The Future of Telecommunication is Happening Now

Check how the experts do it

NOKIA

Book Title:
**The Future of Telecommunication is Happening Now.
Check How the Experts Do It.**

Chief Editor:
Marcela Mężykowska

Content Supervisors:
Michał Bartkowiak, Łukasz Grządko, Grzegorz Olender, Krzysztof Sokołowski, Paweł Wieczorek, Sławomir Zborowski

Editors:
Anna Czołpiński, Szymon Izydorek, Beata Malinowska, Patrycja Mróz, Grzegorz Ogledzki, Katarzyna Pasko – Jarosik, Daniel Rowiński, Aldona Żbikowska

Publisher:
Nokia Solutions and Networks Sp. Z o.o.

We would like to acknowledge the help of all the people involved in this project. Without their support, this book would not have become a reality.

Dear Readers,

The book that you are holding now in your hands is the third part of Nokia Book series published in the European Software and Engineering Center in Wroclaw. Traditionally, our specialists have given their contribution. They have written about the real-life case studies which are described in the following pages.

"The Future of Telecommunication is Happening Now. Check How the Experts Do It" shows practical dimension of advanced telecommunication technologies and System Engineering threads. Our writers, being at the same time developers, testers, researchers, and engineers in general, did not overlook the topic of professional software development.

Taking into consideration the reference to the title of the book we have written, its content extensively covers spectrum of prototyping, creating, and developing new technologies like 5G. The readers curious about the new age of technology will also find several significant articles that deal with Internet of Things, and the future face of Internet.

I sincerely recommend this publication to all enthusiasts of telecommunication, including those readers who encountered previous editions of Nokia Book. I am sure they will not be let down by this part. I would also like to thank the authors for their outstanding contribution and hard work.

I wish you a pleasant read,

Bartosz Ciepluch

Head of Nokia Networks European Software
and Engineering Center in Wroclaw

Advanced Telecommunication Technologies

NOKIA

5G Prototyping: The Key Challenges and Innovations

Jacek Góra
R&D Manager
CIOO Bell Labs

NOKIA

Today we are at the verge of bringing into life the 5th generation (5G) of radio systems. In case of 5G, stronger than ever before, the defined performance targets push for innovative implementation approaches. This paper outlines some of the main challenges of 5G prototyping, and the main innovations done today in this area.

1. The 5G revolution

Looking about three decades into the past one can see four generations of radio systems coming into life. Each generation, in practice, brought a 10 times increase in peak data rates and capacities, and a 10 times decrease in latencies.

The 5G concept is built around this upward trend, but also reaches out to new scenarios and use cases. The main ones being [1]:

- Extreme mobile broadband (eMBB)
- Mission critical communications (MCC)
- Massive machine-type communications (mMTC)

Each of the 5G use cases comes with a different definition for quality of service (see **Figure 1**):

- The eMBB requires more than 10Gb/s data rates and Tb/s/km^2 network capacities.
- The MCC prioritizes less than 1ms latencies and ultra-reliability.
- The mMTC expects 100 times more communication devices to be handled efficiently.

The abundant set of requirements pushes 5G platforms towards flexible and scalable solutions. Furthermore, some of the requirements force a redefinition of the previously used implementation technologies and practices. As a result, the 5G development relies more on hardware (HW) prototyping than the previous generations of cellular systems.

This paper presents some of the key challenges of the 5G prototyping and the observed directions of radio HW research. Firstly, Chapter 2 treats the radio spectrum choices for 5G and the resulting impacts on HW. Next, Chapter 3 discusses evolution of base station (BS) architecture in the context of the 5G requirements. Chapter 4 describes computing requirements for digital platforms aiming to support the 5G baseband (BB). Finally, all the considerations are outlined in Chapter 5.

2. Unlocking new spectrum assets

The capacity and data rate requirements for the eMBB rise bandwidth expectations far beyond its current usage. Satisfying this growing demand is possible by reaching frequency bands above the currently used spectrum (less than 6 GHz). Specifically, the 5G implementations target new bands in the range of 6100 GHz.

As depicted in **Figure 2**, each of the frequency ranges considered for 5G has a different role to play in the overall system concept. The roles relate directly to the propagation phenomena and implementation aspects.

Figure 1 5G scenarios and requirements

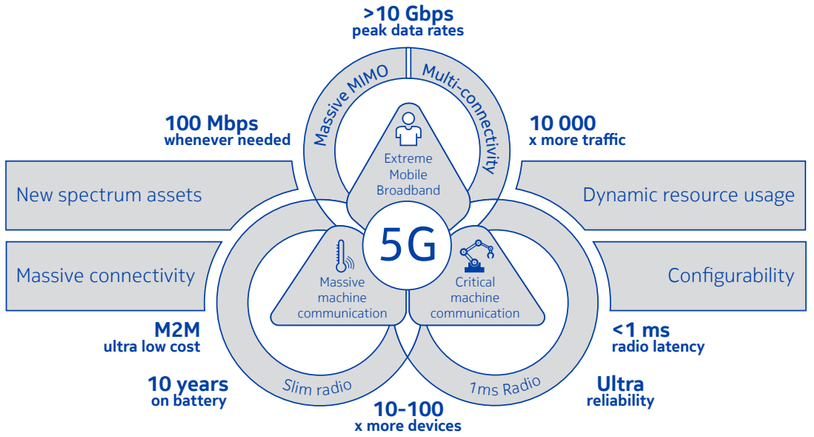
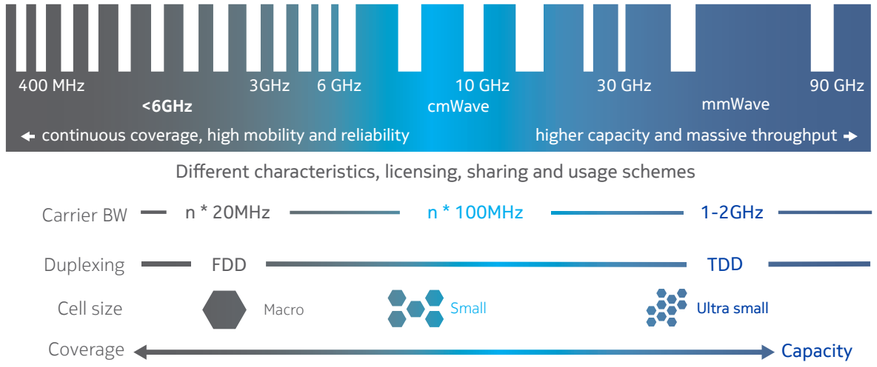


Figure 2 5G spectrum [1]



In order to unlock the new spectrum for commercial mobile communications, breakthrough developments are needed particularly in the analogue microwave and antenna technologies.

2.1. Microwave technology

The transceiver architectures, platforms, and circuit technologies for 6100 GHz are generally known from radar or satellite applications. The main task on the road to 5G is to make these technologies applicable for massive, commercial deployments, that is, cheap, durable, consumer size. Therefore, the main research directions include:

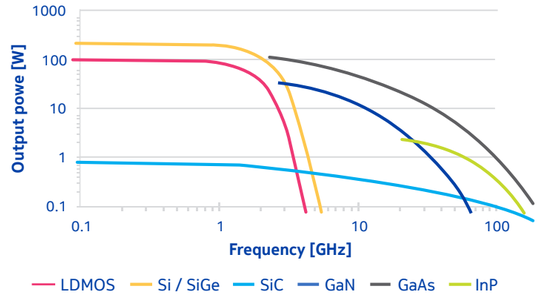
- New materials and assembly technologies suitable for high frequencies
- Sufficient power levels
- Low cost
- Overall integration of the radio transceiver (TRx) chain

The Sibased integrated circuits (ICs) technology has reached the miniaturization level sufficient to enable operation of up to 100 GHz. However, at this frequency the Sibased ICs are limited to maximum output power of ~0.1 W (see **Figure 3**). This issue can be overcome by using IIIV compound materials, for example, GaAs, InP or GaN. The ICs based on those materials can reach 10 times higher output powers, and show better noise figure than, for example, those based on SiGe.

The new 5G spectrum also drives for a significant analog TRx integration. This is motivated by two factors. Firstly, at GHz frequencies every interconnection introduces parasitic effects. Secondly, GHz wavelength and massive multiple inputs multiple outputs (MIMO) concepts (more on that later) impose tight size and power constraints.

Today's microwave radios are build using discrete components (amplifiers, mixers, filters, and so on). The target 5G implementations require four, eight, or fuller TRx chains integrated on a single chip. It is expected that 5G radios will be based on MMICs (monolithic microwave ICs), thus, enabling significant integration and miniaturization. A further step might be integration of CMOS ICs with the MMICs into hybrid or monolithic "allinone" ICs.

Figure 3 Performance of semiconductor materials at 5G radio frequecies



Lastly, higher TRx count and integration require novel assembly and thermal management techniques. Recent concepts include, for example, novel materials and multi-layer structures designed for embedding ICs or pipes for liquid cooling directly into the PCB.

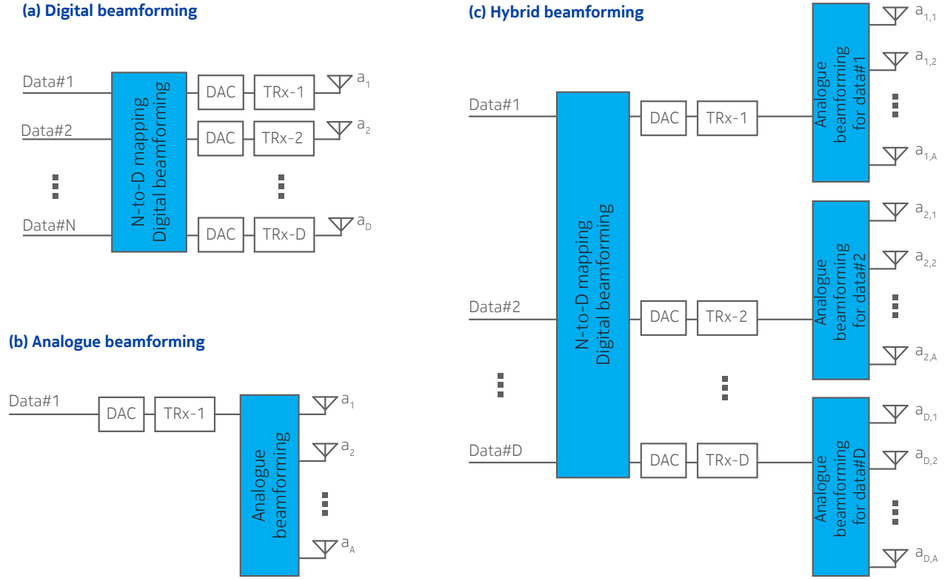
2.2. Antenna techniques

Moving into higher frequencies has one major drawback – increased propagation losses. The simplest relation (free-space path loss model) states that the communication range is inversely proportional to the carrier frequency, for example, 14 times at 28 GHz versus 2 GHz.

On the other hand, the increase of the frequency proportionally reduces linear dimensions of antennas. Specifically, a k-times increase of the carrier frequency, enables k^2 antennas to be placed within the same physical area as occupied by a single antenna at the base frequency. The k^2 antennas, if used properly, can enable up to $20\log_{10}(k)$ [dB] beamforming (BF) gain, thus, compensating for the increased propagation losses at higher bands. The challenge is, however, to efficiently utilize the tens or hundreds of antennas, for example, 200 at 28 GHz versus 2 GHz. Figure 4 depicts the concepts for that.

The digital BF (see **Figure 4**) operates in BB Layer1 (L1). It maps signals to different antennas and modulates their phases. This way, both coverage extension (via BF gains), and spatial multiplexing (such as transmission of multiple data streams) can be achieved. On the down side, each antenna requires a separate converter and TRx chain. Moreover, the complexity of the control algorithms increases with the number of antennas. Currently, LTE uses the digital BF of up to eight antennas (MIMO). In contrast, 5G concepts envisage tens or even hundreds of antennas to be used (massive MIMO), rendering full-digital BF not fit for practical applications.

Figure 4 Beamforming control schemes



The analogue BF (see **Figure 4**), on the other hand, applies phase shifting to analogue signals at the last stage of a TRx chain. This means that only one data stream can be processed at a time (such as no spatial multiplexing). Additionally, it is typically assumed that the antennas for the analogue BF are tightly packed causing radio channels to be highly correlated. This way, very little feedback is needed from a receiver, and computational overheads are minimal even for a huge number of antennas.

The hybrid scheme (see **Figure 4**) combines benefits of the two prior methods. A big array of antennas can be subdivided into modules controlled in the analog way. At the same time, each of the modules can be used as a virtual antenna for digital BF. For example a 512-element antenna, assembled with 8x8 modules could support eight digital channels with ~18dB analogue BF gain. For a 28 GHz implementation, this antenna would occupy space of around 12x24 cm. In comparison, a ~2 GHz commonly used LTE active antenna (10x1, Xpolarized) can support only two digital channels, with just ~10 dB BF gain, and occupies approximately eight times the area.

The hybrid BF scheme seems especially fit for lower 5G bands (around 640 GHz). At those frequencies propagation conditions still provide sufficient diversity to support approximately eight digital channels. Yet, propagation losses already push for employing BF gains.

At the higher 5G bands (around 30-100 GHz), radio channels typically do not allow to reach high transmission ranks. Thus, analogue-only BF, or low rank (for example two) hybrid BF is typically used.

It is important to realize that massive MIMO is a key enabler of 5G. Thus, the above BF methods need to be supported by default in the 5G HW implementations. This, further on, increases requirements for BS architecture and computing functions as discussed in the following chapters.

3. Base station architecture redefinition

A legacy (for example. LTE) BS architecture, as depicted in **Figure 5**, consists of:

- System module (SM) implementing BB functions
- Radio module (RM) implementing digital front-end (DFE)
- Converters
- The analogue front-end (AFE)

The same BS architecture does not work for 5G, though. Throughput, latency and power requirements for 5G demand a complete rethink of the radio signal processing partitioning and the interfaces between individual nodes of the BS.

3.1. CPRI/OBSAI interface

The interface between the SM and RM is typically CPRI/OBSAI [2]. In case of LTE, it carries ~60 b/s/Hz per digital channel (antenna). Having eight antennas and a 20 MHz BW, the interface data rate is ~10 Gb/s with a latency of up to ~0.5 ms. Using the same architecture and interface for 5G, first of all, makes it impossible to reach the <1 ms radio latency required by some of the 5G use cases. Furthermore, the interface requires significantly higher data rates (for example, 160 Gb/s for 400 MHz BW, with eight digital channels).

Figure 5 LTE base station architecture

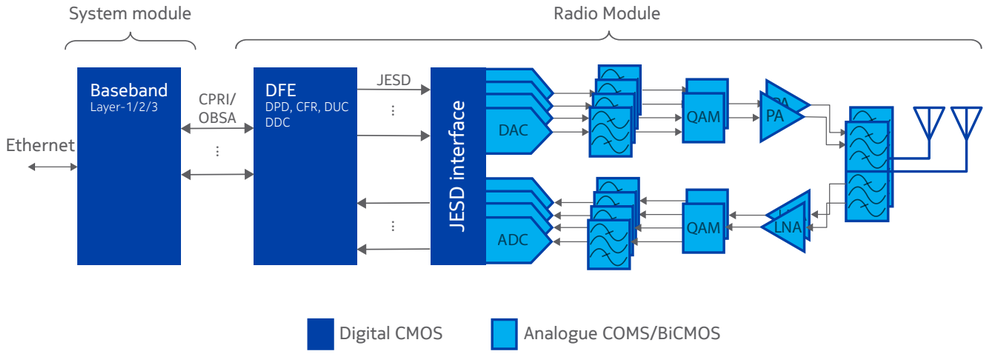
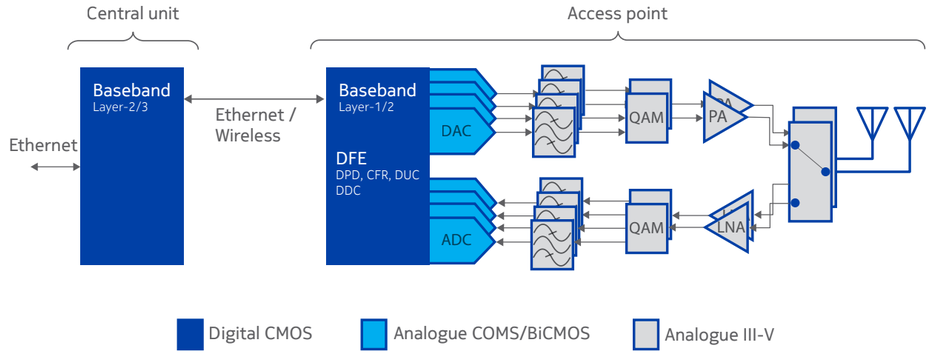


Figure 6 Envisioned 5G base station architecture



Considering the above, there are ongoing studies to define a new architecture for 5G BSs. One promising option is to move BB functions requiring fast feedback loop with the physical interface into RM. This includes BB L1 and some elements of L2 (for example link adaptation). Integration of the lower BB functions with DFE could avoid the problematic interface, or at least enable more efficient implementation.

At the same time, the interface between higher and lower BB layers could be also made more efficient taking advantage of relaxed latency and data rate requirements. The considered options are standard Ethernet (40GbE or faster) or even wireless.

3.2. JESD204 interface

JESD204 is a serial interface interconnecting DFE with converters. The current specification JESD204b [3] allows for up to 12.5 Gb/s per lane. Also multiple lanes can be grouped to create high throughput links. Furthermore, latency of the interface is deterministic and practically negligible.

The main issues with the JESD204 interface are:

a) Adding to system complexity
b) Requiring a high number of highspeed SERDES lines for 5G configurations
c) Using power that can be significant in overall 5G radio power budget

The points b and c can be tackled by increasing JESD204 data rates without increasing the interface power consumption per bit. With this modification the interface would require a lower number of lanes, and overall would consume less power. The industry is now preparing for the JESD204c standard, expected to increase the rates up to 32 Gb/s [4].

Another way to address all the above issues is to integrate digital ICs implementing DFE with mixed-signal converters. It is expected that the existing converter vendors, likewise digital ASIC and FPGA vendors will soon work on products integrating highspeed converters for wireless applications and digital processing optimized for DFEs into single hetero or monolithic chips.

Taking into account the factors described above, the BS architecture is expected to evolve from the one depicted in **Figure 5**, to the one depicted in **Figure 6**. It is also expected, that at some point the digital and converter blocks will be further integrated with the analogue MMICs resulting in a complete radio-on-chip solution.

4. Computational functions

In the traditional BS architecture, computing functions are implemented on a mixture of general purpose processors (GPPs), digital signal processors (DSPs) and application specific integrated circuits (ASICs). Higher BB layers are typically implemented in software (SW), and could easily be moved to cloud environment. On the other hand, L1 and DFEs are computationally demanding and often need dedicated computing platforms.

In case of prototype designs, field programmable gate arrays (FPGAs) are the natural choice. They are the most flexible and yet affordable solutions giving the possibility to tryout various algorithms, functionalities and designs in HW by using software-based methodologies.

Complexity of a 5G BB design is significantly higher than in the case of previous radio generations. To illustrate this, let us consider two configurations:

- LTE 20 MHz radio interface:
  - 30.72 MS/s sampling rate
  - 1 ms transmission time interval (TTI)
  - Up to 1 Mb of data to be processed per TTI
- 5G cmWave 400 MHz interface concept:
  - 491.52 MS/s sampling rate
  - 125 us TTI
  - Up to 2.7 Mb of data per TTI

This simple data already indicates that the 5G BB L1 needs to process ~2.7 times more data in eight times less time than it is in the case of LTE. This results in over 20 times higher performance required from 5G computing platforms.

It is important to notice that LTE sampling rates (up to 30.72 MS/s) are much lower than the maximum frequency rates supported by today's FPGAs, DSPs or ASICs. This makes it possible to significantly oversample the LTE signal and employ the so called resource folding technique, where single processing resource (for example DSP, MAC) can be used to process multiple data in a single sample period. This significantly reduces the required resources, power, and cost. For 5G, however, the sampling rates are significantly higher (for example 491.52 MS/s for 400 MHz BW), making resource folding either impossible or not that efficient.

FPGA and ASIC vendors address this challenge by implementing HW acceleration blocks optimized to perform specific set of computationally intensive operations of DFE, BB L1 or common interfaces. This could include things like FEC, programmable DUC/DDC, generic DSP functions, 100G Ethernet MAC, and so on. All intended to optimize cost and power of computational platforms.

Last but not least, given the different nature of computational requirements for DFE, BB L1 and L2, and given the trend of integrating those three on a single platform, the recent computational chips include many different architectures as a single system-on-a-chip (SoC). This includes:

- Multi-core application processing unit (APU)
- Multi-core real-time processing unit (RPU)
- Complex signal processors, for example, in form of Vector Signal Processor (VSP)
- FPGA logic
- Specialized accelerators and interfaces
- Onchip SRAM and DRAM memories and more

These creates so-called heterogeneous architecture and is a key to meet complex computational requires of future radio systems. What is more, the increasing complexity of these SoC drive innovation in SoC onchip interconnects, where some vendors are moving from AMBA (ARM interconnect family) to SDN type interconnect. The considered onchip SDN interconnect is similar to one already used in data-centers, but in this case at a much smaller scale.

As an example, the recent FPGAs from two major vendors integrate programmable logic with an ARM-based processing system and a set of dedicated peripherals [5] [6]. Those Multi-Processor SoC devices (MPSoC) combine APU and RPU with large FPGA fabric to address various applications. Such MPSoCs enable to tryout various implementation approaches:

- Dedicated HW structures
- Pure SW functions
- Mixed SW with HW accelerators

Because the chips contain general purpose processing cores, as well as FPGA resources, in theory they can reuse some of the legacy SW or HDL designs.

A recent development in terms of HW prototyping is the High Level Synthesis (HLS). This is a method of generating HDL code (HW definition) from a SW implementation [5]. Such tools can be used to extract computationally challenging modules from SW implementations and implement them in form of HW accelerators. The HLS tools allow us to make the prototypes quickly by direct linking of the functional reference model with the target implementation.

5. Summary

In this paper several challenges of 5G prototyping are highlighted, including:

- Compatibility of semiconductor technology with 5G bands
- Beamforming control for massive antenna arrays
- Base station architecture and related interfaces
- Computational complexity of 5G BB functions

For each of the challenges, recent technology trends are described. Those innovations can potentially ease the task of implementing advanced 5G functions proofs-of-concept. With this kind of work the essential features are flexibility of HW platform, and short design cycles from a concept to implementation. The key elements in this direction are novel FPGA-based SoCs and advanced design tools and methodologies.

References

[1] Nokia Networks, 5G Masterplan – *Five Keys to Create the New Communications Era*, whitepaper
[2] M.Koziar and Z.Nowacki, "OBSAI and CPRI – Internal Transport Interfaces in Base Stations", in *Shaping the future of telecommunications*, Nokia book
[3] JESD204B Survival Guide, ADI tech.document
[4] http://www.comcores.com/JESD204C.html
[5] http://www.xilinx.com/
[6] https://www.altera.com/

About the author

I am a graduate of Poznań University of Technology (PhD) and Wrocław University of Technology (MSc). For six years I have been working on radio systems research, contributing to 3G, 4G and 5G system concepts and standards. Currently I am leading a prototyping team committed to putting the 5G concepts into life.

Jacek Góra
R&D Manager
CIOO Bell Labs

Challenges and Solutions for Millimeter-Wave Communications in 5G

Patrick Marsch
Manager, Radio Research
CIOO Bell Labs

Matthias Hesse*
Senior Engineer, Radio Research
CIOO Bell Labs

Arnesh Vijay
Engineer, Radio Research
CIOO Bell Labs

* with contributions from Ali Yaver, Krystian Safjan, Reza Holakouei, Pawel Krysiak and Marcin Rybakowski

NOKIA

Abstract

It is clear that millimeter-wave (mm-wave) technology, referring to wireless transmission using carrier frequencies above 6 GHz and up to 100 GHz, will be an essential aspect introduced in 5G cellular communcations, as this will provide access to sheer amounts of available spectrum. However, wireless transmission at such high frequencies is subject to a very different nature of signal propagation, rendering the usage of highly directive transmission necessary and making wireless links much more volatile than at lower frequencies. In consequence, wireless system design has to be substantially rethought to overcome the aforementioned challenges. This article provides an introduction to mm-wave communications, elaborating in detail on inherent technical challenges, and describing key solutions developed by Nokia Bell Labs to address these challenges.

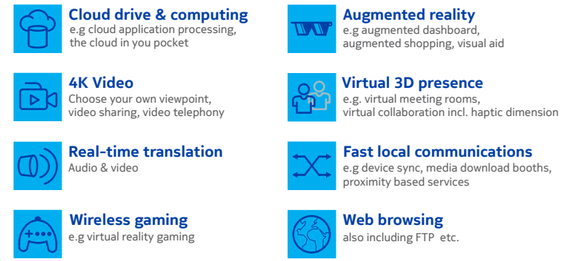
1. Introduction

Cellular wireless technology has been rapidly evolving ever since its early inception in the late 1980s, and in the meantime become an indispensable part of our day-to-day activities. Due to the large market penetration of sophisticated mobile handsets and an ever increasing number of data-hungry mobile applications, cellular network operators are faced with the challenge to continuously provide increasing wireless data capacity to their customers. For the upcoming 5th generation of cellular communications (5G), expected to be commercially rolled out in 2020, the widespread assumption [1] is that single users may require data rates of multiple Gbit/s, for instance due to:

- permanent cloud connectivity (for instance storing photos, videos or documents on cloud servers and accessing them instantly anytime and anywhere, instead of storing these on devices);
- a widespread usage of high definition 4K video, for instance for multi-peer video-telephony on the move, or in the context of personalized video (for instance the option to watch a sports event from an individually chosen camera viewpoint); or
- the usage of augmented reality in everyday life (for example the use of glasses which overlay additional information to the real world, for the purpose of shopping, navigation, education etc. An example for the latter would be that when a person visits ancient ruins, augmented reality glasses would render the image of the original century-old buildings on top of the real world).

These and other data-hungry application examples are illustrated in **Figure 1**.

Figure 1 Examples of future mobile applications that are predicted to be exceptionally data-hungry.



To make things even more challenging, it will not be sufficient any more in the future to provide high data rate wireless communications in some hotspot areas as today, but mobile applications will rely more and more on the fact that extreme mobile broadband connectivity is available on the move anytime and anywhere. Furthermore, users in close proximity may often have correlated connectivity needs. As an example, many visitors of a crowded sporting event may at the same time desire to experience a personalised view of the event on their mobile device. This means that an immense wireless data pipe has to be provided temporarily in the confined local area in which the sport event is taking place.

In general, it is only possible to substantially increase wireless capacity if a) wireless transmissions are made more efficient, which is rather difficult considering that cellular communications have already matured over four generations of mobile technology, b) more access points are deployed per area, and c) more spectrum is used for wireless communications, in the form of more and wider frequency bands used for transmission.

Regarding the latter issue, it is unfortunately becoming very difficult to acquire new spectrum in the “usual” frequency bands deployed for cellular communications, i.e. below 6 GHz. Instead, the only possibility to get access to substantially more spectrum is to go for significantly higher frequencies beyond 6 GHz and up to 100 GHz. At these frequencies, the wavelength of the signals is in the order of a few millimeters (as opposed to 15 centimeters in the case of 4G transmission at or around 2 GHz carrier frequency), and hence one typically refers to these higher frequencies as *millimeter-wave* (or *mm-wave*) frequencies. The key difficulty is that the signal propagation conditions for mm-wave frequencies are very different to those at lower frequencies [2], which means that completely new communication technology has to be developed.

In this paper, we venture in more detail into the specfiic challenges of utilizing mm-wave communications, and discuss various solutions that Nokia Bell Labs are developing to overcome these.

2. Key Challenges for mm-wave Communication

As mentioned before, a major challenge related to the usage of mm-wave frequencies for wireless communication is its very distinctive signal propagation characteristics when compared to lower frequencies. This is mainly due to the fact that signals at mm-wave frequencies hardly experience any diffraction, meaning that any objects that are in the way cast fairly hard shadows, as opposed to lower frequencies (i.e. < 3 GHz), where signals are to some extent capable of "bending around corners". This means that trees, building structures, materials and even human bodies can cause blockage of mm-wave signals, in a similar way as this is the case for visible light. This effect is illustrated in **Figure 2**, where three mm-wave access points are deployed, and the colors on the ground indicate the areas best served by their respective access points (indicated in blue, purple, and yellow).

As can be seen, the mm-wave "shadow" of each object such as a tree, human etc. leads to the fact that an alternative mm-wave acess point has to be used to serve any device in this shadow. Another major challenge is that mm-waves are strongly attenuated when propagating through the air as compared to lower frequencies. Thus, when using higher frequency bands, a comparatively larger *path loss* needs to be compensated, which sets limits to the possible distance between mobile devices and access points. In addition, environmental factors such as high humidity, rain, fog or snow, can lead to further attenuation of mm-wave signals and consequently further increased path loss.

On the positive side, an increased carrier frequency has one benefit, namely the fact that this implies shorter wavelengths, which means that more compact antennas can be used. In particular, this facilitates the usage of *massive antenna arrays*, consisting of grids of many small antenna elements, which allow focusing mm-wave signals into a particular direction. This technique, referred to as *beamforming*, can help overcome the aforementioned high path loss. A typical assumption in this respect would be to use arrays with 4x4, 8x8 or even 16x16 antenna elements, which allow to create narrow signal *beams* that are only a few degrees wide. Please note that the term beamforming does not only apply to the transmitter side, but also a receiver may apply beamforming in the sense of using its antenna array to focus specifically on receiving signals coming from a certain direction. Typically, mobile devices would only be able to accomodate small antenna arrays, for example with 2x2 antenna elements, hence the created receiver-side beams are still comparatively wide. An example illustration of a mm-wave antenna array deployment is given in **Figure 3**.

Figure 2 Example of mm-wave propagation. The different colors on the ground indicate the cells used to serve a particular area.

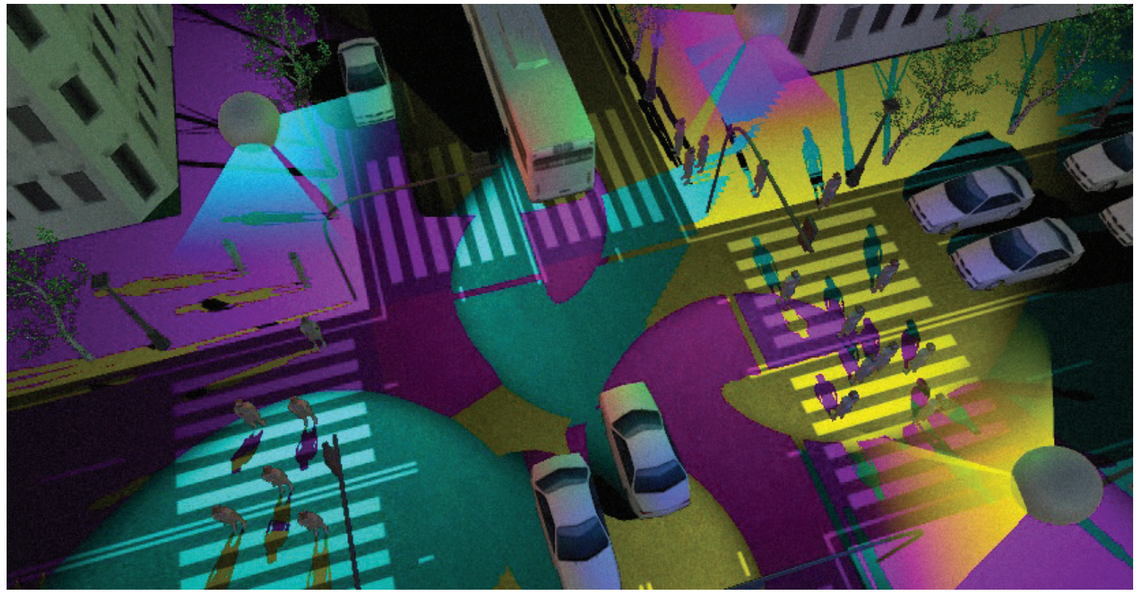
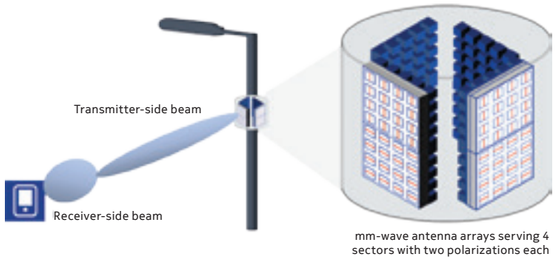


Figure 3 Example of a mm-wave antenna array deployment and the narrow signal beam created towards a user.



In this example, multiple 4x4 antenna arrays are mounted on a lamp post to form 4 sectors pointing in different directions, with each using two different polarizations. For one of the antenna arrays, the narrow beam created towards a device is depicted, along with the less narrow receiver-side beam.

However, even if beamforming can help to overcome higher path losses for mm-wave frequencies, its usage has major implications on the overall design of the communication system: For instance, before any form of communication can actually take place between a device and a mm-wave acess point, the two sides need to "find" each other in the sense of finding out which transmit and receive beams to use to communicate with the other side. It is typically assumed that this is done by letting both sides transmit so-called reference signals over multiple possible transmit beams, while the other side attempts to detect these signals by scanning through all possible receive beams. Once the right transmit and receive beams have been found, one may use *beam tracking* to ensure that the beams are adjusted correctly while a device is moving. A general problem related to mm-wave communications is that it is difficult to handle so-called common information signals, which are signals containing information that is of relevance to all devices in a cell, and which are in conventional wireless systems simply broadcast to everybody. In a mm-wave system, reliable transmission of such common information signals would also require beamforming, which would mean that such signals would have to be sent redundantly using multiple possible beams.

The usage of beamforming has the positive effect that interference between communicating entities, for instance between adjacent access points, is on average strongly reduced. However, in some situations, it may happen that devices or access points are suddenly hit by interfering beams that are exactly pointing at them. To avoid this, means for coordinated interference management are needed, meaning that access points need to be able to coordinate their transmissions and possibly refrain from transmissions that could cause severe interference to others.

Beyond the mentioned propagation challenges, mm-wave communications also suffer from hardware impairments such as phase noise, I/Q imbalance, sampling jitter, sampling frequency offset, and power amplifier nonlinearity, all of which are effects that increase with carrier frequency and must be taken into consideration into the design of the mm-wave radio technology [3].

In conclusion, mm-wave communications are subject to significantly more hostile radio propagation conditions than lower frequency communications, and require the usage of massive beamforming to overcome these. A key problem is that even with – or partially because of – the usage of beamforming, mm-wave signals become very volatile, i.e. it can happen within milliseconds that a previously strong communication link is lost. In the following sections, we will venture into different solutions to overcome this.

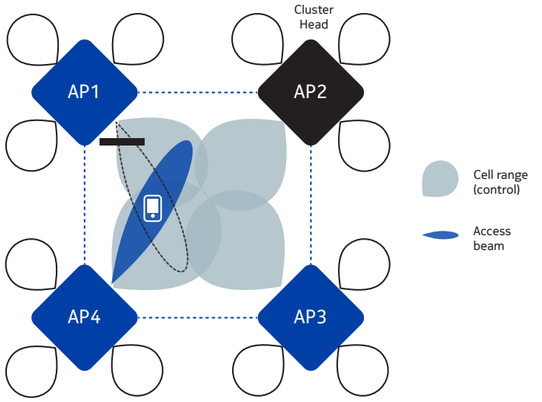
3. Possible Solutions for mm-wave Communications

3.1. Fast device handover by device-centric clusters of mm-wave cells

The volatility of mm-wave links due to aforementioned propagation characteristics and potential link blockages create the need for frequent changes of the access point that is serving a given device. An example of an extremely challenging blockage condition would be to serve a device in a crowd of people where each single person obstructing the direct link between device and access point, or even the simple turning of the head of the person carrying the device, would render a handover to another access point necessary. Hence, the handover procedure between access points has to be extremely fast and lean.

In LTE, any handover between cells involves the mobility management entity and a serving gateway. In case of frequent handovers between mm-wave access points, these entities would be heavily loaded with control signaling, and devices would suffer handover delays related to updating the data path from the old to new access points.

Figure 4 Illustration of cell clusters created for an individual device. In this example, the device is initially served by AP 1, but due to blockage from AP 1 it is then served by AP 3.



For mm-wave communications, it is therefore proposed to handle mobility locally among clusters of access points and without the continuous involvement of the mobility management entity [4]. Such clusters would be created and updated in a device-centric way, meaning that the access points in the vicinity of the device would always form a mobility cluster for that device. In this way, all cells in the cluster would be ready for a quick handover of the device when needed. This is illustrated in **Figure 4**. In each cluster of cells there is a *cluster head*, which is responsible for selecting the serving cell for the device and for routing data in between buffers within the cluster. The allocation and dynamic reallocation of the cluster head is done based on information on the deployment topology, as well as instantaneous device locations and traffic demand.

3.2. Mobility amongst mm-wave access points for inactive users

When designing solutions allowing the mobility of users in between mm-wave access points, one has to also consider inactive users who are temporarily not transmitting or receiving data. In fact, most Internet applications are characterized by short bursts of data transmission followed by periods of inactivity. In particular in the context of very dense mm-wave deployments, it would be tedious to permanently keep users connected and hand them over between different access points, even when they are inactive.

For this reason, there is the consideration in 5G to introduce a novel device state. In addition to the currently existing options in LTE of a device being *connected* (having an ongoing transmission) or *idle* (being disconnected from the system), a device could be in an *inactive_connected* state [5]. In this new state, the device would not be handed over from one cell to another by the system when moving, but would instead measure cells in its close proximity and decide by itself on the most suitable cells to attach to. This would strongly reduce the signaling effort required in the network. The key difference of the new *inactive_connected* state to that of the *idle* state in LTE would be that both the device and the network would maintain the so-called *device context*, which is a data set related to the identity and authentication of the device, among other aspects. In addition, the core network would be left in the belief that the device is constantly connected, such that only the radio access network would be aware that the device is currently inactive. In case the device has to become active again, it could rapidly switch to the fully connected state using light-weight signaling, rather than following the lengthy signaling procedure to switch from *idle* to *connected* state. By the inclusion of this third mobility state, one could hence enhance mobility of an inactive device amongst mm-wave access points with low signaling overhead, and ensure that a device can switch much faster from inactive to active than in legacy systems.

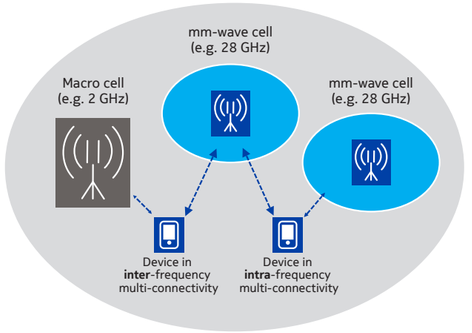
3.3. Multi-connectivity

Instead of handing over a user device from one mm-wave cell to another, an alternative option would be to serve the device from multiple points at the same time. Multi-connectivity is a feature used to aggregate bandwidth, increase reliability, and improve mobility robustness, meaning the reliability at which a device can continuously be served whilst moving among different cellular access points. One typically distinguishes between *intra*-frequency multi-connectivity (a device being simultaneously connected to multiple cells operating at the same frequency band) and *inter*-frequency multi-connectivity (the case where a device is being served simultaneously by multiple frequency bands), as illustrated in **Figure 5**.

In 5G, it is envisioned to use both intra-frequency multi-connectivity, that is, to allow a device to be connected to multiple mm-wave access points for the sake of better robustness against sudden signal blockages, as well as inter-frequency multi-connectivity between mm-wave bands and lower frequency bands. The latter appears especially important to ensure that users which are instantaneously not servable through mm-wave communications at least obtain decent data rates on the lower frequency layer that reliably covers a much larger area than the mm-wave cells.

For the exact implementation of multi-connectivity, one distinguishes between whether cells involved in multi-connectivity are co-deployed or not, or in other words, whether the signal processing for multiple cells is done in the same place or not. In cases of co-deployment, it is possible to realize multi-connectivity on the so-called medium access control (MAC) layer of the communication system, meaning that in every fraction of a millisecond the co-located access points would be able to decide how to best make use of the resources of the different links for a given device. This would allow a most dynamic adaptation to the changing radio conditions of the involved links. If there is no co-deployment, such a tight form of multi-connectivity is not possible, as it would be infeasible to obtain a fast enough information flow between the involved access points. In this case, one would rather aggregate multiple data streams in different cells on the so-called packet data convergence protocol (PDCP) layer, which means that it is decided on a much slower time scale and asynchronously to the radio transmission which data streams are sent over which radio to the device.

Figure 5 Illustration of intra-frequency and inter-frequency multi-connectivity.



It should be noted that in the case of inter-frequency multi-connectivity, the lower frequency layer could be a newly designed 5G wide area radio technology providing wide coverage. However, especially for early 5G deployments this coverage layer could also be LTE-based. This allows for a seamless transition from LTE to 5G, as 5G technology can initially be introduced in local hotspots providing very high throughput, while benefiting from the excellent LTE coverage.

3.4. Decoupling of control and user plane

In the previous discussion, we have so far always considered mm-wave technology that can be operated stand-alone, meaning that this provides all required means for data communication and control. Due to the aforementioned volatility of communication links at mm-wave frequencies, however, there are various considerations to actually use more reliable lower frequency layers for most of the control signaling, and use the mm-wave bands mainly for providing high-rate data communication. In technical terms, one here typically talks about separating the *control plane* from the *user plane*. There are various different options, which differ in the extent of control signals that are handled by a lower frequency band, and those that are still handled by the mm-wave layer itself.

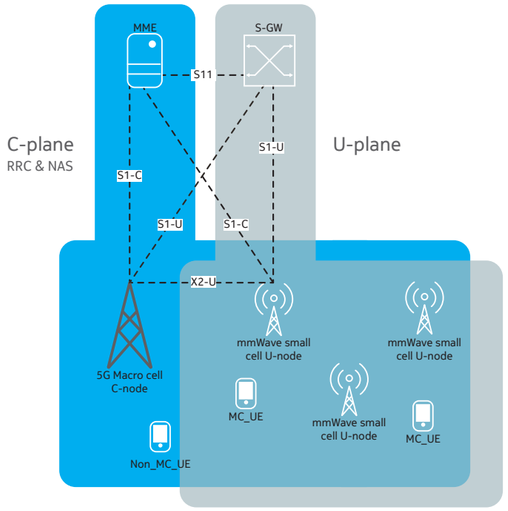
In one option, only those control signals closely related to the actual data transmission, such as synchronization signals, reference signals for channel estimation purposes, acknowledgments for transmissions etc. are handled by the mm-wave cells. All other control signals such as system information, the transmission of scheduling grants, mobility related signaling etc. are handled by the lower frequency layer. In such a setup, a device is in fact not aware which exact mm-wave cell(s) it is communicating with, as it only sees the lower frequency cell as an entity, while the mm-wave cells are transparent to the device. For this reason, this approach is often termed *phantom cell* concept [6]. A key requirement of the approach is that the mm-wave cells are tightly coupled with the lower frequency cell by, for example, using centralized processing of all cells and fiber-optic cables from the central processor to the mm-wave radio heads.

