3

GStreamer Plugin for RDMA Offload on BlueField-3 for Edge Applications

Raphael Frantz[1], Romain Pouillard[2], Idelfonso Tafur Monroy[3], Juan Jose Vegas Olmos[4], and Salvatore Di Girolamo[5]

[1]NVIDIA, Italy / Eindhoven University of Technology, The Netherlands
[2]Université Clermont Auvergne, France / Scuola Superiore Sant'Anna, Italy
[3]Eindhoven University of Technology, The Netherlands
[4]NVIDIA, Denmark
[5]NVIDIA, Switzerland

Abstract

Edge computing enables running applications without the added latency of transferring data to datacenters. With an application processing a video stream, an AI agent can operate near the end user using these decentralized servers. Remote Direct Memory Access (RDMA) is a high-speed network transport protocol (100Gbps and more) that can be run at the edge to facilitate communication between different systems. This allows processing the video stream in Data Processing Units (DPUs), instead of running on servers, where CPU usage is a scarce resource. To demonstrate, we implement a demo application of a virtual try-on by leveraging RDMA and DPUs, allowing a better usage of hardware resources. We leverage GStreamer, an industry-standard framework for processing media streams, and implement an RDMA network plugin for BlueField DPUs.

Keywords: RDMA, DOCA, BlueField, DPU, SmartNIC, GStreamer, Video Streaming, Edge Computing.

3.1 Introduction and Background

Applications based on video processing, such as those required by media streaming platforms, need to perform various operations on the video stream, including decompression, image extraction, AI agent execution, and image compression before sending it back to the viewer. Frameworks such as GStreamer can be used as a middleware to interface these independent operations, allowing a pipeline architecture that can run on multiple servers. To communicate between the hosts, network protocols such as UDP or TCP can be used natively in GStreamer. However, these protocols are processed by the kernel, utilizing precious CPU resources and running memory copies, which wastes memory bandwidth. In contrast, RDMA is a zero-copy protocol that bypasses the kernel. The packets are directly processed by the network card, which places data in memory at the proper virtual address using the PCIe bus. In addition to offloading the network stack, RDMA offers ultra-high bandwidth, which enables transferring raw images of a video stream, offering the possibility to offload encoding and decoding to specialized hardware. GStreamer does not support RDMA out of the box, but can be extended via plugins. Therefore, we implement an RDMA plugin for BlueField DPUs. It uses RDMA over Converged Ethernet (RoCE) to be easily integrated into IP networks.

BlueField is a DPU that integrates an RDMA-capable network card for both InfiniBand and RoCE networks, and lightweight ARM cores that can be used to accelerate the application. To run the demo, we use the BlueField-3 that integrates eight ARMv8.2 A78 cores along with 32 GB of on-board memory.

Recent research has demonstrated the potential of BlueField DPUs to offload servers in distributed architectures. For instance, the study by Kashyap et al. provides a comprehensive performance analysis of BlueField generations 1 through 3, identifying key trade-offs related to integration mode (off-path vs. on-path). Notably, we observe a 30% increase in RDMA latency in on-path mode on BlueField-3 [9]. Additionally, Li et al. show that computational loads such as compression can be efficiently offloaded to DPUs using frameworks like PEDAL [10], significantly reducing host CPU usage. These findings reinforce the relevance of leveraging the hardware acceleration capabilities of BlueField DPUs for edge video processing.

GStreamer is widely used in embedded systems due to its modularity and extensibility. In the GOTE architecture, Robaina and Fiorese demonstrate how it enables a low-latency edge gaming pipeline by integrating NVENC

hardware encoding [11]. Similarly, the work of Primo da Silva et al. integrates a hardware JPEG encoder into a GStreamer pipeline running on an FPGA, confirming that GStreamer adapts well to real-time applications in constrained environments [12].

