**UNIVERSITÀ DEGLI STUDI DI PARMA**

*Dottorato di Ricerca in Tecnologie dell'Informazione*

*XXVIII Ciclo*

# Security in the IoT: from energy-constrained nodes to cellular network architectures

Relatore:

*Chiar.mo Prof. Gianluigi Ferrari*

Tutor:

*Chiar.mo Prof. Luca Veltri*

Coordinatore:

*Chiar.mo Prof. Marco Locatelli*

Dottorando: *Andrea Giuseppe Forte*

Gennaio 2017

*A mia madre e a mio padre, loro sanno.*

# Table Of Content

# List of Figures

# List of Tables

# Introduction

In recent years the research community has focused on the *Internet anywhere, anytime* that is, providing network connectivity at the IP layer regardless of the time of day and the location of users. Today, we can say that such a goal has been reached first and foremost thanks to the many technological advances in wireless and mobile networks. Standards bodies such as the Internet Engineering Task Force (IETF) and the 3rd Generation Partnership Project (3GPP) have spent a substantial amount of time standardizing technologies and protocols aimed at making IP connectivity for mobile devices available all the time, everywhere. Work on handoff has been particularly important as it has allowed users to maintain network connectivity regardless of their movement either within the same wireless technology (i.e., horizontal handoff) or between heterogeneous wireless technologies (i.e., vertical handoff).

Recently, the research community has enhanced the *Internet anywhere, anytime* paradigm by adding the word *anything*. This represents a significant shift from today's Internet as it implies that things and not just humans will be "users" of the Internet. Network connectivity will be available anywhere, at anytime to anything. This process, however, has not just started but it is well underway. We already have many "things" that, without any human intervention, not only consume but also produce information and perform actions based on such information. This is exactly what we refer to when we talk about the Internet of Things (IoT). Things can now communicate with each other, perform complex actions and take decisions in an autonomous way. Machine-to-machine communication, Radio Frequency Identification (RFID) networks, wireless sensor networks, all represent a part of the IoT.

On the commercial side, Microsoft Home OS [1], Microsoft OnX [2] and AT&T's Digital Life [3], represent the first attempts to bring the concept of IoT to consumers.

How do we enable IoT? Currently, there are two approaches. The first approach wants every object to communicate over the Internet Protocol (IP) with other objects and with the Internet, either directly or via a proxy server; the second approach sees many objects as "dummy" thus not capable of communicating over IP. Dummy objects would communicate with smarter objects such as gateways and proxies that would then communicate over IP with the outside world, relaying information to and from the dummy objects.

The first approach is largely supported by the IETF. In particular, the Internet Protocol for Smart Objects (IPSO) alliance demoed a testbed for IoT at two IETF meetings (i.e., IETF 84 and IETF 85), showing how it is possible to run a full IP network stack on extremely small embedded devices by leveraging early versions of the protocols for IoT that the IETF is currently working on. Also, until a few years ago, assigning a unique IP address to every possible object would have been an impossible task given the 32-bit address space of IP version 4 (IPv4) [4] and the consequent scarcity of IPv4 addresses. Today, with IP version 6 (IPv6) [5] this is no longer an issue. IPv6 has a 128 bit address space thus being able to provide a unique IPv6 address for virtually every object in the world.

Choosing IP as the network protocol, however, is not always an obvious choice. In the IoT there will be devices with extremely low energy levels and low computational power possibly deployed in very hostile environments where communication can only be multi-hop, with very low bit-rates and with very high packet loss. Assuming such devices would include a network layer at all, a network protocol with less overhead may be required. The most realistic scenario, however, is that aside from some very specific energy-constrained devices, most devices will use IP over different link and physical layers. Running IP over low-power networks presents several challenges which the IETF is trying to solve.

Generally speaking, the main problems we have to face in the IoT are very low energy consumption, on-off network connectivity usually in a multi-hop configuration, very low computational power. Some of these problems, however, are not new as

they are present also in existing technologies like Delay Tolerant Networks (DTNs) and Wireless Mesh Networks, for example. The significant difference between these technologies and the IoT is the low-power nature of IoT. Because of the strict requirements on power consumption, many techniques such as "store and forward" or "flood routing" used in other technologies, cannot be used in the IoT. Given the limited amount of energy available, each device in the IoT will be mostly selfish in behavior, willing to cooperate only if strictly necessary. In particular, devices will collaborate if there is no other choice or only on those tasks where energy consumption can be minimized by cooperation.

# Chapter 1

# Activities and Research Opportunities in the IoT

## 1.1 Introduction

The Internet of Things (IoT) will change the way the Internet looks today. Users of the IoT will not just interact with documents and different media distributed on connected server but will interact with devices interconnected in the real world. The Internet of the future will be the Internet of things where everything is connected and "things" can consume and produce information autonomously. In this chapter, we present a high-level overview of the different efforts going on in the IoT research community, pointing out the general areas of research where we think the main challenges and opportunities are.

## 1.2 IoT Activities in the IETF

All the protocols standardized by the IETF over the years were not engineered for networks with low-power requirements. Because of this, all those protocols and in particular IPv6 would not "just work" when applied to the IoT and its low-power

networks. Currently, within the IETF, there are several working groups that are trying to solve many of the challenges introduced by the IoT.

The IPv6 over Low-Power Wireless Personal Area Networks (6LoWPAN) working group is defining an adaptation layer between network layer and link layer so to be able to use the IPv6 protocol over low-power networks such as IEEE 802.15.4 [6]. When considering an IEEE 802.15.4 network, there are several issues that prevent IPv6 from "just working". Let us look at some of them.

- In IPv6 the minimum MTU size has been increased from 576 bytes of IPv4 to 1280 bytes. This is a problem for IEEE 802.15.4 where the frame length is limited to 127 bytes in order to ensure low packet error rates. Furthermore, because of the different headers and depending on the type of link-layer security enabled, the actual payload of a IEEE 802.15.4 MAC frame can be as small as 80 bytes [7]. In order to address this issue, the 6LoWPAN adaptation layer introduces stateless hop-by-hop header compression and fragmentation.[1]

- Because transmission power increases polynomially with the transmission range, the transmission range is usually short, on the order of tens of meters or less. This implies that low-power devices must communicate in a multi-hop fashion. In order to address the multi-hop nature of low-power communication and the lack of a single broadcast domain, the 6LoWPAN adaptation layer supports both layer-two and layer-three forwarding of IPv6 datagrams.

- Lastly, in IEEE 802.15.4 networks, nodes spend a good amount of time in sleeping mode to save energy. During this time communication cannot take place and this creates problems for things such as routing.

The Routing Over Low power and Lossy networks (ROLL) working group is looking at all those routing issues that are specific to networks characterized by a high degree of packet loss, low-power and low bit-rate. The Routing Protocol for Lossy networks (RPL) [8] is trying to address these very issues. RPL builds different routing graphs for the same network by taking into consideration different metrics and

[1] In IPv6 fragmentation is end to end in order to have high performance routers and low complexity.

constraints. For example, we might want to know paths with the best expected transmission values while avoiding unencrypted links or perhaps paths with the lowest latency while avoiding nodes operating on batteries. Also, one key point of RPL is slow reaction time. When some paths are lost, the routing protocol should not try to immediately find alternative paths so to re-route around link failures but it should wait and see if those paths recover. This is due to the fact that in lossy networks such as the ones we have in the IoT, lossyness is very transient and reacting too fast to path failures may introduce instability.

Finally, at the application layer, the Constrained RESTful Environments (CORE) working group is standardizing the Constrained Application Protocol (CoAP) [9]. Similar to a binary version of the Hypertext Transfer Protocol (HTTP) [10], CoAP has been designed considering devices with low-power constraints communicating over a very lossy wireless medium.

## 1.3 IoT Devices

The Internet of Things will include different types of devices with very diverse constraints. On one end of the spectrum there will be devices with a powerful CPU and no energy constraints, while on the other end of the spectrum there will be devices with very low computational power and strict energy requirements. In this section we look at this second class of devices. In particular, we give a brief overview on a new type of devices that for their unique properties present their own challenges and research opportunities.

The Energy Harvesting Networked Active Tags (EnHANTs) [11] are a new class of networked devices with the most stringent requirements in terms of energy consumption and computational power. In particular, we can think of these devices as something in between RFID tags and wireless sensors. EnHANTs are different from RFID tags as RFID tags can communicate, either passively or actively, with an RFID reader but cannot communicate with each other. On the other hand, EnHANTs are different from wireless sensors as they must operate at extremely low energy levels and exchange only ID information.

EnHANTs are very small, flexible and networked in a multi-hop configuration. The key difference with other technologies is that they generate the energy they use by harvesting it from light and movement using organic solar cells and piezoelectric materials. Given that the only energy they can use is the energy they can harvest, power consumption must be the smallest possible. In particular, network connectivity is provided by Impulse Radio Ultra Wide Band (IR-UWB) technology [12]. In IR-UWB, devices communicate by transmitting impulses with durations on the order of nanoseconds and then sleep between impulses so to use as little energy as possible. In doing so, energy consumption is on the order of the nano Joules per bit, range is on the order of 1 to 10 meters and bit-rate is between 0.1 and 1 Mbps [11].

Since EnHANTs have to spend most of their time sleeping, high-precision clocks cannot be used to coordinate transmissions. Low-precision clocks can be used but are very inaccurate especially if we consider transmission times on the order of nano seconds. Because of this, in order to receive data, EnHANTs have to stay awake and "listen" to the medium for longer than a few nano seconds, thus using a significant amount of energy. That is, the energy required for transmitting is smaller than the energy required for receiving. This is clearly the opposite of what happens in typical communication systems where the energy for transmitting a signal always exceeds the energy used for receiving a signal. Because of this paradigm change, new link and network layer protocols are required. For example, the decision of a tag to send some data needs to be coordinated with the receiver so to consider the energy levels of the receiver. This problem becomes even more complex if we think that network connectivity is multi-hop and as such many tags may have to receive packets just to forward them to other tags. How to coordinate the sending and receiving of data with the aforementioned power constraints remains an open issue.

## 1.4 Research Opportunities

IoT represents a big shift from how the Internet looks today. Today the Internet is mostly a collection of documents and media files scattered on a very large number of servers which users can search and use. With the Internet of Things everything

will be connected. This means that real objects will be at the same time part of the Internet as well as users of the Internet thus opening the door to radically different applications.

In the following we list some of the key areas where we think there are interesting problems to solve. Security, however, will be discussed separately in the next section.

**Databases.** The Internet will go from some million servers connected with each other, storing documents and media files, to billions of interconnected devices storing but also *generating* content at a very fast pace. The amount of information generated by all of these devices will represent a significant challenge from a database point of view. The amount of information collected at any given time will be so large that it is not clear if current systems will be able to handle such a load and, more importantly, handle queries on such a large amount of data. Relational databases may not be the right model and database parallelization as we do it today may not be enough. Although work on this topic has already started [13], a clear solution has yet to be found.

**Discovery.** Given the extremely large amount of interconnected devices that IoT will expose to users, discovery of such devices will become an important aspect of IoT. Nobody will be able to use devices that are not discoverable. Protocols used today for automatic discovery such as Apple's Bonjour [14] are usually based on multicast transmissions. Multicast packets in wireless networks (e.g., IEEE 802.11) are typically transmitted at the lowest bit-rate so to guarantee their delivery. Unfortunately, this means that power consumption is higher for such packets hence making such discovery protocols not suitable for devices with stringent power requirements. Furthermore, a discovery protocol for IoT should take into account the fact that devices in the IoT will spend a large part of their time sleeping and as such would not always be reachable. Research is needed on low-power discovery protocols that can scale over a very large number of nodes.

**Search.** In the search realm, IoT will enable new and exciting ways to search, allowing users to search for things they could have not searched for before. For exam-

ple, users will be able to query for *"what brand of coffee was I drinking two days ago when I went out with my friend Bob?"*. This will be possible because the mug the user drunk coffee from had IP connectivity and was able to communicate with his cellphone sending information on the beverage he was about to drink. This was correlated to information from his phone calendar (e.g., "Today at 2PM going out with Bob") and to his location-based service registering his location for that meeting. Similarly, Alice could query for *"what is the price of the shoes my friend Mary was wearing at the party last night?"*. These examples show how IoT will enable users to do much more than what they can do today. However, it also shows how critical privacy will become with the IoT, even more than now. We will address privacy in a separate section.

**Management.** As any system administrator knows, managing a large number of machines can be very problematic to say the least. The IoT with a virtually unbounded number of devices will bring the management issue to a whole new level. How do we provision such devices? How do we send patches and updates without compromising nodes functionalities or worse, draining all their power? How do we accomplish all of this in a secure way? These are all problems that need to be addressed.

**Service Interaction.** Many of the devices in the IoT will not necessarily be used by one service at the time. It is very likely that multiple services will run at the same time and in doing so, will interact with the same devices in conflicting ways. A typical example is the home-alarm service turning on lights and TV periodically to simulate people in a house and a power-saving service trying to save as much power as possible by turning off lights and TV. Clearly, the two services have rules that conflict with each other, hence the problem. To solve this problem, priorities have been proposed as a solution [15, 16]. Giving different priorities to different services seems to solve this problem. Using just priorities, however, does not seem to be the best solution as it makes every decision binary based on the service priority that is, the service with the highest priority always wins. In a more IoT-like approach, different services could "talk" to each other and coordinate their actions so that, for example,

a home-alarm system agrees to turn the TV on and off only so often to partially satisfy the requirements of the power-saving service. In another example, the climate control service "tells" the home-alarm service that it is turning on the fan so not to trigger an alarm based on movement. These and other approaches need to be further investigated.

**Power Awareness.** All services deployed within the IoT will have to be power aware. A service should never exhaust a device by enforcing rules that would drain all the power from that device. For example, a service enforcing a high sampling rate on a sensor should monitor the power levels of that sensor so to make sure that the sensor can support such a high rate without consuming too much energy and compromise its future operations.

### 1.4.1 Security Research Opportunities

Communication in the IoT will be in large part between interconnected things or devices, from the largest to the smallest. Devices will share information with each other in an autonomous way, without human intervention. Because of this, it is important to have ways to enforce what kind of information should "flow" freely between devices and what information should be kept private. Similarly, communication between devices should be secure so that no one can eavesdrop on the communication between devices and "steal" or modify sensitive information. This is particularly true for multi-hop scenarios where intermediary nodes may forward information not meant for them and where the presence of rogue devices may introduce additional risks. We will look at these and other security-related problems in the following.

**Secure Communication.** How to secure communication between nodes is a very well-known problem, however, many of the approaches used in today's Internet may not be applicable to the IoT. Typical encryption algorithms such as AES-based encryption [17] (i.e., private key cryptography) and Elgamal [18] (i.e., public key cryptography) may be computationally expensive thus draining a significant amount of power. Further research is required in order to better understand how to enable en-

cryption on devices with stringent power and computational constraints.

**Key Exchange.** In any encryption scheme, key material needs to be exchanged between nodes. In the IoT, nodes operate on very low rate links and on low power trying to avoid unnecessary transmissions. How to provision key material to a very large number of such nodes in a secure way remains an open problem. For example, such cryptographic material could be provisioned to the nodes during installation before any network connectivity is available. This, however, would have to be done in a tamper-proof way. A possible solution to this problem could be the use of Trusted Platform Modules (TPMs) [19]. TPMs, however, use extra circuitry and therefore need additional power. As such, it is not obvious that this would be a valid approach for IoT where power is a very scarce resource.

**Data Integrity.** Many of the devices in the IoT will be very sensitive in nature. For example, IoT will have a big impact on healthcare where doctors will be able to monitor and treat patients remotely. In such scenarios, both doctors and patients need to be completely confident that not only data is private but also that no malicious thing or user can compromise data integrity. Recently, an attack against an insulin pump [20] was shown where deadly amounts of insulin could have been injected remotely in a patient. In order for this not to happen, the integrity of data must be guaranteed so that the doctor is positive to be looking at the right data and patients are completely confident that they are getting the treatment they are supposed to get without interference of any kind taking place.

**Data Availability.** In the IoT not only the integrity of data must be guaranteed, but also its availability. For example, a denial of service attack on a thermostat of a critical industrial process may cause severe damage. Similarly, a denial of service attack on a heart monitor could prevent a doctor from reacting in a timely manner to life-threatening conditions. For devices with low computational power and stringent power requirements, denial of service attacks may be trivial to accomplish. Triggering unnecessary actions on a device so to quickly drain its battery, would not requi-

re much effort. Similarly, overloading a device with very low computational power could be trivial to accomplish. How to guarantee data availability and prevent attacks such as denial of service attacks on devices with low computational power and stringent energy constraints is currently an open problem.

**Data Privacy.** Because of the distributed nature of IoT, devices will not be single user and single ownership. Devices will be shared between different administrative domains and between different users. An important aspect is therefore, how to guarantee privacy in such a complex setting. In particular, a way is needed to guarantee different levels of privacy and functionality to different users and administrative domains.

**Access Control.** Unlike with data privacy where we need to decide what data a user should be able to access, with access control we need to decide what devices a user should be able to access. For example, the insulin pump attack mentioned earlier could be triggered either by changing the data readings from the pump (i.e., data integrity) or by allowing a non-authorized person to remotely access and control the insulin pump itself. Clearly, how to enable and enforce access control in the IoT is a very important topic.

**Misconfigurations.** Either because of mischievous intentions or because of the very complex and distributed nature of IoT, misconfigurations can happen. The detection and solution of misconfigurations is an important problem. A simple misconfiguration on a few devices may trigger a denial of service attack or other undesired effects. For example, a misconfiguration on how a user forwards calls to his mobile phone may prevent the user from receiving any calls; similarly, a wrong geo-fencing rule may create a situation where SMS messages get repeatedly sent at a very high rate when crossing a geo-spatial boundary, thus causing the user having to pay a large amount of money. In IoT where interactions can be of many types and quite complex, a way to detect and resolve misconfigurations is required.

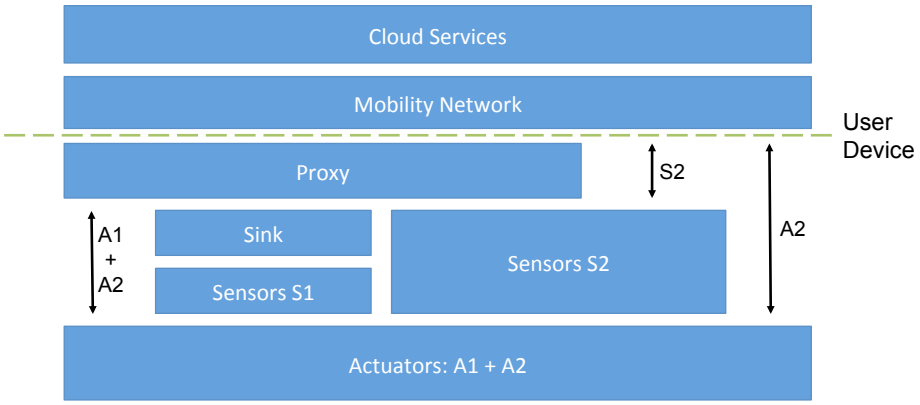



Figure 1.1: *Internet of Things deployment architecture.*

**Mobile IoT.** With Mobile IoT we refer to a situation where a user is away from home but all of his services and personal settings "follow" him. For example, a user traveling abroad, would enter his hotel room and mobile IoT would have already configured all of the devices in the room as if he was at home that is, the room phone would be linked to his personal phone number, the TV would have his recorded shows, the room temperature would be set to his preferred value and the minibar would have his favorite drinks. Finally, his credit card information would be linked to the room. It is easy to see that the information downloaded on all the devices in the room tells a lot about the occupant of that room. How to securely send sensitive data to devices deployed in the public domain is an open problem. Furthermore, it is important to guarantee that whatever personal information was present in the room when the user was there, it will be completely and safely deleted once the user leaves the hotel. How to make sure that personal content is securely deleted from a device in the IoT remains an open problem. Verification of deletion should also be possible.

Specifically to security, we have analyzed and abstracted the classes of attacks that are relevant to the IoT. Fig. 1.1 shows a high-level layered view of a typical IoT deployment architecture. As we can see, we can have two different types of sensors and actuators that is, highly constrained in energy and computational power (i.e., type

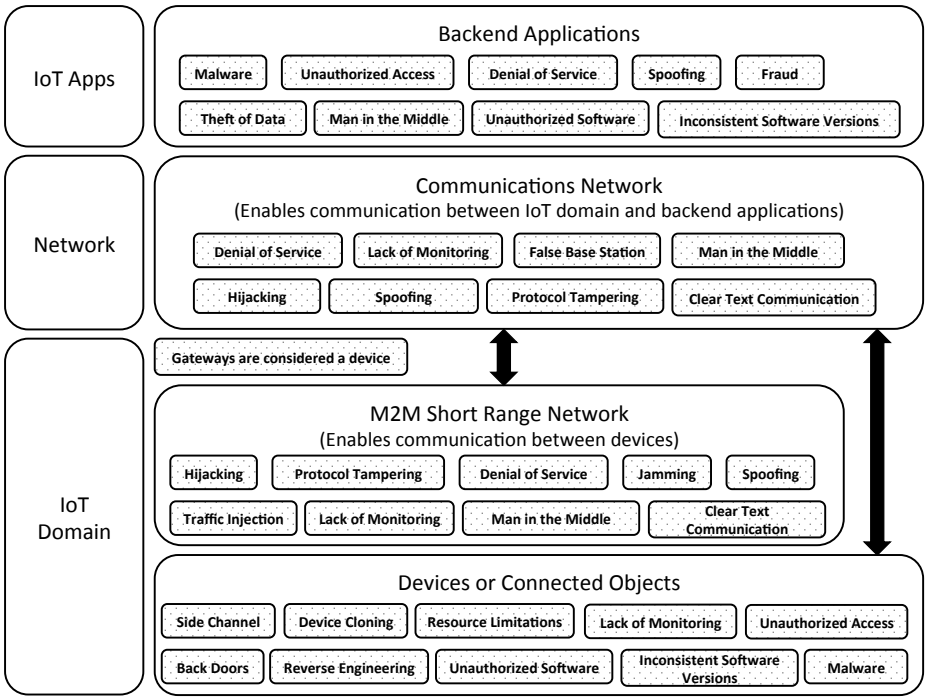



Figure 1.2: *Internet of Things potential threats.*

1) on one end and without any such constrains (i.e., type 2) on the other. Depending on which class of devices we consider, security requirements can be significantly different. For example, type-1 sensors typically communicate in a multihop fashion while type-2 sensors can communicate directly with the cellular network thus bypassing local gateways and nodes. Because of this, type-1 sensors are more exposed to man-in-the-middle attacks while type 2 sensors are less exposed to such attacks given the mutual authentication taking place in 3G cellular network. Fig. 1.2 shows examples of different types of attacks that are possible in the IoT M2M domain. While many solutions are known to many of the problems shown in Fig. 1.2, most of the solutions are not applicable to the IoT because of the very scarce resources that devices have at their disposal. Hence, the need for novel protocols and algorithms.

After surveying multiple technologies and architectures and looking at the state

| Classes of Attacks | Exploited Vulnerabilities | Possible Targets | Risk Severity | Examples | Mitigations |
|---|---|---|---|---|---|
| Device DoS | Limited energy[1] and computational power[2] | Low energy/ computation sensor, actuator, Gateway | HIGH | Exhaust battery of pacemaker by high query rate . Heavy-computation job. | Energy-level and computational-level command validation |
| Data manipulation | No data integrity (e.g., no MAC) | Data-end-point (Sensor, Gateway, Human) | HIGH | MITM modifies data so to trigger fatal defibrillation | Message Authentication Code (MAC) (e.g., Authenticated Encryption) |
| ID spoofing, session hijacking | No or weak authentication | Actuators, Sensors Gateways | HIGH | Pacemaker [incidents] | Strong Authentication (e.g., multi-factor authentication) |
| Eavesdropping, Replay, MITM | No or weak encryption | Actuators, Sensors, Gateways | HIGH | Connected Car [incidents] | Strong Encryption (e.g., AES128) |
| User + network spoofing (e.g., ID, SSID, IP address) | No mutual authentication | Sensors, Actuators, Gateway, eNodeB | HIGH/ MEDIUM | Connected Car [incidents] | Client-Server authentication as well as Server-Client Authentication (e.g., AKA) |
| Device hijacking | Software bugs, backdoor, poor management (e.g., patching) | All devices | HIGH/ MEDIUM | Home monitoring camera [incidents] | Large scale management and patching. |
| Physical damage | Malicious commands or misconfigurations | Actuators, devices controlled by actuator, human | HIGH | Insulin pump [incidents] Set thermostat to heat house to 150F - Furnace explodes | Command sanity check. Configuration verification. |
| Physical tampering | No tamper-proof HW | All devices | HIGH | SIM card stolen from traffic light [incidents] | Tamper-proof HW |
| Data leakage | No data privacy (e.g., weak key management) | Gateway, Sensors | HIGH | Smart meters [incidents] | Strong key management, strong re-keying policy |
| Network DoS, DDoS | Limited network throughput, no network diversity | Gateway, eNodeB | MEDIUM | Volumetric attack (Flooding) | Congestion control, network diversity (e.g., LTE, Wifi, IEEE 802.3) |

Figure 1.3: *Top ten attacks in IoT.*

of the art, we show in Fig. 1.3 what we think are the top ten classes of attacks in the IoT space. In particular, for each class of attack, we show the type of vulnerability exploited, possible targets in the IoT architecture, the severity of such attacks as well as high-level mitigation techniques. Fig. 1.3 also mentions actual incidents as well as examples of attacks that may take place in the future.

## 1.5 A Cellular Network Perspective on IoT

When cellular networks are used as access networks to the Internet by billions of devices, problems arise in terms of load on such networks. While current cellular networks have been able to scale with user population, this may not be the case with the Internet of Things given its extremely large scale. In order to support all the content and signaling generated by such a large number of devices, the research community needs to address several issues.

The most obvious issue is that just the signaling generated by such a large number

of devices may overload the cellular network causing the same effects of a Distributed Denial of Service (DDoS) attack. In an LTE network, where everything runs over IP, this would mean that the whole communication network would be brought down and not even voice calls and SMS messages would go through.

In the cellular network, different operations affect the network in different ways and therefore have a different cost. For example, setting up call forwarding may be considered an expensive operation as it involves interacting with the Home Location Register (HLR). Given that users will be able to apply their own policies on their deployments of IoT, misconfigurations may occur. Malicious intent might be another reason. For example, if an IoT policy involves to automatically set up call-forwarding on the home and office phones of a user and a misconfiguration occurs, each device may enable and disable call-forwarding repeatedly, in a loop. This would not be a serious problem for a single user but it could become very serious if we consider the scale of IoT. Many misconfigurations like the one depicted earlier could trigger the equivalent of a DDoS attack on the HLR [21]. A DDoS attack on the HLR would be enough to prevent the whole network from operating properly. Furthermore, since the HLR is not directly accessible by users, its deployment may have not been dimensioned with such risks in mind.

In reality, the load on the cellular network will not grow linearly with the number of devices in the IoT. A common architecture of the IoT is for multiple devices to talk to a local gateway and then have the local gateway communicate with the outside world, in our case the cellular network. Regardless, the number of devices served by the cellular network may increase at a very fast pace, leaving no time for the network to scale appropriately.

Many of these problems could be addressed by re-thinking the cellular network architecture on one side and by providing more efficient signaling protocols on the other. Re-thinking the cellular network architecture would entitle a drastic re-design of the cellular network aimed at handling such a large number of devices that is, optimized for such a large signaling load. How to provide minimum functionalities such as voice and SMS messages even under high loads should be a priority for LTE cellular networks. On the other hand, simpler signaling protocols that are easier

to process, together with stateless servers, would help reducing the risk of DDoS attacks. A more concise signaling protocol would also allow a more efficient use of radio resources.

The Session Initiation Protocol (SIP) [22] has been chosen as the signaling protocol for the IP Multimedia Subsystem (IMS) [23]. Given the complexity and verbosity of SIP, even if only a small subset of devices in the IoT would implement IMS services, it could be enough to overload the cellular network. For this, a lighter or compressed version of SIP [24] together with the use of stateless SIP Proxies would be highly beneficial for reducing the load on the cellular network.

Finally, given the large number of heterogeneous networks available in homes and urban environments, offloading to non-cellular networks such as WiFi (i.e., vertical handover) might help in situations of high load on the cellular network.

## 1.6 Thesis Research Focus

We research two main aspects of security in the IoT. In particular, we try to answer to the following two questions:

- *"How to provide privacy and confidentiality in energy-constrained devices?"*
- *"How to protect infrastructure supporting the IoT from the IoT itself?"*

The first question applies to IoT devices that are very limited in both energy and computational power such as energy-harvesting devices. For such devices, the standard protocols used in the Internet today are not suitable as they do not take into account the very limited amount of energy and computational power available. Furthermore, for energy-harvesting devices, new protocols should take into account the fact that an energy-depleted device may, after some time, become operational again.

We propose to distribute encryption and decryption operations (i.e., block cipher computations) among a set of trusted nodes and describe a new routing algorithm that enables such cooperation between nodes. This introduces significant energy savings and, consequently, increases the network lifetime.

The second question applies to IoT infrastructure such as the cellular network. For such infrastructure, the IoT represents a highly distributed platform which can be exploited by an attacker to perform distributed attacks (e.g., Distributed Denial of Service (DDoS) attacks) against the supporting infrastructure. In recent years, this type of attacks has increased as the number of inter-connected devices has increased. A recent example of such an attack is a DDoS attack performed in September 2016 against KrebsOnSecurity.com, a website with a proven record of exposing cybercriminals who focus on DDoS attacks. Such an attack was performed using a botnet of more than 1 million internet-connected cameras and other IoT devices generating over 660 Gbps of traffic, one of the largest recorded DDoS attacks in history [25]. It is easy to see how such an attack could be directed against anything and, in particular, against the cellular network thus bringing down cellular communication (i.e., voice and data) for a large geographical area. Generally speaking, in the United States, a cellular network provider has its infrastructure distributed among few national datacenters, typically less than 10. Different datacenters provide service to different geographical areas, while other datacenters are used for redundancy. A DDoS attack on all these datacenters could, potentially, bring down the service all over the country. Given the billions of inter-connected devices we are expected to have by 2020 and the highly-distributed attacks we are starting to see, large-scale attacks trying to bring down critical infrastructure (e.g., cellular networks) does not seem to be a remote possibility any longer.

We propose a new cellular architecture aimed at disabling distributed attacks from happening in the first place and increase the overall security of the cellular network. Furthermore, the new architecture enables cellular network operators to introduce novel security and non-security services to their customers, thus creating new streams of revenue.

# Chapter 2

# Distributing Data Confidentiality

## 2.1 Introduction

The Internet of Things (IoT) will introduce a drastic change in our society as many common things that we use in our every-day life will stop being just objects and will start interacting with each other, with us, and with the environment. As all of these things talk to each other and to us, they will collect and share a very large amount of sensitive information. For example: car engines will collect and share data about the driver's driving habits; people's whereabouts will be shared and used to automatically provide new services; and what we eat and drink could be shared with insurance companies to monitor our eating habits and lifestyle. Similarly, people's vital signs (e.g., heart rate, blood pressure) may be collected and shared in a completely automated way, with privacy implications that are still not very clear. Given the capillarity of the IoT, its pervasiveness in our lives, and given the sensitive nature of the information it will handle, security in the IoT—and, specifically, *data confidentiality*—has become of utmost importance.

The goal of data confidentiality is to prevent disclosure of information to unauthorized parties. When sharing sensitive data, confidentiality is usually enabled by encryption.[1] Over the years, much work has focused on improving both computatio-

[1] Other means include the use of file permissions and access control lists.

nal and energy efficiencies of encryption and decryption algorithms in block ciphers (such as AES). Alternatively, new lightweight block ciphers have been proposed trying to address encryption in energy-constrained networks (e.g., wireless sensor networks). While all of these approaches have their merits, often they are not sufficient to provide the energy savings that would allow them to be implemented in many devices in the IoT and, specifically, in energy-harvesting devices.

We follow a different approach to data confidentiality which motivates the energy measurements which will be presented in the following. In particular, we claim that a large portion of the energy spent to provide data confidentiality can be saved, from the perspective of a single node, by distributing the encryption and decryption operations of a block cipher among a set of *trusted* nodes. This approach takes into account the following two facts.

- In multi-hop networks, such as wireless sensor networks, nodes save energy by transmitting at a much shorter range. Because of this, nodes have to relay each other's packets from the source all the way to the sink. This relaying of packets makes multi-hop networks well-suited for performing distributed computations on packets.

- Block ciphers have an iterative structure, which usually is either a Feistel network or a substitution-permutation network. This iterative structure makes their computation easy to distribute. In particular, AES uses a substitution-permutation network and it works in "rounds", so that a node can decide to perform one round or multiple rounds.