In an alternative approach with less stringent infrastructure requirements, the mm-wave cells would handle essentially all control signaling themselves, except that related to radio resource control (RRC), and non-access-stratum signaling (NAS), both related to mobility aspects. Key benefits are:

- A better and independent scalability of control and user plane, which enables network operators to gradually add capacity to their networks;
- More flexibility in activating and assigning control and user plane entities to users based on service requirements. For instance, high throughput requiring users can obtain more user plane resources (e.g. massive multi-connectivity between several access points), while applications requiring ultra-high reliability can utilize highly redundant control plane signaling;
- A fast user plane setup, which is for instance critical in the context of fast mobility between mm-wave access points as discussed in Section 3.1.

This particular control / user plane split concept is shown in **Figure 5**. Here, control plane connectivity is provided by a macro cell using a low frequency band in order to maintain good connectivity and mobility. On the other hand, the user plane is provided by a small cell layer operating at mm-wave frequencies in order to boost user data rate whenever needed.

Figure 6 Possible C/U-plane split configuration.



S-GW: Serving GateWay
MME: Mobility Management Entity
RRC: Radio Resource Control
NAS: Non-Access Stratum
U-node: User plane node
C-node: Control plane node
MC_UE: Multi-Connectivity-enabled User Equipment
Non_MC_EU: Multi-Connectivity-disable User Equipment

3.5. Self-backhauling

In the previous sections, it has become obvious that mm-wave connectivity may require very dense and redundant access point deployments in order to overcome the stated difficulties related to mm-wave propagation. However, the cost of such deployments would become rather prohibitive if each access point needs to have a wired connection to the core network and Internet, referred to as *backhaul*. Hence, it is essential to enable solutions where mm-wave access points can also establish a wireless connection to other infrastructure nodes for the purpose of *wireless backhauling*. If the same radio technology is used for both access links (between access points and devices) and backhaul links (between infrastructure nodes for the purpose of connectivity to the core network), this is commonly referred to as *self-backhauling* [7], a concept which is illustrated in **Figure 7**.

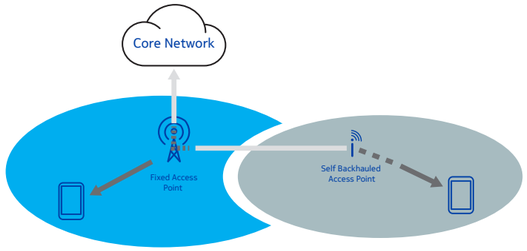
Self-backhauling provides cost-effective means to combat infrastructure constraints especially in deployments where access to fibre is limited due to the high density of access points. Self-backhauling can also play a critical role during early 5G deployments by providing means for incremental deployment, where initially the density of 5G access points with dedicated backhaul is lower, but over time, as the traffic needs grow and fixed infrastructure becomes available, self-backhauling is gradually replaced. Self-backhauled 5G access points are also expected to provide capacity extension to existing deployments by means of multi-connectivity towards multiple donor cells.

It has been seen in the context of LTE that it is difficult to introduce self-backhauling (in LTE termed *relaying* and focused mainly on coverage extension) as an add-on to an existing standard. It is hence especially important to ensure that the 5G standard already natively supports self-backhauling.

4. Summary

Millimeter-wave communications will give access to large amounts of new spectrum and add significant local capacity to 5G networks. To benefit from the new spectrum, however, various challenges related to the propagation at higher frequencies have to be overcome. For this, fast and lightweight mobility solutions are required as well as the support of multi-connectivity among multiple mm-wave access points, or among mm-wave bands and lower frequency bands. As an interesting cellular design option, it was shown that one may split the control signaling from the actual data transmission in 5G networks in such a way that the control signaling is handled over more reliable lower frequency links, while mm-wave bands are used for high-rate data connectivity. It was further highlighted that self-backhauling is an essential aspect of mm-wave deployments, as it cannot be expected that each access point has wired backhaul to the core network available.

Figure 7 Concept of self-backhauling.



References

[1] International Telecommunication Union – Radio (ITU-R) Working Party WP 5D: Draft New Recommendation ITU-R M.2083-0 (09/2015), "IMT Vision – Framework and overall objectives of the future development of IMT for 2020 and beyond", September 2015.

[2] S. Rangan, T. S. Rappaport, and E. Erkip, "Millimeter-wave cellular wireless networks: Potentials and challenges", Proceedings of the IEEE, vol. 102, no. 3, pp. 366–385, March 2014.

[3] S. G. Larew, T. A. Thomas, M. Cudak and A. Ghosh, "Air interface design and ray tracing study for 5G millimeter wave communications", IEEE Globecom 2013 Workshops, Atlanta, GA, Dec. 2013.

[4] M. Saily, A. Vijay and S. Hailu, "Novel connectivity for 5G Telecommunication Systems", EuCNC-2016, June 2016.

[5] I. Da Silva, G. Mildh, M. Säily and S. Hailu, "A novel state model for 5G radio access networks", IEEE International Conference on Communications (ICC), 5G RAN Design Workshop, May 2016.

[6] H. Ishii, Y. Kishiyama and H. Takahashi, "A Novel Architecture for LTE-B, C-plane/U-plane Split and Phantom Cell Concept", IEEE Globecom 2012 Workshops, December 2012.

[7] J. Luo et al., "Millimeter-Wave Air-Interface for 5G: Challenges and Design Principles", ETSI Workshop 2016, January 2016.

About the authors

Patrick Marsch received his Dipl.-Ing. and Dr.-Ing. degrees from Technische Universität Dresden, Germany, in 2004 and 2010, respectively. After leading a research group at TU Dresden, Germany, he is now heading a research department within Nokia Bell Labs, Wroclaw, Poland. He has published 50+ journal or conference papers, obtained 4 best paper awards, and received the Philipp Reis Prize for pioneering work in the field of Coordinated Multi-Point. Patrick was the technical coordinator of the project EASY-C, where the world's first large-scale testbeds for LTE-A techniques were established, and is the technical manager of the 5G PPP project METIS-II.

Patrick Marsch
Manager, Radio Research
CIOO Bell Labs

Arnesh Vijay graduated with a PhD in telecommunication engineering from Warwick University, England – specializing in visible light technology. He is alumnus of University of Glasgow, Scotland, where he obtained a Master's degree in electrical & electronic engineering. Arnesh is a member of a number of technical institutions across the UK, and was awarded Associate Fellowship in 2014 by the HEA (Higher Education Academy). Alongside his current research within Nokia Bell Labs, Arnesh is involved in mm-wave architecture topics in the 5G PPP project mmMAGIC.

Arnesh Vijay
Engineer, Radio Research
CIOO Bell Labs

Matthias Hesse received a Dipl.-Ing. degree in electrical engineering from the Dresden University of Technology in 2006, and his Ph.D. degree in information technology and communications from the University of Nice, Sophia-Antipolis, France, in 2010. Since 2010, he is with Nokia Bell Labs in Wroclaw, Poland, formerly Nokia Siemens Networks. Matthias is currently the technical manager of a large-scale internal research project on 5G radio access architecture, procedures and protocols.

Matthias Hesse
Senior Engineer, Radio Research
CIOO Bell Labs

5G Flexible Design: Enhancements for Retransmission Techniques

Saeed R. Khosravirad
Senior Engineer, Radio Research
CIOO Bell Labs

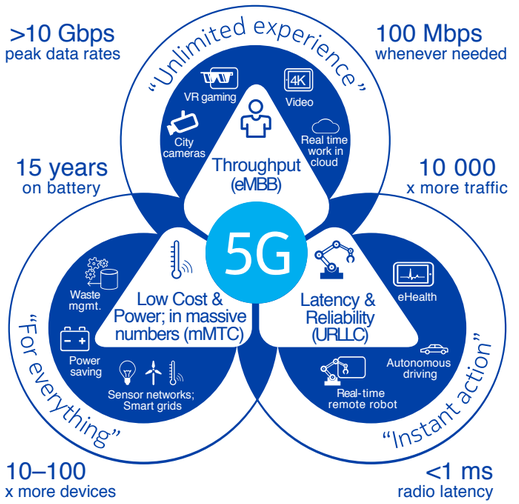
NOKIA

1. Introduction

The research and development work for the new generation of mobile networks, famously referred to as 5G, has taken a great momentum. The broad vision for 5G is to have a system design that acknowledges the ever-increasing diversity of use cases for wireless telecommunication networks. Downloading movies with a Gigabytes per second (GB/s) data rate, streaming real-time high quality video, remote automation for industrial purposes, autonomous driving and ultra-dense sensor networks are among the many different use cases that seek a reliable and ubiquitous wireless communication. 5G is expected to cover this long list of known service requirements as well as the services that are still unknown but yet to be introduced to the market.

This article addresses the essential role of the retransmission techniques in providing a fast and reliable wireless data delivery. We outline disruptive enhancements that are proposed for retransmission techniques in 5G. In summary, the proposed enhancements are motivated by the need for increased timing flexibility, richer information over feedback channel to assist with better utilization of radio resources and having more simplified variants of the retransmission techniques for low-cost/low-energy use cases.

Figure 1 The fifth generation of mobile networks is envisioned to support a vastly diverse set of services with extremely different and somewhat conflicting requirements



We discuss how these enhancements will help add to the design degrees of freedom towards a flexible 5G system.

2. Flexibility trend in the 5G radio technology design

The challenging usage scenarios and applications outlined by the International Telecommunication Union (ITU) for the next generation of mobile radio [1] sets high standards for researchers and engineers towards designing a system that provides services ranging over high reliability, extreme low latency and infinite capacity "anytime/anywhere". In line with these usage scenarios, the 3 Generation Partnership Project (3GPP) has set the related requirements for 5G in [2] as GB/s data rates, less than a millisecond (ms) over the air latency and low-energy-consuming communication with a vast number of low-cost devices (**Figure 1**). Accordingly, the challenging requirements introduced for enhanced mobile broadband (eMBB) with more than 10 GB/s peak data rate and less than 10 ms latency calls for much larger bandwidth for transmission compared to LTE. The massive machine-type communications (mMTC) targets a connection density of a million devices per square kilometer of an urban environment. This, together with a battery life target of 15 years makes mMTC a challenging use case for 5G. The ultra-reliable low latency communication (URLLC) on the other hand, calls for extreme reliability of $1\text{-}10^{-5}$ successful delivery rate in a sub-millisecond latency communication.

The different use cases of 5G technology as explained above have somewhat conflicting optimization targets for throughput, latency and reliability. Therefore, the system design for 5G needs to be extremely flexible and scalable with the possibility to optimize each individual link in coherence with its requirements [3]. A flexible air interface is therefore the forefront vision for 5G technology design. Moreover, 5G is to offer communication over different network implementations, ranging from traditional distributed base station architecture to the advanced centralized network topologies, each encompassing different latency and reliability potentials.

3. Retransmission techniques in radio communication

Fast and reliable data delivery over radio channels is always susceptible to fading due to unpredictable changes in the channel quality. Therefore, regardless of its attributes, the first transmission may fail in delivering the message and one or multiple retransmission attempts will be necessary to eventually guarantee a successful decoding. A retransmission technique in its simplest form performs an error detection on the received packet, for example by the means of cyclic redundancy check (CRC), and sends a positive acknowledgement feedback (ACK) or a negative acknowledgment feedback (NACK) to report success or failure in decoding respectively. This operation relies on a feedback channel that needs to be highly reliable. The transmitter

node in case of receiving an ACK will proceed with transmitting the next available packet in its buffer. In the case of NACK feedback, the same packet will be retransmitted and this will be repeated until the packet is successfully delivered or maximum allowed number of transmission attempts is reached.

The two basic types of retransmission techniques that are widely in use in wireless telecommunication industry today are the automatic repeat request (ARQ) and hybrid ARQ (HARQ). While the two techniques work quite similarly in principles of “feedback” and “retransmission”, they treat the received versions of the packet differently. In other words, decoding of a packet in ARQ technique is only based on the latest received version of it while in HARQ all the previously received versions of the same packet are stored in a buffer and used together with the latest received version to increase the chances of successful decoding. The difference between the ARQ and HARQ techniques is then in the principle of combining which is useful in improving the delivery success rate hence, ARQ can be seen as a simpler variant of HARQ.

In the wireless communications standards such as the 3GPP Long Term Evolution (LTE), HARQ in the form of stop-and-wait (SAW) – as it is explained in **Figure 2** – is typically exploited for transmission over the radio channel where the transmission is more susceptible to fading. In this case, using error correction codes together with the retransmission combining will help generate a reliable and efficient link. ARQ will however be used in a higher layer, on top of HARQ, to increase the reliability of the transmission with a lower complexity compared to HARQ. Thanks to HARQ technique in the lower layer, ARQ will experience a more reliable channel and will typically only rely on error detection coding.

4. Overview of the disruptive design enhancements for retransmission techniques in 5G

The work on introducing enhancements to the HARQ technique for 5G has attracted a significant research effort. The design procedure of a flexible HARQ relates directly with the control channel design, signaling overhead considerations, memory buffer attributes, data delivery latency, resource management, communication reliability and simplicity of operation. Hence, it is crucially important to take all these related parameters into account while adapting HARQ design to the use case. The inevitable trade-offs among the mentioned parameters must be steered carefully towards a best-fit for each service based on its requirements.

Figure 2 The concept of stop-an-wait HARQ channel is depicted. HARQ retransmission technique in LTE is deployed in the form of SAW channels where transmissions are controled by the MAC layer.

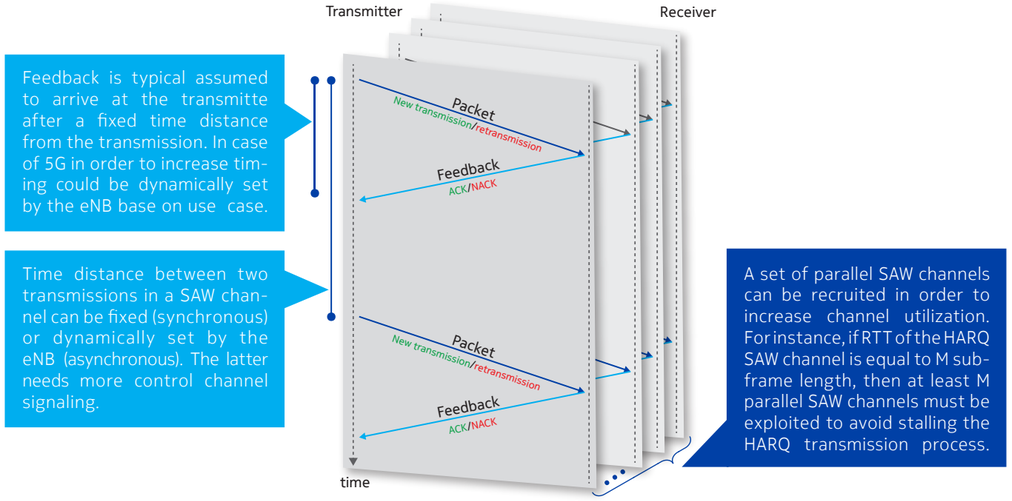
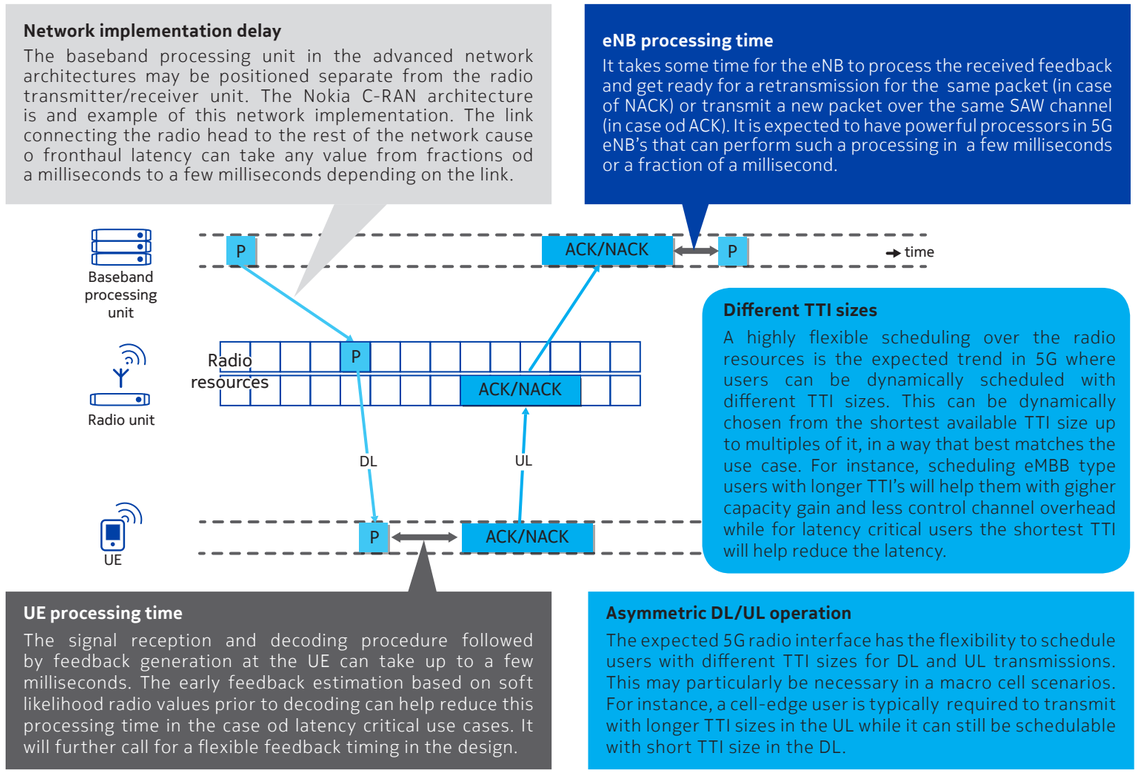


Figure 3 Timing considerations in RTT calculation for HARQ includes system design parameters as well as link operation parameters.

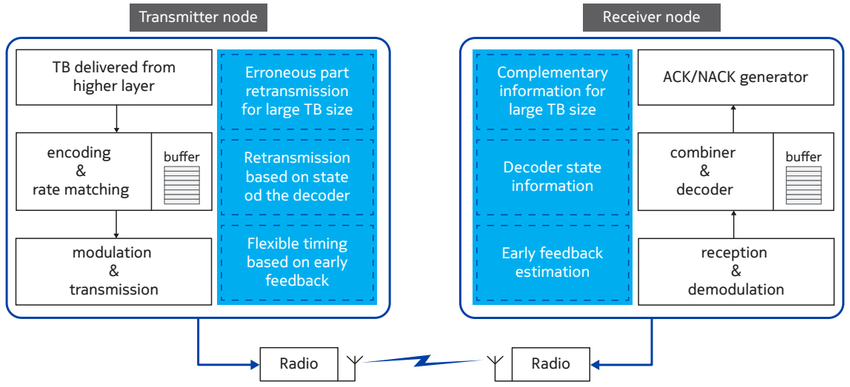


4.1. Timing flexibility

The round trip time (RTT) of a HARQ process is the time it takes from the first transmission until the retransmission of the same data or transmission of a new data can be performed over the same SAW channel. The RTT duration together with the failure chance of the transmissions determine the average latency caused by HARQ technique. For instance, for the case of HARQ process with maximum two allowed transmission attempts and 10% chances of failure for the initial transmission, the average latency of a packet over the SAW channel is 1.1xRTT. Therefore, especially for latency critical use cases it is crucial to efficiently minimize RTT duration. However, for the case of eMBB, in order to serve users with a higher throughput using long transmission time intervals (TTI's), longer RTT is a more suitable choice.

The number of parallel SAW channels and the number of memory units in the buffer in the medium access (MAC) layer corresponds directly with RTT duration. In general, RTT duration is affected by factors like: processing time at the communicating nodes, TTI duration, network implementation delay, and asymmetric link operation. The effect of these parameters on HARQ RTT are depicted and further explained in **Figure 3**.

Figure 4 Enhancements regarding feedback generation for 5G HARQ are depicted in blue boxes together with the traditional HARQ model.



In case of TDD operation, the transmission direction switching between DL and UL adds to the list of parameters that affect RTT duration. Particularly, for flexible TDD operation where it is possible for a cell to switch arbitrarily between DL and UL directions, it is further challenging to monitor the HARQ feedback and retransmission timings. The main mode of timing operation for HARQ in 5G is therefore expected to be asynchronous timing between the transmissions in a SAW channel as well as supporting flexible timing for feedback transmission [4].

Furthermore, the use of early feedback estimation techniques to assist the latency critical use cases by reducing HARQ RTT duration makes the timing considerations more challenging. It is shown that the HARQ feedback can be estimated prior to the decoding process by more than 90% prediction success rate. This could potentially reduce RTT duration by omitting around 60% of the processing time that is contributed by the decoder modules [5]. The RTT can be further reduced by estimating the HARQ feedback only based on a portion of the received packet. This calls for additional research on the related solutions.

4.2. Rethinking feedback

As mentioned earlier in this article, the feedback message for HARQ process is traditionally a rigid Boolean type value that conveys ACK or NACK. However, studies show that this could be severely inefficient compared to the case where feedback carries extra information regarding "how close" the decoder is to a successful decoding [6]. The additional information which is referred to as "enriched feedback" [7] can be used by the transmitter to make an optimal decision on retransmission attributes. For instance, the amount of radio resources to be used by the retransmission attempt can be decided efficiently based on use case, to fulfill a given decoding failure chance while increasing resource utilization.

The reported works on optimization of the retransmission resources based on a multi-bit feedback in some LTE-alike scenarios show that on average the retransmission size can be efficiently reduced to less than 20% of the initial transmission resource size in more than 80% of the cases [7]. This will free up radio resources that could be used to transmit other packets and increases the channel utilization efficiency.

Furthermore, the 5G carriers are expected to span over larger bandwidth compared to LTE. This includes carriers with 100 MHz or up to GHz bandwidths for below 6 GHz frequency and millimeter wave bands respectively [3]. Therefore, it is expected to have increasingly large TB sizes delivered to HARQ SAW channels in the 5G MAC layer. The immediate concern for the large-size TB's is to efficiently optimize the potential HARQ retransmissions. This calls for a flexible-size multi-bit feedback that can dynamically point to the erroneous parts of a failed TB as opposed to a single-bit feedback which could cause throughput loss by triggering retransmission of the whole TB.

4.3. Low-cost adaptability

The challenging battery life and energy consumption requirements for the mMTC use case calls for a fundamentally different HARQ design paradigm as compared to e.g. eMBB or URLLC cases, to support low-cost and low-energy communication. In general, it is expected to have support for devices with minimized memory sizes, simplified narrow-band half-duplex transceivers and low complexity processing units that need to perform small to medium packet size communications with a low average data rate.

The small memory size of the mMTC devices can naturally support only one or a few parallel SAW channels. HARQ timing must preferably support synchronous mode of operation for these devices to limit the unnecessary energy-consuming searches for retransmissions over the radio resources. Furthermore, half-duplex communication limitations for low cost devices needs to be taken into account in the feedback timing of the HARQ transmission.

5. Summary

The work on 5G design targets a flexible and scalable system that can be optimally adapted to best serve each use case. The set of enhancement proposals summarized in **Table 1** are in line with this target in order to accommodate 5G with better service-oriented design parameters.

Table 1 Summarized enhancements expected for 5G HARQ

	Motivation summary	Target enhancements
Flexible timing and early feedback estimation	• Variable TTI size with asymmetric DL/UL operation • Different network implementations with different latencies • Reducing processing time and HARQ RTT for URLLC use cases	• Asynchronous HARQ with per-link configuration of feedback and retransmission timing • Per-link configuration of buffer size and number of SAW channels • Early estimation of the decoder output
Flexible-size multi-bit enriched feedback	• Large TB sizes expected for 5G • Avoid retransmissions of unnecessary segments of the packet and optimize resource usage for retransmissions	• Dynamically point to the erroneous parts of a failed TB • Multi-bit feedback information on "how close" the receiver was to successful decoding • Use and optimal retransmission resource size
Low-cost adaptability	• Communication with a vast number of low-cost and low-energy devices	• One or few HARQ SAW channels per link • Relaxed timing flexibility for simplicity

References

[1] IMT Vision, "Framework and overall objectives of the future development of IMT for 2020 and beyond," International Telecommunication Union (ITU), 2015.

[2] 3GPP, "Technical Report (TR) 38.913. Study on scenarios and requirements for next generation access technologies," 2016.

[3] K. Pedersen, F. Frederiksen, G. Berardinelli and A. Szufarska, "A Flexible 5G Frame Structure Design for Frequency-Division Duplex Cases," *IEEE Communications Magazine*, March 2016.

[4] S. R. Khosravirad, G. Berardinelli, K. I. Pedersen and F. Frederiksen, "Enhanced HARQ design for 5G wide area technology," in *IEEE VTC Spring*, Nanjing, 2016.

[5] G. Berardinelli, S. R. Khosravirad, K. Pedersen, F. Frederiksen and P. Mogensen, "Enabling early HARQ feedback in 5G networks," in *IEEE VTC Spring*, Nanjing, 2016.

[6] L. Szczecinski, S. R. Khosravirad, P. Duhamel and M. Rahman, "Rate allocation and adaptation for incremental redundancy truncated HARQ," *IEEE Trans. Commun.*, vol. 71, no. 6, pp. 2580-2590, 2013.

[7] S. R. Khosravirad, K. I. Pedersen, L. Mudolo and K. Bakowski, "HARQ Enriched Feedback Design for 5G Technology," in *IEEE VTC Fall*, Montreal, 2016.

About the author

Saeed R. Khosravirad received his Ph.D. degree in Telecommunications in 2015 from McGill University in Canada. Before that he had received the M.Sc. degrees in Electrical Engineering from Sharif University of Technology, Iran in 2009 and the B.Sc. degree in Electrical and Computer Engineering from Tehran University, Iran in 2007. He is currently with the Nokia Bell Labs as a researcher in 5G radio access technology. His research fields of interest are information and communication theory and wireless systems design where lately, his work has focused on system related issues, including cellular network modeling and radio resource management.

Saeed R. Khosravirad
Senior Engineer, Radio Research
CIOO Bell Labs

5G Spectrum: Challenges and Perspectives

Kamil Bechta
Senior Engineer, Radio Research
CIOO Bell Labs

Fabiano Chaves
Senior Engineer, Radio Research
CIOO Bell Labs

NOKIA

5G will start dynamic exploration of radio spectrum above 6 GHz. Ranges of centimetric waves (3 GHz–30 GHz) and millimetric waves (30 GHz–300 GHz) will allow the allocation of much wider channel bandwidths per single user than in any previous mobile system, but at the same time introduce more challenging radio propagation conditions. According to the regulation and pre-standardization agreements, study on 5G has been restricted to particular frequency bands. However, extensive investigations on spectrum sharing techniques raise expectations that in the near future 5G will be able to share radio frequency bands with other systems, what may influence future decisions of regulatory bodies in regards to bands globally allocated for 5G.

1. Vision of 5G for different spectrum bands

5G system is expected to operate in a wide radio frequency range. Depending on planned cell layouts, especially cells sizes, different frequency ranges are predefined for 5G cells deployment. In general, due to the decrease of radio waves propagation distance with the increase of radio frequency, large macro cells are planned to be deployed mostly on frequencies below 6 GHz, whereas such restriction does not apply to micro cells, where deployment is expected also in frequencies above 6 GHz.

Based on a study conducted in 3GPP, it is possible to identify optimal mapping between cell type/size and applicable frequency range. Good examples of such mapping can be found in [1] where typical deployment scenarios for 5G have been defined. **Table 1** below presents only a part of deployment scenarios described in [2].

Table 1 Examples of 5G deployment scenarios from the perspective of spectrum allocation

Deployment Scenario	Carrier Frequency	Channel Bandwidth
Indoor Hotspot (micro)	Around 4 GHz	Up to 200 MHz (DL + UL)
	Around 30 GHz	Up to 1 GHz (DL + UL)
	Around 70 GHz	Up to 1 GHz (DL + UL)
Dense Urban (macro + micro)	Around 4 GHz + Around 30 GHz	Around 30 GHz: Up to 1 GHz (DL + UL) Around 4 GHz: Up to 200 MHz (DL + UL)
Rural (macro)	Around 700 MHz	Up to 20 MHz (DL + UL)
	Around 4 GHz	Up to 200 MHz (DL + UL)
	Around 700 MHz + Around 2 GHz	

The size of the cell is associated with the carrier frequency and also the channel bandwidth. As reflected in **Table 1**, the higher the frequency range, the more spectrum can be made available per single channel to meet more demanding expectations of end users on high data rates, especially in highly populated environments like Indoor Hotspots or Dense Urban scenarios.

Also from the point of view of specific 5G services, spectrum is one of the key enablers. For example enhanced Mobile Broadband (eMBB), which is expected to introduce significant increase in end user data rate experience, will require sufficient channel bandwidths, which are available only in higher frequency range. Opposite conditions may be expected in case of massive Machine Type Communication (mMTC), where data rate requirement is less important than wide cell coverage and reliable connection with network, which is easier to obtain in lower frequency range.

2. Decisions of WRC-15 on 5G spectrum studies and other spectrum bands of interest

The latest update on the International Telecommunication Union (ITU) Radio Regulations took place in the World Radiocommunication Conference 2015 (WRC-15). A number of important decisions affecting the mobile industry were made in WRC-15, as the identification of new spectrum for International Mobile Telecommunications (IMT) and the agreement on a new agenda item for 5G spectrum in next WRC, which is going to be held in 2019 (WRC-19).

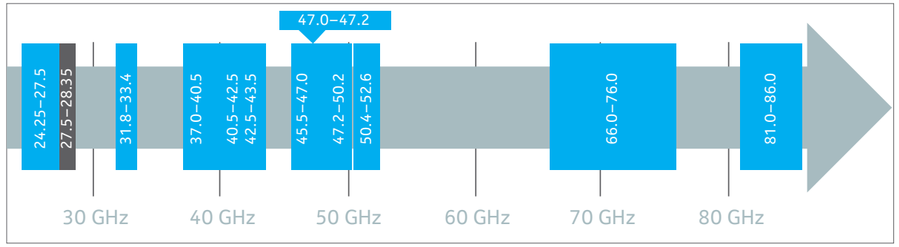
Frequency allocations in ITU Radio Regulations can be defined in a primary or secondary basis – a frequency band can be shared by multiple services of both categories – where secondary services shall cause no harmful interference to, nor claim for protection from primary services [2, Article 5, Section II]. Therefore, global and regional identification and/or allocation of frequency bands for IMT allows global equipment manufacturers to seek and reach economies of scale and reduced product costs.

2.1. New agenda item for 5G spectrum in WRC-19

5G is the concept which consists of existing systems as well as new architectures and technologies which will meet new requirements. The ITU new terminology for the purpose of future development of IMT for 2020 and beyond is IMT-2020.

In WRC-15, a wide agreement on the agenda item for 5G spectrum in WRC-19 has been made. **Figure 1** illustrates in blue the bands which are decided to be studied and potentially considered for identification in WRC-19 according to Resolution 238 (WRC-15). The band in gray is not decided to be studied after WRC-15, but is of high importance, as discussed in next section.

Figure 1 Frequency bands decided to be studied for IMT-2020 before WRC-19



The following frequency bands illustrated in **Figure 1** have allocations to the mobile service on a primary basis: 24.25–27.5 GHz, 37–40.5 GHz, 42.5–43.5 GHz, 45.5–47 GHz, 47.2–50.2 GHz, 50.4–52.6 GHz, 66–76 GHz and 81–86 GHz. The remaining bands, i.e. 31.8–33.4 GHz, 40.5–42.5 GHz and 47–47.2 GHz may require additional allocations to the mobile service on a primary basis. The ITU-R studies will be considered in WRC-19 for decision on additional spectrum allocations to the mobile service on a primary basis and/or identification of frequency bands for IMT.

2.2. Other spectrum bands of interest

Lower frequency bands benefit of better propagation conditions than higher frequencies, and are essential for some use cases and scenarios foreseen for IMT-2020, where high reliability and coverage are key factors. Therefore, new frequency bands below 24 GHz are still of interest for the development of IMT-2020, but the consideration of such bands is likely to happen at WRC-level in 2023.

For other reasons, the 28 GHz band, illustrated in gray in **Figure 1**, is likely to become "the first band in the world" where 5G is deployed. In United States, Federal Communication Commission (FCC) has continued the development of this band (27.5–28.35 GHz). South Korea has decided to give out 1 GHz bandwidth for each 3 operators for 2018 Olympics, i.e. 26.5–29.5 GHz. Therefore, depending on the success of trials and limited commercial operation of 5G in the 28 GHz band, this band can be considered for global harmonization already in 2019.

3. Evaluation of challenges and potentials of 5G spectrum bands

Most mobile traffic demand (between 70% and 80% according to many estimates) is generated from indoors or within buildings. This trend is expected to remain for IMT-2020. Therefore, the outdoor-to-indoor communication is of great importance for 5G deployment, since a typical scenario considered for 5G is composed of outdoor below-rooftop base stations serving numerous users inside buildings in dense urban environments.

The network operation in different frequency ranges has important implications on the following key radio communication components [3]:

- **Channel bandwidth**: Very wide channel bandwidths are needed to provide enhanced data rates for some 5G use cases. These can be achieved in bands above 6 GHz. Although being a key enabler to 5G, wider channel bandwidth also brings technical challenges, as a prominent noise floor rise.
- **Radio propagation**: Propagation conditions are substantially degraded in case of radio signal penetrating buildings at higher carrier frequencies, since some materials produce severe energy losses in such radio signals.
- **Antenna gain**: The increase of carrier frequency allows the deployment of a large number of antenna elements within practical antenna arrays, enabling the use of highly directive beamforming and increased antenna gain to compensate degraded radio propagation conditions. However, this antenna system is limited in practice by the challenging control of many narrow beams in both transmitter (Tx) and receiver (Rx).
- **Transmit power**: The transmit power is also dependent on the frequency of operation. It can be considerably reduced due to poor efficiency of power amplifiers in higher frequencies.

All of these components have substantial impact on the overall performance of the 5G system and can define frequency dependent limitations to the network deployment. In [3], the mentioned characteristics and behaviors of these components are modeled and considered in a feasibility analysis of 5G deployment under the condition of minimum requirements of radio link quality (minimum Signal-to-Noise Ratio (SNR)) and coverage (minimum percentage of cell edge users covered).

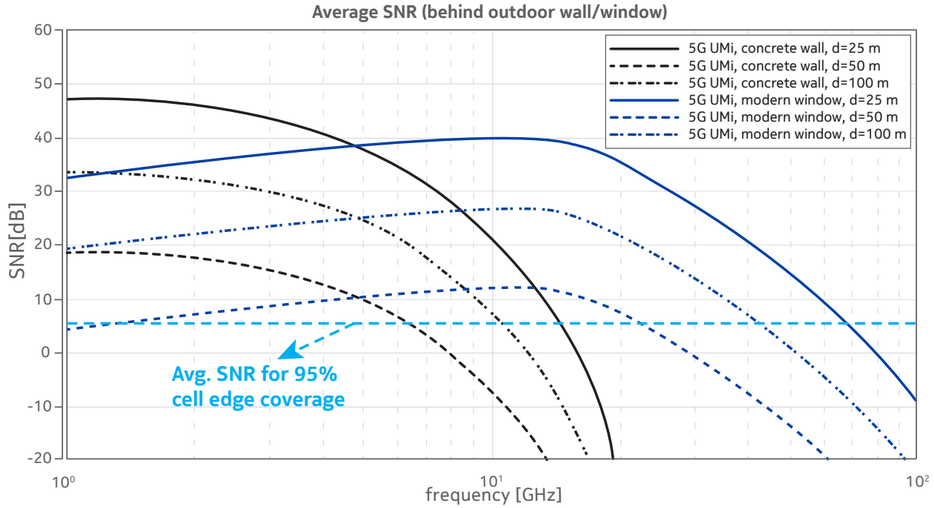
Figure 2 shows the average SNR curves for the 5G outdoor-to-indoor deployment in Urban Micro (UMi) environment, according to the frequency dependence of channel bandwidth, radio propagation, antenna gain and transmit power, modeled in details in [3]. These curves are generated for a frequency range of 1–100 GHz and three different Tx to Rx distances taken as cell edge, i.e. d = 25 meters, d = 50 meters and d = 100 meters. Separate curves for the propagation through concrete wall and modern window allow the observation of the discrepancy of these situations.

Due to the severe losses of radio propagation through concrete wall, the average SNR rapidly falls below the minimum requirement with the increase of carrier frequency, and makes cell ranges as short as 25 meters unable to operate in frequencies above 15 GHz. In case of propagation through modern windows, the increasing antenna gains in the Tx and the Rx play a role in the improvement of the signal strength at the receiver up to 14 GHz, when the Rx antenna gain reaches the maximum value. The 5G network feasibility limits in terms of cell range and carrier frequency are met at 100 meters for 20 GHz, 50 meters for 40 GHz and 25 meters for 70 GHz in case of propagation through modern windows. As the User Equipment (UE) may be subject of technical and regulatory limitations, which prevent Equivalent Isotropically Radiated Power (EIRP) levels as high as the ones of Base Station (BS), a similar link budget analysis for the reverse link could lead to more reduced cell and carrier frequency ranges for 5G outdoor-to-indoor deployments.

4. Key sharing studies: Co-existence between 5G and Fixed Satellite Service (FSS)

Decisions taken during WRC-15 excluded several radio frequency bands from the scope of 5G evaluation and sharing studies. One of these bands is 28 GHz, ruled out mainly due to potential co-existence issues with satellite systems operating in this frequency range. However, by regional regulations in some parts of the globe, 28 GHz band has been allowed for local 5G sharing study. Main activity in this area took place in United States where local mobile operators altogether with Nokia provided extensive analysis to FCC [4], [5], [6]. The aim of this analysis was the evaluation of feasibility of spectral co-existence between 5G and Fixed Satellite Systems (FSS).

Figure 2 Cell edge average SNR as a function of carrier frequency



FSS uses 28 GHz band for transmissions from ground stations towards space stations. Due to this reason, sharing study between 5G and FSS in this band was concentrated on:

- interference generated by FSS ground station transmitter towards 5G receives and
- aggregate interference generated by 5G transmitters towards FSS space station receiver.

4.1. Co-existence between FSS ground station transmitter and 5G receivers

The main purpose of this study was the determination of separation distance between FSS ground station transmitter and 5G receivers, called "compatibility distance". Optimally defined compatibility distance should ensure sufficient protection for 5G receivers against interference coming from FSS ground station, but at the same time do not cause harmful limitation in 5G network coverage in geographical areas where FSS ground stations are located.

Size of compatibility distance depends on FSS and 5G system parameters, mainly power generated by FSS ground station towards horizon and 5G receiver antenna gain. Taking into account:

- FSS ground station EIRP spectral density towards horizon (between -30 dBW/MHz and 20 dBW/MHz),
- 5G BS receiver antenna gain (between 0dBi and 30dBi),
- free space path loss and additional loss of 20 dB between FSS ground station and 5G BS,
- acceptable interference level at 5G BS equal to receiver noise floor increase by 6%,

compatibility distance between FSS ground station and 5G BS can be determined as shown in **Figure 3**.

Due to wide range of FSS ground station EIRP spectral density towards horizon and 5G BS receiver antenna gain, compatibility distance can vary between 98m (-30 dBW/MHz and 0 dBi) and 981 km (20 dBW/MHz and 30 dBi). It should be noted that these are extreme values and real deployment distances are expected to be in lower part of this range due to usage of additional spectrum sharing mechanisms. Compatibility distance can be reduced by spatial coordination of FSS ground station and 5G BS antenna patterns as well as other techniques.

Latest study recommends that to ensure acceptable level of interference in 5G receiver located at compatibility distance of 200 meters, power flux density (PFD) of FSS ground station transmitter should be reduced to -77.6 dBm/m^2/MHz at 10 meters above ground level [5], [6].

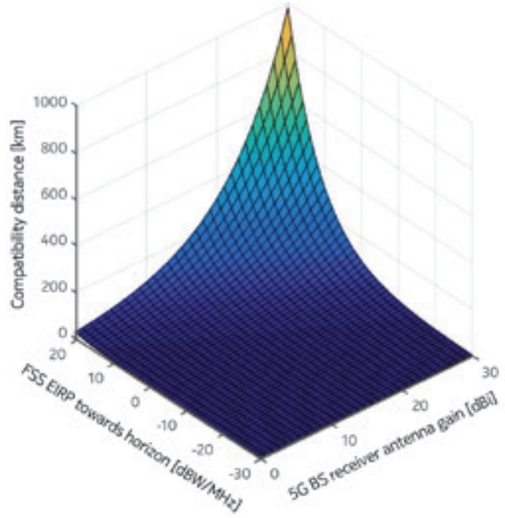
4.2. Co-existence between 5G transmitters and FSS space station receiver

Big question at the beginning of the study on 5G and FSS co-existence was whether aggregate interference from 5G network will cause noticeable increase of noise floor at FSS space station receivers. System level simulations have been conducted, where aggregate power per 5G cell sector was calculated [6]. It should be noted that simulations assumed realistic parameters for:

- 5G system layout,
- 5G BS antenna configuration, including antenna tilting and beamforming,
- 5G BS transmit power,
- path loss model,
- acceptable level of noise floor increase at FSS space station receiver,
- elevation angle towards FSS space station in reference to horizon,
- FSS space station orbit distance,
- FSS space station antenna gain.

Interference generated by UEs is significantly lower than interference generated by BSs and was not taken into account in these simulations.

Figure 3 Compatibility distance between FSS ground station and 5G BS as a function of FSS EIRP towards horizon and 5G BS receiver antenna gain



Simulation results demonstrate that large numbers of simultaneously transmitting BSs can operate within the FSS space station receiving areas without raising interference problems. Therefore, limitations on 5G BS EIRP are not required to manage aggregate interference from 5G networks into existing FSS space station receivers, which are part of current FSS geostationary or non-geostationary operations.

5. Expectations in regards to 5G spectrum allocation before WRC-19

Among the wide range of frequency bands chosen during WRC-15 for 5G studies, particularly visible is lack of bands below 20 GHz, which are essential from the perspective of radio propagation conditions and coverage of large macro 5G cells. However, exclusion from the WRC-19 studies does not forbid a regional study by permission of local regulatory bodies, like 28 GHz being studied by FCC in United States.

Positive finalization of work on 28 GHz band in United States, as well as successful 5G trials in South Korea and Japan due to upcoming Olympic Games, could have significant impact on the global harmonization of this band for 5G.

Studies conducted on 5G spectrum, either for performance evaluation or sharing with other systems, will be very important input to WRC-19 and will shape the final list of bands available globally for IMT-2020.

References

[1] 3GPP TR 38.913 V0.3.0, "Study on Scenarios and Requirements for Next Generation Access Technologies"; Release 14
[2] ITU, "Radio Regulations", 2015
[3] F. S. Chaves, K. Bechta, "5G network deployment: Interplay of key elements in the challenging outdoor-to-indoor scenario", European Conference on Networks and Communications (EuCNC 2016), Jun. 2016
[4] Nokia, "Response to FCC NOTICE OF PROPOSED RULEMAKING (Use of Spectrum Bands Above 24 GHz For Mobile Radio Services) from October 22, 2015", Date of response publication: January 27, 2016
[5] AT&T Services Inc., Nokia, Samsung Electronics America, T-Mobile USA, Inc., Verizon, "Response to FCC NOTICE OF PROPOSED RULEMAKING (Use of Spectrum Bands Above 24 GHz For Mobile Radio Services) from October 22, 2015", Date of response publication: May 6, 2016
[6] AT&T Services Inc., Ericsson, Nokia, Samsung Electronics America, T-Mobile USA, Inc., Verizon, "Response to FCC NOTICE OF PROPOSED RULEMAKING (Use of Spectrum Bands Above 24 GHz For Mobile Radio Services) from October 22, 2015", Date of response publication: June 1, 2016

About the authors

I have received the B.Sc. and M.Sc. from the Federal University of Ceará, Brazil, and the Ph.D. from the University of Campinas, Brazil, and the École Normale Supérieure de Cachan, France, all in Electrical Engineering (Telecommunications). I have interest and experience in interference and radio resource management, modeling of wireless systems and networks, and spectrum management and policy. Currently I work at Nokia Bell Labs on radio spectrum evaluation, co-existence and sharing studies for 5G.

