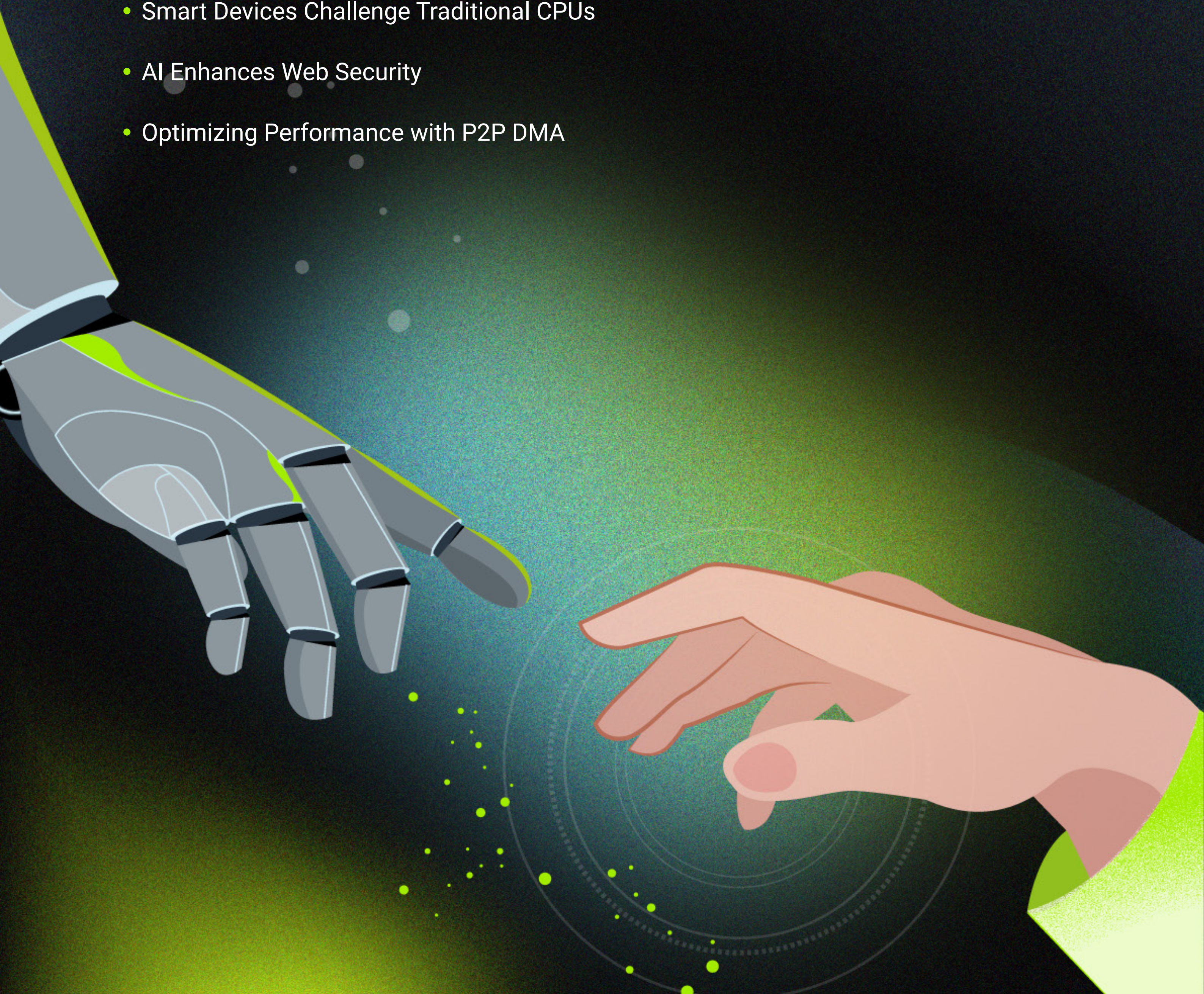
codilime

AI-based Web Application Firewall

Using P2P DMA between NVIDIA GPU and DPU

- Smart Devices Challenge Traditional CPUs
- AI Enhances Web Security
- Optimizing Performance with P2P DMA



Executive Summary

This publication delves into the revolutionary shift in computer architecture driven by the rise of smart devices with their own computational capabilities, reducing the central role traditionally held by the CPU. As technology advances, devices like DPUs (data processing units) and GPUs (graphics processing units) are becoming more autonomous, handling specific workloads more efficiently than general-purpose CPUs.

