How we developed the world's first hardware implementation of Network Time Security

A WHITE PAPER BY NETNOD

Executive Summary

Many of the important security tools that keep us safe online depend on accurate time. But until recently there was no way to ensure that the time you received over the Internet was correct. There was simply no way to tell if you were being fed time from a malicious or trusted source.

This is largely because the current standard for receiving time over the Internet, the Network Time Protocol (NTP), was created in 1985. In those more innocent times, the need to secure NTP, the type of security needed, and how to provide it were less understood. Over the years, a range of security flaws and some high-profile attacks have shown that NTP needed significantly improved security. The new Network Time Security (NTS) standard has been designed to fix that.

This white paper gives an overview of NTS including the benefits, challenges and lessons learned from Netnod's implementation of NTS at a hardware level. For a detailed description of the NTS authentication process, please see our previous NTS white paper available here,[1] which gives a step-by-step description of the key establishment and time stamping process.

[1] https://www.netnod.se/time-and-frequency/white-paper-how-does-nts-work-and-why-is-it-important

Table of contents

NTP security issues

Unsecured NTP is vulnerable to Man-in-the-Middle (MITM) attacks where a malicious actor sits between you and the NTP server, listens in on the conversation, forges messages and lies to you about time. As a lot of processes are dependent on accurate time, the consequences here can be very serious and can include:

- Incorrect timestamps on logs and transactions which can support fraudulent activities or help disguise other criminal actions
- Authentication problems, attacks and issues with authentication security measures
- Issues with cryptographic signatures and establishing encrypted sessions such as Transport Layer Security (TLS)
- DNS Security (DNSSEC) failing to work if the client doesn't have accurate time

NTP vulnerabilities

NTP security threats include: packet manipulation, replay attacks, amplification attacks, and spoofing

What is an amplification attack?

Amplification attacks are a particularly common and damaging form of DDoS attack. They use popular UDP-based protocols, such as NTP or DNS, where the source IP address can be easily spoofed. They can use a small request to generate a huge response which is directed at a third party, the victim, to overwhelm a network or server.

Previous attempts at NTP security

There have been some previous attempts to add security to NTP. Commonly referred to as NTP-AUTH, these attempts did not provide all necessary aspects of security (confidentiality, authentication and integrity including protection against replay attacks).

Moreover, the underlying security mechanisms for NTP-AUTH (MD5 or SHA-1) have not aged well and are today considered weak. NTP-AUTH is based on secret keys but the key agreement is not standardized. This means that any NTP service provider wanting to use NTP-AUTH has to find their own way to exchange keys with the client out-of-band.

This often requires a potential client to send a letter or fax to get the secret key. The need for a server to store unique, secret keys for all clients creates scalability problems: as the number of users grows, the storage requirements and especially the ability to rapidly look up a given key makes the service hard to scale.

Given all these issues, it has long been apparent that a new security mechanism for NTP is needed; one based on modern security mechanisms that could also allow the service to scale to potentially millions of clients.

The NTS standard

NTS has been developed within the Internet Engineering Task Force (IETF). In March 2015, the first Internet-Draft of the NTS standard was published. Over the next five years, the draft went through 28 further iterations until the Internet Draft 'Network Time Security for the Network Time Protocol' was published as a Proposed Standard (**RFC8915**) in September 2020.

NTS uses modern cryptography to add an important layer of security to NTP. It prevents spoofing and MITM attacks by using authenticated packets. Amplification attacks are prevented by ensuring that request and response packets are always the same size.

NTS is really two protocols: a key establishment protocol, and NTP with some new Extension Fields. The reason for using two protocols is separation of concerns:

1. The seldom used key establishment on top of standard Transport Layer Security (TLS), and
2. The (already existing) low latency UDP-based time synchronisation path.

This means that the existing NTP functionality is the same as before, but the time data can now be authenticated.

The NTS authentication process

The authentication process consists of a key establishment and a timestamp request. The key exchange server typically runs on an ordinary computer, but the slim NTS-enabled NTP server is UDP-based and stateless. It can be served from anycast addresses and can be implemented at the hardware level. The NTP server's state about each client is kept in the cookie provided by the client itself with each request. As there can potentially be hundreds of millions of clients, this is crucial for the smooth operation of a large-scale NTS service.

Since the cryptographic operations in the NTS path are symmetric it is both easier to implement them in hardware and possible to make them use constant time. This increases the accuracy of the time synchronisation and keeps the slower key exchange outside of the time synchronisation path.