3.2 Virtual Try-On Application

We base the virtual try-on application architecture on previous work [1], integrating it with GStreamer and RDMA to enable real-time feedback. A virtual try-on enables users to virtually test clothes using a camera and a feedback screen, while running multiple AI model inferences on the backend to replace the client's clothes. The camera captures the video stream and sends it to the preprocessing edge server via UDP. The preprocessing server runs the first AI tasks (*OpenPose* [3–6], *HumanParsing* [7]), enabling segmentation

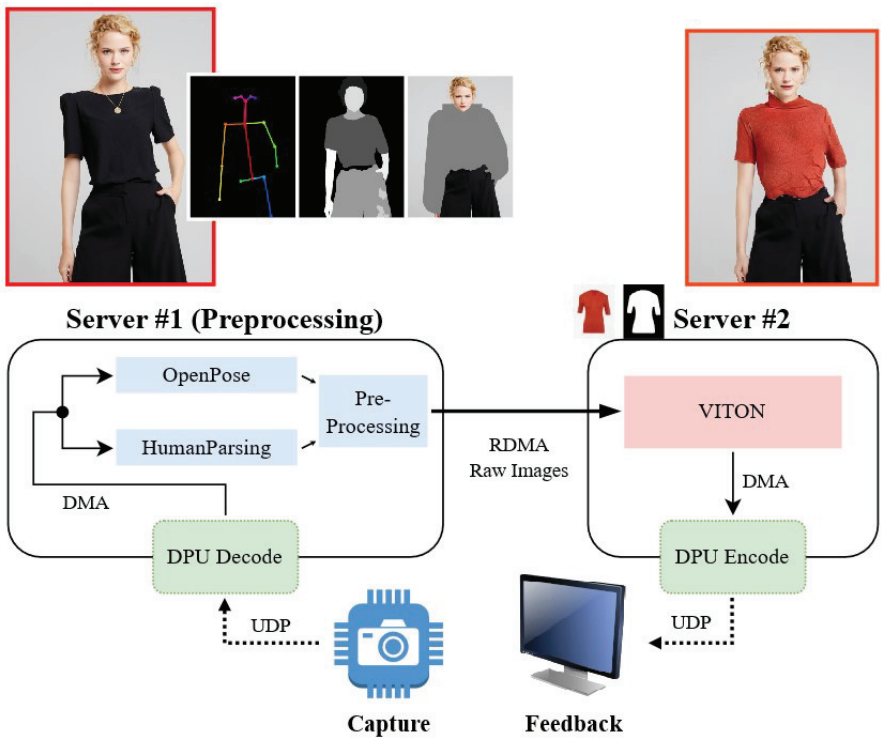

Figure 3.1 Architecture of the virtual try-on application with the two edge servers running the AI inference, and their DPU encoding/decoding the video.

of the human body. The proposed approach follows a distributed pipeline architecture inspired by initiatives like VideoPipe [13], which dynamically maps video processing tasks across multiple edge nodes based on workload and network conditions. This allows pose estimation (e.g., OpenPose) to run on lightweight devices, while the final image generation is offloaded to more powerful nodes. We adopt a similar split: pose detection and semantic parsing are executed on the first server, while the virtual try-on rendering (using VITON) is performed on the second. The original VITON model [14] introduced a two-stage synthesis pipeline to achieve photo-realistic try-on results without requiring 3D information and remains foundational to modern virtual try-on applications. The output of this stage is used as input to the *VITON* model [8] in the second server, which generates the final image from a human mask and the new cloth mask.

We use RDMA to communicate between the two edge servers. Raw images are sent to eliminate one encoding/decoding step, but increase drastically the bandwidth requirements of the video stream, which can be fulfilled thanks to RDMA. In addition, we offload image decoding/encoding to BlueField-3 DPUs. Indeed, RDMA works between distant hosts but also between the DPU and its attached host by leveraging Direct Memory Access (DMA) via the PCIe bus.

To have seamless operation between servers, parts of the application are integrated into a GStreamer pipeline. RDMA is integrated into GStreamer via our plugin, which allows zero-copy transfer between hosts.