Given the multi-hop nature of wireless sensor networks, where nodes already relay each other's packets, a distributed computation of a block cipher would just add the burden for a node to perform one or more rounds of encryption or decryption on packets that it has to relay. The number of rounds a node could perform would be based on various factors, including its energy level.

Distributing block cipher operations may be useful in all those wireless sensor networks deployed in a hard-to-access environment (e.g., under-sea networks, underground networks) where replacing batteries may be extremely difficult and

expensive—if at all possible. In such networks, having a trusted set of nodes is enforced by the very environment in which the networks operate. In particular, in such environments eavesdropping communication between nodes, as well as hijacking nodes, would be extremely hard.

One possible way to distribute encryption and decryption operations would consist, for example, in including an appropriate metric in a multi-hop routing protocol. In routing packets, such protocol would make sure that by the time a packet reaches the edge of the trusted zone, the packet has been correctly encrypted. Furthermore, it would build paths so that nodes with higher energy would perform, in relative terms, more rounds of encryption. The design of such a routing protocol as well as appropriate key-management strategies are left for future study.

Distributing block cipher encryption and decryption operations between a set of *trusted* energy-constrained devices will be shown to bring significant energy savings to a device. The advantage of such an approach is twofold: on one hand, traditional block ciphers can now be used in energy-constrained devices; on the other hand, the energy consumption of "lightweight" block ciphers can further be reduced, thus increasing a node's battery lifetime.

## 2.2 Current Efforts

Data confidentiality in the Internet is usually enabled by encryption. Traditional encryption algorithms, however, are too slow and energy-demanding to be applied to the IoT. In particular, the amount of energy spent in encryption and decryption is a critical factor as we move towards wearables, energy-harvesting wireless nodes, and the IoT.

Efforts in reducing block ciphers energy consumption can be divided into three main categories: hardware optimizations, software optimizations, and new "lightweight" algorithms. Let us look at each one of these.

Hardware optimizations [26, 27, 28] mainly focus on reducing the number of logic gates necessary to implement a block cipher.[2] A smaller circuit consumes less

[2] Another important metric is the number of cycles required to process one block.

energy than a larger one: therefore, by reducing the circuit size, the block cipher consumes less energy. These approaches usually require the use of very specialized hardware, making their feasibility quite difficult if not for very peculiar applications.

Software optimizations [29, 30, 31] aim at reducing both the memory footprint and the computational power of block ciphers. Using a smaller amount of computer memory and fewer CPU cycles translates to a lower energy consumption. These approaches achieve some energy savings which, unfortunately, are not sufficiently significant to satisfy the strict energy requirements that some IoT nodes will likely have. Because of this, over the years new lightweight block ciphers have been proposed [32, 33, 34, 35, 36, 37] trying to address the energy requirements typical of wireless sensor networks and the IoT. The basic idea behind these block ciphers is to consume little energy while providing lower throughput and a moderate level of security. In particular, the implicit assumptions behind this idea are the facts that in a constrained environment the amount of data to encrypt will be small and an attacker will have limited computing capabilities, so that a moderate level of security is appropriate. The key length for such lightweight block ciphers is not too long, typically between 80 and 128 bits, and they usually operate on a smaller 64-bit block size of input data. Their low energy consumption makes them suitable for many resource-constrained devices at the price, however, of limited protection. On the other hand, the reduced level of protection due to the use of smaller encryption keys can be increased by introducing periodic key-changing mechanisms (whose rate strictly depends on the amount of security targeted) in order to make it more difficult to perform brute force attacks on these ciphers [38].

## 2.3 A Distributed Computation Approach

A large number of devices in the IoT will be energy-constrained and will often have to operate using just a few micro-joules of energy [11]. Because of their limited energy, such devices (i.e., nodes) typically form multi-hop networks. In other words, instead of sending packets directly to a far-away gateway, which would require higher transmission power, they relay each others' packets all the way to the gateway

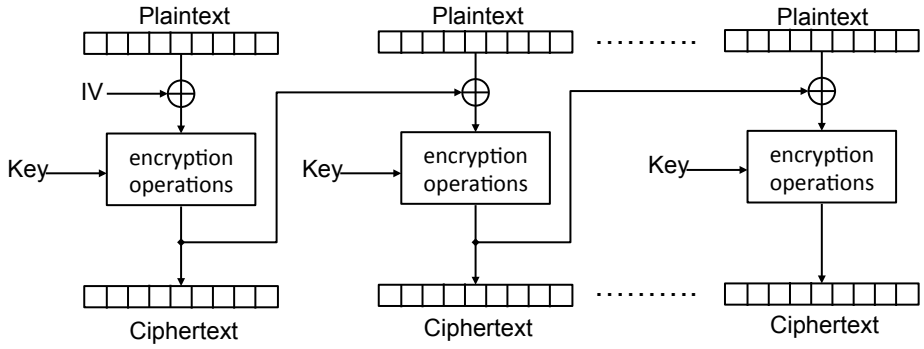



Figure 2.1: Example of block cipher iterative structure in Cipher Block Chaining (CBC) mode.

by using appropriate routing protocols [8]. In doing so, less energy is consumed in transmission as neighboring nodes are typically much closer than the gateway.

We propose to distribute block cipher operations among a set of trusted nodes. In particular, this approach is motivated by two considerations. The first one is the multi-hop nature of communication for energy-constrained nodes: as such nodes relay each other's packets, they can thus easily apply incremental computations on them. The second one is the iterative structure of block ciphers, usually either a Feistel network or a Substitution-Permutation network: an illustrative representation is shown in Fig. 2.1. For both block cipher structures, the same operations are repeated multiple times in different stages or rounds. Because of this iterative computation and because of the multi-hop nature of the communication protocol between nodes, it is natural to think to distribute the rounds of encryption of a given block cipher (e.g., 10 rounds in AES128) among a selected set of nodes in the multi-hop network.

The nodes participating to the distributed computation need to be *trusted*. More precisely, they must be positioned in such a way that: (i) their communication cannot be eavesdropped and (ii) they cannot be hijacked or tampered with. In Fig. 2.2, an illustrative representation of such a scenario is shown. For as long as all the required rounds of encryption are completed within the trusted zone, everything is fine.

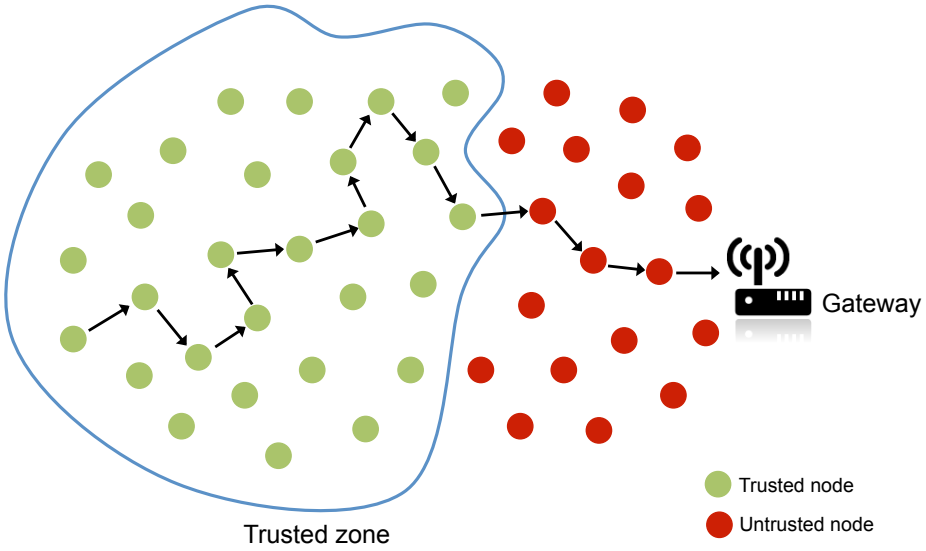



Figure 2.2: Multi-hop network: distributing block-ciphers computations among the trusted nodes.

If, however, a packet reaches the boundary of the trusted zone before all rounds of encryption have been performed, then the content of the packet, as well as the secret key used by the block cipher, may be compromised.

A multi-hop network with a trusted set of nodes, as the one just described, could be any wireless sensor network deployed in hard-to-reach places, such as under-sea or underground networks. Saving energy and prolonging battery lifetime in such networks is very important, given the difficulty in replacing the batteries, if at all possible. For example, a critical infrastructure in the city of New York is the underground network of steam pipes which runs all throughout the city and is meant to keep water pipes temperatures high during winter so to prevent them from freezing and bursting. In an IoT scenario, one can imagine having sensors and actuators attached to these underground steam pipes. The sensors would monitor the temperature level of the water pipes and communicate with nodes above ground. Such “external” nodes would monitor weather conditions and pipes temperatures in order to “tell” the ac-

tuators how much steam to push into the underground system: this would allow efficient management of the pipes and, therefore, economic savings. Clearly, the sensors monitoring the underground water pipes cover a very vast area and collect sensitive information—in fact, misuse of such information may cause severe damage. It is, therefore, important that their communication to the external nodes is kept confidential (i.e., encrypted) while, at the same time, using the least amount of energy. In such a scenario, sensors can distribute block ciphers computations making sure that, by the time packets are sent to sensors above ground, encryption has correctly completed. Sensors operating underground fit our definition of "trusted."

Clearly, the way packets are routed through the multi-hop network is very critical. As part of future work, one attractive research direction is the design of a routing protocol, operating in the trusted zone of the multi-hop network, able to maximize the lifetime of the set of trusted nodes while making sure that packets have been fully encrypted by the time they leave the trusted zone. This opens interesting research questions about possible cooperation strategies among trusted nodes.

## 2.4 Arduino-based Experimental Analysis

AES is a block cipher that operates on 128-bit blocks of plaintext and outputs 128-bit blocks of ciphertext. The key length in AES can be 128, 196, and 256 bits, which correspond to 10, 12, and 14 rounds, respectively. There are several modes of operation for AES. Throughout our Arduino-based experiments we consider the Cipher Block Chaining (CBC) mode, as it is one of the most used.

### 2.4.1 Testbed Setup

In order to measure a node's energy savings when distributing encryption and decryption computations, we use the Arduino testbed shown in Fig. 2.3. This testbed includes two Arduino UNO R3 boards equipped with an Atmel ATMEGA-328p microcotroller and an Adafruit INA219 board. In particular, encryption and decryption are performed by one of the Arduino UNO boards (indicated as, namely, the Arduino

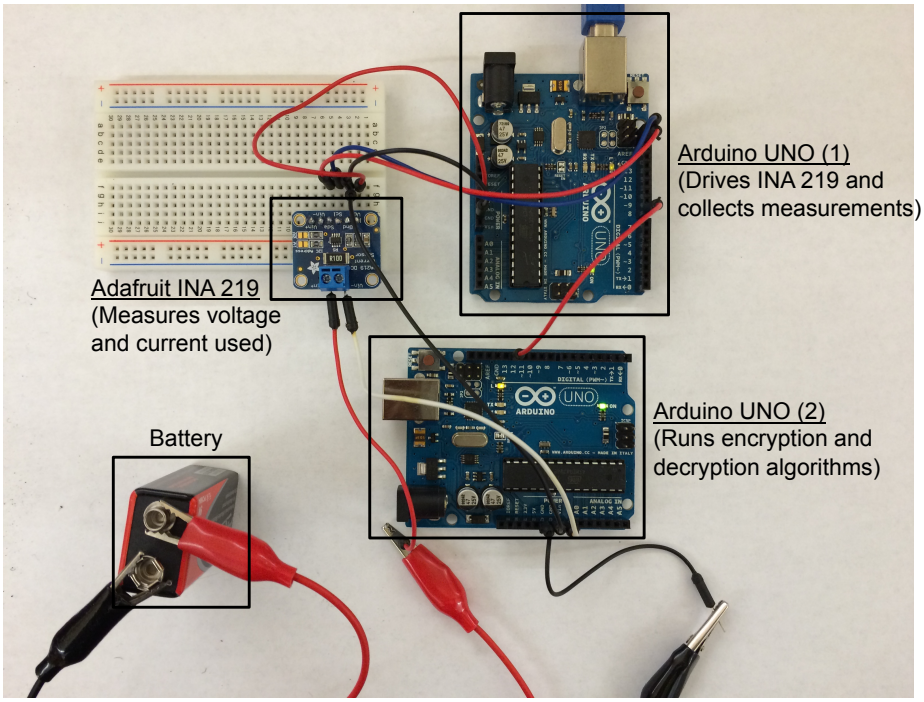



Figure 2.3: Arduino testbed used for the experiments.

UNO (2) in Fig. 2.3). This board was powered by a 9 Volt battery. Voltage and current used by the board were measured by the Adafruit INA219. The Adafruit INA219 was powered and driven by another Arduino board (indicated as, namely, the Arduino UNO (1) in Fig. 2.3) which would also collect all the measurements received from the Adafruit INA219 and would then send them to a laptop via USB. On the laptop, the measurement readings would be cleaned up and further processed using some Python scripts in order to compute the actual energy consumed by the Arduino board.

From a software standpoint, we modified Davy Landman's AESLib library [39] in order to let the Arduino node perform a custom number of rounds of encryption and decryption per each packet sent or received.

### 2.4.2 Experimental Measurements

We measure the energy consumption and savings of the AES block cipher as it is widely used in the Internet (e.g., Transport Layer Security (TLS) protocol [40]). AES is a block cipher that operates on 128-bit blocks of plaintext and outputs 128-bit blocks of ciphertext. The key length in AES can be 128, 196 and 256 bits, which correspond to 10, 12, and 14 rounds, respectively. We focus on AES with a key length of 128 bits and, thus, 10 rounds. Furthermore, there are several modes of operation for AES. Throughout our experiments we consider the Cipher Block Chaining (CBC) mode, as it is one of the most used. In CBC mode, an Initialization Vector (IV) must be used in order to make each message unique. Furthermore, as we can see from Fig. 2.1, at each round a block of plaintext is first XORed with the previous block of ciphertext and then encrypted.

#### Power Measurements

In order to measure the power used by encryption operations in AES128, we wrote an Arduino program that encrypts a packet of 1024 bytes every 10 ms. The program runs for 10 minutes: after this time interval has elapsed, we comment out the line of code responsible for the encryption and reload the program onto the microcontroller of the Arduino board. Except for the one line of code that gets commented out, everything else remains the same, so that only the presence or lack of the encryption process affects the power measurements. This new program with "disabled" encryption runs for another 10 minutes, after which we reload the program with "enabled" encryption and so on and so forth. We repeat this process for 50 times. Fig. 2.4 shows the operational power of the Arduino UNO board during the encryption (crypto) and no-encryption (no crypto) states. As one can see, every time we load the program onto the board microcontroller there is a drop in voltage and current (i.e., power), followed by a small positive spike. These power variations were carefully removed during data analysis.

Current and voltage readings were collected every 100 ms. In particular, we measured the encryption power by measuring the difference in terms of operational po-

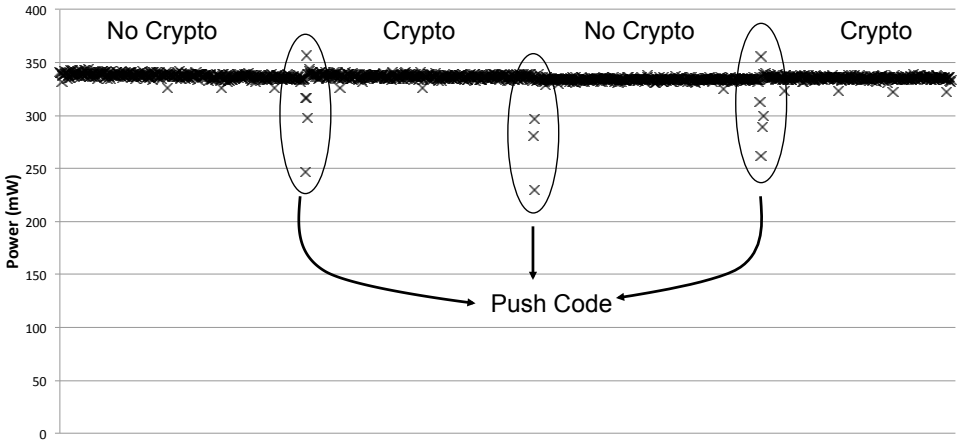



Figure 2.4: Experimental measurements of the encryption and decryption operational power entailed by AES128-CBC.

wer, when going from the no-encryption state to the encryption state and not viceversa. This was done in order to mitigate the effect on the measurements of the natural decay of power due to the battery drain during the experiments. Similarly, the power used by decryption operations was measured.

The obtained results show that encryption and decryption operations use, on average, 1.24 mW and 1.79 mW, respectively. Furthermore, these values do not depend on the number of rounds of the block cipher that the node performs as power is the *rate* at which energy is consumed.

### Energy Measurements

Our first objective is to measure how much energy a node would save when using distributed block cipher computations. In order to do this, we measure the energy consumed by a node running AES128-CBC and compare it with the energy spent by the same node running one single round of AES128-CBC. As previously shown, the encryption operations in our testbed use, on average, 1.24 mW of power, while decryption operations use, on average, 1.79 mW of power. In order to measure the ener-

| | Encryption | | | Decryption | | |
|---|---|---|---|---|---|---|
| **# of Rounds** | **Duration (ms)** | **Energy ($\mu$J)** | **Savings (%)** | **Duration (ms)** | **Energy ($\mu$J)** | **Savings (%)** |
| 1 | 3.25 | 4.05 | 73 | 2.66 | 4.77 | 81 |
| 2 | 4.22 | 5.26 | 65 | 3.92 | 7.02 | 72 |
| 3 | 5.19 | 6.47 | 56 | 5.17 | 9.27 | 63 |
| 4 | 6.16 | 7.68 | 48 | 6.43 | 11.52 | 54 |
| 5 | 7.13 | 8.89 | 40 | 7.68 | 13.77 | 45 |
| 6 | 8.1 | 10.1 | 32 | 8.94 | 16.02 | 36 |
| 7 | 9.07 | 11.31 | 24 | 10.19 | 18.27 | 27 |
| 8 | 10.04 | 12.52 | 16 | 11.43 | 20.48 | 18 |
| 9 | 11.01 | 13.73 | 8 | 12.70 | 22.77 | 9 |
| 10 | 11.98 | 14.93 | 0 | 13.96 | 25.02 | 0 |

Table 2.1: Average time and energy spent for encryption and decryption of 1024 bytes for a different number of rounds of AES128-CBC. The average savings of a node performing a distributed computation of AES128-CBC are also shown.

gy consumed, knowing the used power, one needs to measure how long encryption and decryption operations take.

Table 2.1 shows, among various performance metrics, the duration of encryption and decryption operations for different numbers of rounds for a 1024-byte packet size. One can first observe that while for both one-round and two-round computations encryption takes longer than decryption, the opposite is true for a number of rounds larger than two. In particular, for full AES128 (i.e., 10 rounds), encryption takes, on average, 11.98 ms, while decryption takes, on average, 13.96 ms. On the other hand, for a single round of AES128, encryption takes, on average, 3.25 ms, while decryption takes, on average, 2.66 ms. Interestingly, for both encryption and decryption, the time required to run full AES128 (i.e., 10 rounds) is considerably shorter than ten times the time required to run one round of AES128. Consequently, the overall energy spent by running the non-distributed version of AES128 is much lower than the overall energy spent by running one round of AES128 ten times. As we confirmed earlier, this difference is due to the initial overhead of AES128 caused by initialization operations such as key expansion. In particular, this initial overhead is responsible for

an additional average consumption of 2.84 $\mu$J for encryption and of 2.53 $\mu$J for decryption. In the Arduino AES library used in our testbed, such overhead is present at each round of the distributed version of AES128, whereas it is present only in the first round of the non-distributed version of AES128. This difference in the initial cost is the reason why running one round of AES128 ten times (i.e., distributed AES128) is much more costly, in terms of energy, than running the non-distributed version of AES128. This clearly shows that one of the optimizations required in distributing block cipher computations is to make sure that any initialization cost incurs only once, at the first round, and does not get propagated to subsequent rounds of computation. By taking this into account, the total amount of energy spent by running distributed block cipher computations must match the amount of energy spent when running the non-distributed version of the same block cipher.

As shown in Table 2.1, in our measurements we have observed that the node running the first round of distributed encryption consumes, on average, 4.05 $\mu$J of energy. However, nodes performing subsequent rounds of encryption spend, on average, 1.21 $\mu$J per round of encryption due to the lack of initialization costs. Similarly, for decryption, the node running the first round of decryption would consume, on average, 4.77 $\mu$J, while nodes running subsequent rounds would consume, on average, 2.24 $\mu$J.

Naturally, energy savings reduce as nodes run an increasing number of rounds of block-cipher computations. In Table 2.1, we also show the energy savings of a node when performing a different number of rounds. As one can see, from energy savings of 73% in encryption and of 81% in decryption, when performing 1-round computation, a node reduces its energy savings to 48% in encryption and 54% in decryption when performing 4-round computations of the same block cipher. Such savings drop down to 8% in encryption and 9% in decryption when performing 9-round computations. As expected, the larger the number of computed rounds, the lower the energy savings—obviously reaching zero for 10 rounds (i.e., full AES128).

We now investigate the impact of the packet size on AES128 power consumption. In Fig. 2.5 and Fig. 2.6, the encryption and decryption times are shown, respectively, as functions of the packet size. In both cases, full AES128 (i.e., 10 rounds) and one

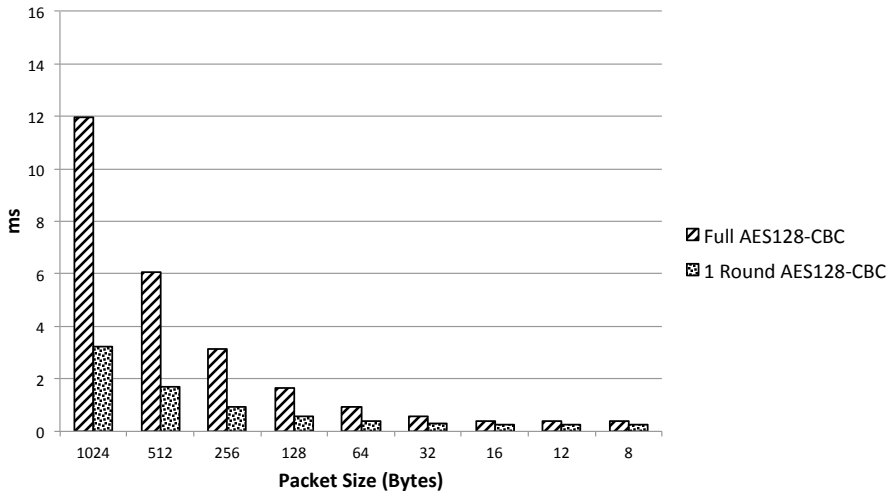



Figure 2.5: Encryption time as a function of the packet size.

single round of AES128 are considered. In particular, we measure the time required by encryption and decryption operations for packet sizes between 8 bytes and 1024 bytes. It can be observed that the duration of encryption and decryption decreases linearly with the decrease of packet size. However, below 16 bytes (i.e., 128 bits), encryption and decryption times tend to remain constant. This behavior is due to the fact that in AES128-CBC packets smaller than 16 bytes are padded to the block size of 16 bytes.

In Fig. 2.7, the energies consumed by (a) encryption and (b) decryption are shown as functions of the packet size, considering full AES128 and one round of AES128. In both encryption and decryption, one can see that by using a distributed block cipher, the highest energy savings are obtained with large packet sizes. As the packet size decreases, so do the energy savings, down to a point where performing full AES128 takes almost the same amount of energy as performing a single round of AES128. This result is very surprising, as one would expect the savings to be roughly the same, in relative terms, for any packet size. By comparing Fig. 2.7a and Fig. 2.7b, it can also be concluded that such savings are greater for decryption than for encryption.

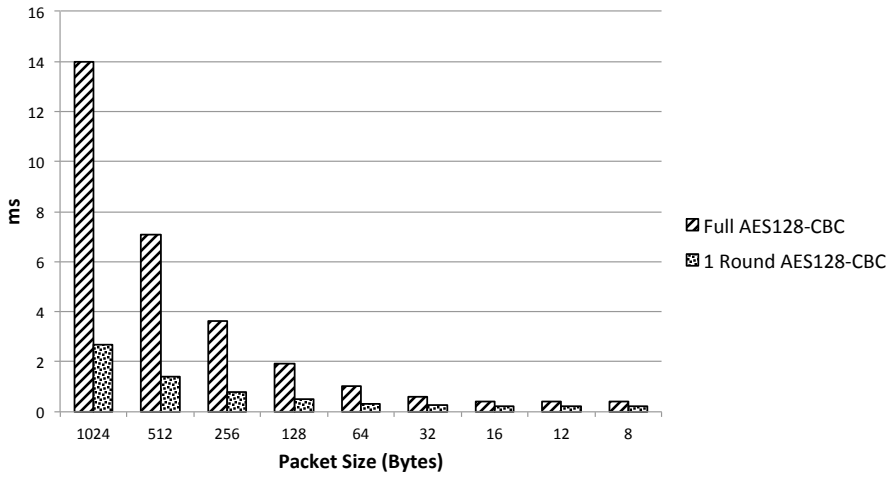
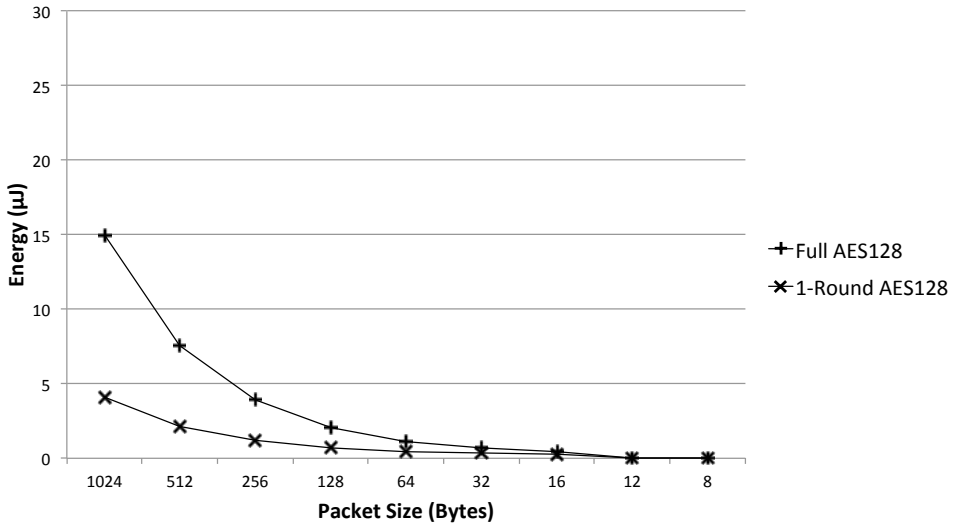


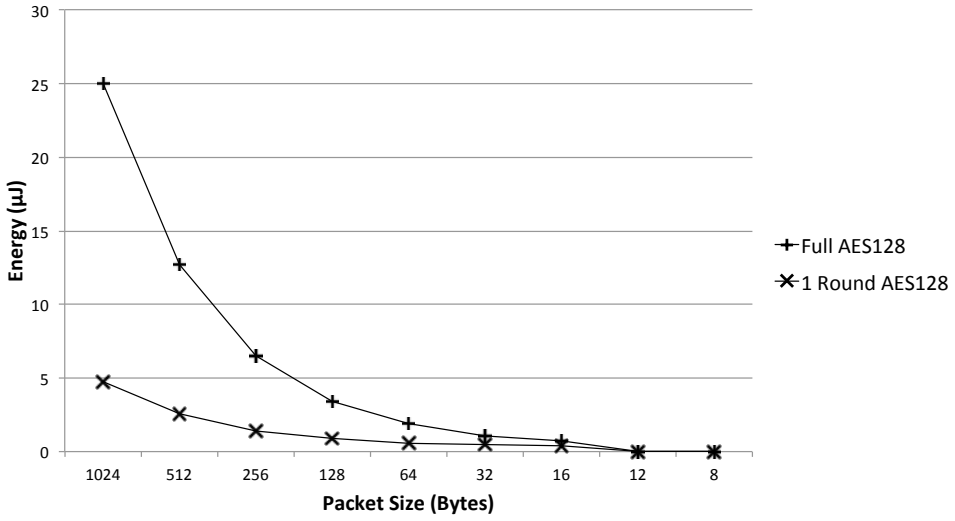

Figure 2.6: Decryption time as a function of the packet size.

Table 2.2 shows the energy savings that a node would have by using a distributed block cipher and computing a single round of AES128 as opposed to running full AES128. As can be seen, for larger packet sizes one can save up to 73% and up to 81% of the energy usually spent during encryption and decryption, respectively. For decreasing packet sizes, the energy savings decrease as well, although they still remain quite significant. For packet sizes smaller than or equal to 16 bytes, the savings remain constant due to padding.

In wireless sensor networks, as well as in many IoT applications, packet sizes tend to be quite small. There is, however, a vast literature on the use of data fusion, data aggregation and other techniques to combine multiple packets together, before sending them to the sink, in order to save transmission energy. Some proposals use clusterheads to combine packets together before sending them to the sink [41] or to other clusterheads [42], while others combine packets together as they travel through the multi-hop network on their way to the sink [43]. The reason for combining packets together is that while sending a short packet consumes less energy than sending a large one, sending many small packets is less efficient due to the packet headers that,




(a) Encryption




(b) Decryption

Figure 2.7: Comparison, in terms of consumed energy as a function of packet size, considering full and 1-round AES128-CBC.

| Size (Bytes) | Encryption (%) | Decryption (%) |
|---|---|---|
| 1024 | 73 | 81 |
| 512 | 72 | 80 |
| 256 | 69 | 78 |
| 128 | 65 | 74 |
| 64 | 59 | 68 |
| 32 | 49 | 58 |
| 16 | 36 | 45 |
| 12 | 36 | 45 |
| 8 | 36 | 45 |

Table 2.2: Average energy savings (in both encryption and decryption), for different packet sizes, when using AES128-CBC as a distributed block cipher and performing 1 round.

overall, reduce the goodput. Therefore, the following observations hold:

- just by looking at the packet size, more energy is spent to transmit large packets than smaller ones;
- considering the transmission energy spent per bit of payload (i.e., excluding headers content), the opposite is true and more energy can be saved by sending a few larger packets.

A data fusion approach thus goes well together with our findings of higher energy savings, in terms of encryption and decryption energy costs, for larger packet sizes. By using data fusion together with a distributed block cipher computation, transmission energy costs, as well as encryption and decryption energy costs, can be simultaneously reduced.

### Lifetime Increase: a Single Node's Perspective

As we discussed earlier and shown in Table 2.1, a node performing one round of encryption can save 73% of the energy normally spent in encryption. For an Arduino UNO board powered by a 9 Volt alkaline battery, this means about 9 minutes of

extra battery lifetime, which roughly corresponds to 3% of the total battery lifetime. While this may seem like a modest energy saving, one needs to keep in mind that we are reducing the energy consumption of encryption and decryption operations which use between one and two milliwatts of power, while the Arduino UNO board has an *operational power* of hundreds of milliwatts, that is, two orders of magnitude higher than both the encryption and decryption powers. Given this huge difference in power consumption, a 3% battery lifetime increase achieved by reducing only the energy consumption of encryption operations is quite impressive. For energy-harvesting devices, where the power used in cryptographic operations has the same order of magnitude of the operational power, significantly greater savings are expected.

Another cause of significant energy consumption is *transmission power*. In our measurements, a Digi XBee S1 802.15.4 RF module, connected to an Arduino UNO board, used about 71 mW of power to send a 1024 byte packet at 9.6 kbps. This power is roughly one order of magnitude higher than what is required by encryption and decryption operations. In the IoT, however, a large class of devices will be energy harvesting devices. For such devices, operation and communication costs, in terms of energy, become comparable to computational cost and, as such, to the cost of cryptographic operations. For example, Energy Harvesting Active Networked Tags (EnHANTs) [44] use a few microjoules of energy to transmit a packet. For such devices, great savings in encryption and decryption translate to great savings in overall node energy consumption, as the energy spent in encryption and decryption represents a large part of the node's energy budget.

#### Lifetime Increase: a Multi-hop Network Perspective

Up to this point we have discussed energy savings from a single node's point of view, without taking into account network dynamics. When we consider not just one node but the whole multi-hop network of trusted nodes (i.e., trusted zone in Fig. 2.2), some important observations can be carried out.

