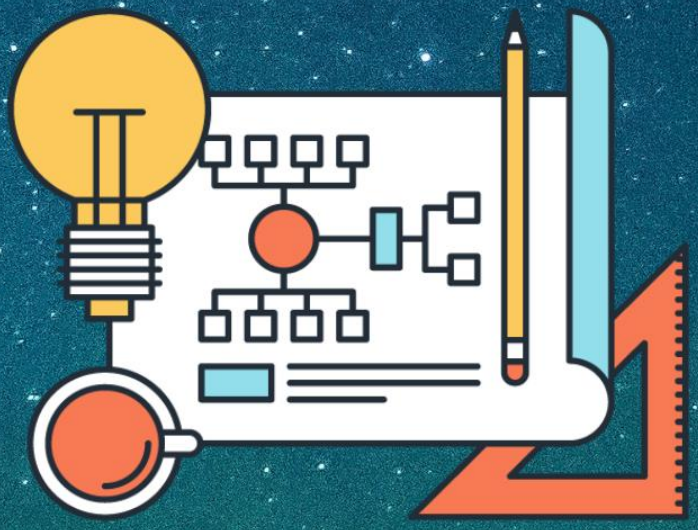
DIPL.ING.(FH)KLAUS ROCK

HTTP-QuSS

HTTP - QUANTUM SPEED AND SECURITY

February 14, 2022

THE ALGORITHM



ROCK TECHNOLOGIES
Bonhoefferstr. 37 | 73432 Aalen | Germany | +49-7367-9222-958

Table of Contents

1.0 Non-Disclosure Notice

This Document contains internal and confidential Information from Dipl.-Ing. (FH) Klaus Rock. Third Parties may not access this Work without the express Permission of Dipl.-Ing. (FH) Klaus Rock. The Information contained in this Document may not be disclosed publicly for two Years following the mutual signed NDA unless otherwise expressly approved by Dipl.-Ing. (FH) Klaus Rock.

2.0 Patent Notice

Dipl.-Ing. (FH) Klaus Rock is the Inventor and Patent Applicant for this new **HTTP-QuSS** Technology mentioned within this Documentation.

Name of the Invention

METHOD AND SYSTEM FOR DATA TRANSMISSION WITH SIGNIFICANTLY REDUCED LATENCY LOSSES

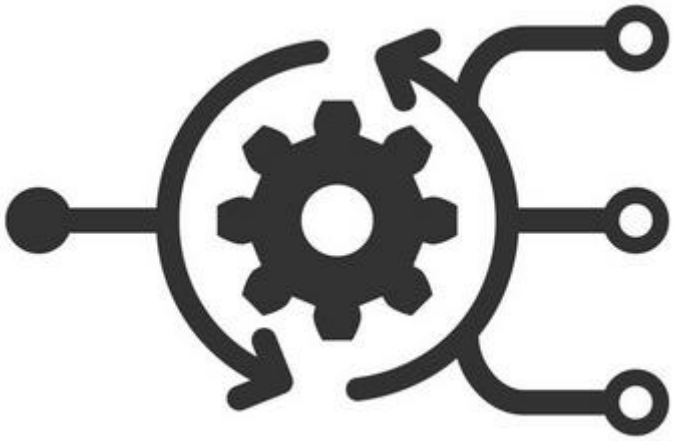
APPLICATION NUMBER	PRIORITY	APPLICANT	COUNTRY
EP 19189655.4	01 August 2019	Rock, Mr. Klaus	Germany
PCT 19189655.4	01 August 2019	Rock, Mr. Klaus	All WTO Member States

Dipl.-Ing.(FH) Computer Science Klaus Rock
Bonhoefferstr. 37
73432 Aalen – Germany
Tel: +49-7367-9222-958
E-Mail: k.rock@rock-technologies.com

3.0 Preface - Algorithms in Computer Science

3.1 Definition

An algorithm is a prescription for solving a class of problems. It consists of a finite sequence of steps with which new output data can be uniquely calculated from known input data.



3.2 Properties

- The definition shows the close relationship with functions.

  (***Input Data*** → ***Output Data***).

- The definition binds the algorithm to solving a class of problems rather than a single task.

3.3 Mathematical Function

- The definition of an algorithm does not contain any requirements for the practical executability of the algorithm on a real machine (computer).

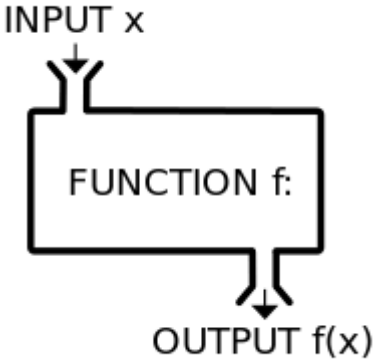
An algorithm can be understood as a function (mathematical definition).

y=f(x) or f: D → Z, x →y

(D: Definition Quantity, Z: Target Quantity)

In mathematics, a function or map is a relationship between two sets that assigns to each element of one set (function argument, independent variable, x-value) exactly one element of the other set (function value, dependent variable, y-value, or f(x) value).

$$f : x \mapsto f\ (x)$$



Interpretation:

- Elements from sets: input and output
- The elements can also be sets, vectors, etc.

Thus, an algorithm is assigned to a function.

However, there are functions to which no algorithm can be assigned. These functions are therefore not calculable.

Definition:

A function f is called computable if there is an algorithm that calculates the function value ***f(x)*** for each argument ***x***.

3.4 Characteristic Features

3.4.1 Finitude

An algorithm must consist of a finite number of solution steps and, after processing these finite many steps, reach the end after a finite time.

3.4.2 Uniqueness

The individual steps of an algorithm and their sequence must be clearly described.

3.4.3 Universality

An algorithm must not only describe the solution of a specific problem (e.g. solution of equation $x^2 + 2x + 1=0$), but must describe the solution of a class of problems (e.g. the solution of all quadratic equations $ax^2 +bx +c =0$).

3.4.4 Determinacy

The repeated application of the algorithm with the same input data must always provide the same output data.

3.4.5 Efficiency

An algorithm must use as few resources as possible for a machine, i.e. as little computing time and memory as possible.

In computer science, efficiency measures for algorithms, the so-called time and memory complexity, have been developed, with which algorithms can be evaluated. This is very important in practice since several algorithms can usually be specified to calculate a function.

3.5 Representation

In addition to the notation of individual elementary actions/instructions, such as
- Enter the values of the coefficients a,b,c

logical elementary structures are also necessary, which represent the temporal sequence, such as the sequence of instructions, such as

1. Enter the values of the coefficients *a,b,c*
2. Calculate *d = b*b – 4*a*c*

but also, the selection

3. If *d<0* continue with step 7

When representing algorithms, the focus is primarily on the representation of the temporal sequence, the so-called control flow, and not on the representation of the individual elementary instruction itself.

Theorem by Böhm and Jacopini

With only three logical elementary structures

- Sequenz
- Selektion
- Zyklus

any algorithmic problem can be represented.