3.3 RDMA Plugin

GStreamer is a widely adopted multimedia framework; however, it does not natively support data transport via Remote Direct Memory Access (RDMA). RDMA enables direct memory access between systems without CPU involvement, providing extremely low latency and high throughput, characteristics that make it well-suited for real-time multimedia workloads. We present here a novel GStreamer plugin designed to transfer multimedia streams using RDMA. Specifically, the plugin leverages the DOCA RDMA API to enable RDMA-based communication within GStreamer pipelines. It supports both inter-host RDMA transfers and host-to-DPU DMA communication. The proposed RDMA plugin integrates seamlessly with existing GStreamer pipelines, serving as a drop-in replacement for traditional transport elements such as TCP or UDP.

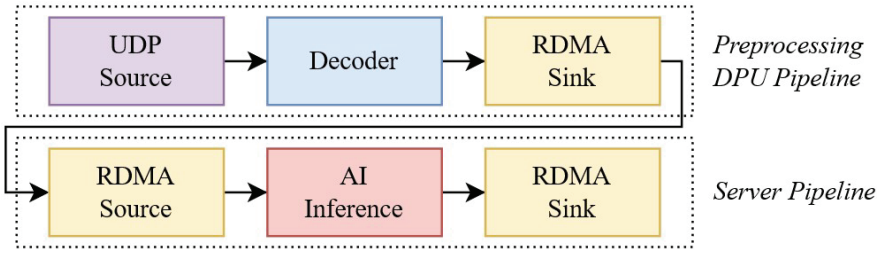

Figure 3.2 RDMA plugin integrated in GStreamer. Each operation is independent, and pipelines communicate via "sources" and "sinks". The DPU decodes the video while the server runs the AI inference.

While GStreamer includes native support for TCP, this protocol consumes significant CPU resources, particularly when handling large data volumes such as uncompressed video streams. UDP provides better performance but suffers from reliability issues and payload size limitations. The maximum UDP payload is restricted to a single MTU (typically less than 8 KiB), whereas a single raw video frame may occupy several megabytes. In contrast, RDMA combines high performance with reliability and supports message sizes up to 1 GB, effectively eliminating this constraint.

The RDMA plugin is implemented in C and C++. Its core is a C++ communication layer built on top of DOCA RDMA, responsible for reliable transmission and reception of media streams as well as credits-based flow control. A C interface bridges this layer with the GStreamer framework. The plugin provides two elements, a source for sending and a sink for receiving, as shown in Figure 3.2, similarly to how other protocols are implemented, adhering to GStreamer standards and permitting smooth integration. It enables zero-copy data transfer, allowing buffers to move directly between hosts without CPU intervention. Furthermore, the plugin ensures reliability by blocking the sending pipeline when the receiver cannot keep up, thereby avoiding data loss. The design remains fully transparent to applications and requires no changes to existing pipeline configurations.

3.4 Experimental Results

3.4.1 Offloading Scenarios

To evaluate the benefits of offloading video processing tasks to BlueField-3 DPUs, we conducted experiments on a dual-socket Intel Xeon Silver 4416+ server (80 cores), with 128 GB of RAM and two BlueField-3

DPUs. The experiment simulates an edge computing scenario in which the server receives a 1920×1080 H.264-encoded, pre-recorded, 30-second video stream, composed of different landscape scenes, over UDP (representing a camera), decodes it, re-encodes it, and transmits the re-encoded stream back to the final client over UDP. In a real-world application, a processing task would operate on the raw images between the encoding and decoding steps on the server, such as an AI model inference.

As shown in Figure 3.3, we compare four configurations, each reflecting whether the encoding step, the decoding step, or both are offloaded. As a baseline, in the "No Offloading" scenario, the entire decoding and encoding pipelines are handled by the host server CPU. For "Full Offloading", decoding and encoding are handled by separate DPUs, with the server managing only data transfer.