We have seen that a single node performing one round of encryption can save 73% of the energy normally spent in encryption operations. If, however, this node performs multiple rounds of encryption, then its energy savings decrease as per Ta-

ble 2.1. In a multi-hop network, a node can generate its own packets but can also relay other nodes' packets. For example, if a node generates one packet and receives nine packets from other trusted nodes, since on each of these ten packets one round of AES128 encryption[3] is performed, it will then consume an amount of energy equal to ten rounds of encryption. This is the same amount of energy the node would have consumed by just encrypting its own packet with "traditional" AES128 (i.e., in a non-distributed manner). Therefore, apparently, there would be no savings in using a distributed computation approach in such a scenario. To make things worse, if the node receives packets from twenty trusted nodes and performs one round of encryption on each one of those, it would spend twice the amount of energy that it would have spent by encrypting only its own packets with "traditional" AES128. More generally, if the trusted zone of the multi-hop network generates and processes a given amount of packets, it will consume, overall, the same amount of energy with or without distributing encryption computations. So, why using a distributed approach for encryption and decryption?

The energy savings obtained by distributing encryption and decryption computations, at a network level, depend on two important factors: battery residual energy distribution among the nodes in the network and network topology. Taking into account those factors, various challenges emerge.

In a wireless sensor network, for example, nodes will detect events, thus generating traffic, with distributions that are not spatially uniform. Because of this, some nodes will generate a large amount of traffic, while others may not generate any traffic at all, but act only as relays. This asymmetry in traffic generation contributes to creating a non-uniform battery residual energy distribution among the nodes in the network—under the implicit assumption that all nodes have the same initial battery energy. Furthermore, if the nodes are energy harvesting nodes, the different rate at which each node can harvest energy will further accentuate the difference in the available energy among nodes. In such a scenario, a routing protocol, as the one described in Chapter 3, would make it possible to route packets so that nodes with a

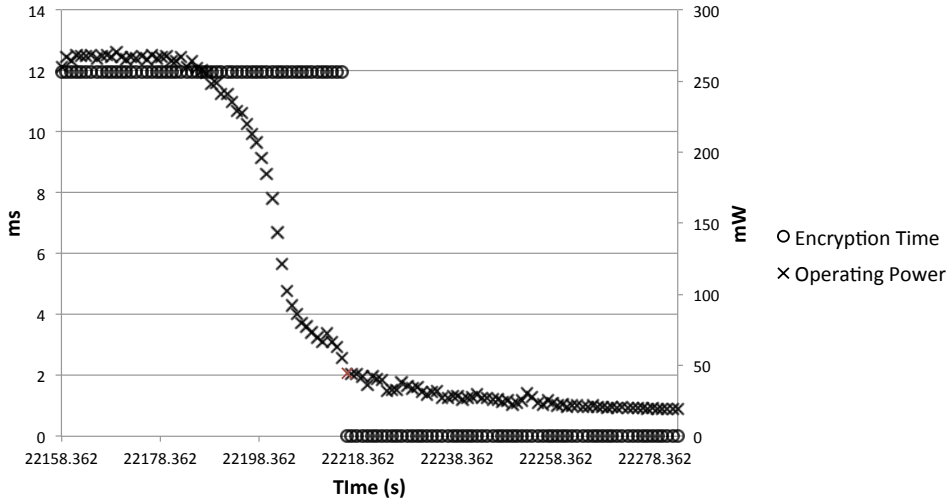
[3] The first, more energy demanding, round for its own packet and a subsequent round for other nodes' packets.

larger amount of available energy would perform more rounds of encryption while nodes with less energy could save it for routing purposes. In doing so, the overall lifetime of the network would increase as nodes with a small amount of energy would prolong their lifetime by delegating encryption to nodes with a higher energy budget. In other words, a routing protocol using distributed encryption computations would increase the network lifetime by distributing encryption operations in a non-uniform way among nodes, following the non-uniform energy distribution. In multi-hop networks, network topology is very important. In particular, because of various reasons (e.g., physical obstructions, nodes layout), there could be a few nodes that have to relay a large amount of traffic between different parts of the network or between the network and the gateway. This is especially true in network topologies considered in IoT scenarios, such as those created by the use of the IPv6 Routing Protocol for Low-Power and Lossy Networks (RPL) [8]. Such nodes are extremely important since, if they die, large parts of the network, if not the whole network, are likely to remain isolated and packets will not reach the gateway any longer. Therefore, it is important, for such nodes, to live as long as possible regardless of their initial energy budget. In such a scenario, by distributing encryption computations, the routing protocol could take this into account and would not assign any encryption computations to such nodes in order to maximize their lifetime and, thus, the lifetime of the network of trusted nodes.

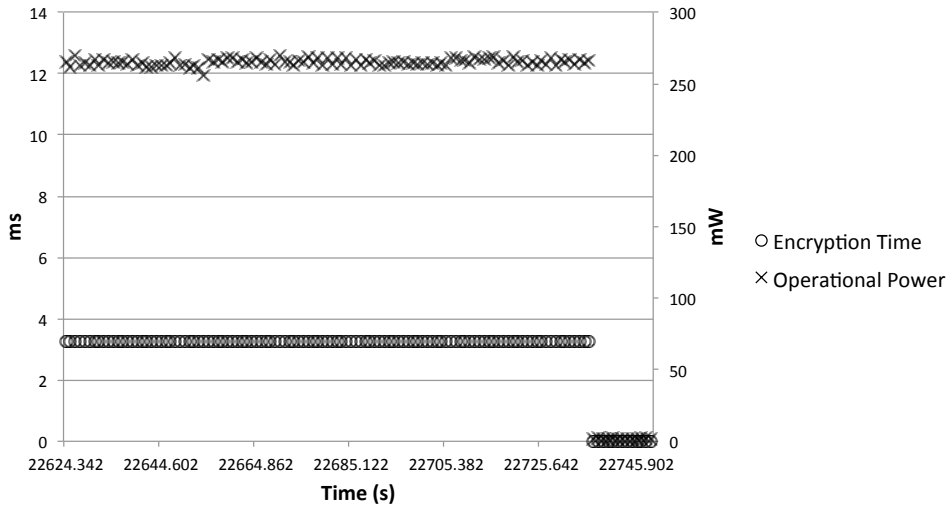
### Battery Discharging Profile

As shown above, nodes participating in a distributed block cipher will be able to encrypt and decrypt packets at a much lower energy level. Because of this, a battery used by such nodes will reach a discharged state with a residual energy lower than that of a node operating the full block cipher.

Fig. 2.8 shows the tails of the discharging profiles of (a) a battery powering a node performing full AES128 encryption and (b) a battery powering a node performing one round of distributed AES128 encryption. In both cases, the same battery type was used (i.e., 9 Volt alkaline battery) and nodes were encrypting a 1024 byte packet every 10 ms. When the encryption time goes to zero, the node has no longer enough




(a) Full AES128-CBC encryption




(b) One round AES128-CBC encryption

Figure 2.8: Tail of the battery discharging profile.

energy to work properly and cannot encrypt or decrypt packets correctly. As can be seen, while discharging for full AES128 is slow and gradual, in the case of 1-round AES128 more energy can be "squeezed out" of the battery, which lasts longer but dies abruptly.

## 2.5 Zolertia-based Simulation Performance Analysis

While measurements on the Arduino testbed have already proven the very high energy savings our approach can provide, we want to explore what kind of energy savings can be expected when distributing computations of lightweight block ciphers that have been engineered specifically for low-power devices. In particular, in the following simulations we focus on the energy spent during encryption by the Scalable Encryption Algorithm (SEA) [32] and the Tiny Encryption Algorithm (TEA) [35]. Furthermore, we compare such savings with the ones achieved by AES128-CBC.

### 2.5.1 Simulator Setup

Simulations have been conducted on Contiki OS-based constrained devices [45], using the Cooja simulator [46]. Contiki OS is a specialized low-power operating system for constrained devices that provides implementations of the IPv6 stack and RPL routing, as well as several MAC protocols. The version of Contiki OS used is 2.7. The experimental platform is based on Zolertia Z1 nodes, with nominal 92 kB ROM (when compiling with 20-bit architecture support) and an 8 kB RAM. In practice, the compilation with the Z1 nodes has been performed with a 16-bit target architecture, which lowers the amount of available ROM to roughly 52 kB. The energy consumption of the CoAP server has been evaluated using Powertrace, a tool for network-level power profiling of low-power wireless networks. Powertrace determines the energy consumption by estimating the amount of time each operation mode of the device is being used. We refer to the current consumption of each component indicated in the Z1 datasheet. In particular, the MSP430f2617 microcontroller consumes 0.515 mA in active mode (CPU) and 0.5 $\mu$A in low-power mode (lpm),

| Block Cipher | Full Encryption | | 1-Round Encryption | | Energy Savings (%) |
|---|---|---|---|---|---|
| | Duration (ms) | Energy ($\mu$J) | Duration (ms) | Energy ($\mu$J) | |
| AES | 18.68 | 28.87 | 4.9 | 7.58 | 73.74 |
| SEA | 20.58 | 31.79 | 3.1 | 4.79 | 84.93 |
| TEA | 20.04 | 30.96 | 1.9 | 2.88 | 90.69 |

Table 2.3: Average time and energy spent by AES, SEA and TEA for full encryption and 1-round encryption of 16-byte, 12-byte and 8-byte payload packets, respectively. The average energy savings of a node performing a distributed computation are also shown.

respectively. Similarly, the CC2420 radio transceiver consumes 17.4 mA in transmitting (TX) mode and 18.8 mA in receiving (RX) mode. In order to obtain the total consumed energy for the node, the following conversion formula has been used:

$$E = \sum_{j \in \mathscr{M}} i_j \cdot v \cdot \Delta t_j \tag{2.1}$$

where: $\mathscr{M}$ is the set of all operation modes of the node (active, lpm, TX, and RX); $v$ is the nominal voltage of the node; and $\Delta t_j$ is the time the node was in the $j$-th operation mode $j$. A linear topology has been used to deploy simulated nodes.

### 2.5.2 Simulation Results

Table 2.3 shows the results of our simulations. As we can see, when doing a full encryption, AES performs much better than SEA and TEA. This is due to the fact that SEA and TEA have been designed to be extremely efficient when implemented using dedicated hardware. In software, however, they do not perform very well and AES represents a better option. On the other hand, by distributing block cipher computations, a single round of SEA and TEA performs better, energy-wise, than a single round of AES. This apparent contradiction is due to the fact that while each round of either SEA or TEA consumes less energy, the total numbers of rounds for SEA and TEA are 51 and 32, respectively: these values are much larger than the 10 rounds of AES. Moreover, in this case we can see that the energy spent by a block cipher when

| Block Cipher | RX Energy (mJ) | TX Energy (mJ) |
|---|---|---|
| AES | 1.12 | 0.37 |
| SEA | 0.94 | 0.5 |
| TEA | 3.24 | 0.88 |

Table 2.4: Average energy spent in RX and TX

performing the full encryption is much lower than the product of (i) the energy spent by a single round of encryption and (ii) the number of rounds. This, as discussed for the Arduino measurements, is due to a higher energy cost of the first round of encryption, which takes into account an initialization phase not present in subsequent rounds. As mentioned earlier, when distributing block cipher computations, we have to make sure to incur in this initialization cost only once so that the total amount of energy spent when distributing the computations equals the amount of energy spent in the non-distributed case.

For all three block ciphers, we have very large energy savings. In particular, from a single node's perspective, with TEA the energy saving for a single node reaches 90.69%, followed by SEA with saving equal to 84.93%, and, lastly, by AES with 73.74%. We remark that, for AES, our simulation results perfectly match, in terms of energy savings, our experimental results. At the network level, however, things are different since the larger number of rounds of SEA and TEA translates into a larger amount of energy spent by the network as a whole. Moreover, routing becomes more challenging given that a larger number of rounds means that a packet may need to be routed through a larger number of nodes (i.e., longer paths). In such cases, there are tradeoffs between route complexity, nodes energy savings and overall network energy level.

Table 2.4 shows the average energy consumption of AES, SEA, and TEA for packet TX and RX. Given that the packets sent and received by the three block ciphers have different sizes, we cannot compare their TX and RX energy consumptions. However, we can see how, for all three block ciphers, RX energy is higher than TX energy. This is consistent with the observations in [11]. In particular, this is due to the

fact that in low-power applications, such as those in the IoT, for packet transmission the radio needs to stay on only for the duration of the transmission; at the opposite, for packet reception the radio needs to stay on for a longer amount of time. Since perfect node synchronization consumes much energy and is, therefore, not implemented, the exact arrival time of a packet is, in fact, unknown

# Chapter 3

# An Encryption-aware Routing Protocol

## 3.1 Introduction

With the approaching of the Internet of Things new scenarios emerge where very low power sensor nodes (e.g., energy-harvesting nodes) need to collect information while operating in "unusual" environments such as under the oceans or underground. Sensor nodes collect data, encrypt it and send it to a sink node for further processing. Usually, communication is multi-hop given the low power requirements. Given the distributed nature of the encryption computations, packets need to be routed accordingly in the multi-hop network. In particular, a new routing protocol using distributed encryption computations would increase the network lifetime by distributing encryption operations in a non-uniform way among nodes, following their non-uniform energy distribution.

## 3.2 Overview

Data confidentiality prevents disclosure of information to unauthorized parties. This is usually achieved through encryption by means of a block cipher (e.g., AES 128).

Block ciphers have an iterative structure (i.e., Feistel network, Substitution-Permutation network) that allows for an easy distribution of their computation. In general, however, data confidentiality is reached only at the very end of the block cipher computations that is, at the very last computation. If data had to be intercepted before the final computation takes place, the original plaintext could be easily recovered and encryption compromised. Because of this, we consider the distribution of block cipher computations in those scenarios where sensors security is provided by the physical environment they are in. For example, intercepting data shared between sensor nodes deployed at the bottom of the ocean would be very hard, if at all possible. In such a scenario, distributing block cipher computations would be considered safe and would assure information to be fully encrypted by the time it reaches the surface. Distributing block cipher computations increases sensor nodes' lifetime therefore reducing the frequency of battery replacements, operation that can be very expensive. In the previous example, we refer to the part of the sensors at the bottom of the ocean as the "trusted zone". Generally speaking, the trusted zone is the part of the sensor network that is physically unreachable to unauthorized users. Other examples where distributing block-cipher computations would be both safe and beneficial include: underground sensors, space sensors, sensors deployed in secure facilities (e.g., the Pentagon).

Given the distributed nature of the encryption computations, packets need to be routed accordingly in the multi-hop network formed by the sensors. In particular, a new routing protocol using distributed encryption computations would increase the network lifetime by distributing encryption operations in a non-uniform way among nodes, following their non-uniform energy distribution.

The basic idea behind the new routing protocol is to attempt to maximize network lifetime by forwarding data along an appropriate path, while accounting for the constraint of every packet needing to undergo M rounds of encryption (e.g., M = 10 for AES128) before reaching either the sink or a node in the untrusted zone. If the network energy levels are high and nodes can do more than M rounds of encryption each, then the routing protocol routes packets according to the Best Minimum-Energy Path (BMEP) irrespective of the cumulative transmission energy consumed along such

path (i.e., number of hops). By doing so, the routing protocol tries to "protect" the nodes with lower energy at the expense of using potentially longer paths (i.e., spending more energy overall). If the network energy levels are low, however, the path to follow is chosen according to the number of rounds of encryption that nodes on the path can perform (i.e., depending on their energy levels). The path satisfying the constraint on the rounds of encryption is picked. If multiple paths satisfy such constraint, the one with the best tradeoff between Minimum Energy (ME) and number of hops to the sink is picked.[1] Unlike traditional routing protocols, when the nodes energy levels are low and the rounds of encryption are a routing constraint, a node may decide to send a packet back to one of its preceding neighbors (i.e., node with high energy but further away from the sink) for performing additional rounds of encryption. This loop-back behavior can repeat itself multiple times, until all the required rounds of encryption have been performed or a TTL has expired. Once encryption completes, the cycle is broken and the loop-free path with best minimum energy is selected for forwarding the encrypted packet to the sink. If the TTL has expired, the packet is dropped.

In our protocol, the control plane carries signaling traffic while the data plane carries packets with *any amount* of application-layer data. In particular, packets with application-layer data that include piggybacked signaling data belong to the data plane.

In designing our routing protocol, we opted for a distance-vector protocol, rather than a link-state one, in order to reduce the size of control packets and minimize the amount of topology information stored at each node. Given that forwarding loops in the data plane waste transmission energy, we seek to avoid them on any given path. However, a distance vector protocol with only *split horizon*[2] or *split horizon with poison reverse*[3] does not resolve all scenarios where a control plane count to infinity

[1]The BMEP may satisfy the constraint on the rounds of encryption (R) but have a number of hops larger than that of another path that still satisfies R but consumes less energy overall as it is closer to the sink. The downside being that this second path may have a worst minimum energy link hence the tradeoff.

[2]Not allowing a router from advertising a route on the same network interface it learned it from.

[3]A variant of split-horizon where the router explicitly advertises a route as unreachable over the

| Parameter | Description |
|---|---|
| ME | Minimum Energy |
| $R_i$ | Num. of rounds of encryption Node $i$ can perform |
| $R_{BMEP}$ | Num. of rounds of encryption that can be performed by all the nodes on the BMEP |
| M | Total number of rounds of encryption required to fully encrypt a packet (i.e., M=10 for AES128) |
| $K_1, K_2, K_3, K_4, \varepsilon_1, \varepsilon_2, \varepsilon_3, \varepsilon_4$ | Protocol parameters |

Table 3.1: Notation used in the routing protocol description.

can occur (count to infinity can result in several control messages going back and forth which is bad for a low-energy system). Hence, we use a path-vector protocol, wherein a node advertises entire routes, and drops advertisements of those routes that include the node itself (i.e., routes with loops).

In the rest of the protocol description we use the notation described in Table 3.1. Each nodes can be in one of the following three phases:

- *Phase 1* (high-energy). The Best ME Path (BMEP) known by the i-th node satisfies: $R_i + R_{BMEP} > K_2 * M$;
- *Phase 2* (average-energy). The Best ME Path (BMEP) known by the i-th node satisfies: $M < R_i + R_{BMEP} < K_2 * M$;
- *Phase 3* (low-energy). The Best ME Path (BMEP) known by the i-th node satisfies: $R_i + R_{BMEP} < M$.

Furthermore, nodes can also be in one of the following four states:

- INITIAL: at time $t \to 0$ when nodes do not know of any paths to the sink;
- AWAKENING: at time $t > 0$ when nodes do not know of any path to the sink;
- AWAKE: at time $t > 0$ when nodes do have at least 1 path to the sink;
- SLEEP: at time $t > 0$ when a node's energy goes below the sleep threshold or the node does not receive any packet for a certain amount of time.[4]

network interfaces it learned it from. This is done by setting the metric for that route to infinity.

[4]From SLEEP state a node can only go to AWAKENING state.

A node in the INITIAL state is becoming operational for the first time ever, therefore, it is assumed to have enough energy to do all M rounds of encryption that is, the node is assumed to be in Phase 1. A node in either AWAKENING or AWAKE state may be in any of the 3 phases, leading to a total of 7 possible state-phase combinations.

We define 3 kinds of paths in the routing protocol: (i) the Best ME Path (BMEP), (ii) the 2nd Best ME Path (2BMEP), (iii) the Best Path as per number of rounds of encryption R (BPR). BMEP is always relevant in all phases; while the other two paths may or may not be relevant depending on the phase the node is in. A path is a list of nodes (or route) with one or more metrics associated with it. In particular, for BMEP and 2BMEP we consider the minimum energy node of the route that is, the energy value of the node with lowest energy level among all the nodes in that route. For BPR, we consider the sum of all the rounds of encryption that each node in a route can do at any given time, given its energy level. In particular, some nodes may do multiple rounds of encryption, others may do zero.

Also, each node builds a cache and a neighbors table. In the cache a node stores its routes to the sink and for each route the associated ME value and R information. In the neighbors table, a node stores information about its 1-hop neighbors such as their identity, the phase they are in, their Highest 2BMEP (H2BMEP) and their $\alpha_i^R$ value.

## 3.3 Initial Neighbor-and-path Discovery

When nodes get deployed they need to discover paths to one or more sinks. Sinks broadcast their presence differently, depending on their energy levels. If they are not energy-constrained, sinks broadcast their presence periodically. If, on the other hand, they are energy-constrained, they broadcast their presence only after a certain time has passed (i.e., discovery timeout) since they started hearing path solicitations by the nodes.

Nodes boot up in the INITIAL state, so that their phase is set to 1. In such state

they broadcast path solicitations if they have a loop-free[5] path to the sink and listen for announcements by the sink as well as by other nodes.

Once nodes have heard at least one loop-free path to the sink, they transition to the AWAKE state. In such state they broadcast their BMEP whenever it changes, and send a BMEP unicast update when receiving a path solicitation message by another node. In order to avoid updating BMEP too often, for small improvements in ME values, a node will update its BMEP to a New Path (NP) only if NP has an ME value significantly better than the one of the current BMEP. In particular, the definition of BMEP is changed to be the best path as per ME subject to a fudge factor of $K_1$. That is, a newly heard path NP will replace the existing BMEP only if $ME_{NP} > K_1 * ME_{BMEP}$ thus reducing signaling overhead. For clarity, when the $K_1$ fudge factor is used, the "best" path as per ME is denoted by $BMEP^{K_1}$.

Eventually, all nodes in the network should transition to the AWAKE state. In a scenario where 50 nodes are added to an existing network on day 2, it is assumed that these 50 nodes can be deployed to wake up in INITIAL state.

## 3.4 Phase 1: High energy

A node in Phase 1 has only to forward data packets along the $BMEP^{K_1}$. This is because, given the high energy level of each node, the rounds of encryption are far from being a constraint in making forwarding decisions. Each node has enough energy to perform M or more rounds of encryption by itself that is, each node can fully encrypt its own packets. In such a phase, control plane operations are as follows.

Each node tracks its own $BMEP^{K_1}$. Any changes in the $BMEP^{K_1}$ are advertised to neighboring nodes in order to avoid loops and so to ensure the use of an optimal[6] path at all times.

[5] Entire routes are advertised. Loops are avoided on all forwarding paths by not considering any path that involves the node itself as a valid path for that node. This is true in all phases of operation at every node, irrespective of the network wide energy.

[6] Except for the fudge factor $K_1$ introduced to reduce signaling overhead.

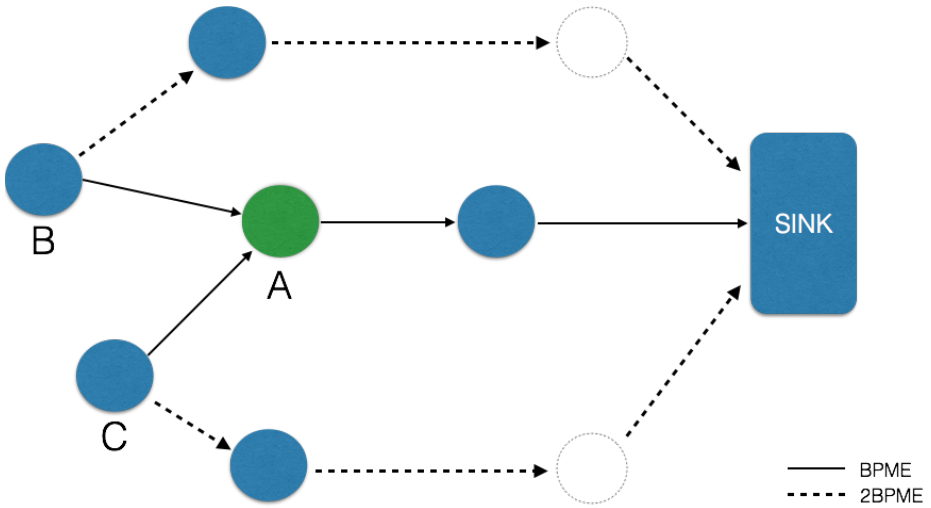



Figure 3.1: Example of BMEP routing.

Each node also tracks its 2nd best ME path ($2BMEP^{K_1}$) and monitors the Highest $2BMEP^{K_1}$'s ME value (H2BME) among all of its downstream neighbours[7] (i.e. neighbours it receives data packets from). This last task is accomplished by having each node piggyback their $2BMEP^{K_1}$'s ME value on data packets sent to their upstream neighbour (i.e. the next hop on their $BMEP^{K_1}$). The purpose of this is to avoid the overhead of having periodic updates of $BMEP^{K_1}$'s ME values and instead use H2BME to trigger control plane updates. Specifically, when a node sees that a downstream neighbor's $H2BME > K_1 * ME_{BMEP^{K_1}}$, it sends a unicast advertisement with its own current $ME_{BMEP^{K_1}}$ to that neighbor so that the neighbor can either switch immediately to its $2BMEP^{K_1}$ with higher ME or forward this advertisement further downstream until the correct node is reached. Once the correct node is reached, it can decide to switch to its $2BMEP^{K_1}$ or not.

Let's look at two examples.

In Fig. 3.1, node A routes packets sent by nodes B and C i.e., the path from node A to the sink is the BMEP for both nodes B and C. Nodes B and C also piggyback,

[7]The sink is the most upstream node.

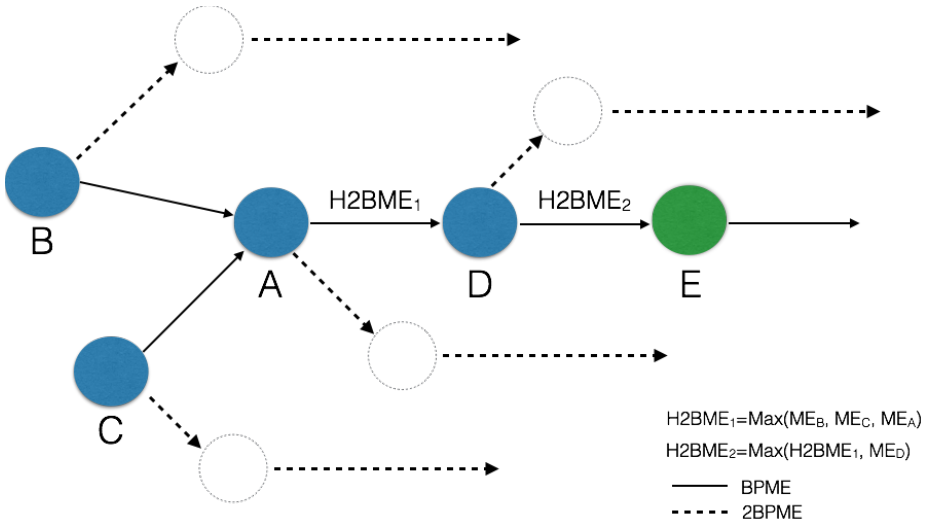



Figure 3.2: A second example of BMEP routing.

on their data packets to node A, information about their $2BMEP^{K_1}$'s ME. If node A sees that node C, for example, has a $2BMEP^{K_1}$'s ME value better than that of its own BMEP, then it sends a unicast update to node C with its $BMEP^{K_1}$'s ME value. In this way, node C knows that its $2BMEP^{K_1}$ is now better than the BMEP via A and can switch to its $2BMEP^{K_1}$, which is now promoted to BMEP for node C. In doing so, node A does not have to send periodic updates to nodes B and C about its $BMEP^{K_1}$'s ME value as this continuously changes due to energy consumption.

In Fig.3.2, nodes B and C piggyback, on their data packets to node A, information about their $2BMEP^{K_1}$'s ME values. Node A computes the maximum between the ME values it just received by the two nodes and its own $2BMEP^{K_1}$'s ME value (i.e., it computes $H2BME_1$) and it piggybacks it on its data packets to node D. Node D will do the same (i.e., computes $H2BME_2$) and so will do subsequent nodes in the path. At some point in time, node E realizes that its $BMEP^{K_1}$'s ME value is now lower than H2BME2 and so will inform node D of this. If H2BME2 is equal to node D's $2BMEP^{K_1}$'s ME value, node D will switch to its $2BMEP^{K_1}$ and will advertise this

change in BMEP as mentioned earlier. If, on the other hand, $H2BME_2$ is equal to the $2BMEP^{K_1}$'s ME value of some other downstream node (i.e., $H2BME_1$), node D will inform node A of the change in ME value on the BMEP. If $H2BME_1$ is equal to node A's $2BMEP^{K_1}$'s ME value, node A will switch to its $2BMEP^{K_1}$ and will advertise this change in BMEP. If not, it behaves as in the previous example. In doing so, we avoid periodic updates in BMEP's ME values and send only triggered updates when needed.

To ensure loop free operations: a) the $2BMEP^{K_1}$ must be loop free, and b) when a node starts using its $2BMEP^{K_1}$ (i.e., now becoming the node $BMEP^{K_1}$), it advertises such change in BMEP to its neighbors via a broadcast packet. This advertisement allows other nodes to make sure that such a change in best path does not create a loop for them. If it does, nodes discard their BMEP with this node, as it now contains a loop, and search for a different (loop-free) BMEP. Furthermore, by the time a node switches to its 2BMEP, information regarding this path may be stale and there may be other paths that are now better. This is a direct consequence of having triggered updates instead of periodic updates. In order to avoid switching to a 2BMEP with stale information, the node does not only advertise the change in best path as described earlier, but it also advertises that such new best path is its second, possibly stale, best path. When the node's neighbors receive this advertisement, knowing that the node has switched to its 2BMEP, they will inform the node of their current $BMEP^{K_1}$ if this is better, in terms of ME, of the second best path the node just switched to. In receiving such information, the node may then decide to switch to one of these paths instead of continuing using the stale 2BMEP.

As data and control packets keep flowing through the network, nodes consume energy. Eventually, the energy levels of some nodes will reduce to the point where the rounds of encryption may soon start becoming an active constraint on the routing. This is Phase 2.

## 3.5 Phase 2: Moderate energy

As soon as $R_i + R_{BMEP^{K_1}}$ at the i-th node drops below $K_2 * M$, the i-th node enters Phase 2. In such a phase, the node operates as in Phase 1 in terms of data forwarding and control plane advertisements, but it also begins collecting data that will be relevant when it enters Phase 3. Specifically, nodes in Phase 2, in addition to their $BMEP^{K_1}$ and $2BMEP^{K_1}$, also begin caching the Best Path as per R (BPR) and up to additional N-3 paths.[8] Nodes will also start to piggyback their own $\alpha_i^R$ (i.e., the amount of node energy allotted exclusively to encryption operations) in both control and data packets so to pass them to their 1-hop neighbors. At the same time, they begin to track neighbors' encryption energy levels ($\alpha^R$) by listening to both control and data packets and recording $\alpha^R$ information, if available. Collecting such additional data does not consume extra energy as the data is piggybacked and nodes are not forced to stay awake just to collect this data.

Eventually, some nodes will have paths with $R_i + R_{BMEP^{K_1}} < M$, at which point R becomes an active constraint on the routing. When this happens, the notion of upstream and downstream neighbors breaks down[9] and the nodes move to Phase 3.

## 3.6 Phase 3: Low energy

When R begins to become a constraint, we start triggering updates based on R. Specifically, in Phase 3, *broadcast updates based on R are triggered when nodes get packets with pending rounds of encryption greater than what they can handle*. Since 2BMEP is no longer tracked in Phase 3, BMEP advertisements cannot be triggered based on that. Instead, BMEP updates are now sent along with BPR advertisements. However, the $\alpha^R$ advertising behavior from Phase 2 continues also in Phase 3. Furthermore, as in Phases 1 and 2, the BMEP is advertised via broadcast whenever it changes.

[8] We assume a cache of size N.

[9] Each packet can be routed in a different direction, even backwards, as a function of the number of rounds of encryption needed.

More specifically, when $R_i + R_{BMEP^{K_1}} < M$, node i enters Phase 3 and data-plane forwarding, path caching, and advertising, are done in a slightly different way than before. In particular, all N paths in the cache can be used to forward data. Furthermore, a node uses and advertises BMEP and BPR as described below.

On the control plane front, a node in Phase 3 acts according to the following behavior.

Receiving a packet that needs more rounds of encryption than $R_i + R_{BPR}$ is treated by a node as a symptom of stale information at the sender node and triggers the broadcasting of the node's BPR to its neighbors. In doing so, neighboring nodes will receive an update on the current value of R for that node's BPR.

On the data plane (packet-forwarding) front, a node in Phase 3 acts according to the following behavior.

It looks at all paths in its cache that satisfy the R constraint for the received packet (i.e., how many rounds of encryption are left to be performed on such packet). If there is only one path satisfying R then that one is used; if there are multiple paths, however, for each pair of paths the node looks at the difference in ME ($\Delta ME_{ij}$) and in the number of hops to the sink ($\Delta \# H_{ij}$) between the paths. After doing so, the node picks the path u that maximizes $\Delta ME_{uv} * K_4 - \Delta \# H_{uv} * Tx$, where Tx is the transmission energy expended for a single-hop transmission of one packet. If no paths satisfying the R requirement exist, the node tries to do the extra rounds of encryption by itself (i.e., a total of $R_i + R_{extra}$) and, if that is feasible, it then forwards the packet along its BPR where the remaining rounds will be done. If the node, however, cannot do the extra rounds of encryption, it does the planned $R_i$ rounds and then loops the packet back to a neighbor that is present in its neighbors table but not present in its cache. Specifically, the neighbor with the highest $\alpha^R$ is picked as the next hop. If, however, $R_i = 0$ and the chosen highest $\alpha^R$ neighbor is the original sender, the packet is dropped as looping it back may lead to a forwarding back and forth of the packet between the two nodes. When deciding to loop a packet back, the node ensures the packet has been marked as a "loopback" packet and adds or updates the TTL field.

