Monitoring and testing in LTE networks: from experimental analysis to operational optimisation

Simone Roma

Department of Information Engineering

University of Pisa

This dissertation is submitted for the degree of

Doctor of Philosophy

May 2016

a mio padre e mia madre

Acknowledgements

Sono tante le persone che mi piacerebbe ringraziare, da coloro che mi hanno aiutato nella realizzazione della mia tesi a coloro che semplicemente mi hanno accompagnato durante la mia esperienza di *dottorando*.

Grazie a Gregorio e Christian. Non dimenticherò le innumerevoli discussioni, le giocate di *regio quarto*, le vie ferrate. Siete delle persone eccezionali, mi mancherete.

Grazie a Rosario e Stefano. Siete stati degli amici prima ancora che i miei tutor, sempre pronti a stimolare la mie curiosità e ad offrirmi un grande supporto.

Grazie a Simone. Sei stato diponibile con me fin dal primo istante, mi hai ospitato a casa tua incurante del fatto che sarei potuto essere un serial killer! Mi hai sostenuto e mi hai dato innumerevoli opportunità di crescita.

Grazie a Gianluca e a tutti gli amici del RadioLab. Calorosi e disponibili, ogni volta era un piacere tornare a Torino sapendo che vi avrei incontrato.

Grazie a Mihai, Carmen, Valentin, Christina e a tutti gli altri amici di Ixia, con cui ho condiviso 6 mesi splendidi a Bucharest (chi l'avrebbe mai detto!).

Grazie a Tiziano. Il Gemello. Dopo 8 anni passati sotto lo stesso tetto ci separiamo, purtroppo. Eppure non dimenticherò quanto fosse piacevole scambiare con te due parole la sera, quando tutti e due tornavamo a casa distrutti.

Grazie a Michele. Il Fratellone. Sei sempre stato generoso e i tuoi consigli sono sempre stati preziosi. I week end passati insieme mi riportavano indietro, a quando tutti e tre vivevamo insieme. Nostalgia canaglia!

Grazie a Serena. Soprattutto nell'ultimo anno hai dovuto sopportarmi anche come coinquilino, eppure sei sempre stata gentile e premurosa. Grazie a Laia. La tua allegria è contagiosa, riesci sempre a strapparmi un sorriso.

Grazie a Felice, mio ultimo compagno di mensa. Sei sempre stato disponibile con me. È sempre un piacere stare in tua compagnia, non sei mai banale.

Grazie a tutti gli amici pisani. Sono innumerevoli le belle serate che abbiamo passato insieme.

Grazie a mamma e papà. Grazie, grazie e ancora grazie. Senza di voi tutto questo non sarebbe mai stato possibile.

Grazie a Giusy. Non potevo che lasciare te per ultima, per dare una degna conclusione a questi ringraziamenti. Non hai mai dubitato di me, sei stata il mio solido appoggio nei miei momenti di difficoltà. Quando avevo l'impressione di non andare nella direzione giusta tu mi hai ascoltato, mi hai consigliato e confortato. Sei la mia dolce metà, senza di te non so come farei.

Abstract

L'avvento di LTE e LTE-Adavanced, e la loro integrazione con le esistenti tecnologie cellulari, GSM e UMTS, ha costretto gli operatori di rete radiomobile ad eseguire una meticolosa campagna di test e a dotarsi del giusto know-how per rilevare potenziali problemi durante il dispiegamento di nuovi servizi. In questo nuovo scenario di rete, la caratterizzazione e il monitoraggio del traffico nonchè la configurazione e l'affidibilità degli apparati di rete, sono di importanza fondamentale al fine di prevenire possibili insidie durante la distribuzione di nuovi servizi e garantire la migliore esperienza utente possibile.

Sulla base di queste osservazioni, questa tesi di dottorato offre un percorso completo di studio che va da un'analisi sperimentale ad un'ottimizzazione operativa.

Il punto di partenza del nostro lavoro è stato il monitoraggio del traffico di un eNodeB di campo con tre celle, operativo nella banda 1800 MHz. Tramite campagne di misura successive, è stato possibile seguire l'evoluzione della rete 4G dagli albori del suo dispiegamento nel 2012, fino alla sua completa maturazione nel 2015. I dati raccolti durante il primo anno, evidenziavano uno scarso utilizzo della rete LTE, dovuto essenzialmente alla limitata penetrazione dei nuovi smartphone 4G. Nel 2015, invece, abbiamo assistito ad un aumento netto e decisivo del numero di utenti che utilizzano la tecnolgia LTE, con statistiche aggregate (come gli indici di marketshare per i sistemi operativi degli smartphones, o la percentuale di traffico video) che rispecchiano i trend nazionali e internazionali. Questo importante risultato testimonia la maturità della tecnologia LTE, e ci permette di considerare il nostro eNodeB un punto di osservazione prezioso per l'analisi del traffico.

Di pari passo con l'evoluzione dell'infrastruttura, anche i telefoni cellulari hanno avuto una sorprendente evoluzione nel corso degli ultimi due decenni, a partire da dispositivi semplici con servizi di sola voce, fino agli smartphone di ultima generazione che offrono servizi innovativi, come Internet mobile, geolocalizzazione e mappe, servizi multimediali, e molti altri. Monitorare il traffico reale ci ha quindi permesso di studiare il comportamento degli utenti e individuare i servizi maggiormente utilizzati. Per questo, sono state sviluppate diverse librerie software per l'analisi del traffico. In particolare, è stato sviluppato in C++14 un framework/tool per la classificazione del traffico. Il progetto, disponibile su github, si chiama MOSEC, un acronimo per MOdular SErvice Classifier. MOSEC consente di definire

e utilizzare un numero arbitrario di plug-in, che processano il pacchetto secondo le loro logiche e possono o no ritornare un valore di classificazione. Una strategia di decisione finale consente di classificare i vari flussi, basandosi sulle classificazioni di ciascun plug-in. Abbiamo quindi validato la bontà del processi di classificazione di MOSEC utilizzando una traccia labellata come ground-truth di classificazione. I risultati mostrano una eccellente capacità di classificazione di traffico TCP-HTTP/HTTPS, mediamente superiore a quella di altri tool di classificazione (nDPI, PACE, Layer-7), ed evidenzia alcune lacune per quanto riguarda la classificazione di traffico UDP.

Le carattistiche dei flussi di traffico utente (User Plane) hanno un impatto diretto sul consumo energetico dei terminali e indiretto sul traffico di controllo (Control Plane) che viene generato. Pertanto, la conoscenza delle proprietà statistiche dei vari flussi consente di affrontare un problema del cross-layer optimization, per ridurre il consumo energetico dei terminali variando dei parametri configurabili sugli eNodeB. E' noto che la durata della batteria dei nuovi smartphone, rappresenta uno dei maggiori limiti nell'utilizzo degli stessi. In particolare, lo sviluppo di nuovi servizi e applicazioni capaci di lavorare in background, senza la diretta interazione dell'utente, ha introdotto nuovi problemi riguardanti la durata delle batterie degli smartphone e il traffico di segnalazione necessario ad acquisire/rilasciare le risorse radio. In conformità a queste osservazioni, è stato condotto uno studio approfondito sul meccanismo DRX (Discontinuous Reception), usato in LTE per consentire all'utente di risparmiare energia quando nessun pacchetto è inviato o ricevuto. I parametri DRX e RRC Inactivity Timer influenzano notevolmente l'energia consumata dai vari device. A seconda che le risorse radio siano assegnate o meno, l'UE si trova rispettivamente negli stati di RRC Connected e RRC Idle. Per valutare il consumo energetico degli smartphone, è stato sviluppato un algoritmo che associa un valore di potenza a ciascuno degli stati in cui l'UE può trovarsi. La transizione da uno stato all'altro è regolata da diversi timeout che sono resettati ogni volta che un pacchetto è inviato o ricevuto. Utilizzando le tracce di traffico reale, è stata associata una macchina a stati a ogni UE per valutare il consumo energetico sulla base dei pacchetti inviati e ricevuti. Osservando le caratteristiche statistiche del traffico User Plane è stata ripetuta la simulazione utilizzando dei valori dell'Inactivity Timer diversi da quello impiegato negli eNodeB di rete reale, alla ricerca di un buon trade-off tra risparmio energetico e aumento del traffico di segnalazione. I risultati hanno permesso di determinare che l'Inactivity Timer, settato originariamente sull'eNodeB era troppo elevato e determinava un consumo energetico eccesivo sui terminali. Diminuendone il valore fino a 10 secondi, si può ottenere un risparmio energetico fino al 50% (a secondo del traffico generato) senza aumentare considerevolemente il traffico di controllo.

I risultati dello studio di cui sopra, tuttavia, non tengono in considerazione lo stato di stress cui può essere sottoposto un eNodeB per effetto dell'aumento del traffico di segnalazione, nè, tantomeno, dell'aumento della contesa di accesso alla rete durante la procedura di RACH, necessaria per ristabilire il bearer radio (o connessione RRC) tra terminale ed eNodeB.

Valutare le performance di sistemi hardware e software per la rete mobile di quarta generazione, cosi come individuare qualsiasi possibile debolezza all'interno dell'architettura, è un lavoro complesso. Un possibile caso di studio, è proprio quello di valutare la robustezza delle Base Station quando riceve molte richieste di connessioni RRC, per effetto di una diminuzione dell'Inactivity Timer. A tal proposito, all'interno del Testing LAB di Telecom Italia, abbiamo utilizzato IxLoad, un prodotto sviluppato da Ixia, come generatore di carico per testare la robustezza di un eNodeB. I test sono consistiti nel produrre un differente carico di richieste RRC sull'interfaccia radio, similmente a quelle che si avrebbero diminuendo l'Inactivity Timer. Le proprietà statistiche del traffico di controllo sono ricavate a partire dall'analisi dalle tracce di traffico reale. I risultati hanno dimostrato che, anche a fronte di un carico sostenuto di richieste RRC solo una minima parte (percentuale inferiore all'1% nel caso più sfavorevole) di procedure fallisce. Abbassare l'inactivity timer anche a valori inferiori ai 10 secondi non è quindi un problema per la Base Station.

Rimane da valutare, infine, cosa succede a seguito dell'aumento delle richieste di accesso al canale RACH, dal punto di vista degli utenti. Quando due o più utenti tentano, simultaneamente, di accedere al canale RACH, utilizzando lo stesso preambolo, l'eNodeB potrebbe non essere in grado di decifrare il preambolo. Se i due segnali interferiscono costruttivamente, entrambi gli utenti riceveranno le stesse risorse per trasmettere il messaggio di RRC Request e, a questo punto, l'eNodeB può individuare la collisione e non trasmetterà nessun acknowledgement, forzando entrambi gli utenti a ricominciare la procedura dall'inizio. Abbiamo quindi proposto un modello analitico per calcolare la probabilità di collisione in funzione del numero di utenti e del carico di traffico offerto, quando i tempi d'interarrivo tra richieste successive é modellata con tempi iper-esponenziali. In più, abbiamo investigato le prestazioni di comunicazioni di tipo Machine-to-Machine (M2M) e Human-to-Human (H2H), valutando, al variare del numero di preamboli utilizzati, la probabilità di collisione su canale RACH, la probabilità di corretta trasmissione considerando sia il tempo di backoff che il numero massimo di ritrasmissioni consentite, e il tempo medio necessario per stabilire un canale radio con la rete di accesso. I risultati, valutati nel loro insieme, hanno consentito di esprimere delle linee guida per ripartire opportunamente il numero di preamboli tra comunicazioni M2M e H2H.

Abstract

The advent of LTE and LTE-Advanced, and their integration with existing cellular technologies, GSM and UMTS, has forced the mobile radio network operators to perform meticulous tests and adopt the right know-how to detect potential new issues, before the activation of new services. In this new network scenario, traffic characterisation and monitoring as well as configuration and on-air reliability of network equipment, is of paramount relevance in order to prevent possible pitfalls during the deployment of new services and ensure the best possible user experience.

Based on this observation, this research project offers a comprehensive study that goes from experimental analysis to operational optimization. The starting point of our work has been monitoring the traffic of an already deployed eNodeB with three cells, operative in the 1800 MHz band. Through subsequent measurement campaigns, it was possible to follow the evolution of the 4G network by the beginning of its deployment in 2012, until its full maturity in 2015. The data collected during the first year, showed a poor use of the LTE network, mainly due to the limited penetration of new 4G smartphone. In 2015, however, we appreciate a clear and decisive increase in the number of terminals using LTE, with aggregate statistics (e.g. marketshare for smartphone operating systems, or the percentage of video traffic) that reflect the national trend. This important outcome testifies the maturity of LTE technology, and allows us to consider our monitored eNodeB as a valuable vantage point for traffic analysis.

Hand in hand with the evolution of the infrastructure, even mobile phones have had a surprising evolution over the past two decades, from simple devices with only voice services, towards smartphones offering novel services such as mobile Internet, geolocation and maps, multimedia services, and many more. Monitoring the real traffic has allowed us to study the users behavior and identify the services most used. To this aim, various software libraries for traffic analysis have been developed. In particular, we developed a C/C++ library that analyses Control Plane and User Plane traffic, which provides corse and fined-grained statistics at flow-level. Another framework/tool has been exclusively dedicated to the topic of traffic classification. Among the plethora of existing tool for traffic classification we provide our own solution, developed from scratch. The project, which is available on github, is

named MOSEC, an acronym for Modular SErvice Classifier. The modularity is given by the possibility to implement multiple plug-ins, each one will process the packet according to its logic, and may or may not return a packet/flow classification. A final decision strategy allows to classify the various streams, based on the classifications of each plug-in. Despite previous approaches, the ability of keeping together multiple classifiers allows to mitigate the deficiency of each classifiers (e.g. DPI does not work when packets are encrypted or DNS queries don't have to be sent if name resolution is cached in device memory) and exploit their full-capabilities when it is feasible. We validated the goodness of MOSEC using a labelled trace synthetically created by colleagues from UPC BarcelonaTech. The results show excellent TCP-HTTP/HTTPS traffic classification capabilities, higher, on average, than those of other classification tools (NDPI, PACE, Layer-7). On the other hand, there are some shortcomings with regard to the classification of UDP traffic.

The characteristics of User Plane traffic have a direct impact on the energy consumed by the handset devices, and an indirect impact on the Control Plane traffic that is generated. Therefore, the acquaintances of the statistical properties of the various flows, allows us to deal with the problem of cross-layer optimization, that is reducing the power consumption of the terminals by varying some control plane parameters configurable on the eNodeB. It is well known that the battery life of the new smartphones is one of the major limitations in the use of the same. In particular, the birth of new services and applications capable of working in the background without direct user interaction, introduced new issues related to the battery lifetime and the signaling traffic necessary to acquire/release the radio resources. Based on these observations, we conducted a thorough study on the DRX mechanism (Discontinuous Reception), exploited by LTE to save smartphones energy when no packet is sent or received. The DRX configuration set and the RRC Inactivity Timer greatly affect the energy consumed by the various devices. Depending on which radio resources are allocated or not, the user equipment is in the states of RRC Connected and Idle, respectively. To evaluate the energy consumption of smartphones, an algorithm simulates the transition between all the possible states in which an UE can be and maps a power value to each of these states. The transition from one state to another is governed by different timeouts that are reset every time a packet is sent or received. Using the traces of real traffic, we associate a state machine to each for assessing the energy consumption on the basis of the sent and received packets. We repeated these simulations using different values of the inactivity timer, that appear to be more suitable than the one currently configured on the monitored eNodeB, looking for a good trade-off between energy savings and increased signaling traffic. The results highlighted that the Inactivity Timer set originally sull'eNodeB was too high and determined an excessive energy consumption on the terminals. Reducing the value up to 10 seconds permits to achieve energy

savings of up to 50% (depending on the underling traffic profile) without up considerably the control traffic.

The results of the study mentioned above, however, do not consider neither the stress level which the eNodeB is subject to, given the raise of signaling traffic that could occur, nor the increase of collision probability during the RACH procedure, needed to re-establish the radio bearer (or RRC connection) between the terminal and eNodeB .

Evaluate the performance of hardware and software systems for the fourth-generation mobile network, as well as identify any possible weakness in the architecture, it is a complex job. A possible case study, is precisely to assess the robustness of the base station when it receives many requests for RRC connections, as effect of a decrease of the inactivity timer. In this regard, within the Testing LAB of Telecom Italia, we used IxLoad, a product developed by Ixia, as a load generator to test the robustness of one eNodeB. The tests consisted in producing a different load of RRC request on the radio interface, similar to those that would be produced by decreasing the inactivity timer to certain values. The statistical properties for the signalling traffic are derived from the analysis of real traffic traces. The main outcomes have shown that, even in the face of an high load of RRC requests only a small part (less than 1% in the most unfavorable of the cases) of the procedure fails. Therefore, even lowering the inactivity timer at values lower than 10 seconds is not an issue for the Base Station.

Finally, remains to be evaluated how such surge of RRC request impacts on users performance. If one of the users under coverage in the RRC Idle is paged for an incoming packet or need to send an uplink packet a state transition from RRC Idle to RRC Connected is needed. At this point, the UE initiates the random access procedure by sending the random access channel preamble (RACH Preamble). When two or more users attempt, simultaneously, to access the RACH channel, using the same preamble, the eNodeB may not be able to decipher the preamble. If the two signals interfere constructively, both users receive the same resources for transmitting the RRC Request message and, at this point, the eNodeB can detect the collision and will not send any acknowledgment, forcing both users to restart the procedure from the beginning. We have proposed an analytical model to calculate the probability of a collision based on the number of users and the offered traffic load, when the interarrival time between requests is modeled with hyper-exponential times. In addition, we investigated some performance for Machine-to-Machine (M2M) and Human-to-Human (H2H) type communications, including the probability of correct transmission considering either the backoff time either the maximum number of allowed retransmissions, and the average time required to established a radio bearer with the access network. The results, considered as a whole, have made possible to express the guidelines to properly distribute the number of preambles in H2H and M2M communications.

Table of contents

List of figures

List of tables

Nomenclature

Roman Symbols

3GPP	3rd Generation Partnership Program
ARQ	Automatic Repeat Request
AWS	Amazon Web Services
BCH	Broadcast Channel
BSC	Base Station Controller
CAGR	Compound Annual Growth Rate
CCCH	Common Control Channel
CFI	Control Format Indicators
CP	Control Plane
CQI	Channel Quality Indicators
CS	Circuited Switched
CSFB	CS Fall Back
DCCH	Dedicated Control Channel
DCI	Downlink Control Information
DL	Downlink
DL-SCH	Downlink Shared Channel
DNS	Domain Name Server

DNS	Domain Name Server
DNS	Domain Name Server
DPI	Deep Packet Inspection
DPI	Deep Packet Inspection
DPI	Deep Packet Inspection
DRX	Discontinuous Reception
DSCP	Differentiated Service Code Point
DTCH	Dedicated Traffic Channel
E-RAB	E-UTRAN Radio Access Bearer
E-UTRAN	Evolved UTRAN
eNodeB	evolved NodeB
EPC	Evolved Packet Core
EPS	Evolved Packet System
GBR	Guaranteed Bit Rate
GPRS	General Packet Radio Service
GSM	Global System for Mobile communications
GTP	GPRS Tunneling Protocol
HARQ	Hybrid ARQ
HSPA	High-Speed Packet Access
HSS	Home Subscriber Server
IMEI	International Mobile Station Equipment Identity
IMSI	International mobile subscriber identity
IP	Internet Protocol
LCID	Logical Channel Identity

LTE	Long Term Evolution
LTE-A	Long Term Evolution Advanced
MBMS	Multimedia Broadcast Multicast Service
MBSFN	Multicase Broadcast Single Frequency Network
MCCH	Multicast Control Channel
MIB	Master Information Block
MIMO	Multiple Input Multiple Output
MME	Mobility Management Entity
MOSEC	MOdular SErvice Classifier
MTCH	Multicast Traffic Channel
OFDM	Orthogonal Frequency-Division Multiplexing
OTT	Over The Top
P-GW	PDN Gateway
PBCH	Physical Broadcast Channel
PCC	Policy and Charging Control
PCFICH	Physical Control Format Indicator Channel
PCRF	Policy and Charging Resource Function
PDCCH	Physical Downlink Control Channel
PDCP	Packet Data Convergence Protocol
PDN	Packet Data Network
PDN-GW	PDN Gateway
PDSCH	Physical Downlink Shared Channel
PHICH	Physical Hybrid ARQ Indicator Channel
PMCH	Physical Multicast Channel

PRACH	Physical Random Access Channel
PUCCH	Physical Uplink Control Channel
PUSCH	Physical Uplink Shared Channel
QoS	Quality of Service
RAN	Radio Access Network
RAT	Radio Access Technology
RI	Rank Indicators
RLC	Radio Link Control
RNC	Radio Network Controller
RNTI	Radio Network Temporary Identifier
RoCH	Robust Header Compression
RRC	Radio Resource Control
RRM	Radio Resource Management
S-GW	Serving Gateway
SCTP	Stream Control Transmission Protocol
SIB	System Information Block
SM	Spatial Multiplexing
SR	Scheduling Request
SRB	Signalling Radio Bearer
TAC	Type Approval code
TILAB	Telcom Italia LAB
TIM	Telecom Italia Mobile
TMSI	Temporary Mobile Subscriber Identify
UE	User Equipment

UL	Uplink
UL-SCH	Uplink Shared Channel
UMTS	Universal Mobile Telecommunication System

Chapter 1

Motivation and Scope

1.1 Motivation and Scope

The evolution and growth of mobile network is fundamentally changing the way users access the Internet and consume content and services. Mobile phones have had a surprising evolution over the last two decades, starting from simple devices with only voice services towards smartphones offering novel services such as mobile Internet, geolocation and maps, multimedia services, and many more. To fulfil the demand for high data rate and meet user expectation a new wireless interface is introduced in mobile networks as part of the fourth cellular network generation (4G).

Indeed, according to Cisco Visual Networking Index (VNI), globally, mobile data traffic will increase 10-fold between 2014 and 2019. Mobile data traffic will grow at a Compound Annual Growth Rate (CAGR) of 57 percent between 2014 and 2019, reaching 24.2 exabytes per month by 2019, and three time faster than fixed IP traffic in the same range period. Global mobile data traffic was 4 percent of total IP traffic in 2014, and will be 14 percent of total IP traffic by 2019. Furthermore, by 2019[1]:

i. there will be 5.2 billion global mobile users, up from 4.3 billion in 2014

ii. there will be 11.5 billion mobile-ready devices and connections, more than 4 billion more than there were in 2014

iii. the average mobile connection speed will increase 2.4-fold, from 1.7 Mbps in 2014 to 4.0 Mbps by 2019

[1] http://www.cisco.com/c/en/us/solutions/service-provider/visual-networking-index-vni/index.html#∼vniforecast

iv. global mobile IP traffic will reach an annual run rate of 292 exabytes, up from 30 exabytes in 2014

The increase of traffic (and services), together with the growth of users expectations and the exponential growth of network complexity, require that telecommunication operators incorporate the right know-how and the right equipment to perform an effective and efficient testing for ensuring proper placement on field of the new technologies, and for evaluating their reliability in particular situations, remarkably in high load conditions. LTE technology needs extensive studies aimed at experimentally understanding how network resources are utilised by real users in a deployed commercial network setting.

1.2 LTE and LTE-Advanced: Fundamentals

Long Term Evolution (LTE) starts from release 8, standardised by the 3rd Generation Partnership Program (3GPP) as the successor of the Universal Mobile Telecommunication System (UMTS) standard. 3GPP Technical Report 25.913 defines the key objectives of LTE as:

i. support of a flexible bandwidth up to 20 MHs,

ii. peak downlink rate of 100 Mbps when using two receiving antennas at the user equipment,

iii. peak uplink rate when using 1 transmitting antenna at the user equipment,

iv. round trip time less than 10 ms on the air interface,

v. improve downlink and uplink spectrum efficiency

LTE was designed and optimised with the assumption that all of the services would be packet-switched rather than circuit switched, thus continuing the trend set from the evolution of Global System for Mobile communications (GSM), to General Packet Radio Service (GPRS), Enhanced Data Rates for GSM Evolution (EDGE), UMTS, and High-Speed Packet Access (HSPA). Nevertheless, it still includes functionality to handle Circuited Switched (CS), e.g. CS Fall Back (CSFB) to UMTS or GSM.

LTE Advanced (LTE-A) is an evolved version of LTE with increased capabilities and improved performance. It starts from 3GPP release 10 and introduces Carrier Aggregation to provide wider effective channel bandwidths. It also introduces MIMO in the uplink direction, as well as increasing the number ot antenna elements that can can be exploited for MIMO in downlink direction.

Table 1.1 LTE and LTE Advanced Requirements

		LTE-A Requirements	LTE Requirements
Peak Throughput	Uplink	500 Mbps	50 Mbps
	Downlink	1 Gbps	100 Mbps
Peak Spectrum Efficiency	Uplink	15 bps/Hz	2.5 bps/Hz
	Downlink	30 bps/Hz	5.0 bps/Hz
Control Plane Latency	From Idle	50 ms	100 ms
	From Connected (DRX)	10 ms	50 ms
User Plane Latency		< 5 ms	< 5 ms

Table 1.1 compares some of the key requirements for LTE and LTE-Advanced, specified in TR 25.913 and 36.913 respectively.

Peak throughput requirements for LTE Advanced are 10 times greater then those for LTE. These improvements are fundamentally achieved using a combination of increased bandwidth and increased multiple antenna transmission capability. Indeed, the maximum 20 Mhz bandwidth in LTE, can be increased up to 100 MHz, and 4x4 MIMO in LTE evolves to 8x8 MIMO (even if the assumption of 100 Mbps in LTE is reached with 2x2 MIMO). Peak spectrum efficiency requirements for LTE-A are 6 times larger than in LTE. Spectrum efficiency is a measure of throughput per unit of bandwidth: thus increasing throughput while increasing the available bandwidth, does not provide an higher spectrum efficiency. The improvement shown in the table primarily resides in MIMO techniques. Control plane latencies represent the delay in moving the UE into a state where it is ready to transfer data with user plane connection. The user plane latencies represents the one-way delay between the IP layer in the UE and IP layer in the eNodeB. This value is strictly related to HARQ process the regulates transmission on the air interface on both side.

Regardless of the network capabilities, the system is nevertheless constrained by the actual capabilities of the receiver mobile equipment. That is, the UE capabilities. LTE defines five UE radio capability categories, to which a given UE has to conform to. These range from a UE not capable of MIMO transmission with a maximum throughput of 10 Mbit/s DL and 5 Mbit/s UL to a 4×4-capable MIMO terminal with up to 300 Mbit/s DL and 70 Mbit/s UL. Table 2.2 details the maximum throughput for both DL and UL, as well as their MIMO Spatial Multiplexing (SM) capabilities.

Table 1.2 UE category and capabilities

		Category				
		1	2	3	4	5
Downlink	max. throughput (Mbps)	10.3	51	102	151.2	302.4
	max. number of supported layers for SM	1	2	2	2	4
	max. number of supported streams for SM	1	2	2	2	4
Uplink	max. throughput (Mbps)	5.2	25.5	51	51	72.5
	support for 64-QAM	No	No	No	No	Yes

As a results of these appealing characteristics, LTE contributed to boost mobile connectivity and the explosion in the consumer market of smartphones and tablets. On the other hand, the pervasive usage of social networks and video on-demand has poured millions of new mobile users into the net, so that Internet mobile traffic is expected to exceed the traffic generated by the computer in the coming years.