Figure 3.4 reports the server CPU usage over time for each scenario, measured as the number of logical cores actively used, averaged over multiple runs. As expected, the "No Offloading" case imposes the highest computational burden on the host, with CPU usage peaking at around four

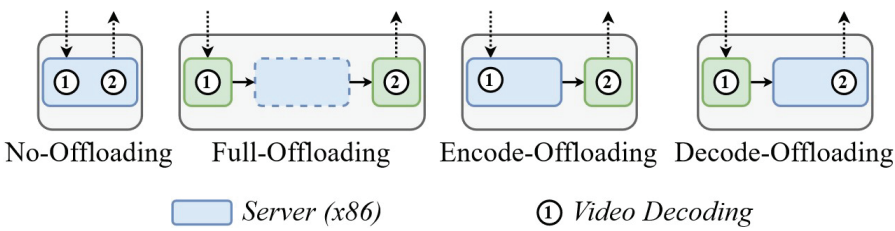

Figure 3.3 Tested scenarios to measure the impact of offloading specific algorithms to the BlueField-3 DPUs.

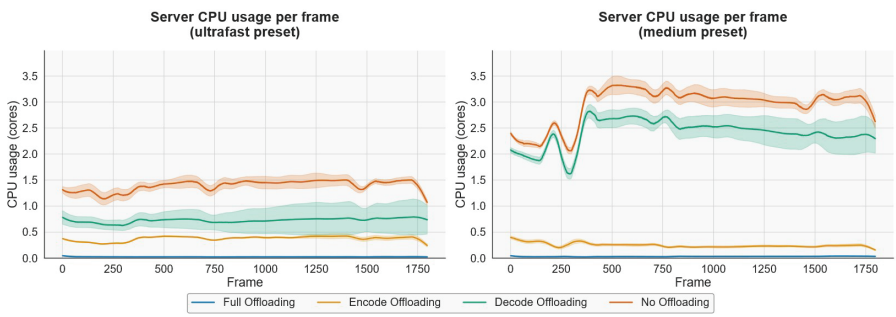

Figure 3.4 CPU usage of the server under different offloading scenarios for two H.264 encoder preset configurations: *ultrafast* and *medium*.

full cores. The "Decode Offloading" case slightly reduces the load, and the "Encode Offloading" case significantly reduces it. We observe that most of the computational cost is in video encoding. In the "Full Offloading" scenario, where both tasks are delegated to the BlueField-3 DPUs, server CPU usage drops consistently to near-zero levels, demonstrating a substantial gain in computational efficiency.

These results highlight the clear advantage of leveraging DPUs to offload media processing in high-throughput, low-latency edge environments. By freeing up CPU resources, such offloading strategies enable the server to host additional applications or scale to handle more concurrent streams (especially for the decoding step, which is a light computation with data movement, making it the ideal task for a DPU), without compromising real-time performance.

In addition to CPU utilization, we measured the end-to-end latency of the video-processing pipeline across the same set of offloading scenarios. Figure 3.5 reports the latency distribution observed between the transmission of a video frame to the remote server and its reception after decoding, re-encoding, and retransmission. The results show that offloading strategies not only affect server resource usage but also directly impact end-to-end latency.

For the *medium* preset, the results show that *Decode Offloading* achieves lower latency than the *No Offloading* configuration. This indicates that offloading the decoding stage to the DPU already provides tangible latency benefits, even when the encoding remains on the host. Since decoding is largely dominated by data movement and control operations, delegating this task to the DPU reduces host-side contention and scheduling overhead, resulting in improved end-to-end latency. In contrast, performing both

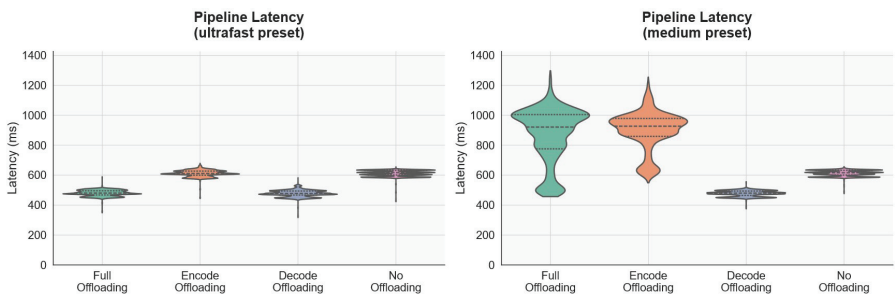

Figure 3.5 Latency of the end-to-end video stream under different offloading scenarios for the two H.264 encoder preset configurations.

decoding and encoding on the server increases latency despite comparable encoder settings.