In the next section we will discuss the loopback behavior in more detail.

## 3.7 Best-Effort Forwarding: Loopback

In Phase 3, when a node receives a packet that requires a number of rounds of encryption beyond the capabilities of the best R path known by the node ($R_{BPR}$) plus the node's own $R_i$, the packet is not immediately dropped. There is still a chance that such packet can be encrypted if higher energy nodes in the system can be leveraged. On the other hand, however, transmitting a packet back and forth for an excessive number of times is also not desirable as it may consume a significant amount of transmission energy. There are, therefore, two scenarios.

If the node receiving such a packet, $Node_i$, has enough encryption energy ($\alpha_i^R > \varepsilon_3$) to do some extra rounds of encryption, it does the extra rounds (i.e., beyond the advertised $R_i + R_{BPR}$) required by the packet. This avoids the need to send the packet "backwards" for encryption rather than forward towards the sink. Furthermore, even though the node spends more energy on this packet than what it originally allotted for encryption, it treats such an event as an indication of stale information at one or more of its neighbors and, therefore, triggers a BPR update. In doing so, neighbors now know the updated number of rounds of encryption a packet can undergo when routed through $Node_i$'s BPR. In other words, downstream nodes are not explicitly trying to exploit an extra rounds of encryption, they just have stale information on R for that path and since such information being stale can now create problems (i.e. beyond the capability of $R_{BPR} + R_i$), it is immediately resolved by triggering an update.

If, on the other hand, $Node_i$ does not have enough energy ($\alpha_i^R < \varepsilon_3$) to perform the extra rounds of encryption beyond $R_i$, it attempts to "loopback" the packet to nodes further away from the sink. Specifically, after doing its fair share of $R_i$ rounds, $Node_i$ tries to send the packet to one of its neighbors, which is not a next hop in its cache. The assumption is that next-hop nodes (i.e., hierarchically higher nodes) in the cache are important for connectivity to the sink, hence should not consume more energy with extra rounds of encryption beyond what they have advertised. In order to pick the appropriate neighbor for loopback, $Node_i$ looks at the available encryption energy $\alpha^R$ of all candidate neighbors in the neighbors table. In particular, among all eligible nodes, the node with the highest $\alpha^R$ is picked as it has the maximum encryption

energy.[10] The packet, if not already so, is marked as a "loopback" packet and a TTL value is added to the packet. The TTL value is equal to the number of remaining rounds of encryption (after $Node_i$ has performed $R_i$ rounds) divided by 2 and then rounded up. Once the loopback process begins, the packet TTL is decremented by 1 at each node until either encryption completes or the TTL expires, in which case the packet is dropped. Thus, packets requiring a number of rounds of encryption above $R_i + R_{BPR}$ are dropped only when either no eligible "loopback" node can be found or the TTL has expired.

The value of $\alpha_i^R$ is sent to one-hop neighbors when they are in Phases 2 and 3. Specifically, all control packets, except going-to-sleep advertisements, include $\alpha^R$ values. Furthermore, data packets can also piggyback $\alpha^R$ values if the current $\alpha^R$ has either increased or decreased by a factor of $K_3$ since the last time it was advertised. All neighbors hearing the packet check for an indicator field in the packet to see if an $\alpha^R$ value has been piggybacked on the packet or not and, if it has, they store it.

## 3.8 Sleeping Nodes

Generally speaking, when a node's energy goes below the sleep threshold or the node does not receive any packet for a certain amount of time, the node broadcasts to its one-hop neighbors a control plane message indicating that it is going to sleep and sets its state to SLEEP and phase to UKNOWN. Once the energy level of the sleeping node increases (i.e., by harvesting energy) and crosses a certain threshold, the node will go back to the AWAKENING state. Nodes receiving going-to-sleep notifications from neighbors, delete such neighbors from their cache and neighbors tables.

In particular, a node decides to go to sleep when its total energy $\alpha$ drops below a certain threshold $\varepsilon_4$. Once a node decides to go to sleep, it broadcasts this decision in a Going to Sleep (GTS) control packet. Upon receiving this GTS packet, its neighbors clear all entries for that node in both their cache and neighbors table. Similarly, the node going to sleep clears its cache and neighbors table, and resets all variables to their default values. The node will continue harvesting energy while sleeping and

[10] Ties are broken randomly.

will wake up once it detects that its total energy $\alpha$ is above a certain value $\varepsilon_1$. When this happens, the node sets its state to AWAKENING, sends out path solicitations and processes them identically as when in the INITIAL state. The only difference here is that the node is not assumed to be in Phase 1. The actual phase is determined after identifying the BMEP and computing $R_i + R_{BMEP^{K_1}}$.

A node may also decide to go to sleep if its cache is empty (i.e., it has no valid path to the sink) and its current total energy $\alpha$ is below a threshold $\varepsilon_5$ where $\varepsilon_5 > \varepsilon_4$.

When nodes that have been sleeping because of low energy wake up, they send a broadcast advertisement with a NULL route to all their neighbors so to request information on all available routes to the sink. Neighbors reply to this advertisement with their current BMEP and, if they are in Phase 3, with their BPR. If no response is received to such a request, neighboring nodes are assumed to not have any path to the sink hence the requesting node must re-probe after a timeout. The use of a re-probing timeout allows nodes that currently do not have any path to the sink to ignore route requests by other nodes (i.e., they do not reply with "I do not have any route" and waste transmission energy).

## 3.9 Triggered Advertisements

When, for some reason, a node ends up directly transitioning from Phase 1 to Phase 3 without having had the benefit of collecting R paths in Phase 2, the node solicits BPR from its neighbors via broadcast packets. In this case re-probing is not required for these nodes since their cache is not empty. Also, the change in BMEP, that caused the transition from Phase1 to Phase 3, must be immediately advertised to avoid loops and should not be suppressed or delayed.

## 3.10 Fudge Factor

The general intuition behind the fudge factor is to use it in order to prevent needless advertising and churn while not causing unnecessary staleness. Accordingly, $K_1$ is only applied when a node is deciding whether or not to switch to a new path and

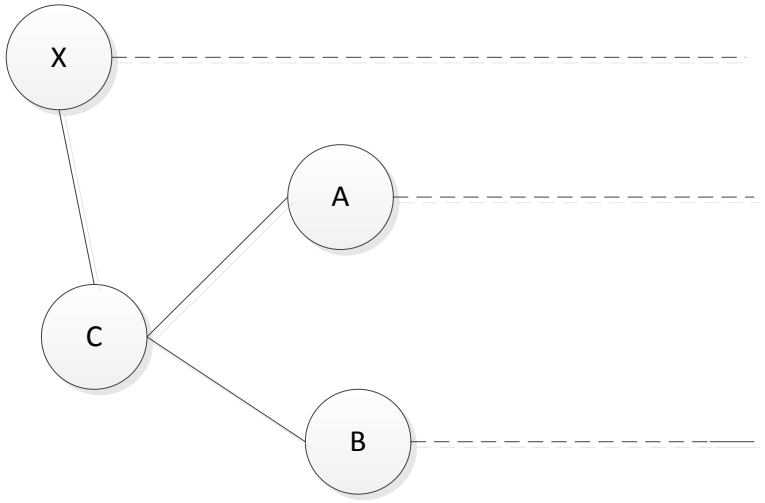



Figure 3.3: Use of the fudge factor.

whether or not to unicast the latest BMEP to a downstream node (except when switching to 2BMEP); $K_1$ is not used when a downstream node is deciding whether or not to piggyback $ME_{2BMEP}$ information upstream. Not applying it to the piggybacking case prevents the scenario where both downstream and upstream parties apply the fudge factor of $K_1$ when updating each other, resulting in an overall fudge factor of $(K_1)^2$.

As mentioned earlier, a node does not apply the fudge factor when it is deciding whether or not to send a unicast advertisement of its BMEP in response to a broadcast advertisement indicating that a neighbor is switching to its 2BMEP. Although this may increase churn, the rationale behind this behavior is that, in this scenario, the additional churn is tolerable in the interest of updating the neighbor with current information. In particular, the neighbor should have up-to-date information regarding other relevant paths to the sink, in view of the fact that our system does not have periodic updates. Note that in this scenario both nodes in question are in either Phase 1 or 2 (i.e. have high to moderate levels of energy). An example of the scenario described above follows.

In Fig. 3.3, solid lines indicate neighbor relationships, while dashed lines indicate the existence of a path from the node towards the sink that does not involve any of the other nodes in the picture.

Say that at time $t = T_1$, node C ends up picking the path via node A as its BMEP, and the path via node B as its 2BMEP. Now at time $t = T_2$, node C switches to the path via node B and promotes it to BMEP. Then, node C will send out a broadcast advertisement announcing the path via node B as its new BMEP and also turns on the indicator that it is switching to its 2BMEP. When node X hears this broadcast, it realizes that its BMEP has a better ME value than the path node C has chosen via B. Then, irrespective of the $K_1$ fudge factor, node X will advertise its BMEP to node C as a unicast. At that point, node C will decide if to switch to the path via node X or not.

## 3.11 Caching Algorithm

The CACHE algorithm is designed for a node to store the most useful paths as a function of its phase.

In Phase 1, the CACHE has only two entries and merely consists of the BMEP and the 2BMEP. In Phases 2 and 3 the CACHE has size N. In particular, in Phase 2, it consists of BMEP, 2BMEP, the BPR and additional N-3 paths. In Phase 3 it consists of the BMEP, BPR and additional N-2 paths. In other words, the following is true:

- In all three phases the best path as per ME is always cached irrespective of R;
- in phases 1 and 2, the $2^{nd}$ best path as per ME is always cached irrespective of R;
- in phases 2 and 3, the best path as per R is always cached.

Paths are added until either the cache is full or there are no more paths left to consider. In phases 2 and 3, for the remaining slots (N-3 and N-2, respectively), the preferred order for caching paths is as follows:

1. Add paths with R > M in descending order as per ME;

2. if cache is not full, add paths with $R > R_{stat}$, in descending order as per ME;

3. if cache is not full, add paths in descending order as per R.

Here, $R_{stat}$ is a statistical quantity based on the per-packet number of rounds of encryption that a node has been doing in the (recent) past. For example, $R_{stat}$ could be the number of rounds of encryption needed by the packets that the node has encrypted that fall within 1 standard deviation above the mean number of rounds.

The justification for the priority ordering described above is, respectively, that:

1. If there exist paths that can do M rounds of encryption per packet, all packets can be routed along them (since no packet requires more than M rounds), so we keep the highest ME paths;

2. when paths with fewer than M rounds of encryption have to be stored, look for paths that can do enough rounds to satisfy a substantial chunk of the packets the node seems to receive (e.g. 1 standard deviation above mean covers $\tilde{6}6\%$ of the packets, if the distribution is Gaussian), and among them pick as per ME;

3. when neither of those two conditions holds, we optimize over R.

Details about nodes energy states, energy computation and translation from $\alpha_i^R$ to $R_i$ are presented in Appendix A.

# Chapter 4

# Cellular Networks

## 4.1 Introduction

Over the years, the cellular network has evolved from a wireless and mobile alternative to making and receiving phone calls, to a complex system providing phone service and Internet connectivity as well as interworking with other technologies (i.e., WiFi), while providing a seamless experience. Generally speaking, we can identify the following two parts in a cellular network: the *access* network, which comprises the air interface, and the *core* network. Over the years, the performance of the air interface has increased drastically, leading to high throughput and low latency hence keeping up to speed with consumers' expectations. On the other hand, changes in the core network have been slower and incremental, with an increased complexity accentuated by the business necessity to continue supporting older technologies such as 2G (i.e., GSM). Because of this, today's core network architecture resembles what is shown in Fig. 4.1: a combination of multiple core networks from different technologies such as 2G, 3G and Long Term Evolution (LTE). All these technologies include different "boxes" with different names and similar functions. The basic architecture, however, has not significantly changed over the years. In particular, with LTE, the signaling and data planes have been decoupled but the overall architecture has remained the same: a monolithic architecture where we have a limited number of servers, gateways

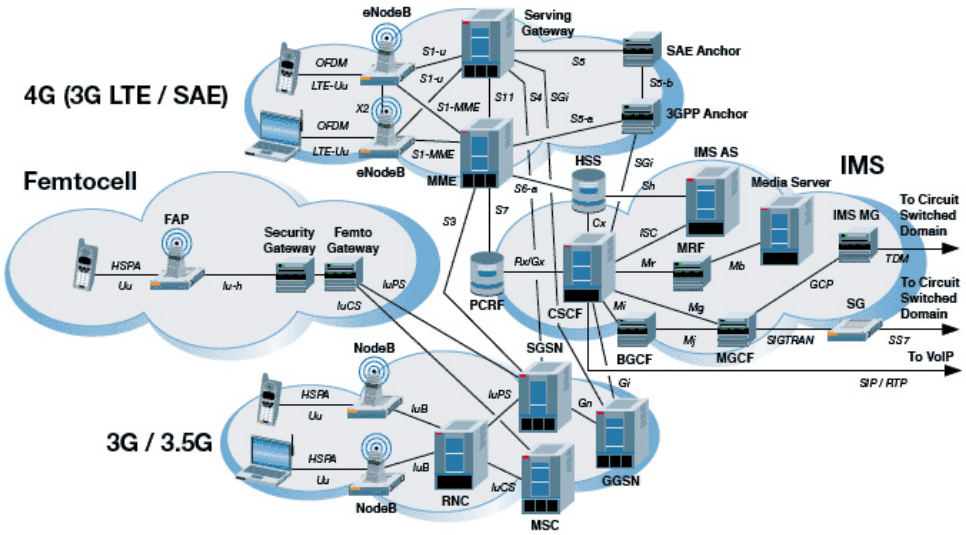



Figure 4.1: A typical cellular network architecture.

and databases serving all User Equipments (UEs).

In the following, we give an overview of LTE basic concepts and operations.

## 4.2 Core Network Architecture

Fig. 4.2 shows a high-level simplified view of the LTE architecture and its main network elements. We focus mostly on the $4^{th}$ Generation (4G) Core Network (CN), also known as Evolved Packet Core (EPC). Due to historical reasons, going back to circuit-switched networks, the EPC has been designed as an IP-over-MPLS (Multi-Protocol Label Switching) network on top of which tunneled IP traffic is conveyed. When discussing our new architecture, in Chapter 5, we will look at such inefficiencies in more detail. In the following, we give an overview of the most important network elements of the EPC.

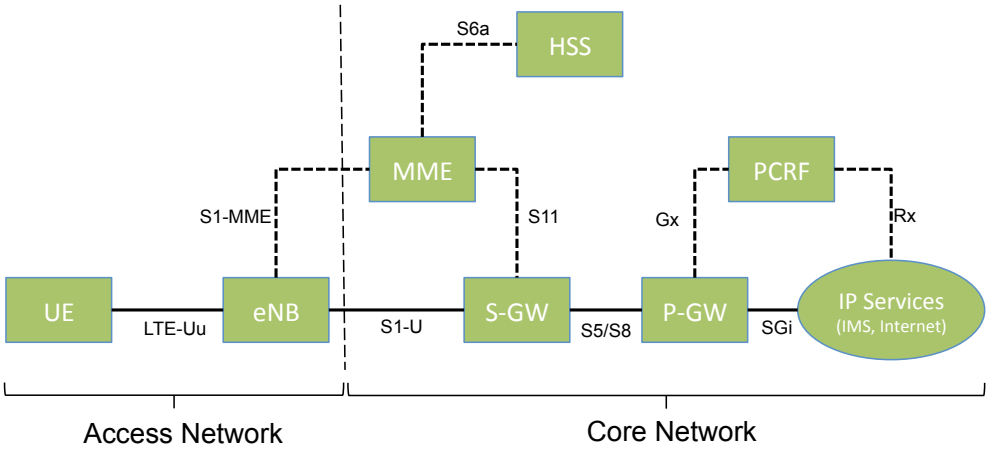



Figure 4.2: LTE cellular network architecture.

### 4.2.1 MME

The Mobility Management Entity (MME) is the main signaling node in the EPC. During the User Equipment (UE) *Attach* procedure[1] with the network, the MME is responsible for helping authenticate and authorize the UE (by interacting with the HSS), inform the eNodeB on which S-GW and P-GW to use for that UE and help establishing the required *bearers* (e.g., tunnels with an associated Quality of Service (QoS)) with the UE in order to provide IP connectivity to the UE. Furthermore, it saves UEs tracking area information and is responsible for the paging procedure. It performs mobility management by supporting different types of handover which may involve a change in MME, S-GW, P-GW. Also, it performs bearer management, roaming management, lawful intercept and load balancing between S-GWs.

### 4.2.2 HSS

The Home Subscriber Server (HSS) is the equivalent of the Home Location Register (HLR) plus the Authentication Center (AuC) in previous cellular technologies (e.g.,

[1] The UE needs to "attach" to the cellular network in order to access voice and data services.

3G). The HSS stores and updates users' subscription information including users identifiers such as the International Mobile Subscriber Identity (IMSI) and the Mobile Subscriber ISDN Number (MSISDN) or mobile telephone number. Furthermore, it also includes user subscription states and the QoS level the user subscribed to (i.e., maximum allowed bit-rate, allowed traffic class). Also, the HSS interacts with other network elements such as a Diameter Server in order to provide the security information needed to perform, together with the MME, mutual authentication between UE and cellular network. In particular, in 4G networks the UE authenticates with the network and the network authenticates with the UE (i.e., mutual authentication). This is done in order to prevent unauthorized UEs from accessing the network and rogue base station (i.e., eNodeBs) from impersonating the network and eavesdrop on UEs communication. The security information provided by the HSS is also used for radio-path ciphering and integrity protection in order to prevent other types of eavesdropping and manipulation of traffic between the UE and the cellular network.

### 4.2.3 S-GW

The Serving Gateway (S-GW) is the termination point of the packet data interface *towards the access network* (i.e., eNodeBs). When a UE moves across eNodeBs, the S-GW functions as the mobility anchor for: intra-LTE handover, inter-MME handover, and for mobility with other cellular technologies such as 2G (i.e., GSM) and 3G (i.e., UMTS).

### 4.2.4 P-GW

The Packet Data Network Gateway (PDN-GW or P-GW), is the termination point of the packet data interface *towards the external Packet Data Network* (i.e., the public Internet). In other words, the P-GW is the gateway to the Internet and other IP networks (e.g., IMS), it performs IP address allocation to the UEs, it supports policy enforcement and charging as well as packet filtering (e.g., deep packet inspection).

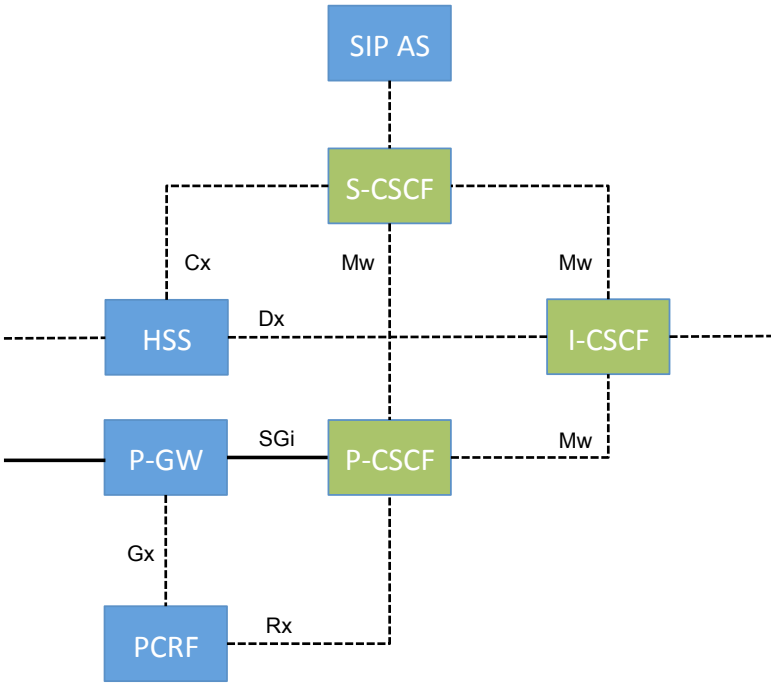



Figure 4.3: Main network elements in the IMS and corresponding interfaces with the EPC.

### 4.2.5 PCRF

The Policy and Charging Rules Function (PCRF) server is the equivalent of the Policy Decision Function (PDF) plus the Charging Rules Function (CRF) in 3G. It manages service policy, it sets QoS for each user session and manages accounting rules. In particular, the PCRF enforces policy decisions based on rules of the network operator and, for example, it decides if to allow or reject media requests and allocation of new resources for a user. Similarly, it applies charging rules to each user data flow such as video and voice streams.

## 4.3 IP Multimedia Subsystem

The IP Multimedia Subsystem [23] (IMS) is the next evolution in fixed and mobile network access and it is currently being deployed in many cellular networks. In the IMS, different access-network technologies converge under one single IP architecture. This convergence allows for the delivery of Internet services (e.g., Voice over LTE) regardless of the access network used. We can think of the IMS as an abstraction layer between the service layer and the transport layer.

The Session Initiation Protocol (SIP) [22] has been chosen as the standard signaling protocol for the IMS and different servers and proxies are used in the IMS to process SIP signaling. In particular, as we can see in Fig. 4.3, we have three main network elements forming a functional group called Call Session Control Function (CSCF): the Proxy-CSCF (P-CSCF), the Interrogating-CSCF (I-CSCF) and the Serving-CSCF (S-CSCF).

### 4.3.1 P-CSCF

The P-CSCF is a SIP Proxy server and represents the first point of contact the UE has with the IMS. It gets assigned to the UE before the SIP registration occurs, either by DHCP or during the *Attach* procedure to the cellular network. Once a P-CSCF has been assigned to a UE, it does not change for the duration of a SIP registration. All SIP signaling between UE and IMS goes through the P-CSCF. All communication between UE and P-CSCF is over IPsec or TLS and the P-CSCF can compress and decompress SIP signaling using SigComp [47]. Furthermore, the P-CSCF can include a policy decision function, similar to the PCRF, authorizing media plane resources, QoS levels, policy control and bandwidth management. Lastly, it can generate charging records.

### 4.3.2 I-CSCF

The I-CSCF is located at the edge of an administrative domain and can be used by remote servers as a forwarding point to its own domain. The IP address of the I-CSCF

is published in the DNS of the domain it serves (e.g., using NAPTR or SRV type records) so that remote servers can discover it and forward SIP registration packets, for example, to its domain. In particular, the I-CSCF queries the HSS in order to retrieve the address of the S-CSCF to assign to a user who wants to register with the IMS while outside of its administrative domain. Furthermore, it forwards requests and responses coming from outside the IMS administrative domain to the S-CSCF.

### 4.3.3 S-CSCF

The S-CSCF is a SIP server and it represents the central element of the IMS signaling plane. It is always located in the home network and interacts with the HSS in order to exchange user profiles and user-to-S-CSCF associations. In particular, the S-CSCF handles SIP registrations, it is on the signaling path of registered users and can inspect messages. The S-CSCF decides to which Application Server[2] (AS) SIP messages should be forwarded in order to receive a certain service. Also, it enforces policies of the network operator. For load balancing and high availability, multiple S-CSCF servers can be deployed in the same network.

## 4.4 Network Attachment

Before being able to "talk" to the EPC, the UE needs to synchronize with the network in both the uplink and the downlink directions and establish communication with the eNodeB. In order to do this, the UE follows a Random Access procedure for synchronization purposes and it then establishes an RRC connection with the eNodeB [48]. Once this is done, the UE can initiate the attach procedure [49] with the EPC. Fig. 4.4 shows a high-level view of the UE network attach procedure and IMS registration. In particular, when the UE initiate the attach procedure with the EPC, the eNodeB needs to discover which MME to assign to such UE. In order to do this, the eNodeB queries an internal DNS (iDNS) of the network operator which returns the IP address of the MME to use. In particular, every eNodeB has a pre-configured list of iDNS

[2] An application server hosts and executes services (e.g., Voice Call Continuity).

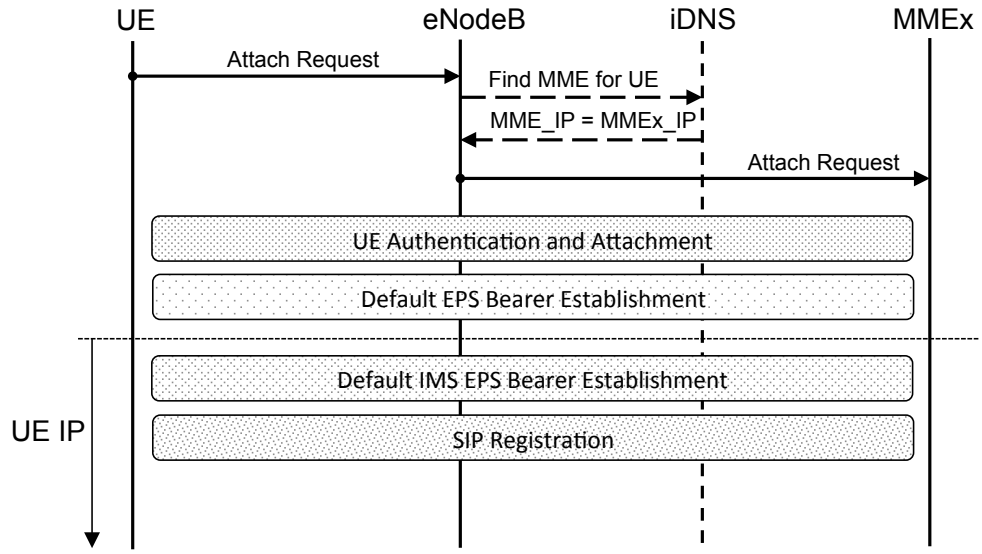



Figure 4.4: UE attachment and SIP registration.

nodes to use for MME discovery. Once an MME has been assigned to the UE, the UE authenticates with the network and gets assigned an S-GW and a P-GW by its MME. Lastly, a default bearer (i.e., EPS bearer) is created between UE and P-GW in order to provide voice and data services to the UE. A *bearer* is 3GPP terminology for a tunnel with an associated QoS. As we will discuss in Chapter 5, a default bearer has an IP address associated with it. Dedicated bearers with different QoS can also be created but they are usually bundled with the default bearer and, as such, share the same IP address of the default bearer. Once a default bearer has been established with the P-GW (i.e., default EPS bearer), the UE has IP connectivity.

In order for the UE to have access to IMS services, a separate default IMS bearer must be established with the EPC. The default IMS bearer is established at either UE attachment or when a SIP registration occurs. The default IMS bearer has a different QoS and IP address than the default EPS bearer and is typically used for SIP signaling and Voice over LTE (VoLTE) service. In particular, SIP signaling uses the default IMS

bearer while voice-data packets use a dedicated IMS bearer with higher QoS that is bundled with the default IMS bearer and created at the time of a VoLTE call. As in the case of the EPS bearer, the default IMS bearer and the dedicated IMS bearer are bundled together, have different QoS, but share the same IP address that was assigned to the default IMS bearer.

# Chapter 5

# A Next-generation P2P Virtual Core Network

## 5.1 Introduction

Recently, with the introduction of Software-Defined Networking (SDN) and Network Function Virtualization (NFV), there have been multiple proposals in both academia and industry to move from a hardware-centric architecture to a software-centric architecture. In this new software-centric paradigm, the hardware deployed in the core network is generic hardware used to run Virtual Machines (VMs) implementing, in software, different functions of the core network. Fig. 5.1 shows an example of a 4G LTE virtualized core network where one can run single and multiple instances of core–network elements as deemed necessary by the current state of the network. Such an approach introduces important changes and innovations in the core network, making the whole network more cost-effective while giving more flexibility and control over the resources spent at any given time. However, such an approach continues mimicking, in software, the thirty-year-old architecture discussed earlier. Even though we now have powerful tools like SDN and NFV, no paradigm shift has really occurred in the core network.

By leveraging the flexibility of SDN and NFV, we propose a drastically different

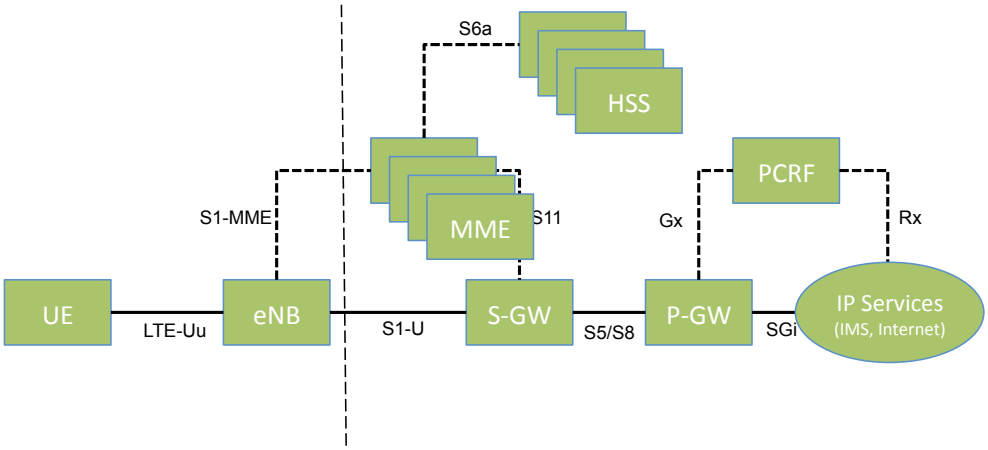



Figure 5.1: VM-based LTE cellular network architecture with multiple virtualized MMEs and HSSs.

architecture of the core network, with each UE getting its own "private" virtual copy of the Enhanced Packet Core (EPC) which we have denoted as Virtual Core Network (VCN). In particular, as we show in Fig. 5.2, in the new architecture, the core network is a P2P network of VMs. Each VM functions as either a VCN or a SuperNode (SN). By leveraging NFV, SNs and VCNs implement all the typical functions of an EPC (e.g., MME, S-GW, P-GW, HSS). However, as we will see later, while a VCN serves a single UE and has, typically, a short lifetime, SNs serve a group of UEs, are responsible for assigning VCNs to UEs and have a very long lifetime.

In the following, we look in detail into the VCN architecture shown in Fig. 5.2, which presents many challenges but, at the same time, has many significant advantages over the current architecture. In particular, due to its highly distributed and personalized nature, the new architecture, for example, disables cellular botnets from performing Distributed Denial of Service (DDoS) attacks and, at the same time, allows to implement per-UE custom security and services. It is now possible, for example, to host a UE's core network (i.e., its VCN) on: public cloud infrastructure (e.g., Amazon AWS, Microsoft Azure); private cloud infrastructure (e.g., Government, enterprise);

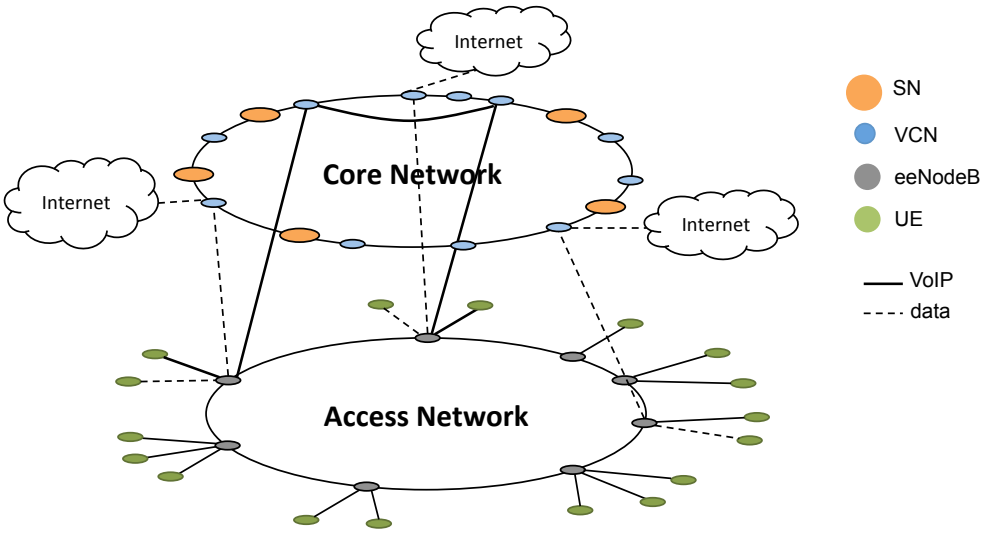



Figure 5.2: New virtualized cellular architecture.

cellular operator infrastructure. Furthermore, due its single IP-layer architecture (see Section 5.3.3) and VCN's live migration (see Section 5.7), the new core network improves on overall latency and delay.