Key points covered in this ebook include:

1. **Traditional vs. Modern Architectures**: A comparison between the hierarchical, CPU-centric architecture and the emerging democratic model where smart devices independently manage their own tasks.

2. **Leveraging AI for Security**: An exploration of how AI models deployed on GPUs can enhance security in web applications, bypassing the CPU to streamline data processing and improve performance.

3. **Practical Applications and Future Potential**: Examples of practical implementations, including a proof of concept for a firewall application using DPUs and GPUs. Future possibilities for using P2P DMA in various fields, such as network monitoring, media streaming, and content delivery, are also discussed.

4. **Solution Architecture and Improvements**: A detailed description of the architecture used in the proof of concept, highlighting the roles of DPUs, CPUs, and GPUs. Potential future improvements and a vision for developing a more flexible and programmable framework for ML and DPU-based solutions are outlined.

This publication provides valuable insights for technology professionals, emphasizing the shift towards smarter, more autonomous hardware that enhances performance, security, and scalability in modern computing environments.

Table of contents

Introduction

For most of computing history, the architecture of a computer remained fixed and largely unchallenged. At the core of the computer, there was a CPU that manages computations, RAM for holding temporary data, a disk for storing permanent files, and a GPU for performing parallel computations faster. Everything was designed in a very hierarchical and autocratic manner - various devices performing their tasks were overseen and managed by the CPU. Now, however, it is time to challenge these hitherto timeless concepts and introduce more democracy.

As technology continues to advance, all devices are becoming smarter. Nowadays, many hardware devices no longer require a host CPU to manage them as they possess their own computational capabilities. The CPU is a circuit that is very well suited to a certain type of task - a structural, sequential one - but doesn't do as well with other workloads. It is more beneficial for performance when running networking-related computations on a DPU and AI on a GPU. When the CPU only exchanges data or synchronizes operations between different pieces of hardware that handle all of the substantive work, it is faster to take it out of the equation altogether. Many cases, commonly seen throughout the technological landscape, could benefit greatly from such a design.

A Novel Approach

Let's consider the following scenario: a simple HTTP web application is deployed and we want to protect it using a firewall. The application has a significant user base and handles a lot of traffic. The popularity of the application creates another problem - its database is a valuable target for a cyber attack.

Given the recent advancements in machine learning development, we aim to harness its power to enhance security measures. Instead of relying on static firewall rules or implementing algorithmic heuristics, let's employ a machine-learning model!

Artificial intelligence models require a lot of parallel computing to be carried out. A graphics card is well suited to this type of task. By deploying artificial intelligence on a GPU, we can enable it to scrutinize every packet and distinguish between benign and hostile traffic. The classic approach to this would consist of the following steps:

- Read each packet from the NIC using the host CPU[1]
- Route it to the model on the GPU
- Wait for the computations on the graphics card to be completed
- According to the verdict from the GPU, either drop it or forward it to the app