In the *ultrafast* preset, the relative differences between scenarios are reduced, as the encoding complexity is significantly lower. In this case, *Full Offloading* achieves the lowest latency, while *Encode Offloading* and *Decode Offloading* exhibit similar distributions, and *No Offloading* remains slightly higher. This suggests that when encoding complexity is minimized, host-side processing becomes less dominant, and the benefit of partial offloading is less pronounced.

Overall, these results demonstrate that the impact of offloading on latency strongly depends on the encoder configuration. While full offloading is most effective for latency-critical scenarios with relaxed quality constraints, selectively offloading decoding can already yield meaningful latency reductions when higher-quality encoder presets are required.

3.4.2 TCP vs RDMA

While both TCP and RDMA are viable transport protocols for DPU offloading, RDMA delivers significantly higher performance and is well-suited to this edge datacenter environment. To demonstrate this advantage, we compare the CPU usage of TCP and RDMA implementations, as shown in Figure 3.6.

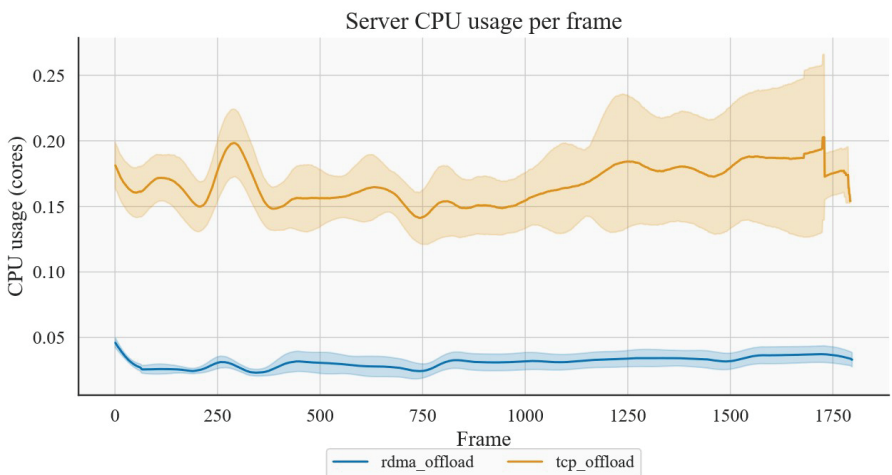

Figure 3.6 CPU usage of the "Full Offloading" scenario over multiple runs, comparing when RDMA or TCP is used as a network protocol.

The results demonstrate a high contrast: TCP transport exhibits high CPU consumption with significant fluctuations, whereas RDMA imposes negligible computational overhead on the server. This near-zero CPU burden makes RDMA the preferred choice for resource-constrained edge environments where server CPU cycles must be preserved for application workloads.

3.5 Conclusion

In conclusion, we show that RDMA can be transparently integrated into GStreamer pipelines via our custom plugin, rather than relying on TCP. This makes it easier to leverage external accelerators, thereby reducing server CPU usage in controlled environments, such as edge datacenters, as shown in our benchmarks. On the other hand, we demonstrate this use case by leveraging BlueField-3 DPUs to offload video decoding and encoding for a virtual try-on application.

Beyond the reduction in server CPU usage demonstrated in our benchmarks, our results highlight the importance of adapting the offloading strategy to the target application requirements. When low end-to-end latency is the primary objective and visual quality can be relaxed, fully offloading both decoding and encoding to DPUs represents an efficient solution, as it minimizes host involvement and reduces processing delay. Conversely, for use cases where output quality is critical, keeping the encoding stage on the host while offloading decoding to the DPU appears to be a better trade-off. In this configuration, the computationally intensive but quality-sensitive encoding process can leverage more advanced encoder settings on the server, while the DPU efficiently handles decoding, which is largely dominated by data movement rather than computationally intensive tasks.