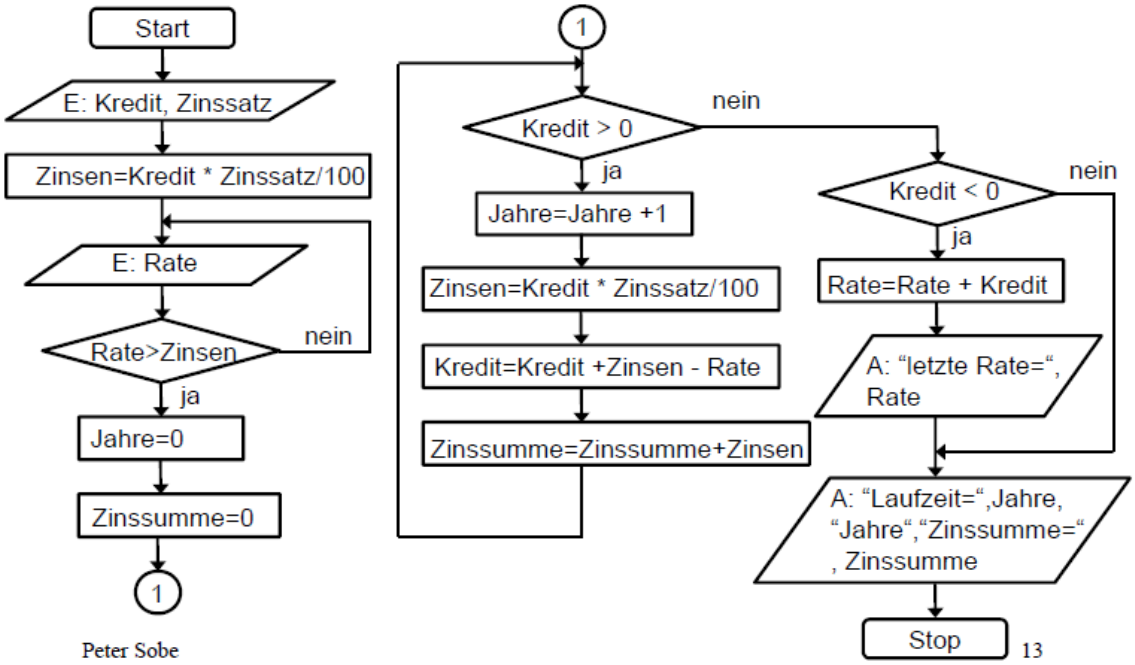
Two different forms of representation have been developed, which are mainly in use and use graphic symbols:

- **Flow chart** according to DIN 66001 (vgl. 3.5.1)
- **Structure chart** according to Nassi and Shneiderman according to DIN 66261 (vgl. 3.5.2)

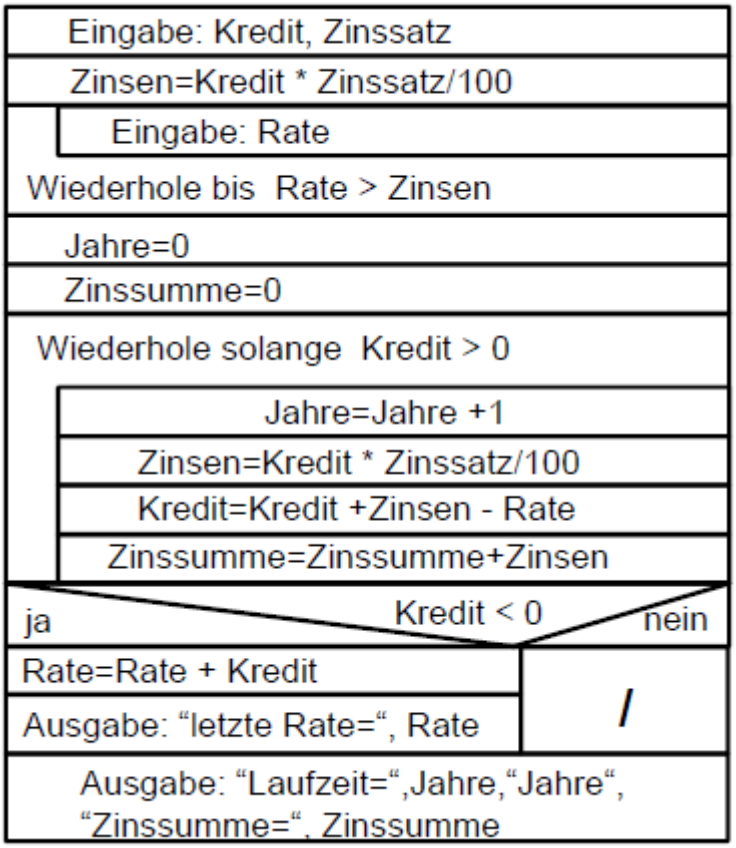
Both represent a so-called syntax, which formally specifies the expressions of the description.

The meaning of the representation is called semantics.

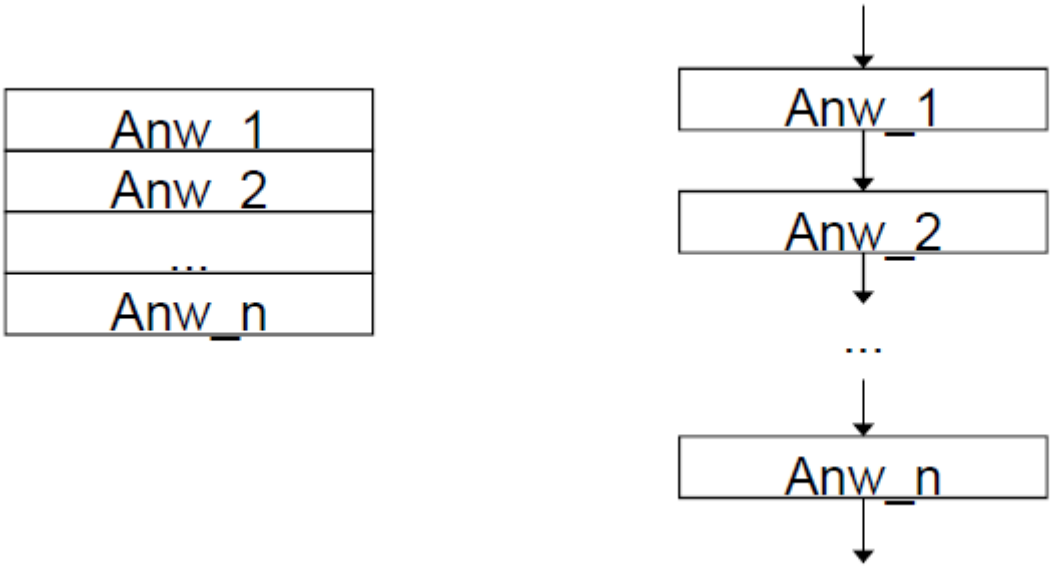
3.5.1 Flow Chart



3.5.2 Structure Chart



3.5.3 Sequence



The essence of a sequence is that it represents only a single action/instruction to the outside, although internally it consists of a sequence of instructions (*Anw*).

4.0 The HTTP-QuSS Algorithm

4.1 The Function

4.1.1 State of the Art Latency TCP Bandwidth Problem

Available TCP Bandwidth depends on Distance expressed in Round Trip Time measured in Milliseconds ms.