Fabiano Chaves
Senior Engineer, Radio Research
CIOO Bell Labs

I have graduated Electronics and Telecommunication from Military University of Technology in Warsaw, where after graduation I worked as junior assistant in the area of wireless communication and electromagnetic weapon.
Currently I work in Nokia Bell Labs department where I lead the team responsible for radio spectrum evaluation, co-existence and sharing studies for 5G. I have experience in radio channel modeling, radio spectrum regulation and 3GPP standardization.

Kamil Bechta
Senior Engineer, Radio Research
CIOO Bell Labs

Clouds Full of Data

Pawel Berestecki
Engineer, Customer Documentation
MN RMA

Edwin Wierszelis
Engineer, Software Development
A&A NM & SON

NOKIA

1. Introduction

Communication benefited from the advances in technology already in the beginning of its existence. It not only aimed at the quality of the offered service, but also on the increase of its capacity and functionality. For instance, compared to original invention, carbon microphone which was invented in 1870s helped to greatly increase both sensitivity and strength of the transmitted signals. Carrier telephony system from 1920s made possible the transmission of multiple calls on a single pair of wire. In turn, invented in 1970 digital networks and ISDN introduced a host of new services, which were not possible before.

Fiber optic gave us almost unlimited capacity for the transmission of signals. All these advances in technology enabled development of sophisticated wireless and mobile terminals which offer almost unlimited set of services. We call these devices smartphones, regardless of how much we are aware of the technology that "drives" them. The transformation of network terminals is clearly visible to the users, but still, the network does not stop evolving; network equipment becomes smaller, faster and more energy efficient. All the complex services and competitive environment demand for faster deployment of services; more complex ones need more volume of data to be kept in the network. Therefore, Nokia puts the network into a Cloud Environment.

2. Challenges of a backup in a Cloud Environment

Telco Cloud does not make the system simpler, but rather adds to it more complexity. More precisely, it usually requires some changes in the architecture of network elements and adds various abstraction layers to physical hardware. The benefit is that all configuration specific to network element – the number of processing nodes, network connections, and storage – is put into virtual domain, where it can be managed by automated processes usage. Therefore, the physical equipment becomes simpler and can be configured in a generic way (regardless of the network element types which will be run on it). Such type of the generic infrastructure is also easier to extend. On the one hand, backing up virtualized network elements poses some challenges, on the other hand, it gives specific benefits.

For one thing, Virtual Network Function (VNF) can consist of multiple Virtualized Network Function Components (VNFC), often located on multiple physical servers. It can be very difficult or even impossible to make consistent backup of such VNF without some cooperation from its side. From the other hand, we can utilize scale-out capability of network element to minimize the size of the backup, and make possible recovery on the equipment of various size. Moreover, intended backup should contain not only content of the VNF component, but also a configuration of the VNF (for example the layout of the system). The other benefit of virtualized infrastructure is that spare cloud for recovered system can be assembled from different hardware than the original one, where backup has been performed. The difference is not visible for the VNF. The backup function must support various recovery scenarios, including reduced hardware or hardware different than the original. As recovery on scaled-down infrastructure is natural for cloud environment, the concept supports most challenging cases of disaster recovery.

There are more benefits of Cloud Environments such as: different storage options, utilization of the server's spare capacity, or replicated object storage which is highly scalable.

3. Evolution or revolution? Core NOKIA NEBR from the Cloud perspective

BR Cloud is a successor of Network Element Backup and Restore (NEBR) product from core NOKIA offer. As a successor, it must provide continuity to existing products, installations and methods, despite significant change of the internal architecture. The solution must both provide new functionality and protect Customer investment by reusing existing systems where possible.

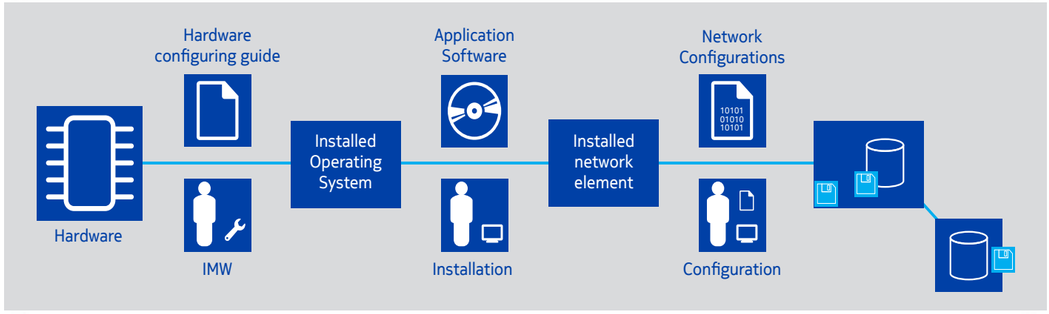
The product must serve for both architectures: physical ("Bare Metal") and virtual network elements. As a backend storage, both scenarios must be supported as well. Small operators may find that backup storage implemented by cloud itself is enough. More complex scenarios will require external device to store data. In such case, Cloud resources are utilized to perform data preparation tasks (like compression or deduplication), while actual data is transferred to some external device. External storage helps to define physical place for data – a feature required by factors both from the Telco domain (disaster recovery) and outside of it, for instance by law requirements.

In such implementation we aim at a modular design, with separate components using common platform to cooperate between storage backend and frontend Agents, which collect data from network elements. Features specific to Telco Cloud are: interface to Telco Cloud Management and Orchestration (MANO) entity and functions to control load of shared Cloud resources utilized by backup system.

4. Architecture of virtual future

Below is an example of typical network element installation for Bare Metal:

Figure 1 Network element installation for Bare Metal



The elements marked with diskette image need to be backed up; the object of the backup is network element (OS, software and configuration) with the processed data. Deployment of virtual network element in the Cloud environment looks different:

Figure 2 Network element installation for Cloud

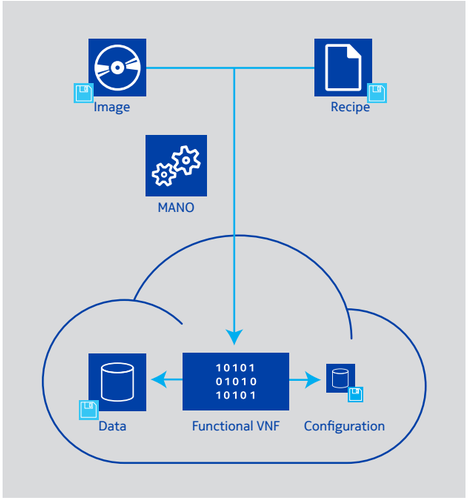


Figure 2 does not contain hardware-related parts. Although the hardware exists, it is configured with the generic process and can be delivered as a universal product, regardless of function it is going to perform.

Table 1 Differences between Bare Metal and Cloud

Property	Bare Metal	Cloud
Hardware configuration	Depends on application	General-purpose
Installation manual	Paper	XML
Installation process	Manual	Automated
Installation	Static	Changes size and location during normal work
Number of components	Fixed	Variable
OS configuration	Persistent	Ephemeral
In case of error	Fix it	Kill and respawn
Installation	Static	Dynamic (can change size and location)
Extensions	Special procedure	Built-in
Upgrades	Special procedure often involving additional hardware to provide continued service	By image change
Backup	System	Function

4.1. Backup function

In a traditional Bare Metal systems the configuration is bound tightly with the Operating System instance. The installation process starts from the Hardware installation, after which the Operating System and suitable application software is installed. System needs complex configuration before it will be able to perform intended function. The changes are distributed over various parts of the system, the optimal way is to back up the entire system in installed and configured state.

For Virtual NF the installation element is decoupled from hardware. Software for network element is delivered via disk (VM) image, which contains complete installation of Operating System and all software required for the network element. Such image is product-specific, and does not contain elements specific for a particular installation. When the Virtual Network Function is started, following elements take place:

- required number of Virtual Machines is created and loaded with suitable Images from special storage that reside on Cloud;
- VMs are preconfigured;
- machines are connected using Virtual Networks, in a layout specific to VNF;
- additional storage and other objects are connected to VMs;
- VMs are started. Part of the VNF configuration is done by VM themselves during their startup.

All this is performed by MANO, which interprets so called Recipe for the VNF deployment. The process is fully automated, so the installation of the VNF is a fast in comparison to its Bare Metal counterpart. Still, it should be mentioned that cloud (virtual) implementation provides benefits for system restore operation. True Cloud application requires the code to be managed separately from the data such application processes. The number of processing elements can vary depending on the needs. Hence, the number of objects for backup is limited and well defined: VNF Recipe, images, VNF configuration and data. The first two can be considered mostly static, VNF configuration does not change often, only data needs to be backed up frequently. This separation makes it easier to construct optimal backup schedule of VNF elements, depending on how often they change. Thus, the choice during the construction of backup schedule for elements of various kinds is obvious.

Virtual Machines (VMs) in Cloud application are considered ephemeral (temporary). In true Cloud implementation there is no need to back them up. Restore rather recreates system function by going through steps used to initialize it (Recipe + disk images), and provides configuration + data from backup. As the size of restored system does not need to match the original, it is possible to recreate system on reduced, spare cloud. Scale-out mechanisms are built into VNF can be used later to extend such VNF back to original size.

Decoupling code from the configuration usually requires some changes in the way the software for network element is prepared. However, the backup of virtualized system is smaller and more elegant, as it consists of fixed Recipe, images and a relatively small amount of configuration data for virtual network function. Separation of data and processing gives new options for the ways on how data is managed.

4.2. Persistent Storage

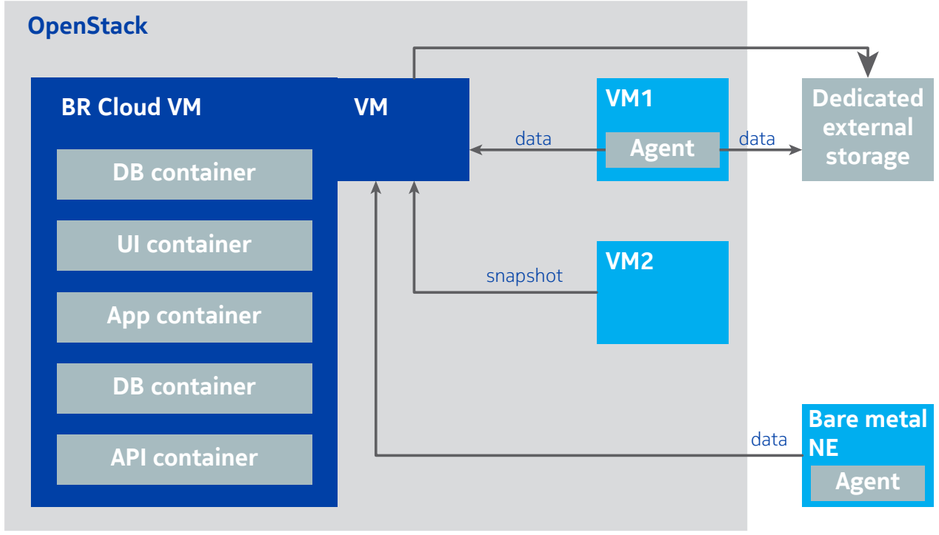
All systems which process information require some kind of a persistent storage. It is a subject of backup and restore. Cloud environment offers several kinds of such storage. The following types of the storage can be distinguished due to different properties of a persistent storage:

- **block storage** is storage mechanism similar to disk. It can contain a filesystem or any other random accessible data. Block storage is provided as an additional disk to virtual machine, which operates on data contained in it. Contrary to ephemeral disk storage used for OS installation, persistent Block Storage is not lost when virtual machine is terminated. It can be rather attached to other virtual machine for further processing. Without special mechanisms block storage cannot be used by two or more Virtual Machines at the same time. Backing up this kind of storage can consist of taking snapshots, and later transferring them to some form of a storage. While the snapshot capability is provided by Cloud infrastructure, some cooperation from the side of virtual network function that uses a Block Storage may be required to guarantee consistency of such backup.
- **object storage** is a form of secondary storage, where files cannot be directly opened and accessed, but they can be downloaded (retrieved) for use and upload after modification. Object storage can be accessed by multiple parties simultaneously. Although object storage is not suitable to be used as a storage to some applications which require continued access to a file on disk, it is a perfect backend storage for various backup scenarios: highly scalable and provides by nature redundancy over geographically separate nodes. Backup of Object Storage contents is easy, as backup system is just another client who uses such storage. VNF may be required to provide some information on which part of Object Storage to back up.
- **shared file storage** combines benefits of block and object storage: files can be opened and used simultaneously by multiple nodes. However, there is a tradeoff between system performance and consistency of data contained in shared file storage. Cloud applications generally should avoid the use of Shared Storage and prefer so called “shared nothing” model instead. Backup of shared storage contents depends on the application. In order to achieve consistent system state for backup, the simplest backup cases may require closing of all files used by virtual machines. As it is not acceptable from service availability POV, architectural changes may be needed to provide both availability and consistency.

- **database as a service** is deployed typically as a secondary service, which uses other mechanisms listed in this section (block or file storage). Database can be backed up using methods specific for backend services, however, depending on database type, other and usually more transparent methods also available.
- **log storage** is not a standard Cloud service, but it can be easily implemented with any combination of mechanisms listed above. Certainly log storage is a subject of the backup, but restore operation is less important than capability to archive logs and analyze them, if needed.

Persistent storage mechanisms listed above may be provided by specialized hardware (like storage arrays) or served by the cloud from resources provided to it as a base. In some cases, a new service is constructed out of the existing ones. For example, Shared File storage can be assembled by specially configured Virtual Machine, using Block Storage as a storage backend. In our Telco Cloud Backup solution we utilize mixed approach: the backup is prepared using natural Cloud resources (deduplication, packaging, management), and then it is transferred to physically separate storage unit or kept in the cloud, using reliable Object Storage.

Figure 3 BR Cloud architecture



4.3. In reality

While the concept of Telco Cloud is revolutionary, putting such idea into reality cannot be done immediately. The reasons which force us to do such a "gradual revolution" come from three directions:

1. Some network element functions require significant changes in the architecture in order to be suitable for Cloud.
2. VNF Elements must cooperate with elements implemented in a traditional architecture; some network protocols which are in use are not "cloud friendly".
3. The Cloud environment, namely, Cloud infrastructure software still evolves and therefore other variables are introduced into application transformation process.

The factors above force to use transitional solutions which affect the backup solution. Firstly, it is a very tempting "shortcut" to use persistent VM, namely VM started from normal image with persistent storage backend as a disk. Although such method is easy and allows to do system administration in "business as usual" manner, this is actually not Telco Cloud solution (we lose scale-out capability, increase backup size and reduce restore options). Such approach should be considered as a last resort, and accepted only for small, mostly static systems.

Some NOKIA products, such as Nokia SDL, explore limits of performance of the underlying Cloud Infrastructure. System like this has to be designed to utilize full potential of distributed processing, ensure database consistency and provide sufficient network bandwidth margin; to allow transfer of backups by the usage of a common network infrastructure, without any additional delays. For such system the process of data preparation for a backup must be specially crafted.

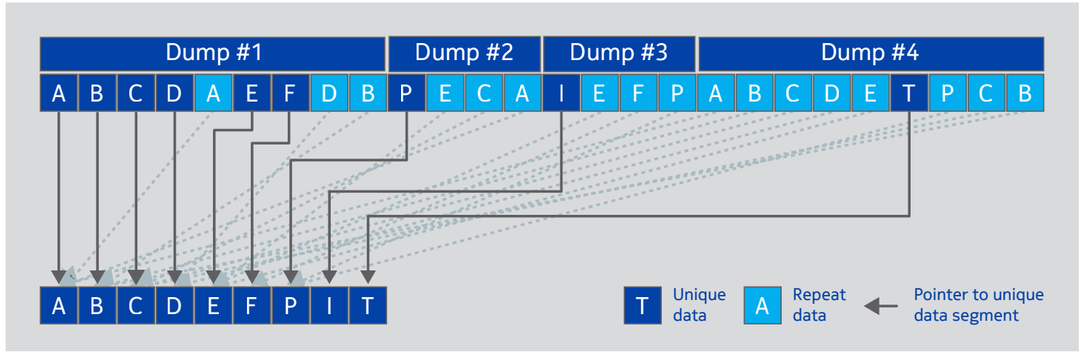
In general, a descriptor (Recipe) for any VNF should contain a list of storage elements contained within that VNF. Such list is a base for backup (additional information is required to define frequency of backup operations). Data stored in dedicated external storage with installation Recipe can be applied to system after its initialization, resulting in restored function.

4.4. Utilizing Cloud Potential for a Backup

NOKIA BR Cloud utilizes the same infrastructure as the Nokia Telco Cloud itself.

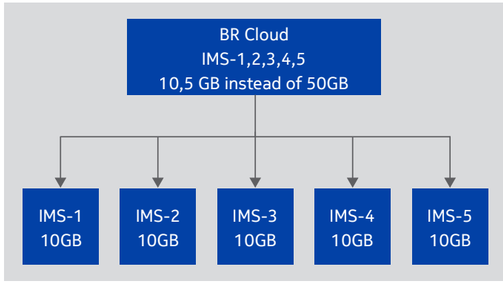
Cloud version utilizes available processing power to provide an efficient data deduplication. It is a sort of data compression process which eliminates duplicate copies of the same data. Deduplication also reduces both network traffic and disk storage requirements. This process looks as below:

Figure 4 Data deduplication



Eliminating redundant data can improve the efficiency of the backup and restore processes. Primary version of the storage has become cheaper over time, and typically many versions of the same information are stored.

Figure 5 Storage needed for deduplication



5. Safety and flexibility: it must be Cloud

NOKIA observes many reasons for migration of network elements into Cloud. With Cloud it is possible to provide universal, easy and expandable platform for deployment of network functions. The first observable benefit for a Customer is an instant installation of network elements, since they are delivered as software. We must stress

that the term “software” is not limited to the code which performs required network function. Delivery (namely backup or installation package) also contains a recipe on how to deploy network element in the virtual domain. This replaces complex and elaborate Hardware Installation Guide, which is specific to every Bare Metal product and requires different set of equipment, depending on the planned function. Hence, the “recovery” term is not only a restoration of existing code, configuration and data, but also virtually installation of all infrastructure as well. Because of that, recovery is faster, even in case of the worst situations when, for instance, original service infrastructure is destroyed.

The capability of network elements to scale-in and scale-out is a part of the architectural changes which are required to achieve the mentioned goals. Hence, the function is split into smaller tasks which are distributed to many processing nodes. The number nodes can dynamically change depending on actual needs. We can restore system in minimal configuration, which reduces recovery time as less data needs to be transferred. Scale-out capability, routinely used during normal VNF operation, can now be reused to extend system back to the original size. This unique capability was not possible before Telco went into Cloud.

Cloud technology provides foundation for the concept of Self-organizing networks, where services are dynamically created, terminated or moved in a way that provides the best utilization of available resources. It is also a perfect way to start small and grow according to the customer needs. Nokia believes that offered solution an important element in the picture of the Telco Network future.

About the authors

Pawel Berestecki
Engineer, Customer Documentation
MN RMA

I settled in Wroclaw a couple of years ago due to law studies at University of Wroclaw. I am a Customer Documentation Engineer and my work in Nokia is devoted to Backup and Restore products and the NetAct domain.
The area of interest which I am mainly focused on lies in the developments in technology. Not surprisingly, my work domain enables me to fully understand issues connected with technology development.

Edwin Wierszelis
Engineer, Software Development
A&A NM & SON

Unix-like system administrator and programmer with 20+ years of experience in Telco domain. As a teenager, he designed and constructed his first ZX-Spectrum compatible computer to play games and learn programming.
Wants IT to be useful, which makes him think both small and big, starting from embedded systems to cloud computing. This certainly does not cover all topics you can talk with him about. With Nokia since 2008.

Advanced Telecommunication Technologies

Internet of Things (IoT) – Small Things, Big Opportunities

Piotr Grzybowski
Senior Specialist, Network Engineering
MN Customer Support

NOKIA

1. Introduction

Internet of Things (IoT) becomes a buzzword that attracts attention lately. Visions of self-driving cars and refrigerators automatically ordering beer from the shop looks exciting, and they are getting closer step-by-step. When 3G deployment started almost 20 years ago, nobody expected a mobile internet explosion. An analyst at the time stated that video calls would be a killer application for 3G. Do you know anyone who uses video calls frequently? Instead of that, Web 2.0 and social networking were the real boosters of the whole industry. With this background we are better prepared for IoT challenges.

Are you experiencing lags seeing online transmission of football matches on your smartphone? Try to imagine what will happen if instead of few mobile phones per household, you will get more than 40 additional IoT connected devices. Forecasts say that we can expect more than 30 billion connected devices in 2025 [Machina Research, 2015]. These visons may be as well accurate as 3G forecasts, but the fact is that we should expect an IoT services explosion.

This is the technology challenge to serve a massive number of IoT users and still reserve some place for ordinary human-being connections. NOKIA is already taking place in that race; we are defining standards for a new IoT radio technology within 3GPP and extending our system to provide complete IoT-optimized end-to-end solution.

2. M2M and IoT technology

Internet of Things origin comes from the Machine to Machine (M2M) communication. M2M allows for autonomous data exchange among devices without human interaction. Communication is within a closed group of devices, usually related to a certain business domain and purpose. Proprietary protocols are used for communication and usually there is no interaction between devices of different domains.

How to migrate from M2M into IoT domain? Simply speaking, it is enough to enable cloud connectivity for sensors, machines, people and other devices, enable them to exchange information, add intelligent applications and services, and we get full flavor of IoT. Comparing to M2M, the center of gravity for IoT was changed from sensors, into data, applications and analytics. This is the biggest advantage of IoT – having certain set of inputs we can use in various ways, depending only on the services we can figure out and correlations that can be derived. It opens new business opportunities for IoT, also because that technology is much closely aligned with IT solutions than usual mobile markets. One simple example is weather data correlated with Smart meter readings for planning power plant electricity and heat production. When cold weather is forecasted and power and heat consumption is already high, we have to start with new sources of energy.

3. IoT applications variety

Various IoT applications act in quite different ways, what brings significantly diverse requirements for the network. A good example is the Smart meter, which tolerates high latency. High throughput is not a focus here, since the device sends small packets infrequently. However, it should operate correctly in a deep basement, hence signal strength has to be sufficient enough to overcome deep indoor losses. Moreover, power consumption has to be low to allow for long battery operation, to reduce maintenance costs. Opposite requirements are in place for surveillance cameras that deliver a continuous stream of high quality throughput-hungry video, but usually are located in good radio conditions and with remote power sources.

Table 1 IoT Use cases

Use case	Required mobility	Required coverage	Typical access pattern & data volume	Battery saving
Smart meter	Stationary	Improved	1/day, small	Important
Outdoors thermometer	Stationary	Standard	1/min, small	Important
Pet tracer	Mobile	Improved	1/min, small	Important
Surveillance camera	Stationary	Standard	Continuous, big	Irrelevant
Fleet tracking	Mobile	Standard	Random, big	Irrelevant
Connected car	Mobile	Improved	Continuous, big	Irrelevant

Addressing all these partially contradicting requirements with legacy technologies is not easy. Even if nowadays, billions of M2M connections are served by 2G or 3G technology, they use network resources in a suboptimal way. Legacy networks are designed to carry voice or mobile broadband data services rather than to accommodate a massive number of devices sending small messages. That is why the development of IoT-optimized radio technologies has started within 3GPP and other regulatory bodies.

4. IoT Objectives

Current development of legacy cellular modems goes into the direction of high speed mobile broadband to allow for high resolution video and entertainment. IoT-driven modem target is just to convey a small message, hence there is a big potential of cost savings by simplifying chipset hardware. Cost reduction is the ultimate goal for massive IoT deployment; some of the devices might be even disposable.

In contrast to the legacy smartphones charged just from the power outlet, IoT sensors can be located in remote locations with difficult access and lack of energy sources. It is not very likely that someone will exchange batteries for them every week. Hence, the target is to reduce maintenance costs and provide means for guaranteed long operation on batteries. It is easy to imagine what happens in case a smoke detector runs out of power.

Difficult access location is usually related to the challenging constructions with high propagation loss. A legacy subscriber in the coverage hole will look for a better location, a Smart meter wouldn't change its position. It is so important to develop techniques improving coverage and devices with high sensitivity.

Targets for IoT-optimized technologies of massive deployment have been specified by 3GPP [1]. IoT radio should comply with the following mandatory objectives:

- Low throughput of the bi-directional data rate of 160 bps,
- Low power consumption – 10 years of operation on batteries for IoT radio modem,
- Support for massive number of devices – up to 40 devices per household,
- Reduced mobility and inter-RAT support.

Those dreaming of self-driving cars should consider the fact that module costs and battery life targets make the first wave of IoT technologies useless for that application. The delivery of a single message takes several seconds for a cost-optimized IoT modem, mostly due to its low processing power. During bad radio conditions, repetitions of the message are needed, hence transmission time increases significantly to the order of minutes. Such values are insufficient for connected cars application, since a reaction time of tens of milliseconds is expected.

First wave of cellular IoT technologies will support low demanding, low throughput, latency-tolerant applications. Smart metering could serve as a good example here. For Smart meter ,only the fact that message is delivered matters, and latency is not an ultimate goal. Note that latency is a direct consequence of improved coverage requirements (message repetition), 10 years battery life (infrequent communication) as well as modem complexity reduction (low costs) and small amount of transmitted data (low throughput).

Targets of 40 devices per household can be easily transferred into the cell capacity. Typical London household density is compared against urban sector coverage area. From the calculations, the capacity target of about 50000 devices per cell is obtained.

For IoT service planning, it is crucial to understand that final performance is a tradeoff of coverage, capacity and power consumption.

5. Traffic model

Human communication has a random nature. Service can be initiated at any place and time. Network planning should consider subscriber mobility and traffic demand. Traffic profiles are easy to estimate – either voice connection, smartphone profile or web browsing.

IoT profiles are easier at first glance – some devices are at fixed locations, most of them generate mobile-originated traffic infrequently. Nevertheless, service variety is much more diverse, and with IoT explosion, it is even difficult to figure out all the options. That is why for correct network planning, the expected user behavior has to be carefully considered in the traffic model.

Table 2 Traffic model

	Inter-arrival time	Packet size (bits)	
Mobile originated Periodic reporting	Distribution: 1 day (40%), 2 hours (40%), 1 hour (15%), and 30 minutes (5%)	UL: 20–200	Smart metering scenario
Network originated reports	Distribution: 1 day (40%), 2 hours (40%), 1 hour (15%), and 30 minutes (5%)	DL: 20 UL: 20–200	UL response required in 50% of messages
Mobile originated Exception report	Months to years	UL: 20	Up to 10 second latency
Software updates	180 days	DL: 200–2000	

Typical traffic model types were specified by 3GPP [1]. They are not concentrating on application and services as such, but rather on the most probable message exchange profile.

The first group will use mobile-originated periodic UL traffic. This kind of traffic is common for cellular IoT applications, such as smart utility (gas/water/electric) metering reports, smart agriculture, smart environment, etc.

Another case is with network-originated reports. The network triggers the device to send an uplink report for example, a Smart meter reading. For some cases, uplink response is not needed, such as a command to switch on the lights.

Exception report models application, which monitor physical condition and trigger an action when an event is detected, such as smoke alarm detectors, power failure notifications from Smart meters, tamper notifications, etc. Such events are expected to be generally rare, however it is required that reports are delivered in near real time.

Purely downlink-related events are remote software updates of modem firmware that may occur several times a year.

6. Capacity planning

So far IoT was discussed on the network level; now is the time to see how it impacts single eNB (eNodeB, LTE base station) cell. We have to estimate cell capacity to guarantee that each IoT sensor will be served in reasonable time.

How does the capacity planning look like for legacy LTE services?

We have some degrees of freedom; firstly, apply capacity-boosting features, the next step is to upgrade frequency resources, either use a broader bandwidth option or add multiple carrier aggregation.

For IoT there are not so many capacity boosters, there is only one bandwidth option and a lack of aggregation opportunity.

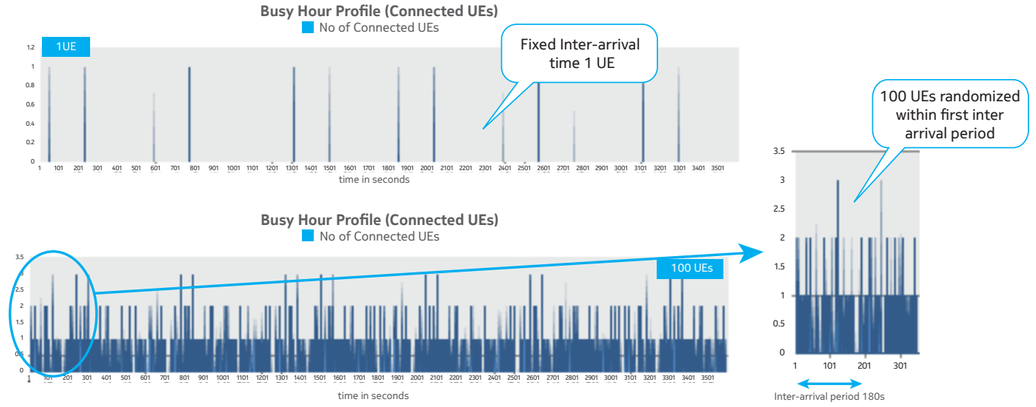
IoT network capacity is strictly related to the definition of the service traffic profile. On the contrary to the legacy services, capacity is expressed in terms of maximum number of subscribers/sensors to be served. Throughput is not a key discriminator, since user payload is usually very low.

IoT radio specifics, especially narrow band, yields restrictions in terms of available resources. It brings slightly different methods and limits for capacity estimation. For legacy mobile broadband services, the number of simultaneously connected users is not seen as a capacity blocker, however for LTE-M (LTE for Machine-Type Communication) or NB-IoT (Narrow Band Internet of Things), the limit is substantially lower, as in 50 connected users. Comparing it against the expectation of a massive number of served devices per cell, it becomes a challenge.

What is the relation between simultaneously connected and served users?

Each IoT sensor follows its connectivity profile, according to the traffic model. Taking regular reporting as an example; sensor reports are sent periodically, it establishes a connection which con-

Figure 1 Regular traffic



veys the message (being in the connected mode) and then goes into idle mode until next transmission. Sensor with its traffic profile is regarded as a served user, and utilizes network resources mostly during active transmission while in the connected mode. Generally, the time spent for transmission is proportional to the achievable throughput and the message size. What will happen if thousands of such served users will appear in our cell?

The whole capacity process is modelled as a stream of independent traffic flows of different sensors, generating cell load according to the predefined traffic patterns. Served users, which are transmitting in the same time can be referred to as simultaneously connected users.

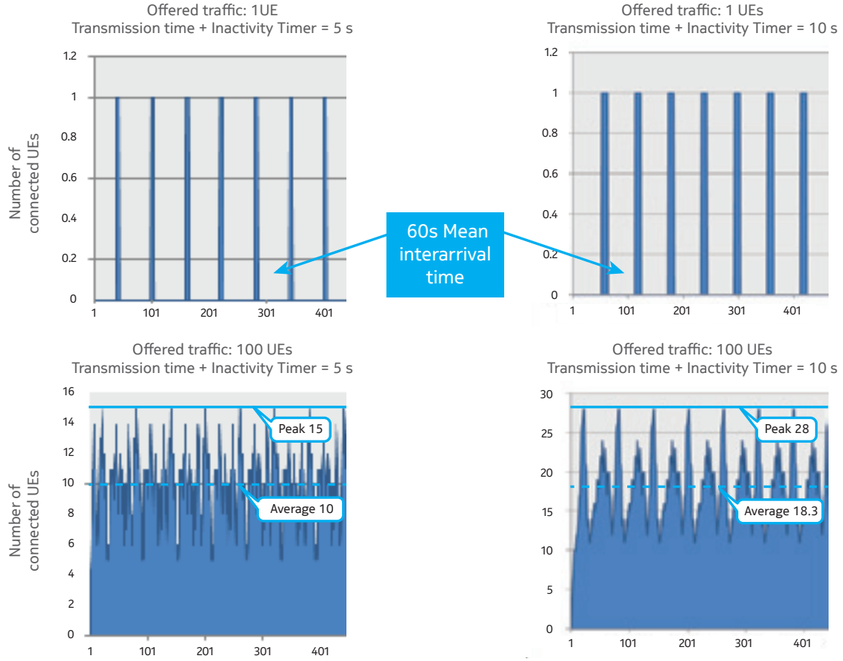
Served users capacity is a figure coming directly from the simultaneously connected limit and time spent for transmission.

In **Figure 1**, periodic transmission with 60 s interval is visualized. With single service and fixed inter-arrival time for all the streams, transmission start points have to be randomized, to avoid risk that all the UEs would start in the same time, blocking network resources. Please note that in our example 100 UEs generate only 3 simultaneously connected users at maximum, providing that transmission takes 1s.

The whole story becomes more complicated when realized that service establishment and single message transmission time for narrowband technologies is much longer than for legacy LTE. The point is that we have much less resources for transmission, and in bad radio conditions they are additionally consumed by several repetitions of the same message.

Normally transmission can take up to several seconds, at worse radio conditions, when tens or hundreds of repetitions are needed, it may take minutes for the whole process to complete. On top of transmission time there is also the inactivity timer; it holds a connection active after packet is conveyed, then after expiry connection is released and mobile goes to the idle state. Please

Figure 2 Connected UEs relations



note that the inactivity timer for legacy services allows for saving signalization resources, as the whole process of connection establishment can be avoided when another request comes. In case of regular reporting with the exact known next transmission occurrence, the inactivity timer might be reduced to minimum. Another point is that UE can quicker go to the idle and power savings mode.

As can be seen in **Figure 2** for the same offered traffic of 100 UEs, with total time of connected state (transmission and inactivity timer of 5 s we can reach 15 connected UEs at maximum, with average of 10, while for 10s it is increasing to 28 UEs peak and almost doubled to 18 on average.

Yet another limitation to keep in mind is the eNB processing power. The eNB baseband configuration has to be powerful enough to process message avalanches coming from a massive number of IoT devices.

From the traffic model and number of UEs, one can easily calculate the number of exchanged system messages per second. If estimated eNB processors load will exceed limits for basic baseband configuration, an upgrade with extension cards has to be considered. Processing power limit is valid mostly for the older releases of the hardware, while the latest state-of-the art eNB baseband is able to cope with a massive number of IoT sensors.

For sensors in bad radio conditions, we need to deploy coverage extensions techniques, such as repetitions and the number of exchanged messages will increase significantly. It is easy to imagine how it affects the overall capacity figures. Generic capacity planning for this case should consider coverage distribution across the cell area, and then it can be transferred into the number of repetitions needed.

7. Practical experience with IoT

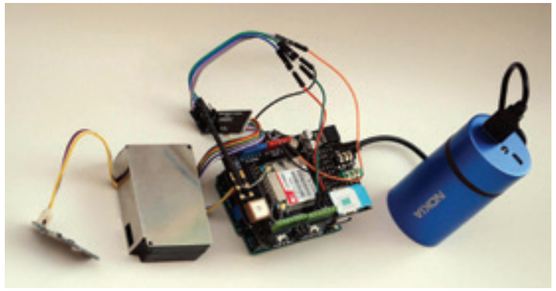
IoT-specific means that it broadly covers aspects from IT and cellular technologies. Apart from the ordinary radio and core network design, there are also sensors, APIs, analytics and end user applications. All the elements were envisaged in our small-scale IoT practical exercise that covers proof of concept for cellular IoT sensor.

The project was led by MN Customer Support Big Data and Network Engineering and organized as IoT Hackathon event.

Firstly, ideas for IoT sensor were gathered, and among many of them, sensor for evaluation of air quality was selected. It's becoming a hot topic lately, as Wroclaw was ranked as one of the most polluted cities and the one with the slowest traffic speed across Europe.

The aim of the project was to prepare R&D prototype of air pollution monitoring system using remote cellular sensors. Usually during the

Figure 3 IoT sensor



Hackathon there are few days of extensive coding, now we need to prepare also sensor hardware platform before the actual coding.

Firstly the engine; Raspberry Pi option seems too heavy for simple reporting, moreover battery power consumption is the issue and that is why Arduino UNO platform was selected.

The next question is, what communication platform should be used? IoT-optimized technologies look very ambitious, however there are no modems on the market yet. 3G is so power hungry, LTE has no sufficient coverage; hence the only option was GSM with GPRS data support. To allow for sensor tracking, the modem was equipped with GPS.

Sensor selection was quite straightforward, there is only one particulate matters (PM, measuring number of particulate matters in the air) sensor available on the market, which reports particles of different size. In addition, sensors for temperature and humidity were also included in the set.

First trials with sensor prototype show which issues have to be addressed during hackathon; actually they are the same as for 'big' IoT – battery savings and mobile data usage have to be addressed.

The hackathon was split into two parts, one focused on the optimization of the sensor code to reduce power consumption and data usage as well as data preprocessing, another part dealt with the visualization layer and mobile application that can be used for data analytics.

The sensor optimization part solves data savings by collecting the messages, and sending them in the one compressed package. Power savings consider switching on the modem and PM sensor only during transmission. Moreover, protocols for data retransmission were also developed.

The visualization layer uses freely available IoT environment, which shows sensor readings. Even with such a simple set of sensors, interesting observations can be derived (**Figure 4**).

The sensor was left in the office overnight. It is easy to see that cleaning service starts at 21:00, there is increasing PM1 and PM10 levels. In the morning at 7:00, the heater has started, and then at 7:30 someone opens the window and it is getting colder. What is interesting – opening the window during rush hour might not be a good idea as it seems, note that particulate matters is constantly rising.

Mobile application developed during the hackathon shows in graphic way, with usage of the heatmap, pollutions levels, humidity and temperature. Further steps of project development assumes the collecting of air quality data with the sensor on a bike, and that is why bike paths can be also visualized.

Even our small project proves that IoT is within your reach and it is easy to develop incredible solutions during a few days of the hackathon run. Moreover, the problems we are facing in a small scale are valid globally and can be solved in a similar way.

8. Conclusions

The biggest advantage of IoT is that through the massive number of various sensors, precious data can be stored and analyzed. It helps to reach the biggest benefit of IoT – services and applications that can be derived.

IoT brings not only a massive numbers of sensors, but also services variety. In order not to be lost with message avalanches, an appropriate network design has to be applied through the traffic model and capacity planning. Restrictions coming from the IoT objectives have to be carefully examined here.

With our smallscale Hackathon exercise, we prove that IoT is technology which can be easily started and trialed.

IoT becomes a big opportunity for service providers and network infrastructure vendors. The technology challenge to serve a massive number of IoT sensors has to be smartly solved to generate revenues for all the members of that race. NOKIA is proactively setting standards and developing new features for IoT network optimization.

Figure 4 Sensor visualization results

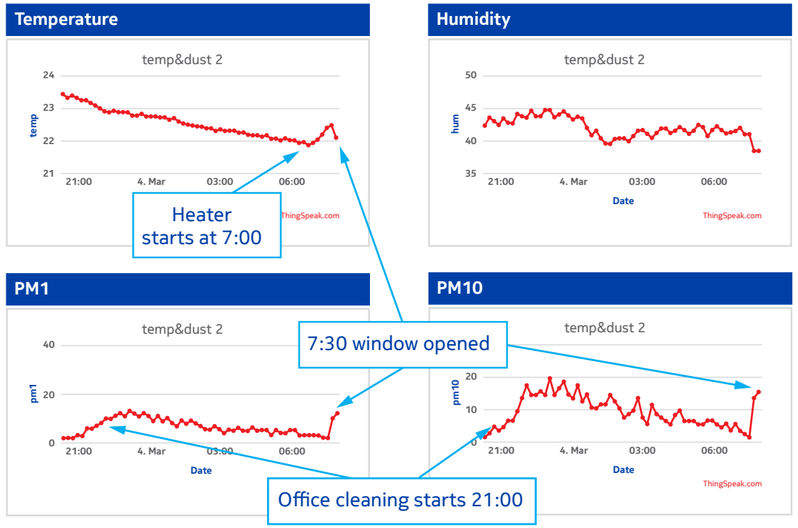
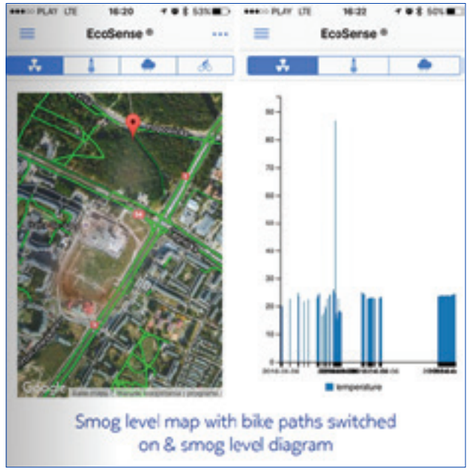
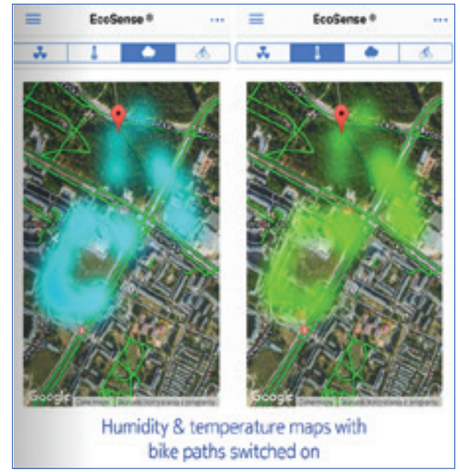
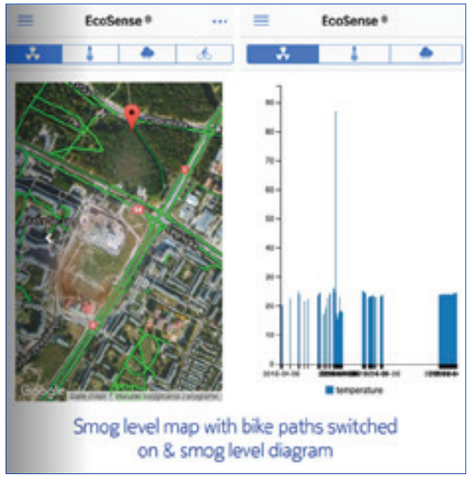


Figure 5 IoT Mobile application







References

[1] 3GPP, "TR 45.820 Cellular system support for ultra-low complexity and low throughput Internet of Things (CIoT) (Release 13)".

About the author

Graduate of Warsaw University of Technology in the faculty of Electronics, Information and Communications technology. Years of experience with mobile radio, from 2G network deployment until LTE features analysis. Currently working in Network Engineering department, dealing with site solutions, radio frequency aspects and Internet of Things areas.