NTS uses Authenticated Encryption using the Advanced Encryption Standard (AES), more specifically what is known as Synthetic Initialization Vector (RFC5297). This is a block cipher mode of operation providing nonce-based, misuse-resistant authenticated encryption. Using AES-SIV enables the encryption processes described below to add integrity and origin authentication. The consequences for this at a hardware level will be discussed below.

NTS hardware implementation

Netnod's previous hardware implementation of NTP (which also supported NTP-AUTH) has been available since December 2015. On 28 October 2019, Netnod launched one of the first software based NTS-enabled NTP services in the world. We worked on developing client-side implementations in Python and Go. At the same time, we wanted to implement NTS on the server side at a hardware level. This hadn't been done before, but we recognised that it brought a range of important security and scalability benefits.

The first benefit of a hardware implementation is that it enables more deterministic processing time. This improves both the quality of the time synchronisation provided and the security of the service by reducing the ability for side-channel attacks on the keys. Side-channel attacks exploit the physical effects caused by processing: for example the variance in power
consumption or the variance in how long it takes to perform a given operation.

The second benefit is that it makes the service much more robust and secure. An attacker able to inject or modify code (for example by specially crafted network packets) in the computer on which a software implementation is running could alter the behaviour of the implementation. A hardware implementation eliminates the possibility of such attacks.

The third benefit is improved efficiency and scalability. NTS processing relies heavily on cryptographic operations more suitable for a hardware rather than software implementation. The former enables an efficient, compact implementation of the NTS core functionality while also allowing for scalability. As opposed to having multiple servers with a load balancer, the hardware implementation allows for everything to be done internally using a smaller amount of equipment and consuming less power. These efficiencies lead to a much lower operational cost compared to using a cluster of servers.

FPGA chips

To understand how NTS was implemented at the hardware level, the first thing to look at is how Field Programmable Gate Array (FPGA) chips work. An FPGA chip contains a fabric with numerous simple digital elements, Look-Up Tables (LUTs), that can be configured to act as logic gates. The LUTs are connected to each other, to registers, memory elements or external pins via electrical wires. These interconnections are reconfigurable.