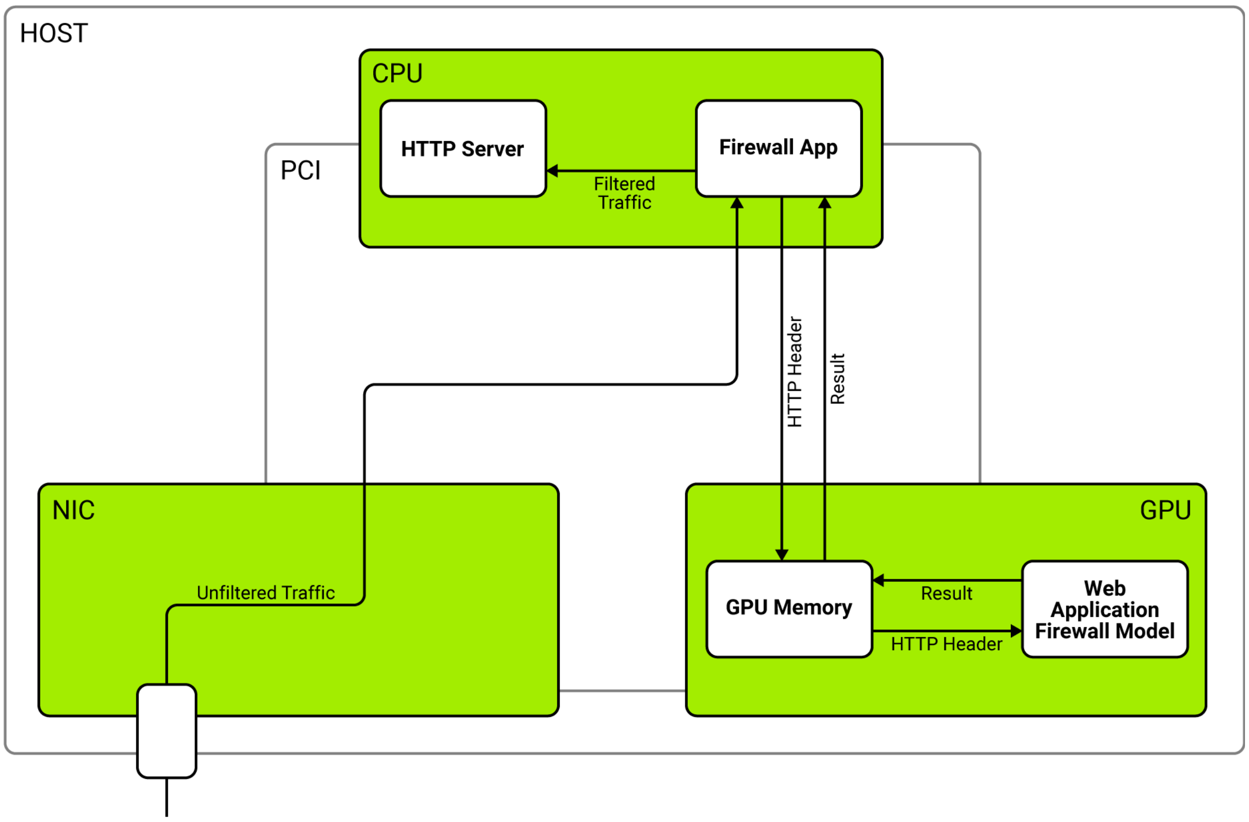


Fig. 1. The diagram shows the classic approach to implementing a GPU-based firewall with a normal network interface card.

[1] Normally, the NIC first transfers the packet to the CPU RAM and then the CPU application reads the packet data. This step is skipped in our description in order to simplify the project description.

This requires several data transfers. First, we have to send the data from the NIC to the CPU, then to the GPU, then back to the CPU. Each of these transfers takes time.

Bypassing the host CPU in this process can enhance security by isolating the host from potential malicious data, and performance by eliminating unnecessary memory transfer and packet processing on the host. The general idea is to use P2P DMA to transfer data directly from the NIC to the GPU and use the peripheral devices' own computational capabilities to manage the necessary control logic.

The NVIDIA BlueField-2 DPU has an ARM CPU of its own that is perfectly suited to this type of scenario. Because it is closer to the network and designed to process network data, it is going to be faster than a general-purpose CPU. Additionally, the NVIDIA graphics unit that we used can schedule work by itself using the Graph API, freeing the CPU from the control workload relating to the GPU. Because we used a BlueField variant without an onboard GPU, the system is more flexible and can be upgraded in subsets if necessary. It could also be modified to include multiple GPUs.