Piotr Grzybowski
Senior Specialist, Network Engineering
MN Customer Support

Telecommunication System Engineering

NOKIA

Internet of Things (IoT): the New Age of Internet

Grzegorz Olender
R&D Manager
MN LTE

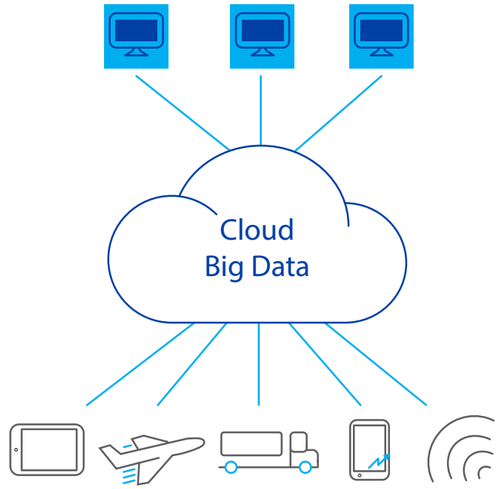
NOKIA

1. Introduction

A vision of everything connected to the Internet excites many people around the world. They hope for innovative applications to improve living conditions and expect to benefit from new amazing business opportunities at the same time. For this reason the so-called **Internet of Things (IoT)** is seen as the next revolution in the mobile communications sector. It is going to change the future of Internet and telecommunication systems and will have a significant impact on our lives.

The Internet of Things is a network of physical objects ("Things") like devices, vehicles, or buildings which autonomously collect and exchange data with big data analytics applications in order to gain insight into the process or to provide additional services with respect to collected data. Benefits and synergy effects go far beyond simple **Machine-To-Machine (M2M)** communications systems deployed so far.

Figure 1 Internet of Things (IoT)



There are predictions of over 20 billion IoT units in 2020 [Gartner, 2015] and over 30 billion IoT units by 2025 [Machina Research, 2015], 23 billion of which are fixed network and short range units and 7 billion are **Low-Power Wide-Area (LPWA)** and **Cellular IoT (CIoT)** units [Machina Research, 2015].

Experts also claim that the IoT market has a total potential economic impact of 3.9 trillion USD up to 11.1 trillion USD annually by 2025 [McKinsey, 2015].

This tremendous business opportunity makes IoT an important IT trend and a driving force in many potential application areas like:

- Agriculture
- Automotive
- Digital health
- Environment
- Public safety
- Smart grid
- Smart metering
- Smart factories
- Smart cities
- Smart home
- Tracking
- Wearables

There are many more business opportunities, yet to be discovered. This article outlines requirements for IoT connectivity system and gives a brief overview of leading IoT connectivity technologies with LTE-based solutions for IoT purposes.

2. Requirements for IoT Connectivity System

IoT connectivity system has to fulfill the following requirements:

- traffic model, that is small and low frequent data messages from/to a large number of sensors
- energy efficiency, that is low power consumption and long-life battery of a device
- enhanced coverage, that is coverage guaranteed for deep indoor and rural areas
- economic viability, that is low operational costs of a device

In particular, IoT connectivity system must take the following objectives into account:

- large number of devices (up to 40 devices per household)
- small amount of traffic per device (packages of few hundreds of bytes)
- low power consumption (minimum 10 years of battery lifetime)
- extended coverage (10-20 dB extra link budget over legacy standard)
- low device costs (less than 5 USD)
- low deployment and maintenance costs (including initial CAPEX and annual OPEX)

Considering all the mentioned requirements and objectives, the 3GPP standardization organization has specified the following performance objectives for IoT connectivity system [1]:

- Limited throughput: data rate of at least **160 bps** for uplink and downlink
- Improved coverage: extended coverage[1] with **20 dB** additional link budget over legacy standard[2]
- Massive number of devices: up to **40 devices** per household
- Energy efficiency: up to **10 years** battery lifetime with **5 Wh** battery capacity (two of-the-shelf AA or AAA batteries) even in locations with adverse coverage conditions where extended coverage is needed
- Network latency: relaxed delay characteristics may be supported, however certain applications (for example alarms) may require at last **10 seconds** of delay[3] for the uplink
- Reduced complexity: low-cost devices are allowed to have limited throughput requirements and may not support circuit switched services

3. Overview of IoT Connectivity Technologies

There are many radio technologies targeted to IoT connectivity system and IoT applications. They are summarized in the **Table 1**, **Table 2** and **Table 3**.

Table 1 Overview of IoT radio technologies

	Licensed spectrum	Unlicensed spectrum
Short range	–	Bluetooth Smart (Bluetooth Low Energy) Wi-Fi (IEEE 802.11ah) RFID ZigBee Z-wave
Long range	Enhanced Coverage GSM (EC-GSM)[4] LTE for Machine-Type Communication (LTE for MTC, LTE-M)[5] Narrow Band Internet of Things (NB-IoT)[6] 5G Massive MTC and Reliable Low Latency Communication[7]	SIGFOX LoRa (IEEE 802.15.4g) Weightless OnRamp Wireless

1 Measured by Maximum Coupling Loss (MCL) [1]
2 MCL equal to 144,0 dB is assumed for legacy GPRS (Non EGPRS) [1]
3 Measured from the application 'trigger event' to the packet being ready for transmission from the base station towards the core network [1]
4 In scope of 3GPP GERAN (Rel-13)
5 In scope of 3GPP RAN (Rel-12/13)
6 In scope of 3GPP RAN (Rel-13)
7 In scope of 3GPP RAN (Rel-15/16)

Table 2 Selected IoT radio technologies for unlicensed spectrum

Technology	Bluetooth Smart	Wi-Fi, IEEE 802.11ah	LoRa, IEEE 802.15.4g	SIGFOX
Range	<100m	<1km	<11 km	<13 km
Link budget	n/a	n/a	157 dB	160 dB
Spectrum	2.4 GHz	900 MHz	900 MHz	900 MHz
Bandwidth	2 MHz	1,2,4,8 MHz	<500 KHz	100 kHz
Battery lifetime	>10 years	>10 years	>10 years	>10 years
Data rates	<100 Mbps	<100 Mbps	<50 kbps	<0.1 kbps
Chipset costs	0.5 $	0.5 $	1.5 $	1.5 $

Table 3 Selected IoT radio technologies for licensed spectrum

Technology	EC-GSM	LTE-M	NB-IoT	5G
Range	<15km	<15km	<15km	<15km
Link budget	164 dB	159 dB	164 dB	164 dB
Spectrum	800-900 MHz	700-900 MHz	700-900 MHz	open
Bandwidth	200kHz	1.4MHz	200kHz	open
Battery lifetime	>10 year	>10 years	>10 years	>10 years
Data rates	<70 kbps	<1 Mbps	<150 kbps	<1 Mbps
Chipset costs	2 $	2 $	<2 $	<2 $

4. LTE-based Solutions for IoT Purposes

From Nokia perspective radio technologies for licensed spectrum are the most promising in the long term. The following section outlines the most important aspects of the LTE technology being under development for IoT purposes: LTE-M and NB-IoT solutions.

New UE categories have been studied [2] and specified [3] for LTE-M and NB-IoT solutions. **Table 4** shows a comparison of UE categories with respect to the IoT specific system requirements.

Table 4 Comparison of UE categories

UE category	Cat. 4	Cat. 1	Cat. 0	Cat. M1	Cat. M2
3GPP standard	Release 8	Release 8	Release 12	Release 13	Release 13+
LTE solution	LTE	LTE	LTE-M	LTE-M	NB-IoT
DL peak rate	150 Mbps	10 Mbps	1 Mbps	1 Mbps	200 kbps
UL peak rate	50 Mbps	5 Mbps	1 Mbps	1 Mbps	144 kbps
Number of antennas	2	2	1	1	1
Duplex mode	Full duplex	Full duplex	Half duplex	Half duplex	Half duplex
UE reception bandwidth	20 MHz	20 MHz	20 MHz	1.4 MHz	200 kHz
UE transmit power	23 dBm	23 dBm	23 dBm	20 dBm	23 dBm
Modem complexity	100%	80%	40%	20%	~10%

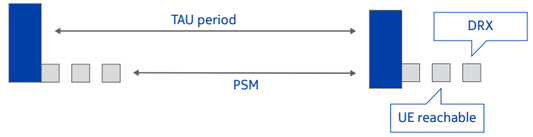
The following techniques have been specified for LTE-M and NB-IoT solutions in order to meet IoT connectivity objectives:

- Techniques to reach extended coverage (up to 20 dB gain):
  - Time diversity with relaxed latency requirements (repetition of data)
  - Power boosting, that is data transmitted with higher power in a narrower bandwidth
- Techniques to reach longer battery lifetime
  - Power Saving Mode (PSM)
  - enhanced DRX (eDRX)

Today's mobile phones can offer only few weeks of a standby time. This requires charging or changing batteries on a monthly basis which is not feasible due to high maintenance costs. Therefore, a device specific **Power Saving Mode (PSM)** was introduced in Release 12 to significantly improve the device battery lifetime.

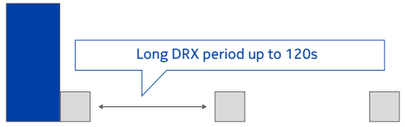
A device that supports PSM connects to the network for a certain period of time that is required for Attach or Tracking Area Update (TAU) procedures. The device remains reachable (by checking for paging according to the regular DRX cycle) for Mobile Terminated (MT) transaction until the transition from connected to idle mode. When the device switches from connected to idle mode, it triggers a timer which, after expiring, switches the device to the power saving mode. In the power saving mode the device is not reachable (it does not check for paging), although still being registered to the network. The device remains in the power saving mode until Mobile Originated (MO) transaction (for example a periodic TAU or uplink data transmission) requires it to connect to the network.

Figure 2 Power Saving Mode (PSM)



In Release 13 further improvement in battery lifetime is standardized to enable **enhanced DRX (eDRX)** with respect to discontinuous reception. With this improvement the eDRX cycle can be configured up to 120 s (instead of previous 2.56 s).

Figure 3 Enhanced DRX (eDRX)



Based on 3GPP power consumption model [1] PSM or eDRX improvements can extend the battery lifetime up to 10 years for delay-tolerant traffic.

5. Conclusion

IoT offers multi-dimensional business opportunities and many technical challenges at the same time. Several IoT solutions are currently under development to reach IoT performance objectives. Energy efficiency, extended coverage, and device costs are the main challenges for the IoT connectivity systems. Additionally, data security and data privacy aspects must be addressed to mitigate threats and make IoT successful in commercial applications.

References

[1] 3GPP TR 45.820; Cellular system support for ultra-low complexity and low throughput Internet of Things (CIoT).
[2] 3GPP TR 36.888; Study on provision of low-cost Machine-Type Communications (MTC) User Equipments (UEs) based on LTE.
[3] 3GPP TS 36.306; Evolved Universal Terrestrial Radio Access (E-UTRA); User Equipment (UE) radio access capabilities.

About the author

I have studied Communication Systems and Networks at the Technische Universität München and graduated from the Wroclaw University of Technology. With over ten years of experience in system research and software development I have been engaged in automobile system software standardization, telematics systems integration, and radio modules development. Currently, I am working in the field of LTE research and development. I am interested in software engineering, discrete mathematics, competence development, and cooperation with local universities. In my free time I practice triathlon.

Grzegorz Olender
R&D Manager
MN LTE

High Data Rates in Heterogeneous Networks

Bartłomiej Świderski
Engineer, Integration & Verification
MN HetRAN

NOKIA

1. Introduction

Popularity of applications such as real time video streaming, tracking or gaming are still increasing and generating huge amount of data. Poor operator performance suspends or blocks to content delivery of data application, which leads to customer dissatisfaction. Mobile operators have to fulfill requirements for good quality connection and sufficient bandwidth if they want to stay on the market. 3G (WCMDA) network infrastructure was deployed successfully in many places in the world. Currently, progressive transition toward LTE Long Term Evolution (LTE) technology can be observed. Such a big investment is causing high costs for the operator and leading to decreasing revenue. On the other hand, old network equipment, which is currently in use, should be smartly exploited. As **Figure 1** shows, 2G technology will be used over the years. Mobile operators endeavor to find out new ways to minimize Total Costs of Ownership (TCO)

Due to growing demands, mobile vendors attempt to meet those expectations. For this purpose Nokia is introducing a new product – Single RAN.

In the simplest way, Single RAN allows for running different radio technologies on a single hardware platform. This architectural approach gives a lot of benefits and reduces incurred cost of modernizing network infrastructure [1].

MN HetRAN designs, specifies and develops Single RAN solution known as SBTS.

Feature development process in Nokia was described in 2nd edition of Nokia Book [4]. Work in MN HetRAN I&V (Heterogeneous Radio Access Networks Integration & Verification in Mobile Networks) is based on similar principals as mentioned in related MBB LTE I&V. The goal is the same, to '*achieve proper software quality and deliver clear information about new functionalities of the product to the customer*'.

At the beginning and at the end of almost every test scenario, a PS (Packet Switch) call is made. It is a mandatory part of the verification process if the specific test step did not change the behavior of the SBTS. LTE session is established and the proper file is transferred in both directions, uplink and downlink. The obtained level of throughput should be consistent with the calculated value.

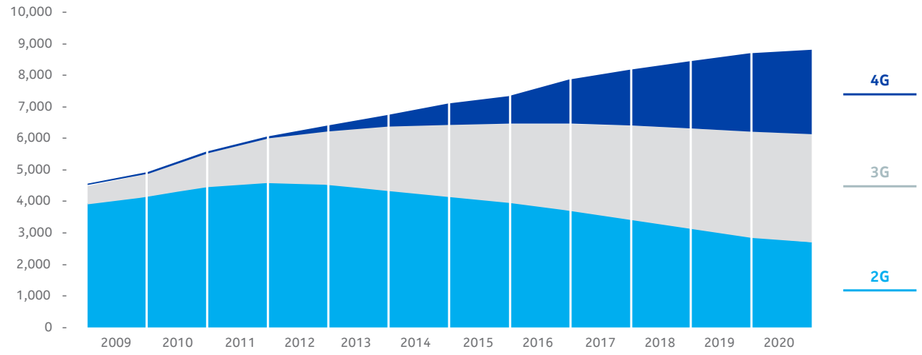
In this article, the data rates achieved from Single RAN Nokia Base Station will be described and compared.

2. Nokia Single RAN

The aforementioned Single RAN solution introduced to the network infrastructure should bring new benefits for mobile broadband operators. At the same time, it should be almost transparent for their clients.

The concept of Single Radio Access Network (RAN) has been shaping since 2008 but came across a lot of problems such as baseband deployment on shared resources through different technologies. Today, SBTS is capable of providing seamless interworking GSM, WCDMA and LTE radio access technologies in the same sector in the network of the mobile telecommunication operator. Via UE (User Equipment), the customer determines which type of network is

Figure 1 World consumption of 2G, 3G, 4G technology (millions connections excluding M2M), GSMA report 2016 [2]



preferred. According to current network conditions, traffic load and provisioned data plan, the management unit determines to which technology UE can be attached. Simultaneously, users are allowed to use different technologies within the same sector.

Shared radio modules are installed on the common multi-purpose hardware platform, with common IP-based transport as illustrated in **Figure 2**. HW is coordinated and operated by common Operational and Management System (O&M) which is integrated with a unified security system.

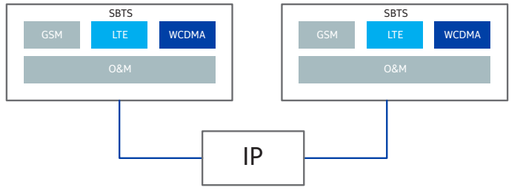
Single RAN is not standardized by 3GPP organization. Functionality of each RAT (Radio Access Technology, GSM, WCDMA and LTE) is ported by Nokia to SBTS solution separately in a flexible and unique way.

LTE was chosen as a base for the implementation of SBTS software due to high complexity and amount of features. First of all, IP transport was added to GSM and WCDMA.

Assessment of Single RAN should be considered from the mobile operator perspective:

- Flexibility
  - re-using base station equipment
  - extending capacity
- Simplicity
- Modularity
- Re-farming frequencies
- Sharing:
  - radio frequency-sharing
  - baseband pooling
  - transport sharing
  - network sharing
- Energy saving

Figure 2 SBTS concept



3. Throughput calculation

3.1. Communication channel capacity

Considering data rates obtained in any radio access technology, Shannon's Theorem should be the starting point. This concept specifies the amount of information which can carry a communication channel.

$$C \approx B \times \log_2(1 + SNR) \quad [2]$$

Where:

- **C** is known as the maximum channel capacity
- **B** as the bandwidth of the communication channel
- **SNR** is known as the signal to noise ratio.

The statement expresses the maximum possible data rate at which information can be transmitted without any error, limited by bandwidth, the level of signal and level of noise. This concept relies on obtained trade-off between bandwidth and SNR.

Shannon's formula allows for defining the capacity limit for a given channel, while taking into account any modulation, combination of any coding scheme, transmission, or decoding scheme. In other words, it estimates the best possible performance limit to be achieved.

3.2. EDGE and HSPA+ vs. LTE

Single RAN should be compatible with all communication standards from 2G to 4G. Considering 2G technology, it would seem that only GSM will be beneficial to implement under SBTS as ideal for the delivery of voice. However, GPRS (General Radio Packet Service) and EDGE (Enhanced data rates for GSM) are still applicable for clients using mobile phones that are not compatible with new 3G or 4G standards. Additionally, payment transactions are transferred through GPRS end terminals. Due to the fact that data rates available in GPRS and EDGE are very limited, they will not be of a concern in this article.

WCDMA with HSPA+ in comparison with LTE is still a leader in mobile broadband connections due to the greater availability and affordability of smartphones and larger network coverage. On the other hand, most HSPA+ networks around the world offer at least theoretical 42 Mbps download speed. Available download data rates implemented in Nokia SBTS equipment oscillates around 150 Mbps. Even though HSPA+ still has quite good potential, the analysis of SBTS performance will be focused on LTE capabilities.

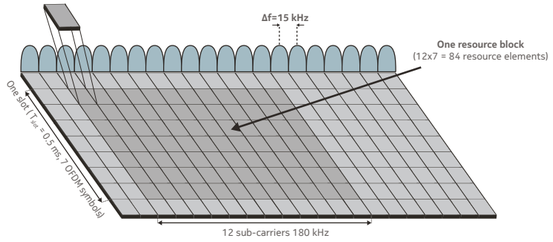
3.3. LTE throughput calculation

The calculation of possible data rates to obtained in LTE technology will firstly be considered from the physical layer perspective. The form of the physical layer is determined by the used OFDM multiplexing in downlink, and SC-FDMA in uplink. Data is transmitted over many narrow band orthogonal subcarriers of 180 kHz each.

In LTE, such terms are introduced – as illustrated in **Figure 3**:

Resource Element, which is the smallest unit of transmission data, consists of 1 subcarrier in frequency domain for the duration of 1 symbol in the time domain
Subcarrier Spacing, known as 15 kHz space between subcarriers
Time Slot, there 2 time slots in LTE 1 ms subframe, equal Transmit Time Interval (TTI), each consists of 7 OFDM symbols,
Resource Block, a transmission unit consisting of 12 subcarriers in one time slot which gives 84 Resource Elements (12 x 7 symbols)

Figure 3 LTE Resource Block [6]



LTE supports flexible cell bandwidth from 1.4, 3, 5, 10, 15, 20 MHz and modulation schemes QPSK (2 bits carrying per symbol), 16 QAM (4 bits), 64 QAM (6 bits).

$$DL_{THR} =$$

$$= \frac{(n \times RB \times RE \times slots\_per\_subfrm \times bit\_per\_mod\_sym \times MIMO\_rnk)}{TTI \times 1000}$$

The UE shall interpret the resource allocation field depending on the PDCCH/EPDCCH DCI format detection. Field consists of two parts, resource allocation header and actual resource block assignment. Firstly, UE should determine Transport Block Index used in the physical downlink shared channel based on Modulation and Coding Scheme. Refer to the above, TBS index and Number of Resource Blocks is assigned Transport Block Size [5]. TBS indicates how many bits can be transmitted in TTI.

For bandwidth 20 Mhz (100 RB) and MCS Index 28 channel transmits 75376 bits.

For MIMO Rank 2 (throughout will be 2 times of single chain throughput) Instantaneous physical DL Throughput is:

$$MAXDLTHR = \frac{75376 \times 2}{1000} = 150.752 \text{ Mbps}$$

The average TBS, including PSS/SSS, MIB, and SIB is equal 74348 bits, then

$$MAXDLTHR = \frac{743486 \times 2}{1000} = 148.695 \text{ Mbps}$$

and is later reduced by 1% of retransmission:

$$PDSCHTHR = 148.695 \times 0.99 = 147.2 \text{ Mbps} \quad (1)$$

After the consideration of MAC D-PDU header (3 Bytes) and RLC PDU header (13 Bytes) and Physical Downlink Shared Chanel throughput (1), there is calculated:

$$RLCTHR = 147.2 - \frac{2 \times (13 + 3) \times 8}{1000} = 146.952 \text{ Mbps} \quad (2)$$

When number of IP packets per transport block is 6.18 and PDCP header is 2 Bytes, then PDCP layer throughput after reduction from (2) is:

$$PDCPTHR = 146.952 - \frac{2 \times 3 \times 8 \times 6.18}{1000} = 146.754 \text{ Mbps} \quad (3)$$

At least considering UDP datagram, where UDP header is 8 Bytes and IPv4 header is 20 Bytes, we obtain UDP Download Throughput:

$$UDPTHR = 146.754 - \frac{2 \times (8 + 20) \times 8 \times 6.18}{1000} = 143.986 \text{ Mbps}$$

Performing an analogical calculation with different cell bandwidth (and some additional assumption in uplink) we obtain results presented in **Table 1**.

Table 1 Calculated UDP throughput for different cell bandwidth

Bandwidth (MHz)	20	20	15	15	10	10	5	5
Direction	DL	UL	DL	UL	DL	UL	DL	UL
MIMO mode	2x2		2x2		2x2		2x2	
Modulation QAM	64	16	64	16	64	16	64	16
UDP-THR (Mbits/s)	143,9	49.19	104.9	37.46	68,53	24,22	29,83	10,09

4. Testing environment

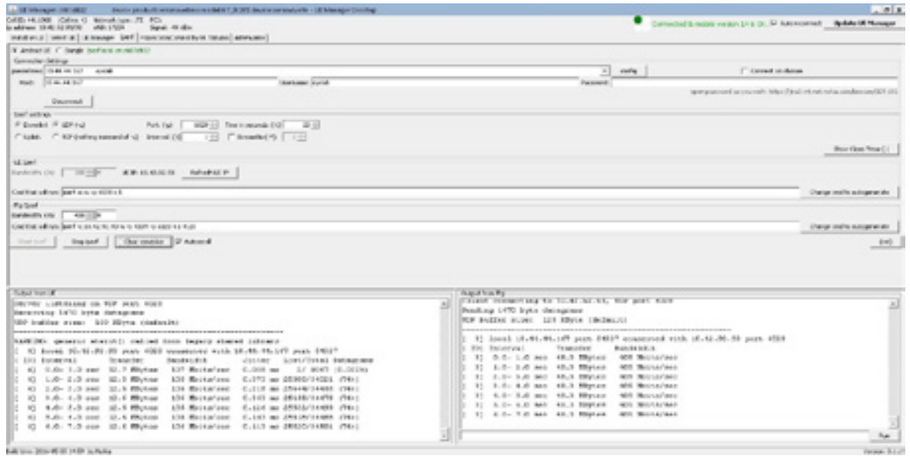
The purpose of the test execution was the verification of achievable throughput compared to theoretically calculated values. Tests were executed and measurements were gathered in Nokia HetRAN Laboratory.

The setup of the testline is exactly the same as the configuration which will be delivered to the client. Furthermore, adverse events like interferences and signal attenuation were not eliminated. The test environment should be as close as possible to the natural environment.

Test environment contains the following elements:

- Nokia SBTS configured according to particular profile
- UE equipment cat IV with Android, i.e. Samsung S4, allows downlink transfer to 150 Mbps
- FTP server
- Server for gathering logs
- Programmable attenuator and shield box to reflect live network behavior
- iperf – open source tool [4] for performing network measurements; allows to generate TCP or UDP data streams and measure throughput of network elements which carry them. Example is depicted on **Figure 4**.
  UE manager tool (Nokia internal tool) with implemented Android Debug Bridge which allows to communicate with and control UE with Android OS over an USB link from computer.

Figure 4 UDP download by iperf in UE Manager Tool



Furthermore, to verify SBTS capabilities and influences between different radio access technologies, testlines were additionally loaded. During LTE calls, downloads and uploads were simultaneously performed in WCDMA technology. WCDMA calls will be considered as background traffic and will not be measured. UE transfers big files in both directions via the ftp protocol.

Details of testline configurations:

Testline 1 (with LTE cell bandwidth 20 MHz or 15 MHz)

```
Band 1 (shared)
LTE 1+1+1 2T/2R
GSM 6+6+6 1T/2R
Band 2
WCDMA 4+4+4 1T/2R
Band 3
LTE 1+1+1 1T/2R
```

Testline 2 (with LTE cell bandwidth 10 MHz or 5 MHz)

```
Band 1 (shared)
WCDMA 3+3+3 1T/2R
GSM 6+6+6 1T/2R
Band 2 (shared)
LTE 2+2+2 2T/2R
GSM 4+4+4 1T/2R
Band 3
WCDMA 4+4+4 1T/2R
```


Table 2 Results of measurements

Bandwidth [MHz]	20	20	15	15	10	10	5	5
Direction	DL	UL	DL	UL	DL	UL	DL	UL
Calculated Throughput [Mbit/s]	143.9	49.19	104.9	37.46	68.53	24.22	29.83	10.09
Avg. Measured Throughput [Mbit/s]	141	49.2	105	35	70	24.2	28.7	9.37
Downloaded/ Uploaded data [MB] in 20s.	349	117	262	83.4	173	57.7	71.1	22.4
Jitter [ms]	0.199		0.38		0.173		0.358	

Figure 5 Measured Throughput for 20 and 15 Mhz during 20s [Mbit/s].

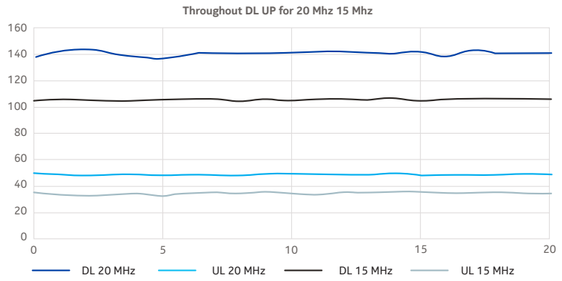
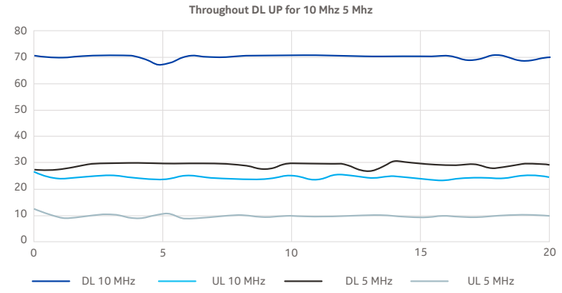


Figure 6 Measured Throughput for 10 and 5 Mhz during 20s [Mbit/s].



5. Summary

Results of the performed tests clearly show that implemented LTE technology on SBTS platform with shared resources does not have an impact on service quality. WCDMA packet traffic generated in the background is properly separated from LTE PS calls. Values of measured LTE throughput do not differ from the theoretical level. Data rates available on Single RAN platform with shared resources are the same as on e-Node LTE station. The new Nokia product – SBTS – is able to use existing RATs to gain the best performance at the same time coordinating their advantages. The nearest years will verify if Single RAN, along with shared resources will be able to substitute typical BTS stations. The ball is in mobile operators' court now.

References

[1] Single RAN Advanced Evolution: The future just got simpler, Nokia White paper, 2014
[2] J.Cichocki, j. Kolakowski, UMTS system telefonii komorkowej trzeciej generacji, 2003.
[3] The Mobile Economy 2015 Report GSMA Intelligence
[4] Iperf, https://sourceforge.net/projects/iperf2/
[5] Evolved Universal Terrestrial Radio Access (E-UTRA); Physical layer procedures3GPP TS 36.213 V13.1.1, 03.2016
[6] http://www.tutorialspoint.com/lte/lte_ofdm_technology.htm

About the author

I graduated as a Telecommunication Engineer of Warsaw University of Technology with a few years of experience in GSM, WCDMA and LTE networks.
I have started my career working for a mobile operator which gives me a broader view and experience in my current employment for a mobile vendor.
Our team is responsible for integration and verification of Nokia Single RAN product.

Bartłomiej Świderski
Engineer, Integration & Verification
MN HetRAN

The concept of Idle-Mode Load Balancing for Distributing UEs According to the Operator's Needs

Paweł Mikołajczyk
Specialist, LTE Control Plane
MN LTE

NOKIA

A growing number of active users within LTE networks has led to a considerable overload of the operators' layers. As a result, network vendors are expected to provide solutions with the aim of leveraging the distribution of User Equipment (UE) among the layers used according to the operators' needs. The purpose of this article is to discuss one such possible solution – a balanced distribution of UEs in idle mode (there is no RRC connection established for a UE).

Thanks to a balanced distribution in idle mode, the UEs which return to connected mode will already be attaching to cells on the desired layers. Thus, this solution prevents both an unequal distribution of active users as well as the overload situations.

The idle mode mobility of UEs is based on cell selection and cell reselection procedures, described in 3GPP standard document 36.304 [1]. These procedures use specific reselection priorities, which can be provided to a UE with common signaling (within System Information Broadcast messages) or with dedicated signaling (within the **RRC:RRCConnectionRelease** message, which is not used for redirection purposes). The former approach affects all UEs in the same way, while the latter one might be used to provide configuration for each UE separately.

The purpose of idle-mode-load-balancing concept is to use the mechanism defined by 3GPP in order to distribute UEs among layers according to operator's needs.

1. The basic concept of Idle-mode Load Balancing

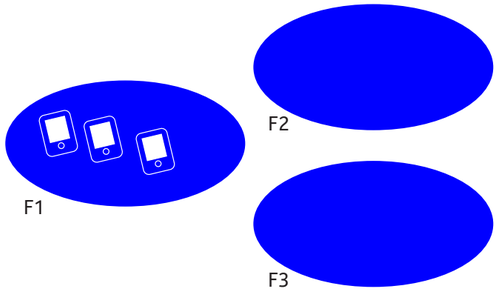
As already mentioned above, the main type of mobility in idle mode is cell reselection. The algorithm itself is implemented inside a UE, but E-UTRAN is able to control it with the parameters provided to the UE within System Information Broadcast (SIB) procedures. The most important parameter for the idle-mode-load-balancing concept is *CellReselectionPriority*, which has been defined by the 3GPP standard document 36.331 [2] in the way illustrated in **Listing 1**.

Listing 1 3GPP standard definition of CellReselectionPriority

```
-- ASN1START

CellReselectionPriority ::=                    INTEGER (0..7)

-- ASN1STOP
```

This parameter defines the priority for each target layer used by a UE in the cell reselection procedure, where value 7 stands for the highest priority. Target layers are measured by the UE, and the layers whose measured quality is acceptable for cell reselection are sorted in descending order according to the values of *CellReselectionPriority*. When an eNB provides this priority value only within System Information Broadcast messages (SIB1 and SIB5 for intra-LTE target layers; SIB6, SIB7, and SIB8 for inter-RAT target layers), then the behavior of all UEs will be the same, which may lead to an unequal UE distribution among target layers and, finally, to overload situations (**Figure 1**).

Figure 1 Without Idle-Mode Load Balancing, UEs always camp on a best-quality layer even if cell reselection priorities for all target layers (F1, F2, F3) are equal.



Therefore, another solution is to use a dedicated value of *CellReselectionPriority*, which can be provided to each UE separately during the transition from RRC_CONNECTED to RRC_IDLE and will be valid for such a UE for a configured amount of time (timer T320 in minutes defined by 3GPP 36.331 [2]). The RRC message **RRCConnection_Release** contains an optional information element (IE) called *IdleModeMobilityControlInfo*, which allows sending a dedicated value of *CellReselectionPriority* for each target layer (please refer to **Listing 2**).

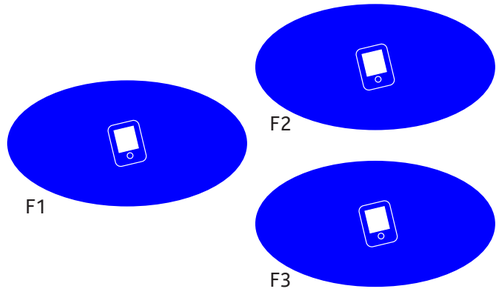
Listing 2 3GPP standard definition of IdleModeMobilityControlInfo

```
IdleModeMobilityControlInfo ::=          SEQUENCE {
   freqPriorityListEUTRA,
   freqPriorityListGERAN,
   freqPriorityListUTRA-FDD,
   freqPriorityListUTRA-TDD,
   bandClassPriorityListHRPD,
   bandClassPriorityList1XRTT,
   t320        ENUMERATED {
       min5, min10, min20, min30, min60, min120, min180, spare1}
   ...,
```

Inside the *IdleModeMobilityControlInfo* IE, each target RAT can be configured as a combination of target layer and a corresponding value of *CellReselectionPriority* for this layer.

The idle-mode-load-balancing concept shall use the *IdleModeMobilityControlInfo* IE for a balanced distribution of UEs among configured layers (**Figure 2**).

Figure 2 With Idle-Mode Load Balancing, UEs may be distributed equally among target layers with the Weighted Round Robin algorithm.



During each RRC Release procedure which does not contain redirection information (*redirectedCarrierInfo* IE), the eNB should check if the algorithm for Idle-mode Load Balancing will be started or not. It can be used for all released UEs, none of them, or just a percentage of UEs according to operator's needs. The UEs which are not handled within the scope of idle-mode-load-balancing algorithm will behave in idle mode according to the values of *CellReselectionPriority* from System Information Broadcast. For the UEs handled by the idle-mode-load-balancing algorithm, the eNB shall try to assign dedicated values of *CellReselectionPriority* and build the *IdleModeMobilityControlInfo* IE. The building of this information element can be divided into two steps, as shown in next sections of this article.

2. Selecting the main target for Idle-mode Load Balancing

The first step of preparing target layers with *CellReselectionPriority* is to find the so-called Primary Target. That is a single target layer which will have the highest possible priority assigned to it (that is, value 7) inside the *IdleModeMobilityControlInfo* IE (note that this could be either intra-LTE or inter-RAT target layer).

The eNB has a list of target layers (F_x) that will be candidates (C) for Primary Target selection: C = F_x, F_2, ... , Fn. All target layers from this list are checked against stored UE capabilities whether a UE is capable of accessing each Fx target layer, and layers which are not supported are eliminated from the candidate list. In case all target layers have been eliminated and a candidate list is empty, the Primary Target is not selected, the highest priority is not set in the *IdleModeMobilityControlInfo* IE, and the eNB can proceed with the next steps of the algorithm. However, if the candidate list is not empty, then the eNB shall try to select one target as the Primary Target from the remaining target layers, using the Weighted Round Robin algorithm.

Each target layer (F_x) has a corresponding weight (W_x) assigned. The weight value can be interpreted as a probability that the F_x target layer will be selected as the Primary Target (in the context of all assigned weight values). By configuring the weights assigned to target layers, the operator is able to distribute UEs among these layers in an expected way. The simplest realizations of the Weighted Round Robin algorithm are based on a randomized function; therefore, the expected distribution weights are achieved in a longer timeframe (for example 24 hours) and within a large number of RRC Release procedures (this is not an issue since there are thousands of RRC Release procedures observed in an LTE cell over a period of 24 hours).

Thus, by employing the configuration parameters and Weighted Round Robin algorithm, the eNB selects a Primary Target layer which will have the highest priority value 7 assigned in the *IdleModeMobilityControlInfo* IE.

3. Additional targets for Idle-mode Load Balancing

The next step of idle-mode-load-balancing algorithm is to fill the *IdleModeMobilityControlInfo* IE completely with the so-called Secondary Targets. There are two solutions offered to the operators to choose those additional target layers with cell reselection priorities.

The first one is to follow the hierarchy of cell reselection priorities configured for System Information Broadcast. This solution is valid only if the Primary Target was selected in the previous step. If the Primary Target is empty, this step is skipped, the *IdleModeMobilityControlInfo* IE is not added to the *RRCConnectionRelease* message, and a UE in idle mode follows the priorities from SIB. However, assuming that the Primary Target is not empty, the eNB will try to use all target layers for Secondary Targets selection (with the exclusion of the target layer chosen as the Primary Target) which are used in SIBs, have assigned *CellReselectionPriority*, and are supported by UE capabilities.

When all candidates for the Secondary Targets are collected, the eNB will lower their priority from *CellReselectionPriority* to prevent using the highest priority value (which is already reserved for the Primary Target) and in order to maintain the hierarchy between priorities of all other candidates for Secondary Targets.

The target layers from such a candidate list with corresponding lowered priorities will be added to the *IdleModeMobilityControlInfo* IE as Secondary Targets. Thanks to this solution, the UE will have had the Primary Target selected by the balancing algorithm in the previous step. However, when reselection to this target in idle mode is not possible, the behavior of the UE will be the same as the common behavior from System Information Broadcast.

The second solution to Secondary Targets' selection is to use a set of configured target layers with corresponding dedicated values of cell reselection priority (whose range excludes the highest priority value 7) which are independent of *CellReselectionPriority* values configured within SIB. Therefore, the operator has to provide a set of candidate target layers F_1, F_2, ... , F_n together with corresponding values of dedicated priorities p_1, p_2, ... , p_n.

The Secondary Targets will be built from all those target layers for which: F_x was not selected as the Primary Target, F_x is a target layer configured for SIB, and F_x is supported by UE capabilities. All target layers which remain on the candidate list will have a dedicated cell reselection priority px assigned to them and will be added to the *IdleModeMobilityControlInfo* IE as Secondary Targets (with a reduced range, excluding the highest priority value). Note that in this particular solution, the hierarchy of priorities is different from the one in System Information Broadcast; thus, Secondary Targets will be added to the *IdleModeMobilityControlInfo* IE even if the Primary Target is empty. With this solution, the behavior of the UE in idle mode is fully based on dedicated signaling.

Obviously, if both the Primary and Secondary Targets are empty, then the *IdleModeMobilityControlInfo* IE is not sent in the *RRCConnectionRelease* message.

So far we have discussed the basic scheme of idle-mode-load-balancing algorithm. However, there are still several optional enhancements of the algorithm which can bring further benefits. We will elaborate on them in the following sections.

4. Differentiation of targets based on UE SPID or PLMN

The idle-mode-load-balancing functionality can benefit the operators even more significantly when it is coupled with the Mobility Profile concept. The use of Mobility Profiles enables differentiating UE behavior based on UE PLMN ID or UE Subscriber Profile Identifier (SPID). The latter is assigned to the UE in Home Subscriber Server (HSS) and may be provided to the eNB via S1 interface messages.

The UEs based on the value of PLMN or SPID are assigned to different Mobility Profiles, in which they receive different mobility schemes (including different percentage of UEs to be balanced and configuration parameters) and target configuration, so their mobility behavior can differ from one another. Idle-mode Load Balancing can make use of such a possibility. In addition, it allows configuring target layers and values for a dedicated cell reselection priority inside a Mobility Profile instance. Therefore, UEs belonging to different operators or having different privileges (defined by SPID) can be configured at present to behave according to the operator's needs in idle mode.

5. Enhancement taking the load of serving cell into account

Another enhancement of the basic algorithm is the relation between an idle-mode-load-balancing trigger and a current load in the serving cell.

The load in a cell is the measured Composite Available Capacity (CAC), which is calculated by Radio Resource Management (RRM) functions. Depending on the available capacity of a serving cell (for example, with the threshold of available capacity used as a configuration parameter), the operator may decide to start the balancing algorithm or not. If the calculated CAC value is below or equal to the configured capacity threshold, the eNB starts the idle-mode-load-balancing algorithm; otherwise, the algorithm is not started, and the UE is released immediately without the *IdleModeMobilityControlInfo* IE.

As a result of this enhancement, the balancing of UEs among target layers will be performed only in high-load situations while during low-load in a serving cell, the UEs in idle mode will stay on the frequency of a serving cell.

6. Enhancement taking the quality of target layer into account

When the basic idle-mode-load-balancing algorithm selects an LTE inter-frequency target layer as the Primary Target, a new optional extension can be used, and an eNB may check the quality of target cells on target layers. To this end, the eNB may trigger event-based measurements (for example A4 measurements). This enables measuring available cells on the target layer and comparing the quality of the best cell on the target layer with the quality of the serving cell (RSRP and RSRQ quantities can be used for this purpose, either one of them or both).

Measurement-based additions to the algorithm can be fully configured by the operator in the sense that, for each layer, it can be separately decided if Idle-Mode-Load-Balancing-related measurements are allowed or not; the RSRP/RSRQ minimum thresholds and delta quality thresholds are also set according to customer needs.

When measurements are not started whatsoever, the eNB uses the *IdleModeMobilityControlInfo* IE as it has been prepared in the basic algorithm. However, when A4 measurements are started, then the eNB waits for a measurement report. Measurements are guarded by a supervision timer, and if a measurement report is not set within a fixed timeframe, idle-mode-load-balancing measurements are

assumed to have been unsuccessful, and the *IdleModeMobilityControlInfo* IE is not sent to the UE.

When the A4 measurement report is provided by the UE on time, the eNB will evaluate the results against the quality thresholds configured by the operator. With the A4 event, the UE should first provide a cell that meets the measurement configuration criteria. The quality of the target cell should reach a minimum quality threshold and be a delta-quality threshold better than the quality of the serving cell. If both conditions are met, the eNB assumes the target layer has the expected quality, and the *IdleModeMobilityControlInfo* IE is sent to the UE; otherwise, the UE will be released without this information element.

Thanks to this enhancement, UEs are shifted to the target layer only when it meets the expected quality criteria. If it does not, the UEs stay on the serving cell layer. As a result, at the relatively small cost of adding a delay to the RRC Release procedure (a maximum of 3000 ms for a supervision timer, but normally UEs report the A4 event and waiting for the whole timer to expire is not required), such a solution reduces the number of reselection procedures in idle mode (since UEs are distributed over better quality layers), which in turn leads to a UE battery life saving.

7. Special configuration for CA-capable UEs

Since many operators have recently started employing the carrier aggregation (CA) functionality within their networks, it is necessary for Idle-mode Load Balancing to be extended to meet their needs and provide special handling for CA-capable UEs.

The eNB is capable of distinguishing between CA-capable UEs and non-capable ones and providing a different behavioral pattern for either of them. A complete configuration of the balancing algorithm can have separate settings for different types of UEs. Keeping the same set of parameters related to the algorithm, those parameters can be grouped and used for CA non-capable UEs and for different types of CA-capable UEs. The eNB will allow configuring different groups for 2CC, 3CC, 4CC, and 5CC UEs.

A carrier-aggregation-related algorithm will check whether the configured target layer can be used for a given component carrier (CC) combination and whether the target layer can be used as a PCell by the UE. All checks can be done against the UE capabilities stored in the UE context (for example, the eNB may use the *SupportedBandCombination* IEs or the *BandParametersUL* IE).