These observations suggest that DPU-based media pipelines should not rely on a single fixed offloading configuration but instead dynamically adapt encoding and offloading strategies according to application-level constraints, such as latency, visual quality, and available computational resources. This flexibility is particularly relevant for edge deployments, where heterogeneous workloads and fluctuating requirements are common.

Acknowledgements

This work was partly funded by the QUARC project by the European Union Horizon Europe research and innovation program within the framework of Marie Skłodowska-Curie Actions with grant number 101073355, and by the

grant PID2021-123041OB-I00 funded by MCIN/AEI/10.13039/5011000110 33 and by "ERDF A way of making Europe".

The demo application was tested in the CNIT testbed located in Pisa, Italy, which allowed us to use their RoCE infrastructure

References

[1] M. Guaitolini, A. Khan, E. Rouzic, F. Paolucci, and F. Cugini, "Virtual Try-On Application leveraging RoCE in Low-latency Edge Computing Networks," in Proc. 2024 24th International Conference on Transparent Optical Networks (ICTON), Bari, Italy, Jul. 2024, pp. 1–4, doi: 10.1109/ICTON62926.2024.10647389.

[2] GStreamer Project, "GStreamer Multimedia Framework," [Online]. Available: https://gstreamer.freedesktop.org/. Accessed: Nov. 2025.

[3] Z. Cao, G. H. Martinez, T. Simon, S. Wei, and Y. A. Sheikh, "OpenPose: Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2019.

[4] T. Simon, H. Joo, I. Matthews, and Y. Sheikh, "Hand Keypoint Detection in Single Images using Multiview Bootstrapping," 2017.

[5] Z. Cao, T. Simon, S.-E. Wei, and Y. Sheikh, "Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields," 2017.

[6] S.-E. Wei, V. Ramakrishna, T. Kanade, and Y. Sheikh, "Convolutional Pose Machines," 2016.

[7] B. Wu and F. Zhao, "2D-Human-Parsing," 2021. [Online]. Available: https://github.com/fyviezhao/2D-Human-Parsing.

[8] J. Kim, G. Gu, M. Park, S. Park, and J. Choo, "StableVITON: Learning semantic correspondence with latent diffusion model for virtual try-on," in Proc. *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024, pp. 8176–8185.

[9] A. Kashyap, Y. Li, D. Ng, and X. Lu, "Understanding the Idiosyncrasies of Emerging BlueField DPUs," Proc. *39th ACM International Conference on Supercomputing*, 2025, pp. 807–821. doi: 10.1145/3721145.3725780.

[10] Y. Li, A. Kashyap, W. Chen, Y. Guo, and X. Lu, "Accelerating Lossy and Lossless Compression on Emerging BlueField DPU Architectures," in *2024 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, May 2024, pp. 373–385. doi: 10.1109/IPDPS57955.2024.00040.

[11] G. Robaina and A. Fiorese, "GOTE: An Edge Computing Architecture for Mobile Gaming," Proc. *25th International Conference on Enterprise Information Systems - Volume 1: ICEIS*, 2023, pp. 721–730. doi: 10.5220/0011962000003467.

[12] H. Lee and J. Jeon, "Accelerating Image Processing on FPGAs using HLS and PYNQ," 2020, pp. 1–2. doi: 10.1109/ICCE-Asia49877.2020.9277085.

[13] M. Salehe, Z. Hu, S. H. Mortazavi, I. Mohomed, and T. Capes, "VideoPipe: Building Video Stream Processing Pipelines at the Edge," Proc. *20th International Middleware Conference Industrial Track*, 2019, pp. 43–49. doi: 10.1145/3366626.3368131.

[14] X. Han, Z. Wu, Z. Wu, R. Yu, and L. S. Davis, "VITON: An Image-Based Virtual Try-on Network," in 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2018, pp. 7543–7552. doi: 10.1109/CVPR.2018.00787.