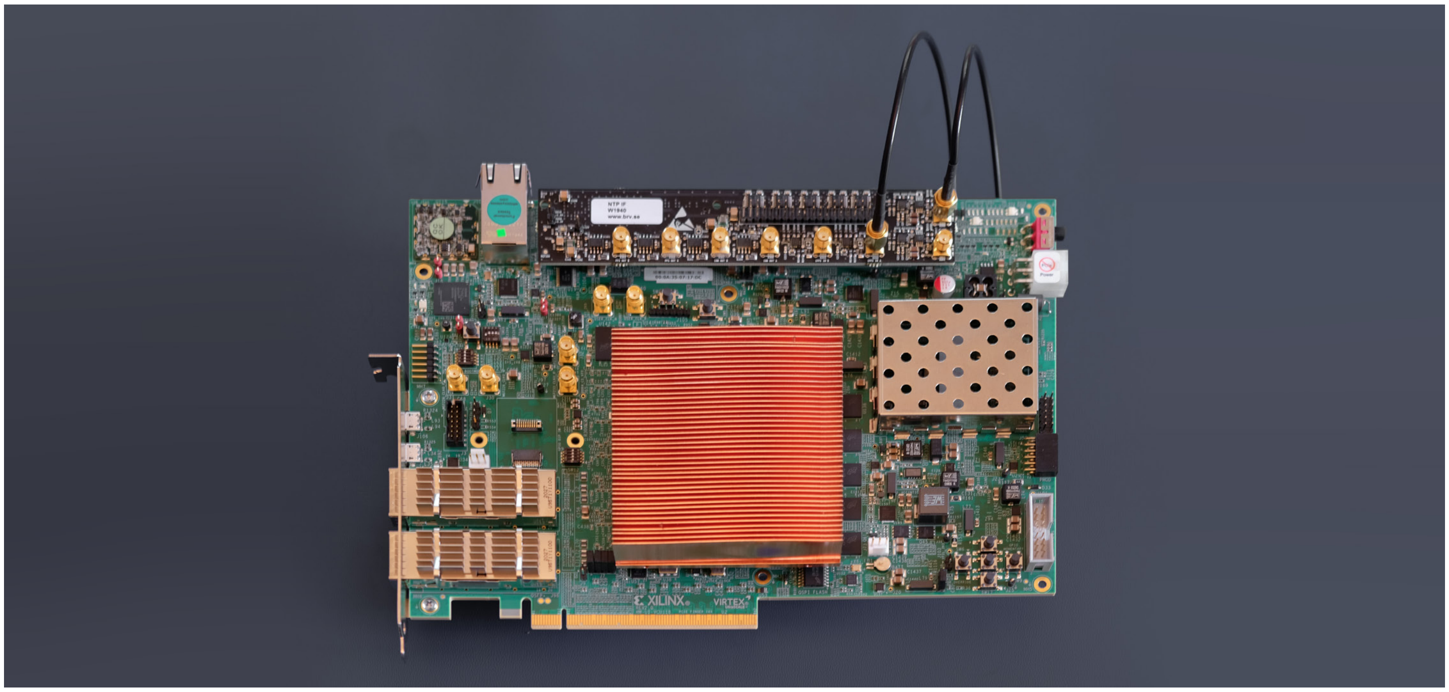


Figure 1: Xilinx VCU118 FPGA Evaluation Board used by Netnod

By configuring the LUTs, the wires and all other resources, the FPGA can implement digital designs, for example a processor, an Advanced Encryption Standard (AES) core, or a complete engine for processing NTS packets. The FPGA used by Netnod to implement the NTS server (Xilinx VU9P) contains almost 2.6 million logic cells.

Redesigning the NTP FPGA architecture

To understand the hardware challenges involved, we need to compare the original NTP (and NTP-AUTH) setup with the much more complex demands of NTS.

NTP is a quite simple protocol that works as follows:

1. The NTP server receives a NTP request
2. It performs NTP time stamping using its clock sources and the time information in the request
3. It sends the result back

Even with NTP AUTH, it is possible to perform all processing as an stream. This means that a NTP packet is processed while flowing through the FPGA without being stored. In terms of cryptography, each packet only requires at most a couple of MD5 or SHA-1 operations. In the original design, a single NTP engine for each port could handle all traffic with the implementation supporting 4 x 10 Gbps performance.

Configuring the FPGA

The digital design implemented in the FPGA is first described in a hardware description language (HDL). Using a FPGA implementation tool, the description is then mapped to FPGA resources and placed on the silicon die. Specific physical resources are allocated and configured with the wires routed between the allocated resources to connect them. The end result is a configuration file that is written onto the FPGA.

The NTS protocol has a series of steps that need to be performed in sequence on the client and server side as shown in the section below.