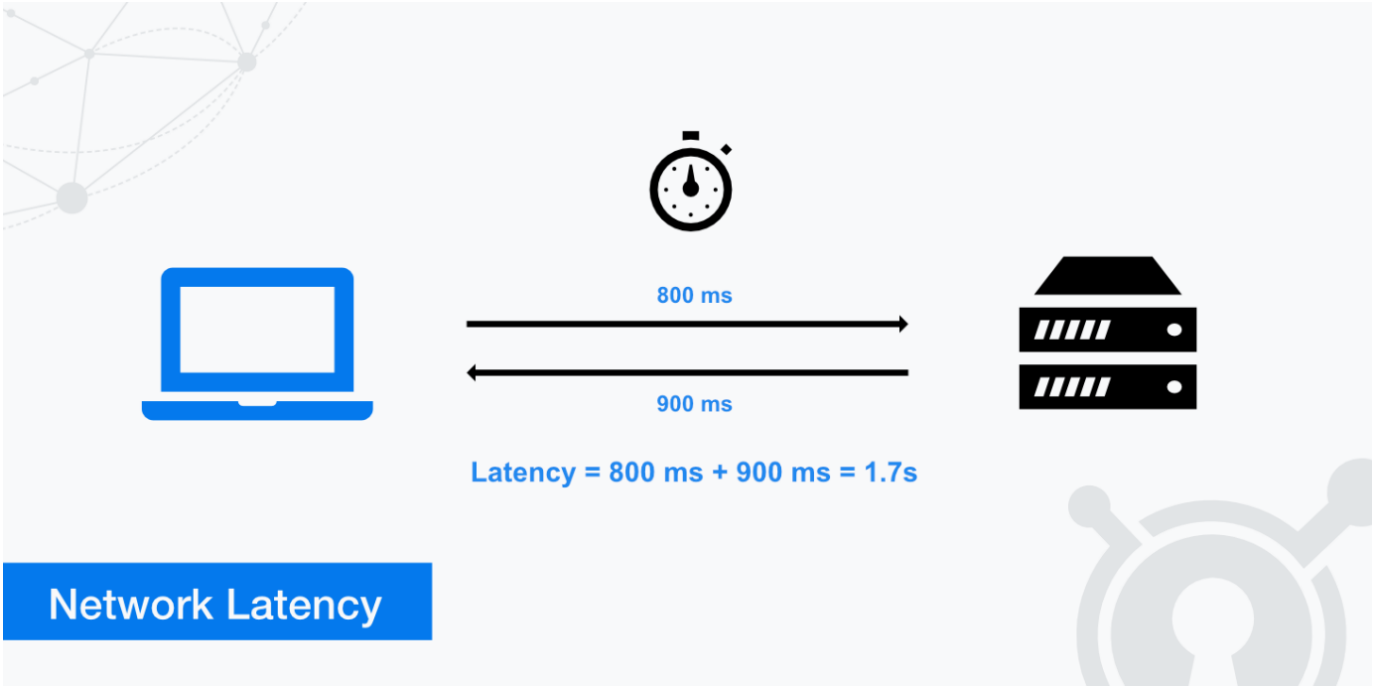
f_{TCP}: RTT_{TCP} → Mbit/s $_{TCP}$ = The Function f_{TCP}: forms from **RTT_{TCP}** the Quantity of **Mbit/s $_{TCP}$**
Mbit/s $_{TCP}$ is functionally dependent on **RTT_{TCP}**

With simple words: Distance (***RTT***) destroys available TCP Bandwidth.

4.1.2 Network Latency Definition

Network Latency refers to the time and/or delay involved in transmitting data over a network. In other words, how long it takes for a data packet to go from the IP transmitter to the IP receiver and back again.

Network Latency is measured in RTT/ms (Round Trip Time in Milliseconds).



4.1.3 The Network Latency is caused by

4.1.3.1 Transport Medium Propagation Speed

Mv = km/s

M = Electricity, Radio, Light
v = Speed in km/s

4.1.3.2 IP Distance between Transmitter and Receiver

Dip = km

D = Distance
ip = Internet IP Address

4.1.3.3 HOP Packet Queue + Processing Time

$HOPqpt$ = ms

HOP = Network nodes (router, switch) between two network segments
qpt = HOP queue + processing time per data packet

4.1.3.4 IP Sender/Receiver Network Card Queue

$NICqt$ = ms

NIC = Network Adapter
qt = Queue time

4.1.3.5 IP Transmitter/Receiver CPU Packet Processing Time

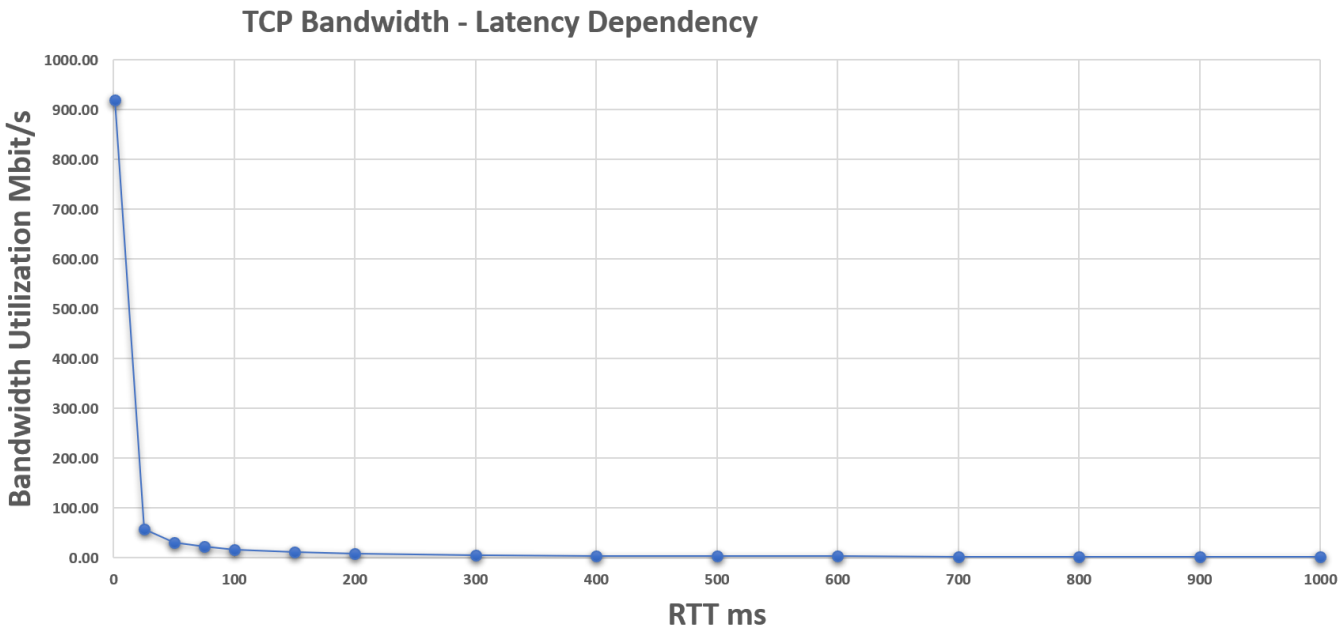
$CPUt$ = ms

CPU = Central processing unit
t = ms

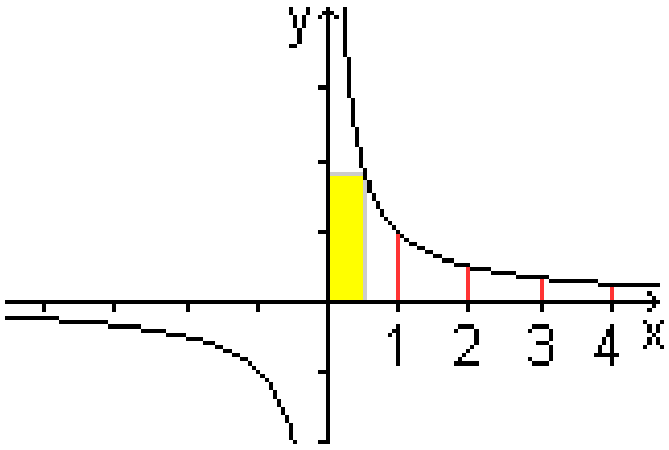
4.1.4 The Network Latency Formula

$$RTT_{ms} = 2 * \left(\sum_{1}^{n} \frac{Dip_x}{Mv_x} + \sum_{1}^{n} HOPqpt_x + 2 * NICqt + 2 * CPUt \right)$$