Besides classical eNodeBs, we also consider a second scenario with an enhanced eNodeB, thus breaking the assumption of backwards compatibility with existing access networks. Furthermore, we discuss additional aspects of the new architecture such as IP addressing and non-security-related novel services enabled by the new architecture. Finally, we add additional cellular traffic measurements in order to better study the feasibility of the new architecture and fine-tune some of its most critical parameters.

## 5.2 Current Efforts

The scientific community and various standardization bodies (e.g., 5GPPP) have put a lot of effort towards new access networks for next-generation cellular networks [50,

51, 52]. Most approaches focus on harmonizing different access technologies (e.g., LiWi, WiFi, LTE) [53] and improve on spectrum efficiency [54, 55] also by deploying micro-cells and pico-cells [56, 57] with a specific focus on mm-wave [58, 59, 60, 61].

In the cellular core network, changes have been more incremental, with the most significant innovation, in recent times, being SDN and NFV. By leveraging SDN and NFV, relevant efforts focus on re-organizing the core network of operators and virtual operators on the cloud [62] and in many other ways [63, 64], but always keeping the same old architecture, although virtualized. In [65], the authors go in the opposite direction, to the extreme of completely removing the core network, with eNodeBs accessing directly the public Internet.

We re-think the cellular core network and propose a new P2P architecture that still leverages SDN and NFV but does not follow the same blueprint of existing and past cellular core networks. In doing so, we are able to provide a new level of network security and flexibility as well as novel services that would not be possible otherwise.

## 5.3 A Novel Architecture

While current efforts go in the direction of virtualizing single elements of the existing architecture (see Fig. 5.1), we propose a completely new virtualized architecture.

The new core network is a P2P network of VMs, where each VM can have the role of either an SN or a VCN. An SN is responsible for managing a certain number of UEs: this includes performing simple tasks (i.e., paging, voicemail and SMS delivery) as well as performing complex orchestration tasks (i.e., creating and destroying VMs that run VCN functionality). On the other hand, the only purpose of a VCN is to provide voice and data services to a single UE. It is important to notice that once a UE is associated with a VCN, the VCN will take over all tasks, including those previously handled by the SN with the exception of orchestration.

In the proposed architecture, two scenarios can be envisioned. In the *first scenario*, in order to be backwards-compatible with the existing access networks, we keep the existing protocol stack with tunneling. Due to historical reasons, in current cellular core networks tunneling is heavily used at a very fine granularity. For example, in

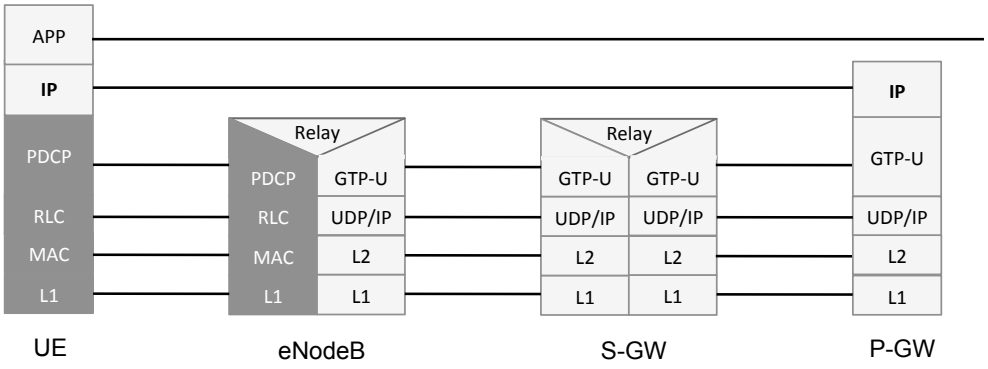



Figure 5.3: Current network protocols with IP-in-IP tunneling.

the downlink, UE's IP packets are fragmented and encapsulated in GPRS Tunneling Protocol (GTP) tunnels over UDP: in other words, the UE's IP layer is encapsulated over the underlying IP layer of the Multi-Protocol Label Switching (MPLS) core network. This encapsulation is shown in Fig. 5.3. Furthermore, a single UE often uses multiple tunnels to different gateways. This is highly inefficient and leads to high latency and long delays regardless of the access network. In the *second scenario*, one can remove the assumption of backwards-compatibility and introduce an enhanced eNodeB (eeNodeB). This allows to use one single MPLS network, with IP on top, for both core network nodes (i.e., SNs and VCNs) and UEs, thus avoiding the IP-in-IP tunneling typical of today's core networks.

Figs. 5.5a and 5.5b show the protocol stack of a new Core Network Node (CNN) for the user plane and the control plane, respectively. As mentioned above, when using the eeNodeB, the new core network uses MPLS with IP on top for both CNNs and UEs, and no IP-in-IP tunneling. Above IP, the usual transport layer protocols are considered and, at the application layer, CNNs must support RELOAD [66], SIP [22] and P2PSIP[1] [67] in the control plane and RTP and RTP-related protocols in the user plane. As will be described in more detail in Section 5.5, RELOAD is used in order for the UE to attach to the network, while SIP and P2PSIP are used for Voice over

[1] P2PSIP is the peer-to-peer version of SIP and relies on the RELOAD overlay.

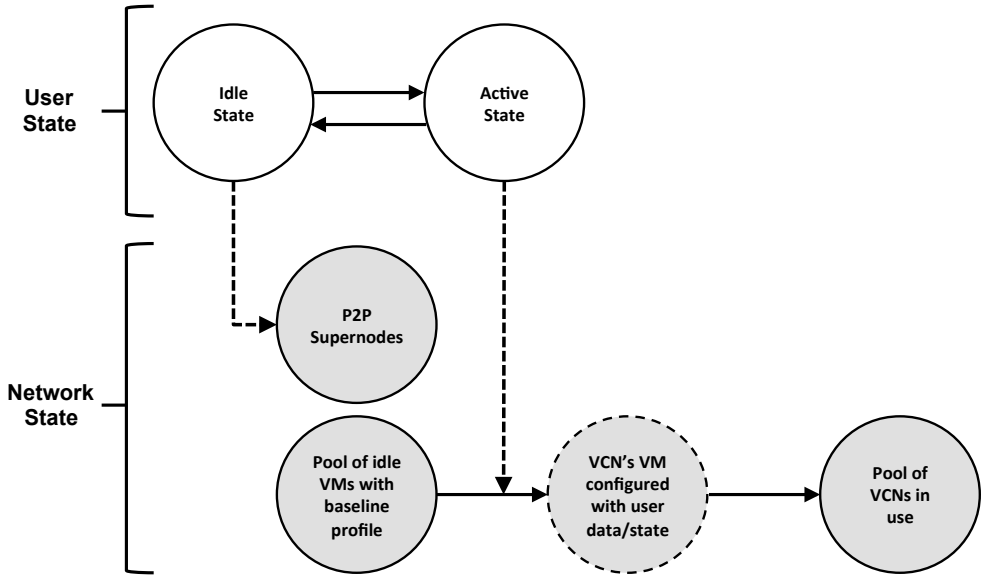


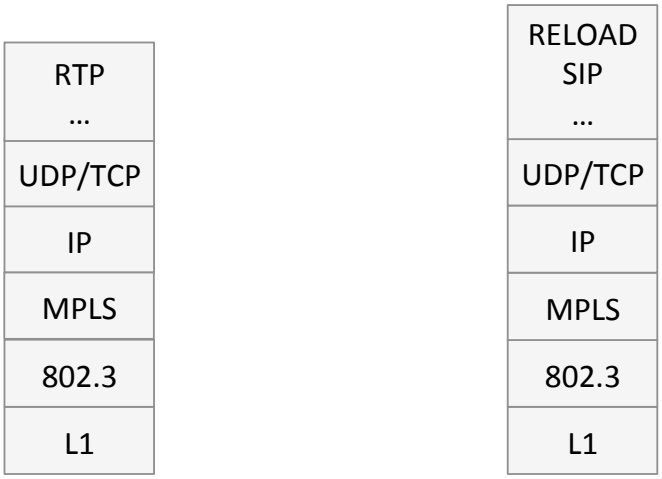

Figure 5.4: Orchestration.

LTE (VoLTE) call session setup. On the other hand, RTP is used for real-time media since a VCN always remains in the media path of a call.

Let us now look at the main three elements of the new architecture: SN, VCN and eeNodeB.

### 5.3.1 SuperNode

When a new user (UE) subscribes to a cellular network operator, his user information (e.g., SIM data, billing) is added to the operator's customer database and provisioned to an SN in the P2P network — how this provisioning happens and the SN selection process are out of scope and reserved for future study. The customer database is a distributed database hosted on the operator's private cloud, which is separate from the cellular core network hosting infrastructure. This database is also used to routinely store and backup UE's state.




(a) User Plane (b) Control Plane

Figure 5.5: Core-network node protocol stack.

The Distributed Hash Table (DHT) used in the P2P core network can be any DHT having as *key* an $ID_{UE} = hash(IMSI)$, where the International Mobile Subscriber Identifier (IMSI) is a unique identifier assigned to a subscriber [68], and as *value* a URL pointing to the UE's entry in the customer database. In other words, the DHT is characterized by the pair $\{key : ID_{UE}, val : DB\_URL\}$. The International Mobile Subscriber Identifier (IMSI) [68] is a unique identifier and is usually 15-digit long; its structure is shown in Fig. 5.6. The first three digits represent the Mobile Country Code (MCC) of the subscriber, the following two or three digits represent the Mobile Network Code (MNC) of the subscriber while the remaining digits represent the Mobile Subscriber Identification Number (MSIN). The MNC identifies the home Public Land Mobile Network (PLMN) of the subscriber and is important in roaming scenarios. We reserve roaming for future study.

The number of active SNs can either shrink or grow depending on network load, security, maintenance and other issues. When a new SN is added to the P2P network, the DHT re-organizes itself and the new SN gets a list of entries in the DHT. Such entries have pointers to the customer database so that the SN can retrieve the current

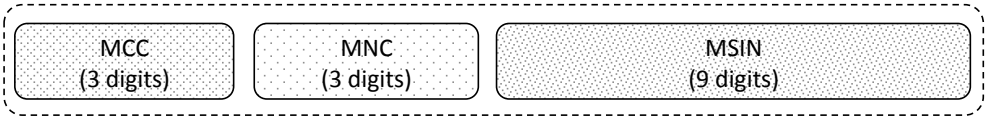



Figure 5.6: IMSI structure.

states of the UEs it has to serve from that moment on.

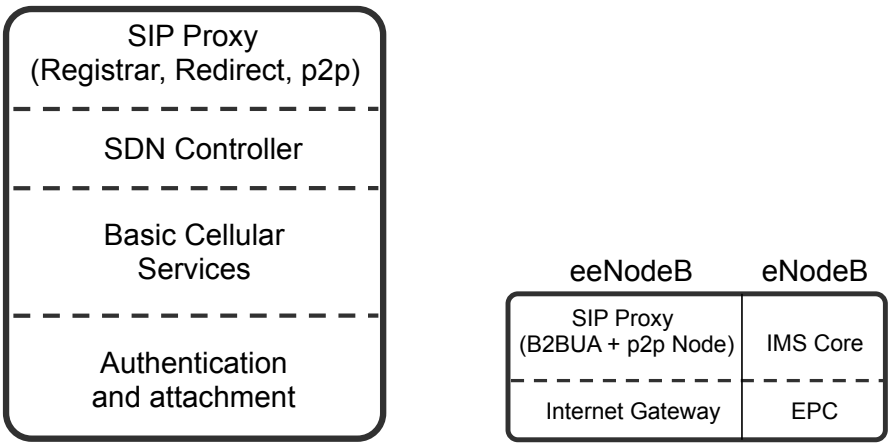
SNs in the P2P core network serve the following four different purposes, as we have summarized in Fig. 5.7a.

1. Since the SN is the UE's first point of attachment to the core network, it is responsible for the UE authentication during the attach procedure. The SN behaves as a standard EPC and, as such, performs the standard authentication process. If the authentication completes successfully, a default bearer is established between the UE and the SN. We discuss the attachment procedure in Section 5.5.

2. After a successful attachment of a UE to the network, and provided that the UE is not associated with a VCN, all basic signaling and "light-weight" tasks are handled by the SN the UE authenticated with. These tasks include: paging, SMS, push services (e.g., email notifications), voicemail.

3. An SN acts also as an orchestrator for its UEs. In particular, when a UE becomes *active* (e.g., initiating or receiving a call, browsing the Internet) its SN deploys a new VCN dedicated to that UE, updates the DHT accordingly (i.e., the UE is now reachable at its VCN) and informs the eNodeB (or eeNodeB) to redirect all traffic to that VCN. Once the UE goes back to *idle* mode and stays in such mode for a certain amount of time (see Section 5.12), the SN destroys the VM hosting the UE's VCN and triggers a "redirect" at the eNodeB so that all data and signaling are again sent to the SN from that point on.

4. The SN also acts as a SIP Registrar and re-direct server. After the UE has authenticated with the SN, it performs a SIP registration to it in order to be rea-




(a) SN (b) VCN

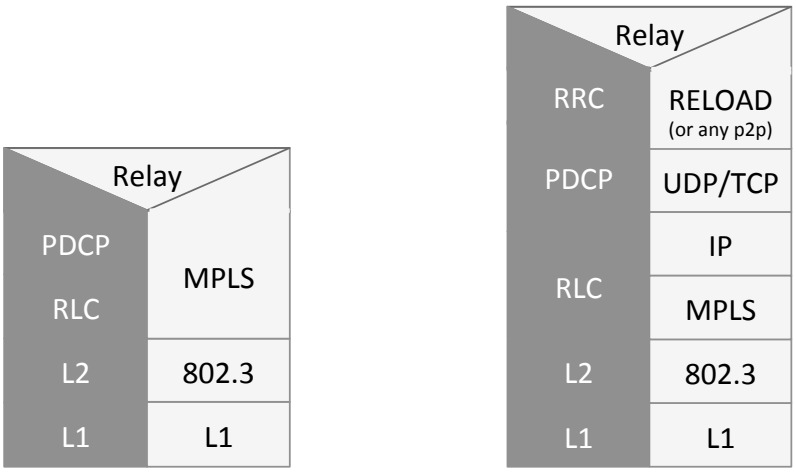
Figure 5.7: Node functionalities.

chable for VoLTE service as per standard SIP operations. The re-direct server functionality is used for redirecting the UE to the VCN and back to the SN, as described in Section 5.6.

### 5.3.2 Virtual Core Network

A VCN is dedicated to a single UE and handles: SIP signaling, real-time media and data traffic. In addition, it also handles all other functionalities typical of an EPC.

Fig. 5.7b summarizes the different behaviors of a VCN. In particular, when using a legacy eNodeB, a VCN implements a dedicated IMS core for VoLTE and a dedicated EPC for Internet access and all non-voice data traffic. When using an eeNodeB, a VCN implements: (i) a dedicated SIP Proxy server and a SIP Back-to-Back User Agent (B2BUA) for VoLTE and (ii) a dedicated Internet gateway for Internet access and all non-voice data traffic. The SIP B2BUA functionality is required as we want the VCN to remain in the call data-path. We will discuss call establishment in Section 5.5.




(a) User Plane (b) Control Plane

Figure 5.8: Enhanced eNodeB protocol stack.

### 5.3.3 Enhanced eNodeB

The key idea behind the eeNodeB is to modify the interface between eNodeB and core network to better take advantage of the new core network architecture. In particular, Figs. 5.8a and 5.8b show the protocol stacks for the eeNodeB referring to both user and control planes, respectively. As one can see, in the user plane the eeNodeB behaves essentially as an MPLS router while, in the control plane, it can "talk" directly to the P2P core network. In particular, in the latter case, it interacts with the P2P core network (i.e., queries the DHT) in order to identify the correct SN for a given UE and to route the traffic accordingly. Moreover, it handles redirects for traffic flows and notifies an SN of specific events (e.g., the UE has moved).

It is important to note that the LTE *Uu* interface between UE and eeNodeB has not been modified and is identical to that between UE and legacy eNodeB. This makes the new eeNodeB backwards-compatible with existing UEs.

## 5.4 IP addressing

An SN uses a private IP address space with its UEs and VCNs. In particular, in Figs. 5.9 and 5.10, the IP addresses labeled with an *IN* suffix belong to the private address space, while the ones labeled with an *OUT* suffix are public IP addresses as they have to be routable.

The IP address of the UE (i.e., a private IP address) does not change when switching between SN and VCN as it does not change its SIP Proxy server's IP address. This greatly simplifies the re-direction between VCN and SN, in terms of SIP and SDP payload, since the SDP information contained in the INVITE does not need to be modified when moving from SN to VCN. As we discuss in Section 5.6, the IP address of the default gateway (i.e., P-GW) can also be left unchanged when moving between SN and VCN. On the other hand, the IP addresses of the SN's Mobility Management Entity (MME) and of the VCN's MME need to be different. This is because, in order for the EPS bearer update procedure to work (see Section 5.6), the two MMEs need to be independently addressable by the eNodeB.

When a UE "moves" between SN and VCN, the external IP address of the UE changes as SN and VCN have different public IP addresses (i.e., addresses labeled with an *OUT* suffix in Figs. 5.9 and 5.10). This breaks all existing TCP connections: while this may seem a negligible issue, we have to remember that, today, smartphones keep several TCP connections open in order to enable, for example, push services. In order to address this issue we create a tunnel between SN and UE so that data on existing TCP connections is forwarded by the SN to the VCN and from this to the UE (see Fig. 5.11). Once all the "old" TCP connections have timed-out or have been closed, the tunnel is terminated by the SN.

## 5.5 Basic Operations

In this section, we describe some of the basic message flows when using the new architecture. We do this by considering two different scenarios. In the first scenario

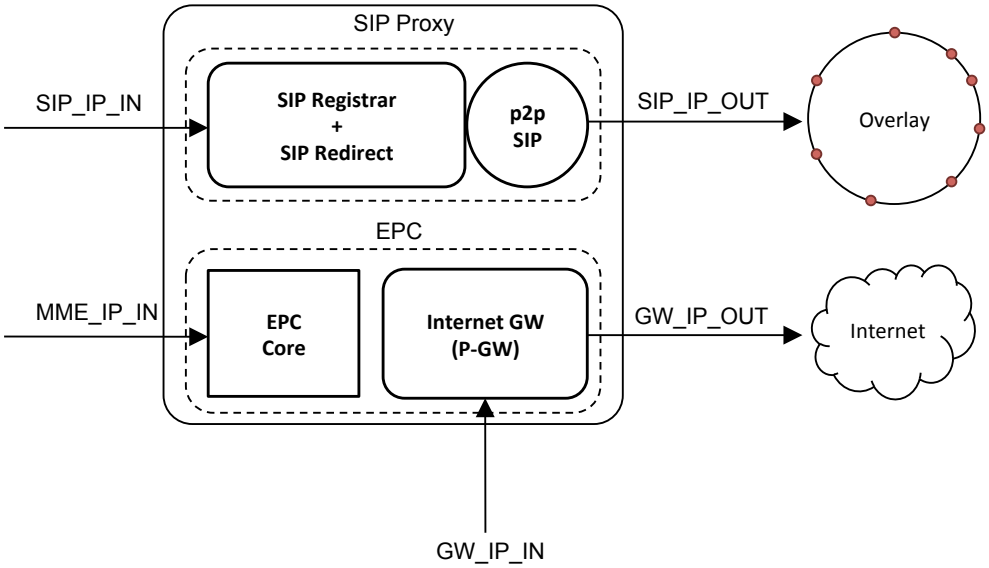



Figure 5.9: SN high-level architecture (SIP Proxy and EPC are multi-homed).

we consider standard eNodeBs, while in the second scenario we consider the new eeNodeBs.

### 5.5.1 UE Attachment and SIP Registration

At a very high level, in order to get Internet access a UE needs to be attached to and authenticated with the network. For VoLTE calls it also needs to successfully register with the IMS (i.e., with its SIP Registrar).

#### Scenario with eNodeB

In the current cellular architecture, when an eNodeB receives an attach request by a UE, it needs to discover which MME to connect to. This is usually done by querying an internal Domain Name System (iDNS) server which returns the IP address of the MME to use. Usually, one or more IP addresses of such DNS nodes are either

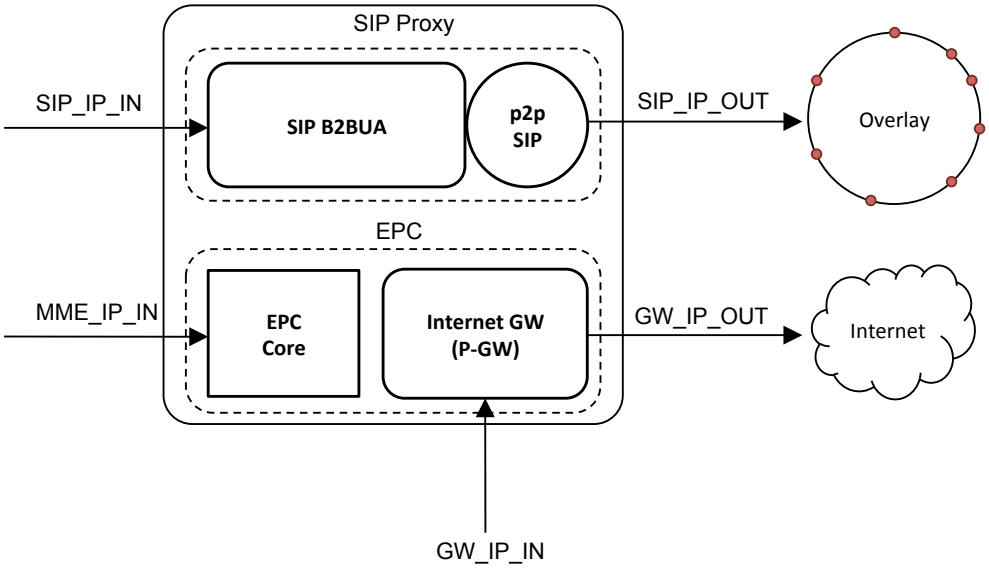



Figure 5.10: VCN high-level architecture (SIP Proxy and EPC are multi-homed).

statically configured at each eNodeB or are provided to the eNodeB by a DHCP server, for example. Once the eNodeB connects to the MME, during the UE attach, the MME queries the iDNS server in order to discover which P-GW and S-GW to use in order to serve the UE [69].

Fig. 5.12 shows how UE attachment and SIP registration[2] work, at a high level, in the proposed architecture when using a legacy eNodeB. When receiving an attach request, the eNodeB queries the iDNS server to discover which SN (i.e., its MME) to talk to for that UE and, once discovered, it performs the attach procedure as usual. As far as the eNodeB is concerned, it is "talking" to an MME which, in turn, directs it to a P-GW and S-GW as per standard procedure. In other words, the eNodeB is not aware of talking to an SN. After the attachment procedure completes, SIP authentication and registration can take place as per usual IMS procedure [70]. In our architecture, DNS node are nodes of the P2P core network and, while they expose a DNS interface

[2]For clarity, we show a simplified SIP registration flow.

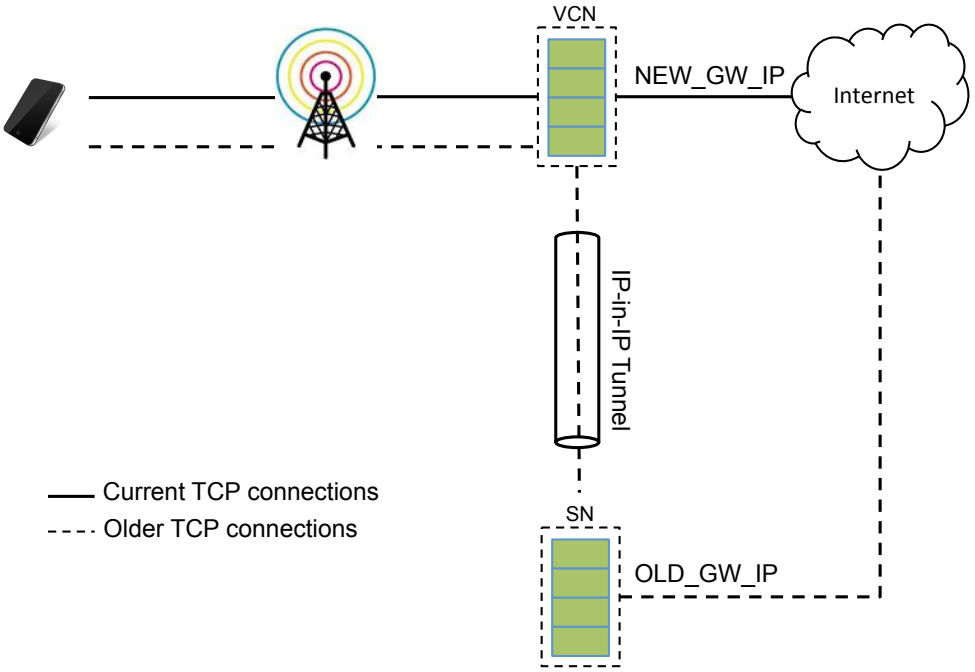



Figure 5.11: SN tunnels data from older TCP connections to the UE via the VCN.

to the eNodeB, they resolve DNS queries by interrogating the DHT of the P2P core network. This can be done by translating between the DNS protocol and the P2P protocol used by the core network (i.e., RELOAD) and viceversa.

### Scenario with eeNodeB

Fig. 5.13 shows UE attachment and SIP registration when using an eeNodeB. In such a scenario, the eeNodeB can directly "talk" to the P2P core network and directly query the DHT in order to find the SN responsible for a specific UE: in other words, no iDNS is necessary. Furthermore, when using eeNodeBs we do not have IP-in-IP tunneling and, therefore, the eeNodeB will only have to setup MPLS routes with the SN which are then used to route IP packets between the UE and the SN. The only bearer left, when using an eeNodeB, is the radio bearer on the *Uu* interface

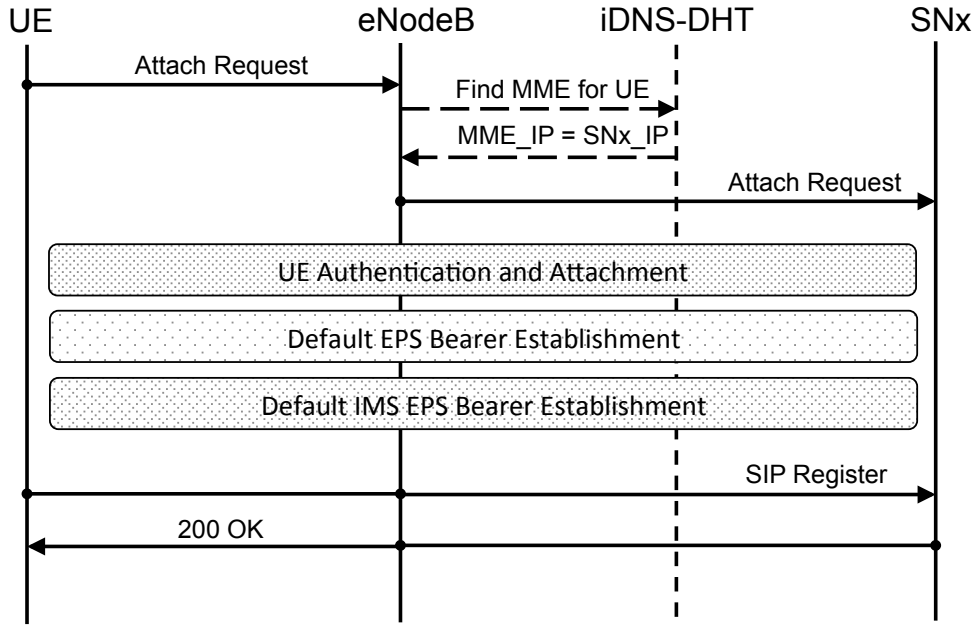



Figure 5.12: UE attach and SIP registration with eNodeB.

between UE and eeNodeB. For establishing MPLS routes, the SN behaves as an SDN controller for that UE and takes care of establishing such routes by leveraging OpenFlow [71], ReSource ReserVation Protocol - Traffic Engineering (RSVP-TE) [72] or any other similar protocol.

In both scenarios (i.e., with eNodeB and with eeNodeB), once the UE performs a SIP registration with the SN, the SN updates the P2PSIP overlay with such information.

### 5.5.2 VoLTE Call Flow

We describe the steps involved in VoLTE call establishment in the new architecture. In particular, we present two scenarios, one with eNodeBs and the other with eeNodeBs.

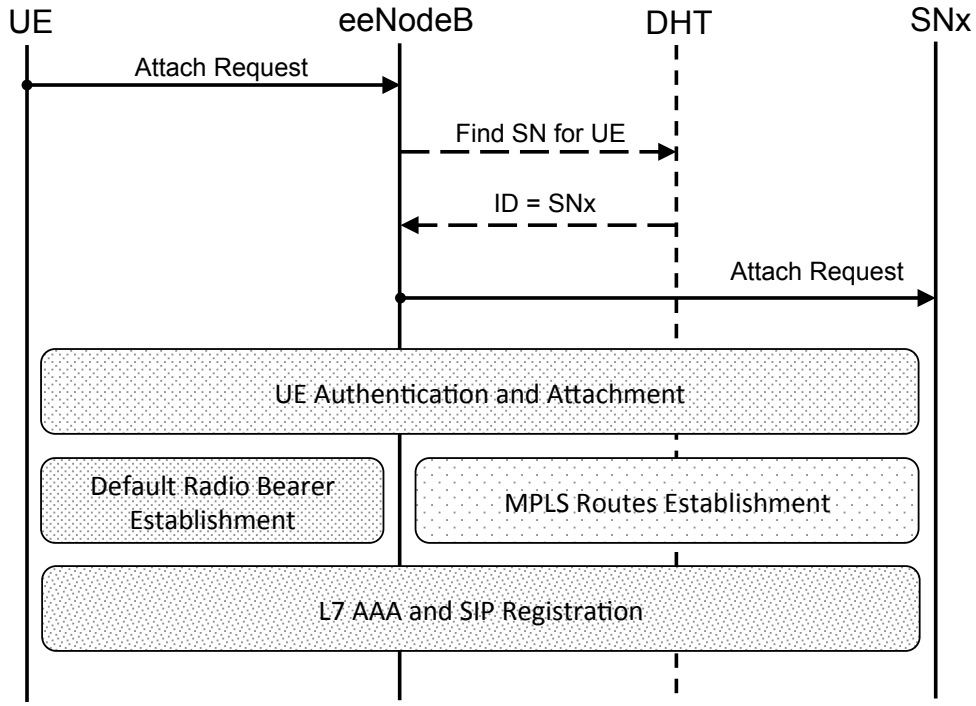



Figure 5.13: UE attach and SIP registration with eeNodeB.

### Scenario with eNodeBs

Fig. 5.14 shows a VoLTE call setup message flow when using a legacy eNodeB. Given that our core network is a P2P network, as mentioned earlier, for VoLTE calls we use P2PSIP. In particular, when UE1 wants to call UE2, UE1 sends an `INVITE` to its outbound SIP Proxy server that is, to SN1.[3] When SN1 receives the `INVITE` from UE1 it replies with a `100 TRYING` to let UE1 know that it has received the `INVITE` and to stop re-transmissions. At the same time, SN1 deploys a VCN for UE1 and transfers UE1's state, including security context and SIP state, to VCN1. An EPS bearer update is also triggered by SN1 in order for UE1 to be able to send and receive

[3]Outbobund SIP Proxy Servers are usually configured in the UE either statically or dynamically via DHCP or DNS, for example.

packets from VCN1. The EPS bearer update process is discussed in Section 5.6. Once VCN1 is up and running, it queries the P2PSIP overlay in order to find which VCN is responsible for UE2's Address of Record[4] (AoR). The P2PSIP overlay replies with the address of VCN2, if this is already up and running; otherwise, it replies with SN2's address. Here, we describe the most general case and assume that it replies with SN2's address. `AppAttach` messages are then exchanged, as per P2PSIP standard process, in order for the two nodes to be able to directly talk to each other: at this point, VCN1 forwards the `INVITE` to SN2. When SN2 receives the `INVITE`, it deploys a virtual core network (i.e., VCN2) for UE2 and starts the EPS bearer update process. At the same time, it replies with a `305 REDIRECT` informing VCN1 that it has to use VCN2 as inbound SIP Proxy for UE2. VCN1 then sends the `INVITE` to VCN2 which forwards it to UE2. From now on, all SIP messages are sent directly from UE1 to VCN1 to VCN2 to UE2, and viceversa (see Fig. 5.15). It is important to notice how P2PSIP is used only for discovering the callee's SN and VCN and how subsequent signaling and data messages do not leverage the P2P overlay.