NTS timestamp request

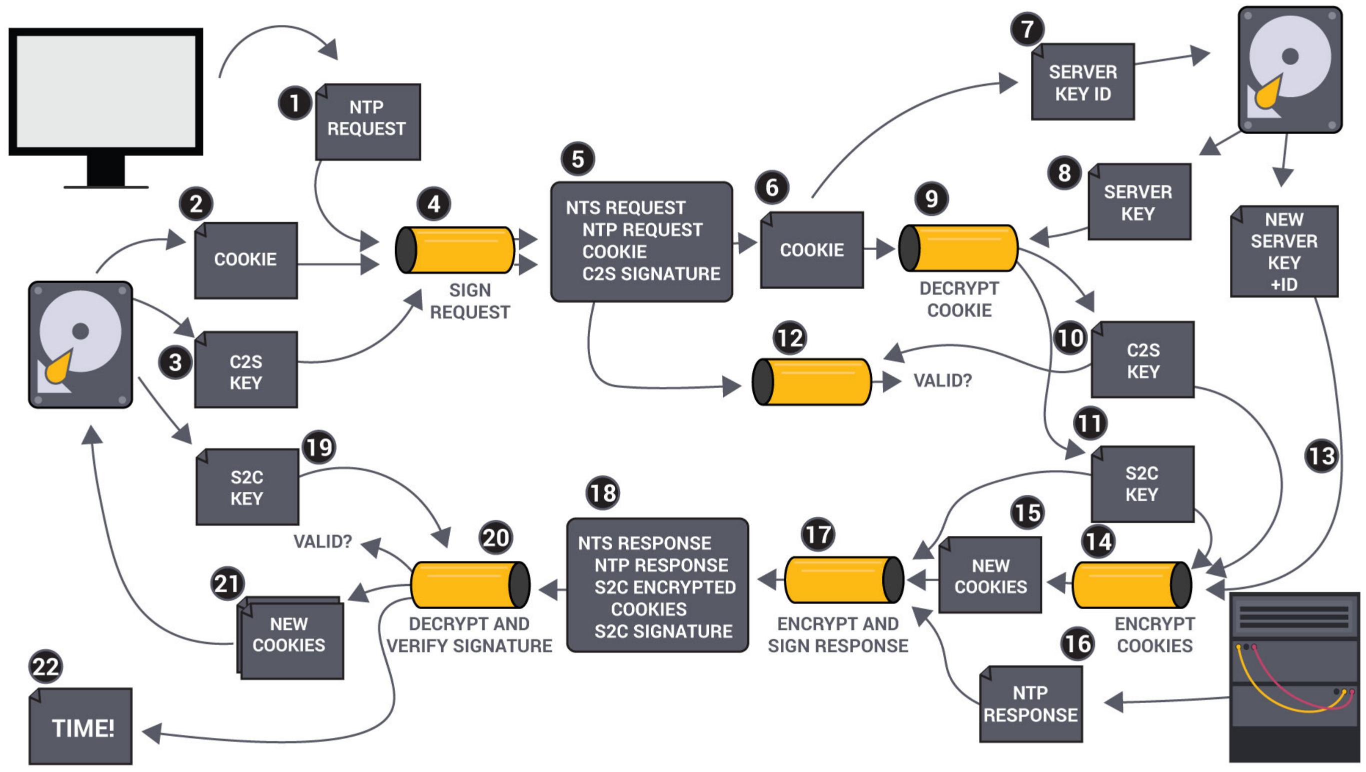


Figure 2: Timestamp request. For more details, including a description of the steps shown above, see Netnod's previous NTS White Paper here.

NTS and the new FPGA architecture

Due to the complex nature of the timestamp request, the packets must be read and modified in multiple passes. Furthermore the security mechanism selected for NTS (AES-SIV, see RFC5297) does not readily allow for streaming without significant duplication of FPGA resources. Instead, multiple passes over different parts of the packet are needed to perform an encryption or decryption operation such as decrypting a cookie or verifying the packet. Each pass requires multiple AES operations.

This means that an NTS packet needs to be stored. It also means that tens to hundreds of cryptographic operations are needed for a single NTS packet. These operations can not be performed in parallel without much unnecessary complexity and duplication of resources.

The end result is that at 10 Gbps the processing time required for a packet is longer than the minimum time between incoming packets. We realised a single NTS engine would not be able to handle all traffic for a port and that we needed multiple engines working in parallel.

This meant that the architecture of the NTP FPGA design had to be changed. The new architecture is illustrated below:

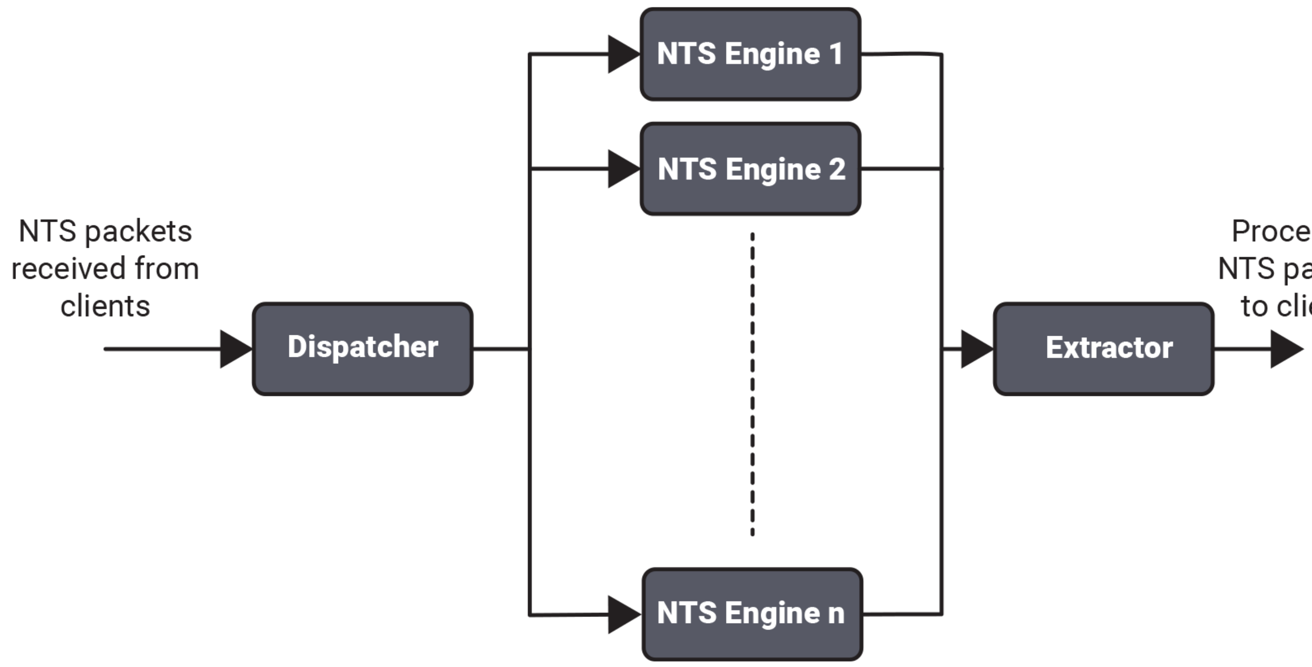

NTS packets received from clients
Dispatcher
NTS Engine 1
NTS Engine 2
NTS Engine n
Extractor
Processed NTS packets to clients

Figure 3: New FPGA architecture

Each NTS engine is fairly small and can not handle the full 10 Gbps wire speed on its own. To reach 10 Gbps multiple engines are needed. Two new IP blocks have been added in the FPGA design: the “dispatcher” which receives NTP and NTS packets from the network at 10 Gbps and distributes them over all the engines; and the “extractor” which collects the responses from all the engines and combines them into a single 10 Gbps stream of packets to be transmitted.

The implemented design supports scaling the number of engines from a single engine to as many engines as needed to meet the expected performance. The implemented design supports NTP including NTP-AUTH with a new, compact NTP engine. This allows for a mixing and control of the NTP and NTS capacity supported in a server.

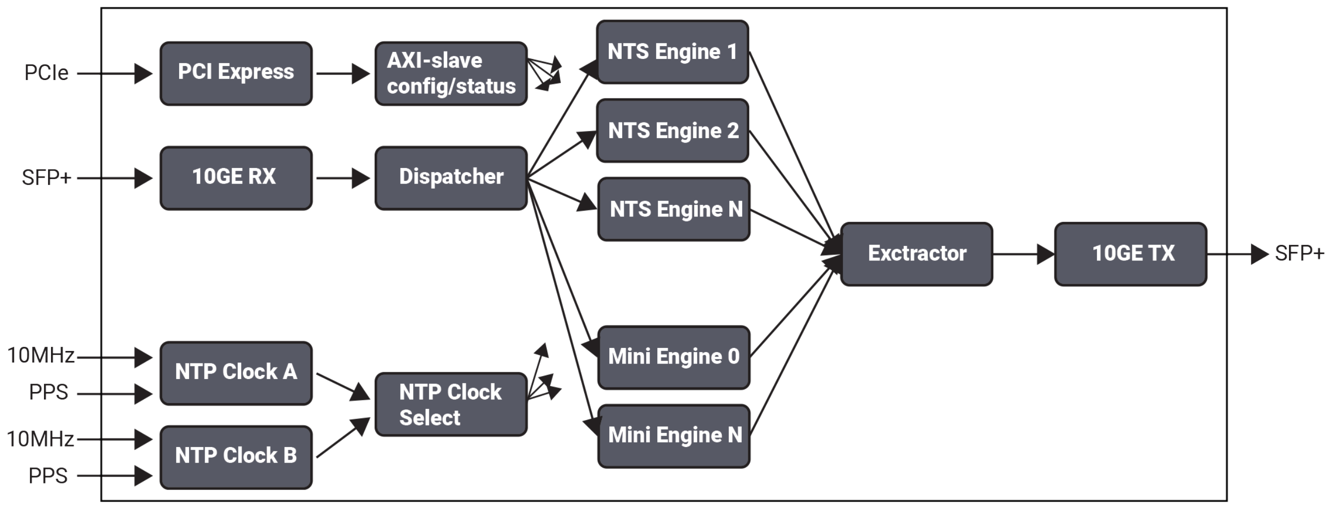


Figure 4: Detailed view of new FPGA design

This is a more detailed view of the FPGA design. The core of the FPGA, the dispatcher, all engines and the extractor are the same as in the previous image. There are actually two different kinds of engines: full “NTS engines”, which handle both NTS and NTP; and “mini engines”, which handle NTP and some other protocols, such as the Address Resolution Protocol (ARP) and ICMP but not NTS.