4.1.5 Correlation between Network TCP Bandwidth and Network Latency RTT



A simple Hyperbel Function



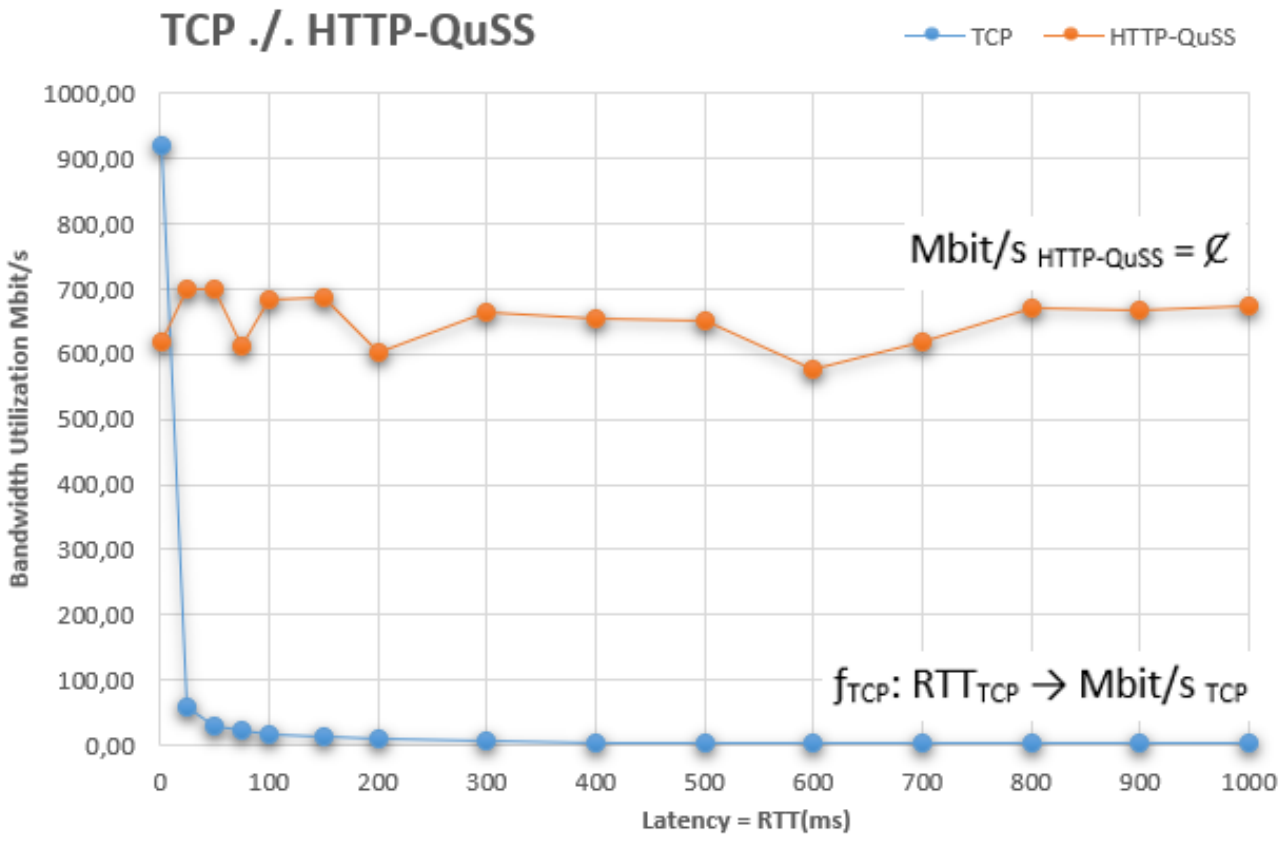
$$\mathbf{f(x) = \frac{1000}{RTT} + 10}$$

4.2 The new Algorithm for Quantum like TCP Bandwidth Speed

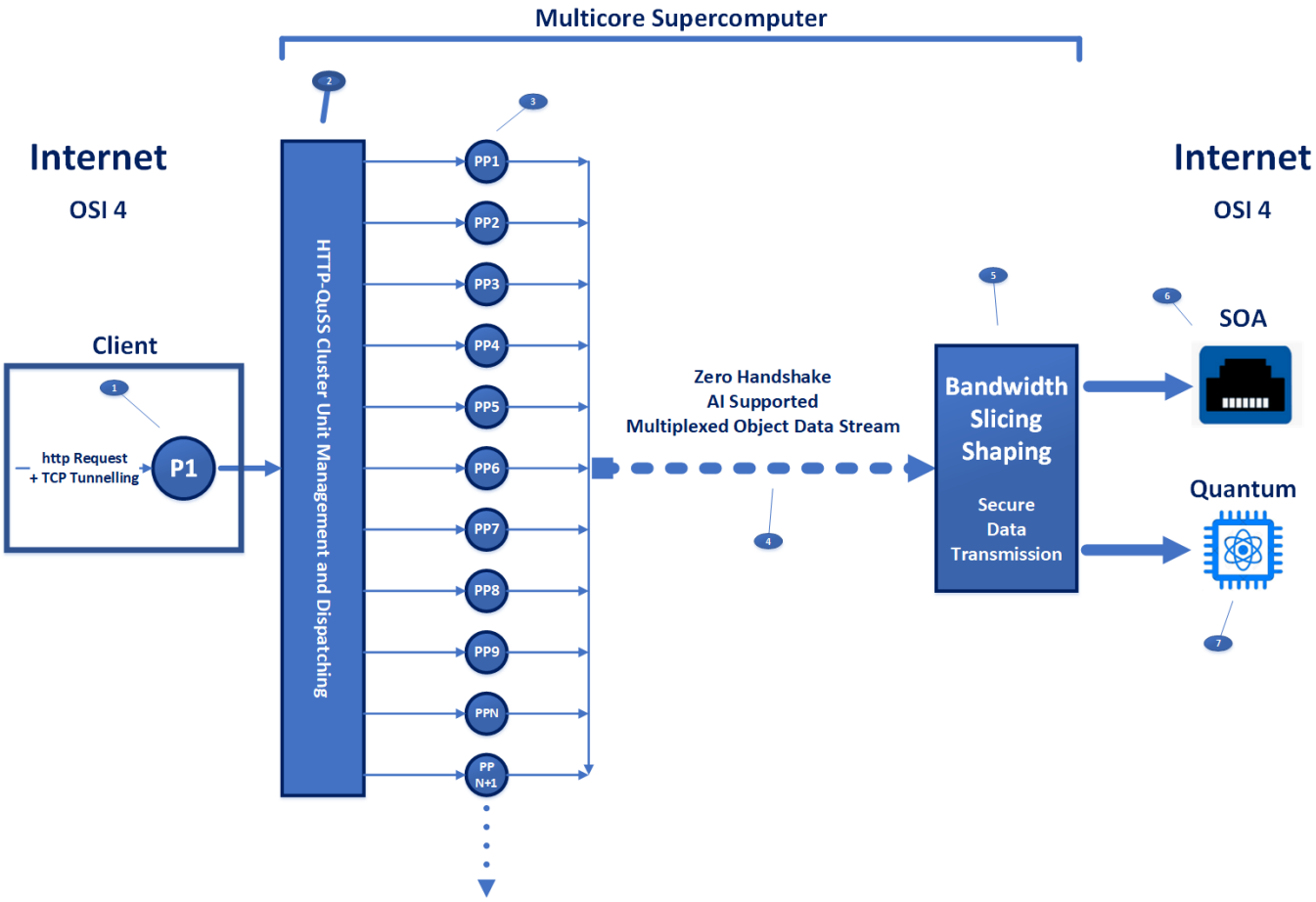
Available TCP Bandwidth does not depend on Distance and is always constant.