Leveraging P2P DMA, selected packet data is sent directly from the DPU to the GPU, where an AI model decides whether to drop or forward the packet, streamlining the DPU's firewall functionality. This approach relieves the host CPU of its workload entirely, enabling it to be used for other computing workloads.

The created solution stands as a proof of concept, demonstrating the possibilities that are unfolding in the era of smart peripheral devices. As smart peripheral devices advance, offering substantial opportunities for acceleration, offloading, and isolation from the main system, the critical significance traditionally assigned to the CPU may diminish. What could be achieved using, for instance, a smart disk? Only time (and our future research) will tell.

Solution Architecture

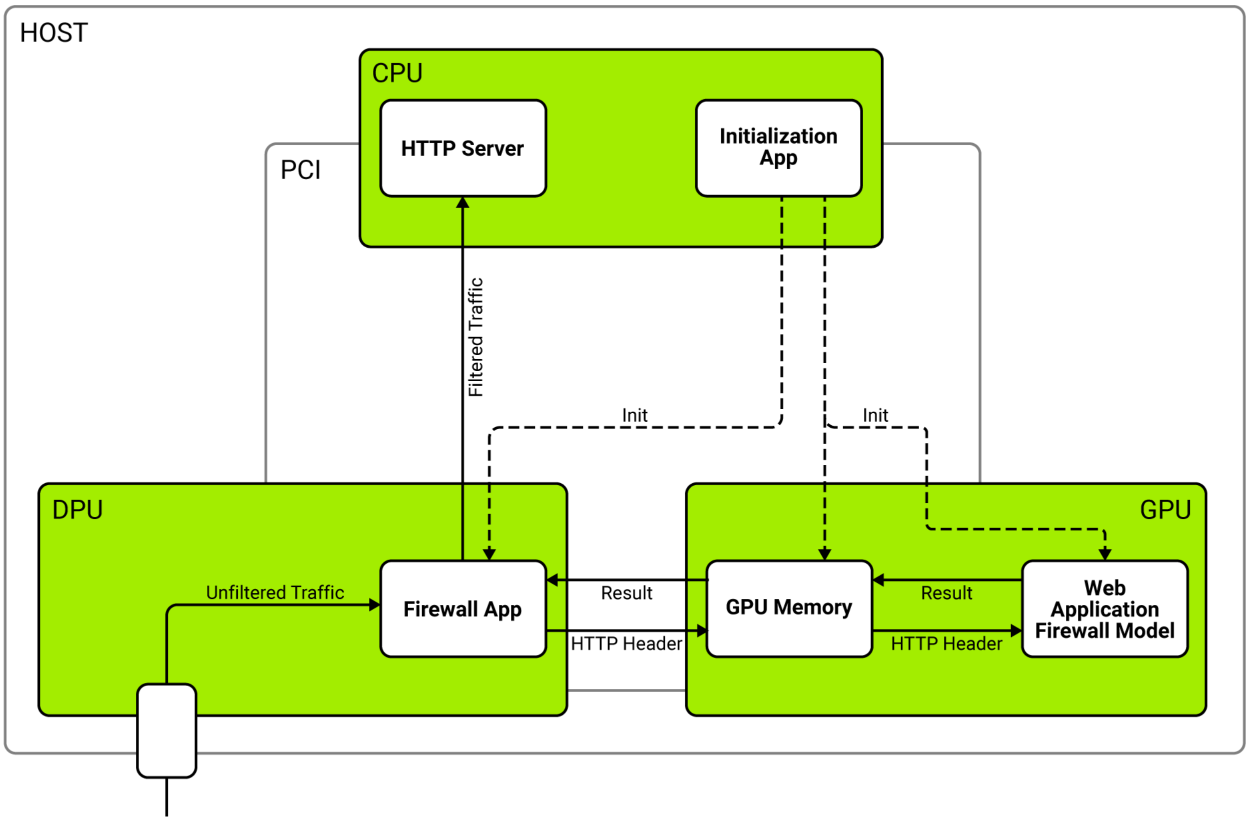


Fig. 2. The diagram shows the novel approach to implementing a GPU-based firewall. The solution uses a DPU for receiving traffic instead of a typical NIC.