The “SFP+” is the physical network interface, a 10Gbit fibre interface that converts the optical signal to electrical signals connected directly to the FPGA. The 10GE RX block receives the electrical signals, decodes the ethernet frames and passes the payload to the dispatcher. At the other end, the extractor passes the payload to be transmitted to the 10GE TX block which encodes the payload to ethernet frames which are passed to the SFP+.

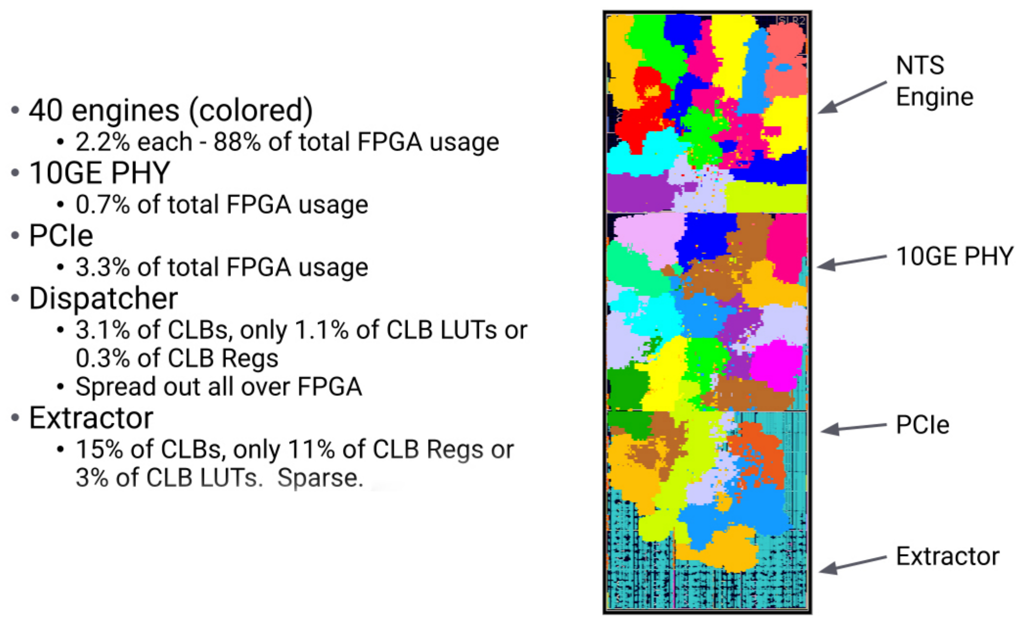
Additionally, there is a block which presents a PCI Express interface to the host PC. The PCI transactions are converted into transactions on an AXI bus. The AXI bus is connected to most other blocks inside the FPGA so that the host PC can configure the blocks and read out status information from them.

Finally, there are two identical “NTP clock” blocks which receive 10MHz and pulse per second (PPS) signals from the caesium clocks and use these to keep track of the current time inside the FPGA. For redundancy each FPGA is connected to two caesium clocks. If one caesium clock were to fail, the “NTP clock select” block can switch over to using the other caesium clock instead. The time output from the NTP clock select block is distributed to all engines and used to timestamp the NTP and NTS packets.

The NTS engine

The NTS engine includes:

- **A complete hardware implementation of AES-SIV, the first such implementation we are aware of**
- **A high performance, compact, cryptographic nonce-generator**
- **A fast packet parser**
- **Packet buffers capable of handling two packets at the same time (with one packet being received or processed while a previously processed packet is waiting to be sent or is in the process of being sent)**



- 40 engines (colored)
  - 2.2% each - 88% of total FPGA usage
- 10GE PHY
  - 0.7% of total FPGA usage
- PCIe
  - 3.3% of total FPGA usage
- Dispatcher
  - 3.1% of CLBs, only 1.1% of CLB LUTs or 0.3% of CLB Regs
  - Spread out all over FPGA
- Extractor
  - 15% of CLBs, only 11% of CLB Regs or 3% of CLB LUTs. Sparse.

Figure 5: VU9P FPGA Utilisation

This is a picture from the Xilinx FPGA tool Vivado which shows the utilisation of the VU9P FPGA. At the top and middle are the NTS engines which represent most of the logic used in the FPGA. There are 40 engines each using about 2.2% of the FPGA resources for a total of 88%.