Mbit/s $_{HTTP\text{-}QuSS}$ = ∠ = **Mbit/s $_{HTTP\text{-}QuSS}$** is always constant
There is not a functionally dependency on **RTT_{TCP}**

With simple words: No TCP Bandwidth Losses caused by long Distances.



4.3 Sequence Display



The HTTP-QuSS Algorithm can be realized with now available powerful embedded Hardware and a new disruptive parallel Processing Chain.

Shortcuts:

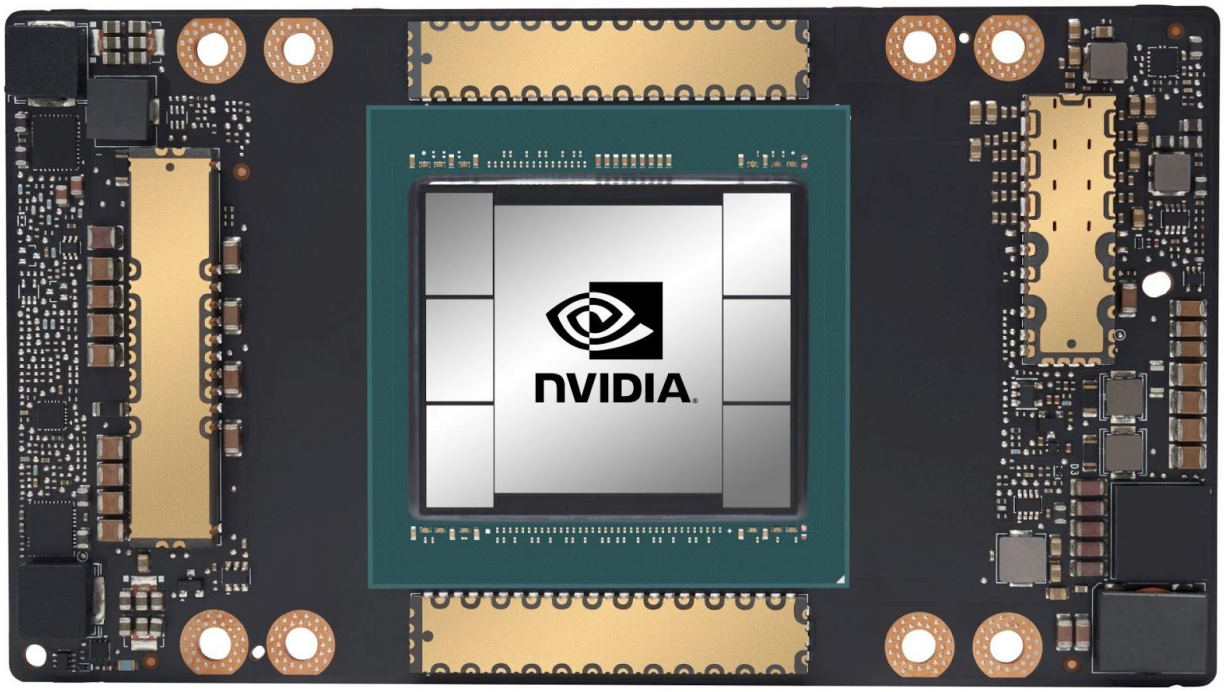
Internet OSI 4	= ISO/OSI-Network Reference Model Layer 4 TCP/UDP
PPX	= Parallel Software Processes
SOA	= Existing Internet - State of the Art
Quantum	= Future Quantum Internet

Legend:

1. Browser http Process Request including File Download Tunnel
2. Embedded Cluster Management and related Process Dispatching
3. Dispatched and executed parallel Processes
4. Generated multiplexed Object Stream
5. DPU supported Bandwidth Slicing, Shaping, and secure UDP Transmission
6. SOA-Router
7. Quantum-Router | Gateway

4.4 Suitable Embedded Hardware

4.4.1 NVIDIA Embedded Supercomputer A100



The NVIDIA A100 GPU is composed of multiple GPU processing clusters (GPCs), texture processing clusters (TPCs), streaming multiprocessors (SMs), and HBM2 memory controllers.

The full implementation of the A100 GPU includes the following units:

- 8 GPCs, 8 TPCs/GPC, 2 SMs/TPC, 16 SMs/GPC, 128 SMs per full GPU
- 64 FP32 CUDA Cores/SM, 8192 FP32 CUDA Cores per full GPU
- 4 third-generation Tensor Cores/SM, 512 third-generation Tensor Cores per full GPU
- 6 HBM2 stacks, 12 512-bit memory controllers

The A100 Tensor Core GPU implementation of the GA100 GPU includes the following units:

- 7 GPCs, 7 or 8 TPCs/GPC, 2 SMs/TPC, up to 16 SMs/GPC, 108 SMs
- 64 FP32 CUDA Cores/SM, 6912 FP32 CUDA Cores per GPU
- 4 third-generation Tensor Cores/SM, 432 third-generation Tensor Cores per GPU
- 5 HBM2 stacks, 10 512-bit memory controllers

4.4.2 NVIDIA Embedded Supercomputer H100

The NVIDIA H100 GPU is composed of multiple GPU Processing Clusters (GPCs), Texture Processing Clusters (TPCs), Streaming Multiprocessors (SMs), L2 cache, and HBM3 memory controllers.

