edge devices,” 7 2023.
- [53] J. C. Zuniga and B. Ponsard, “Sigfox system description,” LPWAN@ IETF97, Nov. 14th, vol. 25, 2016.