A few other considerations are in order. When SN1 deploys VCN1, it transfers SIP state to VCN1. Moreover, as explained in Section 5.4, VCN1's SIP Proxy server has the same private IP address of SN1's (i.e., SIP_IP_IN). Furthermore, an EPS bearer update is performed. This means that, at the application layer, UE1 is completely unaware that this change from SN1 to VCN1 has occurred, as it will continue using the same SIP Proxy server's IP address it was using before. The same thing, however, is not true for SN2 and VCN2. In particular, when VCN1 sends the `INVITE` to SN2, it sends it to SN2's public IP address (i.e., SIP_IP_OUT). This means that when VCN2 is up and running, SN2 will have to send a `305 REDIRECT` in order for VCN1 to point to VCN2's public IP address (i.e., SIP_IP_OUT) instead of SN2's, given that these two are different. Therefore, in this case, transferring SIP state between SN2 and VCN2 is unnecessary, as the `305 REDIRECT` triggers a re-transmission of the `INVITE`. UE2, on the other hand, is still unaware, at the application layer, of the change between SN2 and VCN2 because it continues to see the same SIP Proxy server's IP address (i.e., SIP_IP_IN).

[4] An AoR is a unique SIP identifier in the form *user@domain*.

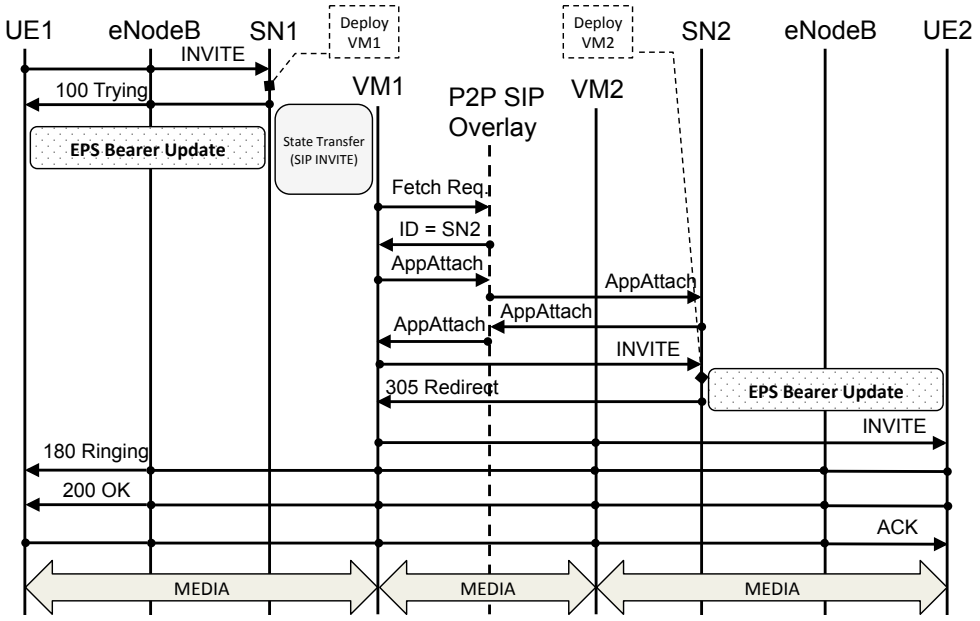



Figure 5.14: VoLTE call initiation with eNodeB.

#### Scenario with eeNodeBs

Fig. 5.16 shows a VoLTE call setup message flow when using an eeNodeB. In this case, almost everything remains the same as with the legacy eNodeB. The only difference is in the way the re-direction between SN and VCN is performed: this no longer involves the UE (see Section 5.6.2). At the lower layers, however, the use of an eeNodeB and the consequent absence of all bearers (except for the Radio bearer) significantly improve latency, as we do not have all the IP-in-IP tunnels typical of a bearer-based architecture.

### 5.5.3 Internet Access

As mentioned earlier, an SN handles basic tasks for multiple UEs. If, however, a UE starts being engaged in data intensive activities (e.g., audio and video streaming, pho-

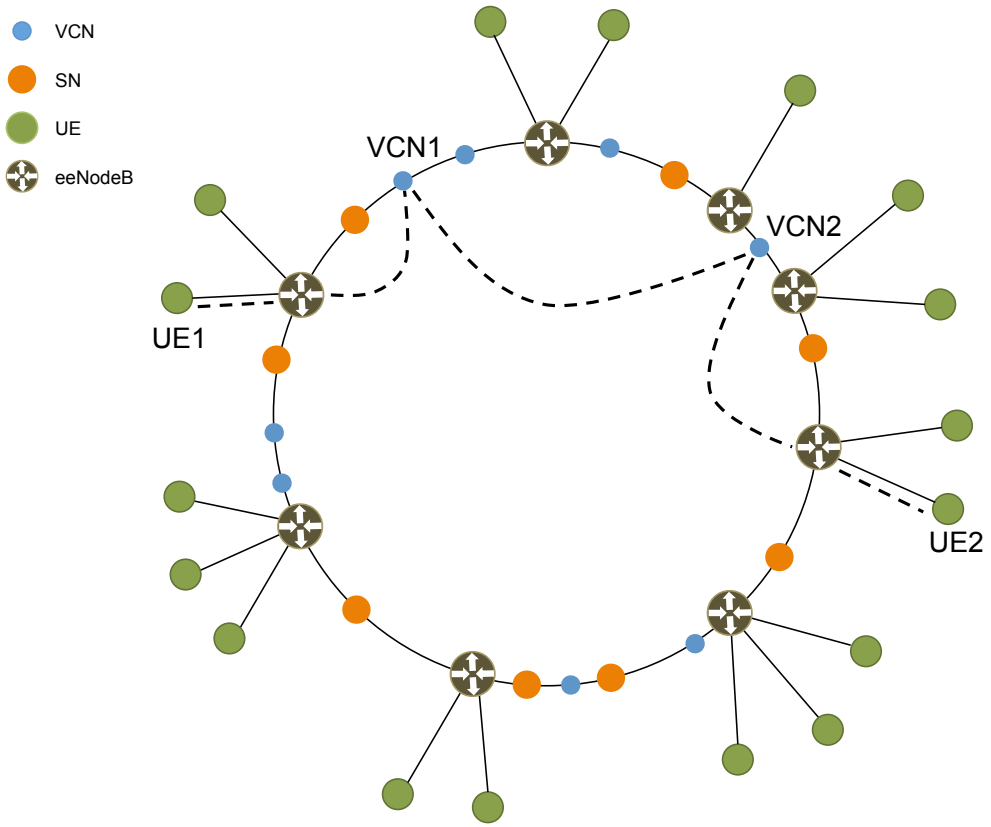



Figure 5.15: Data path after discovery phase.

to uploads), its SN deploys a VM running a dedicate VCN for that UE and redirects SIP signaling and routing (see Section 5.6), so that the UE communicates with the outside world through that VCN. In such a scenario, the VCN behaves as: a dedicated EPC, if a legacy eNodeB is used; a dedicated Internet gateway, if an eeNodeB is used. The usual standard procedures apply.

Once these data intensive tasks end, the VCN notifies the SN, which performs three main actions: it triggers another routing redirect in order to redirect all UE traffic to itself; it updates the P2PSIP DHT in order to point the UE's AoR to itself; it destroys the VM running the UE's VCN as it is no longer needed.

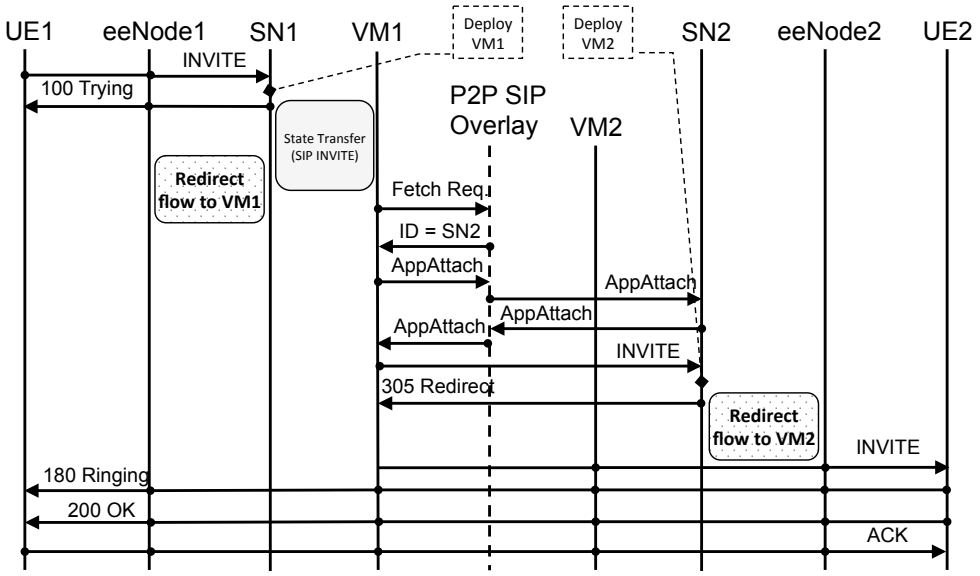



Figure 5.16: VoLTE call initiation with eeNodeB.

## 5.6 Re-direction

When a UE "moves" from the SN to a VCN (i.e., the UE goes form *idle* state into *active* state) and viceversa, its tunnels and IP routes need to be updated accordingly. In particular, we have two different scenarios: one when using a legacy eNodeB and another one when using an eeNodeB. Let us look at these two scenarios.

### 5.6.1 Scenario with eNodeB

Figs. 5.9 and 5.10 show the high-level components of an SN and VCN, respectively, when using an eNodeB. In particular, we show the IP addresses relevant for UE operations and mobility. As mentioned earlier, when a UE attaches to the network, its eNodeB will identify which MME to talk to. The MME will then tell the eNodeB which S-GW and P-GW to talk to. Tunnels are then established between UE and P-GW and between UE and IMS (i.e., SIP proxy or P-CSCF). Such tunnels are also

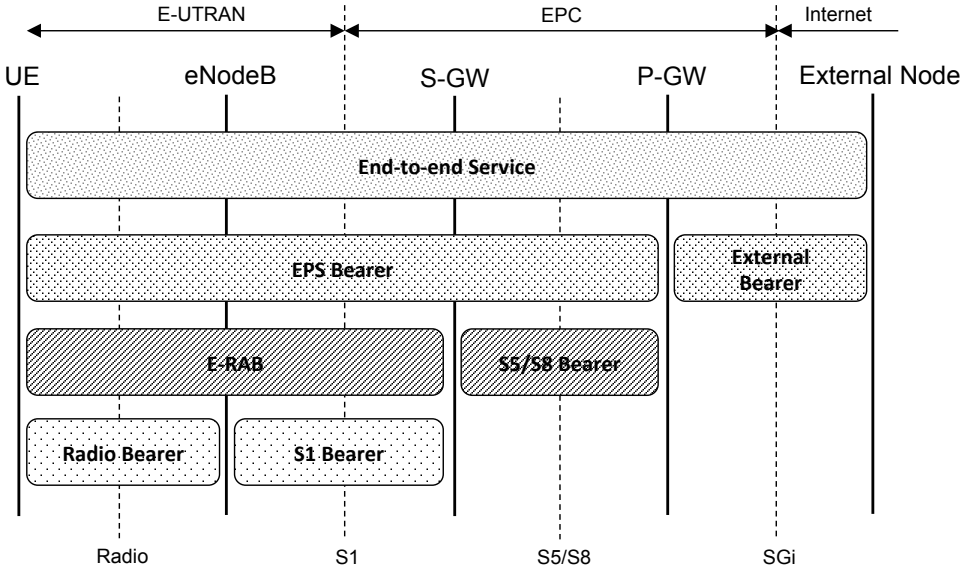



Figure 5.17: Bearers in an LTE network.

known as EPS bearer and IMS bearer, respectively. According to 3GPP terminology, a *bearer* indicates a tunnel with an associated QoS. As we can see in Fig. 5.17, there are different types of bearers. In particular: there are default bearers and dedicated bearers; the bearers can have short or long lifetimes; they can have either a guaranteed bit-rate or not. Each default bearer has an IP address associated with it and multiple dedicated bearers are usually bundled to one default bearer. Each dedicated bearer has its own independent QoS but shares the same IP address originally assigned to the UE when the default bearer they are bundled with was established.

When a UE successfully attaches to the network, a default EPS bearer is created between UE and P-GW. At either UE attachment or when a SIP registration occurs, a default IMS bearer is also created. These two bearers have two different IP addresses: one used for non-voice services (i.e., via the EPS bearer) and one used just for VoLTE service (i.e., via the IMS bearer). In particular, when a VoLTE call is initiated, SIP signaling uses the default IMS bearer while a dedicated bearer with higher QoS

is created and bundled with the IMS default bearer to transport voice-data packets. It is important to remember that these bearers are IP-in-IP tunnels leveraging the underlying MPLS network used in the core network. Therefore, MPLS flow tables are created accordingly in order to support such bearers and the corresponding routing.

When a UE is transferred from an SN to a VCN, one needs to update: EPS bearer, IMS bearer and MPLS routing. In the eNodeB case, the MPLS routing is automatically updated as a consequence of the bearer update – this, however, is not true for the eeNodeB case, as will be discussed later.

### EPS bearer update

When using a legacy eNodeB, we need to act within the constraints of the current 3GPP standards. In particular, in [73] 3GPP defines how to perform load re-balancing between different MMEs, that is, how to move a UE registered with an MME to a different MME belonging to the same MME pool area.

In order to implement load balancing, each MME is associated with a weight factor which is proportional to the probability of being selected by an eNodeB. The higher the weight factor, the larger the number of UEs that will associate with that MME. A weight factor equal to zero, on the other hand, indicates that an MME cannot handle any UE traffic. When an MME must be removed from an MME pool area (because of maintenance, for example), it must off-load the UEs connected to it to some other MME in the same MME pool area. This is achieved by having the MME initiate an *S1 Release* procedure indicating, as release cause, "load balancing with TAU required." This triggers a Tracking Area Update (TAU) procedure on the UE in a way such that the eNodeB selects the new MME according to the weight factors of all the MMEs in the pool. By connecting to the new MME, a new EPS bearer is established between the UE and the new P-GW indicated by the new MME.

In order to apply such a mechanism to our scenario, an SN and its VCNs need to have their MMEs belonging to the same MME pool area. When an SN needs to "redirect" a UE to a VCN, it triggers a load re-balancing and the VCN sets the weight factor for its MME to a high value, while all other VCNs keep a low weight factor for their MMEs. Once the UE connects to the VCN, the latter changes its MME's

weight factor to a low value. During a UE redirection there should be only one VCN's MME with a high weight factor in the SN's MME pool area. To further improve the efficiency of such a mechanism, an SN may create a different MME pool area with each VCN so that, at any given time, in an MME pool area there are only one VCN's MME and the SN's MME. In such a scenario, the SN's MME would have to be part of multiple MME pool areas. The same mechanism would work for re-directing the UE from the VCN's MME back to the SN's MME.

#### IMS bearer update

The IMS network is separate from the EPC and has its own Access Point Name (APN). In particular, the IMS default bearer is established either when the UE attaches to the network or when a SIP registration occurs. Because of this, when the EPS bearer is updated, the new attachment triggers an IMS bearer update as well.

At the application layer, however, information about the SIP Proxy's IP address needs to be updated in order for the SIP client in the UE to point to the SIP Proxy server in the VCN instead of pointing to the one in the SN. This, however, turns out to be a relatively simple task if the exact same IP address is assigned to both the SIP Proxy server in the SN and the SIP Proxy server in the VCN. In particular, this is possible because the concept of a bearer resembles the one of a dedicated circuit in circuit-switched networks. In other words, since we updated the bearer to point to the VCN, the same IP address will now point to the SIP Proxy server in the VCN, if so configured, without any conflict with the SIP Proxy server in the SN. In fact, the SN is no longer reachable by the UE.

### 5.6.2 Scenario with eeNodeB

Fig. 5.18 shows the high-level architecture of a CNN when using an eeNodeB. Similarly to the previous scenario, the SIP proxy server behaves as a Registrar and Redirect server for the SN, while it behaves as a B2BUA for a VCN. The EPC, on the other hand, has now collapsed into a simple Internet gateway.

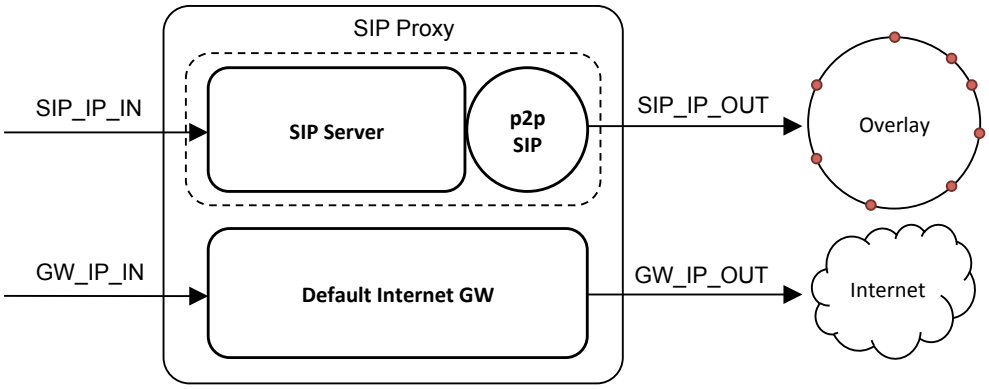



Figure 5.18: CNN high-level architecture (SIP Proxy and Internet gateway are multi-homed).

As mentioned in Section 5.3.3, we do not modify the *Uu* interface between UE and eeNodeB. This means that when a redirection takes place it will not involve the UE. The UE will be unaware of any change taking place while eeNodeB and SN will make sure traffic is redirected accordingly. In particular, as shown in Figs. 5.8a and 5.8b and as described in Section 5.3.3, the eeNodeB is an MPLS router on the user plane and supports P2P protocols on the control plane in order to be able to talk to the P2P overlay of the core network (e.g., RELOAD). When a redirect for a UE needs to take place, the SN behaves as an SDN controller for that UE and it updates the MPLS flow tables accordingly so to route the UE traffic to its VCN. This can be accomplished by leveraging existing protocols such as OpenFlow and RSVP-TE. At the application layer, the SN updates the information in the P2PSIP DHT in order to point the UE's AoR to the newly created VCN. In this case, the UE is completely unaware of this change taking place. As before, the UE's IP address, the SIP Proxy server's IP address and the Internet gateway's IP address do not change. The underlying MPLS network takes care of re-routing the traffic from the SN to the VCN.

## 5.7 Mobility

Mobility management in today's cellular networks is a major component of the cellular infrastructure. In particular, the access network is split in Tracking Areas (TAs) and the MME keeps track of which UEs are in which TAs. Whenever a UE changes TA, it notifies the network which records the new TA for that UE. If a UE needs to be reached by the network (e.g., incoming call), the network pages the UE in the last known TA for that UE. This, together with handover management, makes mobility possible in the cellular network.

One downside of this architecture is that as the UE moves around, the signaling has to always reach the EPC infrastructure in the datacenter the UE is assigned to. This means that when a user from New York goes to LA and makes a call, the signaling has to go all the way to the datacenter in New York and back to LA. This clearly introduces high delays and long Round Trip Time (RTT).

In our new architecture, we propose a significantly different approach to mobility, borrowing concepts from Content Distribution Networks [74]. In particular, we try to deploy a VCN as close to the UE as possible in order to minimize latency and RTT. We distinguish between pre-call mobility and mid-call mobility.

In pre-call mobility, the UE is in idle mode and notifies its SN of changes in TA. When this happens, the SN searches and selects the available hosting infrastructure closest to the UE's new location: if needed, the SN will deploy there a VCN. When the UE goes into active mode, the SN carries out this deployment.

In mid-call mobility, the UE is in active mode (e.g., in the middle of a call) and, as the user moves, the UE changes TA. Because the UE is in active mode, it has already been re-directed to a VCN and, as such, notifies the VCN of the change in TA. On receiving this TA update (TAU), the VCN notifies its SN by forwarding the TAU to it. When the SN receives the TAU from the VCN, it searches and identifies a new hosting infrastructure as close as possible to the UE's new location. Once this infrastructure has been identified, the SN starts a live migration [75] of the VM hosting the UE's VCN to a VM hosted in the new infrastructure — all standard live-migration techniques can be applied [76, 77, 78]. At the MPLS layer, flow tables

must be updated accordingly in order to enable routing to the VCN's new location. As mentioned in Section 5.6.2, this can be enabled by the SN acting as an SDN controller and leveraging protocols such as OpenFlow and RSVP-TE.

The hosting infrastructure closest to the UE's new location (i.e., its new TA) can be discovered by relying on any of the existing protocols used for location-based services, such as Location-to-service Translation (LoST) [79, 80] and DNS [81]. The identified hosting infrastructure can be either: a datacenter of the cellular operator; a specific eNodeB, provided that this has VM hosting capabilities (e.g., fog computing); or anything in between. In particular, for high-profile users (e.g., Government), VCNs can be hosted on private cloud infrastructures not operated by the network operator, hardened with their own security measures in order to achieve a higher level of privacy and security for the UE. For such private infrastructures, the network operator would only retain a control interface between SN and VCN in order to initiate and terminate VCN deployment. In particular, as described in Section 5.5.2, media is exchanged directly between VCNs and UEs with no SN involvement, thus preserving privacy and confidentiality.

## 5.8 Handover

In LTE, 3GPP has defined different kinds of handover [82, 83].

In the *intra-LTE handover* MME and S-GW do not change and the handover can be either an X2 handover or an S1 handover. In the X2 handover, the source eNodeB sends to the target eNodeB, via a *handover_request* on the X2 interface, security context and information on the MME assigned to the UE performing the handover. In the S1 handover, the MME is directly involved in the handover process due to unavailability of the X2 interface or failure of the X2 handover. In both cases, the handover procedure remains unchanged also for the new architecture, when using a legacy eNodeB. In particular, when moving from the old eNodeB to the new eNodeB, the S1 bearer will have to be updated (see Fig. 5.17).

In the *inter-MME handover*, the UE performing the handover changes both eNodeB and MME. In the *inter-MME/S-GW handover*, the UE performing the handover

changes eNodeB, MME and S-GW. Both these types of handover do not happen in the new architecture. This is due to the fact that a UE is always anchored to either its SN's MME or its VCN's MME (i.e., when using a legacy eNodeB). In particular, when using the VCN's MME, if the UE changes TA, the system migrates the VCN in the new TA as close to the UE as possible (see Section 5.7): hence, an inter-MME handover is not needed and only the intra-LTE handover is required. In other words, instead of the UE having to change MME due to mobility, in the new architecture the MME follows the UE as this moves around.

In the case of an eeNodeB, everything mentioned earlier still applies with the exception that there are no bearers to update. Only MPLS routes need to be updated in the core network in order to enable routing between SN or VCN and the target eeNodeB. In particular, similarly to the *Uu* interface, also the *X2* interface is left unaltered in an eeNodeB.

## 5.9 Scalability

Currently, network operators often manage over 100 million connected devices: therefore, *statically* assigning a dedicated core network to each UE can be very challenging in terms of scalability. However, in our architecture a VCN is assigned to a UE only for a limited amount of time: once the UE is back into *idle* mode for a sufficiently long time, it connects back to its SN and its VCN is destroyed. This approach drastically improves scalability as is shown in Section 5.12.

We have four options for creating a VCN instance per UE: processes, containers, clear containers [84], and VMs. Picking one over the other introduces a trade-off between scalability and isolation. *Processes* can scale very well but have very little isolation, if any. This means that, from a security perspective, by running our VCNs as simple processes we implicitly agree on trusting the OS which they run on. This may be a very solid choice depending on the specificities of the network in which VCNs get deployed and the corresponding threat model. *Containers* are much easier to deploy and manage. Moreover, they have better isolation than processes, although they still rely on trusting the OS which they run on. Furthermore, a container with

root privileges may represent a security threat to all other containers running on the same virtual or physical machine. The concept of *clear containers* is relatively new. The basic idea behind clear containers is to maintain the flexibility and scalability of containers while guaranteeing the strong isolation typical of a VM by using various optimizations at the kernel level and CPU-specific virtualization support. Initial results have shown that a clear container can be launched in under 150 ms and has a memory footprint of about 20 MB. This means that 3500 clear containers can be run on a single machine with 128 GB of RAM.[5] While this is far from being "large scale," it looks very promising. Clear containers are gaining much traction as they appear to combine the best of both worlds (i.e., VMs and "classical" containers). Finally, *VMs* achieve the best isolation at the price of reduced scalability.

For a smaller user population, VMs should be used whenever possible. For a larger user population, however, depending on the threat model of the specific network, one of the other three options should be considered and, perhaps, multiple options should be combined together (e.g., containers running in VMs).

Aside from scalability, set up delay is also critical. In the case of VMs, for example, in order to deploy and run VCNs "on demand" and in real-time, the deployment of VMs has to be as fast as possible. To enable this, we use a technique similar to the thread-pool pattern used in multi-threaded programming. In particular, each SN maintains a pool of idle VMs already loaded with an OS image and VCN software. In doing so, when a new VCN needs to be deployed in the P2P network, the SN has only to load UE state and data (e.g., HSS data, billing) into one of the existing VMs in the pool, making the whole process much faster.

In Section 5.12, we will present some measurements and provide some insights on scalability and related issues in real-world cellular networks.

## 5.10 VCNs and Core Network Security

The mobility core network as we know it, that is, a centralized architecture with or without virtualization, is *reactive*. In particular, it tries to react to a problem once it

[5] Source: https://clearlinux.org/features/clear-containers

perceives one. For example, with reference to the LTE architecture shown in Fig. 5.1, when the network detects high traffic to the HLR, it reacts by spawning additional VMs running each an instance of the HLR. In other words, the core network, upon detection of a problem, reacts to it, i.e., it does not prevent the problem from happening in the first place.

The proposed VCN architecture is a *proactive* architecture by design. In particular, by isolating each VCN of each user, certain types of attacks are prevented from happening in the first place. For example, as we show on the left side in Fig. 5.19, a cellular botnet trying to perform a DDoS attack towards the mobility core network would not be able to do so because each UE in the botnet would point to its own copy of the core network. In other words, all the UEs in the botnet would not be able to focus their attack on the same target. On the other hand, a DDoS attack against the cellular core network initiated from the public Internet may successfully compromise one VCN (see Fig. 5.19 on the right side), but this would affect only a single user and the recovery from the attack would simply be to assign the user to a newly spawned VCN. The VCN under attack would then be destroyed or perhaps isolated in order to study the attack and identify the attacker. In any event, all other users would remain unaffected by the attack. Similarly, a piece of malware crafting malformed packets with the intent of "crashing" a core network element would only be able to crash an element of its VCN, leaving all the VCNs of all the other UEs unaffected.

A separate discussion is needed for low bitrate attacks against SNs. For example, a cellular botnet may try to attack an SN by performing a low bitrate attack (i.e., without triggering a re-direct to a VCN) via SMS messages. This type of attacks could affect all the UEs associated with the SN under attack. In such a case, however, the botnet would need to make sure that a sufficient number of its UEs point to the same SN — this is not an easy task, as SNs can be assigned to UEs more or less randomly. Furthermore, should an attack against an SN be successful, new SNs could be created and the UEs of the compromised SN could be split among new and old SNs, thus making the attack no longer effective. The compromised SN would be destroyed and the DHT updated accordingly.

Another way a botnet may try to attack SNs is by having its UEs continuously

switch between *idle* and *active* modes. In doing so, an attacker may try to achieve two possible goals: (i) overload the core network by triggering unnecessary orchestration and signaling traffic; (ii) degrade core network performance by drastically increasing VCNs's lifetime, thus increasing the total number of active VCNs in the network. Given that a VCN is destroyed only if the UE has switched back to idle mode and stayed in that mode for a certain amount of time (see Section 5.12), the extra signaling traffic could be triggered by the attacker at a rate too low to make the attack effective. On the other hand, drastically increasing the number of concurrently active VCNs may lead to scalability issues and, consequently, to performance degradation. This attack, however, would start to be effective with a huge botnet in the size of a few million bots, i.e., a few million compromised UEs. Such a scenario is extremely unlikely to happen and, in any case, countermeasures can be put in place.

By assigning a VCN per UE, security can now be highly customized to the needs of single users. For example, some users may not require encrypted tunnels and their VCNs could be stripped of such functionality or, perhaps, stronger authentication protocols may be deployed for other users. Firewall rules can now be customized to single UEs so that, for example, if a UE is not supposed to exchange web traffic (e.g., M2M), ports 80 and 8080 can be blocked. Similarly, as mentioned in Section 5.7, for environments where a high security level is required (e.g., Government, enterprise), one or more VCNs could be deployed in a private cloud infrastructure thus allowing to use local security policies, special infrastructure and stronger encryption. On the other hand, for the average user, either a public cloud infrastructure (e.g., Amazon AWS), a cellular operator hosting infrastructure or eNodeBs with hosting capabilities (e.g., edge cloud, fog computing) could also be used to host VCNs. Furthermore, in the case of multi-SIM UEs, one user may be associated, at the same time, with multiple core networks hosted in different clouds. If an external hosting infrastructure (i.e., either public or private) is used to host some VCNs (see Fig. 5.20), the network operator would only retain control on setting up and tearing down those VCNs plus some additional signaling required for billing and other management tasks. This, together with specific service level agreements with cloud operators, would enable the cellular operator to provide a new class of custom security services while, at the

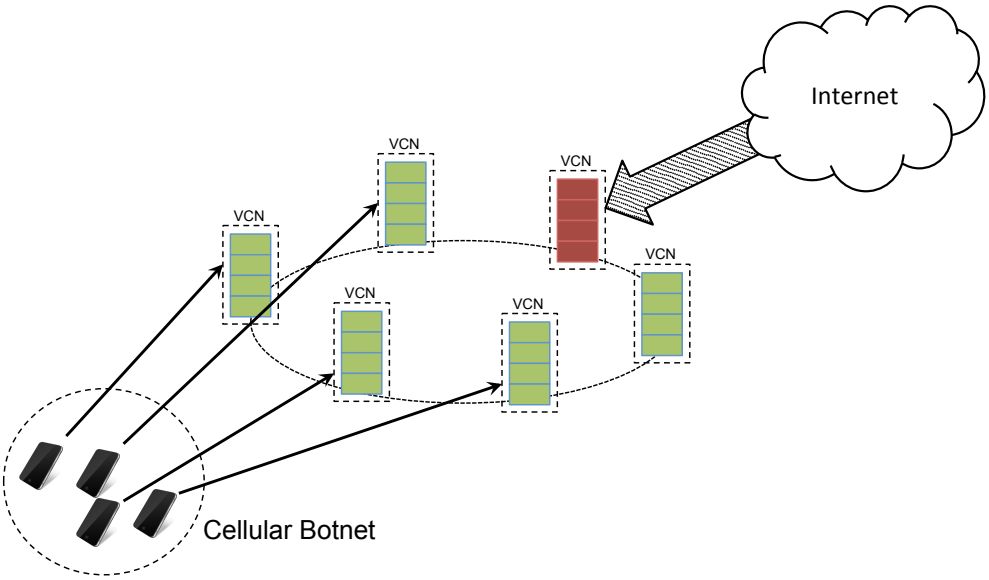



Figure 5.19: Attacks on the new architecture.

same time, reducing operational cost.

## 5.11 Other Advantages

Let us summarize some of the non-security-related advantages the new architecture brings.

By assigning a VCN to each UE, we can now assign network resources at the *user* level. In particular, not all users need all the capabilities a cellular network has to offer. For example, some users are static or nomadic while others are highly mobile. This means that a UE may have no need for high mobility management and, as such, this functionality may be stripped out of its VCN. Similarly, a user may have no need for paging as, perhaps, it is not supposed to receive notifications or calls (e.g., possible in some machine-to-machine scenarios). In such a case, the paging functionality could also be stripped out of the UE's VCN, further reducing its complexity. Another

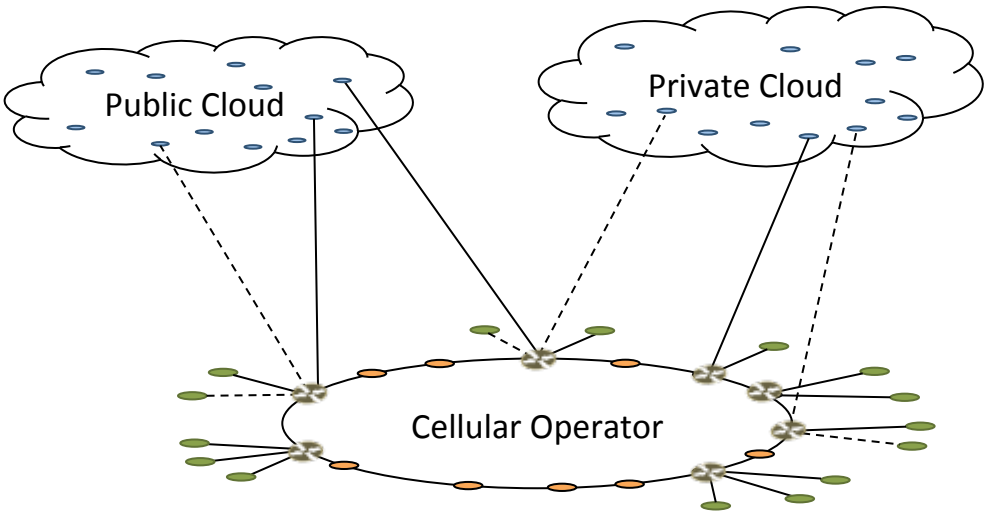



Figure 5.20: New hosting architecture.