1.2.1 LTE Architecture

The Evolved Packet System (EPS) architecture is shown in Figure 1.1. It is organised in four groups: User Equipment (UE), Evolved UTRAN (E-UTRAN), Evolved Packet Core (EPC) and Services. In order to minimise end-to-end latency, the number of network elements was reduced compared to 2G and 3G, resulting in the so-called “flat architecture”. UEs are connected with eNodesB that provides all radio interface-related functions. In contrast to prior architectures, the LTE Radio Access Network (RAN) is a meshed network where the functions previously fulfilled by the Radio Network Controller (RNC) in UMTS and/or the Base Station Controller (BSC) in GSM are integrated into the eNodeB. In order to enable a meshed RAN topology, the eNodeBs are now not only hierarchically connected to the core network but are also able to communicate with each other, which makes it potentially possible to employ eNodeB cooperation schemes to increase network performance. eNodeBs implements the following RAN functionalities:

- All PHY and MAC layer procedures, including link adaptation, Hybrid Automatic Repeat reQuest (HARQ), and cell search
- Radio Link Control (RLC): Segmentation and Automatic Repeat reQuest (ARQ) control of the radio bearers
- Packet Data Convergence Protocol (PDCP): IP header compression by means of Robust Header Compression (RoHC) and encryption of the user data streams.

- Radio Resource Control (RRC): at the C-Plane level, it controls the handover, manages Quality of Service (QoS), establishes and maintains radio bearers, manages keys (security), and controls/reports UE measurements.
- Radio Resource Management (RRM): ensures that radio resources are assigned efficiently and meeting the QoS constraints imposed by the core network. The RRM layer achieves it by means of controlling radio admission and bearers, connection mobility, and UL/DL scheduling.
- Selection of a Mobility Management Entity (MME) at UE attachment.
- Routing of the U-Plane data towards the Serving Gateway (S-GW).

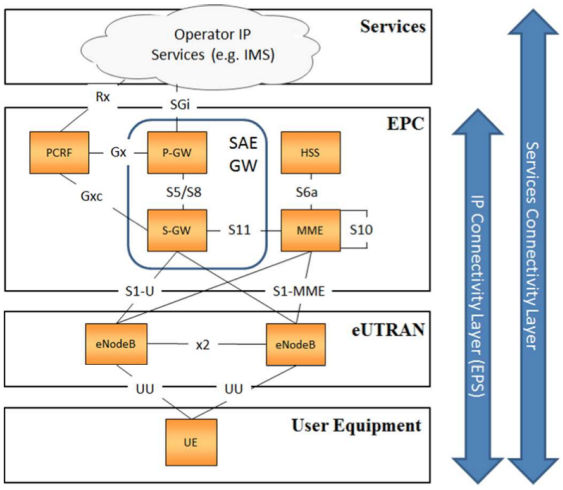

Fig. 1.1 LTE Architecture

The MME manages mobility and is used for all control plane procedures. The MME controls the access to EPC, i.e. it is responsible for attach and tracking procedure, for activation and deactivation of the bearers and for the choice of the S-GW during the initial attach procedure. It is also in charge for ciphering and integrity protection of Non-Access Stratum (NAS) signalling and for the distribution of paging messages to the eNodeB in the same tracking area.

S-GW forwards data packets, and serves as anchor for the user plane during inter-eNodeB and inter-RAT (Radio Access Technology) handovers. It manages and stores UEs contexts, like IP-related information or network internal routing information. Data packets are forwarded from eNodeB towards S-GW over a GPRS Tunneling Protocol (GTP) tunnel. Likewise, S-GW tunnels traffic to the Packet Data Network Gateway (PDN-GW).

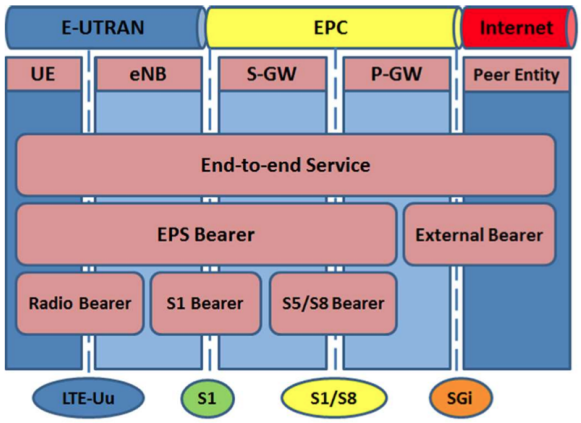

Fig. 1.2 EPS Bearer

PDN-GW, or simply P-GW, offers connectivity towards Internet and other cellular data networks. This network element represents the input/output point for UE traffic, and can block all unwanted traffic like a firewall. P-GW enforces quality of service policies and provides packet filtering and monitoring to perform billing.

Policy and Charging Resource Function (PCRF) is the network element that is responsible for Policy and Charging Control (PCC). PCRF is a server usually located with other CN elements.

EPS uses the concept of EPS bearers to route IP traffic from a gateway in the PDN to the UE. A bearer is an IP packet flow with a defined quality of service (QoS) between the gateway and the UE. The EPS bearer model is shown in Figure 1.2. The E-UTRAN and EPC together set up and release bearers as required by applications. As part of the procedure by which a UE attaches to the network, the P-GW assigns an IP address to the UE, and at least one bearer, denoted as default bearer, is established. The default bearer remains established throughout the lifetime of the PDN connection in order to provide the UE with always-on IP connectivity to that PDN. The initial bearer-level QoS parameter values of the default bearer are assigned by the MME, based on subscription data retrieved from the HSS. Other EPS bearer can be established to connect to other PDN Gateways, or to provide different QoS to the same PDN Gateway. These bearers are known as dedicated bearers, which can be either a (Guaranteed Bit Rate) GBR or a non-GBR bearer (the default bearer always has to be a non-GBR bearer, since it is permanently established). An EPS bearer is generated form the combination of E-UTRAN Radio Access Bearer (E-RAB) and S5/S8 Bearer. The S5 interface provides connectivity between the S-GW and PGW, whereas the S8 interface provides roaming connectivity for the same entities. An E-RAB bearer in turn, is composed from a combination of the Radio Bearer, which provides the connection across the radio

interface and S1 Bearer, which is established at transport network level. Further details about bearer can be found in 3GPP TS 36.300.

1.2.2 LTE Interfaces and Protocol Stacks

This work involves the study of different E-UTRAN devices and interfaces. Among all the network device shown in figure 1.1, we focused our investigation on the eNodeB and all related interface. Therefore, is worth providing further details about the air-interface and the S1 interface.

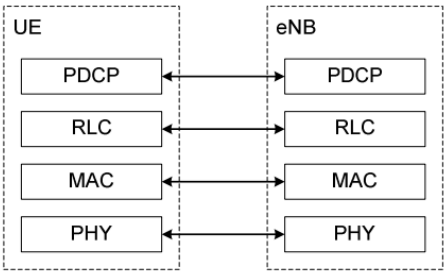

Fig. 1.3 Uu Interface protocol stack: User Plane

The air-interface connection between the UE and eNodeB is known as Uu. The radio protocol architecture of E-UTRAN is given for the user plane and the control plane. Figure 1.3 (figure 4.3.1 at 3GPP TS 36.300 version 8.12.0 Release 8) shows the protocol stack for the user-plane, where PDCP, RLC and MAC sublayers (terminated in eNodeB on the network side) perform the functions listed for the user plane in subclause 6 of 3GPP TS 36.300, e.g. header compression, ciphering, scheduling, ARQ and HARQ.

Figure 1.4 shows the protocol stack for the control-plane, where the PDCP sublayer performs the functions listed for the control plane in subclause 6 for TS 36.300, e.g. ciphering

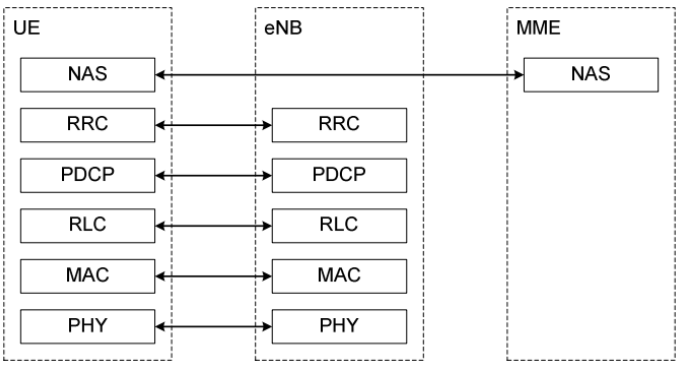

Fig. 1.4 Uu Interface protocol stack: Control Plane

and integrity protection and RLC and MAC sublayers (terminated in eNB on the network side) perform the same functions as for the user plane. The main services and functions handled by the RRC sublayer include

- the broadcast of System Information related to Non-Access Stratum (NAS) and Access-Stratum (AS), where the terminating points are the MME and eNodeB respectively
- establishment, maintenance and release of an RRC connection between the UE and E-UTRAN
- security functions including key management;
- establishment, configuration, maintenance and release of point to point Radio Bearers;
- mobility functions
- UE measurement reporting and control;

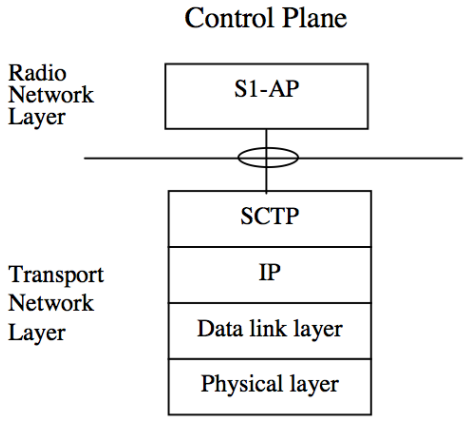

Fig. 1.5 S1 interface protocol stack: Control Plane

As for the air-interface, the S1 interface is given for the user plane, named S1-U, and the control plane, named S1-MME. From a logical standpoint, the S1-MME is a point-to-point interface between an eNodeB within the E-UTRAN and an MME in the EPC. A point-to-point logical interface should be feasible even in the absence of a physical direct connection between the eNodeB and MME. The S1-MME interface supports:

- procedures to establish, maintain and release E-UTRAN Radio Access Bearers;
- procedures to perform intra-LTE handover and inter-RAT handover;
- the separation of each UE on the protocol level for user specific signalling management;

- the transfer of NAS signalling messages between UE and EPC;
- location services by transferring requests from the EPC to E-UTRAN, and location information from E-UTRAN to EPC;

S1-MME interface consists of a Stream Control Transmission Protocol (SCTP) over IP and supports multiple UEs through a single SCTP association. It also provides guaranteed data delivery. SCTP is defined in RFC 4960. The application signaling protocol is an S1-AP (Application Protocol). LTE Transport network layer is built on IP transport, similar to the user plane but for the reliable transport of signaling messages, SCTP is added on top of the Internet Protocol.

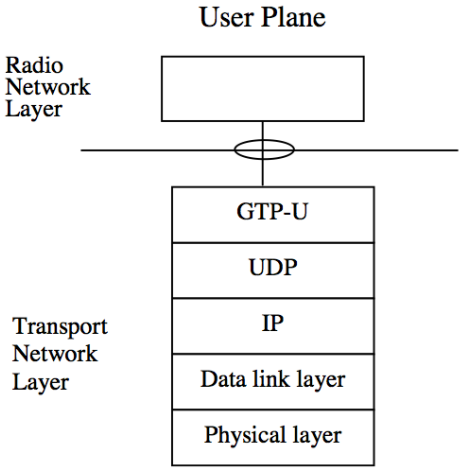

Fig. 1.6 S1 interface protocol stack: User Plane

The S1 user plane external interface (S1-U) is defined between the LTE eNodeB and the LTE S-GW. The S1-U interface provides non guaranteed data delivery of LTE user plane Protocol Data Units (PDUs) between the eNodeB and the S-GW. Transport network layer is built on IP transport and GTP-U. UDP/IP carries the user plane PDUs between the eNodeB and the S-GW. A GTP tunnel per radio bearer carries user traffic.

The S1-UP interface is responsible for delivering user data between the eNodeB and the S-GW. The IP Differentiated Service Code Point (DSCP) marking is supported for QoS per radio bearer.

1.2.3 LTE Channels

Within this thesis, we'll make reference to different LTE channels. Instead of *googling* it, here we provide the whole picture for LTE channels, both for uplink and downlink.

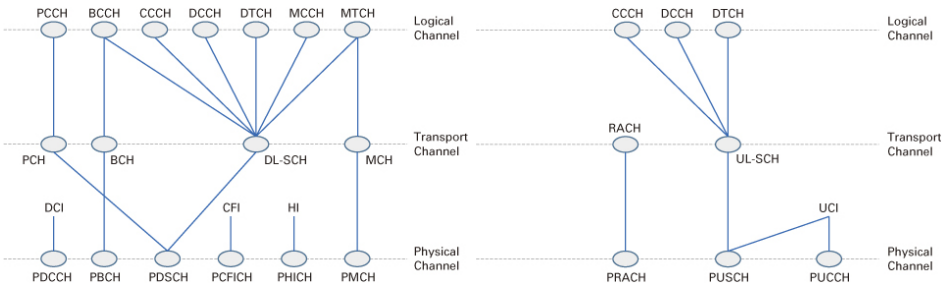

Fig. 1.7 LTE Channel Mapping: Downlink (left) and Uplink (right)

Logical channels define what type of data is transferred. These channels define the data-transfer services offered by the MAC layer. Data and signalling messages are carried on logical channels between the RLC and MAC protocols. Logical channels can be divided into control channels and traffic channels. Control channel can be either common channel or dedicated channel: common channel means common to all users in a cell (Point to multipoint) while dedicated channels means channels can be used only by one user (Point to Point).

Transport channels define how and with what characteristics the data is transferred by the physical layer. Data and signalling messages are carried on transport channels between the MAC and the physical layer.

Data and signalling messages are carried on physical channels between the different levels of the physical layer and accordingly they are divided into Physical Data Channels, comprising the Physical Downlink and Uplink Shared Channel (PDSCH, PUSCH), the Physical Broadcast Channel (PBCH), the Physical Multicast Channel (PMCH), Physical Random Access Channel (PRACH) and Physical Control Channels, comprising the Physical Control Format Indicator Channel (PCFICH), Physical Hybrid ARQ Indicator Channel (PHICH), the Physical Downlink and Uplink Control Channel (PDCCH, PUCCH). Physical data channels are distinguished by the ways in which the physical channel processor manipulates them, and by the ways in which they are mapped onto the symbols and sub-carriers used by Orthogonal Frequency-Division Multiplexing (OFDM). The transport channel processor composes several types of control information, to support the low-level operation of the physical layer.

The BCCH is used to transfer the Master Information Block (MIB) and the System Information Blocks (SIB). The MIB is then mapped to the BCH and PBC, whereas the SIB are mapped to DL-SCH and PDSCH. THe CCCH and DCCH are used to transfer RRC signalling,i.e. data belonging to the set of Signalling Radio Bearer (SRB). All SRB are mapped onto the DL-SCH (UL-SCH) and PDSCH (PUSCH). Application data are delivered through the Dedicated Traffic Channel (DTCH), the DL-SCH (UL-SCH) and PDSCH (PUSCH). For uplink data Uplink Control Information (UCI) can be added to the data from the UL-SCH during physical channel layer processing. This allows UCI to

transferred using the PUSCH when there is RRC signalling or application data to send. Application data belonging to the MBMS service are delivered through the MTCH, whereas the MCCH transfers the MBSFN area configuration message.

The PDDCH, PHICH and PCFICH are not used to transfer higher level layer information, so don't have associated logical or transport channels. The PDDCH is used to transfer Downlink control information (DCI). Further details on DCI and PDDCH con be found in Chapter 4 and Annex A. The PHICH transfers HARQ Indicators, such as the acknowledgements for uplink data and the PCFICH transfers Control Format Indicators (CFI), which specifies how many OFDMA symbols will be used to allocate the PDDCH.

When a UE receives data on PDDSCH, the PDDCH indicates whether the data belongs to the DL-SCH or the PCH. This is done by using specific Radio Network Temporary Identifier (RNTI). For example the P-RNTI indicates PCH data whereas the C-RNTI and SI-RNTI indicates DL-SCH data. From DL-SCH messages are dispatched exploiting MAC header Logical Channel Identity (LCID) where a value of 0 correspond to the CCCH (SRB0) values of 1 and 2 correspond to the DCCH (SRB1 and SRB2 respectively) and values 3 to 10 correspond to DTCH.

In the uplink chain, the PUCCH is used to transfer the UCI. As part of the information carried in the UCI we remark the precence of the Channel Quality Indicators (CQI), Rank Indicators (RI), HARQ acknowledgments and Scheduling Request (SR). The PRACH is associated to RACH transport channel but this transport channel is only used to transfer random access preamble control information from the MAC layer to the Physical layer. We'll talk exhaustively about the RACH procedure in Chapter 4.

1.3 Thesis Structure

The main sections of this thesis, which span Chapters 2 to 4, regard three different topics: i) passive monitoring and analysis of LTE live traffic ii) practical approach to optimise LTE network and extend battery life time of handset devices, iii) impact of network parameters on RRC/RACH eNodeB performance. A short summary of each of the core sections of this thesis, as well as their relationship, can be found in the subsections below.

1.3.1 Why Traffic Analysis

The advent of LTE and its integration with the existent cellular technologies (GSM, UMTS), forced network operators to perform a deep experimental analysis carried out with complex test-beds to discover possible new issues, before the activation of new services.

In this new network scenario, traffic characterisation and monitoring is of paramount relevance in order to prevent possible pitfalls during the deployment of new services.

We present the main issues evidenced by the test activity carried out in TILAB. Telecom Italia Mobile (TIM) is the major mobile operator in Italy. Thus, the selected *lesson learned* can be easily extended and generalised to other mobile operators with their own network infrastructures. The analysis considers the issues encountered during either the tests of new devices and network elements in the LTE test-bed of TILAB, or the usual maintenance and management tasks of TIM network.

This chapter reports on traffic measurements carried out at an LTE eNodeB, located in a business area of Turin. We show the evolution and the growth of the amount of LTE traffic and subscribers across three year.

Traffic analysis has been performed either by means of commercial monitoring system currently available in Telecom Italia, either by means of hand-crafted solution. In particular we developed a C/C++ software framework that analyses CP and UP traffic, which provides corse and fined-grained statistics at flow-level. The main outcome of this investigation becomes the input for the study of energy related issues, as it will be explained in Chapter 3.

Another framework has been exclusively dedicated to the topic of traffic classification. Among the plethora of existing tool for traffic classification we provide our own, developed from scratch, solution. The framework is named MOSEC, which enables modular packet classification. The modularity is given by the possibility to implement multiple plug-ins, each one will "suggest" its own packet/flow classification. Despite previous approaches, the ability of keeping together multiple classifiers allows to mitigate the deficiency of each classifiers (e.g. DPI does not work when packets are encrypted or DNS queries don't have to be sent if name resolution is cached in device memory) and exploit their full-capabilities when it is feasible.

1.3.2 UE Power Saving in LTE

The spread of mobile Internet access introduces new energy-related issues to consider. From the network side, the eNodeB is the main energy hungry element of the radio access network. Most of the power consumed by the eNodeB is due to the base band unit, the power amplifier and the cooling system. Many studies have been focused on the design of techniques for reducing power consumption in the radio access network. These studies consider strategies for the energy-efficient resource allocation, for the carrier aggregation or for switching on/off network elements depending on their load.

From the users side, the battery lifetime represents the main limitation on smartphone usage. To achieve high data rates, higher order modulations (e.g. 64-QAM), advanced coding

and antenna techniques must be used. As a result, newer smartphones need complex circuitry that quickly consumes User Equipment (UE) battery. To cope with this issue, LTE employs different mechanisms to save energy.

This chapter enlightens the Discontinuous Reception (DRX) mechanism that allows UE to power down most of its circuitry when no data needs to be sent/received, and the role of the inactivity timer, which rules the shifting between the possible UE states.

We present our algorithm to estimate the energy consumed by a generic handset device. The algorithm represents an emulator of the underlying UE state machine. It accepts as input one packet with the associated timestamp and the average monitored throughput, and return the accumulated value of the consumed energy.

We test the impact of the RRC_{IT} on the energy consumed by the UE and on the generated signalling load.

The configuration set of the algorithm is exhaustively explained and the results provide an heuristic and a practical approach to optimise LTE network and extend device battery lifetime.

1.3.3 eNodeB performance

The main outcome of Chapter 3 is that decreasing too much the RRC_{IT} leads to high control plane traffic load to eNodeB. Therefore, the drawback is that the eNodeB could be stressed by a large number of RACH/RRC request if the UE under coverage are paged for an incoming packet or need to send an uplink packet.

First of all we focus on the RACH collision probability experienced by the UEs. To this aim, we propose a model to analytically derive the RACH collision probability starting from the inter-arrival times of RACH requests produced by a generic UE. To obtain this result, as already done for energy consumption estimation, we emulate the RRC state machine taking into account real traffic acquired at a commercial eNodeB. Several emulation sessions have been carried out according to different settings of the RRC_{IT}. The output of the emulator allows to reconstruct the time-series associated with the inter-arrival times of RACH requests produced by the average UE. Mixture modelling is then applied to such time-series and used to analytically estimate the RACH collision probability.

We proof the goodness of the model, simulating the scenario in which a large variable number of handset device exists and performs RACH request either generating hyper exponential inter-arrival times or directly using the inter-arrival obtained by parsing the real data set.

The analysis here provided, gives the following enhancement:

- MTC/RAO collision probability. We carried out several simulations, under different hypothesis, and evaluate the collision probability considering both the backoff indicator scheme and the time elapsed in connected by the device
- HTC (Human Type Communication) collision probability. We inferred statistical properties from live traffic and modeled user access requests using Mixture Modeling techniques.
- H2H impact on MTC performance. Given actual traffic statistics, the collision probability has been evaluated if M2M and H2H communication share the same resource, without exploiting Access Class Barring (ACB) functionality.

Then, to evaluate the robustness of the eNodeB against burst of RRC connection requests, we set up a test using the Ixia load generator, IxLoad. According to the RRC_{IT} set on the eNodeB, we change the rate with which the RRC requests are randomly generated by the UEs camped on the base station. Again, the estimation of the RRC rate has been made looking at real traffic capture. To appreciate how much the control plane load affects the capabilities of the eNodeB we decide to measure the latency and the number of failures when establishing the RRC connection.

Chapter 2

Traffic Analysis

2.1 Introduction

As we already pointed out, globally mobile data traffic will increase 10-fold between 2014 and 2019. By 2019[1]: there will be 5.2 billion global mobile users, up from 4.3 billion in 2014, 11.5 billion mobile-ready devices and connections, more than 4 billion more than there were in 2014 and the average mobile connection speed will increase 2.4-fold, from 1.7 Mbps in 2014 to 4.0 Mbps by 2019.

In Italy, after three years of its initial deployment in the major Italian cities, 4G coverages roughly the 80% of the whole country, serving more than three thousand cities, while LTE-A has already been deployed in more than 100 cities [2].

As much the network complexity and application/traffic diversity degree arises, as well the expertise of network operators has to evolve to handle new or consolidate Over-The-Top (OTT) services. OTT services refers to delivery of audio, video, messaging and other media over the Internet without the involvement of a multiple-system operator in the control or distribution of the content. The Internet provider may be aware of the contents of the Internet Protocol packets but is not responsible for, nor able to control, the viewing abilities, copyrights, and/or other redistribution of the content. OTT in particular refers to content that arrives from a third party. The advent of such new services may be disruptive for consolidated business model and forces network operators to pursue new strategies and business opportunities. As an example, third parties provides instant messaging services as an alternative to text messaging services provided by a mobile network operator. Particularly WhatsApp narrowly focused to replace text messaging on internet connected smartphones.

[1] http://www.cisco.com/c/en/us/solutions/service-provider/visual-networking-index-vni/index.html#~vniforecast

[2] Pop outdoor commercial value, at 08/2015

The ability to intercept new trends has become essential for network operators. Thus, in this scenario, traffic characterisation and monitoring is of paramount relevance to understand users behaviour and prevent possible pitfalls during the deployment of new services.

In order to have an in-depth understanding of the evolution of LTE user plane, we carried out four measurement sessions across three years, from 2013 to 2015. Traffic data has been acquired at one eNodeB of an Italian mobile operator.

The rest of the chapter is organized as follows. Section 2.2 points out the motivations the drove our analysis, focusing on the needs for mobile network operators which try to overcome existing commercial limitations. Sections 2.3 and 2.4 provide a survey on previous researches concerning traffic measurements in mobile networks, while section 2.4 describes the measurement scenario. Section 2.6 provides the main outcomes of this study.

2.2 Lesson Learned: Telecom Italia Testing Lab

The analysis and monitoring of data networks try to shed some light on the huge black box of interconnected computers. In particular, the classification of the network traffic has become crucial for understanding the Internet.

The commercial monitoring systems used during the test are affected by several limitations or are very expensive. Many of the solutions available in the market are based on probes able to observe the traffic in the considered network points, and a collector. The collector gathers and elaborates the data acquired by the probes, and generates alarms and data on the network state for the network manager. During the tests some issues derived from the low flexibility given by the commercial solutions. In particular, the key issue is the lack of flexible solutions able to perform repetitive tasks, and to provide relevant performance parameters, taking into account that the concept of relevance is correlated with the particular network scenarios under test. The main manufacturers of network monitoring systems offer flexible and customized solutions at very high costs.

In order to continuously monitor the network under test, it is important to obtain real time information (e.g., counters and alarms), through a seamless dialogue between the probe and the collector. Monitoring is one of the most powerful way to troubleshoot, but often could be quite complicated and tedious. According to 3GPP technical specifications, hundreds of performance and measurements counters can be collected as performance indicators, such as the number of handovers, the number of call drop or the number of RRC connection established. Thus, the complexity of an E-UTRAN system has far surpassed the capability of the operators to manually analyze and diagnose problems. These issues suggest the simplification of O&M functions by means of reconfigurable probes able to

collect symptomatic information (the network manager should flexibly decide a priori which counters or messages are relevant).

The test activity pointed out the need for DPI (Deep Packet Inspection) functionality to retrieve specific information from a "target" message, to facilitate the diagnosis of the problem. For instance, DPI does not limit to the observation of an Attach Reject (or some other "bad notice"), but it analyzes the content of this message, e.g. looking at its "Cause" field. This action is important because different causes may trigger different countermeasures. Being able to quickly identify such problems becomes more important in a LTE network, due to high quality of service expectation of end users. Moreover, during test procedures, gathering immediate feedback when a failure happens in the network can help the operators to rapidly adjust misconfiguration, avoiding loss of time.

To cope with these issues, one promising alternative is the development of self-made systems exploiting open-hardware and open-software platforms. Other than a certain reliability level, these platforms should be characterised by the extreme flexibility necessary to develop self-made systems with a powerful customisation level at relatively low costs. To this aim, the first step is developing a software library able to process LTE control plane and user plane traffic. Such library leverages on the well-known *libpcap* either to capture packet on the monitored interface or to offline analyze packet captures. The library has been customized to work on the S1 iterface. It allows to:

- analyse control plane messaging on the S1-MME interface (S1-AP protocol) to identify basic LTE procedure (i.e. attach, RRC connection establishment, handover) and retrieve performance indicator (e.g. latency, signalling overhead)

- analyse user plane traffic with the aim of generating both coarse statistic (i.e. the number of UE under coverage, protocol statistics) and fine grained statistics (i.e. the number of TCP/UDP flows, the maximum inter-arrival packet within each flow)

- analyse simultaneously control and user plan to infer network parameters (RRC Inactivity Timer) or associate user activity to the correspondent EPS bearer

Explicitly for traffic classification, we designed a tool, called MOSEC. We'll present this tool in section 2.7.