The eNB can optimize UE behavior for the highest supported number of component carrier combinations; however, if there is no suitable target on the highest CC, then lower CCs should be checked as well. With such an approach, the operator may deploy his network in such a way that specific layers are assigned to a given CC type, and UEs are distributed in an idle mode to the layers, in which the UE may get benefits from its carrier aggregation capabilities. Hence, further mobility to shift UEs to CA-capable layers in an active mode will not be required.

8. Avoiding IMLB for special types of UEs or UE behavior

Throughout the evolution of LTE, there have occurred new, specific types of UEs which have to be treated in a unique manner; for example, 3GPP Category-0 UEs or LTE-M UEs. In addition, new functions are supported which cannot be easily merged with the current mobility schemes used by the eNB; for instance, UEs interested in Sidelink communication or MBMS services. Therefore, the idle-mode-load-balancing algorithm has to distinguish between such UEs (whether they are of a specific type or interested in specific functions), and the algorithm is not started for such UEs (the UE is released without the *IdleModeMobilityControlInfo* IE).

9. Deployment examples

The solutions described in the previous sections have been implemented and used in live networks. In this section, two examples, from the networks in Korea and Switzerland, are presented. The first deployment example makes use of the idle-mode-load-balancing algorithm for its main purpose, that is, an equal distribution of UEs on several layers. The Weighted Round Robin algorithm is configured with the same weights for each target layer. This approach enables gathering a similar number of active UEs on all layers so that the resources of serving cells are used in an efficient way, and there is no need to use an active load balancing mechanism (see also **Figure 2**). The other deployment example uses dedicated cell reselection priorities for steering UEs from two operators in shared networks (**Figure 3**).

Figure 3 Without Idle-mode Load Balancing, UEs may be distributed to target layers (example F1, F2, F3 belonging to different RATs) not belonging to their own operator.

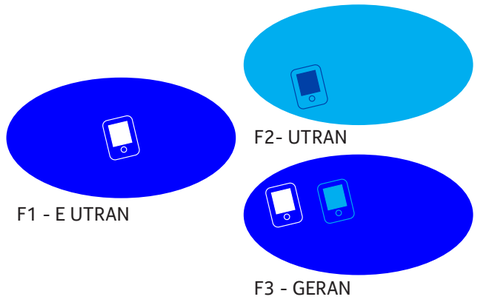
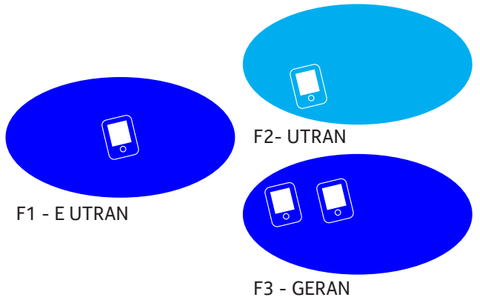


Figure 4 With Idle-mode Load Balancing, UEs should always be distributed to the target layers (example F1, F2, F3 belonging to different RATs) belonging to their own operator (with a configuration of dedicated priorities in Mobility Profiles).



Target layers are defined in Mobility Profiles and, based on the PLMN value, the UE is assigned to a corresponding profile with its own set of target layers with dedicated cell reselection priorities. Thanks to this approach, the UEs belonging to both operators use their own mobility schemes in idle mode instead of a common configuration from SIB (with a specific hierarchy of target layers) because these operators have different inter-RAT deployments, and UEs always camp on the correct layer of their own operator (**Figure 4**).

10. Conclusion

The major objective of this article was to present the basic approaches and solutions to idle mode mobility schemes, with a specific focus on balancing the distribution of UEs across different target layers. The main algorithm is designed in a flexible way, allowing easy extensions for the future. In the foreseeable future, it is expected that first commercial deployments for a CA-related algorithm will come into existence, and further research will be conducted on how to make the UE distribution over specific target layers more precise.

References

[1] 3GPP TS 36.304: "Evolved Universal Terrestrial Radio Access (E-UTRA); UE Procedures in idle mode", March 2016.

[2] 3GPP TS 36.331: "Evolved Universal Terrestrial Radio Access (E-UTRAN); Radio Resource Control (RRC) Protocol Specification", March 2016.

About the author

I have been working for the LTE project system specification since 2007. During this time, I was involved in RNL O&M and configuration parameters modeling, inter-RAT mobility (e.g. CS fallback, SRVCC), Idle-mode Mobility (e.g. redirection procedures, Idle-mode Load Balancing), VoLTE Mobility (e.g. Service-based HO) and recently Bearer-Management-related topics.

Paweł Mikołajczyk
Specialist, LTE Control Plane
MN LTE

A Statistical Method for Estimation of the Maximum Number of Carrier Component Connections per eNB

Martin Kollar
Specialist; Operability
MN LTE

NOKIA

In this paper, a statistical method for estimation of the maximum number of carrier component connections per eNB from per cell based PM counters is presented.

Introduction

Nowadays the pricing in E-UTRAN is done based on the maximum number of active UEs per eNB. The maximum here, is meant as max of peaks measured in a predefined sampling interval (1 second) in a time frame (i.e. one day or month).The counting is done based on PM counters. If the current threshold for the maximum number of active UEs per eNB is reached, operator has to buy a new license to increase the capacity. Introduction of the Carrier Aggregation related features opens the door for higher end user and cell throughput via activating one or more SCells for the UE [2]. Operators are thus no longer interested only to the maximum number of active UEs but maximum number of Carrier Component Connections (CCCs). The UE without an activated SCell is considered as the one with 1 CCC. The UE with only one activated SCell is considered as the one with 2CCCs, with two SCells activated as the one with 3CCCs and finally with three SCells activated as the one with 4 CCCs.

Problem

New pricing based on the maximum number of CCCs is not feasible per eNB basis due to some obstacles in counting UEs with active SCell(s) on eNB level.

Method Proposal to Solve the Problem

The idea is based on the fact that the number of CCCs per *i*-th cell can be seen as a statistical variable ζ_i represented via its mean value $E(\zeta_i)$ and diversity $E(\zeta_i - E(\zeta_i))^2$. Using this statistical parameters then the maximum number of CCCs per the i-th cell can be estimated as $E(\zeta_i) + 3^* \text{sqrt}(E(\zeta_i - E(\zeta_i))^2)$, where $(E(\zeta_i) - 3^* \text{sqrt}(E(\zeta_i - E(\zeta_i))^2), (E(\zeta_i) + 3^* \text{sqrt}(E(\zeta_i - E(\zeta_i))^2))$ represents an interval the sample of the number of CCCs will fall in with 99.5 % probability, known also as the 3σ approach [1].

Considering then an eNB with N cells the maximum number of CCCs can be estimated using the formula:

$$MAX\_CC\_eNB = \sum_{i=1}^{N} E(\zeta_i) + \sqrt{\sum_{i=1}^{N} E(\zeta_i - E(\zeta_i))^2} \qquad (1)$$

The same approach is then applicable within the i-th cell considering that number of UEs with one, two, three and four CCCs are understood as mutual independent statistical variables. To compute then maximum number of CCCs per eNB would first consider to compute mean value and diversity of each statistical variable related to the given number of CCCs and then combine the results of all the cells within the observed eNB applying the above principles and Eq. 1.

The maximum number of CCCs per eNB then can be computed using the PM counters that have been available since LTE16A as follows

$$MAX\_CC\_eNB = \sum_{i=1}^{N} AVG\_CCC\_Cell_i + \sqrt{\sum_{i=1}^{N} (3\sigma\_Cell_i)^2} \qquad (2)$$

where

$$AVG\_CCC\_Cell = avg\left(1.\left(M8051C57 - \frac{M8051C116}{100} - \frac{M8051C117}{100} - \frac{M8051C21}{100}\right) + 2.\frac{M8051C116}{100} + 3.\frac{M8051C117}{100} + 4.\frac{M8051C21}{100}\right) \qquad (3)$$

where

$$1.\left(M\,8051\,C\,57 - \frac{M\,8051\,C\,116}{100} - \frac{M\,8051\,C\,117}{100} - \frac{M\,8051\,C\,21}{100}\right)$$

provides mean value of statistical variable representing number of UEs with one CCC, $2.\frac{M\,8051\,C\,116}{100}$ with two CCCs, $3.\frac{M\,8051\,C\,117}{100}$ with three CCCs and $4.\frac{M\,8051\,C\,21}{100}$ with four CCCs, and

$$3\sigma\_Cell_i = \sqrt{\begin{array}{l}(M8051C58 - M8051C57)^2 - \left(M8051C28 - \frac{M8051C116}{100}\right)^2 \\ -\left(M8051C29 - \frac{M8051C117}{100}\right)^2 - \left(M8051C30 - \frac{M8051C21}{100}\right)^2 \\ +4.\left(M8051C28 - \frac{M8051C116}{100}\right)^2 + 9.\left(M8051C29 - \frac{M8051C117}{100}\right)^2 \\ +16.\left(M8051C30 - \frac{M8051C21}{100}\right)^2\end{array}} \qquad (4)$$

where

$$(M\,8051\,C\,58 - M\,8051\,C\,57)^2 - \left(M\,8051\,C\,28 - \frac{M\,8051\,C\,116}{100}\right)^2$$

provides $(3\sigma)^2$ of statistical variable representing number of UEs with one CCC, $4.\left(M\,8051\,C\,28 - \frac{M\,8051\,C\,116}{100}\right)^2$ with two CCCs, $9.\left(M\,8051\,C\,29 - \frac{M\,8051\,C\,117}{100}\right)^2$ with three CCCs, $16.\left(M\,8051\,C\,30 - \frac{M\,8051\,C\,21}{100}\right)^2$ with four CCCs, i represents the

i-th cell in the given measurement period (15 minutes) and N represents the number of the cells in the observed eNB. The MAX_CCC_eNB represents the maximum number of CCCs in the observed eNB for the given measurement period (15 minutes).

Table 1 summarizes the PM counters used within the above formulae.

Table 1 PM Counters used within the Eq. 3 and 4

PM Counter ID	PM Counter Name
M8051C21/100	Average number of UEs with three activated SCells
M8051C28	Maximum number of UEs with two activated SCells
M8051C29	Maximum number of UEs with three activated SCells
M8051C30	Maximum number of UEs with four activated SCells
M8051C57	Active UE per Cell average
M8051C58	Active UE per Cell max
M8051C116/100	Average number of UEs with one activated SCell
M8051C117/100	Average number of UEs with two activated SCells

Test

The formulae from the previous chapter is based on the PM counters that are going to be available in the next Nokia release, which is scheduled in August 2016. Thus the testing method was done based on legacy counters existing in earlier releases. Due to missing counters providing the maximum number of UEs with active two, three and four SCells the testing focused on estimating the maximum number of active UEs based on average and maximum number of active UEs measured via PM counters in the cells within the observed eNB given by the following formula:

$$MAX\_CCC\_eNB = \sum_{i=1}^{N} M8001C223_i + \sqrt{\sum_{i=1}^{N}(M8001C224_i - M8001C223_i)^2} \quad (5)$$

where i represents the i-th cell from the observed eNB in the given measurement period (15 minutes) and N represents the number of the cells in the observed eNB. **Figure 2** summarizes the PM counters used within the Eq. 5.

Table 2 PM Counters used within the Eq. 5

PM Counter ID	PM Counter Name
M8001C223	Active UE per Cell average
M8001C224	Active UE per Cell max
M8018C1	Active UE per eNB max

Then the obtained results were compared with the legacy counter M8018C1 (see the last row in the **Table 2**) for the same eNB and measurement period. The tests were done on the "live data" network with five days of continuous observation. The Fig. 1 shows the graph of the counter M8018C1 in dark blue and the KPI MAX_CCC_eNB in light blue using the Eq. 5.

Figure 2 shows a graph of the relative error between the counter M8018C1 and the KPI MAX_CCC_eNB calculated as follows:

$$\delta = \frac{(M8018C1 - MAX\_CCC\_eNB)}{M8018C1} * 100\% \quad (6)$$

It can be observed that majority of the error samples, so called uncertainty, ranges between +5 and -5%. Considering longer aggregation intervals, like day, the relative error was about only 1%.

The test was performed to check the reliability of the results for max number of connections on eNB level obtained from cell based average and max counters. Although, it does not consider any 2CCC, 3CCC or 4CCC in the equation, the methodology remains the same.

Figure 1 Graph of the counter M8018C1 in dark blue and the KPI MAX_CCC_eNB in light blue using the Eq. 5 as function of time.

Conclusion

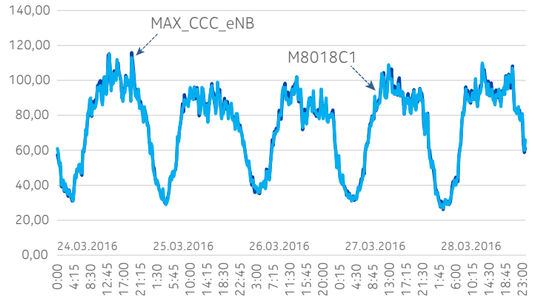
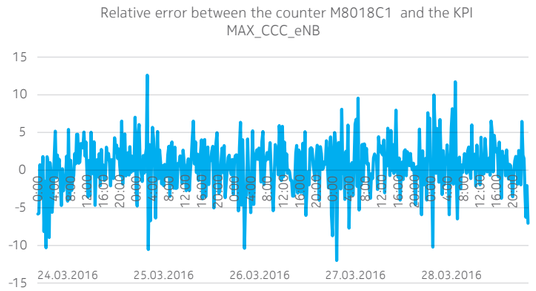


Figure 2 Graph of the relative error between the counter M8018C1 and the KPI MAX_CCC_eNB as function of time.



A method for the measurement of the maximum number of CCCs per eNB has been presented. The method tends to provide the reliable results of the maximum number of connections on eNB level from the average and maximum number of connections on cell level from all the cells within the observed eNB. In next Nokia release scheduled in August 2016 some other tests will be done to check reliability of the obtained results using the Eq.2.

References

[1] Kanti V. Mardia, J. T. Kent, J. M. Bibby, Multivariate Analysis (Probability and Mathematical Statistics), 2000.

[2] A.Z.Yoins, M.F.L. Abdullah, Carrier Aggregation Technique for Improving LTE-Advanced System, 2013.

About the author

I was born in Spišská Nová Ves, Slovakia, on 7th December, 1974. I gained an Ing. (M.Sc.) degree in electronics and multimedia telecommunications from the Faculty of Electrical Engineering and Informatics (FEI), Technical University (TU) Košice and a Ph.D. degree in measuring techniques from the FEI TU Košice, in 2000 and 2003, respectively. From 2003 till 2006, I was an assistant professor at the Department of Theory of Electrical Engineering and Measurement, FEI, TU Košice. Since 2006 I have been an operability specialist at Nokia (Nokia Solutions and Networks) on the O&M Performance Measurements area for GSM/GPRS/EDGE and E-UTRAN.

Martin Kollar
Specialist; Operability
MN LTE

Telecommunication System Engineering

Optical Interface Baseband Signal to RF Signal Delay Measurements for Radio Modules

Wojciech Zmyślony
Engineer, Software Development
MN RF & AA

NOKIA

1. Introduction

Radio modules are the last component of a mobile network in downlink transmission, and the first component in uplink reception. They introduce specific latency to the system, related to intentional buffering. The latency has to be known in advance and adjusted by the system module (the second component of a Base Transceiver Station) according to the particular needs. This latency is required by the 3rd Generation Partnership Project (3GPP), Common Public Radio Interface (CPRI) and Open Base Station Architecture Initiative (OBSAI) specifications [1], [2], [3]. The specifications cover also other details, like adjustments between downlink and uplink timing and maximum time alignment error [2].

All timing parameters (given by the system module to the radio module) are mapped by radio module software to the specific register configuration of integrated circuits responsible for digital signal processing.

Modern radio modules are not just simple transceivers. They feature many functionalities, such as:

- Supporting Multi-Radio Access Technologies (GSM, WCDMA, LTE) and wideband signals
- Using a large variety of adaptive algorithms, which help to increase the efficiency of power amplifiers (thus reducing power consumption)
- Improving spectral performance
- Compensating for analog effects, like aging or the impact of temperature on the radio transmission

This requires flexible and complex hardware solutions, configured by no less complex software. But high flexibility means that it is difficult to control the delay introduced by the signal processing path within the device. It is much simpler to measure the delay in the real system, and correct it if needed. Delay measurements are also needed for restart stability verification and regression testing purposes.

All the factors mentioned above force R&D teams to create specialized environments for radio module delay measurements. The challenge is to measure precisely the actual latency of a radio module, which is the delay between the digital data transmitted by fiber and the radio signal at frequency ranging from hundredths of MHz up to a couple of GHz.

2. Measurement

The most problematic issue in such measurements is the difference in the representation of the two signals. At one end of the radio module, there is a serial optical interface (OBSAI or CPRI) with complex data (commonly called IQ data, consisting of the real part I and the imaginary part Q) interleaved by control and empty data. At the other end, there is an RF signal OBSAI and CPRI interfaces use 8 b/10 b coding. In this type of coding special bit sequences are allowed, called control characters or K marks. In these protocols K marks are used to indicate the beginning of a 10 ms frame: K28.7 for OBSAI and K28.5 at hyper frame number (HFN) zero for CPRI. These K marks can be used to trigger a capture of OBSAI or CPRI data (by a dedicated OBSAI or CPRI sniffer). The relative position of the RF signal and the K mark can be determined through performing an additional synchronous capture of these signals. To observe K marks from OBSAI or CPRI data stream on an oscilloscope, a special adapter must be attached to the optical transceiver.

Once this data is available, all we need to do to calculate the delay is to compare both signals. The simplest way to do this is to transmit a single pulse at the beginning of 10 ms frame, and zero the rest of the payload. The comparison of the pulse position in radio data with the K mark indicates the delay.

However, such a simple approach has certain drawbacks. Firstly, the standard signal cannot be used for the measurement. Testing and measurements should always be performed on standard data, in this case wideband carriers. Secondly, preparing special patterns for measurement purposes only (instead of using standard 3GPP test models) requires additional effort. Moreover, the measurement performed this way is very imprecise, as is every method based on comparing rising or falling edges.

Instead, a more complex, but also much more accurate method of crosscorrelation can be used for this purpose. From the two captured signals a new vector can be calculated using the following formula [4]:

$$\hat{R}_{xy}(k) = \begin{cases} \sum_{n=0}^{N-k-1} x_{n+k}\, y_n^*, k \geq 0 \\ \hat{R}_{yx}^*(-k), k < 0 \end{cases}$$

Two vectors y and x of length N produce a new vector of length equal to 2N–1. If one signal is a shifted copy of the second one, the position of the maximum value (arg max function) of the new vector corresponds to the delay between these two signals.

The two signals used for crosscorrelation are:

- The RF signal captured at the oscilloscope
- IQ data extracted from OBSAI or CPRI stream, quadrature modulated and upconverted to center frequency of the RF carrier, and placed at the position of the K mark

The IQ data processing and cross-correlation has to be done by external specialized software. The position of the peak in the correlation plot corresponds to the delay between the IQ data and the RF signal, that is, to the radio module latency.

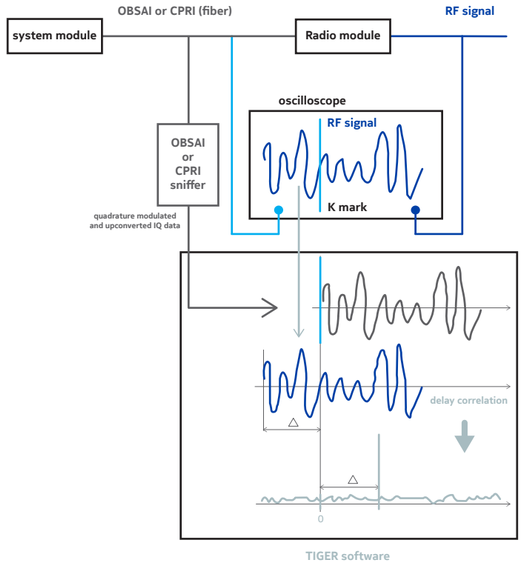
Timing in uplink direction can be measured in a similar way. An additional signal generator, connected both to the radio module and the oscilloscope, has to be installed. In uplink measurement, the correlation is also calculated between the RF signal captured by the oscilloscope and the IQ data captured by OBSAI or a CPRI sniffer.

3. Implementation

Nokia developed its own environment which allows engineers to measure precisely the delay in a given radio module, using standard signals (see **Figure 1**). It uses the following devices:

- A high speed oscilloscope
- A vector signal generator
- A test PC with a dedicated OBSAI/CPRI sniffing card equipped with a fiber optic socket

Figure 1 A simplified block diagram of the measurement



All devices are connected via a LAN network. The SCPI (Standard Commands for Programmable Instruments) interface is used to control the external devices and fetch data. The whole system is controlled by dedicated software, called TIGER (abbreviation for **TI**min**G** **E**nvi**R**onment). TIGER is available both as a GUI application (for individual measurements done manually, see **Figure 2**), and as a remote interface application (for automated, repeatable measurements).

The standard 3GPP test models are used both for downlink and uplink measurement. These signals guarantee continuous transmission, and good autocorrelation properties.

The measurement procedure starts with a coarse estimation of expected delay. Calculating delay correlation over very large number of samples has high memory requirements (which is limited, especially the oscilloscope capture memory) as well as processing requirements. This coarse estimation is done by passing to the application the requested OBSAI or CPRI timing related parameters. The application calculates the expected position of the quadrature modulated and upconverted IQ data relative to the RF signal and shifts the IQ data accordingly. Ideally, after such operation the two signals should overlap, and the peak of cross-correlation plot should be visible at point zero (see **Figure 3**). If the peak is not at zero, the timing settings of the radio module should be modified so that the measured and expected positions are aligned.

As the signal processing is done at radio frequencies, the delay correlation plot will contain many local maxima and minima, related to higher and lower correlation values caused by periodical carrier phase alignment (equal to reciprocal of carrier center frequency). As the actual delay corresponds not to the maximum point of delay correlation plot, but to the maximum point of envelope of this plot, an additional step is needed. The neighborhood of the maximum is approximated by a polynomial function, and the delay corresponding to the maximum value of this approximation is taken as the radio module latency (see **Figure 4**). This makes the measurement more accurate, and makes the result independent of carrier center frequency.

Aside from the relative delay value, the TIGER software calculates the absolute latency. The calculation is based on delay correlation result and the known coarse delay estimation calculated from OBSAI or CPRI parameters.

Nokia's TIGER software supports measurements for OBSAI and CPRI interfaces of all speeds, uplink and downlink, LTE TDD, FDD and WCDMA, all radio frequencies and all fiber lengths.

Figure 2 TIGER software graphical user interface

Figure 3 Time domain plots (top) and cross-correlation plot (bottom) depicting good time alignment (misalignment of ~22 ns**).**

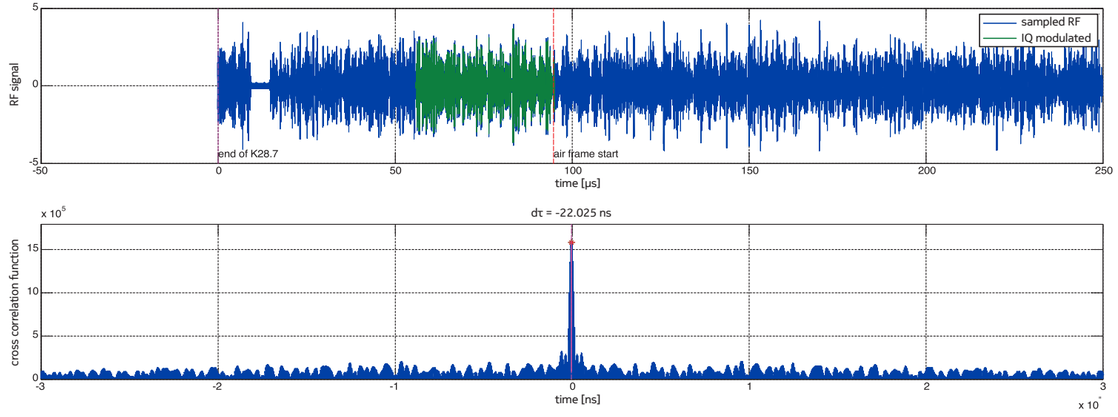
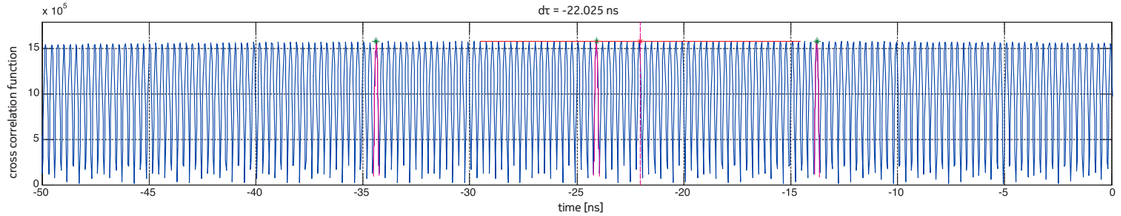


Figure 4 Zoom of peak of crosscorrelation plot. Visible are: local high correlation values (due to correlation at radio frequency; blue trace), the approximation of the envelope by polynomials (red horizontal line), and maximum value (magenta vertical line).



4. Accuracy and precision

Reliable results can be obtained when a good accuracy and precision are ensured.

To ensure good accuracy, measurement devices undergo periodical internal calibration of (especially the oscilloscope). Also, the latencies (group delays) of all setup elements like attenuators, coaxial cables and fibers used in the environment have to be measured, and then compensated for by the TIGER software.

The level of precision depends mainly on correct synchronization. To ensure that, all devices in the environment use external common reference clocks (the devices being the system module, the radio module, the oscilloscope, the signal generator and the OBSAI or CPRI sniffer). Best performance is achieved in daisy chain topology, when every device uses the reference clock of 10 MHz generated by the previous device.

5. Summary

The results achieved by the TIGER software and the environment discussed above are precise enough to verify the radio module timings against requirements.

Standard deviation of 50 consecutive measurements of 1800 MHz module is equal to ~0.023 ns in downlink and ~0.049 ns in uplink. Similar standard deviation values can be achieved also for other radio modules operating at different frequencies.

References

[1] Open Base Station Architecture Initiative, Reference Point 3 Specification Version 4.2, 2010.
[2] 3GPP TS 36.104 V13.3.0. Evolved Universal Terrestrial Radio Access (E-UTRA); Base Station (BS) radio transmission and reception, 3GPP, 2015.
[3] Common Public Radio Interface, CPRI Specification V6.0, 2013.
[4] S. Orfanidis, Optimum Signal Processing. An Introduction. 2nd Edition, Englewood Cliffs, New Jersey: Prentice-Hall, 1996.

About the author

I joined the company – then Nokia Siemens Networks – in July 2010, just after graduating from Wrocław University of Technology. Initially I worked at Hardware Baseband department as an integration engineer. In 2013 I moved to RF Software department, as a member of DAPD team which is responsible for implementation of various DSP algorithms for radio modules, including digital adaptive predistortion.

Wojciech Zmyślony
Engineer, Software Development
MN RF & AA

Optimization of LO Leakage Spurious Emission in RF Modules

Łukasz Małek
Engineer, Software Development
MN RF & AA

Mariusz Rusiniak
Engineer, Software Integration
MN RF & AA

NOKIA

1. Introduction

In a typical radio frequency device a transmitter path is divided into several processing paths. Most of the signal processing is done in digital domain, in the frequency range centered at zero frequency. Such frequency range is called the baseband. It cannot be directly transmitted to the air and therefore needs further shift to required operating frequency. To do so, radio frequency (RF) transmission systems typically use various approaches which include, for example, direct upconversion or digital intermediate frequency (IF) architectures.

In some situations local oscillator signal can leak to the RF output. This leakage can be observed on a spectrum analyzer as a large spike located at LO frequency. Such leakage might result in violation of technical specifications for mobile network communication systems [1]. The 3GPP requirements impose that transmitter must not generate illegal inband spurious emissions above the specified level. Typically, RF band in mobile networks is shared among numerous network operators. Spurious emission in such case might fall into other operator band, and result in a decrease of its performance.. This article describes one of the methods used to minimize local oscillator (LO) leakage problem in modern radio frequency systems which use digital-IF transmitter architecture.

Moreover, such unwanted signal might, under some circumstances, be introduced into its own transmission band. This may result in a deformation of the modulated signal, especially in multi-carrier wideband systems which use direct up conversion [2]. The same can happen to the receiving band. Last but not least, LO leakage may increase power consumption and thus lower the device power efficiency. Therefore, such unwanted signal needs to be removed from the transmitted frequency spectrum to provide more reliable service and increase the power efficiency in the final product.

In majority of RF devices which use digital-IF transmitter LO leakage effect is minimized by filtering out artifacts using analog filters. Situation gets more complicated in wideband and multi-carrier devices, where LO frequency is not fixed and can be tunable in a wide range. Finding the alternative solution requires understanding of the principles of RF transmission systems as well as mechanisms behind LO leakage phenomena.

2. LO leakage in digital-IF transmitter

Figure 1 shows block architecture of digital-IF transmitter, which is widely used in RF systems.

Digital-IF up-conversion architecture in general consists of digital-upconverter (DUC) which shifts the baseband signal to digital IF (intermediate frequency), and digital-to-analog (DAC) converter which transforms signal from digital to analog domain. Such a signal is filtered in Band Pass Filter (BPF) and upconverted using radio frequency LO mixer (RF LO). The last step is the amplification of the final signal by a power amplifier (PA). After each processing step some filtering is done to remove signal artifacts. In an ideal system PA output signal should contain only the baseband signal, which is amplified and shifted in frequency domain to the desired bandwidth. Due to nonlinearities and imperfections of used devices as well as different noise effects, the output signal is usually distorted. The signal behind PA is decoupled and it returns to digital domain via a feedback line. The feedback signal is first downconverted using nulling LO mixer (NULLING LO). Then the signal is transformed to a digital domain in an analog-to-digital (ADC) converter. Finally, digital-downconvertion (DDC) shifts signal to baseband.

Among the effects which might result in the output signal distortion one can find LO leakage. Taking into account that real mixer has finite isolation between the LO input and the output, LO leakage will be visible at the mixer output [4]. Additionally, direct current (DC) offset is often present at the output of DAC which, further mixed with LO signal, adds up to the leakage level at the mixer output. **Figure 2** presents how LO leakage is generated in digital-IF transmitter in a single carrier case.

Figure 1 Block diagram of digital-IF transmitter. Based on [3]

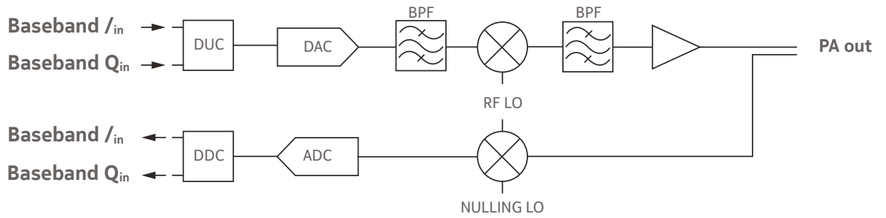
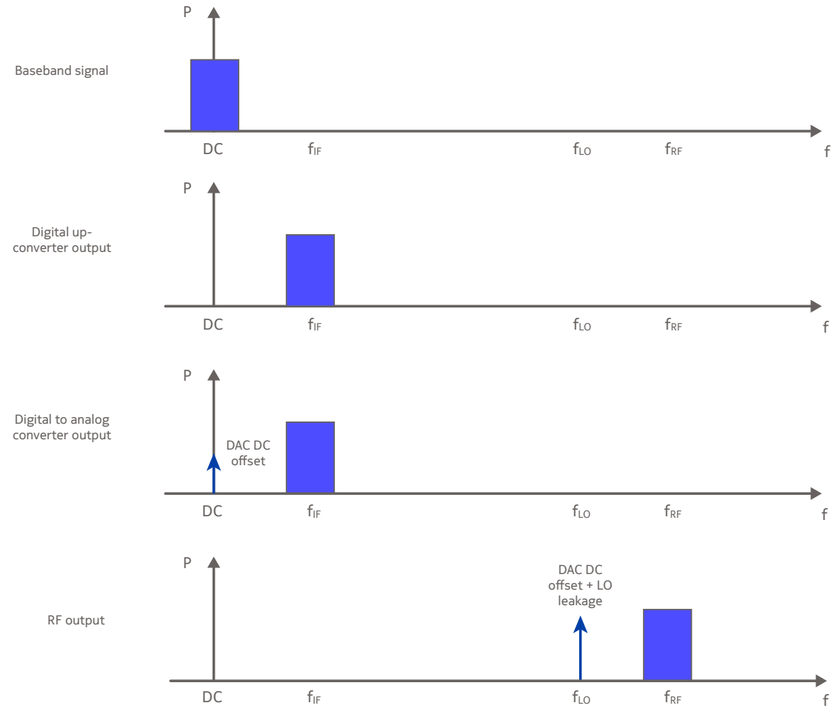


Figure 2 LO leakage generation mechanism.



Pure baseband signal is shifted in digital domain to IF by digital up-converter block. The signal is further converted to analog domain. In theory, at this point the signal should not have any constant component. Unfortunately, in a real system there is some DC offset visible at the output of DAC. It is introduced due to mismatches and imperfections of used baseband components [2].

In the next step, when IF signal is mixed with LO both useful signal and DAC DC offset (if any) are shifted to the final radio frequency. As a result of mixer's finite isolation between the mixer input on LO side and the mixer output, additional LO leakage signal is added to shifted DAC DC offset. This produces a spurious frequency emission which can be observed in signal spectrum relatively close to the carrier. Spurious LO signal is then amplified by PA and requires additional actions to prevent it from reaching the antenna line. Elimination process of LO spurious emission is commonly named as LO nulling. If LO frequency is not fixed for a given product and can be widely tunable, such LO leakage cannot be easily filtered out by the usage of conventional analog filters. Moreover, such filter requires relatively high level of attenuation in the blocking band, and that makes the cost and complexity of such solution not acceptable.

3. LO leakage cancelation

Although there are different methods used widely in the industry to overcome the LO leakage, most of them require complex hardware design. In **Figure 2** spurious emission at the RF output is the sum of two signals,namely LO leakage introduced by non-ideal mixer and DC offset.

3.1. Passive LO nulling systems

The simplest approach to eliminate the LO spurious emission in RF systems is the usage of analog filter. However, this solution limits useful LO frequency range and thus limits frequency range in which carrier can be located. This is a significant disadvantage because it makes the product less universal. Additionally, such solution is not cost effective.

3.2. Active LO nulling systems

Theoretically, if the system could influence DC signal level in a way that it would be at the same level but opposite due to the LO leakage signal, then we would be able to achieve a leakage cancellation. For this purpose Look-Think-Act approach can be applied. In this method dedicated subsystem is responsible for introduction of additional signal to transmission path to minimize LO leakage level. At first we will go through Look and Act activities because they are related to hardware capabilities. Later on we will go to Think activity that is hardware independent.

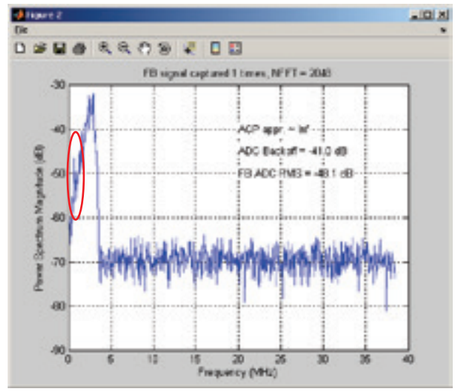
4. Look

To perform LO leakage level measure it is required to enable the observation of a signal after mixer. Commonly in RF devices, there is a dedicatd feedback line that is coupled to transmission path after power amplifier. This line can be used for LO leakage level measurements purpose (bottom part of **Figure 1**). Measurement of LO leakage is equivalent to measurement of signal amplitude at f_{LO} frequency. The easiest way to do this is to shift signal by $-f_{LO}$ what will move LO leakage to DC. This will boil down the problem of LO leakage measurement to calculation of mean from feedback signal amplitude. It can be done either by dedicated hardware component or by software calculation from captured signal samples. In order to eliminate unwanted DC component on ADC, frequency shift is separated into two steps. The first frequency shift is performed in analog domain and the second is done in digital domain. Another problem that might appear in practical systems is caused by aliasing and it makes LO leakage to be covered by carrier. In such case additional filtering in analog domain is required before leakage power can be measured. The outcome of such filtering is presented in **Figure 3**.

5. Act

Beside the possibility of measuring LO leakage it is essential to be able to influence its level. DC signal might be modified either in digital or analog domain. In digital domain typical DAC [5] devices offer a possibility to add digital offsets to the signal before actual digital

Figure 3 Feedback sample spectrum. LO leakage spike selected by a circle.



to analog conversion. Such approach is limited by DAC resolution. In analog domain additional DC voltage offset might be added in the RF mixer. In typical application signal mixers [6] have a built in current source which allows the control of the DC offset. In this case range and resolution is limited by hardware.

6. Think

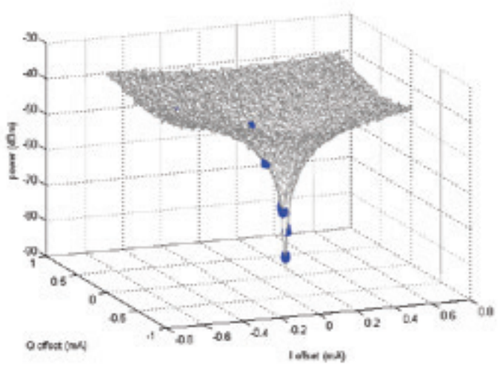
LO nulling problem is a typical optimization task where a goal is to minimize LO leakage by controlling signal offset levels. The wide range of optimization techniques can be applied for this task, however, practical limitation enforces additional requirements.

- The necessity of measuring feedback signal in specific configuration might exclude other crucial functionalities like gain control or digital adaptive pre-distortion (DAPD).
- The capture of signal samples and performing calculation on them is a time consuming task therefore, algorithm should be based on a small number of measurements.
- In order to provide reliable results also for small leakage levels that are comparable to noise level, algorithm needs to be robust against noise in input data.
- Due to nonlinear dependency between DC offset value and LO leakage level, algorithm needs to be applicable for nonlinear optimization.
- Algorithm has to be applicable for two-dimensional optimization because of two offset parameters.

7. Algorithms

The most common approaches for optimization of nonlinear systems are iterative methods. The classic solution is the application of gradient-based methods like Newton method, gradient descent or conjugate gradient [7]. Two-dimensional gradient calculation requires several additional measurements, what is not acceptable due to previously mentioned requirements. Therefore, for LO nulling optimization non-gradient based iterative algorithm shall be considered. In Nokia we use custom iterative algorithm, which is based on criteria function properties. Algorithm iteration steps are marked on **Figure 4** with blue dots.

Figure 4 Relation between LO leakage level and DC offsets



Finding optimal DC offsets can be also done after the identification of the model parameters which are describing relation between offsets and LO leakage level. This relation can be described with the following formula:

$$P_{LO}(I,Q)=10\log\sqrt{(I-I_o)^2+(Q-Q_o)^2+\Delta^2},$$

where P_{LO} – power of the LO leakage, I_0, Q_0 – optimal offset, I, Q – current offset, Δ – negligible constant that arises from the physical properties of the system. The form of the model allows application of generalized least square method to find unknown model parameters [7]. This approach requires at least 5 measurements to obtain estimation of system parameters. However, in practical application a number of measurements needs to be higher, to eliminate the impact of noise in input data. Finally, the most critical problem for this approach is the selection of measurement points that will guarantee proper conditions for identification – suboptimal selection might cause numerical issues.

One can also consider heuristics as an alternative approaches to optimization problems such as genetic algorithm, swarms or stochastic search [8]. The majority of available heuristic algorithms requires a large number of measurements, and that makes them not applicable to this particular problem.

8. Conclusions

LO leakage cancelation is an important practical issue in radio transmission devices from performance perspective and 3GPP requirements. The elimination of this spurious emission requires understanding of its origin and adjustment of hardware design to support planned method of leakage cancelation. Active LO nulling cancelation approach provides large flexibility in selection of used algorithm, but practical limitation reduces a number of possible optimization methods. Iterative algorithm used in Nokia devices provides both fast and reliable results.

References

[1] "3GPP TS 25.104; Base Station (BS) radio transmission and reception (FDD)".

[2] C. Lanschützer, A. Springer, L. Maurer, Z. Boos and R. Weigel, "Integrated Adaptive LO Leakage Cancellation for W-CDMA Direct Upconversion Transmitters," *IEEE Radio Frequency Integrated Circuits Symposium*, 2003.

[3] A. Behzad, Wireless LAN Radios: System Definition to Transistor Design, Piscataway: Wiley-IEEE Press, 2007.

[4] M. Golio and J. Golio, The RF and Microwave Handbook, Boca Raton: CRC Press, 2008.

[5] *AD9142A Dual, 16-Bit, 1600 MSPS, TxDAC+ Digital-to-Analog Converter*, Analog Devices, 2014.

[6] *TRF372017 Integrated IQ Modulator PLL/VCO (Rev. E)*, Texas Instruments, 2016.

[7] G. C. Goodwin and R. L. Payne, Dynamic System Identifications: Experiments Design and Data Analysis, New York: Academic Press, 1997.

[8] R. K. Arora, Optimization: Algorithms and Applications, Boca Raton: CRC Press, 2015.

About the authors

I have a PhD in Control Automation and Robotics and work as a Software Development Engineer in Radio Frequency Software department. I am responsible for development of algorithms related to control of RF signal quality in Nokia products. My main research activity is focused on linearization of nonlinear systems.

Łukasz Małek
Engineer, Software Development
MN RF & AA

I am a graduate of Faculty of Microsystem Electronics and Photonics at Wrocław University of Technology. My journey in Nokia began in 2010. Currently I work as an Embedded Software Integration Engineer in MBB RF RFSW. In my department we are responsible for the whole radio module software development life-cycle for various Nokia products. The work itself is very challenging as it touches a very wide variety of problems, architectures and technologies which makes it a great opportunity for constant self-development as well.

Mariusz Rusiniak
Engineer, Software Integration
MN RF & AA

Software Based Detection of Antenna Connection Issues

Karol Sydor
Engineer, Software Development
MN RF & AA

NOKIA

1. Introduction

Nowadays Cellular Base Stations are autonomous units which should operate as long as possible without any need of attention. Self-diagnostic and problem reporting are ones of the most demanded features of a standalone Base Station, reducing the need of additional maintenance, hence lowering the overall system cost.

One of the places where monitoring is especially important is the radio module, responsible for the transmission and reception of the radio signal from the antenna line. The connection between the radio transmitter and the antenna is constantly monitored by the radio module and in case of a broken link system can provide additional information about the approximate location of the issue.

2. Reflected power

Figure 1 Sources of a power reflection in a typical antenna connection

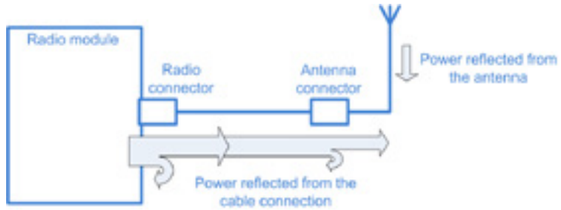


Figure 1 presents a typical antenna connection scheme. Connection of the radio module to the physical antenna commonly requires several cables and adapters. The radio waves are routed from the transmitter output up to the antenna through a transmission line. At every junction there is a mechanical mismatch which size determines the amount of the consequent impedance mismatch. This causes a portion of the radio waves to be reflected back down the transmission line. A part of the signal is reflected from the antenna elements as well. Those signal's levels can be measured; the results, together with its relation to the power of the transmitted signal, is called the return loss.