The full implementation of the GH100 GPU includes the following units:

- 8 GPCs, 72 TPCs (9 TPCs/GPC), 2 SMs/TPC, 144 SMs per full GPU
- 128 FP32 CUDA Cores per SM, 18432 FP32 CUDA Cores per full GPU
- 4 Fourth-Generation Tensor Cores per SM, 576 per full GPU
- 6 HBM3 or HBM2e stacks, 12 512-bit Memory Controllers
- 60 MB L2 Cache
- Fourth Generation NVLink and PCIe Gen 5

The NVIDIA H100 GPU with SXM5 board form-factor includes the following units:

- 8 GPCs, 66 TPCs, 2 SMs/TPC, 132 SMs per GPU
- 128 FP32 CUDA Cores per SM, 16896 FP32 CUDA Cores per GPU
- 4 Fourth-generation Tensor Cores per SM, 528 per GPU
- 80 GB HBM3, 5 HBM3 stacks, 10 512-bit Memory Controllers
- 50 MB L2 Cache
- Fourth Generation NVLink and PCIe Gen 5

The NVIDIA H100 GPU with a PCIe Gen 5 board form-factor includes the following units:

- 7 or 8 GPCs, 57 TPCs, 2 SMs/TPC, 114 SMs per GPU
- 128 FP32 CUDA Cores/SM, 14592 FP32 CUDA Cores per GPU
- 4 Fourth-generation Tensor Cores per SM, 456 per GPU
- 80 GB HBM2e, 5 HBM2e stacks, 10 512-bit Memory Controllers
- 50 MB L2 Cache
- Fourth Generation NVLink and PCIe Gen 5

Using the TSMC 4N fabrication process allows H100 to increase GPU core frequency, improve performance per watt, and incorporate more GPCs, TPCs, and SMs than the prior generation GA100 GPU, which was based on the TSMC 7nm N7 process.

Note that the H100 GPUs are primarily built for executing datacenter and edge compute workloads for AI, HPC, and data analytics, but not graphics processing. Only two TPCs in both the SXM5 and PCIe H100 GPUs are graphics-capable (that is, they can run vertex, geometry, and pixel shaders).

4.4.2 NVIDIA DPU for Bandwidth Slicing | Shaping and SOC Client

4.4.2.1 What's a DPU?

Specialists in moving data in data centers, DPUs, or data processing units, are a new class of programmable processor and will join CPUs and GPUs as one of the three pillars of computing.

Of course, you're probably already familiar with the central processing unit. Flexible and responsive, for many years CPUs were the sole programmable element in most computers.

More recently the GPU, or graphics processing unit, has taken a central role. Originally used to deliver rich, real-time graphics, their parallel processing capabilities make them ideal for accelerated computing tasks of all kinds. Thanks to these capabilities, GPUs are essential to artificial intelligence, deep learning and big data analytics applications.

Over the past decade, however, computing has broken out of the boxy confines of PCs and servers — with CPUs and GPUs powering sprawling new hyperscale data centers.

These data centers are knit together with a powerful new category of processors. The DPU has become the third member of the data-centric accelerated computing model.

"This is going to represent one of the three major pillars of computing going forward," NVIDIA CEO Jensen Huang said during a talk earlier this month.

"The CPU is for general-purpose computing, the GPU is for accelerated computing, and the DPU, which moves data around the data center, does data processing."

So, a DPU is a System on a chip that combines:

- Industry-standard, high-performance, software-programmable multi-core CPU
- High-performance network interface
- Flexible and programmable acceleration engines

2.3.2.2 CPU v GPU v DPU: What Makes a DPU Different?

A DPU is a new class of programmable processor that combines three key elements. A DPU is a system on a chip, or SoC, that combines:

- An industry-standard, high-performance, software-programmable, multi-core CPU, typically based on the widely used Arm architecture, tightly coupled to the other SoC components.

- A high-performance network interface capable of parsing, processing and efficiently transferring data at line rate, or the speed of the rest of the network, to GPUs and CPUs.

- A rich set of flexible and programmable acceleration engines that offload and improve applications performance for AI and machine learning, zero-trust security, telecommunications, and storage, among others.

All these DPU capabilities are critical to enable an isolated, bare-metal, cloud-native computing platform that will define the next generation of cloud-scale computing.

4.4.2.2 DPUs Incorporated into SmartNICs

The DPU can be used as a stand-alone embedded processor. But it's more often incorporated into a SmartNIC, a network interface controller used as a critical component in a next-generation server.

Other devices that claim to be DPUs miss significant elements of these three critical capabilities.

For example, some vendors use proprietary processors that don't benefit from the broad Arm CPU ecosystem's rich development and application infrastructure.

Others claim to have DPUs but make the mistake of focusing solely on the embedded CPU to perform data path processing.

4.4.2.3 A Focus on Data Processing

That approach isn't competitive and doesn't scale, because trying to beat the traditional x86 CPU with a brute force performance attack is a losing battle. If 100 Gigabit/sec packet processing brings an x86 to its knees, why would an embedded CPU perform better?

Instead, the network interface needs to be powerful and flexible enough to handle all network data path processing. The embedded CPU should be used for control path initialization and exception processing, nothing more.

At a minimum, there 10 capabilities the network data path acceleration engines need to be able to deliver:

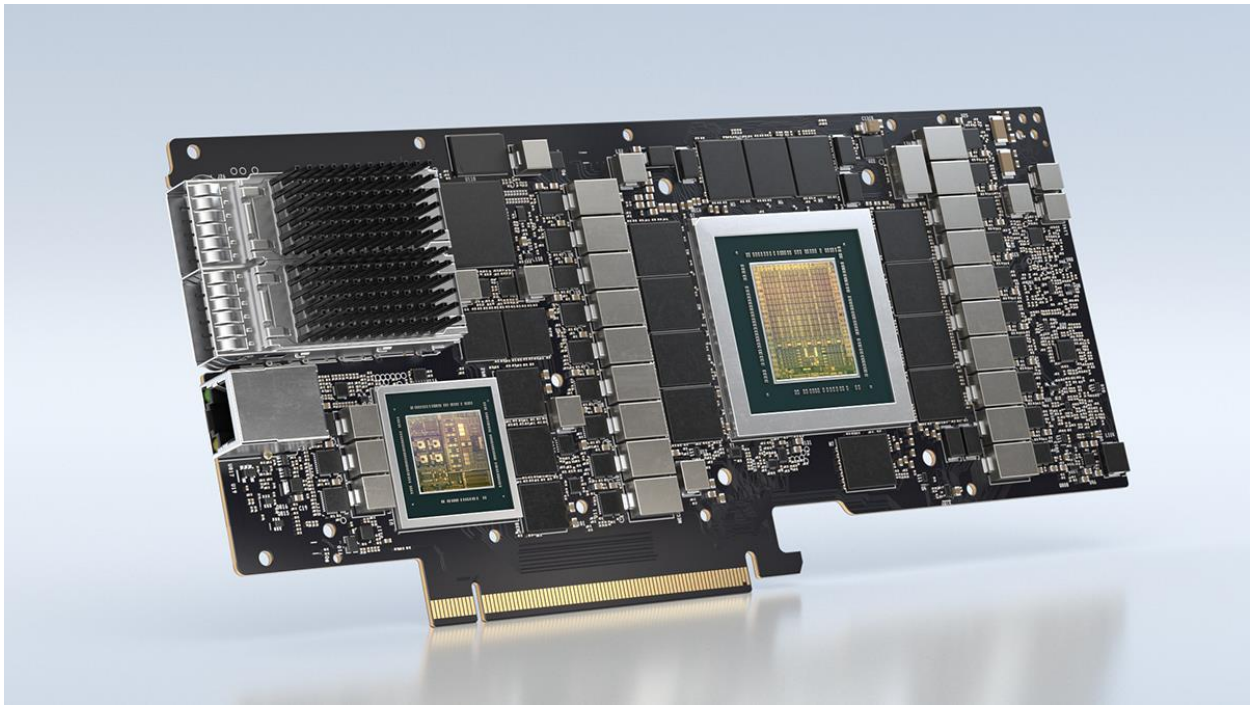
1. Data packet parsing, matching and manipulation to implement an open virtual switch (OVS)
2. RDMA data transport acceleration for Zero Touch RoCE
3. GPUDirect accelerators to bypass the CPU and feed networked data directly to GPUs (both from storage and from other GPUs)
4. TCP acceleration including RSS, LRO, checksum, etc.
5. Network virtualization for VXLAN and Geneve overlays and VTEP offload
6. Traffic shaping "packet pacing" accelerator to enable multimedia streaming, content distribution networks and the new 4K/8K Video over IP (RiverMax for ST 2110)
7. Precision timing accelerators for telco cloud RAN such as 5T for 5G capabilities
8. Crypto acceleration for IPSEC and TLS performed inline, so all other accelerations are still operational
9. Virtualization support for SR-IOV, VirtIO and para-virtualization
10. Secure Isolation: root of trust, secure boot, secure firmware upgrades, and authenticated containers and application lifecycle management