Beyond the scope of this thesis, we design an integration of our software package with the NetFPGA, to allow line-rate packet processing, high–performance and cost–effective solution to LTE network monitoring lead us to adopt NetFPGA as the basic technological platform to develop our system prototype. The NetFPGA is a low cost development platform designed in the framework of the Clean Slate Project at Stanford University. In its basic version, it comes

equipped with four 1 Gigabit Ethernet that becomes four 10 Gigabit Ethernet ports in its advanced version. In addition, it is equipped with a PCI/PCIe (depending on the version) bus and it allows the hardware/software co-design of advanced networking prototypes/devices. The software part is developed on the hosting PC under Linux OS, while the hardware part is developed by using Xilinx CAD tools (usually Verilog HDL). Preliminary results for our prototype can be found in [45].

2.3 Related Works: Passive Measurement Analysis

Since the beginning of the mobile data era there has been a great interest on the characterization and measurement of mobile traffic. The different studies in this area can be classified in terminal–based and network–based studies. Terminal–based studies are aimed at characterizing applications and user behavior by acquiring data on terminals (as examples, see [29] and [53]), whereas the network–based ones attempt to evaluate network performance and usage by measurement sessions carried out through equipment installed in the network. Hence, in this latter case the user has no information about the underlying monitoring process.

Among the terminal-based studies, in [33], the authors report the results of a two-day-long user-based measurement of mobile traffic offloading by over 400 android smartphone users in Japan. They intended to characterize the usage of the 3G and WiFi of smartphones in terms of traffic offloading. On the basis of 255 users of two different smartphone platforms, the authors of [29] characterized users activities and their impact on network and battery. They found immense diversity among users and some statistics reveal differences about one or more order of magnitude. In [53], the authors investigate about diverse usage behaviors of smartphone apps. These studies investigated various aspects such as the diversity of smartphone users and the popularity of mobile applications. The 3G test study presented in [35] adopts another approach by publishing an app that actively measures various network performance metrics on users' handsets. All these measurements have the drawback that the user's behavior could be influenced by the knowledge of the presence of the application that monitors its usage of the mobile network services.

The network-based studies solve this biasing problem, in fact the user has no information about the underlying monitoring process. In [43] the authors conducted a detailed measurement analysis of network resource usage and subscriber behavior by using a large scale data set collected inside a 3G cellular data network. They studied the behavior of mobile subscribers in terms of the traffic they generate, their mobility and their activity, and find a significant variation of network usage among subscribers. Recently, in [34], the authors presented an in–depth study of the interactions among applications, network transport

protocol, and the radio layer in the LTE system. They highlighted that LTE has significantly shorter state promotion delays and lower RTTs than those of 3G networks, and pointed out various inefficiencies in TCP over LTE.

2.4 Related Works: Traffic Classification

Finding an efficient method for classifying network traffic using packet layer information is not an easy problem. An exhaustive review on traffic classification techniques is available at [51] and [54]. In general, it is possible to distinguish two main classification algorithms: i) Port and Payload-based algorithms and ii) Behavioural algorithms.

The first one leverages on packet inspection technique to retrieve service related information. Port-based techniques are widely regarded as the simplest techniques for traffic classification. Nevertheless today traffic monitoring reveals port-based techniques weakness when dealing with peer-to-peer (P2P) applications, which do not use a well-defined port [2]. Some users also configure their programs to use the well-known port for another application to avoid port-based traffic shaping or filtering mechanisms.

Pretty common and useful is to use DNS information to map each IP flow to the correspondent domain name. Mellia et al. in [17] propose DN-Hunter, a system that tags network traffic flows with their associated domain name. Their solution comprises a DNS response sniffer that decodes DNS response and, for each response, stores the set of server IPs returned for the fully qualified name (FQDN) queried. To identify the application running on a given port all the FQDNs associated to flows that are directed to that port are tokenized, and most-present tokens are used to tag the targeted port. In our previous work [47] we use a similar approach for labelling network traffic starting from DNS queries and answer. Thus, we apply a string-matching algorithm to bind each flow to a restricted number of service and gather per-service statistic results. Nevertheless, at a given point in time, using stand-alone DNS information may not be sufficient to understand the delivered service. Foremski et al. [31] leverages on the information carried in domain names and port numbers for immediate traffic classification. The weak point of DNS based algorithm is that a lot of popular services as Facebook, Youtube ar Twitter can be served by multiple CDN networks as Google CDN, Amazon Web Services or Akamai. Therefore, if in the FQDN there is no explicit reference to one of the aforementioned application, it is impossible to discern the correct application. In addition, P2P are commonly not tagged by DNS-based mechanism, given that P2P data flows are usually not preceded by DNS resolution, as has been shown in [17].

Traffic classification approaches based on deep packet inspection are considered very accurate, however, two major drawbacks are their invasiveness with respect to users privacy,

and their significant computational cost. Thus, building high performance DPI techniques require careful design, to do not become the bottleneck of the architecture. Cascarano et al. in [21] provides some advices to improve DPI performance by exploiting some common characteristics of the network traffic. They present and evaluate some optimisations that may be helpful to decrease the processing cost and can even improve the classification precision. Bujlow et al. in [20] propose a comparison of the most valuable DPI inspection techniques. Seeing which kind of information are hidden inside the packet, it allows to immediately know which kind of services that packet belongs to. For example, HTTP "Content Type" header field gives knowledge if we are getting a video, an application or an image on the fly. Fiadino et al. [30] present HTTPtag an on-line HTTP classification mechanism based on pattern matching and tagging. More in detail, every new HTTP transaction is parsed and the contacted hostname is compared against defined regular expression. If a matching pattern exists, the flow is signed with the correspondent service. Lin et al. [39] provide a survey on limitation and advantages of string matching algorithms for DPI. Expressive pattern specifications, such as regular expressions, can accurately define the signatures to speed up traffic classification. Thus, string matching, a problem once believed to be a bottleneck, has become less critical given the latest advance. Alcock et al. [15] implements a “Lightweight Packet Inspection”, where only a maximum of four bytes of payload are examined for each packet. This intuition is given by the idea that in almost all application signatures start and finish within the first 32 bytes of payload, indicating that a lightweight approach using only a small portion of payload could be viable. Finally, Portload [14] is an hybrid approach that leverages on the speed, simplicity and reduced invasiveness of port-based approaches, on a side, and the classification accuracy of DPI on the other one. Although, DPI is an effective way to identify unencrypted traffic, it is quite useless when traffic is encrypted.

The weakness against encrypted traffic pushes the interest towards behavioural classification techniques. Statistical methods propose new techniques for classifying traffic at application layer without accessing the top level payload, but using only network and transport layer information. In this way, any problem that we encounter for accessing application layer is avoided. The main concept here is that each application behaviour reflects on the network and transport level statistical properties. Traffic classification can be made observing, the number of requests/responses by any host in the network, the inter-arrival time between two packets, the size of each flow and so on and so forth. The cost that has to be payed is a training period needed to accumulate all the desired information. Crotti et al. [27] use three simple properties of the captured IP packets: their size, inter-arrival time and arrival order. These quantities are processed in order to be compared with pre-loaded structures called protocol fingerprints, which express such quantities in a compact and efficient way. The

algorithm then classify flows dynamically as packets pass through the classifier, deciding if a flow belongs to a given application layer protocol, basing on normalised threshold. In the field of statistical classification techniques, unsupervised machine-learning algorithm has gained a lot of interest in the recent years ([42]). Singh et al. [48] use unsupervised K-means and Expectation Maximization algorithm to cluster the network traffic application based on similarity between them, and compare their performance. Moore at al. [41] use a Naıve Bayes estimator to categorise traffic. At the beginning the classifier is trained throughout sets of data consisting of a service category combined with a group of flow features (e.g., flow length, port numbers, time between consecutive flows). Upon trained, the classifier is able to estimate which service a flow belongs to, just extracting those features from the flow. The drawback for behavioural methods is that the service classification depends strictly on the knowledge of the application behavior. If a new application comes out it will be not identified by the classifier, since we'll know its actual traffic feature.

Dataset Desciption

We acquired data at the S1-U interface of an eNodeB currently used in live network. The monitored eNodeB works in the bandwidth of 1800 MHz, and is located in a business area of Turin. In the same site are present other two eNodeB operative at 800/2600 Mhz. The data are acquired by means of a Tektronix K18 GbE probe, connected to a Tektronix NSA server. Traffic was treated according to security procedures and properly anonymised to respect customers privacy. Thus, no payload data is considered except for HTTP headers, and no personal information is used to develop this study.

The measurement sessions have been carried out across three years: two measurements studies in 2013 (February and October), one in 2014 (February) and one in 2015 (January). From now on we also refer to the sessions as dataset I, II, III and IV, respectively. Each sessions last seven consecutive days. Table 2.1 summarizes for each sessions the overall amount of data captured.

Table 2.1 Measurement Session: Overall Statistics

Measurement Session	Number of Days	Aggregate Amount of Data
2013 (Feb)	7	4 GB
2013 (Oct)	7	3 GB
2014	7	25 GB
2015	7	120 GB

2.5 Coarse Results

As it was expected, year–by–year the amount of traffic increases. It is worth reminding that the commercial launch of LTE took place the 7th November, 2012. Hence, the low amount of traffic in dataset I and II is mainly due to the early stage of LTE services and the lack of LTE terminal. In our previous work [46] it is possible to find further details.

Such trend is confirmed by the average number of User Equipments (UEs) under coverage in each day (figure 2.1).

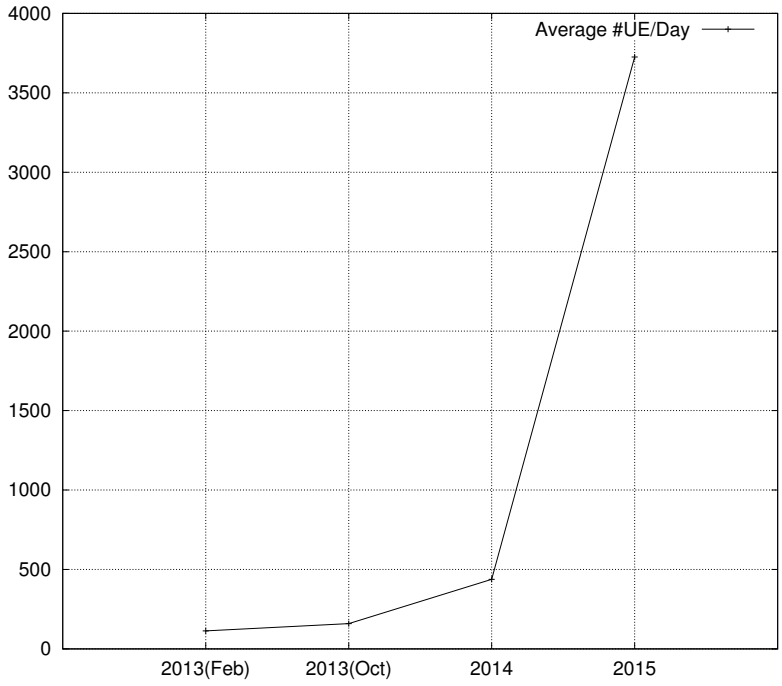

Fig. 2.1 Average UE number in each day

To obtain the average UE number we collect all the different UE IP addresses (encapsulated in the GTP tunnel) which belong to a well known IP mask. To count the effective number of users "saw" by the eNodeB in one day and obtain a more accurate analysis, we should consider the International Mobile Subscriber Identity (IMSI), or track the Temporary Mobile Subscriber Identify (TMSI), instead of the IP address. Due to the presence of NAT in cellur networks, timers are used to disconnect connections that are idle for too long. Nevertheless, LTE devices born as always-on device and NAT timers typically are large enough to ensure that no changes occur during one day. Figure 2.1 shows that in dataset I and II the average number of UE transited under the coverage of the eNodeB is slightly more than 100, and become slightly less than 500 in 2014. In 2015 more than 3500 UE are

captured during our analysis. Such surge of LTE customer witness the penetration of LTE devices and the success of new business offer.

Figure 2.2 shades some light on the handset types and operating systems which compose the device pool. To this aim, we explore the "User Agent" field in the HTTP message and associate to each IP the deducted handset type (e.g. iPhone or Andorid devices). We fail to associate one IP to the correspondent device if no HTTP message are sent or if the User Agent field either is not present or doesn't contain useful information for handset type deduction.

To fairly recognize the kind of device, the Type Approval code (TAC) has to be used. Depending on the device, the TAC consists of the first 6 or 8 numbers of the International Mobile Station Equipment Identity (IMEI). Due to privacy constraints, deep inspection of control plane messaging is not allowed, thus we cannot pursue this approach. Among all the correctly identified users, we found two kind of smartphone operating system that are prevailing: iOS and Android. We mark all the other devices (including dongle) as *Others*.

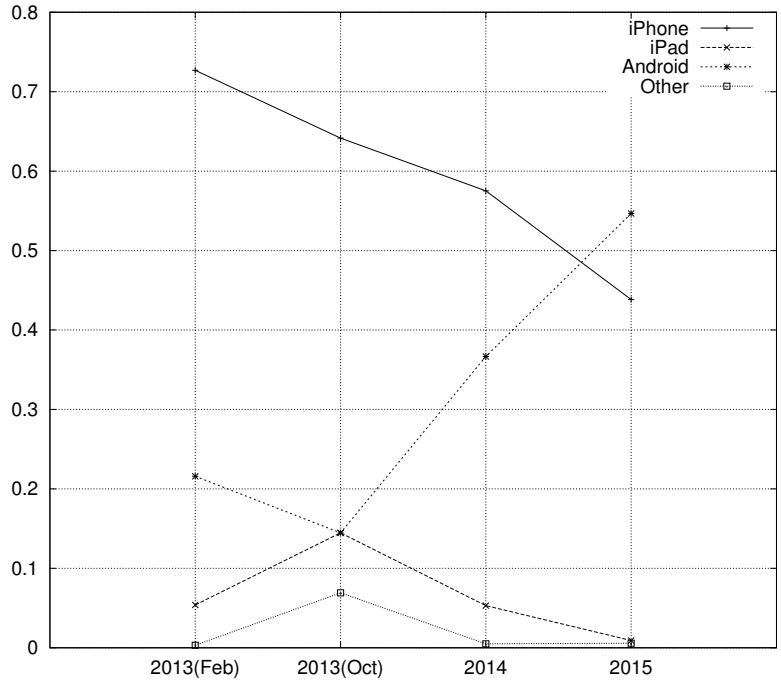

Fig. 2.2 Operating System Market Share from selected vantage point

This statistic shows the market share held by the leading operating systems for smartphones from 2013 to 2015, from our vantage point. As we can see, *Apple*'s devices represented the most used devices during the first years of LTE deployment. In 2013 they represented almost the totality of the handset device, whereas *Android* and *Other* devices account for less than 10%. This statistics contradict the actual worldwide market share for

2013 [3]. According to a forecast by IDC, the worldwide market share of Apple's iOS, which is exclusively used by Apple hardware, will remain relatively stable between 2013 and 2017, around 16%. Android OS held the highest global market share in 2013, with roaughly 80%. The explanation of why we are getting such different result is that the few number of LTE user biased this statistics. Indeed, as much as the number of LTE device increases, the amount of Andorid smartphone increases as well, showing a positive trend over these three years, and becoming the most present operating system in 2015. On the other side, the percentage of Apple device decreases, reaching less then 50% in 2015. This percentages are roughly the same as the ones presented in [3] for italian country. This important outcome testifies the maturity of LTE technology, and allows us to consider our monitored eNodeB as a valuable vantage point for traffic analysis.

2.6 Traffic Analysis

A further analysis is aimed at evaluating the traffic volume of different applications/services. To the sake of traffic classification we pursue the following approach. Firstly, we identify all the flows basing on the canonical 5-tuple: Protocol, IP source address, IP destination address, layer 4 source port, layer 4 destination port. Then, we detected the packets belonging to each session, and computed the number of exchanged bytes as well as the sessions duration. For traffic classification we used either an HTTP-based either a DNS-based classification. HTTP-based classification works simply on the *Hostname* header field in the HTTP header. The DNS-based classifier used the information retrieved by the DNS protocol to associate each resolved IP destination addresses with the associated query. One single map is maintained for all UEs under coverage to increase the hit ratio when one new flow is discovered. To avoid misleading IP resolution, we discard an entry from the map if it so not refreshed or overridden within 60 seconds. In addition, the HTTP-based algorithm has precedence over DNS-based classification. Finally, we use a service–map to marge different domains to the same service. For example, flows resolved with ".star.c10r.facebook.com", ".fbcdn-profile-a.akamaihd.net", ".fbcdn-sphotos-c-a.akamaihd.net", belong to Facebook application.

2.6.1 Application/Service Analysis

Basing on our approach, we grouped TCP connection in different applications/services and select the four ones with the largest number of TCP connections. These are Facebook, Google, Apple, Whatsapp, and Mail applications.

[3] http://www.idc.com/prodserv/smartphone-os-market-share.jsp

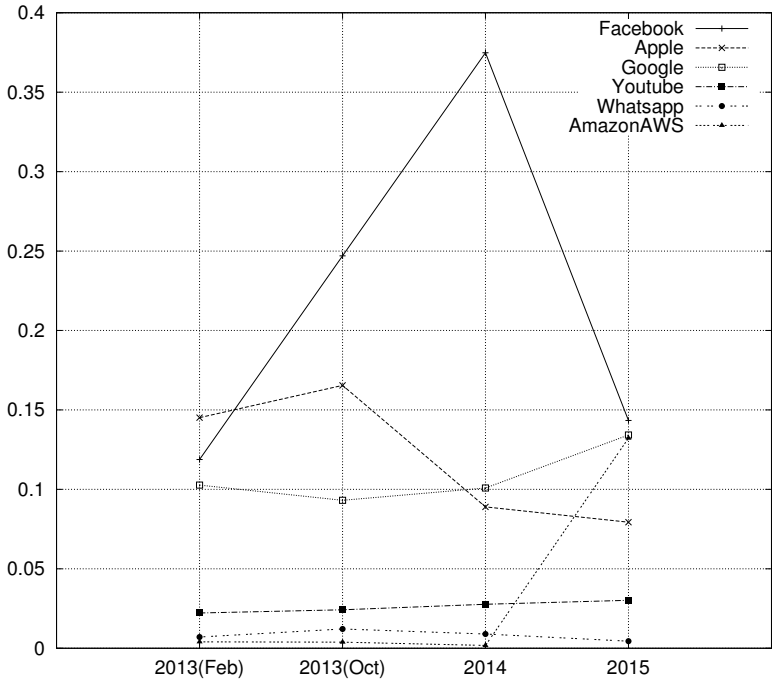

Fig. 2.3 Application Analyisis: Flow Percentage

In figure 2.3 we show the percentage of the aforementioned applications. Facebook always accounts for the highest number of flows, whit a spike in dataset III. Facebook is among the world's most popular social networking sites therefore this result is not surprising. Nevertheless, the drop of Facebook percentage in dataset IV deserves an extra explanation. As we can see, the drop of Facebook percentage coincides with an unexpected surge of traffic related to Amazon. In particular, we found a lot of connection towards Amazon Web Services (AWS). AWS is a collection of remote computing service, that make up a cloud-computing platform offered by Amazon.com. A Facebook application is, effectively, a hosted web application that utilizes the Facebook Developer API to be accessed from within the Facebook environment. Developers can host their Facebook applications on Amazon Web Services [4]. This means that, possibly, the traffic towards Amazon AWS is some kind related to Facebook as well.

Moving on, we notice a negative trend for Apple and a positive trend for Google, as the one depicted in figure 2.2 in Section 2.5. In 2014 and 2015, the higher number of Android devices increases the number of native Google applications, which contribute to augments the number of Google flows. The number of flows which belongs to Whatsapp and Youtube is stable across all the datasets, and is equal to 2-3% and 0.5%, respectively.

[4] https://aws.amazon.com/it/facebook-application-hosting/

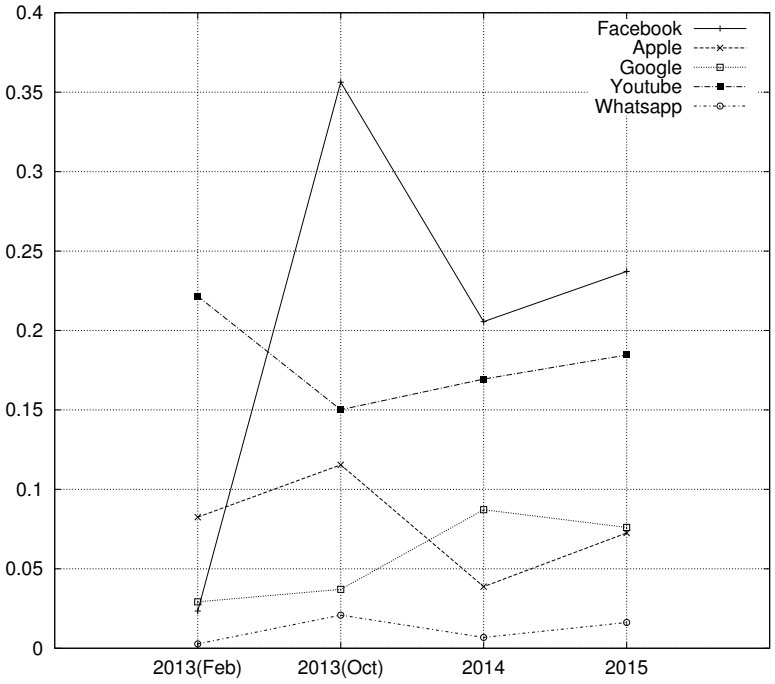

Fig. 2.4 Application Analyisis: Aggregate Data

In figure 2.4 we evaluate the aggregate amount of traffic associated to each applications.

Except for dataset I, Facebook represents more than the 20% of the traffic, with a spike in dataset II. Although counts few number of flows, Youtube is responsible, on average, of the 20% of all the traffic generated. Google and Apple give approximately the same contribution in 2015, that is slightly more than 7%. Finally, Whatsapp contribution varies little across all the years to stabilize around 2% in 2015.

2.6.2 Video Analysis

As we already pointed out, according to Cisco VNI, nearly three-fourths of the world's mobile data traffic will be video by 2019. Mobile video will increase 13-fold between 2014 and 2019, accounting for 72 percent of total mobile data traffic by the end of the forecast period. Due to the relevance of video traffic, we analyze HTTP video traffic, which can be easily identified by means of the Content Type header field.

In figure 2.5 we propose the number of video flows delivered as different container formats, normalized to the maximum value, that is 15051066. The largest number of video has been produced in dataset IV. In 2015, there is the greatest variety of containers as well, even if the .mp4 is by far the most used.

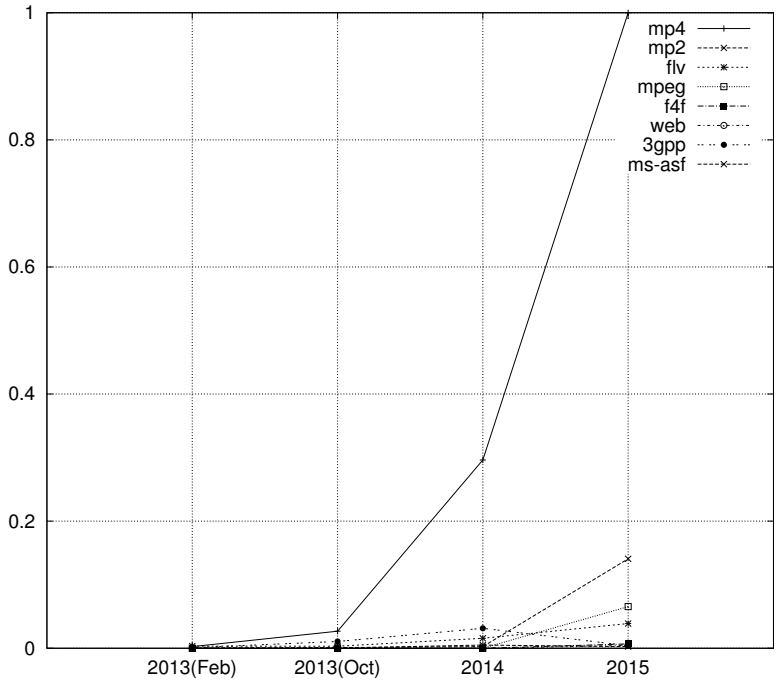

Fig. 2.5 Number of video flow

The first valuable outcome is that Advanced Video Codec (AVC or H.264) is doubtless the most widespread video codec for the mobile market. .mp4, .mp2 and .mpeg file containers are wrapper to the MPEG-4 AVC/H.264 codec, and are the most used file formats in all the datasets. Surprisingly, in dataset IV the second most used video file is the .ms-asf, which is not supported by Youtube[5]. Advanced Systems Format is Microsoft's proprietary digital audio/digital video container format, especially meant for streaming media. The format does not specify how the video or audio should be encoded; it just specifies the structure of the video/audio stream. Flash Video FLV files usually contain material encoded with codecs following the Sorenson Spark or VP6 video compression formats [6]. The most recent public releases of Flash Player also support H.264 video. We found two different flash video file formats: .flv and .f4f. .f4f indicates a Partial FLV file, thus an .f4f file contains a fragment of a Flash video. Fragmenting separates larger video files into smaller segments easy to be sent and received. Marginally, we found .web and .3gp containers. 3GP (3GPP file format) is a multimedia container format defined by the 3GPP. It is used on 3G mobile phones but can also be played on 4G phones. WebM web video format is based on VP8 and VP9 codec.

In figure 2.6 we show the amount of traffic generated per each video file format. All values are normalized with the maximum value. Until 2014 almost all video traffic belonged

[5]https://support.google.com/youtube/troubleshooter/2888402?hl=en-GB
[6]https://en.wikipedia.org/wiki/Flash_Video

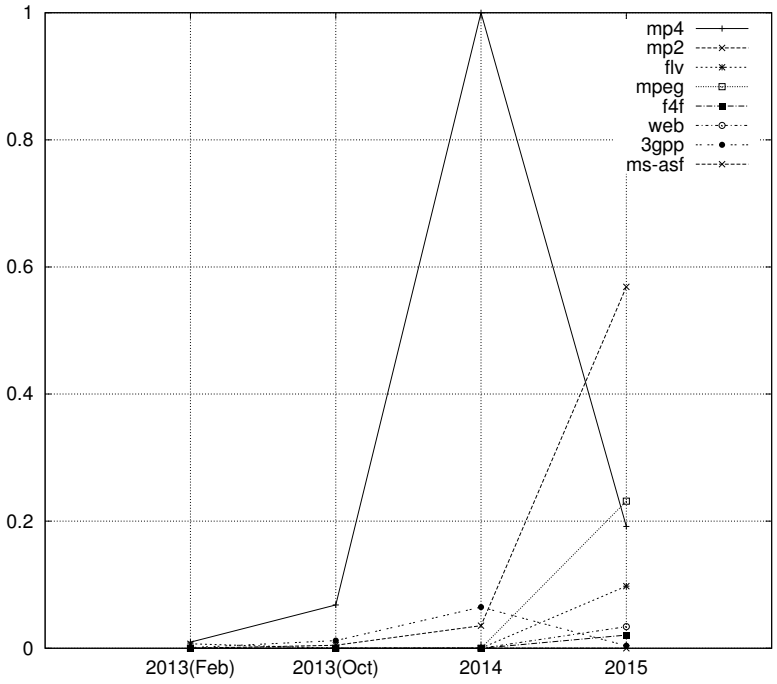

Fig. 2.6 Video data

to .mp4 file format. In 2015, even if .mp4 still accounts for the larger number of video flows (figure 2.5), the amount of video traffic is more equally spread across multiple containers. Conversely from all others video formats .3gpp is the only one which decrease in the last dataset.