Return loss is one of the most important parameters of a radio system. The reflected power directly reduces the antenna efficiency, thus decreases the system performance. Another problem is that a portion of the reflected waves arriving to the transmitter output will be reflected back to the antenna, traveling with the original signal. As a result, the additional signal is delayed and acts like an echo of the original signal, causing deterioration of the signal quality and increase of the error rate during decoding phase on the receiver side.

Antenna connectors and cables are often exposed to extreme environment conditions and it is assumed that their performance will decrease over time. It is important to monitor the return loss of each antenna and to detect the when an additional service is required. It is also important to react in abnormal situations, for example connection breaks. When antenna is disconnected from the radio module, almost all of the transmitted power is reflected to the transmitter which can lead to severe hardware damage. The radio module front-end is capable of withstanding such conditions for a limited time, so it is critical to detect those situations properly and start recovery actions.

3. Used criteria

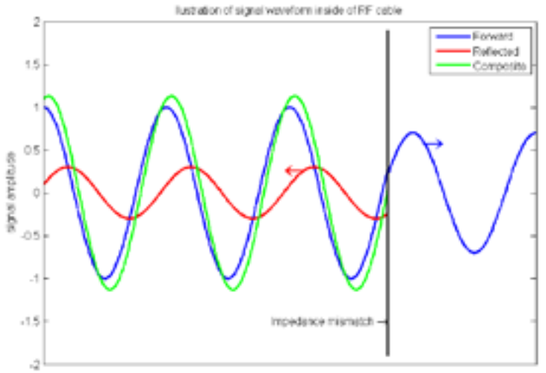
The Return Loss describes the difference between the incident power at the output of the transmitter and the reflected power:

$$RL\,[dB] = P_{incident}\,[dBm] - P_{reflected}\,[dBm]$$

When perfect impedance match is present in the whole antenna line path, the Return Loss is equal to the output power.

An alternate measurement criterion is the Voltage Standing Wave Ratio (VSWR). In a transmission line a standing wave is created by the interference of the original forward signal, and a wave reflected by an impedance mismatch. **Figure 2** illustrates this process.

Figure 2 Standing wave as a result of reflection



VSWR is defined as the ratio of the absolute values of maximum and minimum amplitudes of the signal voltage:

$$VSWR = \frac{|V_{max}|}{|V_{min}|} = \frac{1+|\Gamma|}{1-|\Gamma|}$$

where Γ is the reflection coefficient defined as ratio of the reflected and forward waves:

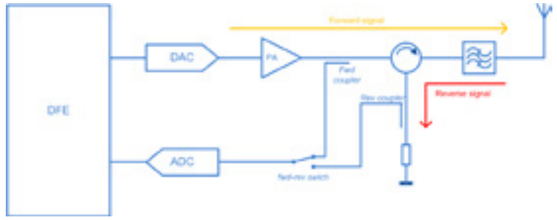
$$\Gamma = \frac{V_{reflected}}{V_{forward}}$$

VSWR value falls within the range of [1,∞]. Return loss can be calculated from VSWR value and vice-versa:

$$RL = -20\log\left[\frac{VSWR-1}{VSWR+1}\right]$$

In practice, both values are in use, depending on the used algorithm and interface specification.

Figure 3 Simplified diagram of the analog radio path



4. Hardware requirements

Figure 3 illustrates a simplified diagram of an analog radio module path. The DFE (Digital FrontEnd) chip generates the digital signal which is converted to an analog wave by the DAC. Then the signal is amplified by the PA, and filtered by the frontend filter. After that, the forward wave leaves the radio module and is fed to the antenna. To give the system the ability to measure the signal, a dedicated receiver path is used, with the ability to connect with different points on the path. The first directional coupler provides the feedback information for all algorithms which require information about amplified forward signal. Those algorithms are responsible for the PA Gain control, and power measurements. The second directional coupler provides the reverse signal information. This coupler is located after the frontend filter and the circulator. Typical algorithms use forward signal taken from the reverse directional coupler, and the forward digital signal captured within the DFE chip. Signal captured on the forward directional coupler can also be used, but it requires additional filtering and processing. In cases where the accuracy is not crucial it is easier to capture the signal in the digital part of the transmitter and apply gain correction, based on the information from PA gain control algorithms. For this reason the feedback path can be simplified and only one converting circuit can be used, together with an analog switch changing the connected coupler.

5. Monitoring

During regular operation the radio module should periodically measure the return loss to detect issues appearing randomly when the antenna connection is broken or disconnected. The primary role of this functionality is to provide rough measurement in constant periods to make sure that in case of a failure the radio hardware can be safely shut down. To fulfill those requirements, simple power meters are used to measure the power of a signal in the digital domain. Software calculates the incident and reflected power, and the resulting return loss. The measurement is done inside the transmitter, where presence of the additional components in the signal path impacts the reflected power.

The accuracy of this measurement is limited due to the presence of the noise in the power measuring circuit, and the used methodology but it is sufficient to ensure that severe degradation will be detected. The accuracy is not the most important factor in this case; once an anomaly is observed, a more precise measurement can be performed to acquire more detailed information.

6. Precise measurement

Once an abnormal situation on the antenna connection is detected, a more precise measurement is performed. This measurement requires more processing time, but gives improved accuracy and it is used to confirm the presence of an issue. Once confirmed, software starts recovery actions to preventhardware damage, and informs the rest of the core network system about the fault.

The transmitted signal is captured together with the reflected signal taken from the directional coupler. Capturing is precisely synchronized to ensure that exactly the same portion of the signal is processed. When both the forward and reversed signals are acquired, the data is ready for further software processing.

Since the reverse signal is captured after passing through the transmitter frontend filter, it is additionally distorted by its characteristic. In addition the circulator leakage has to be taken into account. During processing software is able to correct these distortions, using factory calibrated transmitter path characteristics, and the forward signal as a reference. The calibrated characteristic of the unwanted distortion is scaled and removed from the reverse signal. This procedure allows estimating the reverse signal which is present at the antenna connector.

Additional improvement is achieved by using Fast Fourier Transform (FFT). The signal is transformed into frequency domain, where emissions outside of the used bandwidth are removed. Afterwards the signal is transformed back with the use of inversed FFT. With those improvements the forward and reverse power can be calculated with much better accuracy which is enough to make sure that an abnormal situation is present.

7. Distance to fault estimation

The antenna cabling can be as long as 100m. Estimating the distance to a fault can greatly improve the maintenance procedure, by providing approximate information as to where the damage could be located. The correct determination of the reflection location requires an estimation of the delay between the incident and the reflected signal. In this algorithm, signal samples filtered and corrected during the precise measurement procedure are taken into use.

The delay between incident and reflected signal consists of the delay introduced by different location of capture points, the group delay of the frontend filter, and distance to the reflection point.

The place of the reflection can be estimated using cross correlation calculation. This is an iterative process; during each iteration the phase of the forward samples is rotated by a small angle, and the signal is correlated with the reverse samples. Every result of such correlation is stored as an element of a vector. Since the phase rotation corresponds to the delay, the resulting vector can be considered as a function, where the X values are the added delay, and the Y values indicate the correlation of the forward samples to the reverse samples. The maximum value of this vector indicates how much the reversed signal is delayed compared to the forward signal.

The calibration values and runtime information about the carrier frequency are used to eliminate the impact of additional delays. The result can be directly converted into distance to the reflection source.

8. Conclusion

The Return Loss monitoring systems are included in most of the currently available Nokia radio modules. By addition of several external components and dedicated software algorithms, the need of additional maintenance is greatly reduced. The operator is informed every time an abnormal situation on the antenna connection is observed. The information about antenna connection quality can be used to plan maintenance actions in advance, and additional information about distance to fault simplifies the service procedures and shortens the service time.

References

[1] VSWR Supervision, Flexi BTS Multiradio, Armin Splett, Jan Hellman, Nokia 2012
[2] Return Loss and VSWR, Back to Basics in Microwave Systems, Derren Oliver, http://www.commscope.com
[3] Standing Waves and Resonance, All About Circuits, EETech Media, 2016

About the author

I work as a Software Development Engineer in MBB RF RFSW team developing software for Nokia radio modules. In our daily work we are challenged by encountering analog radio elements and high speed digital processing. Constant evolution of the systems and searching for better solutions are the driving forces behind our professional development. In our work technology is not only a tool, but also a goal.

Karol Sydor
Engineer, Software Development
MN RF & AA

Professional Software Development

NOKIA

Java Concurrency

Wojciech Konopka
Engineer, Software Development
A&A NM & SON

Creating threads in the Java environment is easier than you might think at first. It comes down to proper building and managing Thread class together with interface Runnable. You can approach creating a new thread in two ways – either by implementing interface Runnable or expanding Thread class.

While reading the first lines of this article, it is almost irresistible to ask yourself – "Which one is the best solution while programming?"

There is no straightforward answer to this question. Each of the methods is dependent on a number of factors. When making a final decision, for example, if you want to use inheritance to extend the Thread class, then bear in mind that it is a method that can be only applied to a current thread.

Let's take a closer look at the first example presented in **Listing 1**.

Listing 1 Code fragment

```
public class DemoMainThread {
   public static void main(String[] args) {
       Thread mainThread = Thread.currentThread();
       System.out.println("Main thread : " + mainThread);

       mainThread.setName("main thread");
       mainThread.setPriority(Thread.MAX_PRIORITY);

       System.out.println("Main Thread : " + mainThread);
   }
}
```

The outcome of **Listing 1** is as follows:

```
Main thread : Thread[main,5,main]
Main Thread : Thread[main thread,10,main]
```

Before we go into more details about multithreading in Java, let's explain a term – **main thread**. It is created automatically, and it constructs its child threads. It usually finishes operation as the last one, and it also can be responsible for the cleaning system, for example, closing internet connections.

In the provided example, we obtain a reference to the main thread by means of a static method *currentThread* aggregated to the Thread class. Once the reference to it is obtained, then you can change from its default name "main" to "main thread". What is more, you can also change, in a given system, the thread's priority from a default one to a *MAX_PRIORITY* one.

You can easily notice that the applied changes are visible in the second line output and mean: *thread name*, *priority* as well as thread group name. The default value of the *thread group name* for the main thread is "*main*" and you can manage the thread group with it.

Changing priority has been done in the provided example by means of the *setPriority* method for a given thread. The priority is an integer value ranging from MIN to MAX defined in the Thread class. Threads with higher priority are more likely to get a processor's cycle after finishing with the current thread.

Context switch – changing across from the current thread to a different one – it happens when a processor's timeslot is given away intentionally or is simply preempted. Multitasking mechanism with preemption takes place when a scheduling algorithm suspends the currently running thread with a lower priority and invokes executing a thread with a higher priority.

In the actual system, allocating processor's timeslot to a given thread mostly depends on how the multitasking has been implemented in the operating system, not necessarily on the priority itself. You need to bear in mind that you can start your application in an environment deprived of any preemptive mechanism. This implies that you should always follow the best engineering practices and design your code in such a way to release the processor's resources to other threads from time to time.

1. Runnable Interface

Listing 2 Code fragment

```
public class SimpleThread implements Runnable {
   Thread thread;

   SimpleThread(String name) {
       thread = new Thread(this, name);
       thread.start();
   }

   @Override
   public void run() {
       try {
           for (int i = 0; i < 3; i++) {
               System.out.println("Thread name :
               " + thread.getName() + " index : " + i);
               Thread.sleep(1000);
           }
       } catch (InterruptedException e) {
           e.printStackTrace();
       }
   }
```

```
    public static void main(String[] args) {
        new SimpleThread("a");
        new SimpleThread("b");
        new SimpleThread("c");
    }
}
```

Listing 2 provides an example of how to create a thread by means of class implementing the Runnable Interface. It contains one public method – *run* in which there is a piece of code containing a thread's logic. The thread lifecycle is defined in the body of *run* method – which means that it starts and ends in it. This is an ordinary method and like in any other we can create objects and refer to other lines or other classes' methods.

The thread is created in a constructor of the *SimpleThread* class by means of triggering:

```
thread = new Thread(this, name);
```

You forward a class implementing Runnable Interface and thread name to the constructor of the Thread class. Once the object is created, you need to run the thread by the start method from the Thread class. As you can easily notice, the *start* method invokes the *run* method. Forwarding *this* to the method causes the *run* method to be run in the current object.

Starting thread is as follows:

```
thread.start();
```

In the *run* method there is a loop which provides a current thread name together with current index. Next, the thread is put to sleep for 1000 milliseconds. The method is invoked in the *try-catch* block because the *sleep* method can throw the *InterruptedException* exception. There are three threads created in the system, which run simultaneously and their results can be as follows:

```
Thread name : a index : 0
Thread name : b index : 0
Thread name : c index : 0
Thread name : a index : 1
Thread name : b index : 1
Thread name : c index : 1
Thread name : a index : 2
Thread name : b index : 2
Thread name : c index : 2
```

Bear in mind that the outcome is dependent on the system load and their values can be different at every run.

2. Synchronization

Synchronization mechanism is required when many threads modify or read shared resources. Java programming language has been designed to have synchronization in place to keep data consistent.

Let's take a look at an example without such mechanism and what problems it might generate:

Listing 3 Code fragment

```
class Hello {
    public void sayHello(String name) {
        System.out.print("Hello ");
        System.out.println(name);
    }
}

public class HelloThread extends Thread {
    Hello hello;
    String name;

    HelloThread(Hello hello, String name) {
        this.name = name;
        this.hello = hello;
        Thread thread = new Thread(this);
        thread.start();
    }

    @Override
    public void run() {
        hello.sayHello(name);
    }

    public static void main(String[] args) {
        Hello hello = new Hello();

        new HelloThread(hello, "Camilla :-)");
        new HelloThread(hello, "Alice :-)");
        new HelloThread(hello, "Bob :-)");
        new HelloThread(hello, "John :-)");
    }
}
```

The way the program works is pretty straightforward. It simply displays "*Hello*" message for a given sequence of characters, for example "*Hello Camilla ;-)*". It consists of two classes: *Hello* and *HelloThread*. In the first class there is only one method – sayHello, which has the task to display, in this case on a computer screen, a provided welcome message. In the light of multitasking, the whole text has been divided into two invocations of printing methods. The *Hello* object together with a sequence of characters is passed to the *HelloThread* constructor. They are assigned to fields in a class and then a thread is created in the constructor, which is run at once. In the *run* method there is *sayHello* method invoked from the *Hello* class on the object forwarded to the constructor. Invoking the main method creates three thread objects, which automatically triggers three threads operating simultaneously.

Let's take a look at the possible outcome of the running application. Are these values what we expected?

```
Hello Hello Hello Hello Bob :-)
Alice :-)
John :-)
Camilla :-)
```

As you can see in the above example, Bob has been welcomed four times, whereas other people, not even once. Is this how it was meant to be? You might have thought that invoking the *sayHello* method many times should display character strings in the correct order. Unfortunately, when it comes to multitasking programming, such a situation looks differently.

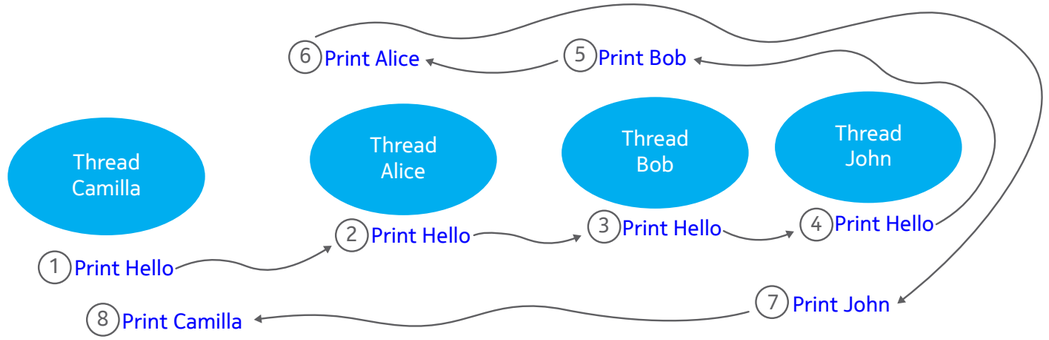
Let's pause for a moment and think how could the application run and what actually happened.

Four threads which simultaneously share the same object of the *Hello* class have been run. To simplify a bit, let's tag the invoked threads with names which have been used in the constructors of the *HelloThread* class.

To analyze the example, first let us begin with a simple scenario. The threads will run in an expected sequence – it is simply to understand the provided example. Of course, in a real system, threads do not have to start in the order of their instantiation. To understand it better, let's take a look at **Figure 1** and what is happening in the application. After starting, the *Camila* thread has already invoked the first method displaying the *Hello* sequence and at the same time it is preempted by another thread. The processor's time is allocated to another thread and the other method printing name of *Camila* is not invoked at that moment. The same situation happens with other threads such as *Alice*, *Bob* and *John*. The other method displaying a given name is not invoked. The thread manager resumes the *Bob* thread and the display screen shows "*Bob :-)*" – the thread finishes because it is the last instruction in the *sayHello* method and at the same time, invoking in the *run* method stops. The mechanism is repeated for other threads such as: *Alice*, *John* and *Camila*. The application finishes together with the last thread.

For the sake of better understanding the issue, let's modify the *Hello* class. The *sleep* method, which puts to sleep the thread for 1 second, has been added between two methods displaying the text. In this situation, managing has been intentionally given to other threads. The outcome will be similar to the previous one.

Figure 1 Program flow



Listing 4 Code fragment

```
class Hello {
   public void sayHello(String name) {
       System.out.print("Hello ");
       try {
           Thread.sleep(1000);
       } catch (InterruptedException e) {
           e.printStackTrace();
       }

       System.out.println(name);
   }
}
```

Synchronization in Java has been applied by means of a keyword – *synchronized*. Thanks to this you can create a piece of code – called critical section – which can be executed just by one thread at a time. You can synchronize methods or blocks of code. Invoking the synchronized section causes entering to Monitor – an object, which imposes a blockade that is also known as mutex. This translates into the fact that if a thread has triggered a method or a synchronized block for a given object, then the other objects need to wait until the first thread finishes its action and until that time, they cannot trigger the synchronized section of the object.

In the provided example, the four triggered threads participated in the so called "*chase*". To prevent code from performing actions by more than two threads simultaneously, you need to declare the *sayHello* method with the keyword *synchronized*.

```
class Hello {
   public synchronized void sayHello(String name) {
       System.out.print("Hello ");
       System.out.println(name);
   }
}
```

The outcome after applying the said changes is as follows:

```
Hello Camilla :-)
Hello Alice :-)
Hello Bob :-)
Hello John :-)
```

As it has been already mentioned, Java language supports synchronization a block of code and it looks as follows:

```
synchronized (object) {
// synchronized code
}
```

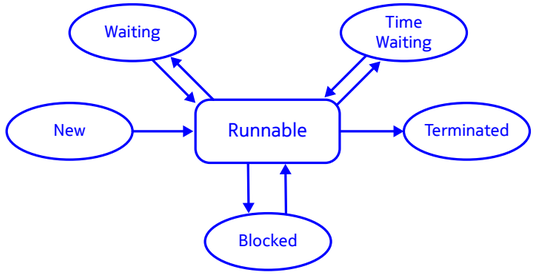
The block of code has been preceded with the keyword *synchronized* and there is a reference to a synchronized object in the curly brackets.

The example has been modified to use the synchronized block of code.

Listing 5 Code fragment

```
class Hello {
   public void sayHello(String name) {
       System.out.print("Hello ");
       System.out.println(name);
   }
}

public class HelloThread extends Thread {
   Hello hello;
   String name;

   HelloThread(Hello hello, String name) {
       this.name = name;
       this.hello = hello;
       Thread thread = new Thread(this);
       thread.start();
   }

   @Override
   public void run() {
       synchronized (hello) {
           hello.sayHello(name);
       }
   }

   public static void main(String[] args) {
       Hello hello = new Hello();

       new HelloThread(hello, "Camilla :-)");
       new HelloThread(hello, "Alice :-)");
       new HelloThread(hello, "Bob :-)");
       new HelloThread(hello, "John :-)");
   }
}
```


Figure 2 Thread states



3. Thread state

Thread can have the following states:

- New – a thread has not been started yet
- Terminated – a thread has finished
- Runnable – a thread is currently being run or is waiting for the operating system resources, for example, access to processor
- Waiting – a thread has been suspended
- Timed Waiting – a thread has been suspended for a while
- Blocked – a thread has suspended action and is waiting for access to monitor

Figure 2 presents thread states with their proper relations. Some of the transitions are on direction only, for example, a thread cannot change state from Terminated to Runnable. To check the current state of a thread, run the *getState* method of *Thread class*.

Listing 6 *getState* method example

```
public class DemoThreadState {
   public static void main(String[] args) {
       Thread thread = Thread.currentThread();
       Thread.State state = thread.getState();

       System.out.println(state);
   }
}
```

The application most likely prints the string of characters: RUNNABLE. You should not use this method to synchronize threads as the thread state can change just after running *getState* and the returned value will not provide the actual thread state.

4. Thread communication

Managing by means of communication between threads is one example of good programming practices. Let's take a closer look at a common producer-consumer problem.

Listing 7 Code example

```
class Bread {
   int counter;

   Bread(int counter) {
       this.counter = counter;
   }

   public int getCounter() {
       return counter;
   }
}

class Oven {
   int counter;
   Bread bread;
   boolean isReady;

   synchronized Bread get() {
       if (!isReady) {
           try {
               wait();
           } catch (InterruptedException e) {
               e.printStackTrace();
           }
       }
       System.out.println("got : " + bread.getCounter());
       isReady = false;
       notify();
       return bread;
   }

   synchronized void put(Bread bread) {
       if (isReady) {
           try {
               wait();
           } catch (InterruptedException e) {
               e.printStackTrace();
           }
       }
       this.bread = bread;
       isReady = true;
       System.out.println("put : " + bread.getCounter());
       notify();
   }
```

```
}

class Baker implements Runnable {
   Oven oven;

   Baker(Oven oven) {
       this.oven = oven;
       new Thread(this).start();
   }


   @Override
   public void run() {
       int counter = 0;

       while (true) {
           oven.put(new Bread(counter++));
       }
   }
}

class Client implements Runnable {
   Oven oven;

   Client(Oven oven) {
       this.oven = oven;
       new Thread(this).start();
   }

   @Override
   public void run() {
       while (true) {
           oven.get();
       }
   }
}

public class Runner {

   public static void main(String[] args) {
       Oven oven = new Oven();
       new Baker(oven);
       new Client(oven);
   }
}
```

The above-mentioned producer-consumer problem lies in the fact that two threads, namely Producer and Consumer operate simultaneously on the same buffer. In the provided example, the Producer is an object of *Baker* class, whereas a Consumer is *Client* and the buffer is *Oven*.

The goal of the Producer is to produce data – objects of the *Bread* type and place them in buffer so that a Consumer could easily fetch them. The core of the problem lies in providing a proper synchronization so that a Consumer will not obtain twice the same data or even empty data, and what is more, the Producer does not add data to a full buffer. Methods *wait* and *notify* of Object class, which can be only invoked in synchronized blocks, will be applied to solve the problem. The *wait* method puts to sleep the current thread until the other thread invokes the *notify* method for the object itself.

The *wait* method is invoked in the *Oven* class in the *get* method to suspend action until the Producer places data in the buffer. The Producer informs the Consumer by means of the *notify* method and then the *get* method is resumed.

The *wait* method is invoked in *put method* to suspend action until the Consumer obtains data. The whole process is in an endless loop, therefore you need to stop the application manually.

Now let's take a look at the outcome to confirm that the producer-consumer problem is handled correctly.

The outcome is as follows:

```
put : 0
got : 0
put : 1
got : 1
put : 2
got : 2
put : 3
got : 3
```

5. Summary

A well-developed and implemented concurrency mechanism in applications translates into improving their efficiency and usability. This article has illustrated that multithreading in Java is quite simple and it is provided by Java language by default. The prescription for success is to change your mindset and start programming based on concurrency principles. This article is just a preface to this exhaustive topic of concurrency, which has a lot more to offer.

About the author

I work as a Java Software Developer in Nokia in Poland. My professional interests are focused on agile methodologies and good engineering practices. Privately I am an amateur radio operator with callsign SQ6WK.

Wojciech Konopka
Engineer, Software Development
A&A NM & SON

Professional Software Development

Building Microservices with Docker

Krzysztof Bulwiński
Engineer, Software Development
A&A NM & SON

NOKIA

Software industry often struggles to build systems in a way to plug components together in the easiest possible manner. Much in the same way, we can observe is a physical world. Firstly we have to decide what a component is. Our definition proposal is that a component is a unit of software that is independently replaceable and upgradeable [1].

Writing applications the traditional way, often leads to a situation in which a code base grows substantially. Mostly it is caused by adding new features. Developers nowadays are more and more accustomed to clean code policies. Introducing modules and defining strict interfaces between them improves the code quality significantly. However, in bigger projects there is a risk that the code implementing similar responsibilities starts to creep in to several components, causing code duplication. As a result, bug fixing and new development becomes harder.

Microservices-oriented development is a style of creating large scale applications consisting of small, self-contained, highly specialized and cooperative services. Each one concentrates on a particular job and is autonomous. It is highly desirable that communication between them is done through language independent and a lightweight Application Programming Interface (API). Such approach provides an architecture which is modular, reliable, easy to scale and deployable.

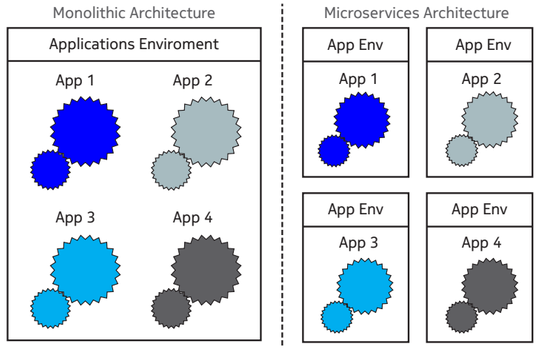
1. Homogeneity or heterogeneity – understanding architectural differences

To understand microservices, it is useful to begin with an explanation of monolithic architecture. A good example here are enterprise applications. Often consisting of database and application server tiers. In most cases, the application server plays a key role in the system. It handles requests and data flows from a user, acts as a middleware between the user and the database, executes domain logic, etc. As it is single logical executable unit, any changes require applications rebuilding and redeploying.

Monolithic applications have been operating successfully for many years. But today's demand to be prepared for changes, which are not suitable for them. A small system or application change often results in the entire monolith to be rebuilt and redeployed. Over time, it becomes harder to keep clean and modular structure. Moreover, demand on processing power has increased over recent years. The scalability of certain parts of the application can be hard to achieve in monolithic systems.

These design flaws lead to changes on how the applications are being built. The idea is to build them as a suite of small, independent and scalable services, with a firm boundary between them. This allows different units to be developed in different programming languages, for instance. And this is completely acceptable, because they communicate via language agnostic API. **Figure 1** briefly depicts the differences between the Monolithic and Microservices architectures.

Figure 1 Difference between the Monolithic and Microservice architecture



2. Key benefits of Microservices architecture

Microservices technology is becoming more and more popular. Let us take a closer look at some of the key benefits.

2.1. Technology diversity

Systems composed of multiple independently collaborating services gives a freedom to choose the right tool for the job. This is better than having a standardized, one-fits-all approach. Usually a particular technology suits one use case but not necessarily other.

Frequently, in typical enterprise systems, workload drives the demand to improve performance. But it does not always apply to the whole system, but only to its weak spots. At this point, current architecture may not be sufficient, thus forcing an administrator to switch to different technology stack. But big concern to try out new things is the risk associated with it. Changes such as programming language or used frameworks will impact bigger part of the system. With system build as a convolution of many small services, there is a plenty of room to try out a new pieces of technology. Starting new services alongside already running ones, integrating them to the system could provide an answer, if there is a real benefit of transition.

If a system is well designed, the risk of breaking things is reduced to absolute minimum. Obviously, there may be a need to adjust communication protocols.

2.2. Fault tolerance

Key concept in resiliency engineering is a system partitioning. This is especially important when one of the system's components is failing. In that case, erroneous part can be isolated, while the remaining services keep running uninterrupted. Of course this is possible, when the problem is not cascading. In contrary, in monolithic systems, when a service fails, there is a big chance that it impacts the whole system. Applying clustering and failure detection may reduce that risk. With microservices, in most cases, system does not fail entirely at once.

Microservices usually work in a network environment. It can and will fail sooner or later. That also applies to machines hosting microservices. This yields the need of understanding the nature of failures, and how they will impact the system as a whole.

2.3. Ease of scaling

Substantial workload in the system usually means the need for scaling. In monolithic systems, it almost always means to scale everything together. Often, a fraction of the system causes performance issues. If that little gear is tight up in a large monolith, sometimes we are obligated to scale everything. If those services were running as a small independent units, there would be only a need to scale those identified as poor performers. Remaining services are not even aware of any kind of changes and run almost intact. Technology nowadays gives a possibility to utilize on-demand scaling for those pieces of software that needs it.

2.4. Fast and easy deployment

In practice, deployments of a large monolithic system does not happen too often. The procedure usually is complicated, and understandable fear of the high risk that something may go wrong is discouraging. Therefore, customers are forced to work with a software that technologically is old-fashioned. Updates indeed happen, but mostly with the release of major versions. Unfortunately, they usually bring many changes accumulated with bug fixes and new features. Because the bigger the delta is, the higher the probability, that something may brake during upgrade.

With microservices, the changes can be applied to a single service, independently to the rest of the system. Of course, the problem can occur and upgrade may fail. But it can be identified and isolated really fast, and rollback procedure can be applied immediately. For a customer it means less fear for the changes. Thus, new features and bug fixes can be delivered more frequently and helps the customer to keep up with technology. Positive side effect is higher trust being built between a supplier and a customer.

2.5. Composability by services

Nowadays, the constantly changing trends drive the applications usage in a variety of ways on different devices. Therefore, the need for an architecture that can keep up is essential. Especially when industry moves away from thinking in terms of narrow channels to more composite concepts of customer engagement. Taking all that into account, this is one of the reasons why a service-oriented architecture gained a great interest in recent years.

Microservices come ahead of such expectations and open up opportunities for reusability. A mindset of system designers is focused on solving problems underlying in the customers' needs. Whenever requirements change, it is possible to organize things differently.

3. Docker containers technology at a glance

"Docker allows you to package an application with all of its dependencies into a standardized unit for software development.

Docker containers wrap up a piece of software in a complete file-system that contains everything it need to run: code, run-time, system tools, system libraries – anything you can install on a server. This guarantees that it will always run the same, regardless of the environment it is running on." [2].

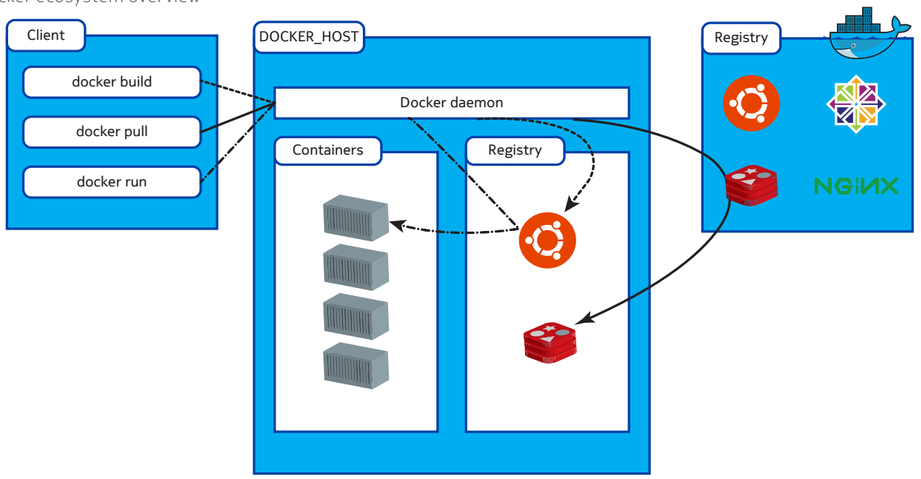
3.1. Docker ecosystem

Docker technology utilizes concept of client-server architecture. The client talks to the daemon which handles building, running and distribution of containers. The **Figure 2** depicts the Docker platform architecture.

For a better architecture understanding, few key terms should be briefly explained.

- **Docker daemon**, a process running on the host machine. An engineer does not interact directly with a daemon, but via Docker client usually shipped with an installation package.
- **Docker client**, a user interface that handles demands from the user and passes them to a daemon.
- **Docker image**, read-only templates used to create containers. The platform allows to create new, update existing or reuse own or community created images.
- **Docker registry**, a deposit for images which can be private or public. The public registry is called the Docker Hub. It is a huge collection of images prepared by contributors.
- **Docker container**, created from an image, holds everything that is needed for an application to run. Each is an isolated and secure application environment.

Figure 2 Docker ecosystem overview



https://docs.docker.com/engine/understanding-docker/

Existence of every image starts from a base image, for example Fedora base image. Own images can be used as a basis, as well. Images are build using a simple and descriptive set of instructions. They define actions like run a command, add a file or directory, create an environment variable, what process should be started when container is launched. Instruction set to build the image is stored in so called *Dockerfile*, a text base script file.

A container is a composition of an operating system, user-added files and metadata. When the container is run from the read-only image, a read-write layer is added on top of the image in which the application can run. In order to start the container, at least the image and a command to start must be provided.

4. Building microservices with Docker

When deploying Docker to a production it is good to consider a hosting environment beforehand. To fully utilize the distributed system features, containers should be deployed on an orchestration platform like Docker Swarm or Apache Mesos combined with Apache Marathon.

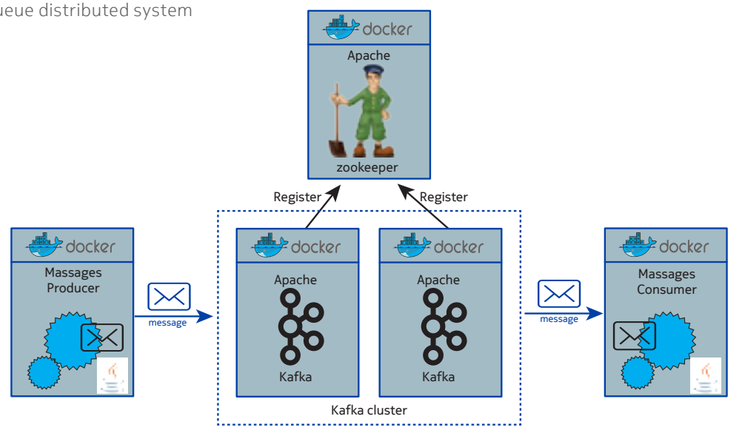
To simplify things, a local machine can do this as well. A distributed system presented in this article will consist of the containers each separately wrapping Apache Kafka, Apache Zookeeper, simple messages producer and consumer applications.

4.1. Exemplary distributed system overview

System which will be discussed in this article can be depicted in the **Figure 3**. It is a simple send and retrieve messages platform. It is advisable to have basic understanding of used Apache and Docker tools [3], [4], [5]. The reading covers only essentials needed to understand overall concept.

The way the system works is shown in the diagram. Message producer is a Java application producing messages in a time interval. Then they are being sent to the Kafka Cluster consisting of two physically independent brokers. These are the system core services. Generally speaking, Kafka is a distributed, publish-subscribe message queuing system. In the presented example, one acts as a leader, the other is the data replicator. Such configuration is useful, when one broker is down, then the other can take over the queuing responsibility. The common thing they share is the queue topic name to which the messages are being sent to and read from. Both instances are registered to the Zookeeper service. It acts as a centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services [3]. Message consumer is a simple Java application, responsible for consuming messages. To simplify things, there will be one consumer and one producer. More information can be found in [4].

Figure 3 Messaging queue distributed system



4.2. Dockerizing services

To run the whole system, appropriate Docker images must be prepared. Hub infrastructure provides ready-made images that can be re-used. The ones which do not exists must be prepared. Kafka and Zookeeper are available so it is possible to pull them. **Figure 3** presents downloading the images from the Hub with the **docker pull <image_name>** command.

In order to create producer and consumer the following **Dockerfile**s were prepared, **Listing 1** and **Listing 2**. The file is an instruction set for the Docker telling it how to build an image. The best reference of how to create containers in details is to read the user guide [5].

Listing 1 Kafka producer application container Dockerfile

```
FROM anapsix/alpine-java
RUN apk update
COPY kafka-producer-app.jar /opt/nokiabook/kafka-producer-app.jar
ENTRYPOINT [“java”]
CMD [“-jar”, “/opt/nokiabook/kafka-producer-app.jar”]
```

Listing 2 Kafka consumer application container Dockerfile

```
FROM anapsix/alpine-java
RUN apk update
COPY kafka-consumer-app.jar /opt/nokiabook/kafka-consumer-app.jar
ENTRYPOINT [“java”]
CMD [“-jar”, “/opt/nokiabook/kafka-consumer-app.jar”]
```

Figure 4 Pulling Docker images from the Hub

```
$ docker pull jplock/zookeeper
Using default tag: latest
Trying to pull repository docker.io/jplock/zookeeper ... latest: Pul
ling from jplock/zookeeper
a59bbaa9fb89: Pull complete
ddfc888505c8: Pull complete
                ⋮
b0febc9f50f3: Pull complete
abace57f3a36: Pull complete
Digest: sha256:e416e0a80ccb0e06d2d712f25bdf85627734c9c56f05b4974ef35
2d6b6ce3894
Status: Downloaded newer image for docker.io/jplock/zookeeper:latest

$ docker pull wurstmeister/kafka
Using default tag: latest
Trying to pull repository docker.io/wurstmeister/kafka ... latest: P
ulling from wurstmeister/kafka
ca26f34d4b27: Pull complete
d73951340db3: Pull complete
                ⋮
be3c7a9a095e: Pull complete
0059cb4bf620: Pull complete
Digest: sha256:d49dc5a509791c59ec235d3045b15ad67b153d1eb4d0a31b4e19e
40c7f3cea49
Status: Downloaded newer image for docker.io/wurstmeister/kafka:late
st

$ docker images
REPOSITORY                      TAG                 IMAGE ID
   CREATED             VIRTUAL SIZE
docker.io/wurstmeister/kafka    latest              0059cb4bf620
   16 hours ago        221.1 MB
docker.io/jplock/zookeeper      latest              abace57f3a36
   3 weeks ago         156.4 MB
```

Before building the images, the applications JAR file must be prepared. In the presented example, the archive is a fat JAR, that is containing the application and necessary libraries. Any method packing in such way will suffice. In the example, a **Maven** building tool was used with properly configured **pom.xml**. Because of that, the whole process of building the JAR as well as packing it to the Docker image comes down to single command execution.

Building fat JAR is not the only option. It is possible to build images with separate archives for the application and dependencies. That however requires preparing appropriate java classpath. To make things easier, all items were packed into a single package. It is up to the designer to decide which approach is the best.

The source code of the producer and the consumer applications is listed in **Listing 3** and **Listing 4**.

Listing 3 Kafka producer application source code

```
public class SampleProducerApp {

  public static void main(String[] args) {
    Properties props = new Properties();
    props.put(
            ProducerConfig.BOOTSTRAP_SERVERS_CONFIG,
            "172.17.0.3:9092");
    props.put(ProducerConfig.ACKS_CONFIG, "all");
    props.put(ProducerConfig.RETRIES_CONFIG, 0);
    props.put(ProducerConfig.BATCH_SIZE_CONFIG, 16384);
    props.put(ProducerConfig.LINGER_MS_CONFIG, 1);
    props.put(ProducerConfig.MAX_BLOCK_MS_CONFIG, 2000);
    props.put(ProducerConfig.BUFFER_MEMORY_CONFIG, 33554432);
    props.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG,
             "org.apache.kafka.common.serialization.
              StringSerializer");
    props.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG,
             "org.apache.kafka.common.serialization.String
              Serializer");

    Producer<String, String> producer =
            new KafkaProducer<>(props);
    for (int i = 0; i < 100; i++) {
        Future<RecordMetadata> recordMetadataFuture =
                producer.send(
                new ProducerRecord<>("sample-topic", "Message
                key number: " +
                  Integer.toString(i), "Message value number: " +
                  Integer.toString(i) + "at " + new Date().
                  toString()));
        try {
            RecordMetadata recordMetadata =
            recordMetadataFuture.get();
            System.out.println(recordMetadata);
            Thread.sleep(500);
        } catch (InterruptedException e) {
            e.printStackTrace();
        } catch (ExecutionException e) {
            e.printStackTrace();
        }
    }

    producer.close();
  }
}
```

Listing 4 Kafka consumer application source code

```
public class SampleConsumerApp {

  public static void main(String[] args) {
    Properties props = new Properties();
    props.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG,
              "172.17.0.3:9092");
    props.put(ConsumerConfig.GROUP_ID_CONFIG, "test");
    props.put(ConsumerConfig.ENABLE_AUTO_COMMIT_CONFIG, "true");
      props.put(ConsumerConfig.AUTO_COMMIT_INTERVAL_MS_CONFIG,
              "1000");
      props.put(ConsumerConfig.SESSION_TIMEOUT_MS_CONFIG,
              "30000");
      props.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG,
              "org.apache.kafka.common.serialization.
               StringDeserializer");
      props.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG,
              "org.apache.kafka.common.serialization.
               StringDeserializer");
      KafkaConsumer<String, String> consumer =
      new KafkaConsumer<>(props);
      consumer.subscribe(Arrays.asList("sample-topic"));

      while (true) {
        ConsumerRecords<String, String> records = consumer.
        poll(100);
          records.forEach(record ->
            System.out.printf("offset = %d, key = %s,
            value = %s\n",
                              record.offset(), record.key(),
                              record.value()));
      }
    }
}
```

Basically, the producer application produces a message in a loop and sends that message to the Kafka's sender API. The consumer, also using API, subscribes to the topic and pulls the messages from the queue.

4.3. Running the whole system

In real life scenario, starting a system is not a trivial task to do. Some services may depend on existence of others. Therefore the precedence in which the particular nodes are started is important. The presented system also follows that principle. To be more precise, there is no point of starting producer or a consumer, when the Kafka is not running. That however depends on the Zookeeper. Therefore, the precedence is as follows:

- start a Zookeeper
- start a Kafka and configure it,
- start a consumer application,
- start a producer application.

The system's core services starting commands are presented in the **Listing 5**.

Listing 5 Bash shell commands to start the system's core services

```
$ docker run jplock/zookeeper
$ docker run
--env KAFKA_ZOOKEEPER_CONNECT=”172.17.0.2:2181”
--env KAFKA_PORT=”9092”
--env KAFKA_ADVERTISED_PORT=”9092”
--env KAFKA_ADVERTISED_HOST_NAME=”172.17.0.3”
--expose=9092 wurstmeister/kafka
$ docker run
--env KAFKA_ZOOKEEPER_CONNECT=”172.17.0.2:2181”
--env KAFKA_PORT=”9093”
--env KAFKA_ADVERTISED_PORT=”9093”
--env KAFKA_ADVERTISED_HOST_NAME=”172.17.0.4”
--expose=9093 wurstmeister/kafka
```

IP addresses used in the example, are assigned by the Docker system networking service. Started cluster requires configuration. In order to do so, it is necessary to enter container's shell. This can be achieved with the **$ docker exec -it <container_id> bash** command. The container id can be fetched from the **$ docker ps** command listing running Dockers. The configuration relies on the setting up the topic, partition and replication factor. This is done by executing command in the **Listing 6** inside one of the started Kafka instances.