The architecture can be split into three areas: DPU, CPU, and GPU.

On the CPU, there are HTTP Server and the Initialization app. The HTTP Server handles requests coming from clients. The traffic coming to the server must first be filtered to ensure the requests are secure.

The Initialization app controls the initialization process for the GPU and DPU. This consists of several steps:

- **GPU memory initialization**: The CPU configures common buffers for the DPU and GPU. The DPU will use these buffers to send extracted packet data to the GPU. The GPU will use these buffers as input (PyTorch tensors) to the machine learning model.
- **Web application firewall (WAF) model initialization:** The CPU prepares the model for receiving packet data. The goal is to minimize the CPU workload as much as possible. To achieve that, we used PyTorch CUDA Graphs support, which allowed us to store the PyTorch computation model in the CUDA Graph structure. This allowed us to use CUDA Graphs tail launch to schedule model computation directly from the GPU and fully process packet data batches on the GPU. Thanks to that, the CPU doesn't need to control the ML model execution.

Besides packet batch processing, the GPU side is responsible for the WAF model execution. We used the mobilebert-sql-injection-detect model to detect SQL injection, but we don't see major obstacles in using different models. The main requirement is that the model's input should be able to be filled by the DPU via memory operations.

The DPU firewall app is responsible for network traffic processing – receiving packets, extracting the selected portion of packet data, and writing it to the GPU memory via DMA. After the GPU WAF model execution, the firewall app either drops the packet or forwards it to the host HTTP server. The DPU lets us extract the important parts of the packet data (in our case, the HTTP header) before writing it to the GPU memory. This is the main advantage of using a DPU instead of a regular network card supporting GPUDirect RDMA technology. Sending parts of the packet only, instead of the whole packet, to the GPU reduces the PCI traffic between the DPU and GPU, improving performance and latency.

Summing up, the typical packet flow is the following:

1. A packet arrives at the DPU firewall app.
2. The DPU extracts the HTTP request data from the packet.
3. The extracted data is sent via a DMA write operation to the shared GPU memory buffer.
4. On arrival, the GPU runs the WAF model on the received data.
5. After the WAF model execution, the results are read by the DPU Firewall app via the DMA read operation.
6. Based on the results, the DPU decides whether to drop the packet or send it on to the host HTTP server.

Note that the CPU is idle during firewall traffic processing and can focus on HTTP request handling or other tasks.

Possible Improvements to the Solution

The main focus of the project was on preparing a proof of concept that demonstrates the possibilities of utilizing the DPU and GPU for traffic control. There are a number of potential improvements to the use case, and some of them are described in the subsequent sections.

Advanced Traffic Preprocessing and Postprocessing

As stated in the description of the solution architecture, DPUs, unlike regular NICs, can perform packet preprocessing (in our case, HTTP header extraction) before sending the data to the ML model, saving bandwidth and reducing latency. This idea can be used for other types of computation that can be executed before and after model execution.

A more advanced example of this approach we came up with but haven't implemented was extending the DPU application to process traffic streams similarly; e.g. after recognizing an attack, we could block the entire stream instead of operating on individual packets, which would again greatly improve performance. This could be achieved by incorporating the DOCA Flow mechanism.

Creating a Framework

In our project, we chose a specific ML model and integrated it with the rest of the application. However, we see great potential in simplifying the creation of ML and DPU-based solutions for arbitrary models.

There are multiple use cases that could utilize ML for traffic processing. For example, in the case of network traffic control, the solution could be used to detect and block DDoS attacks. Another example, from a slightly different field, could be monitoring network traffic based on ML models and showing processed statistics. These use cases require different models that may require greatly different input data and would produce results that should be interpreted and processed differently. It would be beneficial to implement a framework that could simplify the programmability of such various solutions.