2.6.3 Daily App Distribution

Here we discuss about the distribution of application flows on daily basis. Dataset I and II depicts a premature scenario, with irregular spike of traffic during working day and almost zero traffic during the week–end. This observation can be explained taking into account that the monitored eNodeB is located in a business area of Turin. Hence, we can deduce that the main part of traffic is generated by workers and trialist during the week. Moreover, the results show that, typically, the maximum throughput is reached in the interval 12:00–16:00 (e.g. the lunch break and the early afternoon), and then it progressively decreases (refer to [46]). Of course, this characteristic enforces the idea that LTE devices are not yet spread outside business environment. Due to the lack of space, we present result only for dataset III and IV. which offer more useful insights.

Our analysis compares application flow distribution for a generic working day and the week end. Each pair of figure represents the normalized distribution of flows number (left)

and the amount of data in MByte transmitted in every hour of the day (rigth). During this study we consider the following applications flow: Facebook, Apple, Google, Mail and Youtube (from left to rigth in the histogram). Figures 2.7, 2.8 and 2.10 show the well-known day-night trend. Figure 2.9 has a less pronounced day-night behavior, with low traffic not only during the night, that is from 2:00 to 7:00, but also from 17:00 to 21:00, which testifies a lack of LTE technology penetration, even in 2014. Nevertheless, all figures already point out common characteristics.

Looking the night period, there is a steady percentage of Mail flow, indicating that always-on smartphone keep receiving update notifications for their mailbox. Working days in 2014 and 2015 have a peak of usage of Facebook or Youtube between 14:00 and 15:00, and between 13:00 and 14:00 respectively. Both Facebook and Youtube traffic can be considered as entertainment, therefore it might be representative of the traffic generated during launch break. A new peak of entertainment traffic can be seen just after working hours and after dinner time, around 21:00.

Traffic distribution during week end is smoother and less predictable. Surge of entertainment traffic is not correlated to any particular human behavior and is spread more randomly during the daylight hours. Interestingly, we have spike of such kind of traffic after midnight: from 00:00 to 04:00 in 2014, and form 00:00 to 03:00 in 2015.

Apple and Google traffic can be seen as background traffic, either automatically generated by proprietary application, as push or pop notification or backup functionality (i.e. iTunes and iCloud), either triggered by generic web browsing, which involves advertisements as well (i.e. Google Advertisements Doubleclick). Spikes of such kind of traffic might represent direct interaction of the users with this services.

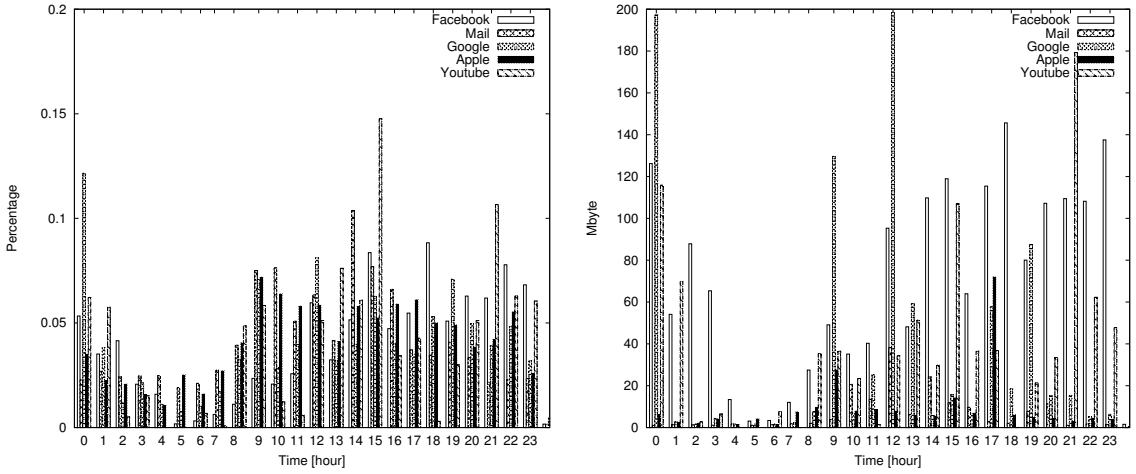

Fig. 2.7 Working Day 2014

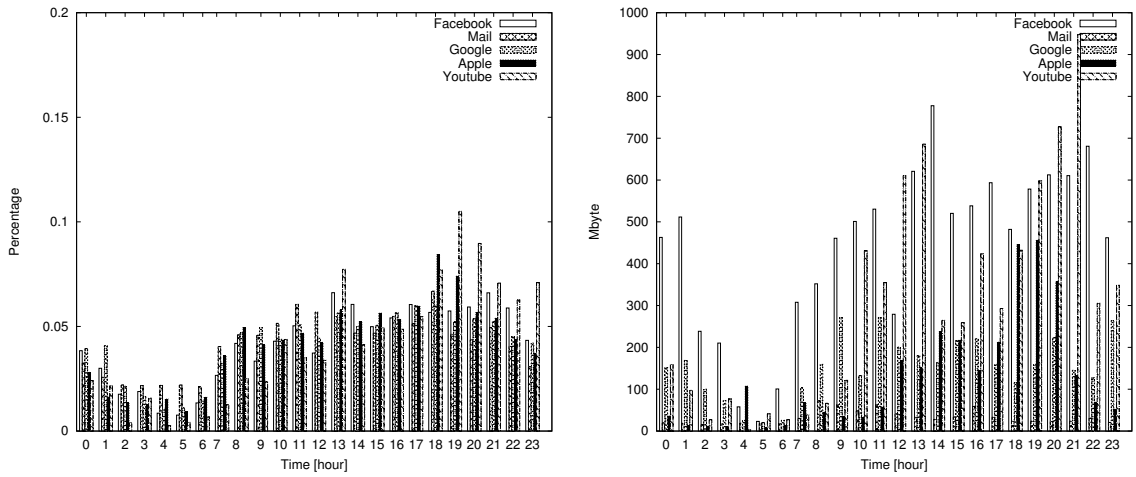

Fig. 2.8 Working Day 2015

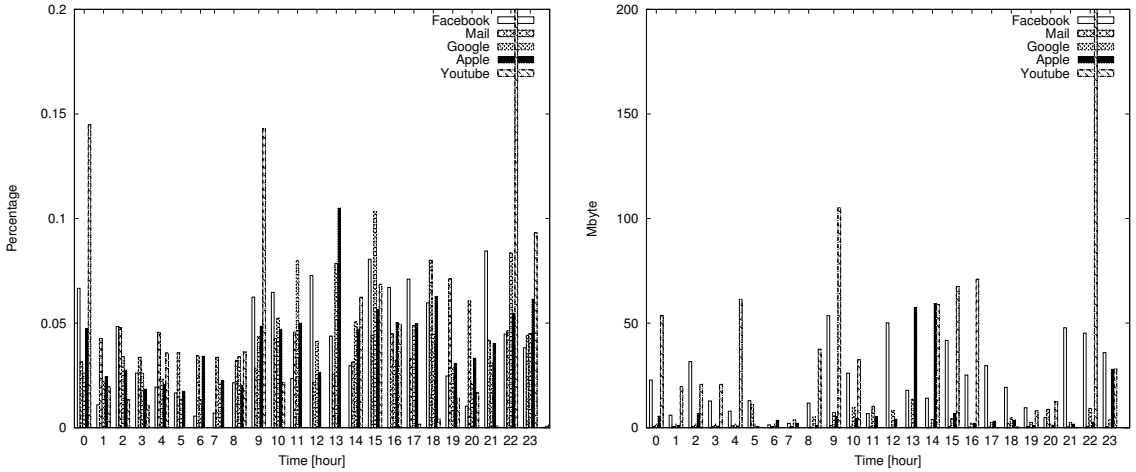

Fig. 2.9 WeekEnd 2014

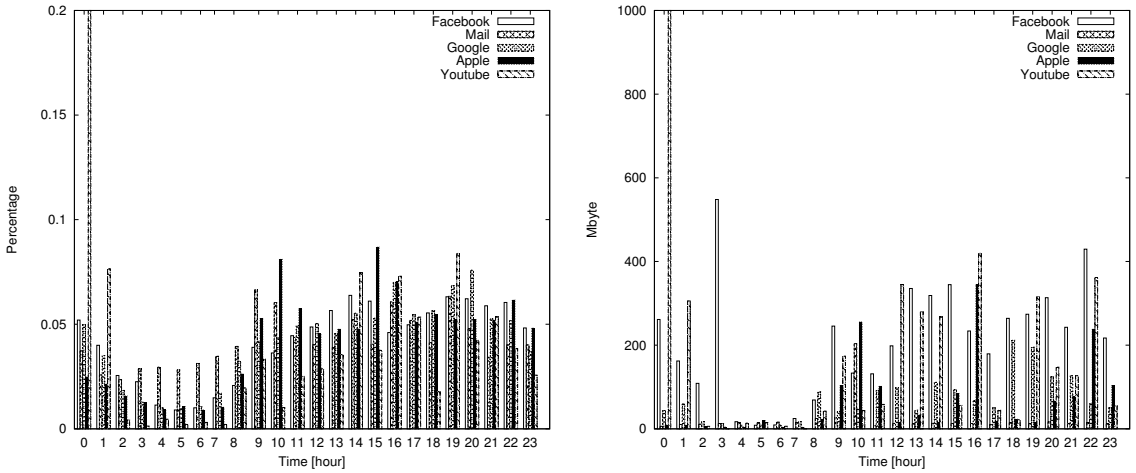

Fig. 2.10 WeekEnd 2015

Algorithm 1 Basic Classifier Algorithm

```
1:  procedure N BASIC CLASSIFIERS
2:      for each Packet do
3:          key ← make_tuple(Packet)
4:          flow_state[N] ← map.find(key)
5:          for each Classifier i ∈ N do
6:              if flow_state[i].flag = false then
7:                  Classifier ← Packet
8:                  flow_state[i].verdict = ClassType
9:                  flow_state[i].flag = true/false
10:             end ifreturn
11:             if flow_state[i].flag = true then
12:                 flow_state[i].verdict
13:             end ifreturn
14:         end for
15:     end for
16: end procedure
```

2.7 MOSEC: MOdular SErvice Classifier

All the methods described in Section 2.4 have intrinsic weak and strong points. Therefore, instead of choosing to adopt just one of them, MOSEC try to keep the best from all available classifiers.

MOSEC is a customisable framework that allows to bring all together all the traffic classifier that we need. Each classifier becomes an MOSEC plug-in which is able to produce its own per-packet classification result, either independently or dependently by other plug-in states and results. Plug-in classification results are gathered all together and the network administrator can easily implement a decision algorithm to make the final decision. A complete view of MOSEC design is provided in this section.

There are a lot of available tools for traffic classification either open source, as Tstat [6], nDPI (now nTop) [4], l-7 Filter [1], either commercial, as PACE [5]. MOSEC is an open portable framework that allows to classify packets and flows, with an arbitrary number of plug-ins. Even the aforementioned tool can be slightly modified to become one MOSEC plug-in. It takes advantages of new functionality provided in C++14 language and has been tested on Linux, Windows and iOS platforms.

Pseudo-code in 0 show the logical processing inside mosec. MOSEC has five fundamental building blocks: network processing, engine, plug-ins, decision algorithm and statistics.

2.7.1 MOSEC: Network Processing

The network processing is responsible for decoding packet in the appropriate way. Leveraging on our self-developed network processing library allows MOSEC to be portable on all the main platforms. We run MOSEC on Linux, iOS, and Windows platform successfully. MOSEC comes with standard packet processing capabilities which include:

- Ethernet and 802.1Q at layer 2
- IPv4 and IPv6 at layer 3
- TCP, UDP, ICMP and SCTP at layer 4
- DNS, HTTP and SSL at application layer

The modular structure of this tool makes easy to integrate new protocol decoding capabilities. The efforts required to parse different packet protocol involves only the modification of the *buffer* construction.

The *buffer* is a logical structure which points to each protocol element which composes the network packet. The code in figure 2.11 represent the actual implementation of the buffer.

Once a new protocol needs to be parsed in MOSEC, it is sufficient to insert the new protocol at the right position of the stack. For example, in figure 2.12 we propose the parsing procedure at layer 4.

As a result of the *make_buffer* utility we obtain the pointers to each protocol of the packet, allowing an easy access to every field in packet. The buffer structure is then passed to all the plug-ins, which can exploit the information carried out by the packet for the decision process.

```
#pragma once

#include <xmosec/network/ether.hpp>      // ethernet
#include <xmosec/network/icmp.hpp>       // icmphdr
#include <xmosec/network/ipv4.hpp>       // iphdr
#include <xmosec/network/udp.hpp>        // updhdr
#include <xmosec/network/tcp.hpp>        // tcphdr
#include <xmosec/network/sctp.hpp>       // sctphdr

#include <functional>

namespace xmosec { namespace network {

#ifdef __clang__

    // C++14 does not prevent aggregate initialization
    // when member initialization is used. clang already
    // implement this!

    struct buffer
    {
        uint8_t      const *base      = nullptr;
        size_t              size      = 0;
        ethhdr       const *eth       = nullptr;
        eth802_1q    const *eth8021q  = nullptr;
        iphdr        const *ip        = nullptr;
        tcphdr       const *tcp       = nullptr;
        udphdr       const *udp       = nullptr;
        icmphdr      const *icmp      = nullptr;
        sctphdr      const *sctp      = nullptr;
        uint8_t      const *payload   = nullptr;
        size_t       payload_len      = 0;
        size_t       number           = 0;
        uint32_t     tv_sec           = 0;
        uint32_t     tv_usec          = 0;
    };

#else
```

Fig. 2.11 MOSEC: Buffer

2.7.2 MOSEC: Engine

MOSEC engine comprises three main entities: the *Flow Information*, the *Hash Table* (or *Decision Map*), and the *Classifier Tuple*.

Flow Information structure is defined in figure 2.13.

Each plug-in maintains its own view of the flow. The internal flow *State* is defined by each plug-in (by default is *null*). The classifier::response<ClassType> is defined by the plug-ins as well, and represent the classification itself. In addition, a timestamp is updated every time one packet arrives.

MOSEC internal design represents each flow with a tuple. By default, it is possible to use the well-known 5-tuple, which comprises transport layer source and destination port, IP source and destination address and the protocol type, as well as the 3-tuple, which only consists of the IP addresses and the protocol type. The correspondence between each flow and the relative classification is maintained by means of an *Hash Table*. The *Hash Table* is

```
    if (!buf.ip)
        return buf;

    switch(inner_type(*buf.ip))
    {
        case IPPROTO_ICMP: {
            buf.icmp = parser.get<network::icmphdr>(ptr);
            buf.payload = ptr;
            buf.payload_len = buf.size - (buf.payload - buf.base);
            return buf;
        }

        case IPPROTO_TCP: {
            buf.tcp = parser.get<network::tcphdr>(ptr, [](auto *tcp) { return size(*tcp); });
            buf.payload = ptr;
            buf.payload_len = buf.size - (buf.payload - buf.base);
            return buf;
        };

        case IPPROTO_UDP: {
            buf.udp = parser.get<network::udphdr>(ptr);
            buf.payload = ptr;
            buf.payload_len = buf.size - (buf.payload - buf.base);
            return buf;
        };

        case IPPROTO_SCTP: {
            buf.sctp = parser.get<network::sctphdr>(ptr);
            buf.payload = ptr;
            buf.payload_len = buf.size - (buf.payload - buf.base);
            return buf;
        };
    }

    return buf;
}
```

Fig. 2.12 MOSEC: Decoding Process at Layar 4

an *unordered_map* whose *key* is the hash value calculated from the packet tuple, and whose *value* is a tuple of *Flow Information* structures, which carry the classifications provided by each classifier.

All *Classifiers* process the packet according to their behaviours and may or may not return a classification. The number of plug-in involved in the decision process is unknown to the engine. At code level, we leverage on *variadic-template* to inject this behavior. Plug-in may decide to not process anymore the packets which belong to a classified flow, or may decide to provide an earlier classification and keep processing flow packets until they find it necessary. This behaviour is modelled through an *enum*-flag which specifies if one packet should be delivered to the classifier. For every accepted packet each classifier attempts to provide a classification. Each plug-in returns a pair value containing the optional classification result and the *enum*-flag, which can be *Strong*, *Weak*, or *Abort*. *Strong* classification overrides previous classifications and informs MOSEC engine to not pass other packet of that flow to that classifier. *Weak* classification overrides previous classifications and inform MOSEC

```
template <typename State, typename ClassType>
struct Flow
{
    State state;
    time_point tstamp;
    classifier::response<ClassType> response;
};
```

Fig. 2.13 MOSEC: Flow Inforamtion Structure

engine to keep delivering packets of that flow. *Abort* classification does not override previous classification and informs MOSEC engine to keep delivering packets of that flow. Figure 2.14 show the actual implementation.

```
template <typename C, typename TimePoint>
auto run(C & class_i, network::buffer const &buf, optional<FlowKey> const &key, FlowStates *flow_state, TimePoint tstamp)
{
    if (flow_state)
    {
        auto & flow = std::get<Flow<typename C::FlowState, typename C::ClassType>>(*flow_state);

        flow.tstamp = tstamp;

        if (flow.response.type != classifier::resp_type::strong &&
            flow.response.type != classifier::resp_type::abort)
        {
            auto res = class_i.classify(buf, key, &flow.state);
            flow.response = flow.response << res;
        }

        return flow.response.value;
    }

    return class_i.classify(buf, key, static_cast<typename C::FlowState *>(nullptr)).value;
}
```

Fig. 2.14 MOSEC: Classification Process

All plug-ins share the same structure. Common structure enables modularity and interoperability between all plug-ins easily and efficiently. For example, figure 2.15 shows the structure for the Debug plug-in. As we can see, for each plug-in is defined:

- the name of the plug-in
- the ClassType, which defines the classification type which may be returned by the classifier. It can be **int**, **string** or whatever type you choose to define.
- the FlowState, which defines the Flow State and is used to store flow-related information
- the *init* function, for initialization purpose
- the *finish* function, to eventually flush pending information
- the classify function, which defines the logic used to classify the packet

- the *stringfy* function. We'll come back later on the stringfy concept

```
namespace xmosec { namespace classifier {

    struct debug
    {
        struct null { };

        using ClassType = int;
        using FlowState = null;

        std::string name() const
        {
            return "debug";
        }

        void init()
        {
        }

        void finish()
        {
        }

        template <typename Key>
        response<ClassType>
        classify(network::buffer const &     buf,
                 optional<Key> const &       /* key */,
                 FlowState *                 flow
                 )
        {
            std::cout << "packet@" << (void *)buf.base << " len:" << buf.size << " -> flow@" << (void *)&flow << std::endl;
            std::cout << " -> ";

            if (buf.ip)
            {
                std::cout << " " << show(*buf.ip);
                if (buf.tcp)
                    std::cout << " " << show(*buf.tcp);
                if (buf.udp)
                    std::cout << " " << show(*buf.udp);
            }

            std::cout << std::endl;

            return classifier::weak(nullopt);
        }

        std::string
        stringify(ClassType const &) const
        {
            return "debug";
        }

    };
```

Fig. 2.15 MOSEC: Debug Plug-in

Plug-in may communicate to each other by means of these data structures. For example, one classifier might ask for the classification provided by a previous classifier, and acts accordingly.

The packet flow throughout the framework is customisable as well: packet flow can be stopped at the first plug in which returns a valid classification or can be go through all classifiers until the end. MOSEC way of working may be set depending on the circumstances. For example, traffic classification by means of string matching algorithms in network applications is computationally heavy and could be reasonable to skip this kind of classification if we already have one available. On the other hand, if we leverage on behavioural algorithms

or we simply would like to proof their goodness, we can pass the packet to all classifiers and merge the results. In particular, we design three possible behaviour:

- **Basic**: each packet is forwarded to all classifiers.
- **First valid classification**: each packet is forwarded until one classifier returns a valid classification tag. In this case is important to take care of classifiers order.
- **First non-valid classification**: each packet is forwarded until one classifier returns a non valid classification tag. For port/payload based classification a good approach is to order plug-ins according to the network layer they are working on.

Figure 2.16 show the design for the *Basic* configuration.

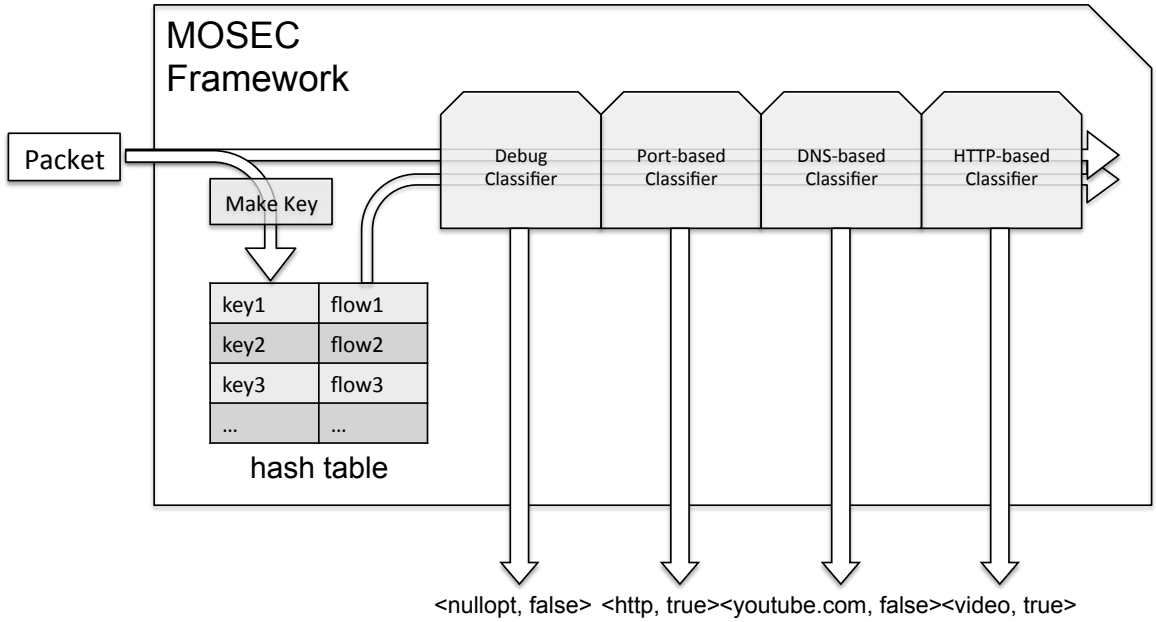

Fig. 2.16 MOSEC: Framework Design

2.7.3 MOSEC: Plug-ins

MOSEC comes with a set of already available plug-ins. Here we discuss about the implementation of these plug-ins, namely the *IP-Protocol-based*, *Port-based*, the *DNS-based*, the *HTTP-based* and the *SSL-based*.

IP-Protocol-based

The IP-Protocol-based is the simplest classifier. It extracts the *Protocol* field in the IP header and use an intarnal map to associate the correct classification.

Port-Based Plug-in

The Port-based classifier marks each packet using the implicit information provided by the port number. The list of assigned port number is available here [2], and has been used in our plug in. More precisely, The Port-based plug-in inspects the *port filed* inside TCP and UDP packets. The plug-in holds an internal map to associate each port to the correspondent service. For TCP packets, the plug-in checks that is an uplink or downlink packet. In the first case it refers to destination port, otherwise it refers to the source port. For UDP packets, the plug-in attempts to find a correspondence first selecting the source port, then the destination port. If the selected port matches one of the entry of the map, the flow is classified with the correspondent service, the classification type is tag as *strong*, port-based classifier is not call anymore for that flow. In the same way if there is no match between any of the port an the reference map, the classification type is of kind *null-strong* and the classifier is not call anymore, allowing to increase performance. In figure 2.17 we show code details.

```
template <typename Key>
response<ClassType>
classify(network::buffer const &     buf,
         optional<Key> const &       k,
         FlowState *
         )
{
    if (!k)
        return classifier::weak(nullopt);

    auto & key = k.value();

    if (buf.tcp)
    {
        if (buf.tcp->syn && !buf.tcp->ack) {
            auto it = port_map::instance().find(ntoh(key.dst_port));
            if (it != port_map::instance().end())
            {
                if (it->second.first == network::IPPROTO_ANY || it->second.first == network::IPPROTO_TCP)
                    return classifier::strong(string_view{it->second.second});
            }

            return classifier::strong(nullopt);
        }

        if (buf.tcp->syn && buf.tcp->ack) {
            auto it = port_map::instance().find(ntoh(key.src_port));
            if (it != port_map::instance().end())
            {
                if (it->second.first == network::IPPROTO_ANY || it->second.first == network::IPPROTO_TCP)
                    return classifier::strong(string_view{it->second.second});
            }

            return classifier::strong(nullopt);
        }
    }

    auto it = port_map::instance().find(ntoh(key.src_port));
    if (it != port_map::instance().end()) {
        if (it->second.first == network::IPPROTO_ANY || it->second.first == key.protocol)
            return classifier::strong(string_view{it->second.second});
    }

    it = port_map::instance().find(ntoh(key.dst_port));
    if (it != port_map::instance().end()) {
        if (it->second.first == network::IPPROTO_ANY || it->second.first == key.protocol)
            return classifier::strong(string_view{it->second.second});
    }

    return classifier::strong(nullopt);
}
```

Fig. 2.17 MOSEC: Port Plug-in

DNS-Based Plug-in

The DNS-based classification leverages on the information carried out by DNS packets. First, it inspects the *Query* inside the DNS *Answer* message, and then extracts all the CNAME and IP address within the packet.

From the *Query* name, the plug-in removes the right-most label which conveys the top-level domain. In the DNS protocol, the hierarchy of domains descends from right to left; each label to the left specifies a subdivision, or subdomain of the domain to the right. The plug-in preserve only the first subdomain level. For example, the query for *www.facebook.com* is converted to *Facebook*, which becomes the label for traffic classification. At this point the classifier maps every domain name aliases (record CNAME) and IP addrasses (record A) to the retrieved classification tag. All these entries populate one table which is managed and update by the DNS plug-in.

For every incoming packet other than DNS, the plug-in takes the IP address (first source, otherwise destination) and looks for a match inside the table. If one match occurs, the flow is tagged *strongly* and the plug-in is not call anymore.

It is worth noticing that the plug-in holds one table for all the users. Keeping one table for all UEs allows to increase performance but introduces false positive matches, meaning that flows initiated by some UEs can find a match leveraging on DNS Query/Answer initiated by other UEs. To mitigate this drawback we introduce two workarounds. First, we parse all the DNS packets and override the information inside the table, keeping the most recent IP/CNAME<–>classification mapping. This mechanism takes into account the possibility that one IP address had been relocated to convey a new service, for example *Youtube* instead of *Facebook*. Second, we introduce a timeout to validate or invalidate one match. Thus, every time the plug-in inserts a new entry in the map it adds a timestamp as well. When one match is found, if the gap between the entry timestamp and the packet timestamp exceeds the timeout value the flow remains untagged. We trained the DNS plug-in with different timeout values to tune it correctly. We used one of the available trace of dataset IV and for every timeout value we measured DNS performance in terms of classified flows. With large values (e.g. 120 seconds) we obtain great perfomance (in the order of 90%), but decreasing them the number of classified flows decreases as well, showing that previous results were drugged by false positive matches. Around 30 seconds we found approximately constant result, thus we set the timeout value to 30 seconds.