UE, still, may not need VoLTE services but just Internet access: this would allow to remove all IMS-related functionalities from the UE's VCN. In other words, the proposed architecture allows for customization of the cellular core network at the UE level, giving a much more personalized experience and services. This is beneficial for both the user and the network operator as it allows for new business models and, at the same time, simplifies operations. Also, it reduces VCN complexity, simultaneously reducing the VCN's attack surface.

As mentioned earlier, in today's cellular networks, when a user travels from New York to Los Angeles, for example, whenever he/she makes a call, the signaling has to go all the way to New York and back to Los Angeles, introducing greater latency and delays. To make things worst, as discussed in Section 5.6, each UE establishes multiple IP-in-IP tunnels with the core network, further adding to RTT and delays. By using the proposed architecture, these two sources of delay are no longer present. In particular, as discussed in Section 5.6, by removing all bearers in the core network and simplifying the core network to one single MPLS network, there are no more multiple fine-grained tunnels to be established by the UE. Also, as discussed in

Section 5.7, by having a VCN per UE, we can "move" the VCN as close to a UE as possible, further reducing latency and delay due to physical distance.

The current core network architecture, virtualized or not, has a bottleneck at the datacenter where all traffic converges with consequent load peaks and, potentially, service degradation. On the other hand, as we show in Fig. 5.20, our core network architecture is distributed and very dynamic, as VCNs can be deployed in different hosting infrastructures and datacenters and can be moved as needed. This minimizes load peaks and avoids bottlenecks turning VCNs placement into an optimization problem with many parameters to consider (e.g., distance from UE, current load at hosting infrastructure, security requirements).

Finally, a per-UE customization of the core network also helps in Operations and Management thanks to easier access to single-user traffic, easier billing, easier policy enforcement and more.

## 5.12 Measurements

**Goal:** We define an "entity" as either a VM, a clear container, a container or a process (see Section 5.9). Also, let us recall that a dedicated VCN is allocated to each UE where a VCN is set up within an entity. In this section, we address two critical design questions: given the user population, what is the total number of entities required to accommodate the traffic volume? What is the life cycle of an entity from its creation to its destruction?

**Dataset:** The analysis and evaluation is based on a sample of fully anonymized voice call records collected within a large U.S. cellular network provider. This is a subset of Call Detail Records (CDRs) which are used for billing. The anonymized voice call data contains a random sample of 200,000 users within the whole population, over a period of nine days from February 10, 2016 to February 18, 2016. There are approximately twenty five million records observed in this period of time for the random subset of users. All phone numbers are anonymized with randomized unique values in order to protect privacy. Each voice call record is simply a time-stamped tu-

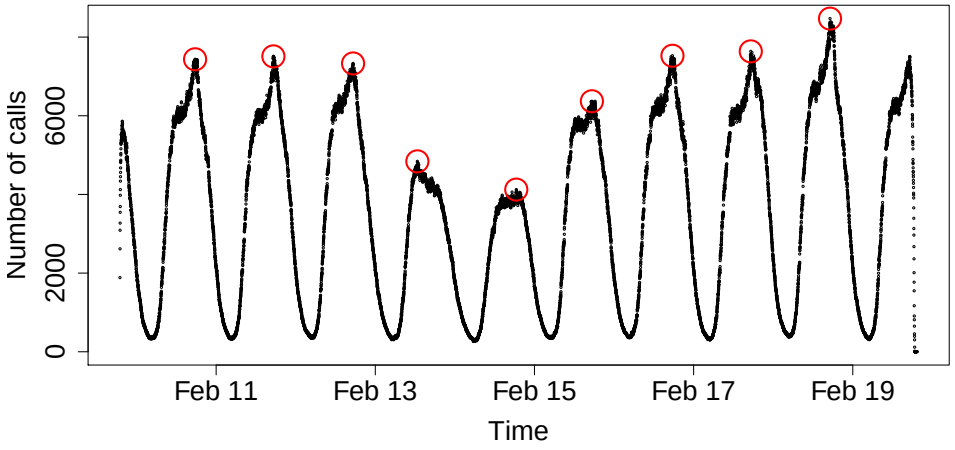



Figure 5.21: The number of concurrent active calls over nine days. The red circles mark the daily maxima.

ple, indicating: the start time of a voice call; the caller and callee anonymized phone numbers; as well as the duration of the call.

**Analysis 1:** First, let us look at the total number of entities required to serve the traffic load in the collected user sample. We define *concurrent calls* as ongoing calls that fall into the same minute bin (i.e., active or placed within the same minute). Fig. 5.21 shows the number of concurrent active voice calls over nine days. As we can see, there is a clear diurnal cycle, where: the daily minimum is achieved around 2-3AM; and an increased call activity is observed during working hours and early at night (9AM to 9PM). These observations are consistent with the previous study by Becker et. al [85]. The artifacts at the beginning and at the end of the figure are due to our data sample not including full-day data before February 10th, 2016 and after February 18th, 2016. We are interested in knowing the maximum number of concurrent calls that could occur in the system, as this tells us the maximum number of entities that have to be simultaneously active. Weekends (February 13th and 14th) have smaller traffic maxima than weekdays. Therefore, we only focus on analyzing data from the seven weekdays.

We use a simple time-series model, seasonal AutoRegressive Integrated Moving

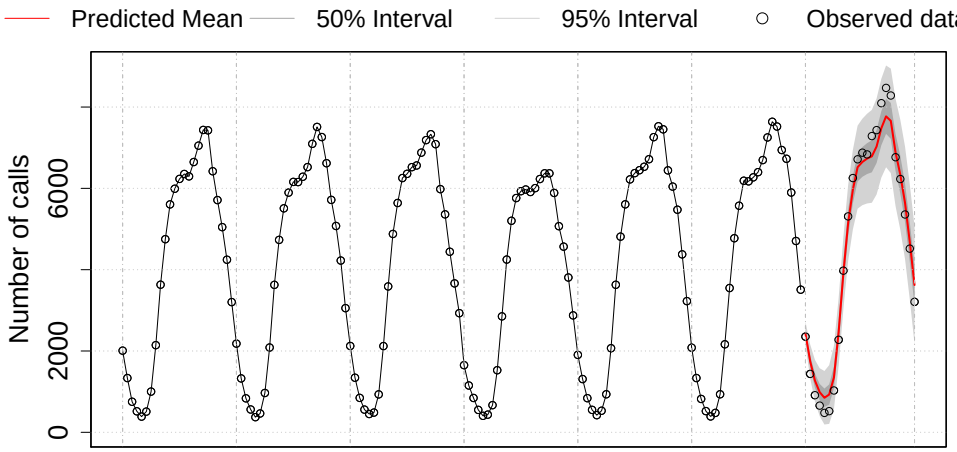



Figure 5.22: Forecast of one-day traffic from a seasonal ARIMA model fitted on concurrent voice-calls data over 6 weekdays.

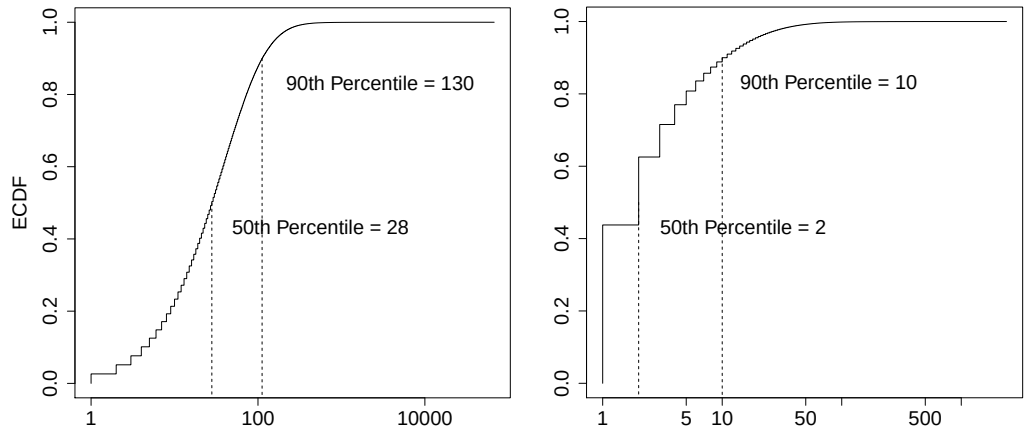
Average (ARIMA), to predict concurrent traffic load in the network. The model treats all weekdays equally. Seven days of data are divided into 6 days of training and one day of test. Since we are interested in the maximum traffic load in the network, for each hour during the training, we keep the maximal count among the original non-aggregated data points. We use the R package `forecast` [86] to fit an ARIMA model to our hourly count time-series. Fig. 5.22 is the forecast of one-day traffic according to the best ARIMA model learned from the training: `ARIMA` $(1,1,0) \times (0,1,1)_{24}$. The prediction of the next one-day per-hour call load is shown at the end of the graph as a solid red curve, with prediction intervals of 50% and 95% represented by the two shaded areas (smaller and larger, respectively). The true test data for one day lies within the 95% prediction interval with a maximum of 8,400 concurrent calls, thus showing an accurate prediction. The highest count of the 95% interval represents 9,126 concurrent calls, which correspond to $\sim 4.6\%$ of the total number of users.

**Recommendation 1:** If both caller and callee use the proposed system, then there will be two VCNs per call: therefore, the maximum number of required VCNs is twice the maximum number of concurrent calls. On the basis of the previous model and

assuming VCNs get destroyed as soon as UEs move back to *idle* mode, the maximum number of simultaneously active entities (i.e., one entity per VCN) would be around 9.2% of the user population. If the network is utilized by a small or medium size population of users, such as in the hundreds of thousands, it is reasonable to use VMs to host VCNs. However, if the population of users goes up to hundreds of millions, containers and processes are better options, although this corresponds to trading off higher scalability for a lower degree of VCN isolation.

**Analysis 2:** We study users' traffic patterns in order to determine the life cycle of an entity between its creation and its destruction. Given our sample of call records, we found some calls to be extremely short (shorter than ten seconds). One common pattern for these short calls is that two records of a phone number *A* show the same start time and duration but with different phone numbers, that is, one towards phone number *B* and one towards phone number *C*. These cases correspond to calls for which the callee (i.e., phone number *A*) does not answer the caller (i.e., phone number *B*). Instead, the calls are redirected to voicemail (i.e., phone number *C*), a service that stores voice messages. In such a case, setting up a VCN is inefficient as no real call happens. This suggests a change in our protocol for VoLTE calls (see Section 5.5). In particular, instead of deploying a VCN when the SN receives an `INVITE`, the VCN gets deployed once the SN receives the `200 OK` to the `INVITE`, that is, once the callee has accepted the call. For the rest of the analysis, we remove calls that last less than ten seconds.

We measure the duration of a call in minutes. Figs. 5.23a and 5.23b show the Empirical Cumulative Distribution Function (ECDF) of the number of calls and call duration over a ten-day period, respectively. Notice that both the number of calls and the call duration are discrete variables; this makes the ECDF look like a step function. The median number of calls per user over ten days is 28, with approximately 3 calls per user per day. About 90% of the users have less than 130 calls over ten days (i.e., approximately13 calls/day) which is consistent with the results in [87]. The median of the call duration is two minutes and approximately 90% of the calls are less than ten minutes long. This means that the majority of the calls has a short duration, that is, the lifetime of an entity could be short if based only on the call duration. However,




(a) # of calls (b) Call duration in minutes

Figure 5.23: ECDF of the # of calls and call duration over 10 days.

we need also to consider the time between calls since we do not want to destroy an entity just to find out that we have to deploy it again a few seconds later.

A *call interval* is defined as the lag between the time a call terminates and the time the next call is placed, for a given user. As before, we measure the call interval in minutes. Fig. 5.24 shows the box-and-whisker plot of the ratio of calls with different call interval levels per user. The levels are indicated by the labels on the left side of the Y-axis. For example, a level of (2, 5] indicates a call interval that goes from above 2 to 5 minutes. The levels are evenly partitioned with a step size of five minutes, excluding the last level which represents a call interval longer than 30 minutes. The first 5-minute level is further broken into two levels to show more details on very short call intervals (i.e., less than two minutes). The labels on the second Y-axis on the right indicate the percentage of outliers for each level (i.e., shown as black dots in the plot).

If the time interval between two calls is short, or if the majority of the calls are placed within a certain time interval, it may be inefficient to destroy the entity around that interval. For example, if a user places a call every three minutes, it is inefficient to destroy his VCN at the 2 minute mark as it will have to be deployed again 1 minute

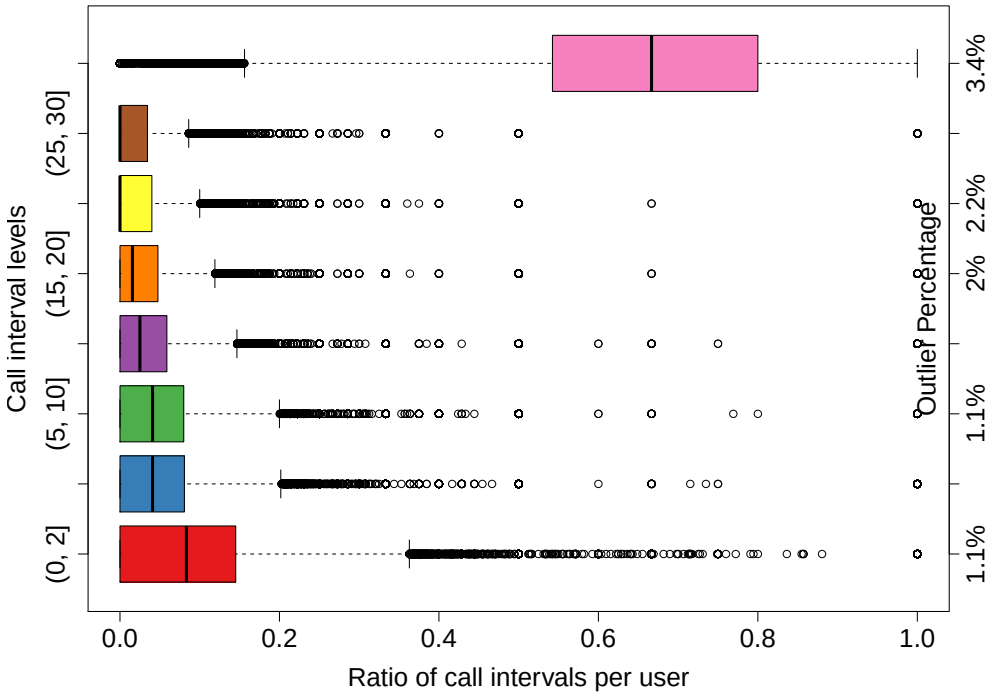



Figure 5.24: The ratio of calls with intervals at different levels.

later.

**Recommendation 2:** In Fig. 5.24, a small ratio value on the X-axis for a given call interval tells us that the percentage of calls per user that have a time between calls as indicated by the interval on the Y-axis is small and, thus, unlikely. From our collected data sample, the median for interval not longer than 2 minutes is 8%, meaning that half of the users have 8% of the call intervals shorter than or equal to two minutes. The median and third quartile are 4% and 8%, respectively, for the next two call interval levels, i.e., (2, 5] and (5, 10]. The median for the $(10, 15]$ call interval level is 2% and the third quartile is 6%. It is encouraging to see that long call intervals (i.e., more-than-30-minute bin) are the largest percentage of all call intervals. In particular, the median is 67%, meaning that half of the users have 67% of the call intervals longer than 30 minutes, with the first and third quartiles being

54% and 80%, respectively. In other words, this means that if a VCN is idle for more than 10 minutes we can destroy it and, with high probability, there will be no need to deploy it again for another 20 minutes or so. To conclude, it is reasonable to measure and use the user call interval patterns in order to determine the life cycle of an entity, i.e., its restart rate.

**Summary:** By combining both recommendations, we show how the maximum number of concurrent entities that the network must support is affected by both the number of concurrent calls and the amount of time the UE has to spend into *idle* mode before its entity gets destroyed. By considering both conditions, we compare three cases: (i) the UE's entity gets destroyed right after the UE goes back to idle mode; (ii) the UE's entity gets destroyed 5 minutes after the UE goes back to idle mode; (iii) the UE's entity gets destroyed 10 minutes after the UE goes back to idle mode. Fig. 5.25 shows the average, over 9 days, of the ratio between the maximum number of active entities reached in cases (ii) and (iii) over the one reached in case (i). From the results in Fig. 5.25, one can observe that: in case (ii), the maximum number of active entities in the network can be twice the number reached in case (i); in case (iii), the maximum number of active entities in the network can be three times the number reached in case (i). This means that, in case (ii), the maximum number of entities is roughly 20% of the user population, while it grows to about 30% in case (iii). Picking a threshold value sets a tradeoff between entity restart rate and maximum number of active entities. A waiting time of 10 minutes would have a lower entity restart rate, but a quite large maximum number of active entities. A waiting time of 5 minutes, on the other hand, would still take care of most of the very short calls (as shown in Fig. 5.24), while keeping the maximum number of concurrent entities active in the network anywhere between 9% and 18% of the user population.

When to spawn and when to destroy VCNs, hence changing their number, is an optimization problem that, due to scalability, is critical to our new core network architecture. Our current analysis started focusing on voice-call traffic in order to estimate VCN scalability. In the future, we plan to analyze also IP data traffic in order to estimate VCN scalability for non-voice data traffic and, in doing so, engineer a system optimized for all types of traffic.

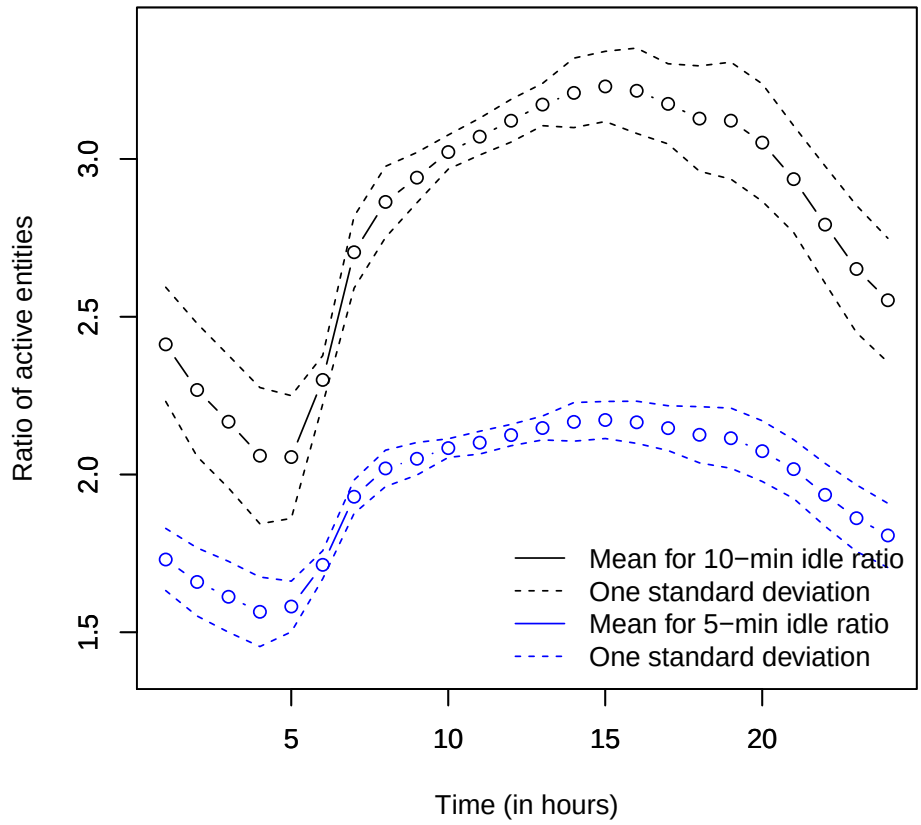



Figure 5.25: The ratio of active entities when destroying them after 10 and 5 minutes, over the baseline of destroying them right away, once UEs go back to idle mode.

# Appendix A

# An Encryption-aware Routing Protocol: Node Energy Computations

## A.1 Introduction

We present node energy allocation and node energy states and transitions between states for both non-stub nodes and stub nodes. Finally, we discuss how to convert $\alpha_i^R$ values to $R_i$ values.

## A.2 Fair Allocation

Table A.1 shows the parameters used for a fair allocation of energy between packet transmission and packet encryption. We can compute the ratio of energy transmission to energy spent for 1 round of encryption *per bit* as:

$$F = \frac{E_{TX}}{E_{ENR-1}}.$$

| Parameter | Description |
|---|---|
| $E_{TX}$ | 1-bit transmission energy |
| $E_{ENR-1}$ | Per-bit encryption energy for 1 round of encryption |
| $F$ | Encryption energy overhead |
| $\alpha_i$ | Total energy budget of node $i$ in Joules |

Table A.1: Parameters used in fair allocation.

If we have 1 J of energy, and we allocate it "fairly" between transmission and encryption, we have that

$$E_{TX} + E_{ENR-1} = 1.$$

Therefore, if we had only 1 J of energy, we would allocate:

$$E_{ENR-1} = \frac{1}{(1+F)},$$

$$E_{TX} = \frac{F}{(1+F)},$$

and the fair allocation would be:

$$\alpha^{R} = \frac{\alpha}{(1+F)},$$

$$\alpha^{TX} = \frac{\alpha * F}{(1+F)},$$

that is, for node $i$, fair allocation would be given by the following:

$$\alpha^{FR_i} = \frac{\alpha_i}{(1+F)}, \tag{A.1}$$

$$\alpha^{FTX_i} = \frac{\alpha_i * F}{(1+F)}, \tag{A.2}$$

where $\alpha^{FR_i}$ is the energy allocated for encryption at node $i$ and $\alpha^{FTX_i}$ is the energy allocated for transmission at node $i$.

During protocol operations, and in the interest of extending the network lifetime, nodes may decide to deviate from a fair allocation of energy. In this context, network

| Parameter | Description |
|---|---|
| $z$ | Number of paths a node has in its cache (# of routes to the sink) |
| $\Delta\alpha/\Delta T$ | Rate of change of energy measured in some interval T |
| $G$ | Set of 1-hop neighbors ($\|G\|$ = number of 1-hop neighbors) |
| $d_p$ | Distance of node from sink on path p |
| $d$ | Average distance of node from sink across z paths |
| $R^*$ | Best value of R among all paths known by the node (without adding the node own $R_i$) |
| $d_{min}$ | the minimum distance, in number of hops, across all paths to the sink from node $i$ |
| $h_1, h_2$ | Protocol constants greater than 1, with $h_1 > h_2$ |
| $\varepsilon_1$ | Threshold (on transmission energy) below which we want to prioritize transmission |
| $\varepsilon_2$ | Threshold (on transmission energy) above which we want to prioritize encryption |

Table A.2: Parameters used when deviating from fair allocation.

lifetime is defined as the contiguous period of time during which more than X% of the nodes in the network can route packets to the sink after completing all required rounds of encryption.

In the following, for ease of notation, we drop the subscript $i$ although we are still referring to computations at a single node. The additional parameters used to decide when and how to deviate from fair allocation are shown in Table A.2.

## A.3 Energy States

In the following we discuss the different energy states a node may be in and the reason behind the fudge factor $K_1$.

When deviating from a fair allocation of energy, we take away from one type of energy and give to the other. Somehow, we need to bind the amount of energy swapped in some reasonable way. From the routing protocol point of view, we have a rule that says that BMEP is subject to a fudge factor $K_1$, whereby a newly heard path NP will replace the existing BMEP only if $ME_{NP} > K_1 * ME_{BMEP}$. This implicitly says that we don't care about a new path if this has a minimum-energy node with transmission energy that is not at least $K_1$ times more than the ME of the existing path. Thus, the proposal is to bound the amount of energy taken away, $g(\alpha)$, using

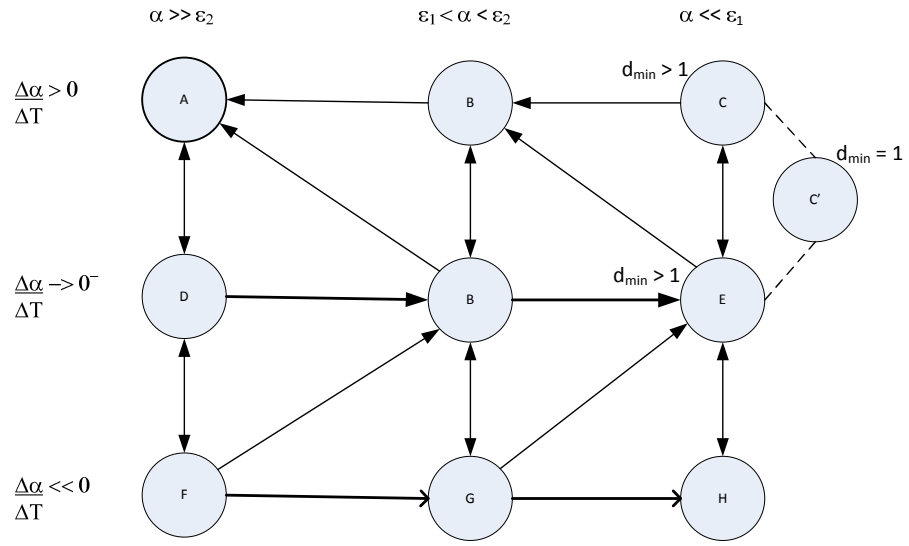



Figure A.1: Energy-state transition graph for non-stub nodes (i.e., $|G| > 1$).

the same factor of $K_1$. Specifically, we restrict the amount of energy taken away to be $min\{g(\alpha), (1-K_1) * \alpha^{FTX}\}$ when we take away from transmission energy. In doing so, taking away energy from transmission energy won't trigger "artificially created" updates in BMEP when in phases 1 and 2.

When taking away from encryption energy (we only do this in state G), we use $min\{g(\alpha), (1-K_1) * \alpha^{FR}\}$ as well. In such case, however, we don't care about artificially gaining up to a $K_1$ fudge factor of energy from encryption, (which doesn't have anything to do with the protocol as no fudge factors are used for BPR) but rather, we use the same metric just to be reciprocally fair with the previous case when giving to transmission energy and taking away from encryption energy.

Let us look at the two different scenarios of the node having more than one 1-hop neighbor and the node having only one 1-hop neighbor (i.e., the node is a stub node in the network).

### A.3.1 Non-stub node: $|G| > 1$

Fig. A.1 shows the energy-state transition graph for non-stub nodes. Let us describe each state in detail, including the node's energy assignments.

**State A:** $\alpha >> \varepsilon_2$**;** $\Delta\alpha/\Delta T > 0$

Energy allocation:
$g(\alpha) = |\Delta\alpha/\Delta T| * \Delta T * z$,
$\alpha^{TX} = \alpha^{FTX} - min\{g(\alpha), (1-K_1) * \alpha^{FTX}\}$,
$\alpha^{R} = \alpha^{FR} + min\{g(\alpha), (1-K_1) * \alpha^{FTX}\}$

where:
$z > 1$: there are 1 or more entries in the node's cache,
$\alpha >> \varepsilon_2$: the node has high energy,
$\Delta\alpha/\Delta T > 0$: the rate at which energy is growing (energy-harvesting nodes).

*Rationale*: deviate from fair allocation by taking from transmission energy and giving to encryption energy. The more paths there are in the cache, the more we take from transmission energy, but the larger the magnitude of growth in energy (in T), the more we take away from transmission energy.

**State B:** $\varepsilon_1 < \alpha < \varepsilon_2$**;** $\Delta\alpha/\Delta T -> 0$ **or** $\Delta\alpha/\Delta T > 0$

Energy allocation: *Fair allocation* as per Eq. A.1 and Eq. A.2

where:
$z > 1$: there are 1 or more entries in the node's cache,
$\varepsilon_1 < \alpha < \varepsilon_2$: moderate energy,
$\Delta\alpha/\Delta T -> 0$ or $\Delta\alpha/\Delta T > 0$: the rate at which energy changes it is slightly negative (i.e., going down slowly) or slowly growing.

**State C:** $\alpha << \varepsilon_1$**;** $\Delta\alpha/\Delta T > 0$**;** $d_{min} > 1$

Energy allocation:
$g(\alpha) = |\Delta\alpha/\Delta T| * \Delta T * (z^{h_1} * d^{h_2})$,
$\alpha^{TX} = \alpha - min\{g(\alpha), (1-K_1) * \alpha\}$,
$\alpha^R = min\{g(\alpha), (1-K_1) * \alpha\}$

where:
$z > 1$: there are 1 or more entries in the node's cache,
$\alpha << \varepsilon_1$: the node has low energy,
$|G| > 1$: the node has more than 1 neighbor,
$d >> 1$: the node is on average far from the sink,
$\Delta\alpha/\Delta T > 0$: the rate at which energy changes it is either 0 or slightly growing.

*Rationale*: initially give all energy to transmission, then take away some and give it to encryption energy. The more paths there are in the cache or the farther we are from the sink, the more we take away from transmission energy, and give to encryption. The higher the growth in energy (in T), the more we take away from transmission.

**State C':** $\alpha << \varepsilon_1$**;** $\Delta\alpha/\Delta T > 0$ **or** $\Delta\alpha/\Delta T \to 0^-$**;** $d_{min} = 1$

Energy allocation:
$\alpha^{TX} = \alpha$
$\alpha^R = 0$

where:
$d_{min} = 1$: the node is 1 hop from the sink.
All other parameters are the same as in state C.

*Rationale*: since the node is 1 hop away from the sink, in a low-energy scenario such as this one, the node chooses transmission over encryption and allocates all of its

energy to transmission. In order words, such node is not willing to do any encryption operations.

**State D:** $\alpha >> \varepsilon_2$**;** $\Delta\alpha/\Delta T \rightarrow 0^-$

Energy allocation:
$g(\alpha) = z/(1+|\Delta\alpha/\Delta T| * \Delta T)$,
$\alpha^{TX} = \alpha^{FTX} - min\{g(\alpha), (1-K_1) * \alpha^{FTX}\}$,
$\alpha^R = \alpha^{FR} + min\{g(\alpha), (1-K_1) * \alpha^{FTX}\}$

where:
$z > 1$: there are 1 or more entries in the node's cache,
$\alpha >> \varepsilon_2$: the node has high energy,
$\Delta\alpha/\Delta T \rightarrow 0^-$: the rate at which energy changes is very slow or 0.

*Rationale*: deviate from fair allocation by taking from transmission and giving to encryption energy. The more paths there are in the cache, the more we take from transmission energy, but the larger the magnitude of decline in energy (in T), the less we take out from transmission energy.

**State E:** $\alpha << \varepsilon_1$**;** $\Delta\alpha/\Delta T \rightarrow 0^-$**;** $d_{min} > 1$

Energy allocation:
$g(\alpha) = (z^{h_1} * d^{h_2})/(1+|\Delta\alpha/\Delta T| * \Delta T)$,
$\alpha^{TX} = \alpha - min\{g(\alpha), (1-K_1) * \alpha\}$,
$\alpha^R = min\{g(\alpha), (1-K_1) * \alpha\}$

where:
$z > 1$: there are 1 or more entries in the node's cache,
$\alpha << \varepsilon_1$: the node has low energy,
$|G| > 1$: the node has more than 1 neighbor,
$d >> 1$: the node is on average far from the sink,

$\Delta\alpha/\Delta T \rightarrow 0^{-}$: the rate at which energy changes it is 0 or slightly declining.

*Rationale*: initially give all energy to transmission, then take away some and give it to encryption energy. The more paths there are in the cache or the farther we are from the sink, the more we take away from transmission energy, and give to encryption. The higher the decline in energy (in T), the less we take away from transmission.

**State F:** $\alpha >> \varepsilon_2$**;** $\Delta\alpha/\Delta T << 0$

Energy allocation: *Fair Allocation* as per Eq. A.1 and Eq. A.2

where:
$z > 1$: there is more than 1 entry in the node's cache,
$\alpha >> \varepsilon_2$: node has very high energy,
$\Delta\alpha/\Delta T << 0$: the rate at which energy changes it is very negative (i.e., energy is going down fast).

*Rationale*: since energy is declining rapidly we do not take from transmission and give to encryption even though there is a lot of energy.