Listing 6 Kafka cluster configuration command

```
# /opt/kafka_<version>/bin/kafka-topics.sh
--create --zookeeper 172.17.0.2:2181
--replication-factor 2
--partitions 1
--topic sample-topic
```

The replication factor configures the cluster in high availability mode. It is achieved in such way, that the messages are being sent to the leader are automatically replicated to slave nodes as well. To verify which broker is doing what, the command from **Listing 7** should be executed.

Listing 7 Checking brokers status in Kafka cluster

```
# /opt/kafka_<version>/bin/kafka-topics.sh
--describe --zookeeper 172.17.0.2:2181
--topic sample-topic
```

From the command's output it is possible to conclude, which broker acts as a leader, which ones are included in messages replication, and which are in sync.

When the core is up and running it is time to start remaining services. The **Listing 8** presents commands to start the consumer and the producer applications. The order here is of no significance. The Kafka stores messages in a queue, so the consumer normally can start reading messages from the point whenever chooses.

Listing 8 Commands for starting consumer and producer services

```
$ docker run kafka-consumer-app
$ docker run kafka-producer-app
```

Figure 5 presents exemplary output of the producer and consumer applications.

As expected, messages are being produced and consumed.

4.4. High availability from microservices

The advantage of a cluster is that its operation remains not disrupted in case of a Kafka's broker failure. In order to simulate such scenario, one of the Kafka Dockers were stopped during messages transfer. **Figure 6** shows the moment when the container hosting Kafka was stopped. Interesting thing is that, the stopped service was the one, to which the producer initially was connected to.

As it may be observed, at this particular example the producing messages was 18 seconds delayed, but no message was lost.

Cluster's tolerance to brokers' failure has nothing to do with the Docker technology itself. The application and dependent libraries are wrapped by containers which are easy to deploy and instantiate. In real life operation, the underlying cloud framework could automatically handle the failure case by restarting and configuring

Figure 5 Output of the producer and consumer applications.

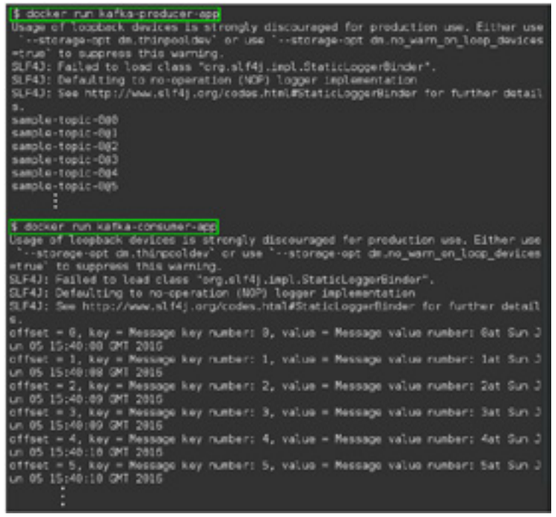
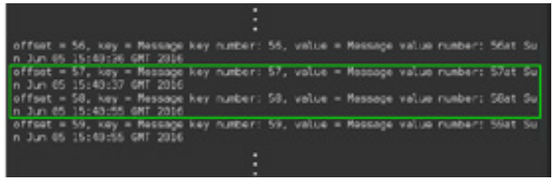


Figure 6 Simulated failure scenario result



the node. Additionally, what happens in the container stays in the container. Meaning, if the containers fails for some reason, it does not spread on the rest of the system. Which was shown in the simple example.

5. Summary

Demand for processing of big amount of data grew rapidly over recent years. This changed the way of thinking about how the applications should be build. Because of its advantages, microservices are fitting in a new trend, gaining more and more popularity. In skilled hands they are very powerful tool which makes building of high quality services in easy and efficient way. Those advantages are something that is highly desired on a competitive IT market. That is why, more software companies introduce them eagerly to be their portfolio's foundations.

References

[1] M. Fowler, “Microservices,” [Online]. Available: http://martin-fowler.com/articles/microservices.html.
[2] “What is Docker?,” [Online]. Available: https://www.docker.com/what-docker.
[3] “Apache Zookeeper,” [Online]. Available: https://zookeeper.apache.org/.
[4] “Apache Kafka,” [Online]. Available: http://kafka.apache.org/documentation.html.
[5] “Docker Docs,” [Online]. Available: https://docs.docker.com/.
[6] [Online]. Available: https://github.com/spotify/docker-maven-plugin.
[7] [Online]. Available: https://github.com/wurstmeister/kafka-docker.
[8] “Introduction to Docker,” [Online]. Available: https://www.javacodegeeks.com/2016/04/introduction-docker-part-1.html.
[9] [Online]. Available: https://hub.docker.com/r/jplock/zookeeper/.
[10] S. Newman, Building Microservices, O'Riley Media Inc, 2015.

About the author

I am a graduate of the Electronics faculty of Wroclaw University of Science and Technology. I work as a Java Software Engineer in the A&A NM & SON department. I am involved in software development software that offers end-to-end solutions in the areas of configuration, fault and performance management for mobile networks operators. The job is very interesting even though challenging at times. All in all, it is good to feel that the work we do in our department serves a good cause.

Krzysztof Bulwiński
Engineer, Software Development
A&A NM & SON

Pyro4 – Python Remote Objects in an Embedded Use Case

Rafał Cichoń
Engineer, Software Development
CIOO Bell Labs

Piotr Rotter
Engineer, Software Development
CIOO Bell Labs

In the era of interconnected networks, remote control and automation seem to be fairly common concepts. In fact, it seems so common to control systems, no matter where they physically are, that we tend to forget it was not always so. Even embedded engineers tend to neglect the complexities and intricacies of remote control. Nowadays one would expect to simply pick a generic remoting component and configure it instead of creating one from scratch. Thus it came so naturally, to answer 'yes we can', when our customer prompted for a remote controller to automate interactions between a number of embedded systems. As is still common in the software business, one does not simply talk to a computer and tell it what to do, therefore we ended up writing some code along with a nice story to share. This article is not meant to act as a tutorial per se, it is our personal and subjective look at using Python for remote systems interaction.

1. Context

The customer, being a hardware engineer, presented the following requirements:

- The system is going to have a Controller managing multiple embedded systems (a.k.a. Boards).
- The solution should be Operating System (OS) independent, however the Boards ship with a GNU/Linux Board Support Package (BSP) and the Controller shall be a Windows machine (sic).
- The Boards and their Controller work in a physically separated Local Area Network (LAN).
- The Controller needs to expose a scripting interface which needs to be easy to use and understand since it is going to be used by hardware engineers – Application Programmable Interface (API).
- The scripting interface should expose objects dynamically based on the embedded system's state.
- At some point the Controller might need a Graphical User Interface (GUI).
- The solution should make it possible for a Controller to stream binary data to the Boards. The Boards should also be capable of streaming data to the Controller.

After numerous brainstorming sessions and coffee talks, the focus shifted towards Python2 because at that time it seemed like a good idea to abstract the Board and Controller in one language. Having a common ecosystem on both ends by definition makes a system more compatible and the verification framework much lighter. The abundance of ready-to-use libraries was also a critical factor in our decision.

Last, but not least, there was no intention to scare off the hardware team by some fancy tech – Python seems to be the lingua franca of modern engineers. Considering the loose security requirements provided by a physically isolated network environment, the remaining step was to pick a Remote Procedure Call (RPC) solution. Given the fact that Python already supports __future__ and import antigravity, using it for RPC seemed pretty straightforward...

2. A few words about Pyro4

Pyro4 stands for Python Remote Objects and is a library written by Irmen de Jong to handle Remote Procedure Calls (RPC) with practically no effort on the user's side. As the slogan puts it – it is designed to be very easy to use and get out of your way as much as possible, but still provide a lot of flexibility when you do need it.

The library is written in pure Python and thus works across different hardware (32-bit, 64-bit, Intel, PowerPC) and operating system (Windows, Linux, OS X) configurations. It also does nice out-of-the-box error handling such as automatically reconnecting all the services in case of connection issues.

For all the details about Pyro4, please refer to its home page [1] which also provides a great tutorial [2]. For the sake of the story, some basic concepts need to be covered and hopefully made clearer than they are in the official documents.

Let us assume that a Server needs to expose some properties (i.e. objects) so that a Client can manipulate them. Later on the process involves a lot more specialized clients and servers under-the-hood, however the general, top-level Client-Server relationship remains valid throughout the discussion.

In order to expose objects, the Server needs to be running a Pyro Daemon and use it to register the objects. The Daemon is then responsible for dispatching requests to the registered objects and returning results to the Client.

Putting the details aside, the Server creates URIs by registering objects with the Pyro Daemon and the Client consumes URIs to provide users with Proxies to the remote objects.

A Proxy is basically a Client-side representation of the Server-side object and contains most of the methods and properties of the original. In most cases the user can manipulate the Proxy just as if it were the original object, unaware of its special, remote nature.

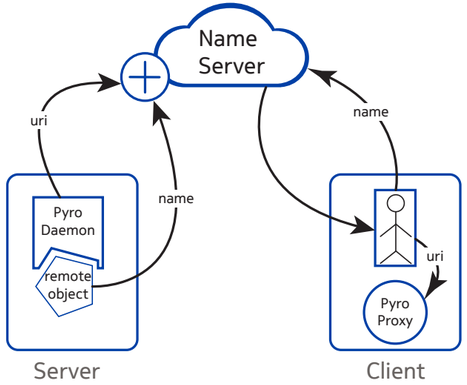
Whenever a method of the Proxy is called, a request is sent to the corresponding Server-side object. By default the Proxy waits for the completion of such a request to yield its result.

The question remains – how to share the URIs between parties at run-time? As an answer, Pyro provides the Name Server service. The Server uses it to register URIs, and the Client uses it to lookup URIs

– obviously a Name Server needs to be visible both to the Client and Server.

It looks like the problem of sharing URIs was replaced by a problem of looking up a Name Server... The missing piece to this puzzle is called a Broadcast Server. The Broadcast Server knows where the Name Server is and shares this piece of information in response to UDP broadcasts. Needless to say, all these interactions are wrapped within the Pyro API.

Figure 1 Object registration



2.1. Launching and stopping

The single most important thing when interfacing libraries is to create a set of test cases to prove that the reality matches ones understanding of the library's interface. The same set of test cases should be used as a sanity check when migrating to a newer version of the library.

The simplest, most degenerate test case for a service is always to start it and stop it, which leads us to implementing server threads.

2.1.1. Server threads

Each of the specialized servers provided by Pyro comes with a blocking request loop which easily fits into a threaded context. The ServerThreadHelper presented in **Listing 1** was one of the first pieces of code to be written. Notice that the __init__ accepts a server object as its argument – this allows the configuration step, which distinguishes one server from another, to happen before a thread is started. Also, the reference to the server object comes in handy when one needs to call server.shutdown().

Listing 1

```
import threading
class ServerThreadHelper(threading.Thread):
    def __init__(self, server, daemon=True):
        super(ServerThreadHelper, self).__init__()
        self._server = server
        self.running = threading.Event()
        self.running.clear()
        self.setDaemon(daemon)

    def run(self):
        self.running.set()
        self._server.requestLoop()
```

2.1.2. Name server reset

A critical functionality from our point of view was the ability to start and stop a Name Server during a single test case. We simply did not want to share the same instance across multiple test cases to avoid configuration spillover.

Sometimes it took significant amounts of time to shut down the Name Server which in turn caused tests to run too long to be practical. After tackling the problem for a while it became evident that the Broadcast Server uses a blocking receive routine and unless it times out, the server does not check for its exit condition. As a workaround, a fake Name Server broadcast lookup is issued as a part of the shutdown request as seen in [**Listing 2**]. This makes the receive function skip the timeout and immediately check the shutdown condition.

Listing 2

```
def _fakeNSLookupBroadcast(self):
    REQUEST_NSURI = "GET_NSURI" if sys.platform == "cli" else
                    b"GET_NSURI"
    port = Pyro4.config.NS_BCPORT
    sock = Pyro4.socketutil.createBroadcastSocket(reuseaddr=
             Pyro4.config.SOCK_REUSE, timeout=0.0)
    for bcaddr in Pyro4.config.parseAddressesString
             (Pyro4.config.BROADCAST_ADDRS):
        try:
            sock.sendto(REQUEST_NSURI, 0, (bcaddr, port))
        except socket.error:
            pass
    sock.close()
```

```
def _shutdownNameServer(self):
    if self._nameServer is not None:
        self._nameServer.shutdown()
        self._nameServer = None
        if self._broadcastServer is not None:
            self._broadcastServer.close()
            self._broadcastServer = None
            self._fakeNSLookupBroadcast()
        if self._nameserverThread is not None:
            self._nameserverThread.join()
            self._nameserverThread = None
            time.sleep(.1)
```

2.1.3. Object re-registration

As was already mentioned, Pyro is able to re-establish connections between the Proxies and their respective remote objects, however the Name Server entries are not automatically recovered. In case the Name Server is reset, a user will not be able to find previously registered entries. To solve this issue, devices need to be registered whenever a Name Server connection is re-established [**Listing 3**].

Listing 3

```
import threading
class NameServerRegistrationHelper(threading.Thread):
    def __init__(self, ipAddress):
        threading.Thread.__init__(self)
        self._nameServer = None
        self._ip = ipAddress
        self._uris = {}
        self._uriLock = threading.Lock()
        self._stop = threading.Event()

    def registerDevice(self, name, uri):
        self._uriLock.acquire()
        self._uris[name] = uri
        if self._nameServer:
            self._nameServer.register(name, uri)
        self._uriLock.release()

    def _reRegisterDevices(self):
        if self._nameServer == None:
            return
        self._uriLock.acquire()
        for name, uri in self._uris.items():
            self._nameServer.register(name, uri)
        self._uriLock.release()
```

```
    def removeAllDevices(self):
        if self._nameServer != None:
            for name in self._uris.keys():
                self._nameServer.remove(name)

    def run(self):
        while not self.stopped():
            try:
                if self._nameServer != None:
                    if self._nameServer.ping():
                        time.sleep(5)
                else:
                    self._nameServer = Pyro4.locateNS()
                    self._reRegisterDevices()

            except Pyro4.errors.NamingError:
                pass
            except Pyro4.errors.ConnectionClosedError:
                self._nameServer = None
        self.removeAllDevices()
```

2.2. Working with Remote Objects

2.2.1. dir() or help()

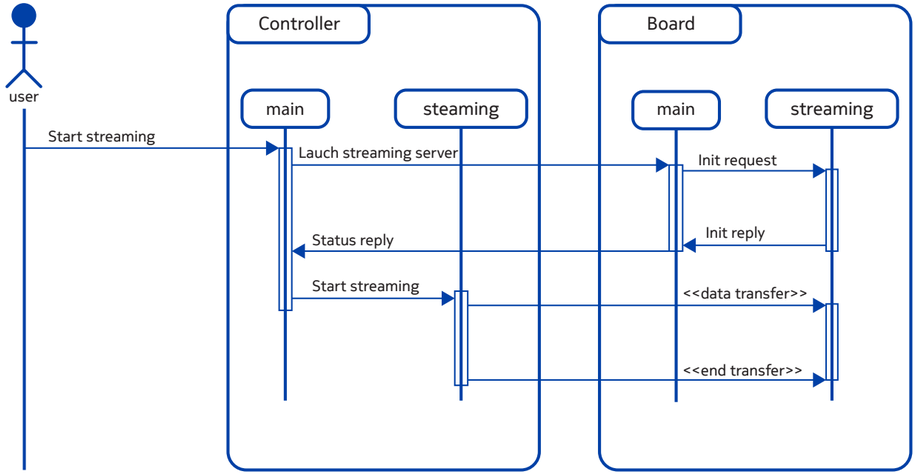
One not-so-obvious caveat of using Pyro is that the Python built-in dir() and help() functions return details of the Proxy and not the remote object it references. There seems to be no easy way around this one and thus we ended up implementing a set of helper methods to use with our classes. The trick is to expose methods which call dir() or help() internally.

2.2.2. Pyro4 vs. iterable objects

Pyro is sometimes very picky when it comes to methods or properties which get to be proxied. One such case is the infamous __getitem__() method which is purposefully skipped "not to promote inefficient coding patterns" [3]. Although the efficiency argument makes sense, most of the time it is more important to make it work and only then make it fast.

Our solution to this problem was to provide users with classes that behave like certain container types, but perform all kinds of magic tricks under the hood, not to cause too many remote calls. The problem effectively comes down to a compromise between caching and networking congestion. Thus the Controller in our case needs to have at least some awareness of the Board side.

Figure 2 Streaming example



2.2.3. Streaming in Pyro4

One of the system requirements is to be able to stream big amounts of data. Passing data as method arguments allows to transfer roughly 2GB from the Controller, but does not cover streaming from the Board, thus it was decided to create an independent streaming service on each end.

The initial idea was to implement the stream handlers efficiently, without the overhead of running additional Python interpreters. The streaming management is still performed using Pyro4, by starting and stopping external processes, which act as stream handlers [**Figure 2**].

3. Conclusion

In our opinion, Pyro not only speeds up the development of RPC solutions but also provides the extensibility which allows the solution to grow in almost any direction. This extensibility gives Pyro a great leverage over most proprietary RPC solutions which tend to be locked to a single operating system and thus distract designers with vendor specific details.

By scripting Pyro with Python, the initial development process felt like "configuring" an already developed solution. It allowed us to quickly start experiments and explore the limitations of our concepts. This early feedback allowed us to shift the design towards clear waters right from the start.

One of the great things about Python is the ability to kick-start the project and allow designers to polish any rough edges as they go. When we needed a GUI for a demo, it took us literally minutes to come up with something operational enough to fit the need. Later on, when the priorities changed, we could put the functional, but not quite ready, GUI aside and get our hands dirty with something else. There was no need to invest time in documenting the customer needs, we could showcase the possible solutions almost instantly.

All in all, thanks to Pyro we had a great time working with the project – as far as we are concerned, there is no better recommendation than to give it a try. Happy coding!

References

[1] I. d. Jong, "Pyro – Python Remote Objects – 4.45," [Online]. Available: http://pythonhosted.org/Pyro4/. [Accessed 27 May 2016].

[2] I. d. Jong, "Pyro4 – tutorial," [Online]. Available: http://pythonhosted.org/Pyro4/tutorials.html. [Accessed 27 May 2016].

[3] "github.com," [Online]. Available: https://github.com/irmen/Pyro4/issues/48. [Accessed 01 06 2016].

About the authors

I'm a software blacksmith and Nokia has been my forge for nearly 6 years. Python and C++ are the tools of my trade. I like C++ a bit more, because when all magic fails, the only thing you can really trust is bare metal. My work is Test Driven and when I say KISS, I really mean Keep It Simple and S.O.L.I.D.
At present I'm supplying the lucky horseshoes to keep our silicon steed running and the nails to build a next generation hardware platform for 5G research.

Rafał Cichoń
Engineer, Software Development
CIOO Bell Labs

I'm a software gardener and I like caring for my code as much as I like caring for the people I work with. I've been with Nokia for roughly two years, patching up broken binary trees, fertilizing the radio plants and reaping the fruits of technology.
Currently I'm supporting a team of brave explorers heading towards the distant 5G galaxy in search of the Jungle of Things (JoT).

Piotr Rotter
Engineer, Software Development
CIOO Bell Labs

Professional Software Development

C++ Idioms – Type Erasure and Expression Templates

Adam Badura
Architect, Software Development
MN HetRAN

NOKIA

1. Introduction

Language idioms are code constructs that are common in a particular language. They are something different (not worse!) than general design patterns because, unlike design patterns, they are strongly bound to their language. Probably the most recognizable C++ idioms are:

- RAII (Resource Acquisition Is Initialization),
- CRTP (Curiously Recurring Template Pattern),
- SFINAE (Substitution Failure Is Not An Error), and
- this day's a bit obsolete, copy-and-swap.

This article shows two less recognizable idioms that deserve more attention, as they are very useful. Those are:

- type erasure idiom, and
- expression templates idiom.

Finally, also the interaction of the idioms is explained.

2. Type erasure idiom

Type erasure idiom allows for hiding an object of many possible concrete types behind an interface common to all of them.

This seems a bit counter-intuitive in C++. After all, C++ prides on strong and static type checking. If so, then why would we want to get rid of the type information, why erase it?

While this is a valid doubt, the answer is as usual: a trade-off. At the extra cost of possible run-time type errors (rather than compile time ones) and longer execution time (some indirection) we gain better compilation time and code reuse possibilities.

The most well-known examples of type erasure are probably **`std::function`** and **`std::shared_ptr`**, although with the second type, we are not typically aware of the type erasure taking place. There are, however, more examples. To mention only some from Boost:

- **`boost::any`**
- **`boost::type_erasure::any`**
- **`boost::spirit::hold_any`**
- **`boost::any_range`**

Some also consider **`boost::variant`** (and similar) to be type erasure. This is true to some extent. However, the article focuses on different aspects of the idiom.

2.1. How about object-oriented programming approach?

Some could say that there is nothing new here. After all, any **`i_something`** interface or **`abstract_something`** base class is a way to do type erasure. It hides all the possible implementations (concrete types) behind a common interface.

While this is obviously true, the intent of type erasure idiom in C++ is a bit different. It achieves what OO approach cannot. For instance:

- The use of such interfaces does not work well with built-in or 3rd party types, which do not implement them. For example, **`int`** is not **`i_comparable`**.
- This gets even worse if you are working with two libraries and each of them uses its own interfaces for same type erasure.
- With this approach, you find yourself operating mostly with pointers and heap allocated objects, which might be hard to accept in some situations due to efficiency.

2.2. How about generic programming approach?

Another way would be to use templates. As different approach as it is, in the end it works just the same – single code handles many different concrete types.

This also has an obvious advantage: the compiler knows everything, which allows wider optimizations. However, there are also drawbacks:

- Compilation time and code dependencies grow significantly.
- Templates cannot be used in virtual methods or through binary module boundaries.

2.3. What is left, then?

Here comes the type erasure idiom. It connects both approaches to leverage on their advantages while work around their drawbacks.

2.3.1. The `any_iterator` example

Have you ever written a virtual method that takes a sequence of elements? Was it by template iterator? No, it was not! Because virtual methods cannot be templates.

Most probably, you had to pick some container and stick to it. Then each call of the method requires the caller to pass the sequence in

this particular container. If the caller happens to have the sequence in some other container, a copy must be made. It does not feel nice, does it? After all, we have iterators for flexibility reasons and not having to stick to particular containers.

Now type erasure comes to the rescue! Let us consider a simple idea – type erased iterator. It behaves like an iterator. It can be constructed from any iterator. However, it does not depend on the actual iterator concrete type. Such a construct is often called *any_iterator*.

First, let us express the abstract iterator interface in OO style.

Listing 1 Abstract iterator interface

```
template<typename ValueType>
class iterator_base
{
public:
  virtual ~iterator_base() = default;

  virtual void advance() = 0;
  virtual ValueType& dereference() const = 0;
  virtual bool is_equal(
     iterator_base const& right) const = 0;

  virtual iterator_base* clone() const = 0;
};
```

Behind such an interface, we can hide any concrete iterator and easily implement required methods using ordinary means.

Listing 2 Iterator interface implementation

```
template<typename IteratorType>
class iterator_impl
  : public iterator_base<
     typename std::iterator_traits<IteratorType>::value_type>
{
private:
  typedef
     typename std::iterator_traits<IteratorType>::value_type
     value_type;

public:
  iterator_impl(IteratorType it)
     : it(std::move(it))
  {
  }

  virtual void advance() override
  {
     ++(this->it);
  }

  virtual value_type& dereference() const override
  {
     return *(this->it);
  }

  virtual bool is_equal(
     iterator_base<value_type> const& right) const override
  {
    iterator_impl const& typed_right =
     dynamic_cast<iterator_impl const&>(right);
    return this->it == typed_right.it;
  }

  virtual iterator_impl* clone() const override[1]
  {
    return new iterator_impl(this->it);
  }

private:
  IteratorType it;
};
```

So far so good. We used a bit of the OO approach by having a classic interface and its implementations. We used a bit of the generic programming approach by "generating" (from template) implementation classes that adapt existing iterators to our interface. Now we only need a nice wrapper providing a typical iterator interface (the so called "external interface").

Listing 3 The **any_iterator** itself

```
template<typename ValueType>
class any_iterator
  : public boost::iterator_facade<
            any_iterator<ValueType>, ValueType,
            boost::forward_traversal_tag>
{
public:
  any_iterator() = default;

  template<typename IteratorType>
  any_iterator(IteratorType it)
    : holder(new iterator_impl<IteratorType>(std::move(it)))
  {}

  any_iterator(any_iterator const& right)
    : holder(right.holder ? right.holder->clone() : nullptr)
  {
  }

  /* ...constructors and assignments... */

private:
  friend class boost::iterator_core_access;

  void increment()
  {
    this->holder->advance();
  }

  bool equal(any_iterator const& right) const
  {
    return this->holder->is_equal(*(right.holder));
  }

  ValueType& dereference() const
  {
    return this->holder->dereference();
  }

  std::unique_ptr<iterator_base<ValueType>> holder;
};
```

The above **any_iterator** implementation uses the class template **boost::iterator_facade**. The facade makes it much easier to implement new iterators by covering work around operators overloading. While using it, a programmer needs only to implement a few intuitively-named functions and **iterator_facade** does all the magic (with CRTP) to make it behave like a real iterator.

The above implementation clearly is not complete. Even for an iterator interface it covers only forward iterators. Nevertheless, it is sufficient to make the point.

Listing 4 **any_iterator** usage example

```
void print_any_sequence(
  any_iterator<int> begin, any_iterator<int> end)
{
  for (; begin != end; ++begin)
    std::cout << *begin << '\n';
}
int main()
{
  std::vector<int> v {1, 2, 3, 4};
  print_any_sequence(v.begin(), v.end());
  print_any_sequence(v.rbegin(), v.rend());
  int a[] {1, 2, 3, 4};
  print_any_sequence(a, a + sizeof(a) / sizeof(a[0]));
}
```

As we can see in the example above, **print_any_sequence** is not a function template. Nevertheless, it does accept and work with any kinds of iterators!

2.3.2. Other examples

Other examples were also mentioned at the beginning, among them **std::function**, **std::shared_ptr** and **boost::any**. They all use a similar or even the same approach. The differences are only in the "external interface" they present.

Perhaps the **std::shared_ptr** deserves more explanation here, as it might not be obvious how it uses this idiom. When a **shared_ptr** is constructed from a C pointer, it creates an internal counter structure. That structure, however, stores more than just the use counter. It also stores the delete function (which by default uses **delete**). Thanks to this approach, **shared_ptr** type doesn't depend on the delete function (or functor) type, unlike other smart pointers which expose deleters in their own types by template parameters.

Another notable case is **boost::type_erasure::any** type from Boost.TypeErasure library [1], which is a template implementation of the idiom. **any** type allows to define (by "template magic") type-erasing wrappers which have required "external interface". This way the core implementation is done just once while programmers can focus on providing operations for "external interface" that they need. In fact, it seems that the Boost.TypeErasure library could be the final word on type erasure idiom and any use of the idiom (including previously mentioned ones) could or even should be implemented using it.

2.4. Summary

As the example showed, **any_iterator** will work with... well... any iterator. However, since it does not depend on the underlying iterator type – that type was *erased* – we can use it without templates. It still behaves like a value and does not force its user to implement any new concrete types for new iterators – internal templates already handle this.

1 The return type of this method is not a mistake. It is a language feature called return type covariance. Although it is rarely used, it fits perfectly in such cases.

However, as always, there are disadvantages. Firstly, we do need heap allocation after all. This problem could be reduced by using small buffer optimization, however, another problem remains. Now, every single operation on the concrete type (invoked through the interface) requires a virtual method call. Depending on your context, this might be acceptable or not.

3. Expression templates idiom

Expression templates can be thought of as an improved method chaining technique. While method chaining is common to object-oriented languages, C++ with its templates pushed it further, making its own specific idiom.

The core idea behind expression templates is lazy evaluation. Instead of computing the expression at run-time, we only produce (at compile time) a type that describes what needs to be computed. Thanks to such an approach, we can compute only those parts of the result that we are actually asked for. Furthermore, while doing that computation, we already know about the entire expression, and thus can avoid some intermediate costs.

3.1. Example of the idiom

The idiom found numerous uses in libraries for high performance computations – for example, Boost.uBLAS. Another typical use case is creating domain-specific languages within C++, such as Boost.Spirit.

This article, however, shows a less popular example – concatenation of **vector**s.

Listing 5 Concatenation without idiom

```
template<typename T>
std::vector<T> operator ,(
  std::vector<T> left, std::vector<T> const& right)
{
  left.reserve(left.size() + right.size());
  left.insert(left.end(),
    right.cbegin(), right.cend());
  return std::move(left);
}
```

The above implementation of the concatenation is simple but also inefficient. If multiple **vector**s are concatenated that way, then each time we will have to allocate new memory for more elements.[2]

The use of expression templates allows for avoiding this problem. The expression does not produce a direct result. Instead, it produces a "recursive template" which describes the result. This way once the actual **vector** must be made (upon assignment), the entire expression is "encoded" in the result type and can be used for example, to determine the needed buffer size.

Listing 6 Concatenation with idiom

```
template<typename Left, typename Right>
struct concatenation
{
  typedef typename Left::value_type value_type;

  operator std::vector<value_type>() const
  {
    std::vector<value_type> result;
    result.reserve(size());
    append(result);
    return result;
  }

  typename std::vector<value_type>::size_type size() const
  {
    return left.size() + right.size();
  }

  void append(std::vector<value_type>& result) const
  {
    appender<Left>::append(left, result);
    appender<Right>::append(right, result);
  }

  template<typename Container>
  struct appender
  {
    static void append(
      Container const& container,
      std::vector<value_type>& result)
    {
      container.append(result);
    }
  };

  template<typename T>
  struct appender<std::vector<T>>
  {
    static void append(
      std::vector<T> const& container,
      std::vector<value_type>& result)
    {
      result.insert(result.end(),
        container.cbegin(), container.cend());
    }
  };

  Left const& left;
  Right const& right;
};

template<typename T>
concatenation<std::vector<T>, std::vector<T>>
  operator ,(
    std::vector<T> const& left,
    std::vector<T> const& right)
{
  return {left, right};
}

template<typename Left, typename Right, typename T>
concatenation<concatenation<Left, Right>, std::vector<T>>
  operator ,(
    concatenation<Left, Right> const& left,
    std::vector<T> const& right)
{
  return {left, right};
}
```

The **operator ,** no longer returns the new **vector**. Instead, now it returns an object of **concatenation** template class without doing any actual concatenation. Arguments to **operator** are stored by reference in the returned object and nothing else is done.

Actual concatenation is done only once a **vector** is requested, which happens in **operator std::vector<value_type>**. This conversion operator will be typically called when the concatenation expression is assigned to a **vector** object.

The **concatenation** type resembles a degenerated binary tree (which follows the expression structure itself). Every **concatenation** object has **left** and **right** elements. Each of them might be another **concatenation** object or **vector** object, in which case they are leafs. Before continuing, consider the below example of the use of concatenation (same code works with both versions):

Listing 7 Concatenation use example

```
std::vector<int> const vec {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
std::vector<int> const result = (vec, vec, vec, vec, vec);
```

Such a construction results in generation (at compile time) of the following type:

Listing 8 Example concatenation expression type

```
concatenation<
  concatenation<
    concatenation<
      concatenation<
        std::vector<int>,
        std::vector<int>
      >,
      std::vector<int>
    >,
    std::vector<int>
  >,
  std::vector<int>
>
```

The main advantage of the idiom is full knowledge at compile time. When finally the conversion operator is called on the outermost **concatenation** object, it can easily sum sizes of all **vector**s and do only one allocation instead of allocating anew at each level.

Here, only the conversion operator to **vector** is shown. The potential is, however, much bigger. Just for example:

- We could provide more conversions.
- Or we could check **vector**s used in concatenation and those which are r-value references could be reused (if they have large enough buffer) to avoid allocation.
- Or we could even skip making a new object altogether and just provide a **vector**-like interface. This way if for example, the caller would then only use **std::find**, we would never allocate any memory at all!

4. Idioms interaction

Finally, let us look at how those two idioms interact.

Expression templates result in complex concrete types that cannot be easily used without **auto** (for local use) or template arguments (when passing to the next function).

2 This implementation at the least has left argument as a value (copy elision), then acts on it (as if with a local object), and finally moves it for return value. Thanks to that buffer allocated in one call will be reused in the next call (if large enough). Considering the typical **vector** strategy of doubling buffer, this is a significant gain for longer concatenation chains.

Obviously, we can immediately convert such an expression to some "ordinary type" as we did in the example above.

However, since expression templates rely on lazy evaluation then typically, the later we do the conversion, the better, as we might be able to avoid it in the end.

Now we are left with **auto** and template arguments. The first one is usually heavily limited in use, since expression templates often use temporary values, which are invalidated past the end of their statement.[3]

The other alternative, passing to another function within the same statement, allows for avoiding the temporary value problem. Its lifetime is stack-based, therefore it will exist for the entire execution of the function to which it is passed. The problem here is, however, that such an approach requires the function to accept the template argument. This means that function body will be visible to everyone. Compilation time will suffer. Dependencies will grow. Not to mention that virtual methods just cannot be templates.

Now it can be easily seen just how the idioms complement each other. Type erasure idiom addresses exactly the same problem that we have with the expression templates idiom. It allows for avoiding propagation of templates and hiding complex concrete types behind abstracted interfaces.

It is often difficult to use both idioms, and it is not always even worth doing so considering that type erasure adds run-time cost while we use expression templates to avoid run-time costs. Nevertheless, when they do work together the results are great!

References

[1] "Boost.TypeErasure library," [Online]. Available: http://www.boost.org/doc/libs/1_61_0/libs/type_erasure/.

[2] "Type erasure on Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/Type_erasure.

[3] "Expression templates on Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/Expression_templates.

[4] "More C++ Idioms," [Online]. Available: https://en.wikibooks.org/wiki/More_C%2B%2B_Idioms.

3 There is a trick here. A temporary value can be bound to a **const** reference local variable to extend its lifetime to the end of the block.

About the author

I am a graduate of the Computer Science and Management Faculty at Wrocław University of Technology. However, my fascination about programming began much earlier, starting at the age of 10, with introduction to C++ at age of 14. After all those years of experience in home projects, academic projects and work, this programming language and the discipline in general still surprises me with its power and possibilities.

Adam Badura
Architect, Software Development
MN HetRAN

Data Structures and Algorithms for Set Operations in Mobile Load Balancing

Łukasz Grządko
Senior Engineer, Software Development
MN LTE

NOKIA

Problem statement

Future Long Term Evolution (LTE) Radio Access Network (RAN) will benefit from a significant degree of self-organization. The principal objective of introducing self-organization is to substantially reduce the necessary human intervention in network operations and improve network quality [13]. To avoid adjusting parameters manually, the parameter tuning is done automatically based on measurements. This is the main LTE concept of Self-Optimizing Networks (SON). A challenge is to deliver additional performance gain further improving network efficiency. Because Load Balancing (LB) is an important way to reduce eNB overloading and improve system capability, Autonomic Load Balancing is considered as a part of self-organization for LTE RAN [13]. The term LB can be used to describe any mechanism whereby highly loaded cells distribute some of their traffic to less heavily loaded neighbours in order to make use of the radio resource more efficiently across the whole network [8]. The load balancing algorithm aims at finding the optimum handover (HO) offset value between the overloaded cell and a possible target cell. This optimized offset value will assure that the users that are handed over to the target cell will not be returned to the source cell and thus the load in the current cell is diminished [9]. The HO procedure begins with a measurement report (MR) transmission from a UE to its source cell. The UE periodically performs downlink channel measurements based on cell specific reference signals and checks whether the signal strength satisfies the conditions for MR transmission. The entering condition for event $A3$, which is one of the MR triggering events generally used for the HO procedure is defined as[10]: $M_1 - M_0 > Hys_0 + A3Offset_0 - CIO_{0,1}$ where M_0 and M_1 are the signal strengths of the serving cell ($Cell\ 0$) and the target cell ($Cell\ 1$) respectively. Hys_0 is the hysteresis parameter and $A3\,Offset_0$ is the offset parameter for event $A3$. For simplicity, we disregard these two parameters. By substituting Hys_0 and $A3Offset_0$ zero we obtain: $M_1 - M_0 > -CIO_{0,1}$, where $CIO_{0,1}$ is a cell-specific offset parameter set by $Cell\ 0$ for $Cell\ 1$ and is called Cell Individual Offset. If condition is satisfied for duration of Time to Trigger, the UE sends the MR to $Cell\ 0$ and the HO procedure from $Cell\ 0$ to $Cell\ 1$ is initiated [10].

The operational principle of HO Mobile Load Balancing (HO MLB) is illustrated in **Figures** 1 and 2. As shown in **Figure** 1, 8 UEs and 2 UEs belong to $Cell\ 0$ and $Cell\ 1$ respectively. $Cell\ 0$ is more loaded than $Cell\ 1$. In this case as illustrated in **Figure** 1, the HO MLB scheme increases $CIO_{0,1}$ to make the HO timing from $Cell\ 0$ to $Cell\ 1$ earlier, and UEs of $Cell\ 0$ near $Cell\ 1$ will hand off to $Cell\ 1$. In addition, as shown in **Figure** 2, the HO-MLB scheme can decrease $CIO_{0,1}$ to make HO timing from $Cell\ 1$ to $Cell\ 0$ later in order to keep the handed off UEs from $Cell\ 0$ staying in $Cell\ 1$ [10].

There is also another Load Balancing scheme based on Cell Reselection[10].

Figure 1 Increasing CIO, dually means decreasing of –CIO. New logical Cell perimeter is illustrated as dashed.

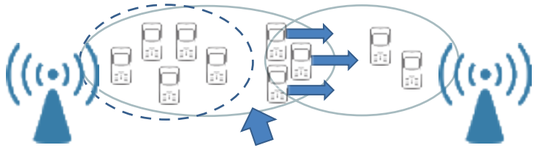
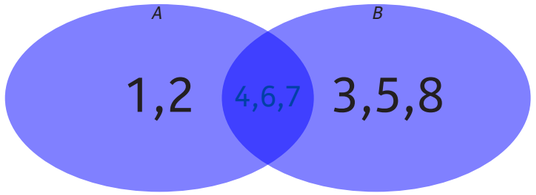


Figure 2 Decreasing CIO means that Cell radius decreases. It is equivalent to increasing – CIO value. All Cells are load balanced.

1. Bulk load balancing

We introduce the bulk notion. It means that from the system point of view, many UEs can be reconfigured in one pass. The same notion will be used for handling Set Operations like union, intersection and difference. Suppose now that 8 UEs (compare to **Figure** 1) are numbered from 1 in BTS $Cell\ 0$. All the UEs must be reconfigured due to CIO changing. We can imagine that these UEs are placed in an ordered list or set ueList $\{1,2,3,...,8\}$. Next, UE is popped from the head of the ueList and reconfigured. After reconfiguration of UE 1, UE 2, the remaining UEs to be reconfigured are: $\{3,4,5,6,7,8\}$. Suppose now that three of the UEs are handed off to $Cell\ 1$ (compare to **Figure** 1). Let it be UEs $\{3,5,8\}$. $Cell\ 0$ contains then UEs from set $\{1,2,4,6,7\}$ and all of them must be reconfigured again. In some cases it is not efficient to wait with reconfiguration of UEs numbered 4,6,7 because, for example, it may cause starvation of these UEs. Therefore UEs numbered from 1 to 2 should be reconfigured later. Now we have two lists, $A=\{1,2,4,6,7\}$ and our operational list $B=\{3,4,5,6,7,8\}$. We may interpret set A as actual $Cell\ 0$ content, whereas set B contains UEs which have not been reconfigured yet in $Cell\ 0$. UEs from set $A \cap B$ have a higher priority. Afterwards, UEs from A / B are processed. Observe that UEs $\{3,5,8\}$ are excluded from further processing, they were handed off to another eNB. We may illustrate this in **Figure** 3.

Figure 3 Processing Cell 0's UEs from $A \cap B = \{4,6,7\}$ and $A / B = \{1,2\}$



From now on we may forget about the Load Balancing issue and focus on effective approaches to Set Operations calculation. The purpose is to do it as efficiently as possible due to the large number UEs to reconfigure.

2. Data Structures and Algorithms for Set Operations

Because there are many approaches to solve the problem, we should find the optimal solution or one that is as good as possible in the given environment. Some mathematical methods are very necessary to analyze the complexity of a given algorithm. Often the shortest solution does not mean that it optimally uses resources such as time and space.

2.1. Set Operations using Bitwise operations

Let U be some Universum and $A,B \subseteq U$.

Encode the numbers using corresponding bits in 2^k bits number where k is an integer. Suppose also n and m are sizes of A and B respectively.

The set $\{1,2,4,6,7\}$ is encoded to $\{2^1+2^2+2^4+2^6+2^7\}$. In a binary form, this number is $01101010b$. The second set $\{3,4,5,6,7,8\}$ is encoded to $11111000b$.

The bitwise and result is $01101010b \,\&\, 11111000b = 01101000b = \{4,6,7\}$

Using deMorgan's law, we can rewrite the difference of sets A and B, i.e. A/B as follows:
$\forall_x x \in A/B \Leftrightarrow x \in A \wedge x \notin B \Leftrightarrow x \in A \wedge x \in U/B \Leftrightarrow x \in A \wedge x \in B' \Leftrightarrow x \in A \cap B'$

In bitwise language this can be encoded as $A \& \sim B$

$01101010b \,\&\, \sim 11111000b = 01101010b \,\&\, 00000111 = 110b = \{1,2\}$

Note that $std::bitset$ can be used for bitwise operations. We may also go through a smaller set, say A, and check the corresponding bit in bitset B. It costs $O(min\{n,m\})$ time. One may ask how to go through just the only ones in the bitstring. If id id is the binary value of set A, then all the $1-bits$ can be read by binary operation: $id = (id \,\&\, -id)$. This can be compared to Fenwick Trees implementation on the TopCoder website [19].

2.2. C++ STL primitives

For set intersection, $std::set\_intersection(A, B)$ might be used. The condition that must be met is that elements of A and B must be ordered. $std::set\_intersection$ works in time and space complexity $O(n+m)$. The algorithm is similar to merging lists in Merge Sort [11, 13]. Imagine two pointers i and j, each per one list, which move from left to right of each list. If $a_i < b_j$ then move i to the right, if $a_i > b_j$ then move j to the right, otherwise $a_i = b_j$ and it is a common value of two sets, i and j are both moved to the right. For A and B from our example, the order of pointers' movement is: a_1, a_2, b_1, b_2, a_3, b_3, a_4, b_4, a_5, b_5, b_6.

Suppose that $m \le n$. Then the overall time complexity of set intersection is $O(nlogn)$, because of elements sorting. However, the inputs can be sorted in linear time using Radix Sort, Bucket Sort [1,3].

For set difference, $std::set\_difference$ might be used. Time and space complexity is the same like in the intersection algorithm.

2.3. Binary searching, Baeza Yates algorithm

Before we describe the main idea of Baeza algorithm we will show another simple approach based on binary searching. Each element $a_i \in A$ is being searched in B. If B is not ordered, then the algorithm based on simple iterative search costs $O(nm)$ time. However if B is sorted, then binary search can be used (**Listing 1**). Note that instead of B defined as $std::vector$ container, $std::set$ container can be used. Then we do not worry about elements order and such an algorithm works in time $O(nlogm)$, because searching in $std::set$ costs $O(logm)$ time. However, the constant in complexity can be larger than using sorted vectors.