In figure 2.18 we show code details about the logic of the DNS-based classifier.

```
template <typename Key>
response<ClassType>
classify(network::buffer const &    buf,
         optional<Key> const &      k,
         FlowState *)
{
    if (buf.udp && (src_port(*buf.udp) == 53 || dst_port(*buf.udp) == 53))
    {
        if (buf.payload)
        {
            auto dns_msg = network::dns::parse_message(buf.payload, buf.payload_len);

            if (!dns_msg.query.empty())
            {
                update_state(dns_msg, buf.tv_sec);
            }

            if (is_symmetric(k.value()))
                return classifier::weak(std::string("dns"));
            else
                return classifier::weak(std::string(dns_msg.response ? "dns-response" : "dns-request"));
        }
    }

    // normal flow
    //
    if(!k)
        return classifier::weak(nullopt);

    auto now = buf.tv_sec;

    auto & key = k.value();

    if(buf.ip && (buf.ip->protocol == network::IPPROTO_TCP || buf.ip->protocol == network::IPPROTO_UDP))
    {
        auto it = service_class_.find(show(network::ipv4_t{key.src_addr}));
        if (it != service_class_.end() && ((now - it->second.tv_sec) < timeout))
            return classifier::strong(it->second.name);

        it = service_class_.find(show(network::ipv4_t{key.dst_addr}));
        if (it != service_class_.end() && ((now - it->second.tv_sec) < timeout))
            return classifier::strong(it->second.name);
    }

    return classifier::weak(nullopt);
}
```

Fig. 2.18 MOSEC: DNS Plug-in

HTTP-Based Plug-in

The HTTP-based classified inspects the HTTP header fields to provide the correct classification. The plug-in is more properly a template plug-in, which can accept an external value to specify the mode of use. There are three *modes* available:

- Mode 0: the plug-in looks for the "Hostname:" header field in the HTTP header and extract the first subdomain level, in the same way as the DNS classifier (actually both the plug-ins use the same utility function). This modified "Hostname" is the returned classification.

- Mode 1: the plug-in looks for the "Content-Type:" header field in the HTTP header. The "Content-Type" value is the returned classification.
- Mode 2: the plug-in looks for both the "Hostname:" and "Content-Type:" header field. The returned classification has the syntax "Hostname|Content-Type".

SSL-Based Plug-in

The SSL-based plug-in tries to retrieve the organization name from the Server Certificate, that is exchanged as part of the SSL Handshake mechanism. First of all the plug-in checks that the port field in the TCP packet is one of the following: 443, 465, 636, 989, 990, 992, 993, 994, 995, 5061, and 5228. The client initiates a session by sending a Client Hello message to the server. The client sends the version number corresponding to the highest version it supports. Thus the plug-in verifies that the TSL version is one among version 1.0, 1.1. and 1.2. The server responds with a Server Hello message. Among all the other fields, the Server Hello message includes the Server Certificate. The server certificate contains the server's public key, which the client will use to authenticate the server and to encrypt the premaster secret, and the organization name which delivered the certificate. In particular, there should be a list of certificate length and certificates. The first certificate is the server certificate, the second it's signing CA, the third the CA that signed the CA, etc. When more than one certificate is present the SSL plug-in refers to the first certificate.

Once the organization name is acquired, the plug-in uses this information to *strongly* label the flow. In addition, a cache mechanism is used to augment the hit ratio when no certificates are found.

In figure 2.19 we show code details about the logic of the SSL-based classifier.

```
if (is_ssl_port(sport) || is_ssl_port(dport))
{
    /* abort after 20 packets */

    if (flow->counter++ > 20)
    {
        auto it = this->cache.find(network::ipv4_t{buf.ip->saddr});

        if (it != this->cache.end())
            return classifier::strong(it->second);
        it = this->cache.find(network::ipv4_t{buf.ip->daddr});
        if (it != this->cache.end())
            return classifier::strong(it->second);

        return classifier::abort();
    }

    if (buf.payload_len > 9)
    {
        if (is_TLS_handshake(buf.payload) && get_TLS_ver(buf.payload+1))
        {
            switch(buf.payload[5])
            {
                case 0x1:
                case 0x2:
                case 0xb:
                    flow->set(buf.payload[5]);
            }
        }

        switch(flow->value)
        {
            case 0x1: {

                auto r = ssl_client_hello(buf.payload+5, buf.payload_len-5);
                if (r.value) {
                    this->cache[network::ipv4_t{buf.ip->daddr}] = r.value.value();
                    flow->reset();
                }
                return r;
            }
            case 0x2:
            case 0xb: {
                auto r = ssl_server_hello(buf.payload+5, buf.payload_len-5);
                if (r.value) {
                    cache[network::ipv4_t{buf.ip->saddr}] = r.value.value();
                    flow->reset();
                }
                return r;
            }
            default:
                return classifier::weak(nullopt);
        }
    }

    return classifier::weak(nullopt);
```

Fig. 2.19 MOSEC: SSL Plug-in

2.7.4 MOSEC: Decision Algorithm

As we mentioned, each plug-ins provides its personal flow/packet classification. Wouldn't be more interesting if we were able to provide an ultimate classification for each flow, considering all the suggestions provided by each classifier? MOSEC gives the opportunity to consolidate the classification provided by all plug-ins, in one unique classification. The logic behind the *Decision Algorithm* is customizable as well. You can choice to prefer one plag-in classification among the others, or combine them based on the *weakness* or *strogness* of the classification.

In general, combining the classifications provided by all plug-ins is a difficult task. That is because each plug-in has its own logic and provides the classification according its rules. For example, each plug-in has its own ClassType, which can be an int, a string or an enum. This means that the same flow, i.e. *facebook*, can be represented in different manners, even if all plug-in agrees that flow is a *facebook* flow. For this reason, we introduced the *stringfy* concept. Every plug in has a *stringfy* function which allows to represent the classification as a string. In such a manner, the comparison of the classifications returned by the plug-ins is reduced at nthe comparison of a group of strings, and we can easily exploit the C++ string library.

Current MOSEC version comes with two available strategies: the *all* and *cooperative*. The *all* strategy simply prints all the returned classification, separated by the "pipe" symbol. If one plugin doesn't provide any classification a blank space is added. The *cooperative* strategy implement a simple decision mechanism which prefers HTTP-classification and HTTPS-classification over DNS classification. Then we stacked up all *stringfied* classification in one label, that we use as a reference to retrieved aggregate statistics (see figure 2.20). The label takes the following form:

```
:Port-Based Classification:HTTP/HTTPS/DNS-Based Classification
```

which could be like:

```
:tcp:HTTPS:Facebook
```

```
namespace xmosec { namespace strategy {

    namespace details
    {
        struct naive
        {
            template <typename ...Cs>
            std::string
            operator()(std::tuple<Cs...> const &classifiers,
                       std::tuple<optional<typename Cs::ClassType>...> const &resp) const
            {
                std::string ret; ret.reserve(128);

                if (auto & r = get_class<classifier::ip_protocol, Cs...>(resp))
                    append(ret, std::get<classifier::ip_protocol>(classifiers).stringify(r.value()));

                if (auto & r = get_class<classifier::port, Cs...>(resp))
                    append(ret, " ", std::get<classifier::port>(classifiers).stringify(r.value()));

                if (auto & r = get_class<classifier::ssl, Cs...>(resp))
                    append(ret, " ", std::get<classifier::ssl>(classifiers).stringify(r.value()));
                else
                if (auto & r = get_class<classifier::http<2>, Cs...>(resp))
                    append(ret, " ", std::get<classifier::http<2>>(classifiers).stringify(r.value()));
                else
                if (auto & r = get_class<classifier::dns, Cs...>(resp))
                    append(ret, " ", std::get<classifier::dns>(classifiers).stringify(r.value()));

                return ret;
            }
        };
    }

    constexpr details::naive naive = details::naive{};

} // namespace strategy
} // namespace xmosec
```

Fig. 2.20 MOSEC: Cooperative Strategy

2.7.5 MOSEC: Statistics

MOSEC uses an efficient and fine-grained method to provide per-classifier statistics at flow, packet and byte level. After receiving one packet, the packet and byte number statistics are update for the flow which the packet belongs to. The *update* function in figure 2.21 shows how the process is performed.

The function receives the key, which identifies the flow, and the results provided by all plug-ins (notice that this is a variadic template function). For all the plug-ins, which return a non-nullopt classification, that is a valid classification, the total number of packets and bytes "classified" by that plug-in is updated (the +=*operator* has been overloaded to update both the packet and byte number - look at figure 2.22), as well as the packet and byte number related to that particular label. Indeed, the *stringfied* response is used as a key to address and update the correct entry inside a map defined in the *stats struct*.

```
using UniqueKey = std::pair<Key, size_t>;

stats()
: bytes{}
, packets{}
, counters{}
, flows_ver_{}
{ }


template <typename ...Ts>
void update(optional<Key> const &key, std::tuple<optional<Ts>...> const &res, size_t b)
{
    bytes += b;
    packets++;

    if (key)
    {
        UniqueKey ukey = std::make_pair(key.value(), flow_ver(key.value()));

        tuple_foreach_index([&](auto integral_index, auto &stat_i) {

            auto const & class_i = std::get<decltype(integral_index)::value>(res);
            if (class_i)
            {
                stat_i.packet.total += b;
                stat_i.packet.view[class_i.value()]+= b;

                stat_i.flow.total[ukey] += b;
                stat_i.flow.view[class_i.value()][ukey]+= b;
            }

        }, counters);
    }
}
```

Fig. 2.21 MOSEC: Per-Classifier Statistics (1)

As you notice, also the total number of *unique* flow is updated. To correctly interpreter MOSEC stats, the flow *uniqueness* concept needs to be clarified. MOSEC engine doesn't distinguish between connectionless or connection-oriented protocol. Therefore for connection-oriented flows (e.g. TCP flows), it doesn't look for FIN or RST packets to determine when the connection is closed, but use an internal, configurable timeout. Thus, if no packet are received or transmitted for more than *timeout* seconds the connection is closed and the classification results are dumped to desired output (e.g. the standard output). This choice is motivated by the fact that we are in a mobile environment, where user can camp on other eNodeB due to handover procedure. Thus, it is reasonable to consider a flow closed (from our eNodeB perspective) even if it is still going somewhere else.

Therefore, when one flow is flushed away from the hash table, in the same time the *flow version number* is incremented by one, to infer *uniqueness* to the flow from statistics perspective. When one packet with the same key is received again, a new entry in the hash table is added, the classification process starts from the scratch, and the statistics are update for the flow with the *unique* key, that is a pair of the packet key and *flow version number*.

As the reader probably knows, is very important to provide statistics in intelligible format. When dealing with high amount of traffic and, likely, with thousand of different flow label, which has to be reported to every plug-in, it is easy to loose yourself in a great amount of information, many of those probably refer to punctual information, which do not provide particular insight on user behaviours.

To facilitate the comprehension of the output, and to be able to quickly understand plug-in performance, standard output for plug-in statistics offers an overview of the total number of the packet, byte, and flow classified. Then, the details for each flow label is provided only if certain threshold are exceeded. Is it possible to filter basing on the percentage of bytes *or* the percentage of packets (the percentage is calculated considering the overall traffic statistics) *or* considering a minimum amount of *unique* flow with a certain label.

```
template <typename Tp>
struct unit
{
    unit()
    : value{}
    , bytes{}
    { }

    Tp value;
    Tp bytes;

    inline unit &
    operator+=(Tp b)
    {
        value++;
        bytes += b;
        return *this;
    }
};

//
// per-packet, per-flow counter
//

template <typename Kind, typename Tp>
struct counter
{
    counter()
    : total{}
    , view{}
    { }

    void clear()
    {
        total = Tp{};
        view.clear();
    }

    Tp total;
    std::unordered_map<Kind, Tp> view;
};

//
// classifier's stats
//

template <typename Key, typename Kind>
struct stat
{
    counter<Kind, unit<size_t>> packet;
    counter<Kind, std::unordered_map<Key, unit<size_t>>> flow;

    void clear()
    {
        packet.clear();
        flow.clear();
    }
};
```

Fig. 2.22 MOSEC: Per-Classifier Statistics (2)

2.8 MOSEC Validation

Deep Packet Inspection is the state-of-the-art technology for traffic classification. According to the conventional wisdom, DPI is the most accurate classification technique. Consequently, most popular products, either commercial or open-source, rely on some sort of DPI for traffic classification. However, the actual performance of DPI is still unclear to the research community, since the lack of public datasets prevent the comparison and reproducibility of their results. Here we provide qualitative result about the potentiality and the goodness of our approach. To this aim we use the labeled trace provided by UPC Broadband Communication Group. In [20], Bujlow et al. presents a comprehensive comparison of 6 well-known DPI tools, which are commonly used in the traffic classification literature. Thier study includes 2 commercial products (PACE and NBAR) and 4 open-source tools (OpenDPI, L7-filter, NDPI, and Libprotoident). They studied their performance in various scenarios (including packet and flow truncation) and at different classification levels (application protocol, application and web service). The authors carefully built a labeled dataset with more than 750 K flows, which contains traffic from popular applications. They used the Volunteer-Based System (VBS), developed at Aalborg University, to guarantee the correct labeling of the dataset and released it, including full packet payloads, to the research community. We believe this dataset could become a common benchmark for the comparison and validation of network traffic classifiers.

The dataset used in the paper "Independent Comparison of Popular DPI Tools for Traffic Classification?" consists of 767690 flows, which account for 53.31 GB of pure packet data. The application name was present for 759720 flows (98.96 % of all the flows), which account for 51.93 GB (97.41 %) of the data volume. The remaining flows are unlabeled due to their short lifetime (usually below 1 s), which made VBS incapable to reliably establish the corresponding sockets. The dataset has been artificially built in order to allow its publication with full packet payload. However, the authors claimed to have manually simulated different human behaviours for each application studied in order to make it as representative as possible. The dataset consists of a pcap traces and an INFO file. Each line in the INFO file corresponds to a flow in the pcap trace and is described as follows:

```
flow_id # start_time # end_time # local_ip # remote_ip #"...
local_port # remote_port # transport_protocol # operating_system # ...
process_name # HTTP Url # HTTP Referer #  HTTP Content-type #
```

For your convenience here we provide a summary of the classified flows present in dataset. They provided classification at protocol, application and web service level (figures 2.23,2.24,2.25).

Application Protocol	#Flows	#Megabytes
DNS	18 251	7.66
HTTP	43 127	7 325.44
ICMP	205	2.34
IMAP-STARTTLS	35	36.56
IMAP-TLS	103	410.23
NETBIOS Name Service	10 199	11.13
NETBIOS Session Service	11	0.01
SAMBA Session Service	42 808	450.39
NTP	42 227	6.12
POP3-PLAIN	26	189.25
POP3-TLS	101	147.68
RTMP	378	2 353.67
SMTP-PLAIN	67	62.27
SMTP-TLS	52	3.37
SOCKSv5	1 927	898.31
SHH	38 961	844.87
Webdav	57	59.91

Fig. 2.23 UPC: Application Protocol

Application	#Flows	#Megabytes
4Shared	144	13.39
America's Army	350	61.15
BitTorrent clients (encrypted)	96 399	3 313.98
BitTorrent clients (non-encrypted)	261 527	6 779.95
Dropbox	93	128.66
eDonkey clients (obfuscated)	12 835	8 178.74
eDonkey clients (non-obfuscated)	13 852	8 480.48
Freenet	135	538.28
FTP clients (active)	126	341.17
FTP clients (passive)	122	270.46
iTunes	235	75.4
League of Legends	23	124.14
Pando Media Booster	13 453	13.3
PPlive	1 510	83.86
PPStream	1 141	390.4
RDP Clients	153 837	13 257.65
Skype (all)	2 177	102.99
Skype (audio)	7	4.85
Skype (file transfer)	6	25.74
Skype (video)	7	41.16
Sopcast	424	109.34
Spotify	178	195.15
Steam	1 205	255.84
TOR	185	47.14
World of Warcraft	22	1.98

Fig. 2.24 UPC: Application

To collect and accurately label the flows, the adapted Volunteer-Based System (VBS) developed at Aalborg University. The task of VBS is to collect information about Internet traffic flows (i.e., start time of the flow, number of packets contained by the flow, local and remote IP addresses, local and remote ports, transport layer protocol) together with detailed information about each packet (i.e., direction, size, TCP flags, and relative timestamp to the previous packet in the flow). For each flow, the system also collects the process name associated with that flow. The process name is obtained from the system sockets. This way, we can ensure the application associated to a particular traffic. Additionally, the system collects some information about the HTTP content type (e.g., text/html, video/x-flv). The captured information is transmitted to the VBS server, which stores the data in a MySQL database.

Web Service	#Flows	#Megabytes
4Shared	98	68.42
Amazon	602	51.02
Apple	477	90.22
Ask	171	1.86
Bing	456	36.84
Blogspot	235	10.53
CNN	247	3.66
Craigslist	179	4.09
Cyworld	332	13.06
Doubleclick	1 989	11.24
eBay	281	8.31
FaceBook	6 953	747.35
Go.com	335	25.83
Google	6 541	532.54
Instagram	9	0.22
Justin.tv	2 326	126.33
LinkedIn	62	2.14
Mediafire	472	27.99
MSN	928	23.22
MySpace	2	2.54
Pinterest	189	3.64
Putlocker	103	71.92
QQ.com	753	10.46
Taobao	387	24.29
The Huffington Post	71	21.19
Tumblr	403	52.56
Twitter	1 138	13.67
Vimeo	131	204.45
Vk.com	343	9.59
Wikipedia	6 092	521.95
Windows Live	26	0.16
Wordpress	169	33.31
Yahoo	17 373	937.07
YouTube	2 534	1 891.79

Fig. 2.25 UPC: Web Service

Retaining this method clear and reliable, we decide to integrate the labeled INFO trace in MOSEC package, to offer a benchmarking platform and a tool for immediate checking the goodness of new developed plug-ins or the enhancements of already provisioned ones. Colleagues from UPC, gave us the .pcap trace as well, thus we can verify MOSEC capabilities.

Here we focus on a single performance parameter: the classification accuracy. We acknowledge that other performance parameters are also important, such as speed, scalability, complexity, and robustness. However, given that the research community is mainly using DPI based technique for offline ground-truth generation, we think that those metrics are less decisive.

The comparison between the UPC INFO file and MOSEC results is automatically offered specifying the option –upc from command line. In this way, a map containing the INFO file information is uploaded. Every time one packet is processed, we look up in table if the packet timestamp matches one of the *end_time* taken from the INFO file. If one match is found, that means that that packet is the last one of the relative flow and the classification has to be provided.

We provide two kind of statistics: direct and reverse.

Direct statistics indicates the percentage of our classification which match UPC ones.

- 100% -> every flow that we classified is marked in the same way in the UPC INFO file. *Tagged in the same way* means that same matching or submatching criteria is satisfied.

- 0% -> we got a false positive. None of the recognized flow is tagges in the same way in the UPC INFO file. Notice that sometimes, even if there is no matching, the two classification are almost equivalent (e.g. GoogleVideo on side has no matching with Youtube, but actually they are the same thing)

Reversed statistics indicates the percentage of UPC flow that we marked in the same way. The usage of these two mechanism is necessary to recognize which the weak point of our framework.

Below, the direct statistics follows:

```
UPC direct statistic:
    tcp HTTPS Go: {0 0 0 7 0} 100 %
    tcp HTTP Craigslist: {0 124 0 0 0} 100 %
    tcp HTTP Steamcommunity: {0 0 46 0 0} 100 %
    tcp HTTP Steampowered: {0 0 169 0 0} 100 %
    tcp HTTPS Adadvisor: {8 0 0 0 0} 0 %
    tcp HTTP Skype: {0 19 0 0 0} 100 %
    tcp HTTPS Adnxs: {24 0 0 0 0} 0 %
    tcp HTTPS Windows Azure: {37 0 0 0 0} 0 %
    tcp HTTPS Skype: {0 198 0 0 0} 100 %
    udp Skype: {0 16 0 0 0} 100 %
    tcp POP3-TLS Google|Mail: {0 0 0 12 0} 100 %
    tcp IMAP-TLS Google|Mail: {0 0 0 103 0} 100 %
    tcp HTTP Yieldmanager: {13 0 0 0 0} 0 %
    tcp HTTPS Hotmail: {13 0 0 0 0} 0 %
    tcp HTTP Anyclip: {18 0 0 0 0} 0 %
    tcp HTTP Z5x: {27 0 0 0 0} 0 %
    tcp HTTP Taobao: {19 453 0 0 0} 95.9746 %
    tcp HTTP Tanx: {21 0 0 5 0} 19.2308 %
    tcp HTTP Synacast: {109 0 0 0 0} 0 %
    tcp HTTP Pplive: {1 174 0 0 0} 99.4286 %
    tcp HTTP Qq: {32 0 0 760 0} 95.9596 %
    tcp HTTP Cyworld: {0 280 0 0 0} 100 %
    tcp HTTPS Windows: {6 0 0 0 0} 0 %
    tcp HTTPS Vimeo: {0 40 0 0 0} 100 %
    tcp HTTPS Mydisk: {55 0 0 0 0} 0 %
    tcp HTTP 4shared: {1 135 0 0 0} 99.2647 %
    tcp HTTP Wp: {8 0 0 0 0} 0 %
    tcp HTTP Adfunky: {10 0 0 0 0} 0 %
```

```
tcp HTTP Sport: {20 0 0 0 0} 0 %
tcp HTTPS Linkedin: {0 18 0 0 0} 100 %
tcp HTTPS Edgecast: {11 0 0 0 0} 0 %
tcp HTTPS Apple|iTunes: {0 31 0 0 0} 100 %
tcp HTTPS Apple: {38 2 0 0 0} 5 %
tcp HTTPS Mozilla: {13 0 0 0 0} 0 %
tcp HTTP Iqiyi: {60 0 0 0 0} 0 %
tcp HTTP Battle: {7 0 0 0 0} 0 %
tcp IMAP: {1 0 0 35 0} 97.2222 %
tcp POP3: {1 0 0 26 0} 96.2963 %
tcp HTTPS Tor: {0 87 0 0 0} 100 %
tcp HTTPS Gazeta: {68 0 0 0 0} 0 %
tcp HTTP Scdn: {19 0 0 0 0} 0 %
tcp HTTP Adnxs: {95 0 0 0 0} 0 %
tcp HTTPS Mediafire: {0 328 0 0 0} 100 %
tcp HTTPS Homebroker: {7 0 0 0 0} 0 %
tcp HTTPS Tumblr: {0 11 0 0 0} 100 %
tcp HTTP Ebay|com.au: {0 6 0 0 0} 100 %
tcp HTTPS Msn: {53 1 0 0 0} 1.85185 %
tcp HTTPS Dropbox: {0 90 0 0 0} 100 %
tcp HTTP Pps: {0 0 0 33 0} 100 %
tcp Back Orifice: {46 0 0 0 0} 0 %
udp Computer Aided Design Software Inc: {29 0 0 0 0} 0 %
tcp Sub7's: {33 0 0 0 0} 0 %
tcp BitTorrent: {17 0 0 15018 0} 99.8869 %
tcp HTTP Huffingtonpost: {0 0 0 41 0} 100 %
tcp Nintendo Wi-Fi Connection: {6 0 0 0 0} 0 %
tcp HTTPS Amazon|CloudFront: {7 0 0 0 0} 0 %
tcp HTTPS Scorecardresearch: {29 0 0 0 0} 0 %
tcp FTP: {0 0 0 11 0} 100 %
tcp FTP-active: {0 0 0 58 0} 100 %
tcp HTTPS Akamai: {44 0 0 0 0} 0 %
tcp HTTPS Google|Maps: {4 82 0 0 0} 95.3488 %
tcp HTTPS Google|Mail: {0 12 0 0 0} 100 %
tcp HTTPS Msads: {32 0 0 0 0} 0 %
tcp Microsoft-DS|Samba Aau: {0 0 0 21076 0} 100 %
tcp Webmin: {12 0 0 0 0} 0 %
tcp HTTP Amazon|CloudFront: {10 0 0 0 0} 0 %
udp Battlefield 2 and mods: {8 0 0 0 0} 0 %
```

```
tcp Emule: {36 0 0 487 0} 93.1166 %
tcp HTTP Apple: {173 568 0 0 0} 76.6532 %
tcp HTTP Ebay: {0 48 0 0 0} 100 %
tcp Bittorrent: {36 0 0 110500 0} 99.9674 %
tcp BitTorrent tracker: {0 0 0 36 0} 100 %
tcp Apache Tomcat: {150 0 0 0 0} 0 %
tcp SMTP-TLS: {0 0 0 68 0} 100 %
tcp HTTPS Wikipedia: {0 36 0 0 0} 100 %
tcp DBGp: {32 0 0 0 0} 0 %
tcp HTTP Huffpost: {28 0 0 0 0} 0 %
tcp HTTP eBay: {29 217 0 0 0} 88.2114 %
tcp HTTPS: {238 0 0 0 0} 0 %
tcp HTTP Bing: {3 471 0 0 0} 99.3671 %
udp: {126851 0 0 0 0} 0 %
tcp HTTPS Twitter: {13 246 0 0 0} 94.9807 %
tcp HTTPS Microosft|Live: {35 0 0 8 0} 18.6047 %
udp BitTorrent: {95 0 0 6564 0} 98.5734 %
udp Netbios: {0 0 0 10198 0} 100 %
tcp HTTP Wordpress: {0 169 0 0 0} 100 %
tcp HTTP Justin.tv: {0 0 0 853 0} 100 %
tcp Half-Life: {7 0 0 0 0} 0 %
tcp HTTP Go: {0 0 0 257 0} 100 %
tcp HTTPS Facebook: {196 186 0 0 0} 48.6911 %
tcp HTTP Putlocker: {0 81 0 0 0} 100 %
tcp Edonkey2000: {15 0 0 0 0} 0 %
tcp HTTP Tumblr: {1 390 0 0 0} 99.7442 %
tcp HTTP Wikipedia: {0 6055 0 0 0} 100 %
udp Webmin: {87 0 0 0 0} 0 %
tcp HTTP Facebook: {101 6762 0 0 0} 98.5283 %
udp MS-Streaming: {13 0 0 0 0} 0 %
tcp HTTP Google: {337 4826 0 0 0} 93.4728 %
tcp BEA WebLogic: {551 0 0 0 0} 0 %
tcp HTTP Google|YouTube: {13 2234 0 0 0} 99.4215 %
udp SafetyNET: {58 0 0 0 0} 0 %
tcp HTTPS Aspnetcdn: {31 0 0 0 0} 0 %
udp NTP: {0 42227 0 0 0} 100 %
tcp HTTP Bittorrent: {0 0 0 6 0} 100 %
tcp HTTP Google|Mail: {0 7 0 0 0} 100 %
tcp HTTP Twitter: {0 841 0 0 0} 100 %
```

```
udp H323: {15 0 0 0 0} 0 %
tcp HTTP Pandomediabooster: {0 28 0 0 0} 100 %
tcp HTTP Yahoo: {12 17115 0 0 0} 99.9299 %
tcp HTTP Leagueoflegends: {0 9 0 0 0} 100 %
tcp RDP (Microsoft): {1 153888 0 0 0} 99.9994 %
udp Half-Life: {6 0 0 0 0} 0 %
tcp HTTP Images-amazon: {0 153 0 0 0} 100 %
tcp MySQL Database system: {31 0 0 0 0} 0 %
tcp HTTP Instagram: {0 9 0 0 0} 100 %
tcp HTTP Linkedin: {0 44 0 0 0} 100 %
tcp HTTP Dr: {15 0 0 0 0} 0 %
tcp HTTPS Google|DoubleClick: {3 145 0 0 0} 97.973 %
tcp HTTP Justin: {0 0 0 1297 0} 100 %
tcp HTTPS Google: {163 1478 0 0 0} 90.067 %
udp DNS dns: {28 18250 0 0 0} 99.8468 %
udp DropBox|Sync: {14 0 0 0 0} 0 %
tcp HTTP: {2557 0 0 0 0} 0 %
tcp SSH: {0 36697 0 0 0} 100 %
tcp HTTP Amazon: {1 279 0 0 0} 99.6429 %
tcp HTTP Chango: {56 0 0 0 0} 0 %
tcp HTTP Ask: {0 171 0 0 0} 100 %
tcp HTTP Vimeo: {0 84 0 0 0} 100 %
tcp HTTP Mediafire: {0 145 0 0 0} 100 %
tcp HTTPS Yahoo: {0 67 0 0 0} 100 %
tcp HTTP Pinterest: {0 198 0 0 0} 100 %
tcp POP3-TLS Interia: {0 0 0 89 0} 100 %
udp Call of Duty 2: {13 0 0 0 0} 0 %
tcp VNC: {18 0 0 0 0} 0 %
tcp HTTP Scorecardresearch: {17 0 0 0 0} 0 %
tcp HTTP Cnn: {0 247 0 0 0} 100 %
tcp HTTP Utorrent: {36 0 0 0 0} 0 %
tcp HTTPS Atdmt: {20 0 0 0 0} 0 %
tcp HTTP 9yx: {9 0 0 0 0} 0 %
udp VTun: {6 0 0 0 0} 0 %
tcp RTMP Yahoo: {0 15 0 0 0} 100 %
tcp HTTPS Amazon: {3 148 0 0 0} 98.0132 %
udp Aleph One: {11 0 0 0 0} 0 %
udp BitTorrent Bittorrent: {0 0 0 8 0} 100 %
tcp HTTP Pptv: {280 0 0 0 0} 0 %
```

```
tcp SafetyNET: {601 0 0 0 0} 0 %
tcp WebCT e-learning portal: {0 0 0 63 0} 100 %
tcp Tuxanci game: {255 0 0 0 0} 0 %
tcp Microsoft DCOM services: {134 0 0 0 0} 0 %
tcp SHOUTcast: {78 0 0 0 0} 0 %
tcp HTTP Craigslist|com.tr: {0 55 0 0 0} 100 %
tcp psyBNC: {23 0 0 0 0} 0 %
tcp HTTPS Wordpress: {0 8 0 0 0} 100 %
tcp HTTP Microosft|Live: {0 0 0 19 0} 100 %
udp NetBus: {13 0 0 0 0} 0 %
tcp HTTPS Taobao: {0 7 0 0 0} 100 %
tcp HTTP S-msn: {0 0 0 428 0} 100 %
tcp HTTP Cyworld|co.kr: {0 45 0 0 0} 100 %
tcp PPTP: {6 0 0 0 0} 0 %
tcp HTTP Cdn-apple: {0 6 0 0 0} 100 %
tcp SMTP-TLS Google|Mail: {0 0 0 50 0} 100 %
tcp MS-Streaming: {8 0 0 0 0} 0 %
tcp NetBus: {31 0 0 0 0} 0 %
tcp HTTP Msn: {1 494 0 0 0} 99.798 %
tcp FileMakerPro: {24 0 0 0 0} 0 %
tcp Tor: {0 62 0 0 0} 100 %
tcp HTTP Blogspot: {31 179 0 0 0} 85.2381 %
tcp X11: {6 0 0 0 0} 0 %
udp Gnutella-svc: {35 0 0 0 0} 0 %
udp Cisco HSRP: {122 0 0 0 0} 0 %
tcp HTTPS 4shared: {0 109 0 0 0} 100 %
tcp HTTP Google|DoubleClick: {17 2479 0 0 0} 99.3189 %
tcp Netbios: {0 0 0 11 0} 100 %
udp VLC: {12 0 0 0 0} 0 %
tcp: {136381 0 0 0 0} 0 %
udp Xbox: {10 0 0 0 0} 0 %
tcp Microsoft-DS|Samba: {1 0 0 21732 0} 99.9954 %
tcp HTTP Vk: {0 0 0 343 0} 100 %
tcp Symantec VOPIED (VERITAS): {31 0 0 0 0} 0 %
udp Nintendo Wi-Fi Connection: {12 0 0 0 0} 0 %
tcp HTTP 2nike: {6 0 0 0 0} 0 %
udp SSMP Message: {7 0 0 0 0} 0 %
tcp Freeciv multiplay: {29 0 0 0 0} 0 %
tcp Winbox: {9 0 0 0 0} 0 %
```

```
    tcp HTTP Akamai: {84 0 0 0 0} 0 %
    udp Zabbix-Server: {9 0 0 0 0} 0 %
    udp Stun: {78 0 0 0 0} 0 %
    tcp RuneScape: {26 0 0 0 0} 0 %
    tcp HTTP Putlockerdownloader: {0 0 6 0 0} 100 %
    udp Unreal-turn: {74 0 0 0 0} 0 %
    tcp HTTP Adform: {35 0 0 0 0} 0 %
    tcp FTP-active Webd: {0 0 0 62 0} 100 %
    udp Octopus Multiplexer: {11 0 0 0 0} 0 %
    tcp HTTPS Google|YouTube: {3 96 0 0 0} 96.9697 %
    tcp Usermin: {24 0 0 0 0} 0 %
    udp Emule: {0 0 0 4240 0} 100 %
    tcp RTMP: {87 366 0 0 0} 80.7947 %
    tcp HTTP Google|Maps: {147 85 0 0 0} 36.6379 %
    tcp Skype: {0 28 0 0 0} 100 %
    udp cft-0: {12 0 0 0 0} 0 %
    tcp IRC: {1981 0 0 0 0} 0 %
    udp BMC Software: {8 0 0 0 0} 0 %

total_packets: 64308125, elapsed time: 227:834215 (sec:usec), rate: 0.282258 Mpps (2.00773 Gbp
```