The 10GE PHY is the physical interface to the SFP+. It contains something called a serialiser/deserialiser (SERDES), which translates between the serial stream of bits on the electrical interface at 10 Gbps to 64 bit wide words. This means that most of the internal logic in the FPGA runs at 10G/64=156MHz. The logic for the 10GE PHY uses less than 1% of the FPGA resources. A large part of the SERDES logic is actually a “hard IP block” which is not part of the reconfigurable FPGA resources and, as a result, does not show in the utilisation image.

The PCIe interface converts between the PCI Express bus and the internal AXI bus. PCI Express is a fairly complex protocol and uses 3.3% of the FPGA resources. The dispatcher is quite spread out. The FPGA tools have distributed most of that logic into the NTS engines themselves which means that it’s hard to see it in the utilisation image. The dispatcher still uses about 3.1% of the FPGA resources. The extractor is the single largest block when it comes to FPGA utilisation. It uses about 15% of the FPGA resources.

Hardware challenges

We experienced some interesting challenges during the development process. One challenge was implementing the NTS parser to handle all corner cases (e.g. flexibility in extensions) while meeting clock frequency requirements and ensuring low latency (e.g. using very few cycles to find extensions, parse them, and decide how to process the packet). We solved this through running a vast number of test cases and simulations as well as fuzzing of network traffic.

Another challenge was getting the AES-SIV to work efficiently at the hardware level. We needed to optimise the AES core to a single cycle/round and to find out how to perform key expansion on-the-fly. We developed a compact AES engine that supports on-the-fly key expansion and with 10 cycles/operation latency to allow 10 cycles/AES operation, which is the highest performance possible for AES. Implementing the packet dispatcher and dispatch network to handle multiple NTS engines also presented a significant challenge as did the logic for merging the packets from each engine into one 10Gbit output stream. This required numerous builds of the bitstream and fine-tuning of the route of the design on the FPGA device.

More information on Netnod's NTS service

How does NTS work and why is it important? (Netnod white paper)
https://www.netnod.se/sites/default/files/2021-01/Netnod_NTS_Whitepaper_2020.pdf

Netnod's NTS-enabled NTP service (freely available to the public)
More information is available at:
https://www.netnod.se/time-and-frequency/network-time-security

How to set up an NTS client and connect to Netnod's NTS servers
https://www.netnod.se/time-and-frequency/how-to-use-nts

Netnod's FPGA implementation
Netnod's FPGA implementation has been running since the summer of 2020. This is an open source licensed project using the BSD 3 clause
https://opensource.org/licenses/BSD-3-Clause

The public repository for the FPGA_NTP_SERVER can be found at:
https://github.com/Netnod/FPGA_NTP_SERVER

NTP clients that support NTS

Some current NTP clients supporting NTS include:

ntpsec (written by Eric Raymond)
https://gitlab.com/NTPsec/ntpsec

Chrony (written by Richard Curnow, currently maintained by Miroslav Lichvar)
https://chrony.tuxfamily.org/

A Python implementation (written by Christer Weinigel, Netnod)
https://github.com/Netnod/nts-poc-python

A Go implementation (written by Michael Cardell Widerkrantz, Daniel Lublin and Martin Samuelsson)
https://gitlab.com/hacklunch/ntsclient

Netnod: the time and frequency experts

As one of the leading figures in NTS, Netnod has worked on all stages of NTS development from the IETF standard to software and hardware implementations at client and server levels. Netnod provides NTP, NTS and Precision Time Protocol (PTP) services offering a robust, reliable and highly accurate source for time and frequency. Netnod's time service, funded by the Swedish Post and Telecom Authority (PTS), uses a distributed timescale on multiple, autonomous sites throughout Sweden to provide a time service available over IPv4 or IPv6. The time is traceable to the official Swedish time UTC(SP). Each site has full redundancy with multiple servers, caesium clocks and FPGA boards providing an extremely fast hardware implementation of NTP and NTS. This means you speak NTP or NTS directly to the FPGA chip. As there is no software involved, you get the most accurate time possible. The service is available to the general public worldwide for free on ntp.se, which resolves to anycast IPv4 and IPv6 addresses. The NTS-enabled service is available at: nts.netnod.se

www.netnod.se - info@netnod.se - +46 8 562 860 00