These are just 10 of the acceleration and hardware capabilities that are critical to being able to answer yes to the question: "What is a DPU?"

4.4.2.4 So what is a DPU?

This is a DPU:

4.4.2.5 HTTP-QuSS Server | NVIDIA BLUEFIELD-2/3 DPU



For:

- Bandwidth Slicing
- Bandwidth Shaping
- Secure quantum encrypted UDP Data Transmission

4.4.2.6 HTTP-QuSS Client | NVIDIA SOC DPU

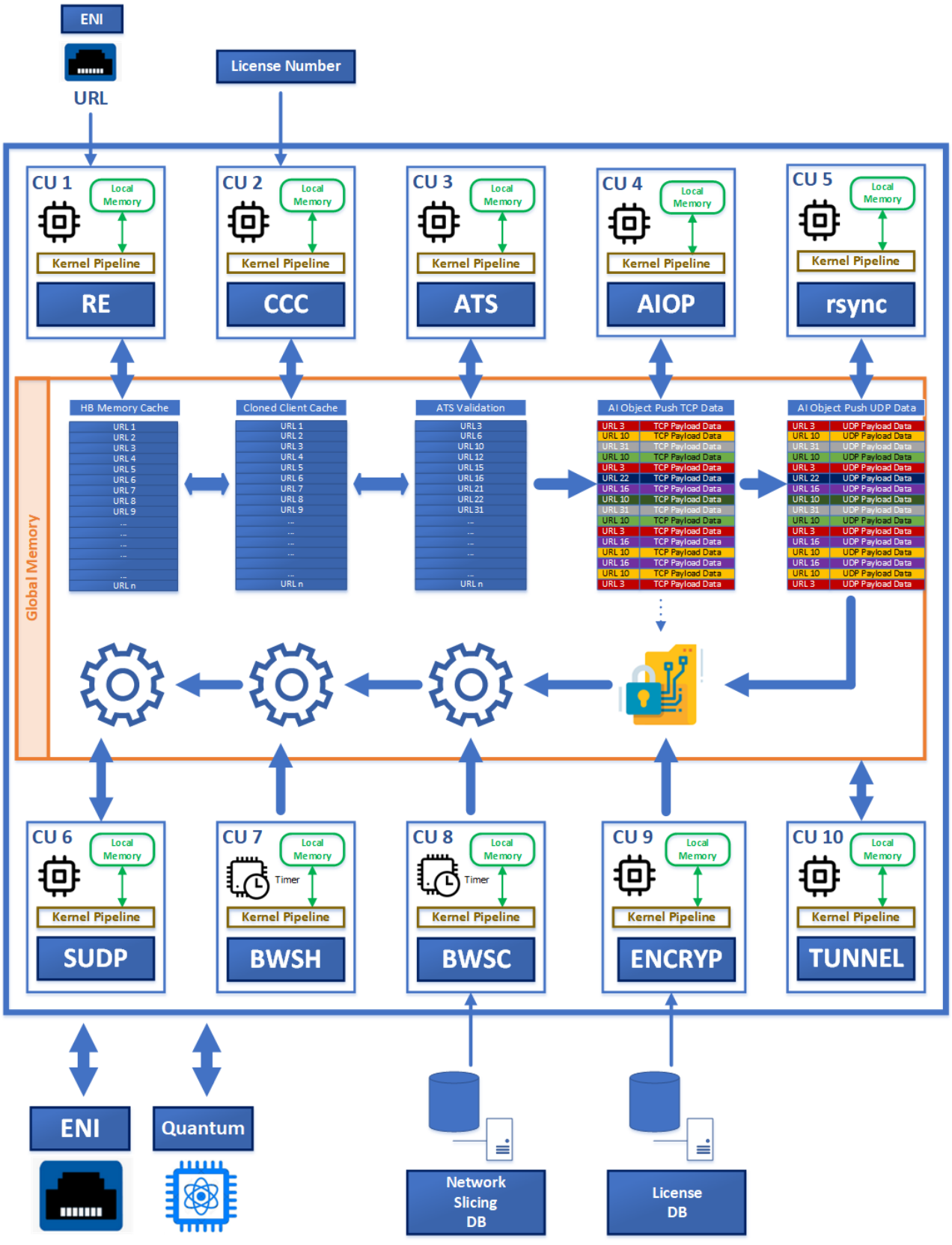
For:

- Processing the incoming high-speed HTTP-QuSS secure UDP Object Stream without stressing the CPU's
- Forwarding Object Data to proper TCP WinSock's
- Providing a non-interruptible Error Correction and Secure UDP Data Transmission