**State G:** $\varepsilon_1 < \alpha < \varepsilon_2$**;** $\Delta\alpha/\Delta T << 0$

Energy allocation:
$g(\alpha) = |\Delta\alpha/\Delta T| * \Delta T/(z^{h_1} * d^{h_2})$,
$\alpha^{TX} = \alpha^{FTX} + min\{g(\alpha), (1-K_1) * \alpha^{FR}\}$,
$\alpha^{R} = \alpha^{FR} - min\{g(\alpha), (1-K_1) * \alpha^{FR}\}$

where:
$z > 1$: there are 1 or more entries in the node's cache,
$\varepsilon_1 < \alpha < \varepsilon_2$: the node has moderate energy,
$\Delta\alpha/\Delta T << 0$: the rate at which energy changes it is very negative (i.e., energy is

going down fast).

*Rationale*: deviate from fair allocation by taking from encryption energy and giving to transmission energy. The amount taken is directly proportional to the magnitude of change of energy, and inversely proportional to the number of paths in the cache and the average distance of the node i from the sink. That is if energy is declining rapidly, more energy is taken away from encryption; but if there are a large number of paths in cache or the node is very far away from the sink on average, a lesser amount of energy is taken away from encryption (recall that there is a moderate amount of energy available, it just happens to be getting used up fast). The relative weighting factors are such that number of paths has more impact than average distance when they are both bigger than 1.

**State H:** $\alpha << \varepsilon_1$**;** $\Delta\alpha/\Delta T << 0$

Energy allocation:

$\alpha^{TX} = \alpha$,
$\alpha^{R} = 0$

where:

$z > 1$: there are 1 or more entries in the node's cache,
$\alpha << \varepsilon_1$: node has very low energy available,
$|G| > 1$: node has more than 1 neighbor,
$\Delta\alpha/\Delta T << 0$: the rate at which energy changes it is very negative (i.e., energy is going down fast).

*Rationale*: since the node has very little energy, give all energy to transmission and no energy to encryption.

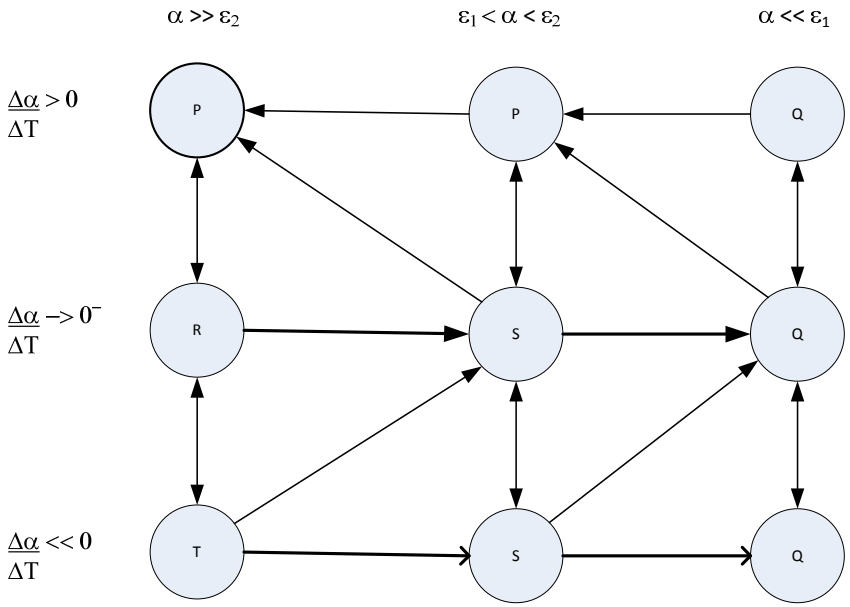



Figure A.2: Energy-state transition graph for stub nodes (i.e., $|G| = 1$).

### A.3.2 Stub node: $|G| = 1$

Fig. A.2 shows the energy-state transition graph for stub nodes. Let us describe each state in detail, including the node's energy assignments.

**State P:** $\alpha >> \varepsilon_2$**;** $\varepsilon_1 < \alpha < \varepsilon_2$**;** $\Delta\alpha/\Delta T > 0$

Energy allocation:
$g(\alpha) = |\Delta\alpha/\Delta T| * \Delta T$,
$\alpha^{TX} = \alpha^{FTX} - min\{g(\alpha), (1-K_1) * \alpha^{FTX}\}$,
$\alpha^{R} = \alpha^{FR} + min\{g(\alpha), (1-K_1) * \alpha^{FTX}\}$

where:

$\alpha >> \varepsilon_2$ or $\varepsilon_1 < \alpha < \varepsilon_2$: high or moderate energy,

$\Delta\alpha/\Delta T > 0$: the rate at which energy changes is slightly positive (i.e., available energy grows slowly).

*Rationale*: since harvesting is increasing the available energy, the stub node takes away from transmission and gives energy to encryption.

**State Q:** $\alpha << \varepsilon_1$**;** $\Delta\alpha/\Delta T > 0$ **or** $\Delta\alpha/\Delta T \to 0^-$ **or** $\Delta\alpha/\Delta T << 0$

Energy allocation: *Fair Allocation* as per Eq. A.1 and Eq. A.2

where:

$\alpha << \varepsilon_1$: node has very low energy available,

$\Delta\alpha/\Delta T > 0$ or $\Delta\alpha/\Delta T \to 0^-$ or $\Delta\alpha/\Delta T << 0$: energy either grows slowly or decreases very fast.

*Rationale*: since the node already has very little energy and its energy can go down very fast or slowly increase, follow fair allocation.

**State R:** $\alpha >> \varepsilon_2$**;** $\Delta\alpha/\Delta T \to 0^-$

Energy allocation:

$g(\alpha) = (z+1)^{h_1}/(1+|\Delta\alpha/\Delta T| * \Delta T)$,

$\alpha^{TX} = \alpha^{FTX} - min\{g(\alpha), (1-K_1) * \alpha^{FTX}\}$,

$\alpha^{R} = \alpha^{FR} + min\{g(\alpha), (1-K_1) * \alpha^{FTX}\}$

where:

$\alpha >> \varepsilon_2$: the node has high energy,

$\Delta\alpha/\Delta T \to 0^-$: the rate at which energy changes is very slow or zero.

*Rationale*: since this is a stub node, it should take away more energy from transmission than a non-stub in state D and give it to encryption.

**State S:** $\varepsilon_1 < \alpha < \varepsilon_2$**;** $\Delta\alpha/\Delta T \rightarrow 0^-$ **or** $\Delta\alpha/\Delta T << 0$

Energy allocation:
$g(\alpha) = 1/(1+|\Delta\alpha/\Delta T| * \Delta T)$,
$\alpha^{TX} = \alpha^{FTX} - min\{g(\alpha), (1-K_1) * \alpha^{FTX}\}$,
$\alpha^R = \alpha^{FR} + min\{g(\alpha), (1-K_1) * \alpha FTX\}$

where:
$e_1 < \alpha < e_2$: the node has moderate energy,
$\Delta\alpha/\Delta T \rightarrow 0^-$ or $\Delta\alpha/\Delta T << 0$: the rate at which energy changes is either zero or very negative (i.e., energy levels decrease either slowly or very fast but do not increase).

*Rationale*: since this is a stub node, it behaves as a state-D non-stub node except that we drop the variable z from the equations since z is either 1 or 2 for stub nodes.

**State T:** $\alpha >> \varepsilon_2$**;** $\Delta\alpha/\Delta T << 0$

Energy allocation: *Fair Allocation* as per Eq. A.1 and Eq. A.2

where:
$\alpha >> \varepsilon_2$: node has very high energy,
$\Delta\alpha/\Delta T << 0$: the rate at which energy changes it is very negative (i.e., energy is going down fast).

*Rationale*: since energy is declining rapidly we do not take from transmission and give to encryption even though there is a lot of energy.

## A.4 Translating $\alpha_R^i$ into $R_i$

We now need to translate the energy available for encryption at $Node_i$, $\alpha_R^i$, into the number of rounds of encryption, $R_i$, that $Node_i$ can do on a per packet basis.

Assuming that data packets are of fixed size Q bits then, we compute the total number of rounds of encryption that $Node_i$ can do as shown in Eq. A.3 that is, the total encryption energy available at the node over the energy needed for 1 round of encryption on 1 packet,

$$R_i^T = \frac{\alpha_i^R}{(E_{ENR-1Round})}, \tag{A.3}$$

where $E_{ENR-1Round}$ is given by the encryption power times the time needed to do 1 round of encryption on a packet of size Q.

Eq. A.4 shows how to compute from $R_i^T$ the number of rounds of encryption *per packet* that $Node_i$ can do,

$$R_i = \left\lfloor \frac{R_i^T}{\sum_{j=1}^{10} j * p_j} \right\rfloor, \tag{A.4}$$

where $p_j$ is the number of packets in the last T seconds for which $Node_i$ did j rounds of encryption.

# Appendix B

# Hierarchical Flooding: A Simple Routing Protocol for WSNs

## B.1 Introduction

We give a high-level description of a very simple routing protocol for wireless sensor networks. Although the routing protocol does not include any considerations on security, we think that its extreme simplicity may be beneficial to some IoT applications.

Topology wise, initially, we assume one sink and multiple sensor nodes. We will discuss extensions to a multi-sink topology in the last section. Also, we assume all nodes to be static. The routing protocol has two phases: *discovery-and-setup* phase, *routing* phase. The protocol will guarantee a route towards the sink; will also guarantee routes to be without loops. Packets will be implicitly acknowledged and control packets will be present only in the discovery-and-setup phase, hence, contributing to significant savings in node energy.

## B.2 Discovery-and-Setup Phase

When the network is "turned on" for the first time, nodes need to discover each other and the sink. In order for this to happen, the sink periodically sends a broadcast saying: "Hello, I am the sink". Whichever node can hear this broadcast will label itself as a level-zero node (L0). After doing this, each L0 node broadcasts a message saying: "Hello, I am an L0 node". Other nodes, including other L0 nodes, will hear this message. Each non-L0 node hearing the L0 broadcast will label itself as a level-one node (L1) and will broadcast a message saying "Hello, I am an L1 node", so on and so forth.

At the end of this process we will have a multihop network where each node has a label assigned telling how many hops away from the sink that node is (e.g., L1 nodes are one hop away from the sink, L2 nodes are two hops away).

For discovery operations, it is important to notice that each node will wait for a certain amount of time before labeling itself as a Layer-X node and in turn broadcast back. Similar to the discovery phase of other protocols (e.g., IEEE 802.11), this amount of time is meant to somewhat guarantee that the node has received all the broadcast advertisements it was supposed to receive.

If a node is deployed in the network at a later stage, it will listen to the wireless medium for some broadcast packets. Depending on what level nodes it can hear broadcast packets from, the node will label itself accordingly. As mentioned earlier, such a node will listen for broadcast packets for a certain amount of time before deciding on its own label (i.e., level).

## B.3 Routing Phase

In the routing phase, when a node has a packet to send, it sends it as a broadcast packet and it includes in the packet information about its own level (e.g., L10 node). Because of the way the discovery-and-setup phase works, the broadcast packet will be heard by at least one node of higher level (e.g., L9) and by zero or more nodes of lower level (e.g., L11). Since the node sending the packet has included its own

level in the packet (i.e., L10), L11 nodes hearing such broadcast will disregard it, as we do not want to route packets away form the sink. However, those nodes of higher level (i.e., L9) that hear the broadcast packet will receive it, will replace the originating node level information with their own and will broadcast it if no other node at their same level has already done so. The assumption made here is that for two nodes to receive the same packet from the same source it means that these two nodes must be physically close to each other that is, within broadcast range. If there is an obstacle between the two so that they cannot hear each other, both will forward the same packet but, eventually, this duplicate should be resolved at lower-level nodes by following the same rationale.

Each packet will contain a packet identifier that uniquely identifies that packet and the label indicating the level of the node currently broadcasting the packet.

When a lower-level node broadcasts a packet received from a higher-level node, the higher-level node will also hear that broadcast. By hearing at least one broadcast of lower-level nodes, the higher-level node knows that its broadcast message has been successfully received by at least one other node and considers this event as an implicit acknowledgement. In the case where a higher-level node (e.g., L2) does not hear the broadcast by a lower-level node (e.g., L1) of a packet it just sent, it drops the packet and stops relaying packets. It will resume normal routing operations once it hears at least one lower-level node (e.g., L1) broadcasting a packet of some other node. This means that the higher-level node has at least one fully operational lower-level node in its broadcast range which will be able to relay its packets.

## B.4 Final Observations

Although very simple and limited in functionalities, this routing protocol has some virtues. It minimizes control messages hence reduces energy consumption, it guarantees messages will be routed towards the sink, it provides routes with no loops as packets are always routed “forward” in the direction of the sink. Furthermore, it leverages broadcast packets for implicit acknowledgements.

Given the broadcast nature of the routing protocol, there is a chance for the same packet to be relayed by two different nodes at the same level. We believe, however, this event to be unluckily as it would require two nodes to receive process and send a packet at almost exactly the same time so that one node does not hear the broadcast of the other.

The described protocol can be easily extended to support multiple sinks by having each node save its level relative to each sink and by using a sink identifier. For example, one node could be a level-10 node for sink A (A10) and a level-2 node for sink B (B2). By including the correct label in the packet, other nodes will know how to route packets.

# Bibliography


[1] Colin Dixon, Ratul Mahajan, Sharad Agarwal, A. J. Brush, Bongshin Lee, Stefan Saroiu, and Paramvir Bahl. An operating system for the home. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, NSDI'12, pages 25–25, Berkeley, CA, USA, 2012. USENIX Association.

[2] Microsoft. on{X}, 2012. `https://www.onx.ms`.

[3] AT&T. Digital Life, 2012. `https://my-digitallife.att.com`.

[4] J. Postel. Internet Protocol. RFC 791 (INTERNET STANDARD), Sep 1981. Updated by RFCs 1349, 2474, 6864.

[5] S. Deering and R. Hinden. Internet Protocol, Version 6 (IPv6) Specification. RFC 2460 (Proposed Standard), Dec 1998. Updated by RFCs 5095, 5722, 5871, 6437, 6564.

[6] IEEE Standard for Local and metropolitan area networks. *Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs)*. IEEE Std 802.15.4-2011, 2011.

[7] J. Hui and P. Thubert. Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks. RFC 6282 (Proposed Standard), Sep 2011.

[8] T. Winter, P. Thubert, A. Brandt, J. Hui, R. Kelsey, P. Levis, K. Pister, R. Struik, JP. Vasseur, and R. Alexander. RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks. RFC 6550 (Proposed Standard), March 2012.

[9] Z. Shelby, K. Hartke, and B. Frank. Constrained Application Protocol (CoAP). draft-ietf-core-coap-13 (Work in progress), December 2012.

[10] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2068 (Proposed Standard), Jan 1997. Obsoleted by RFC 2616.

[11] Maria Gorlatova, Peter Kinget, Ioannis Kymissis, Dan Rubenstein, Xiaodong Wang, and Gil Zussman. Energy harvesting active networked tags (EnHANTs) for ubiquitous object networking. *IEEE Wireless Communications*, 17(6):18–25, December 2010.

[12] IEEE. *Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs): Amendment to add alternate PHY*. IEEE Std 802.15.4a-2007 (Amd. to IEEE Std 802.15.4-2006), 2007.

[13] Anne James, Joshua Cooper, Keith Jeffery, and Gunter Saake. Research Directions in Database Architectures for the Internet of Things: A Communication of the First international Workshop on Database Architectures for the Internet of Things (DAIT 2009). In *Proceedings of the 26th British National Conference on Databases: Dataspace: The Final Frontier*, BNCOD 26, pages 225–233, Berlin, Heidelberg, 2009. Springer-Verlag.

[14] Apple Inc. Bonjour, 2002. `http://www.apple.com/support/bonjour`.

[15] Mario Kolberg, Evan H Magill, and Michael Wilson. Compatibility issues between services supporting networked appliances. *Communications Magazine, IEEE*, 41(11):136–147, 2003.

[16] Masahide Nakamura, Hiroshi Igaki, and Ken-ichi Matsumoto. Feature interactions in integrated services of networked home appliances. In *Proc. of IntÕl. Conf. on Feature Interactions in Telecommunication Networks and Distributed Systems (ICFIÕ05)*, pages 236–251, 2005.

[17] Joan Daemen. Vincent Rijmen The design of Rijndael: AES – the Advanced Encryption Standard, 2002.

[18] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Advances in Cryptology*, pages 10–18. Springer, 1985.

[19] Trusted Computing Group Incorporated. Trusted Platform Module (TPM), 2011. `http://www.trustedcomputinggroup.org/resources/tpm_main_specification`.

[20] Chunxiao Li, Anand Raghunathan, and Niraj K Jha. Hijacking an insulin pump: Security attacks and defenses for a diabetes therapy system. In *e-Health Networking Applications and Services (Healthcom), 2011 13th IEEE International Conference on*, pages 150–156. IEEE, 2011.

[21] Patrick Traynor, Michael Lin, Machigar Ongtang, Vikhyath Rao, Trent Jaeger, Patrick McDaniel, and Thomas La Porta. On cellular botnets: Measuring the impact of malicious devices on a cellular network core. In *Proceedings of the 16th ACM conference on Computer and communications security*, pages 223–234. ACM, 2009.

[22] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. SIP: Session Initiation Protocol. RFC 3261 (Proposed Standard), June 2002. Updated by RFCs 3265, 3853, 4320, 4916, 5393, 5621.

[23] 3rd Generation Partnership Project (3GPP). *TS 123 228 v11. 7.0 (2013-01): IP Multimedia Subsystem (IMS) – Stage 2*. European Telecommunications Standards Institute (ETSI), 2013.

[24] Andrea Forte and Henning Schulzrinne. Template-based signaling compression for push-to-talk over cellular (PoC). *Principles, Systems and Applications of IP Telecommunications. Services and Security for Next Generation Networks*, pages 296–321, 2008.

[25] VICE Motherboard. How 1.5 Million Connected Cameras Were Hijacked to Make an Unprecedented Botnet. `http://motherboard.vice.com/read/15-million-connected-cameras-ddos-botnet-brian-krebs`, Sept. 2016.

[26] Akashi Satoh, Sumio Morioka, Kohji Takano, and Seiji Munetoh. A compact Rijndael hardware architecture with S-box optimization. In Springer, editor, *Proceedings of the 7th International Conference on the Theory and Application of Cryptology and Information Security* (ASIACRYPT) – *Advances in Cryptology*, pages 239–254. Gold Coast, Australia, December 2001.

[27] Martin Feldhofer, Johannes Wolkerstorfer, and Vincent Rijmen. AES implementation on a grain of sand. *IEE Proceedings – Information Security*, 152(1):13–20, November 2005.

[28] Johannes Wolkerstorfer, Elisabeth Oswald, and Mario Lamberger. An ASIC implementation of the AES S-Boxes. In Springer, editor, *Topics in Cryptology* (CT-RSA), pages 67–78. San Jose, CA, USA, February 2002.

[29] Stefan Tillich and Johann Großschädl. Instruction set extensions for efficient AES implementation on 32-bit processors. In Springer, editor, *Proc. of the 8th International Workshop on Cryptographic Hardware and Embedded Systems* (CHES), pages 270–284. Yokohama, Japan, Oct. 2006.

[30] Guido Bertoni, Luca Breveglieri, Pasqualina Fragneto, Marco Macchetti, and Stefano Marchesin. Efficient software implementation of AES on 32-bit platforms. In Springer, editor, *Proc. of the 5th International Workshop on Cryptographic Hardware and Embedded Systems* (CHES), pages 159–171, Cologne, Germany, Sept. 2003.

[31] Elisabeth Oswald and Kai Schramm. An efficient masking scheme for AES software implementations. In Springer, editor, *Information Security Applications*, pages 292–305. August 2006.

[32] François-Xavier Standaert, Gilles Piret, Neil Gershenfeld, and Jean-Jacques Quisquater. SEA: A scalable encryption algorithm for small embedded applications. In *International Conference on Smart Card Research and Advanced Applications*, pages 222–236. Springer, 2006.

[33] A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. Robshaw, Y. Seurin, and C. Vikkelsoe. PRESENT: An Ultra-Lightweight Block Cipher. In Springer-Verlag, editor, *Proceedings of the 9th International Workshop on Cryptographic Hardware and Embedded Systems* (CHES), pages 450–466. Vienna, Austria, September 2007.

[34] Wenling Wu and Lei Zhang. LBlock: a lightweight block cipher. In Springer, editor, *Proc. of the 9th International Conference on Applied Cryptography and Network Security* (ACNS), pages 327–344. Nerja, Spain, June 2011.

[35] David J Wheeler and Roger M Needham. TEA, a tiny encryption algorithm. In Springer, editor, *Proceedings of the 2nd International Workshop on Fast Software Encryption*, pages 363–366. Leuven, Belgium, Dec. 1995.

[36] Kyoji Shibutani, Takanori Isobe, Harunaga Hiwatari, Atsushi Mitsuda, Toru Akishita, and Taizo Shirai. Piccolo: an ultra-lightweight blockcipher. In Springer, editor, *Proceedings of the 11th International Workshop on Cryptographic Hardware and Embedded Systems* (CHES), pages 342–357. Lausanne, Switzerland, Oct. 2011.

[37] Christophe De Canniere, Orr Dunkelman, and Miroslav Knežević. KATAN and KTANTAN – a family of small and efficient hardware-oriented block ciphers. In Springer, editor, *Proceedings of the 11th International Workshop on Cryptographic Hardware and Embedded Systems* (CHES), pages 272–288. Lausanne, Switzerland, September 2009.

[38] Luca Veltri, Simone Cirani, Stefano Busanelli, and Gianluigi Ferrari. A novel batch-based group key management protocol applied to the Internet of Things. *Ad Hoc Networks*, 11(8):2724–2737, 2013.

[39] Davy Landman. Arduino AESLib. `https://github.com/DavyLandman/AESLib`, 2011–2013.

[40] T. Dierks and E. Rescorla. RFC 5246 – The Transport Layer Security (TLS) Protocol version 1.2, August 2008.

[41] Hasan Çam, Suat Özdemir, Prashant Nair, Devasenapathy Muthuavinashiappan, and H Ozgur Sanli. Energy-efficient secure pattern based data aggregation for wireless sensor networks. *Computer Communications*, 29(4):446–455, February 2006.

[42] Wendi Rabiner Heinzelman, Anantha Chandrakasan, and Hari Balakrishnan. Energy-efficient communication protocol for wireless microsensor networks. In *Proceedings of the 33rd IEEE Annual Hawaii International Conference on System Sciences* (HICSS), pages 3005–3014, Maui, HI, USA, Jan. 2000.

[43] Stephanie Lindsey and Cauligi S Raghavendra. PEGASIS: Power-efficient gathering in sensor information systems. In *IEEE Aerospace Conference Proceedings*, volume 3, pages 3–1125, Bi Sky, MT. USA, March 2002.

[44] M. Gorlatova, R. Margolies, J. Sarik, G. Stanje, J. Zhu, B. Vigraham, M. Szczodrak, L. Carloni, P. Kinget, I. Kymissis, and G. Zussman. Prototyping Energy Harvesting Active Networked Tags (EnHANTs). In *Proc. of the 32nd IEEE International Conference on Computer Communications* (INFOCOM) – mini conference, pages 585–589, Turin, Italy, April 2013.

[45] The Contiki Operating System. `http://www.contiki-os.org`.

[46] F. Osterlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt. Cross-Level Sensor Network Simulation with COOJA. In *Proceedings. 2006 31st IEEE Conference on Local Computer Networks*, pages 641–648, Nov 2006.

[47] R. Price, C. Bormann, J. Christoffersson, H. Hannu, Z. Liu, and J. Rosenberg. Signaling Compression (SigComp). RFC 3320 (Proposed Standard), January 2003. Updated by RFC 4896.

[48] 3rd Generation Partnership Project. Evolved Universal Terrestrial Radio Access (E-UTRA); Radio Resource Control (RRC); Protocol specification (Release 14). 3GPP TS 36.331.

[49] 3rd Generation Partnership Project. Technical Specification Group Core Network and Terminals; 3GPP System Architecture Evolution; (Release 8). 3GPP TS 24.801.

[50] Aleksandra Checko, Henrik L Christiansen, Ying Yan, Lara Scolari, Georgios Kardaras, Michael S Berger, and Lars Dittmann. Cloud RAN for mobile networks — a technology overview. *IEEE Communications Surveys & Tutorials*, 17(1):405–426, 2015.

[51] Jingchu Liu, Tao Zhao, Sheng Zhou, Yu Cheng, and Zhisheng Niu. CONCERT: a cloud-based architecture for next-generation cellular systems. *IEEE Wireless Communications*, 21(6):14–22, 2014.

[52] Mugen Peng, Yong Li, Zhongyuan Zhao, and Chonggang Wang. System architecture and key technologies for 5G heterogeneous cloud radio access networks. *IEEE Network*, 29(2):6–14, 2015.

[53] Moussa Ayyash, Hany Elgala, Abdallah Khreishah, Volker Jungnickel, Thomas Little, Sihua Shao, Michael Rahaim, Dominic Schulz, Jonas Hilt, and Ronald Freund. Coexistence of WiFi and LiFi towards 5G: Concepts, opportunities, and challenges. *IEEE Communnications Magazine*, 2015.

[54] Ekram Hossain, Mehdi Rasti, Hina Tabassum, and Amr Abdelnasser. Evolution toward 5G multi-tier cellular wireless networks: An interference management perspective. *IEEE Wireless Communications*, 21(3):118–127, 2014.

[55] Syed Yunas, Mikko Valkama, and Jarno Niemelä. Spectral and energy efficiency of ultra-dense networks under different deployment strategies. *IEEE Communications Magazine*, 53(1):90–100, 2015.

[56] Volker Jungnickel, Konstantinos Manolakis, Wolfgang Zirwas, Berthold Panzner, Volker Braun, Moritz Lossow, Mikael Sternad, Rikke Apelfröjd, and Tommy Svensson. The role of small cells, coordinated multipoint, and massive MIMO in 5G. *IEEE Communications Magazine*, 52(5):44–51, 2014.

[57] Naga Bhushan, Junyi Li, Durga Malladi, Rob Gilmore, Dean Brenner, Aleksandar Damnjanovic, Ravi Sukhavasi, Chirag Patel, and Stefan Geirhofer. Network densification: the dominant theme for wireless evolution into 5G. *IEEE Communications Magazine*, 52(2):82–89, 2014.

[58] Theodore S Rappaport, Shu Sun, Rimma Mayzus, Hang Zhao, Yaniv Azar, Kangping Wang, George N Wong, Jocelyn K Schulz, Mathew Samimi, and Felix Gutierrez. Millimeter wave mobile communications for 5G cellular: It will work! *IEEE Access*, 1:335–349, 2013.

[59] Tianyang Bai and Robert W Heath. Coverage and rate analysis for millimeter-wave cellular networks. *IEEE Transactions on Wireless Communications*, 14(2):1100–1114, 2015.

[60] Jian Qiao, Xuemin Shen, Jon Mark, Qinghua Shen, Yejun He, and Lei Lei. Enabling device-to-device communications in millimeter-wave 5G cellular networks. *IEEE Communications Magazine*, 53(1):209–215, 2015.

[61] Sundeep Rangan, Theodore S Rappaport, and Elza Erkip. Millimeter-wave cellular wireless networks: Potentials and challenges. *Proceedings of the IEEE*, 102(3):366–385, 2014.

[62] Peter Rost, Carlos Bernardos, Antonio Domenico, Marco Girolamo, Massinissa Lalam, Andreas Maeder, Dario Sabella, et al. Cloud technologies for flexible 5G radio access networks. *IEEE Communications Magazine*, 52(5):68–76, 2014.

[63] Carlos Bernardos, Antonio La Oliva, Pablo Serrano, Albert Banchs, Luis Miguel Contreras, Hao Jin, and Juan Carlos Zúñiga. An architecture for software defined wireless networking. *IEEE Wireless Communications*, 21(3):52–61, 2014.

[64] Kostas Pentikousis, Yan Wang, and Weihua Hu. Mobileflow: Toward software-defined mobile networks. *IEEE Communications Magazine*, 51(7):44–53, 2013.

[65] Byoung-Jo J Kim and Paul S Henry. Directions for future cellular mobile network architecture. *First Monday*, 17(12), 2012.

[66] Cullen Jennings, Bruce Lowekamp, Eric Rescorla, Salman Baset, and Henning Schulzrinne. REsource LOcation And Discovery (RELOAD) Base Protocol. RFC 6940, October 2015.

[67] Cullen Jennings, Bruce Lowekamp, Eric Rescorla, Salman Baset, Henning Schulzrinne, and Thomas C. Schmidt. A SIP Usage for RELOAD. Internet-Draft draft-ietf-p2psip-sip-18, Internet Engineering Task Force, April 2016. Work in Progress.

[68] 3rd Generation Partnership Project. Numbering, addressing and identification (Release 13). 3GPP TS 23.003.

[69] 3rd Generation Partnership Project. Domain Name System Procedures - Stage 3 (Release 13). 3GPP TS 29.303.

[70] 3rd Generation Partnership Project. IP Multimedia (IM) Subsystem - Stage 2 (Release 13). 3GPP TS 23.228.

[71] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. OpenFlow: Enabling Innovation in Campus Networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74, March 2008.

[72] D. Awduche, L. Berger, D. Gan, T. Li, V. Srinivasan, and G. Swallow. RSVP-TE: Extensions to RSVP for LSP Tunnels. RFC 3209 (Proposed Standard), December 2001. Updated by RFCs 3936, 4420, 4874, 5151, 5420.

[73] 3rd Generation Partnership Project. General Packet Radio Service (GPRS) enhancements for Evolved Universal Terrestrial Radio Access Network (E-UTRAN) access (Release 13). 3GPP TS 23.401.

[74] Gang Peng. CDN: Content Distribution Network. *CoRR*, cs.NI/0411069, 2004.

[75] Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. Live migration of virtual machines. In *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2*, pages 273–286. USENIX Association, 2005.

[76] Robert Bradford, Evangelos Kotsovinos, Anja Feldmann, and Harald Schiöberg. Live wide-area migration of virtual machines including local persistent state. In *Proceedings of the 3rd international conference on Virtual execution environments*, pages 169–179. ACM, 2007.

[77] Franco Travostino, Paul Daspit, Leon Gommans, Chetan Jog, Cees De Laat, Joe Mambretti, Inder Monga, Bas Van Oudenaarde, Satish Raghunath, and Phil Yonghui Wang. Seamless live migration of virtual machines over the MAN/WAN. *Future Generation Computer Systems*, 22(8):901–907, 2006.

[78] Michael R Hines, Umesh Deshpande, and Kartik Gopalan. Post-copy live migration of virtual machines. *ACM SIGOPS operating systems review*, 43(3):14–26, 2009.

[79] Hannes Tschofenig, Henning Schulzrinne, Andrew Newton, and Dr. Ted Hardie. LoST: A Location-to-Service Translation Protocol. RFC 5222, October 2015.

[80] Andrea Forte and Henning Schulzrinne. Location-to-Service Translation (LoST) Protocol Extensions. RFC 6451, October 2015.

[81] Arnt Gulbrandsen, Paul Vixie, and Levon Esibov. A DNS RR for specifying the location of services (DNS SRV). Internet RFC 2782, February 2000.

[82] 3rd Generation Partnership Project. Evolved Universal Terrestrial Radio Access (E-UTRA) and Evolved Universal Terrestrial Radio Access Network (E-UTRAN); Overall description; Stage 2 (Release 13). 3GPP TS 36.300.

[83] 3rd Generation Partnership Project. Evolved Universal Terrestrial Radio Access Network (E-UTRAN); X2 application protocol (X2AP) (Release 13). 3GPP TS 36.423.

[84] Mike Richmond. Clear Linux Project Wraps Containers in Speedy VMs. `https://blogs.intel.com/evangelists/2015/05/19/clear-linux/`, May 2015.

[85] Richard A. Becker, RamoÌĄn CaÌĄceres, Karrie Hanson, Ji Meng Loh, Simon Urbanek, Alexander Varshavsky, and Chris Volinsky. Clustering Anonymized Mobile Call Detail Records to Find Usage Groups. In *The First Workshop on Pervasive Urban Applications*, 2011.

[86] Rob J Hyndman and Yeasmin Khandakar. Automatic time series forecasting: the forecast package for R. *Journal of Statistical Software*, 26(3):1–22, 2008.

[87] Informate. International Smartphone Mobility Report. `http://informatemi.com/blog/?p=133`, 2015.

# Acknowledgments

The author would like to thank Prof. Gianluigi Ferrari for the time, the guidance and the patience but, above all, for the friendship. Also, he would like to thank Prof. Luca Veltri for all the very interesting discussions held in his office and for his friendship. Lastly, the author would like to thank Wei Wang, Simone Cirani and Ramesh Subbaraman for the work done together over the years and for the fun times.