These results show that MOSEC works very well with TCP/HTTP, whereas has some difficulties when dealing with UDP traffic. This is an expected result. As you probably noticed, most of the plug-in that we developed works specifically with HTTP/HTTPS information, while we did not provide such drill down for UDP traffic. Indeed, we're able to classify UDP traffic only exploiting the port-based algorithm or, when it is possible, the DNS classifier. As you can see, for UDP flows we experienced many times false positive matching. This is due because UDP ports are often used randomly by upper layer applications. Therefore, without an adequate deep packet inspection also for UDP traffic, we wrongly assumed that that traffic is generated by the application that should use that *not-so-well-known* port.

Reverse statistics (not shown here) highlight the same weakness.

2.9 Traffic Analysis with MOSEC

As an example, in this section we provided classification results obtained with MOSEC, when we analyze one the .pcap trace captured on the S1-U interface of our eNodeB. In

particular, this analysis refers to one day taken from dataset IV. We exploited the *Basic* configuration with five classifiers, namely the *IP-Protocol*, *Port-based*, the *DNS-based* and the *HTTP-based-Mode0/1/2*, and the *SSL-based* classifier. An additional module is added for dubugging and is named *Debug Classifier*.

```
Stats summary:
  bytes        : 17544175555
  packets      : 32434526
  flows        : 343379
classifier 'IP-protocol'
  total:
      bytes  : 17544175555 (100 %)
      packets: 32434526 (100 %)
      flows  : 343376 (99.9991 %)
  class packets:
    sctp    -> {    packets: 9428157 (29.0683 % 29.0683 %) -
                    bytes: 1147506454 (6.54067 % 6.54067 %) -
                    flows: 75 (0.0218418 % 0.0218419 %) }
    tcp     -> {    packets: 21905347 (67.5371 % 67.5371 %) -
                    bytes: 15841524137 (90.2951 % 90.2951 %) -
                    flows: 225854 (65.774 % 65.7745 %) }
    udp     -> {    packets: 1101022 (3.3946 % 3.3946 %) -
                    bytes: 555144964 (3.16427 % 3.16427 %) -
                    flows: 117447 (34.2033 % 34.2036 %) }
classifier 'Port'
  total:
      bytes  : 15842846753 (90.3026 %)
      packets: 22020238 (67.8914 %)
      flows  : 322188 (93.8287 %)
  class packets:
    Imaps   -> {    packets: 494070 (1.52328 % 2.24371 %) -
                    bytes: 269668088 (1.53708 % 1.70214 %) -
                    flows: 3554 (1.03501 % 1.10308 %) }
    HTTP    -> {    packets: 10469874 (32.28 % 47.5466 %) -
                    bytes: 8559944283 (48.7908 % 54.0303 %) -
                    flows: 99699 (29.0347 % 30.9444 %) }
    HTTPS   -> {    packets: 9838526 (30.3335 % 44.6795 %) -
                    bytes: 6649191129 (37.8997 % 41.9697 %) -
```

```
                    flows: 99131 (28.8693 % 30.7681 %) }
    others  -> {    packets: 1217768 (3.75454 %) -
                    bytes: 364043253 (2.07501 %) -
                    flows: 119804 (34.8897 %) }

classifier 'dns'
  total:
      bytes  : 15247166370 (86.9073 %)
      packets: 21246011 (65.5043 %)
      flows  : 304927 (88.8019 %)
  class packets:
    Ororo               -> {    packets: 587541 (1.81147 % 2.76542 %) -
                                bytes: 621713435 (3.5437 % 4.07757 %) -
                                flows: 100 (0.0291223 % 0.0327947 %) }
    Ticdn               -> {    packets: 2123705 (6.54767 % 9.99578 %) -
                                bytes: 2079022727 (11.8502 % 13.6355 %) -
                                flows: 156 (0.0454309 % 0.0511598 %) }
    Amazon CloudFront   -> {    packets: 231038 (0.712321 % 1.08744 %) -
                                bytes: 194645286 (1.10946 % 1.2766 %) -
                                flows: 1021 (0.297339 % 0.334834 %) }
    Rncdn3              -> {    packets: 204480 (0.630439 % 0.962439 %) -
                                bytes: 164831618 (0.939523 % 1.08106 %) -
                                flows: 86 (0.0250452 % 0.0282035 %) }
    Xvideos             -> {    packets: 762841 (2.35194 % 3.59051 %) -
                                bytes: 606357042 (3.45617 % 3.97685 %) -
                                flows: 1372 (0.399559 % 0.449944 %) }
    Apple               -> {    packets: 1167418 (3.59931 % 5.49477 %) -
                                bytes: 882361719 (5.02937 % 5.78705 %) -
                                flows: 12732 (3.70786 % 4.17543 %) }
    Google|Mail         -> {    packets: 409728 (1.26325 % 1.9285 %) -
                                bytes: 237376965 (1.35302 % 1.55686 %) -
                                flows: 2192 (0.638362 % 0.718861 %) }
    Hotmail             -> {    packets: 418929 (1.29161 % 1.9718 %) -
                                bytes: 141832728 (0.808432 % 0.930224 %) -
                                flows: 2001 (0.582738 % 0.656223 %) }
    Google              -> {    packets: 4687307 (14.4516 % 22.0621 %) -
                                bytes: 3868824839 (22.0519 % 25.3741 %) -
                                flows: 26393 (7.68626 % 8.65551 %) }
    Instagram           -> {    packets: 338347 (1.04317 % 1.59252 %) -
```

```
                                bytes: 225723174 (1.2866 % 1.48043 %) -
                                flows: 3113 (0.906578 % 1.0209 %) }
    Whatsapp            -> {    packets: 394768 (1.21712 % 1.85808 %) -
                                bytes: 207278207 (1.18146 % 1.35945 %) -
                                flows: 3568 (1.03909 % 1.17012 %) }
    Facebook            -> {    packets: 4843788 (14.934 % 22.7986 %) -
                                bytes: 3368801731 (19.2018 % 22.0946 %) -
                                flows: 30245 (8.80805 % 9.91877 %) }
    Akamai              -> {    packets: 282132 (0.869851 % 1.32793 %) -
                                bytes: 201376764 (1.14783 % 1.32075 %) -
                                flows: 3295 (0.959581 % 1.08059 %) }
    others              -> {    packets: 4793987 (14.7805 %) -
                                bytes: 2447020135 (13.9478 %) -
                                flows: 218653 (63.6769 %) }

classifier 'http[host]'
  total:
      bytes  : 8433384547 (48.0694 %)
      packets: 10060924 (31.0192 %)
      flows  : 64242 (18.7088 %)
  class packets:
    Xvideos             -> {    packets: 761812 (2.34877 % 7.57199 %) -
                                bytes: 605570342 (3.45169 % 7.18063 %) -
                                flows: 1354 (0.394316 % 2.10766 %) }
    Instagram           -> {    packets: 344406 (1.06185 % 3.4232 %) -
                                bytes: 232816445 (1.32703 % 2.76065 %) -
                                flows: 3016 (0.87833 % 4.69475 %) }
    Amazon CloudFront   -> {    packets: 162302 (0.500399 % 1.61319 %) -
                                bytes: 140448847 (0.800544 % 1.66539 %) -
                                flows: 526 (0.153184 % 0.818779 %) }
    Ororo               -> {    packets: 587535 (1.81145 % 5.83977 %) -
                                bytes: 621713035 (3.5437 % 7.37205 %) -
                                flows: 98 (0.0285399 % 0.152548 %) }
    Google              -> {    packets: 2246290 (6.92561 % 22.3269 %) -
                                bytes: 1899938242 (10.8295 % 22.5288 %) -
                                flows: 7706 (2.24417 % 11.9953 %) }
    Apple               -> {    packets: 854789 (2.63543 % 8.49613 %) -
                                bytes: 782780907 (4.46177 % 9.28193 %) -
                                flows: 2165 (0.630499 % 3.37007 %) }
```

```
    Ticdn              -> {     packets: 2238023 (6.90013 % 22.2447 %) -
                                bytes: 2202481580 (12.5539 % 26.1162 %) -
                                flows: 157 (0.0457221 % 0.244388 %) }
    Rncdn3             -> {     packets: 213259 (0.657506 % 2.11968 %) -
                                bytes: 171309072 (0.976444 % 2.03132 %) -
                                flows: 91 (0.0265013 % 0.141652 %) }
    others             -> {     packets: 2652508 (8.17804 %) -
                                bytes: 1776326077 (10.1249 %) -
                                flows: 49129 (14.3075 %) }

classifier 'http[content-type]'
  total:
      bytes  : 8417006546 (47.9761 %)
      packets: 9986040 (30.7883 %)
      flows  : 58763 (17.1132 %)
  class packets:
    image/png                -> {      packets: 238963 (0.736755 % 2.39297 %) -
                                       bytes: 166472189 (0.948874 % 1.97781 %) -
                                       flows: 3088 (0.899298 % 5.25501 %) }
    application/json         -> {      packets: 210958 (0.650412 % 2.11253 %) -
                                       bytes: 110076020 (0.627422 % 1.30778 %) -
                                       flows: 6670 (1.94246 % 11.3507 %) }
    image/jpeg               -> {      packets: 1016636 (3.13443 % 10.1806 %) -
                                       bytes: 686773701 (3.91454 % 8.15936 %) -
                                       flows: 11260 (3.27918 % 19.1617 %) }
    video/mp4                -> {      packets: 3293270 (10.1536 % 32.9787 %) -
                                       bytes: 2984535822 (17.0115 % 35.4584 %) -
                                       flows: 1219 (0.355001 % 2.07443 %) }
    image/gif                -> {      packets: 204172 (0.62949 % 2.04457 %) -
                                       bytes: 104738979 (0.597001 % 1.24437 %) -
                                       flows: 7185 (2.09244 % 12.2271 %) }
    text/html                -> {      packets: 321249 (0.990454 % 3.21698 %) -
                                       bytes: 152012816 (0.866457 % 1.80602 %) -
                                       flows: 12128 (3.53196 % 20.6388 %) }
    text/plain               -> {      packets: 807299 (2.48901 % 8.08428 %) -
                                       bytes: 740486565 (4.2207 % 8.7975 %) -
                                       flows: 2792 (0.813096 % 4.75129 %) }
    application/octet-stream -> {      packets: 1400461 (4.31781 % 14.0242 %) -
                                       bytes: 1250574341 (7.12815 % 14.8577 %) -
```

```
                                        flows: 3582 (1.04316 % 6.09567 %) }
    others                    -> {      packets: 2493032 (7.68635 %) -
                                        bytes: 2221336113 (12.6614 %) -
                                        flows: 10839 (3.15657 %) }

classifier 'http[host|content-type]'
  total:
      bytes  : 8433384547 (48.0694 %)
      packets: 10060924 (31.0192 %)
      flows  : 64242 (18.7088 %)
  class packets:
    Xvideos|video/mp4               -> {    packets: 735377 (2.26727 % 7.30924 %) -
                                            bytes: 593223357 (3.38131 % 7.03423 %) -
                                            flows: 321 (0.0934827 % 0.499673 %) }
    Ticdn|video/mp4                 -> {    packets: 350981 (1.08212 % 3.48856 %) -
                                            bytes: 394009620 (2.24581 % 4.67202 %) -
                                            flows: 124 (0.0361117 % 0.19302 %) }
    Instagram|image/jpeg            -> {    packets: 173175 (0.533922 % 1.72126 %) -
                                            bytes: 110226965 (0.628282 % 1.30703 %) -
                                            flows: 2393 (0.696898 % 3.72498 %) }
    Rncdn3|video/mp4                -> {    packets: 213259 (0.657506 % 2.11968 %) -
                                            bytes: 171309072 (0.976444 % 2.03132 %) -
                                            flows: 91 (0.0265013 % 0.141652 %) }
    Google|video/mp4                -> {    packets: 870878 (2.68503 % 8.65604 %) -
                                            bytes: 729709069 (4.15927 % 8.65262 %) -
                                            flows: 182 (0.0530027 % 0.283304 %) }
    Ororo|video/mp4                 -> {    packets: 582613 (1.79627 % 5.79085 %) -
                                            bytes: 616734926 (3.51533 % 7.31302 %) -
                                            flows: 81 (0.0235891 % 0.126086 %) }
    Google|application/octet-stream -> {    packets: 1151780 (3.55109 % 11.4481 %) -
                                            bytes: 1033915412 (5.89321 % 12.2598 %) -
                                            flows: 1602 (0.46654 % 2.4937 %) }
    Apple|text/plain                -> {    packets: 740996 (2.28459 % 7.36508 %) -
                                            bytes: 709381299 (4.0434 % 8.41158 %) -
                                            flows: 62 (0.0180559 % 0.0965101 %) }
    others                          -> {    packets: 5241866 (16.1614 %) -
                                            bytes: 4074874827 (23.2264 %) -
                                            flows: 59386 (17.2946 %) }
```

```
classifier 'SSL'
  total:
      bytes  : 6815864957 (38.8497 %)
      packets: 9998067 (30.8254 %)
      flows  : 74147 (21.5933 %)
  class packets:
    Whatsapp    -> {    packets: 249006 (0.767719 % 2.49054 %) -
                        bytes: 219809305 (1.25289 % 3.22497 %) -
                        flows: 559 (0.162794 % 0.753908 %) }
    Hotmail     -> {    packets: 435994 (1.34423 % 4.36078 %) -
                        bytes: 146924452 (0.837454 % 2.15562 %) -
                        flows: 1908 (0.555654 % 2.57327 %) }
    Facebook    -> {    packets: 4450805 (13.7224 % 44.5167 %) -
                        bytes: 3111073319 (17.7328 % 45.6446 %) -
                        flows: 21330 (6.2118 % 28.7672 %) }
    Google|Mail -> {    packets: 391147 (1.20596 % 3.91223 %) -
                        bytes: 232892970 (1.32747 % 3.41692 %) -
                        flows: 1733 (0.50469 % 2.33725 %) }
    Google      -> {    packets: 2465442 (7.60129 % 24.6592 %) -
                        bytes: 2014508036 (11.4825 % 29.5562 %) -
                        flows: 13061 (3.80367 % 17.615 %) }
    Akamai      -> {    packets: 531443 (1.63851 % 5.31546 %) -
                        bytes: 367322151 (2.0937 % 5.38922 %) -
                        flows: 5113 (1.48903 % 6.89576 %) }
    others      -> {    packets: 1474230 (4.54525 %) -
                        bytes: 723334724 (4.12293 %) -
                        flows: 30443 (8.86571 %) }

Stats summary:
  bytes        : 17544175555
  packets      : 32434526
  flows        : 343379
classifier 'aggregate'
  total:
      bytes  : 17544175555 (100 %)
      packets: 32434526 (100 %)
      flows  : 343379 (100 %)
  class packets:
    sctp                      -> {     packets: 9428157 (29.0683 % 29.0683 %) -
```

```
                                    bytes: 1147506454 (6.54067 % 6.54067 %) -
                                    flows: 75 (0.0218418 % 0.0218418 %) }
tcp HTTP Ororo              -> {    packets: 587541 (1.81147 % 1.81147 %) -
                                    bytes: 621713435 (3.5437 % 3.5437 %) -
                                    flows: 100 (0.0291223 % 0.0291223 %) }
tcp HTTPS Akamai            -> {    packets: 531076 (1.63738 % 1.63738 %) -
                                    bytes: 366936895 (2.0915 % 2.0915 %) -
                                    flows: 5251 (1.52921 % 1.52921 %) }
udp                         -> {    packets: 816396 (2.51706 % 2.51706 %) -
                                    bytes: 519993244 (2.96391 % 2.96391 %) -
                                    flows: 9853 (2.86942 % 2.86942 %) }
tcp HTTP Rncdn3             -> {    packets: 213259 (0.657506 % 0.657506 %) -
                                    bytes: 171309072 (0.976444 % 0.976444 %) -
                                    flows: 91 (0.0265013 % 0.0265013 %) }
tcp HTTP Ticdn              -> {    packets: 2238357 (6.90116 % 6.90116 %) -
                                    bytes: 2202672524 (12.555 % 12.555 %) -
                                    flows: 163 (0.0474694 % 0.0474694 %) }
tcp HTTPS Google            -> {    packets: 2448187 (7.54809 % 7.54809 %) -
                                    bytes: 2012535286 (11.4712 % 11.4712 %) -
                                    flows: 15835 (4.61152 % 4.61152 %) }
tcp HTTPS Facebook          -> {    packets: 4494118 (13.856 % 13.856 %) -
                                    bytes: 3116983861 (17.7665 % 17.7665 %) -
                                    flows: 26581 (7.74101 % 7.74101 %) }
tcp HTTPS Hotmail           -> {    packets: 423923 (1.30701 % 1.30701 %) -
                                    bytes: 143538676 (0.818156 % 0.818156 %) -
                                    flows: 2012 (0.585941 % 0.585941 %) }
tcp Imaps Google|Mail       -> {    packets: 386263 (1.1909 % 1.1909 %) -
                                    bytes: 228909891 (1.30476 % 1.30476 %) -
                                    flows: 1873 (0.545461 % 0.545461 %) }
tcp HTTPS Whatsapp          -> {    packets: 340375 (1.04942 % 1.04942 %) -
                                    bytes: 232031756 (1.32256 % 1.32256 %) -
                                    flows: 2155 (0.627586 % 0.627586 %) }
tcp HTTP Apple              -> {    packets: 856194 (2.63976 % 2.63976 %) -
                                    bytes: 783258583 (4.46449 % 4.46449 %) -
                                    flows: 2330 (0.678551 % 0.678551 %) }
tcp HTTP Xvideos            -> {    packets: 763591 (2.35425 % 2.35425 %) -
                                    bytes: 606722421 (3.45826 % 3.45826 %) -
                                    flows: 1384 (0.403053 % 0.403053 %) }
tcp HTTP Google             -> {    packets: 2251615 (6.94203 % 6.94203 %) -
```

```
                                        bytes: 1900344654 (10.8318 % 10.8318 %) -
                                        flows: 15835 (4.61152 % 4.61152 %) }
tcp HTTPS Facebook          -> {        packets: 4494118 (13.856 % 13.856 %) -
                                        bytes: 3116983861 (17.7665 % 17.7665 %) -
                                        flows: 26581 (7.74101 % 7.74101 %) }
tcp HTTPS Hotmail           -> {        packets: 423923 (1.30701 % 1.30701 %) -
                                        bytes: 143538676 (0.818156 % 0.818156 %) -
                                        flows: 2012 (0.585941 % 0.585941 %) }
tcp Imaps Google|Mail       -> {        packets: 386263 (1.1909 % 1.1909 %) -
                                        bytes: 228909891 (1.30476 % 1.30476 %) -
                                        flows: 1873 (0.545461 % 0.545461 %) }
tcp HTTPS Whatsapp          -> {        packets: 340375 (1.04942 % 1.04942 %) -
                                        bytes: 232031756 (1.32256 % 1.32256 %) -
                                        flows: 2155 (0.627586 % 0.627586 %) }
tcp HTTP Apple              -> {        packets: 856194 (2.63976 % 2.63976 %) -
                                        bytes: 783258583 (4.46449 % 4.46449 %) -
                                        flows: 2330 (0.678551 % 0.678551 %) }
tcp HTTP Xvideos            -> {        packets: 763591 (2.35425 % 2.35425 %) -
                                        bytes: 606722421 (3.45826 % 3.45826 %) -
                                        flows: 1384 (0.403053 % 0.403053 %) }
tcp HTTP Google             -> {        packets: 2251615 (6.94203 % 6.94203 %) -
                                        bytes: 1900344654 (10.8318 % 10.8318 %) -
                                        flows: 8706 (2.53539 % 2.53539 %) }
tcp HTTP Amazon CloudFront -> {         packets: 179774 (0.554267 % 0.554267 %) -
                                        bytes: 152466041 (0.869041 % 0.869041 %) -
                                        flows: 611 (0.177937 % 0.177937 %) }
tcp HTTP Instagram          -> {        packets: 345146 (1.06413 % 1.06413 %) -
                                        bytes: 233102795 (1.32866 % 1.32866 %) -
                                        flows: 3130 (0.911529 % 0.911529 %) }
others                      -> {        packets: 6130554 (18.9013 %) -
                                        bytes: 3104149967 (17.6933 %) -
                                        flows: 263229 (76.6584 %) }
```

2.10 Conclusion

This chapter reports on reports on the traffic measurements carried out at an eNodeB of Telecom Italia located in a business area in Turin. Through subsequent measurement campaigns, it was possible to follow the evolution of the 4G network by the beginning of its deployment

in 2012, until its full maturity in 2015. The data collected during the first year, showed a poor use of the LTE network, mainly due to the limited penetration of new 4G smartphone. In 2015, however, we appreciate a clear and decisive increase in the number of terminals using LTE, with aggregate statistics (e.g. marketshare for smartphone operating systems, or the percentage of video traffic) that reflect the national trend. This important outcome testifies the maturity of LTE technology, and allows us to consider our monitored eNodeB as a valuable vantage point for traffic analysis.

A further analysis is aimed at evaluating the traffic volume of different applications/services. We select the four ones with the largest number of TCP connections. These are Facebook, Google, Apple, Whatsapp, and Mail applications. Facebook always accounts for the highest number of flows, this is not surprising given that Facebook is among the world's most popular social networking sites, We notice a negative trend for Apple and a positive trend for Google, as effect of the the higher number of Android devices in 2014 and 2015. The number of flows which belongs to Whatsapp and Youtube is stable across all the datasets, and is equal to 2-3% and 0.5%, respectively. We evaluate the aggregate amount of traffic associated to each applications, as well. Facebook represents more than the 20% of the traffic Although counts few number of flows, Youtube is responsible, on average, of the 20% of all the traffic generated. Google and Apple give approximately the same contribution in 2015, that is slightly more than 7%. Finally, Whatsapp contribution varies little across all the years to stabilize around 2% in 2015.

In addition, in this chapter we present a new framework/tool exclusively dedicated to the topic of traffic classification. Among the plethora of existing tool for traffic classification we provide our own solution, developed from scratch. The project, which is available on github, is named MOSEC, an acronym for Modular SErvice Classifier. The modularity is given by the possibility to implement multiple plug-ins, each one will process the packet according to its logic, and may or may not return a packet/flow classification. A final decision strategy allows to classify the various streams, based on the classifications of each plug-in. Despite previous approaches, the ability of keeping together multiple classifiers allows to mitigate the deficiency of each classifiers (e.g. DPI does not work when packets are encrypted or DNS queries don't have to be sent if name resolution is cached in device memory) and exploit their full-capabilities when it is feasible. We validated the goodness of MOSEC using a labelled trace synthetically created by colleagues from UPC BarcelonaTech. The results show excellent TCP-HTTP/HTTPS traffic classification capabilities, higher, on average, than those of other classification tools (NDPI, PACE, Layer-7). On the other hand, there are some shortcomings with regard to the classification of UDP traffic.