Vision for the Future

Using P2P DMA to transmit data from the DPU to the GPU and back is only one example of utilizing this technology. This is a wide field with a lot of potential in other areas as well.

Applications that rely on storage are another example of how transferring large amounts of data bypassing the CPU could improve the performance of the whole solution. For instance, in an app that realizes content delivery network functionality, the data can be sent directly from the disk to the network interface card, saving CPU cycles. The same could be said for various solutions that have a storage component, such as a media streaming service. The control logic surrounding data transfers could be managed by the computational capabilities of the smart devices themselves, and the DMA could be realized independently of the rest of the system.This kind of application would naturally require a smart storage device that could initiate and realize a DMA operation. Depending on the computational power of the disk device, it might also be possible to create a simple database that would run on the disk itself and make itself available through the network for direct access.

A monitoring/data-gathering solution could be another option for utilizing P2P DMA for the NIC and disk. The network traffic could be mirrored directly to the storage device, which would introduce performance gains. This kind of solution could be used, for instance, to gather data to train AI models for network monitoring and security.

Caching intermittent or final results of machine learning computations could also leverage P2P DMA between the GPU and disk. In a scenario where it is likely that the same data will be processed multiple times by the GPU and the output is sizable, this could potentially be faster than just running the AI model on the same data again. Additionally, by introducing parallelism, the solution could run the next batch of data on the GPU and fetch the cached results from the disk at the same time, vastly increasing performance.

Conclusions

P2P DMA could improve the efficiency of many applications by utilizing hardware more efficiently. Because multiple processes happen simultaneously on different devices, not only is the data transfer faster, but it also introduces opportunities for parallelism, further improving the overall performance.

About the Authors

Marcin Parafiniuk
Software Engineer

Marcin Parafiniuk is a Software Engineer at CodiLime, where he focuses on firmware and driver development for SmartNICs. He also participates in the creation of next-generation networking software and hardware, including integration of SmartNICs with the Tungsten Fabric SDN platform and implementation of hardware offloading for 5G infrastructure. He specializes in the Rust, C/C++, and P4 programming languages. Marcin earned a bachelor's degree in informatics at the University of Warsaw (Poland).

Michal Niciejewski
Software Engineer

Michał is a Software Engineer at CodiLime, specializing in network drivers and traffic offloading. He excels in the C and C++ languages and has expanded his skills in Rust through projects at CodiLime. His experience includes designing and implementing network drivers and applications in DPDK, integrating SmartNICs with Tungsten Fabric SDN, and implementing packet processing offloading on GPUs. Michał's passion for operating systems and eagerness to learn new technologies drive his contributions in the field.

Artur Jaworski
Senior Software Engineer

Artur Jaworski is a Senior Software Engineer at CodiLime with over ten years of experience in financial and networking technologies. With languages like C/C++, Rust, Python and P4, he has participated in the development of several solutions for data plane applications, SDN controllers, 5G infrastructure and performance measurement tool sets. As a technical leader in the Acceleration and Offloading Business Unit, he manages the group that extends competencies and sets the direction for projects with acceleration and offloading technologies.

About CodiLime

Since 2011, CodiLime has been the engineering partner of choice for semiconductor companies, networking vendors, telecom services, and software solution providers.

We're home to 300 top-notch software developers, network engineers, DevOps experts, and solution architects. We appreciate long-term collaborations above all, as well as our partners. Below are some of the names that have already trusted us:

CISCO paloalto NETWORKS EQUINIX

JUNIPER NETWORKS NUTANIX CLOUDIFY

SELECTOR FluxNinja FREEDOMFI

AT&T NVIDIA GIGASPACES

MESH7 neptune.ai

CodiLime aims to link network engineering talent with business domain expertise – we focus on five N.E.E.D.S. - Networks, Equipment, Environment, Data and Security.

contact@codilime.com