The solution with ordered vector also costs time $O(nlogm)$ as binary search works in time $O(logm)$. Observe that if $n > m$ then $\frac{n}{m} > \frac{logn}{logm}$.

Therefore whenever $n > m$ it is better to swap list A and list B and complexity $O(mlogn)$ is more effective. Next nice property is when $m = \frac{n}{logn}$ then the result complexity is linear $O(m)$.

Baeza Yates' algorithm is based on double binary search [16]. The idea of the algorithm is to find median M of A in B. M divides A into two parts, similarly insertion rank index divides B into two parts illustrated in **Table 1**.

Listing 1

```
vector<int> intersect(vector<int> const & input1,
                      vector<int> const & input2)
{
  vector<int> output;
  copy_if(input1.begin(), input1.end(), back_inserter(output),
  [&](int t) {
    return binary_search(input2.begin(), input2.end(), t);
  });
  return output;
}
```

Table 1

Median $M = 12$

A	1	5	8	9	12	15	17	18	19

Insertion rank index of M is $r = 3$

B	7	8	9	12	13	17	19	20	23	24	25	27	29

The green parts and blue parts of the tables are recursively and independently calculated. On each level, if the size of A is larger than the size of B, then A and B should be swapped. In this case, in the green part, M is selected in B, while in the blue part, M is selected in A.

The depicted algorithm performs on average less comparisons than $n+m$ when $n = \alpha m$, $\alpha > 1$. In the worst case the number of iterations is $(2m+1)\lg\left(\frac{n+1}{m+1}\right) + 2m + O(logn)$. If $n = \alpha m$ the total complexity is $O(n)$ [15].

2.3.1. Median searching issues

What if table A is unordered? In that case the total complexity of the algorithm increases. For median searching, several approaches can be applied such as:

- Hoare Selection algorithm $O(nlogn)$
- Median of medians algorithm (linear time)
- Static Interval Tree with query time $O(logn)$
- Floyd Rivest algorithm (average time $O(n)$)

2.3.2. Binary searching issues

Binary searching may yield a slower CPU performance when the array is very large and partially cached. Division factor increase can be done to reduce cache misses [17].

If the array size is unbounded, then galloping searching, introduced by Bentley and Yao, can be used. It searches insertion rank p of a key x using consecutive steps which sizes are: $1,2^1,2^2,...,2^i$. It resolves the search problem in $2\log(p+1)$ comparisons. In some cases it is a better result than standard binary searching [12].

2.4. Probabilistic data structures

When processing large data sets, sometimes it is more important to find an element than adjusting data structure. There is not known a single algorithm for each weather. In case query and space are crucial, then it is useful to take hash tables into account. A probabilistic approach is required for huge data sets, because such data cannot be stored directly in the system memory [18].

2.4.1. Hash tables

A hash table is a data structure which maps keys to values, for example, a function from the universe set U of all keys to the range $[q] = \{0,1,\ldots,q-1\}$. A hash table uses hash function to compute an index into an array of buckets from which the value can be found. The simplest hash function is $f(key) = key \bmod q$. It is possible that two different keys will generate the same hash value pointing to the same bucket, causing hash collision. For example, if $f(x) = x \bmod 5$ is a hash function, then $f(1) = f(6)$ is causing collision. There are several strategies to handle such events. These are Separate chaining, Open addressing, Robin Hood hashing [5]. In an average case the time complexity is constant, however in the worst case it depends what kind of data structure is used for the chaining list. Using hash representation of sets can in average speed up the calculation of intersection of sets A and B (where $sizeof(A) < sizeof(B)$) significantly. Expected time complexity is $O(n)$ because of looking up all n elements from A in hash representation of B. Sample implementation of intersection and difference can be found in **Listing 2**. In the worst case, if all values are in the same bucket, the time complexity is $O(nm)$. There exists a method which ensures $O(n)$ time in pessimistic variant too, however some additional conditions must be met.

2.4.2. Perfect Hash Function (PHF)

A hash function h is called perfect hash function for a subset $S \subseteq U$ of size $n \le q$ if h is bijection on S. There exist algorithms which use average $O(n)$ time construction, $O(n)$ storage and $O(1)$ for query.

Listing 2

```
vector<int> intersect(vector<int> const& input1,
              unordered_set<int> const& input2)
{
  std::vector<int> output;
  copy_if(input1.begin(), input1.end(), back_inserter(output),
   [&](int t) {
     auto it = input2.find(t);
     return it != input2.end();
   });
  return output;
}
void  difference(vector<int> const& input1,
         unordered_set<int> const& input2)
{
  input1.erase(remove_if(input1.begin(), input1.end(), [&](int t) {
     auto it = input2.find(t);
     return it != input2.end();
   }), input1.end());
}
```

PHF construction

We assume that $p=q+1$ is a prime number. One lemma and two corollaries will be necessary for the construction.

Lemma. Given $W \subseteq U$ with $|W| = r$ and given $k \in U$ and $s \geq r$, let $B(s,W,k,j)=|\{x | x \in W \text{ and } (kx \bmod p) \bmod s = j\}|$ for $0 \leq j < s$. Then there exists $k \in U$ such that $\sum_{j=0}^{s-1}\left(\frac{B(s,W,k,j)}{2}\right) < \frac{r^2}{s}$.

Corollary 1. There exists $k \in U$ such that $\sum_{j=0}^{r-1} B(r,W,k,j)^2 < 3r$.

Corollary 2. There exists $k' \in U$ such that mapping (g) $x \to k'x \bmod p \bmod r^2$ is injection when restricted to W [14].

Given $S \subseteq U$, T is the table for storage. $T[0]$ is used to partition S into n blocks W_j. Each block is determined by the value of the function $f(x)=kx \bmod p \bmod n$; pointers to corresponding blocks T_j are provided in locations $T[j+1]$. k is chosen to satisfy *Corollary 1*, with $W=S$ and $r=n$, so that $\sum |W_j|^2 < 3n$. That means the storage for S is $O(n)$. The amount of space allocated to the block W_j is $|W_j|^2+2$. W_j is resolved within this space by using the perfect hash function provided by *Corollary 2*, setting $W=W_j$ and $r=|W_j|$. In the first location of T_j, $|W_j|$ is stored. In the second value k' is provided. $x \in W_j$ is stored in location $(k'x \bmod p \bmod |W_j|^2)+2$ of block T_j [14].

Example: $S=\{2,6,10,15\}$, $q=16$, $p=17$. First of all, we search for the appropriate k. Extend f definition, so that $f(S)=\{f(x) | x \in S\}$. In other words, $f(S)$ is an ordered set of remainders with restriction to $f(x)$ definition. Then $f\{2,6,10,15\}=\{2,2,2,2\}$. $k=1$ is not a good candidate as $4^2 > 3*4$. If $k=2$, then $f\{2,6,10,15\}=\{0,0,3,1\}$ and it is a good candidate as $2^2+1^2+1^2 < 3*4$. $W_0=\{2,6\}, W_1=\{15\}, W_2=\emptyset, W_3=\{10\}$. For example, we calculate k' for $W=W_4$ and $r=2$. For $k'=1$, $g(2)=g(6)$, therefore the mapping of g is not $1-1$. If $k'=3$ then $g(2)=2$ and $g(6)=1$. The result T is illustrated in **Table 2**.

Table 2 Constructed Table T. For example $T[4]$ representing block W_3 points to element 14th

T	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	2	5	11	⊥	14	2	3	⊥	6	2	⊥	1	1	15	1	1	10
	k					$\lvert W_1\rvert$	k'					$\lvert W_2\rvert$	k'		$\lvert W_3\rvert$	k'	

A query for 2 is processed as follows:

1) $k=T[0]=2, j=2k \bmod 17 \bmod 4=0$.

2) $T[0+1] \neq \perp$ and $T[T[1]+1]=k'=3$, $2k' \bmod 17 \bmod 4=2$. The cell $5+2+2$th indeed contains element 2. The construction time is $O(nm)$ in the worst case. However, the representation can be constructed in randomized expected time $O(n)$ [14].

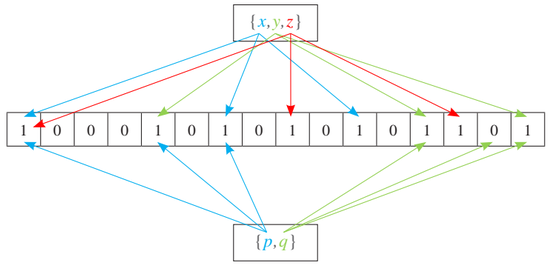
There is also the known "hash-and-displace" approach with data compression on a sequence of hash function indices [11]. You may also find some very nice approach implemented by algorithmic world champion Tomasz Czajka [2].

2.4.3. Bloom filters

A Bloom Filter (conceived by Burton Howard Bloom in 1970 [5]), similarly to hash table, allows us to quickly decide whether a given element is in the set or not. However, space requirements are much smaller than those of a hash table. For example, consider elements like strings with average size of 800 bits. A hash table for storing 10^8 elements would require $8*10^{10}$ bits. A basic Bloom filter-based structure would only require $145*10^6$ bits which can be easily stored in memory. On the other hand, it requires a certain additional cost [6]. Because of probabilistic nature, a query returns either "possibly in set" or "definitely not in set". If the key is present in the structure then Bloom filter always returns the correct answer, however if key is absent it might return the wrong answer [5, 6].

A Bloom filter is an array of q bits for representing a set $S=\{x_1, \ldots, x_n\}$. The key idea is to use k hash functions $h_i(x)$ to map items $x \in S$ to random numbers uniform in the range $[q]$. An element x is inserted into the filter by setting bit in each $h_i(x)$. x is possibly a member of S if the bits for each $h_i(x)$ are set and guaranteed not to be a member if any bit is not set [7].

Figure 4 Sample Bloom filter for three elements x, y, z. position 0 has a collision. A sample query for elements p and q. q is reported not present, while p reported presented (false positive), though never added.



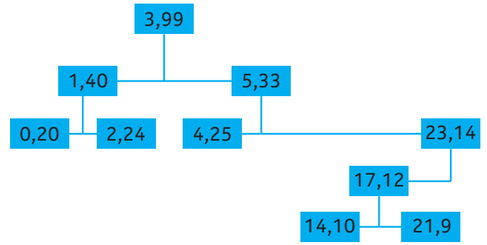
The probability that the bit is one after inserting n elements is: $1-(1-\frac{1}{q})^{kn}$. For an element membership, the probability of "false positive" is: $(1-(1-\frac{1}{q})^{kn})^k \approx (1-e^{\frac{-kn}{q}})^k$.
$e^{\frac{-kn}{q}}$ is a very close approximation of $(1-\frac{1}{q})^{kn}$ [7]. The false positive probability decreases as q increases and probability increases with n. Optimal k value is minimum probability of false positive, which can be calculated by derivative of $(1-e^{\frac{-kn}{q}})^k$. Then k_{opt} is $\frac{q}{n} ln 2$.

2.4.4. Parallelization using probabilistic data structures

Something about the parallelization scheme should also be mentioned. The main key to manage parallelization is to use the appropriate data structure. One of the known data structures are random balanced binary trees (treaps). For two sets of sizes $m \leq n$ the algorithm run in expected time $O(mlg(\frac{n}{m}))$ and $O(lgn)$ depth parallel time on an Exclusive Read/Write Parallel Random Access Machine (EREW PRAM) with unit time scan operations. The serial time, in which the sets are of varying and different sizes, is better than using a regular linear time merge $O(n+m)$. The main advantage of treaps operations are their simplicity. Merging two ordered sets requires at least $\lceil \log(\frac{n+m}{n}) \rceil$ comparisons. That means that in practice, an algorithm based on treaps is optimal because for $m \leq n$, $\lceil \log(\frac{n+m}{n}) \rceil = \theta(mlg(\frac{n}{m}))$. The data structure is also competitive to Splay Trees, Skip Lists and Red Black trees [4].

Each node in the tree has a key and an associated random priority [4]. The nodes appear in-order by their keys and are heap-ordered by their priorities [22]. For any node x all nodes in the left subtree of x have keys less than x, while in the right subtree they have keys larger than x, like Binary Search Tree (BST)[11, 13]. All ancestors of x have priorities larger than x's priority, and all descendants of x have smaller priorities than x [4]. Please compare to Heaps data structure [11,13].

Figure 5 Treap with each node's (key, priority) values



Two basic operations *insert* and *delete* can be implemented using node rotation [1,22]. For insertion of x we create a new node v with key x and some random priority. Afterwards, we use BST (for example, AVL tree [1]) algorithm so that v is the leaf of the tree. In this case heap property can be not met, therefore we can use our rotation to restore the heap property of the treap [23]. Now we can move on to two very important operations, *split* and *join*, which are basic operators.

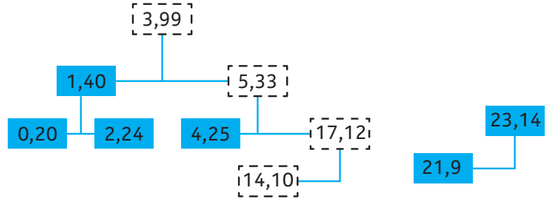
Main primitives

$$Split(T, key) \rightarrow (T_l, x, T_g)$$

Split(T, *key*), splits T into two trees: T_l with key values less than *key* and T_g with key values greater than *key*. If T has a node x with a key value equal to key, then x is returned too [4].

join(T_l, T_g), combines two trees into one treap T. The largest value in T_l is less than the smallest value in T_g [4]. These two primitives will be used to define further operations. The time complexity of both operators is $O(logn)$.

Figure 5 Splitting treap from Figure 5 by the key = 20 into Trees T_l (*left tree*) and T_g (*right tree*).



To join two treaps T_l with keys less than key and T_g with keys greater than key, a join traverses the right spine of T_l and the left spine of T_g. For instance, the right spine is in dashed rectangles [4]. Now we can use split and join operations to make some bulk operations.

Sets Union and Intersection

We will show the intersection operation in detail. Suppose we have two treaps with roots $r1$ and $r2$ respectively and $r1$ $priority$ is larger than $r2$ $priority$. $r2$ is split by $r1$ key and the result of this operation is stored to T_l and T_g tree. Next we find recursively intersection between $r1$ left subtree and T_l tree and intersection between $r1$ right subtree and T_g tree. Note that the keys in $r1$ left subtree and in T_l tree are smaller than $r1$ key. Analogically, the similar situation happens for the right subtree. If no duplicate is found by $split$, then the intersection is $join$ between the last two results of recursive sub-intersections. Otherwise, if key appeared in both trees, the results of sub-intersections become the left and right children of the root used to split [4]. Because of the divide and conquer [1, 3] nature of the algorithm, it can be parallelized per each recursive call. A parallel union algorithm using minimum number of blocks, say k, in a partitioning of A such that no value of B lies within a block runs in $O(klog(\frac{n+m}{k}))$ [4]. For more information about sets union and difference, please refer to Blelloch and Reid-Miller [4]. Please note that Bloom filter can be also applied for parallelization, but more straightforward [7].

3. Conclusions

There are many approaches to solve our problem of Set Operations. Unfortunately, a single algorithm is not known for each weather. However, we have a very big bundle of interesting ideas and research directions. We can use algorithms from very simple to very sophisticated ones. We faced some issues in binary searching and hash mapping. If we deal with a huge amount of data some probabilistic and parallelization ideas must be concerned. It is also necessary to predict the number of insertion and search operations before we make the decision of which data structure is most suitable. A summary for all algorithms is depicted in **Table 3**.

Table 3 Table of algorithms' complexities based on several approaches. Legend: O(n) in blue – expected time complexity.
* $n = \alpha m$, $\alpha > 1$. ** $e = \frac{l}{\varepsilon}$, where ε is false positive rate.

Algorithm based on	Intersection time complexity	Construction time	Space complexity
Bitwise operations	$O(\min\{n, m\})$	$O(n+m)$	$O(\log\max\{a_i, b_j\})$
Brute force searching	$O(nm)$	$O(n+m)$	$O(n+m)$
Binary searching on vectors	$O(nlogm)$	$O(n+m)$	$O(n+m)$
Searching on sets	$O(nlogm)$	$O(n+mlogm)$	$O(n+m)$
Binary searching Baeza-Yates	$O(nlogm)$ $O(n)$*	$O(n+m)$	$O(n+m)$
std::intersection /difference, Merge algorithm	$O(n+m)$	$O(n+m)$	$O(n+m)$
Hash function	$O(nm)$ $O(n)$	$O(n+m)$	$O(n+m)$
Perfect hashing	$O(n)$	$O(n+m^3logq)$ $O(n+m)$	$O(n+m)$
Bloom filters	$O(n+k)$	$O(n+k)$	$O(n+mloge)$**
Treaps	$O\,(mlog\,\frac{n}{m})$ Parallel union: $O(klog(\frac{n+m}{k}))$	$O(n+m)$ [24]	$O(n+m)$

References

[1] "Algorytmy i struktury danych" Banachowski Lech, Diks Krzysztof, Rytter Wojciech
[2] "Algorithmic Engagements 2014, task: Kółko informatyczne" Tomasz Czajka https://sio2.mimuw.edu.pl/c/pa-2014-1/s/43890/source/
[3] "Introduction to Algorithms" Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest
[4] "Fast Set Operations Using Treaps" Guy E. Blelloch, Margaret Reid-Miller
[5] https://en.wikipedia.org/wiki/Hash_table, https://en.wikipedia.org/wiki/Bloom_filter
[6] "An Optimal Bloom Filter Replacement Based on Matrix Solving", Ely Porat
[7] "Theory and Practise of Bloom Filters for Distributed Systems" Sasu Tarkoma, Christian Esteve Rothenberg, and Eemil Lagerspetz
[8] "On Mobility Load Balancing for LTE Systems" Raymond Kwan, Rob Arnott, Robert Paterson, Riccardo Trivisonno, Mitsuhiro Kubota
[9] "Load Balancing in Downlink LTE Self-Optimizing Networks" Andreas Lobinger, Szymon Stefański, Thomas Jansen, Irina Balan
[10] "Mobility Load Balancing Scheme based on Cell Reselection" Toshiaki Yamamoto, Toshihiko Komine, Satoshi Konishi, 2012, 8th International Conference on Wireless and Mobile Communications
[11] "Hash, displace, and compress", Djamal Belazzougui, Fabiano C. Botelho, Martin Dietzfelbinger
[12] "An experimental Investigation of Set Intersection Algorithms for Text Searching" Jérêmy Barbay, Alejandro López-Ortiz, Tyler Lu, and Alejandro Salinger
[13] "Design of distributed and autonomic load balancing for self-organization LTE" Heng Zhang, Xuesong Qiu, Luoming Meng, Xidong Zhang
[14] "Storing a sparse table with O(1) worst case access time", Michael L. Fredman, János Komlós, Endre Szemerédi
[15] "A Fast Set Intersection Algorithm for Sorted Sequences" Ricardo Baeza-Yates
[16] "Experimental Analysis of a Fast Intersection Algorithm for Sorted Sequences", Ricardo Baeza-Yates, Alejandro Salinger
[17] "Binary Search Is a Pathological Case for Caches" Paul Khuong
[18] "Introduction to Probabilistic Data Structures" Yin Niu, https://dzone.com/articles/introduction-probabilistic-0
[19] https://www.topcoder.com/community/data-science/data-science-tutorials/binary-indexed-trees/
[20] https://github.com/nokia-wroclaw/nokia-book/tree/master/03
[21] Open Bloom Filter, http://www.partow.net/programming/hashfunctions/
[22] "Uniquely Represented Data Structures with Applications to Privacy" Daniel Golovin
[23] http://opendatastructures.org/ods-java/7_2_Treap_Randomized_Binary.html
[24] "Linear-time construction of treaps and Cartesian trees" Mark Allen Weiss

About the author

My computer adventure began at the age of 8 and just two years later I started programming interactive video games in Basic language. I have participated in many maths and programming competitions and decided to widen and strengthen my math and computer skills by attending and graduating in Computer Science at the University of Wrocław. I have interests that span from the theory and mathematics to algorithms and data structures to application and software. I am also a TopCoder algorithm member since March 2005.

Łukasz Grządko
Senior Engineer, Software Development
MN LTE

Theory Meets Practice: Register Allocation in Minimal Compiler

Michał Bartkowiak
Engineer, Software Development
MN LTE

Paweł Wieczorek
Engineer, Software Development
MN LTE

1. Introduction

Within the previous edition of Nokia Book [1] we learned how to build a toy compiler for a very simple language. Our compiler was able to transform source code directly into a sequence of x86 assembly instructions. To keep the translation process as simple as possible, straightforward rules were used for handling local variables and the results of intermediate computations. As a consequence, we obtained an easy to understand compiler. However, the produced assembly code was of low quality – it contained many unnecessary transfers between memory and registers.

This issue constitutes a great opportunity to present an advanced topic – the register allocation problem. The problem is how to use hardware registers in produced code to minimize register and memory transfers. We base this article on Chaitin's papers about the subject [2] [3], where the author discovered that the problem can be seen in terms of graph coloring. An improved variant, known as the Chaitin-Briggs algorithm [4], is still used in leading compilers like GCC. Jumping from fundamentals directly to the advanced topic requires a brief introduction to machinery like intermediate representation (IR) and static analysis. We present these topics superficially and encourage the reader to visit Nokia Book's source code repository [5] where one can find the updated MiNiK compiler equipped with detailed comments and further references. We strongly advise to read the article alongside the source code of the MiNiK compiler.

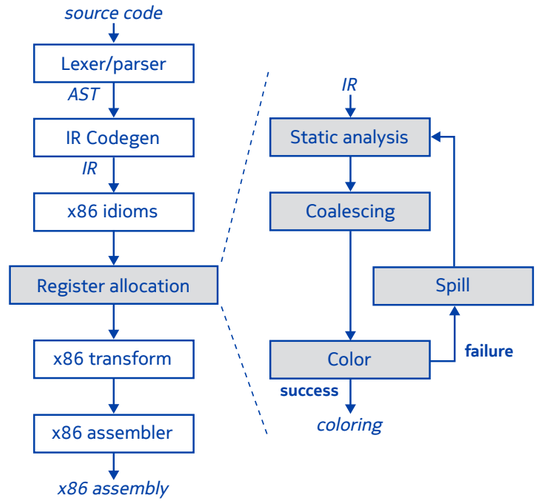
2. Pipeline overview

The first version of the MiNiK compiler [1] was an example of an extremely simple compiler. It provided direct translation of MiNiK source code to x86 assembly. In the current version of MiNiK compiler [5], we start dealing with a nontrivial pipeline. Its overall structure is presented in **Figure 1**.

After creation of the abstract syntax tree (AST), the new code generator produces an intermediate representation for each of the functions which constitute the MiNiK program. Next we perform IR transformation which makes the target machine (x86) idiosyncrasies explicit. From the IR we can build control flow graphs (CFG) for every function and perform static analyses on them. The output of the static analysis for a single CFG forms the input of an iterative process of register allocation.

The first step of the register allocation process is register coalescing, which eliminates register-to-register moves. The IR can be modified at this point, so the CFG has to be rebuilt and static analyses need to be run again. Finally, we try to perform coloring. If it succeeds then the allocation calculation is finished. Failure at the coloring stage means that some local variables or results of intermediate calculations have to be kept in memory. In that case an additional procedure called register spilling is performed, and the register allocation process is repeated.

Figure 1 Overview of MiNiK compiler's pipeline



Having the registers allocated and spills determined for each function, we can perform the final steps. The IR registers are pinned to the x86 registers. Final transformations of the IR are performed, e.g. to take into account the functions' calling convention. Lastly, the IR is converted to x86 assembly code.

3. Tools

3.1. Intermediate representation

Compilers usually do not perform analysis of and transformations on the source language due to its complexity. Instead, intermediate languages are used to allow compilers to understand and manipulate computer programs easily. Thus, the main characteristic of the IR is its simplicity: it resembles the way that the processor executes its instructions. To fulfill our main goal we require the IR to contain an infinite number of temporary registers. They can be assigned to any data used by the source program. This is the second crucial feature of the IR: every intermediate result is assigned to a fresh

temporary register. Later, the registers are exchanged for machine registers or memory locations. The final aspect is that despite the IR abstractions, it has to be adequate to handle the target's idiosyncrasies which could impact register allocation.

In our case the first step of the translation from source language to intermediate one is the generation of one or more IR instructions from each MiNiK operation. For example, a division operation can be a basic part of some elaborate expression. During the translation, a simple **div** instruction is generated. It takes two arguments (registers) and leaves its result in the register provided as a first argument. This register can be later used in the operations constituting the expression.

The second step handles the target's (x86) idiosyncrasies. In the case of the exemplary **div** instruction, we need to take into account that arguments have to be stored in the **EAX** and **EDX** registers and the result is written to the same registers. Such peculiarities have to be considered during the design of the intermediate language.

Due to the aspects described above, our IR needs to be able to differentiate between:

- temporary registers (**%tN**): the number of them is not constrained and they will be the actual subject of the register allocation process
- placeholders for any general purpose hardware register (**%hN**): these registers appear during the register allocation process and their number is constrained by the number of available hardware registers
- general purpose hardware registers (**%HN**): each register denotes a particular hardware register
- special purpose hardware registers (**%SN**): some hardware registers are present in the IR code but we cannot use them for storing results of computations

General-purpose hardware registers can be one of: **EAX**, **EBX**, **ECX**, **EDX**, **ESI**, and **EDI**. The set of special registers is restricted to **EBP** and **ESP**.

In **Listing 1** we can see how a piece of code is transformed to the general intermediate representation. Each variable, constant or result of intermediate calculations uses a separate temporary register **%tN**. During the pass responsible for handling x86 idiosyncrasies, the **div** instruction is transformed into four other instructions. Their goal is to explicitly state where operands and the results of x86's **DIV** instruction are placed.

3.2. Static analysis

To be able to implement the desired register allocation algorithm we require basic tools to allow the compiler to understand what is going on during execution of a given program.

Listing 1 Two-step transformation of MiNiK source code to intermediate representation

```
var x := 10

while(x > 1)
{

  x := x/2

}
```

```
const %t1 10
mov %t0 %t1
L0:
const %t2 1
if leq %t0 %t2
   goto L1
const %t3 2
mov %t4 %t0
div %t4 %t3


mov %t0 %t4
goto L0
L1:
```

```
const %t1 10
mov %t0 %t1
L0:
const %t2 1
if leq %t0 %t2
   goto L1
const %t3 2
mov %t4 %t0
mov %H0 %t4
const %H3 0
divmod %H0 %H3 %t3
mov %t4 %H0
mov %t0 %t4
goto L0
L1:
```

A control flow graph represents all paths that might be traversed during execution of the given function. Normally, each vertex in the graph symbolizes a block of instructions without any jumps or labels (i.e. targets of a jump). For the sake of simplicity we took the convention that the block consists of a single instruction or label. This way the construction of the CFG from the IR is straightforward: every instruction and label is represented by a single vertex, and edges connect consecutive (in the sense of flow) instructions. Additional edges are added between the source and target of each jump. Moreover, the CFG is supplemented with isolated entry and exit vertices which aggregate entry and all exits from the function respectively. The graph provides sufficient granulation for static analyses. We can see the elements of the example CFG in **Figure 2**. It is created from the IR's instructions presented in **Listing 1** in the rightmost column.

The main benefit of having a representation of how the control goes through the program is the ability to perform data flow analysis (DFA). It is direct compiler tool for understanding the program. The compiled program may have branches and loops so the desired data cannot be computed in a straightforward way. Instead, we use a framework where we compute data in a loop. We start with the initial data. Then, during each iteration, we extend the already computed knowledge about the program. The iterations are repeated until knowledge is saturated. Behind the design of such computations lies advanced mathematics and for details we refer the reader to the classical textbook on the subject [6]. The result of the analysis is data computed for entry and exit of each vertex present in the function's CFG.

Figure 2 Example of control flow graph

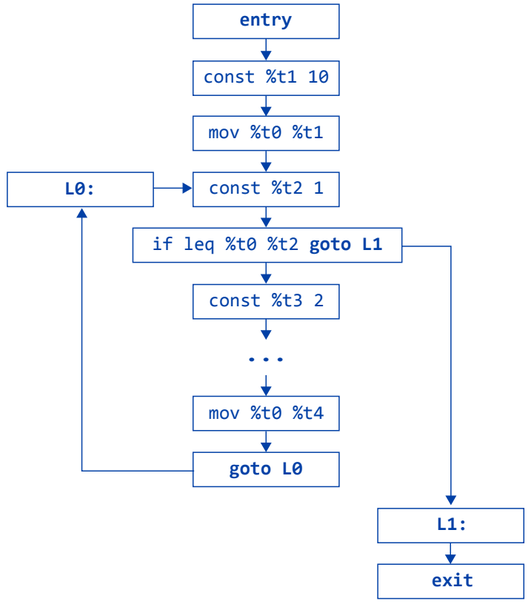
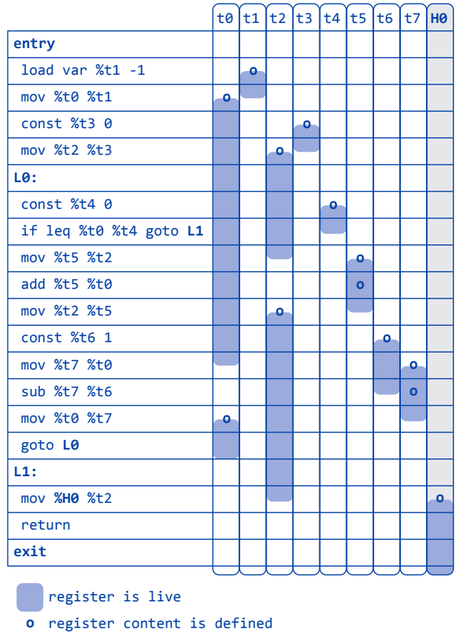


Figure 3 Initial result of live variable analysis



The register allocation algorithm performs a DFA known as live variable analysis. Formally speaking, for each point in the function we compute a set of IR registers which could be used before being modified on a control path starting from the point. In other words, it computes the set of registers whose content may be used during further execution before being overwritten.

In **Figure 3** we present the result of analysis for the given IR program. Each column shows the "liveness" of a particular IR register present in the code and each row represents knowledge related to a particular instruction. The upper or lower half of a cell is marked when a particular register is live at the instruction's entry or exit respectively. A circle in the cell is drawn when the register's content is overwritten. A good example is in the second row where the content of a register is loaded from memory. The next instruction copies it to register **%t0** and the previous register is not used anymore. The short live range in the column **%t1** reflects this behavior.

Liveness looks like a simple linear range, but it is a false impression. All possible control paths, with all branches and loops, are taken into account. Pay attention to the live range of the register **%t0** – at first glance it looks like two simple ranges, but actually it is one complex structure. It has two start points: one before the loop and the second one at the loop's body ending. From both starting points, control goes to the beginning of the loop's body. Keep in mind also that the **%H0** (x86 **EAX**) register is marked as alive after function exit – it holds the return value.

4. Register allocation

Let us start with the graph coloring problem. It involves coloring vertices in a graph in such a way that two vertices connected by an edge cannot get the same color. Additionally, the number of available colors is constrained. It is a commonly known NP-complete [7]

problem in computer science. It means that we do not know an algorithm which could always find such a coloring or decide if it exists in a reasonable time for any graph. Such abstract problems in complexity theory seem to be not related to the real world, but they actually reflect problems occurring in our daily work.

Chaitin has shown that the register allocation problem is equivalent to graph coloring [2]. As a consequence, it is not known how to construct a compiler which is able to find (in reasonable time) the best usage of machine registers for any source program. So compiler constructors make tradeoffs when choosing an algorithm for register allocation. For example, in languages like Java, where just-in-time compilers are used, it is common to choose an algorithm producing worse code to get better compilation time [8]. Due to the limited count of available machine registers, the compiler can encounter a situation where it is impossible to keep all local variables in them. In such situations, the compiler has to decide how to use computer memory to efficiently handle intermediate results. The problem of minimizing the cost of memory transfers due to spills is also NP-complete.

4.1. Interference graph

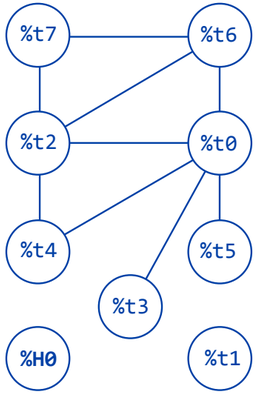
Chaitin's algorithm uses live variable analysis to construct an interference graph (IG) of a given IR program. In the graph, vertices represent the IR's registers and two vertices are connected by an undirected edge if one of them is live when the second one is modified. In other words, the edge signifies that two IR registers cannot occupy the same hardware register, because there exists a fragment of the program where the content of both registers is required and one cannot overwrite the other.

In **Figure 4** we see an interference graph built for the program from **Figure 3**. Recall that liveness of the register **`%t0`** intersects with liveness of other registers, e.g. **`%t2`** or **`%t4`**. It is clear that these registers cannot occupy the same hardware register. Those constraints are reflected in the graph by edges between **`%t0`** and the other registers.

The graph coloring problem does not allow neighboring vertices to get the same color. In the IG we put edges between registers which interfere. Therefore, seeing hardware registers as colors allows us to state the register allocation problem in terms of graph coloring. In x86 hardware we have six general purpose registers. Thus, finding a coloring of the IG where at most six colors are used, allows us to safely exchange IR registers with hardware ones.

Let us think of the x86 idiom about division from **Listing 1**. Presence of the **`divmod`** instruction brings hardware registers **`%H0`** and **`%H3`** (x86 **`EAX`** and **`EDX`** respectively) into use. In that situation we have to ensure that those registers get distinct colors

Figure 4 Example of interference graph



in the IG. We can easily prevent unwanted coloring by adding edges between all hardware registers present in the given intermediate code.

4.2. Register coalescing

The intermediate code produced by the code generator contains a lot of temporary registers and copying between them. Often one register is used only to be copied to another one, like the **`%t1`** or **`%t3`** registers from **Figure 3**. This enlarges the interference graph and so can slowdown compilation time.

The register allocator contains an additional pass where temporary registers are coalesced into one. This optimization is also known as register subsumption or variable propagation. The algorithm iterates over all **`mov`** instructions present in the given code and checks if the source register goes dead after instruction. We can interpret that situation as a renaming of the source register to the destination one. We coalesce registers only if the degree of coalesced registers will not exceed the number of available colors. After that we can remove unnecessary **`mov`** instructions.

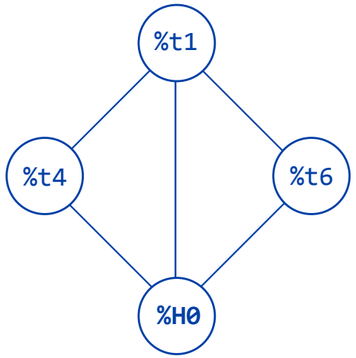
In **Figure 5** we present a liveness analysis of the program from **Figure 3** after the register coalescing pass. The resulting simplification is significant. Observe that the simplified code computes its result (return value) immediately in the %H0 register. In **Figure 6** we can see how variable propagation affected the interference graph.

Figure 5 Liveness analysis example after register coalescing

	t1	t4	t6	H0
`entry`				
`load var %t1 -1`	o			
`const %H0 0`				o
`L0:`				
`const %t4 0`		o		
`if leq %t1 %t4 goto L1`				
`add %H0 %t1`				o
`const %t6 1`			o	
`sub %t1 %t6`	o			
`goto L0`				
`L1:`				
`return`				
`exit`				

register is live
o register content is defined

Figure 6 Interference graph after register coalescing



4.3. Graph coloring

As we mentioned previously, the graph coloring problem is NP-complete. Instead of an exact solution, we use a simple heuristic which tries to find an acceptable coloring in linear time and space.

If a vertex has a smaller degree than the desired number of colors then it can be colored easily. No matter what color its neighbors will get, it is always possible to choose another color without collision. This observation allows us to conclude that coloring of the graph can be reduced to the coloring of a smaller graph where that vertex is removed. If we find a coloring for the smaller graph then we are able to assign a color to the removed vertex.

It is a crucial observation used during register allocation via graph coloring. The heuristic picks up a vertex with lower degree than the number of available colors and pushes it onto a stack. Then it executes recursively on the smaller graph. In the best case, the whole procedure will end up with an empty graph, which we can treat as already colored. When the procedure returns from the recursion it restores a vertex and assigns a color to it.

A problem arises when there are no more vertices with degree lower than the desired amount of colors and the graph is still not empty. We cannot easily predict if those vertices can be colored. In those situations the algorithm chooses a vertex and marks it as a potential spill, then continues normally. When the procedure returns from recursion and it is not possible to pick a color for the vertex, then the coloring heuristic is aborted and one temporary register has to be spilled.

4.4. Register spilling

When the heuristic is not able to find a coloring of the interference graph, the spill pass is executed. One of the temporary registers is selected and memory on stack (local variable) is allocated. The intermediate code is rewritten to load content from memory before each use of the temporary register and store content to memory after each modification. This transformation splits the live range of the register into smaller pieces and thus removes many interferences. After the spill code is inserted, the whole register allocation procedure is repeated.

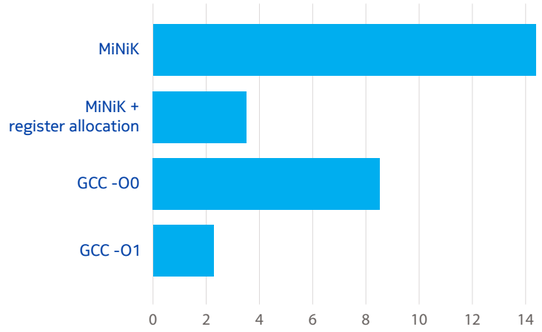
Note that spilling a register into memory introduces register/memory transfers which slow down the compiled program. Thus, to select the best register we need to estimate how much spilling a particular register costs. We could use a simple formula for counting each usage and modification of the register, but it would not distinguish registers used inside loops. Instead, we compute heuristics for all registers marked as a potential spills. Static analysis is used to detect loops in the intermediate code and each use or modification is

counted as 10^d, where d is the loop nesting level. This example heuristic allows us to determine variables that are heavily used inside loops in simple programs.

5. Summary

We constructed a program to check how register allocation can affect the performance of the code produced by our compiler, which does not perform any other optimizations. It checks if a given number is prime, but the used algorithm is very inefficient in order to produce tangible differences in execution time. In **Figure 7** you can see how much time is consumed by the program to check if the number 97579267 is prime. We implemented the program also in C language to compare our results with the GCC 5 results.

Figure 7 Performance comparison



Notice that adding register allocation to the MiNiK compiler allowed it to produce a program that was four times faster. Additionally, the produced code is two times faster than code produced by the GCC compiler with all optimizations turned off. These results shows how expensive transfers between registers and memory are and how important is the register allocator as part of the compiler.

Through the years a lot of improvements have been made to Chaitin's original algorithm. For further reading we recommend to start with Briggs [4].

References

[1] M. Bartkowiak, "Beginning the Adventure: Writing a Minimal Compiler," *Nokia Book II,* 2015.

[2] G. J. Chaitin, M. A. Auslander, A. K. Chandra, J. Cocke, M. E. Hopkins i P. W. Markstein, „Register Allocation via Coloring," Yorktown Heights, NY, 1980.

[3] G. J. Chaitin, „Register Allocation and Spilling via Graph Coloring," Yorktown Heights, NY, 1982.

[4] P. Briggs, „Register Allocation via Graph Coloring," Houston, TX, 1992.

[5] M. Bartkowiak i P. Wieczorek, „Nokia Book's source code repository," Nokia, 2016. [Online]. Available: https://github.com/nokia-wroclaw/nokia-book/tree/master/03

[6] F. Nielson, H. R. Nielson and C. Hankin, Principles of Program Analysis, Springer, 2004.

[7] M. Sipser, Introduction to the Theory of Computation, Thomson Course Technology, 2006.

[8] M. Poletto i V. Sarkar, „Linear Scan Register Allocation," *ACM Transactions on Programming Languages and Systems,* tom 21, 1999.

About the authors

I work as Software Development Engineer in MN LTE C-Plane's K3 project. In order to facilitate testing of software components we are creating a compiler, runtime environment and tools for Testing and Test Control Notation version 3 (TTCN-3) programming language. TTCN-3 is standardized by the ETSI and is aimed to provide well-defined syntax for writing tests.

Michał Bartkowiak
Engineer, Software Development
MN LTE

I work as Software Development Engineer in MN LTE C-Plane's K3 project. In our daily work we are challenged not only by C++ or Python quirks, but mainly by the tasks related to the creation of a compiler for over a million-line TTCN-3 codebase. We are trying to use academic knowledge to identify and solve problems in our project.

Paweł Wieczorek
Engineer, Software Development,
MN LTE

Who we are

Nokia is a global leader in the technologies that connect people and things. Powered by the innovation of Bell Labs and Nokia Technologies, the company is at the forefront of creating and licensing the technologies that are increasingly at the heart of our connected lives.

With state-of-the-art software, hardware and services for any type of network, Nokia is uniquely positioned to help communication service providers, governments, and large enterprises deliver on the promise of 5G, the Cloud and the Internet of Things.

Our vision:

We've always been excited by where technology will lead us.

We continue to reimagine how technology blends into our lives, working for us, discreetly yet magically in the background, enriching our lives.

We're purposefully shaping the new revolution in connectivity to positively impact people's lives each day: where networks, data, devices seamlessly integrate and are effortless to use; where technology is designed with humanity, integrity and dependability at heart.

What makes us uniquely Nokia?

We have always been purposefully driven by our values, and have a strong legacy of designing technology for people, to meet real human needs. Technology designed to serve mankind, not the other way round. Effortless, Intuitive, simple, and very human technology.

It is our integrity and the dependability of our technology, design for people, that makes us uniquely Nokia.

Our strengths:

Technology that thinks ahead: We deploy self-managing technology that works invisibly and magically in the background, intelligently learning and adapting to anticipate people's needs, re-shaping dynamically to fulfill them.

Making sophisticated technology simple:
We select, create, and apply technology thoughtfully. Our considered approach ensures our technology seamlessly integrates, and is effortless and intuitive to use, for both customers and end users.

Integrity of design and execution:
Our technology, networks, and data are resilient and dependable. Privacy and security are built in from the start, not an afterthought. We are a values-driven business, and quality is designed into everything we do, from our systems, processes, and software interfaces, to the service we provide to customers.

Grounded in real life:
We are realistic about how technology works for people in their lives each day. Our innovation focuses on meeting real human need, using technology to positively impact the everyday experience of people.

Strzegomska Street 36
53-611 Wrocław
Reception 1st Floor, Green Towers B
Opening hours 8:00 – 18:00
Phone: +48 71 777 3800
Fax: +48 71 777 30 30

Strzegomska Street 54a
53-611 Wrocław
Reception Ground Floor, Wrocław
Business Park
Opening hours 8:00 – 18:00
Phone: +48 777 38 01

Bema Street 2
50-265 Wrocław
Reception Entrance A, 4th Floor
Opening hours 8:00 – 18:00
Phone: +48 71 777 41 30
Fax: +48 71 777 41 98

Lotnicza Street 12
54-155 Wrocław
Reception Ground Floor, West Gate
Opening Hours 8:00 – 18:00
Phone: +48 71 777 4002
+48 71 777 4001

e-mail: kontakt@nokiawroclaw.pl
www.nokiawroclaw.pl