4.5 New disruptive HTTP-QuSS Server/Client Process Architecture

4.5.1 HTTP-QuSS Server | NVIDIA A/H100 Process Architecture

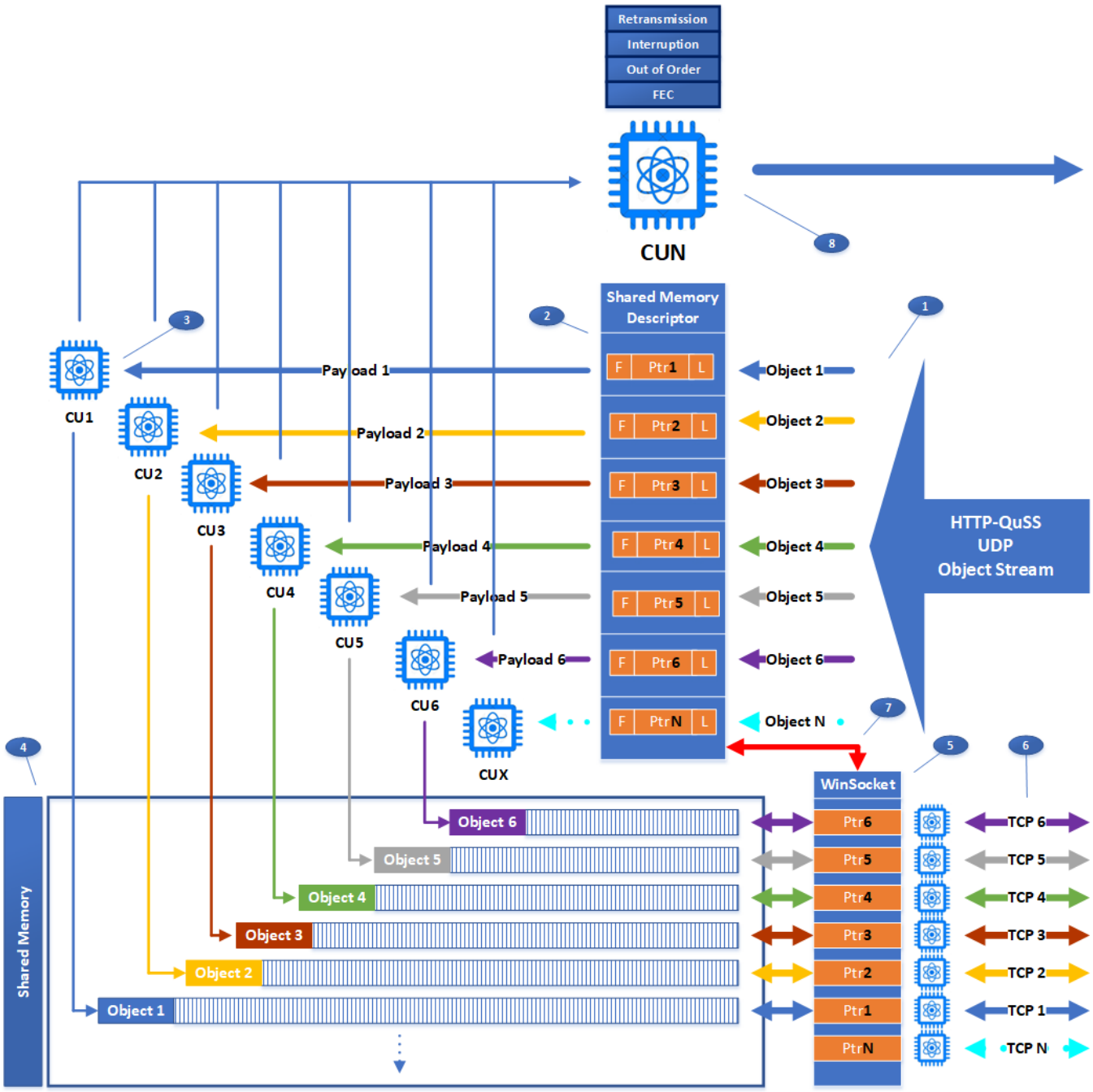
Coding Details are Part of the **HTTP-QuSS_Release_Specification.pdf**

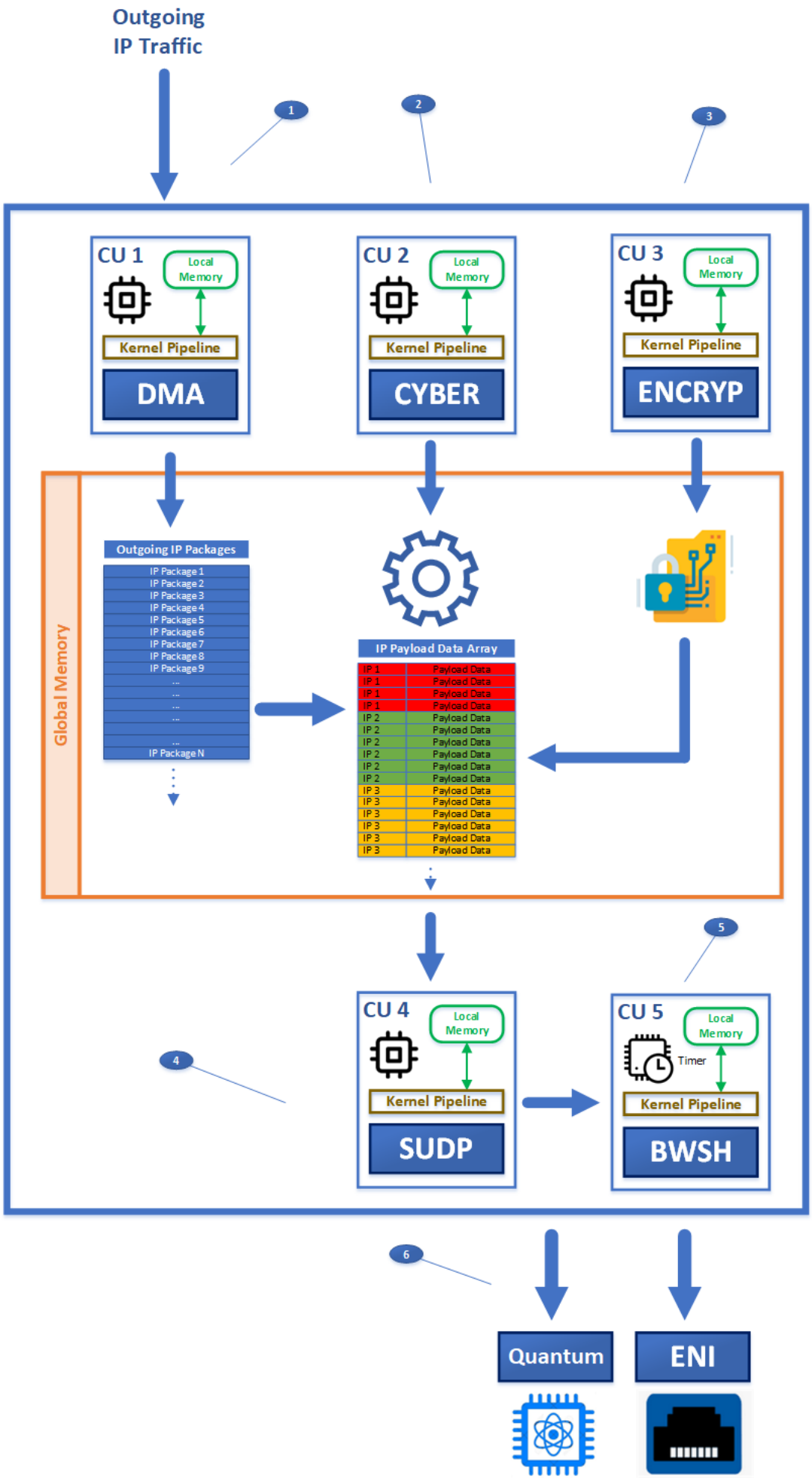


4.5.2 HTTP-QuSS Client | NVIDIA SOC DPU Process Architecture

Coding Details are Part of the **HTTP-QuSS_Release_Specification.pdf**

Incoming IP Traffic




Outgoing
IP Traffic
1
2
3
CU 1
Local Memory
Kernel Pipeline
DMA
CU 2
Local Memory
Kernel Pipeline
CYBER
CU 3
Local Memory
Kernel Pipeline
ENCRYP
Global Memory
Outgoing IP Packages
IP Package 1
IP Package 2
IP Package 3
IP Package 4
IP Package 5
IP Package 6
IP Package 7
IP Package 8
IP Package 9
...
IP Package N
IP Payload Data Array
IP 1 Payload Data
IP 2 Payload Data
IP 3 Payload Data
5
CU 4
Local Memory
Kernel Pipeline
SUDP
CU 5
Local Memory
Timer
Kernel Pipeline
BWSH
4
6
Quantum
ENI