Chapter 3

Energy Consumption

3.1 Introduction

The evolution and growth of mobile network is fundamentally changing the way users access Internet and consume content and services. Mobile phones have had a surprising evolution over the last two decades, starting from simple devices with only voice services towards smartphones offering novel services such as mobile Internet, localization and maps, multimedia services, and many more. The new wireless interface introduced in mobile networks, i.e. the Long Term Evolution (LTE) technology, fulfills the demand for high data rate and meets user expectation. Simultaneously, the spread of mobile Internet access introduces new energy-related issues to consider.

From the network side, the eNodeB is the main energy hungry element of the radio access network. Most of the power consumed by the eNodeB is due to the base band unit, the power amplifier and the cooling system. Many studies have been focused on the design of techniques for reducing power consumption in the radio access network. These studies consider strategies for the energy-efficient resource allocation, for the carrier aggregation or for the simply switching on/off of network elements depending on their load ([19], [50]).

From the users side, the battery lifetime represents the main limitation on smartphone usage. To achieve high data rates, higher order modulations (e.g. 64-QAM), advanced coding and antenna techniques must be used. As a result, newer smartphones need complex circuitry that quickly consumes User Equipment (UE) battery. To cope with this issue, LTE employs different mechanisms to save energy. The main one is Discontinuous Reception (DRX) that allows UE to power down most of its circuitry when no data needs to be sent/received.

Huang et al. [34] present an analysis of the power characteristics in LTE systems. They developed an empirically derived power model of a commercial LTE network, inferring network parameters through a publicly deployed tool called 4G Test. By applying the power

model to a comprehensive dataset, they identify that the smartphone energy consumption is significantly influenced by the RRC inactivity timer. Deng et al. [28] measured the power consumption and inactivity timer values using the Monsoon Power Monitor. Based on the obtained results, they provide a simplified energy consumption model. To reduce 3G/4G energy consumption, a control module can predict when to put the radio into its Idle state, and when to move from Idle to Active state. The test of the proposed solution required to modify the socket layer and to add a control module inside the Android OS source code.

Aucinas et al. [16] studied the energy and network costs of mobile application that are continuously attached to the network, revealing the effect of such always-on application on smartphone battery lifetime. Qian et al. [44] propose a novel TCP extention called STC (Silent TCP connection Closure) to close TCP connection silently without exchanging SYN or RST packet between endpoints, and quantify the saved smartphone energy. However, the proposed solution requires the modification of the smartphone operating system and the introduction of a middle box in the radio access network. Stea et al. [49] investigate how to configure DRX parameters and explores the trade-off between energy saving and per-user QoS for different services.

J. Fan et al. proposed to use the so-called RRC semi-connected state to reduce simultaneously the signalling load and the energy consumed by the UE. As a general principle on RRC connected to idle transition the eNodeB shall delete all the keys associated to the access stratum, needed for RRC protection. The authors proposed that when UE returns to idle state, the generated RRC protection keys are kept both in the network and the UE. UE is then in RRC semi-connected state which resembles both the idle and connected state. In this state the UE can still listen paging message but all other AS functionality can be stopped or semi-stopped. When a paging message comes, the device wakes up, triggering the transition to connected state. Thus, the device can return to a communication state without necessarily requiring RRC connection setup procedures. As a results, Uu and S1 signalling overheads decreased and, consequently, UE power consumption will be relatively reduced.

With respect to these previous works, the main contribution of this paper is to provide an analysis aimed at defining how to properly set the RRC Inactivity Timer to achieve a trade-off between energy consumption and traffic overhead on the control plane. The main novelty of the presented analysis is that the key parameters used inside the energy consumption model are inferred directly from passive measurements carried out by monitoring a commercial eNodeB of one of the Italian Mobile Operators. Furthermore, in the presented analysis, we account for the traffic overhead on the control plane, needed to establish/remove one RRC connection, and for other procedures, such as handover.

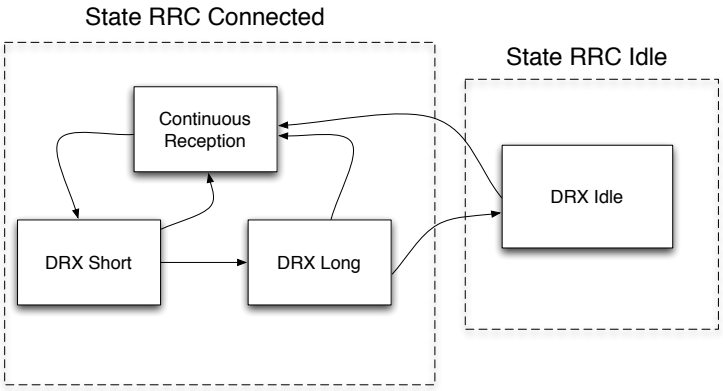

Fig. 3.1 The RRC state machine

3.2 Discontinuous Reception – DRX

To reduce UE energy consumption, the existing wireless mobile networks adopt the DRX mechanism, exhaustively explained in [18]. The DRX mechanism is an effective way to reduce the UE's battery power usage, but it simultaneously introduces a cost in terms of signalling load and an average increase of packets delivery delays. In LTE systems, DRX is configured at a per-UE basis and controlled by a list of parameters. From the RRC perspective, a UE can be found in two different states, depending on whether radio resources are assigned to it or not. The two states are denoted as RRC Connected and RRC Idle. In both states, DRX is supported. Figure 3.1 shows the RRC states and the permitted transitions.

In the RRC Connected state, the UE can be in one of the three modes: Continuous Reception, Short DRX, and Long DRX. On the contrary, in the RRC Idle state, UE is only in DRX mode. When a UE in the RRC Idle receives/sends a packet, a state transition from RRC Idle to RRC Connected is completed in an almost constant time interval, denoted as T_{Pro}, LTE promotion time. During T_{Pro}, radio resources are allocated to the UE. By default, the UE switched to the RRC Connected state enters the Continuous Reception mode and monitors the Physical Downlink Control Channel (PDCCH) to identify DL data. The UE also starts the DRX Inactivity timer, which is reset every time the UE receives/sends a packet. When the DRX Inactivity timer expires, the UE enters Short DRX mode. Furthermore, if no data are transmitted/received during the Short DRX cycle, the UE enters Long DRX mode.

A DRX cycle includes an "On Duration Time" during which the UE monitors PDCCH, and an "Off Time" to save energy. The trade–off between battery saving and latency is the guideline for tuning the parameters of DRX cycle. Once the "On Duration Time" is fixed, the increase of the DRX cycle reduces energy consumption at the expense of a growth of the network latency perceived by the user.

Every time a packet is sent/received, the UE goes back to the Continuous Reception mode. The DRX Inactivity timer and the RRC Inactivity timer are reset. When the RRC Inactivity timer expires, the UE switches from the RRC Connected state to the RRC Idle state, and the RRC connection state is released.

3.3 Energy Consumption Model

The considered energy consumption model is a simplified version of the model presented in [34]. In particular, with respect to the original model, we assume that i) no energy is consumed during RRC Idle state, and ii) the power is considered constant and equal to P_{DRX}, when the UE is in one of the two DRX modes.

Taking into account the RRC state machine, the different terms of the energy consumption model can be calculated as follows. The first term is related to the case of a packet transmission/reception event when the UE is in the RRC Idle state (assume that this happens at time t_0). In this case, the necessary energy is equal to:

$$E_{t_0} = T_{Pro} * P_{Pro} \tag{3.1}$$

where T_{Pro} and P_{Pro} are the time and power needed to switch the UE to the RRC Connected state, respectively. In case a new packet has to be received/sent at time t_1, two alternatives are possible. If the elapsed time between this packet and previous one is less than DRX Inactivity Timer (DRX_{IT}), the required energy is equal to

$$E_{t_1} = P_N * (t_1 - t_0) \tag{3.2}$$

where $P_N = P_{TX}$ if we have a packet transmission or $P_N = P_{RX}$ if we have a reception. Otherwise, the energy can be calculated as

$$E_{t_1} = P_N * DRX_{IT} + P_{DRX} * (t_1 - t_0 - DRX_{IT}). \tag{3.3}$$

As last case, if a packet transmission/reception event occurs at a time t_2 – when the RRC Inactivity Timer (RRC_{IT}) has expired – we have to take into account that the UE must switch from the RRC Idle state. Then the consumed energy is equal to

$$E_{t_2} = P_N * DRX_{IT} + P_{DRX} * RRC_{IT} + T_{Pro} * P_{Pro} \tag{3.4}$$

Depending on the evolution of the RRC state of the UE on a per packet basis, the overall energy, E, necessary to a UE for its traffic activity can be computed as

$$E = \sum_{k=1}^{N} E_{t_k} \tag{3.5}$$

Based on the model presented in [34], the power needed for a transmission and a reception (P_{TX} and P_{RX}, respectively) depends on the throughput and can be evaluated as:

$$\begin{aligned} P_{TX} &= \alpha_{ul} * T_{ul} + \beta \\ P_{RX} &= \alpha_{dl} * T_{dl} + \beta \end{aligned} \tag{3.6}$$

where T_{ul} and T_{dl} represent the uplink and the downlink throughput, respectively. The parameters α_{dl}, α_{ul} and β can be extracted from the linear model which, as shown in [34], fits well both the uplink and the downlink relations for the P_{TX} and P_{RX} values. The values of these parameters are shown in table 3.3. In our analysis, T_{ul} and T_{ul} are computed by averaging the observed traffic over a moving time window of 1 s.

3.4 RRC Parameters Inference

To infer the parameters RRC_{IT} and T_{Pro}, and to quantify the network overhead, we monitored real network traffic through a suitable probing system based on Tektronix commercial solution.

3.4.1 RRC Inactivity Timer

The RRC_{IT} parameter regulates the transitions from the RRC Connected state to the RRC Idle state. This switching implies that the RRC Connection of the UE is closed and the associated radio resources are released. In particular, when the RRC_{IT} expires the eNodeB initiates the procedure for the RRC Connection release, sending the message UE Context Release towards the MME on the S1 interface. This message signals to the Serving Gateway (SGW) to release the S1-U bearer. On the reverse path, the SGW communicates to the Mobility Management Entity (MME) if the procedure is correctly concluded, then the MME transmits the UE Context Release command to the eNodeB. On air interface the eNodeB sends the RRC Connection Release message to the UE and removes the UE context.

To calculate the RRC_{IT}, we analyze both the Control Plane (CP) and the User Plane (UP) traffic. Taking into account the RRC procedures, we identify each connection by analysing the acquired traffic data on the monitored S1 interface. For each connection, we evaluate the

gap between the last packet sent/received on UP (S1-U protocol), and UE Context Release message captured on CP (S1-AP protocol). Figure 3.2 shows the sequence of all RRC_{IT} values estimated from a subset of the collected traffic data.

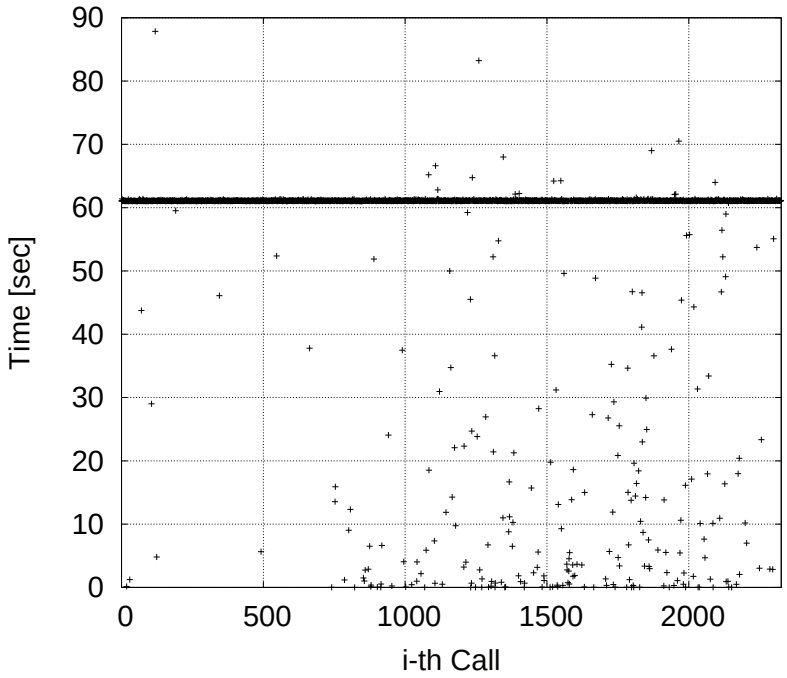

Fig. 3.2 RRC Inactivity Timer

To better understand what is happening, we explore the *Network id-Cause* field in the UEContextRelease message for all connections. For those released at 61 seconds, the id-Cause is equal to 20, meaning that the RRC connection is released due to *user-inactivity*. Unequivocally, these results suggest that the RRC_{IT} inside our monitored network is about 61 seconds and confirm our previous analysis [46], based on a commercial monitoring system. We filtered those connections released before 61 seconds and analyze the *Network id-Cause*. Figure 3.3 shows the percentage with which each case occur. Only 4 distinct causes are observed, with value 21, 23, 24 and 28.

id-Cause equals to 21 means *radio-connection-with-UE-lost*: it happens either when the UE goes out of radio coverage, or when eNB does not support RRC Connection reestablishment. In this case, the eNodeB responds with an RRC Connection Reestablishment Reject message on e-UU interface and simultaneously, request the connection to be release to the MME with cause value set to 21.

id-Cause values equal to 23 or 24 stand for *ue-not-available-for-ps-services* and *cs-fallback-triggered* respectively. Both of them are related to Circuited Switched Fallback (CSFB) procedure to allow subscriber to use existing CS based service such as voice. Thus,

when a mobile terminating or originating voice call must be performed, CSFB temporally moves subscribers down to 2G/3G and the RRC connection is released.

id-Cause with value 28 means *interrat-redirection*. Redirection in LTE is one of the mobility cases, along with handover and reselection. Same as for handover, it is triggered by eNodeB and can happen only if the UE is in connected mode. Redirection can be intra-LTE or inter-RAT. Upon the RRC Connection Release is received, the UE goes in idle mode and based on redirection information, tries to camp on a certain frequency on certain technology.

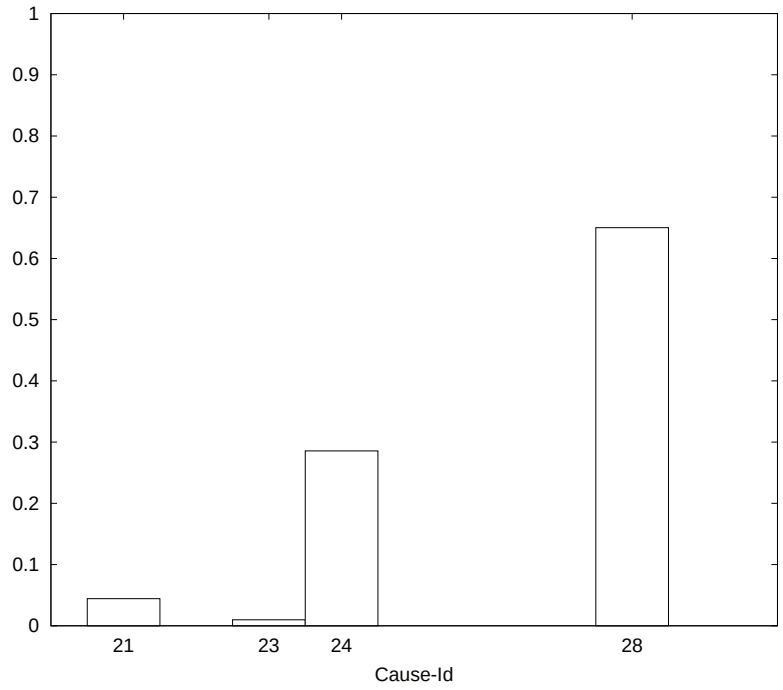

Fig. 3.3 id-Cause analysis for RRC Connections released before 61 seconds

3.4.2 Estimation of the Network Re-entry Time

During the RRC Idle state, the UE may be paged when there are data addressed to it. Paging messages are sent by the MME to all eNodeBs in a Tracking Area, to be transmitted on air. The eNodeBs compute the Paging Frame and Paging Occasion to find exact time when the UEs wake up and check the PDCCH for incoming Paging message. If the PDCCH indicates that a paging message is transmitted in the subframe, then the UEs demodulate the PCH to acquire the destination of the paging message. As shown in figure 3.4, the UE destination of the paging message starts the exchange of messages with the eNodeB so as to complete the setup of the RRC connection. Upon reception of the RRC Connection Setup message, the UE switches to the RRC Connected state and sends the Service Request message to the

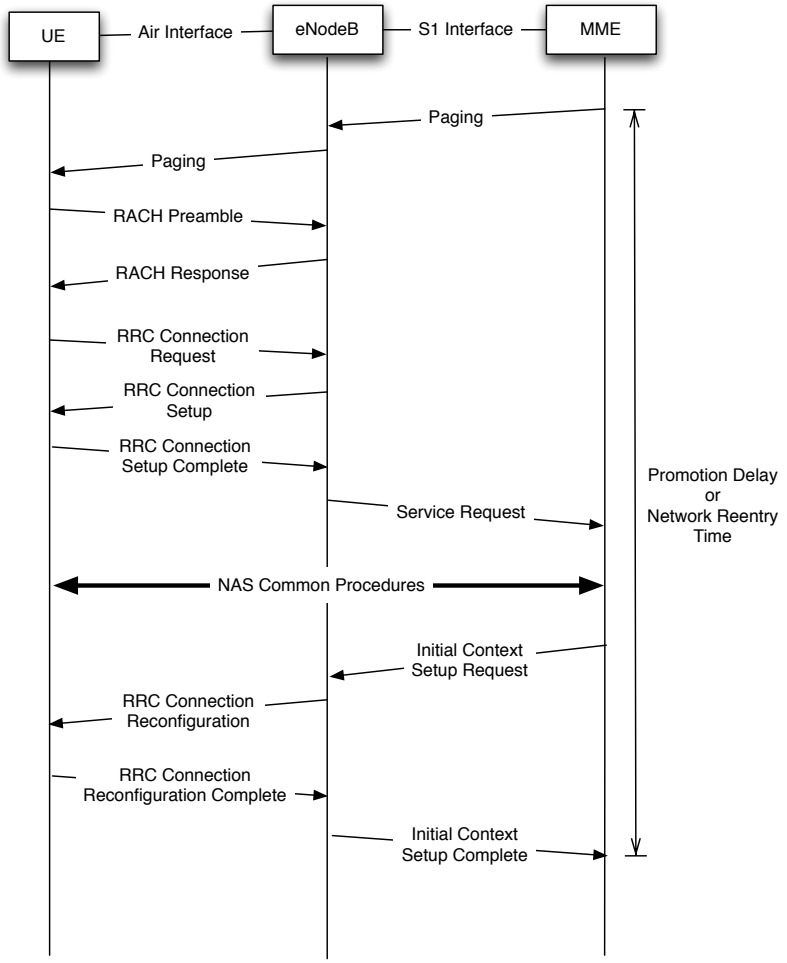

Fig. 3.4 Flow Message for paged UE

MME. Common Non-Access Stratum (NAS) procedure may follow. After receiving the RRC Connection Reconfiguration Complete message by the eNodeB, the UE is able to send an uplink packet. At this point, the eNode sends the Initial Context Setup Complete to the MME to trigger the update bearer request for downlink data.

Traffic data captured at the S1 interface allows inferring the parameter T_{Pro} by computing the time elapsed between the Paging message and the associated Initial Context Setup Complete message. The observed values of T_{Pro} for one subset of the acquired data traffic are shown in figure 3.5. In the figure, the average value (0.9165 s) obtained over the observed values during one of the one day monitoring session is shown. The relative large deviation from the average value is due to different factors. Firstly, paging DRX cycle affects the observed T_{Pro}. Indeed, after the reception of the paging message sent by the MME, the eNodeB cannot immediately send the paging, but it must wait for the paging occasion to

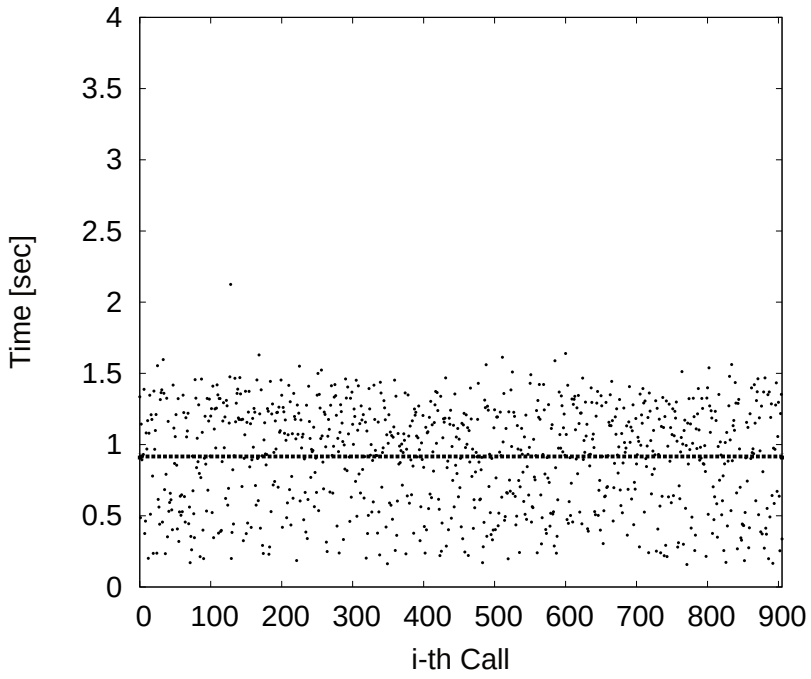

Fig. 3.5 LTE Promotion Time for paged UE

send this message to the UE, as specified in TS 36.304 [11]. Further delay variance is added by the messages exchanged to allow the UE to complete the RRC Connection Setup. Lastly, delay variance is increased because of the UL bandwidth grant.

3.4.3 Network Overhead

In this subsection, we evaluate the cost to complete the UE transition from the RRC Connected to the RRC Idle (and viceversa) in terms of the number of *Bytes* transmitted on the CP. Furthermore, we evaluate the number of *Bytes* of signalling needed to complete some relevant procedures when the UE is in the RRC Connected state. The results derived from the analysis of the traffic data acquired during one day of monitoring session are shown in figure 3.6. The figure depicts the amount of bytes exchanged inside each observed RRC connection. We observed similar results by analysing the traffic data acquired in different one-day measurement sessions.

In the figure, we can identify different groups of values that appear more frequently than others. In particular, we can recognize six main groups of values.

The first group accounts for connections where 646–658 bytes are exchanged in the CP. The values in this range are the basic amount of data needed to build and remove RRC

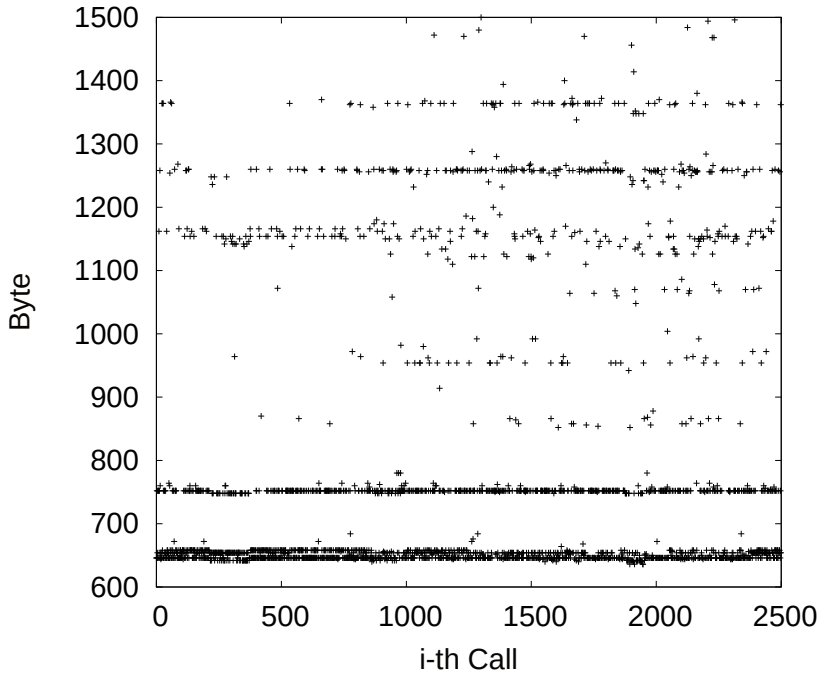

Fig. 3.6 Network Overhead

connections triggered by Mobile Originated (MO) calls. More than 50% of RRC connections are MO.

The second group (752–764 bytes) represents the basic amount of data needed to setup and remove RRC connections triggered by Mobile Terminating (MT) calls. About 25% of RRC connections are MT. The difference between MO and MT calls, is given by Paging message (106 bytes).

The third group (around 1150–1200 bytes) is composed of several contributions: i) MO calls with NAS common procedures, ii) MO calls released by Handover procedure, and iii) MO calls with Context Modification procedure inside. This group encloses less than 10% of all detected connections.

The fourth group encloses less than 10% of the observed connections, and includes the same contributions of the previous one, but for MT calls.

Fifth and sixth groups incorporate MO and MT calls released by CSFB procedure, respectively. Both represent more or less the 5% of all calls.

Table 3.1 Protocol Analysis

Session	HTTP	HTTPS	DNS	Other TCP	Other UDP
Day1	54.114%	33.53%	1.17%	2.99%	8.13%
Day2	69.00%	22.57%	1.01%	3.25%	4.12%
Day3	51.87%	42.27%	1.38%	3.24%	0.98%
Day4	6.05%	0.11%	33.87%	57.74%	2.13%
Day5	2.53%	0.60%	63.39%	32.28%	1.12%
Day6	2.73%	0.53%	50.20%	45.16%	1.32%
Day7	3.61%	4.76%	46.14%	43.96%	1.46%

3.5 Which Inactivity Timer is suitable for LTE network?

In the previous section, RRC_{IT} and T_{Pro} were inferred analyzing real traffic. Tuning properly these parameters it is a mandatory to save power and resource inside LTE network. In particular, RRC_{IT}, which mostly affects power consumption, should be set according to statistical nature of traffic. To evaluate and characterize traffic behaviour, we analyse traffic of dataset III, by using the methodology explained in Chapter 2. extending the *libpcap* library. Traffic was treated according to security procedures and properly anonymized to respect customers privacy. Thus, no payload data is considered except for HTTP headers, and no personal information is used to develop this study. The measurement session has been carried out in winter 2014. As for your convenience, the measurement session has been carried out for 7 days in winter 2014, providing 25 GB of LTE traffic data.

The first analysis is devoted to find the percentage of the most common protocols. Table 3.1 summarizes the results.

TCP packets, and in particular HTTP and HTTPS packets, always represent the dominant portion of the traffic. It means that understanding the TCP/HTTP connection behaviour is crucial to properly set RRC_{IT} and T_{Pro} in order to achieve the trade-off between energy savings and signalling traffic overhead. For each observed connection, we collect the exchanged frames using the canonical 5-tuple: Protocol, IP source address, IP destination address, source port, and destination port. Then, we compute the number of exchanged bytes as well as the duration of the observed TCP connections. For TCP flows, we set the beginning of the connection when the three-way-handshake is completed successfully, and declare the connection finished when the last FIN or RST packet is received. HTTP persistent connections use a single TCP connection to send and receive multiple HTTP requests/responses. From one side, this strategy is efficient since it avoids opening a new TCP connection for each request/response pair. On the other side, an open TCP connection wastes radio resources if it is unused. In the HTTP 1.0, there is no official specification about

how keep-alive operates. If the client supports keep-alive, it adds an additional header to the request:

```
Connection : keep-alive
```

Under HTTP 1.1 any HTTP connection is a persistent connection by default. Thus, unless otherwise indicated, the client should assume that the server will maintain opened the current connection, even if some errors occur. To close a persistent connection either the client or the server can use the Connection header field as follows:

```
Connection : close
```

In this way, the connection will be closed after the completion of the response. In addition, a default connection timeout can be used to close the connection if no new requests are received by the server before the timeout expires. For example, Apache 2.0 and 2.2 use 15 and 5 seconds timeout, respectively. The server can notify the keep-alive timeout using the Keep-Alive header field as follows:

```
Keep-Alive : timeout = 'value'
```

Exploiting DPI functionality, we analyse HTTP header for HTTP GET, POST, CONNECT or Response messages. Table 3.2 summarizes the main results. For each TCP/HTTP connection we evaluate the HTTP version both on client side, analysing HTTP GET, POST or CONNECT messages, and on server side, analysing HTTP Response messages. If client and server use the same HTTP version we labeled the connection with that version, otherwise we labeled the connection with the lowest version. Under each connection, if both client and server use the Keep-Alive command, we tagged the connection as *Keep Alive*. If one of them, or both, use Close command, we marked the connection as *Close*.

Approximately, 99% of HTTP connections uses HTTP 1.1. Out of them, the 88% uses Keep-Alive command.

Using persistent connection, increases the possibility to reuse the same connection and query more objects to the same server, reducing the overhead due to three way handshake and slow start. On the other hand, delaying the closure of a TCP connection can impact resource usage in wireless network: delayed FIN packets force the reestablishment of the radio bearer to exchange just few packets, that will remain active until the RRC_{IT} expires. Knowing the statistical properties of the underlying traffic is essential to configure radio parameters conveniently. From our perspective, it is important to provide statistical information about the *Maximum Packet Delay* seen inside each TCP connection. Intuitively, we defined the

Table 3.2 HTTP Protocol Analysis

	Measurement Session	Connection Version	Keep-Alive	Close
Day1	HTTP 1.1 HTTP 1.0	99,90% 0,10%	88,40% 20,00%	11,60% 80,00%
Day2	HTTP 1.1 HTTP 1.0	98,27% 1,73%	81,66% 80,00%	18,37% 20,00%
Day3	HTTP 1.1 HTTP 1.0	99,12% 0,88%	84,86% 20,83%	15,14% 79,17%
Day4	HTTP 1.1 HTTP 1.0	98,15% 1,85%	88,64% 77,01%	11,36% 22,99%
Day5	HTTP 1.1 HTTP 1.0	99,50% 0,50%	79,61% 8,00%	20,39% 92,00%
Day6	HTTP 1.1 HTTP 1.0	98,03% 1,97%	83,44% 89,06%	16,56% 10,94%
Day7	HTTP 1.1 HTTP 1.0	98,80% 1,20%	82,71% 45,72%	17,29% 54,28%

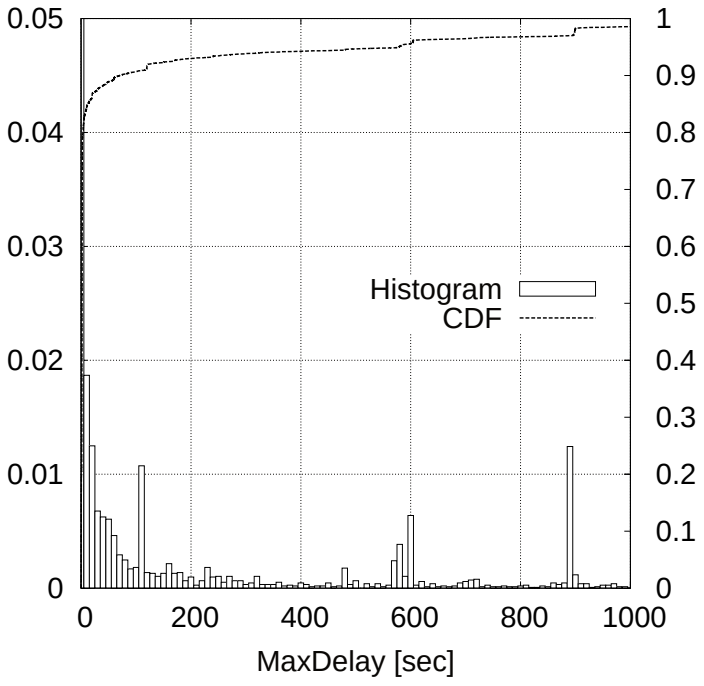

Fig. 3.7 Maximum Gap: CDF and Histogrm

Maximum Packet Delay as the maximum gap between two consecutive packets inside one TCP connection. Referring to a one day monitoring session, the histogram and the CDF of the observed *Maximum Gap* are shown in figure 3.7. The results obtained in the traffic data acquired in the other days are quite similar.

Figure 3.7 points out that about 85% of TCP connections have a *Maximum Gap* value lower than 5 s, whereas approximately more than 99% have a value lower than 1000 s. For values higher than 60 s, we observe several TCP connections showing a *Maximum Gap* of 110, 600 and 890 s. Later in this section, we'll explore *who are* this connection.

To better appreciate TCP connection behavior, we correlated the *Maximum Gap* with the connection duration. Thus, we defined *R*, as the ratio between the TCP connection duration and relative maximum packet delay:

$$R = \frac{ConnectionDuration}{MaximumPacketDelay} \tag{3.7}$$

In figure 3.8 is plotted the cumulative density function (CDF) for the R value.

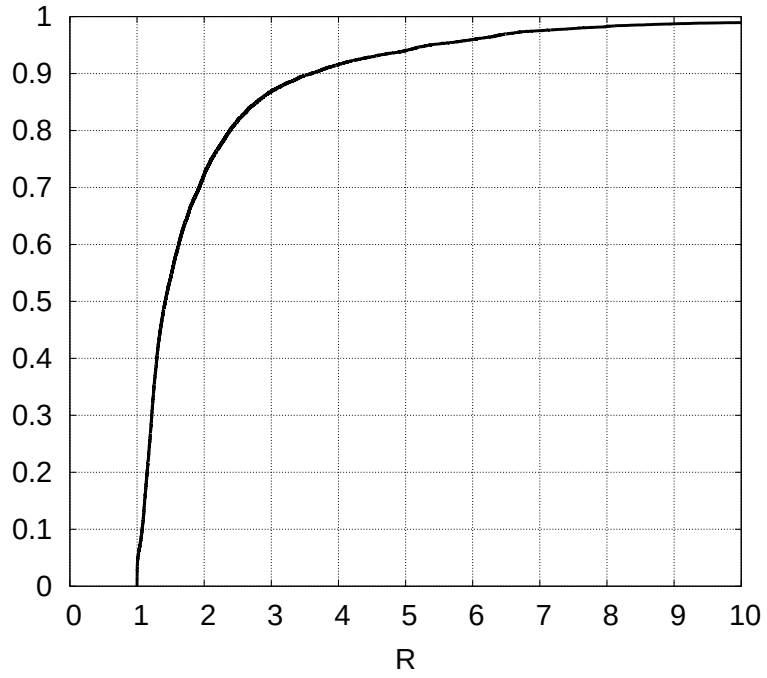

Fig. 3.8 *R* value : CDF

The figure shows that, in most of TCP connections, the maximum gap between two consecutive packets belonging to the same connection is comparable with the duration of the connection itself. In other words, it means that, during its lifetime, the connection is kept open without transferring packet. This is probably due to the HTTP keep-alive mechanism described before.

The next step was to investigate how the most common applications behave. By applying the procedure described in Chapter 2, section 2.6, we select the four applications/services with

the largest number of TCP connections: Facebook, Akamai, Google and Apple. Facebook is responsible for TCP connections with *Maximum Gap* equal to 110 and 890 s, whereas connections to Apple mostly contribute to the observation of 600 s. Apparently, no match for a particular behaviour can be deducted for Google or Akamai connections (figure 3.9).

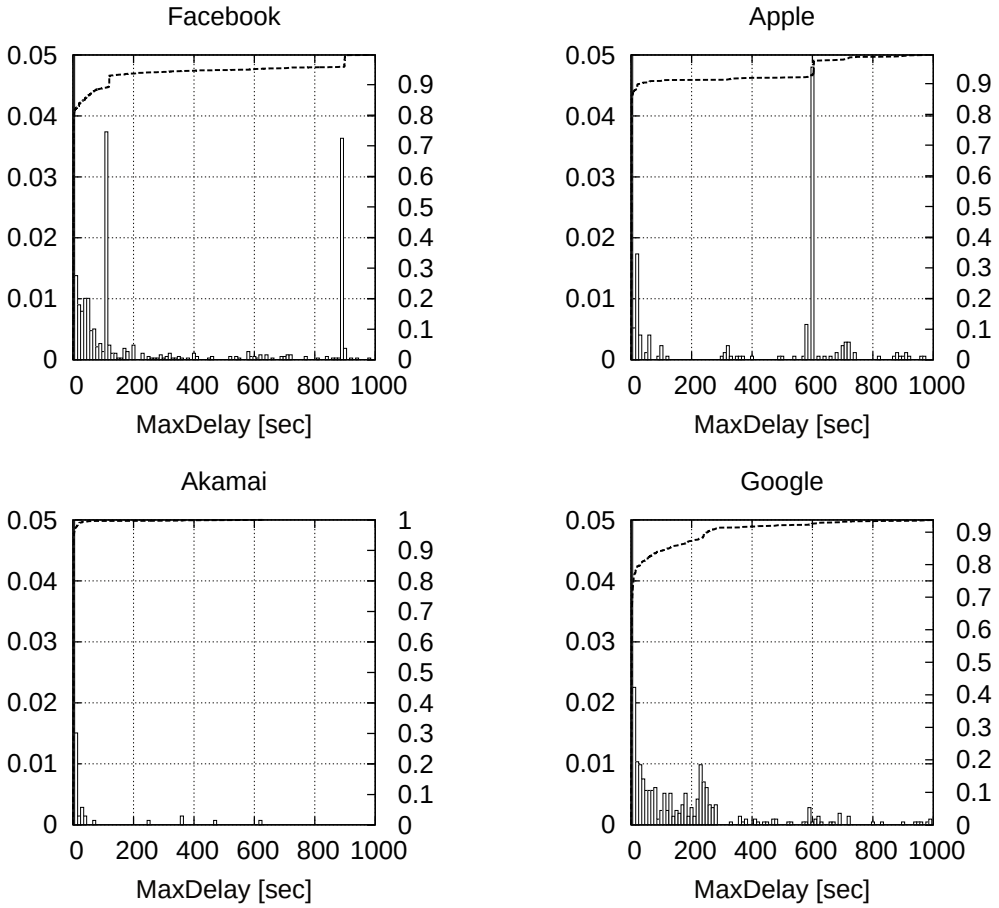

Fig. 3.9 Maximum Gap for considered applications: CDF and Histogram

Finally, in figure 3.10 we propose the CDF of the R value, for the TCP connections belonging to the aforementioned four service groups in comparison with the CDF for all TCP connections.

Our results show that especially Facebook and Akamai connections have a *Maximum Packet Delay* comparable with the connection duration: more than 80% of these connections have a silence period that is at least one half of the overall duartion.

In conclusion, this section has provided useful information about TCP/HTTP(s) behavior and packet distribution inside connections. Even if some applications present peculiar characteristics, the larger part of TCP connections have a maximum inter-packet delay that

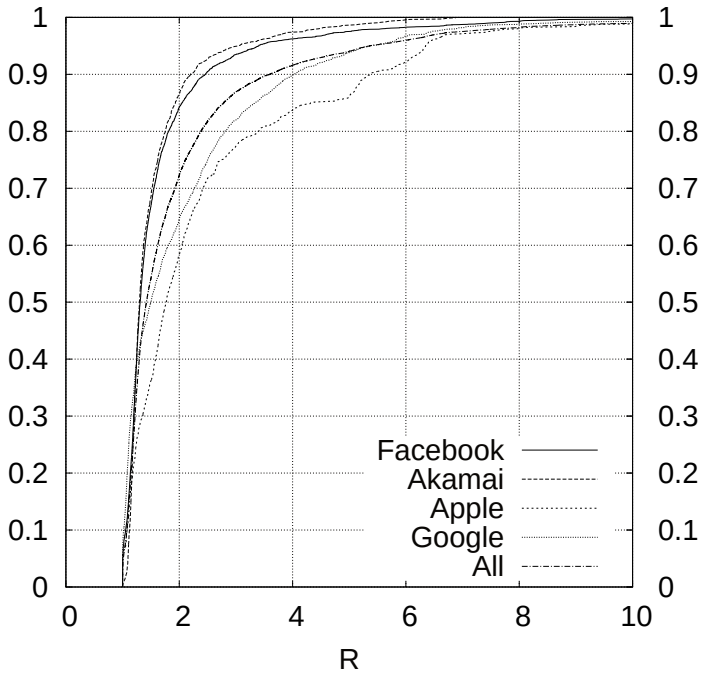

Fig. 3.10 *R* value for considered applications: CDF

is less than 60 seconds, therefore it seems that setting the RRC_{IT} equals to 61 seconds is over-sized, and causes to the smartphone to spend more power than it is needed.

3.6 Experimental Results

To estimate the power consumed by handset devices we apply our power model to each user, observed during one day of monitoring session. Table 3.3 summarizes the parameters used during our simulations.

Table 3.3 Power Model Parameters

RRC Inactivity Timer	61 - 70.554 - 12.154 - 3.275 - 2.065 s
Promotion Delay	0.916 s
DRX Inactivity Timer	0.100 s
α_{ul}, α_{dl}, β	438 mW, 52mW, 1288mW
Power DRX Connected	1325 mW
Power Promotion Delay	1210 mW
Power Idle	0 mW

It is worth noting that the DRX Inactivity timer has been set to default value, T_{Pro} to the value estimated with the analysis presented in section 3.4.2, and the power values are taken from [34].

Firstly, we calculated the consumed energy setting the RRC_{IT} with the value obtained in section 3.4.1, considering it as the reference value.

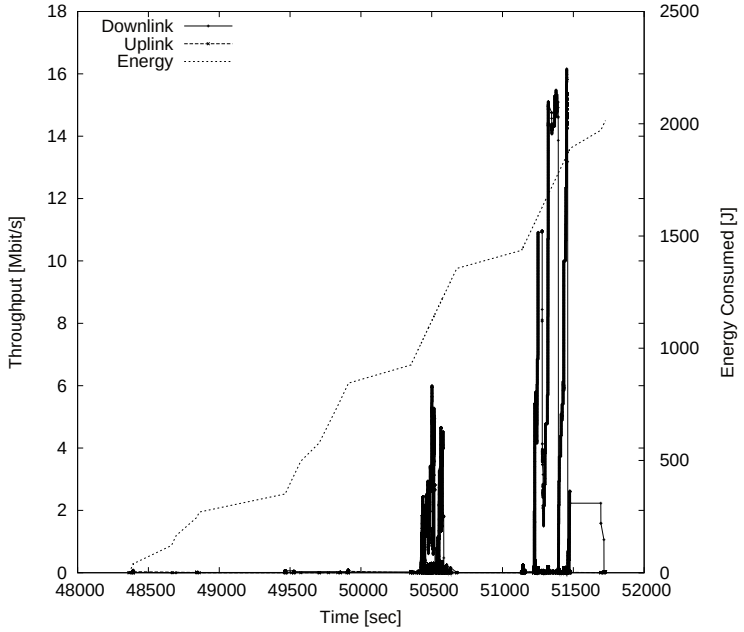

Fig. 3.11 Energy Consumed vs Throughput

In figure 3.11 is shown the output of our model together with the uplink and downlink throughput. It is worth to be noted that, depending on user throughput, the growth of energy consumed is more or less pronounced.

Accounting for the characteristics of UP traffic highlighted in section 3.5, we decide to estimate the energy consumption in different scenarios characterized by a different value of the RRC_{IT}. In particular, we considered the reference scenario, where $RRC_{IT} = 61s$, and other 4 other scenarios with RRC_{IT} equal to 70.449, 12.154, 3.275 and 2.065 s. These values respectively represent the 90, 85, 80 and 75 percentile of the empirical cumulative density function of the *Maximum Gap*, shown in figure 3.7. Hence, for each scenario, we simulate the evolution of the RRC state machine of each user. Simultaneously, tacking into account the traffic exchanged by each user, we computed the number of RRC Connection procedures triggered by the expiration of the RRC_{IT}.

The analysis considers the eight users that exchange the highest number of bytes with the network. The results obtained in each scenario have been normalized with those obtained with the reference scenario. Figure 3.12 and 3.13 show the total amount of consumed energy and the number of RRC connection procedures respectively.

Setting $RRC_{IT} = 70.449s$ does not provide significant differences with respect to the reference scenario. The scenario $RRC_{IT} = 12.154s$ allows saving energy from 30% to 50%

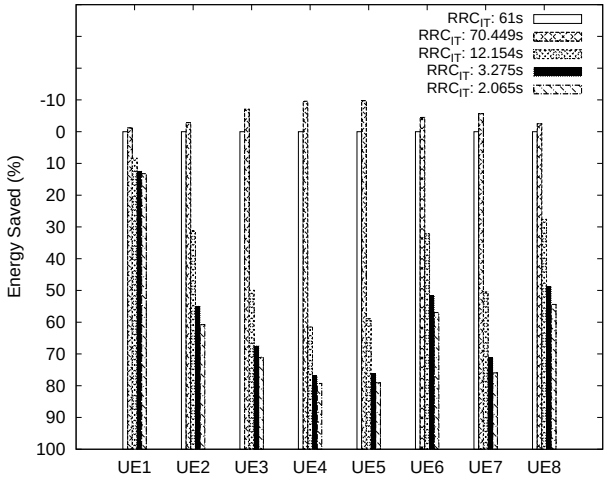

Fig. 3.12 Per-UE consumed energy, normalized to the reference value

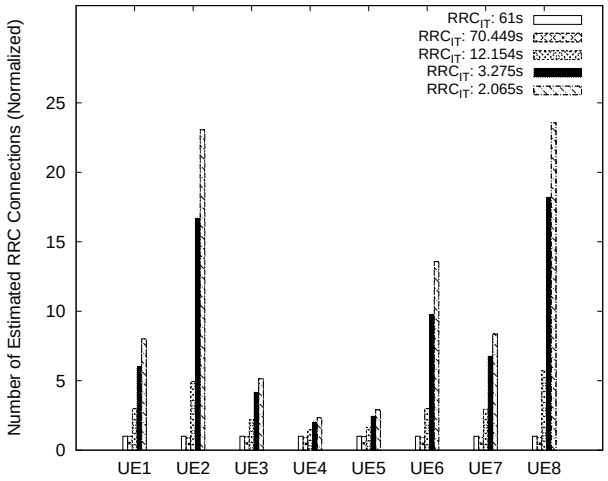

Fig. 3.13 Per-UE umber of RRC connection procedures, normalized to the reference value

for all UEs, except for UE1. The cost of the energy savings is the increase of the number of RRC connections. We explore UE1 and UE8 to understand the impact of traffic profile in the simulations. Looking at UE1 we noticed that it was performing one video streaming session. In this case, changing the RRC_{IT} has no impact on the evolution of the RRC state machine, because the *Maximum Gap* is very low, less then RRC_{IT}. The device remains in RRC Connected state during the whole video session. UE8 traffic profile, instead, is much more varied and involves different applications, such as Twitter, Facebook and Web Browsing

sessions. In this case, we can save up to 30% of the consumed energy, increasing the number of RRC connections of 5.72 times.

Configuring $RRC_{IT} = 3.275s$ allows saving, on average, an additional 20% as compared to the previous value. Depending on the user traffic, we observe an increase of the number of RRC connections up to 18.2 times.

Changing the RRC_{IT} from 3.275 s to 2.065 s does not produce significant energy savings, whereas the number of RRC connections is considerably increased.

It is worth noticing that by increasing the number of RRC connections on a large time scale, it is approximately costless for handset device, because the average amount of network overhead has been estimated around 650 or 756 bytes for MO or MT calls, respectively. Therefore, rather than estimating per-user network overhead, it is reasonable to evaluate the cumulative impact of all users. Figure 3.14 shows the total number of estimated RRC connections triggered by the users under coverage. In this case, increasing too much the number of RRC procedures could be troublesome for the eNodeB. Heuristically, we can say that decreasing the RRC_{IT} beyond the 80 percentile of the observed *Maximum Gap* does not produce significant energy savings, but a significant increase of signaling traffic overhead. In addition, it should be considered that forcing too many UEs to enter the RRC Idle state increases the probability of collision during RACH procedure whenever more UEs decide to wake up at the same time.

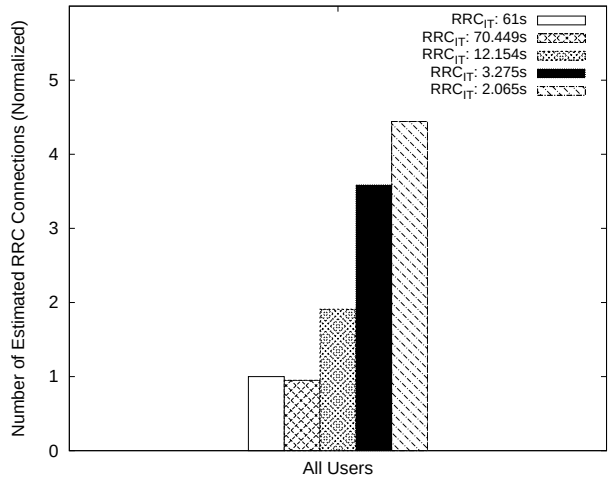

Fig. 3.14 Overall number of RRC connection procedures, normalized to the reference value

3.7 Conclusion

Based on a simplified power model, this Chapter presents the analysis of the energy consumed by the UE and of the control traffic overhead for different values of the RRC Inactivity Timer. The key parameters used during the simulation analysis are inferred directly from passive measurements carried out monitoring a commercial eNodeB of one of the Italian Mobile Operators. In particular, reasonable values for RRC_{IT} were deducted looking at UP traffic and exploring TCP/HTTP(s) behavior and packet distribution inside connections. Even if some applications present peculiar characteristics, the larger part of TCP connections have a maximum inter-packet delay that is less than 60 seconds, therefore it seems that setting the RRC_{IT} equals to 61 seconds is over-sized, and causes to the smartphone to spend more power than it is needed. Thus, we compared simulation results with the reference value and heuristically we noticed that decreasing too much the RRC_{IT} does not provide significant energy savings, at the cost of an high increase of network overhead. The simulation results suggest that RRC Inactivity Timer set in the monitored devices is not optimized to obtain a good trade-off between traffic signalling load and smartphone battery usage. The analysis shows that values of the timer obtained analysing the User Plane traffic can lead to saving UE's energy, at the cost of a low increase of the signalling traffic overhead. Good values for RRC_{IT} should be in the range of 12-3 seconds. The results of this work open promising research opportunities, such as the definition of algorithms for adaptive setting of the RRC Inactivity Timer, and the evaluation of the RRC performance under different (specially, heavy) load conditions, which will be evaluated in the next Chapter.

Chapter 4

RACH/RRC Performance

4.1 Introduction

In the LTE systems, battery lifetime and network traffic overhead on control plane may be largely affected by the Discontinuous reception (DRX) configuration and the RRC Inactivity Timer. In Chapter 3 we propose an analysis aimed at defining how to properly set the RRC Inactivity Timer to achieve a trade-off between energy savings and traffic overhead on the control plane. The results suggest lowering the RRC inactivity timer to save energy of user's device. Nevertheless, diminishing the RRC_{IT} and forcing too UEs to release radio resource, may cause the eNodeB to be stressed by a burst of RACH/RRC connection requests if such UEs are paged for an incoming packet or need to send an uplink packet. Therefore, beyond the heuristic provided in the previous chapter, the question on *how much decrease the Inactivity Timer* is still open and constrained by eNodeB capability to handle surge of control plane traffic.

In this Chapter, in order to evaluate the robustness of the eNodeB against burst of signalling traffic, we provide:

- experimental results, testing a real eNodeB with burst of RRC connection request
- analytical models to forecast the collision probability for RACH procedure, depending on the number of under coverage devices and the actual traffic statistics
- simulation results and guidelines for preamble separation between Human Type and Machine Type Communications

In section 4.2 we'll show how we set up RRC stress-test using the Ixia load generator, IxLoad.

In section 4.6 we propose an analytical model to evaluate the collision probability on the RACH as a function of the number of UEs, the number of available preambles and the Interarrival times of the RACH Requests of the average user. The model for the IRR of the average user is obtained from real traffic data captured at the eNodeB of a mobile operator, and is derived by emulating the RRC state machine for different RRC_{IT} settings.

In section 4.7 we provide a set of guidelines for the resource allocation task in the RACH. In particular, the study investigates the impact of both the backoff indicator scheme and the maximum number of retransmissions on the RACH performance parameters. The rate of RACH requests associated with the HTC traffic is modelled by inferring their statistical properties starting from a dataset acquired in an operational eNodeB. The estimation of the average delay and the average number of maximum retransmissions gives insights on how many preambles should be reserved for HTC in order to meet the target performance, and provides suggestions on the configuration of the backoff indicator.

4.2 Stress Test with Ixia

Diminishing the RRC_{IT} and forcing too UEs to release radio resource, may cause the eNodeB to be stressed by a burst of RRC connection requests if such UEs are paged for an incoming packet or need to send an uplink packet. In order to evaluate the robustness of the eNodeB against burst of RRC connection requests, we set up a test using the Ixia load generator, IxLoad. b IxLoad is a commercial solution for testing network devices throughout the emulation of data, voice, video and their associated protocols. IxLoad works with Ixia's hardware platforms to transmit and receive control and data plane traffic with the device under test. Ixia's chassis are populated with hot-swappable test interface cards. Each test port is equipped with an independent processor and substantial memory in addition to specialized traffic stream generation and capture hardware. For our test we used the XGS12 Chassis with three Xcellon-UltraTM NP, each one equipped with 12-1Gb port.

The test is designed as follow. On the eNodeB, we set the maximum number of UEs that can be RRC connected at the same time and the RRC_{IT}. We change the RRC_{IT} across multiple tests. According to the RRC_{IT} set on the eNodeB, we change the rate with which the RRC requests are randomly generated by the UEs camped on the base station. The estimation of the RRC rate has been made looking at real traffic capture and is described below. Once the UE is in connected, it creates a dummy HTTP 1.1 Request, throughout a GET command for a file of 4KB size. Upon the reception of the HTTP response, the UE doesn't perform traffic anymore and comes back in idle state.

4.2.1 Rate for RRC Connection Requests

We calculate the rate distribution for the RRC_{IT} values shown in table 3.3. As we've done to estimate the number of RRC connections (section 3.6), we use one live traffic capture as reference and, basing on the UP packet inter-arrival times, we calculated the inter-arrival times for the triggered RRC connection requests. Figures 4.1, 4.2, 4.3 show the histograms and the empirical cumulative density functions ($ecdf$) for the RRC connection inter-arrival time, setting the RRC_{IT} to 70.554, 12.154 and 2.035 seconds. The $ecdf$s for RRC_{IT} equal to 61 and 3.275 seconds are similar to those calculated for 70.554 and 2.035 seconds, respectively.

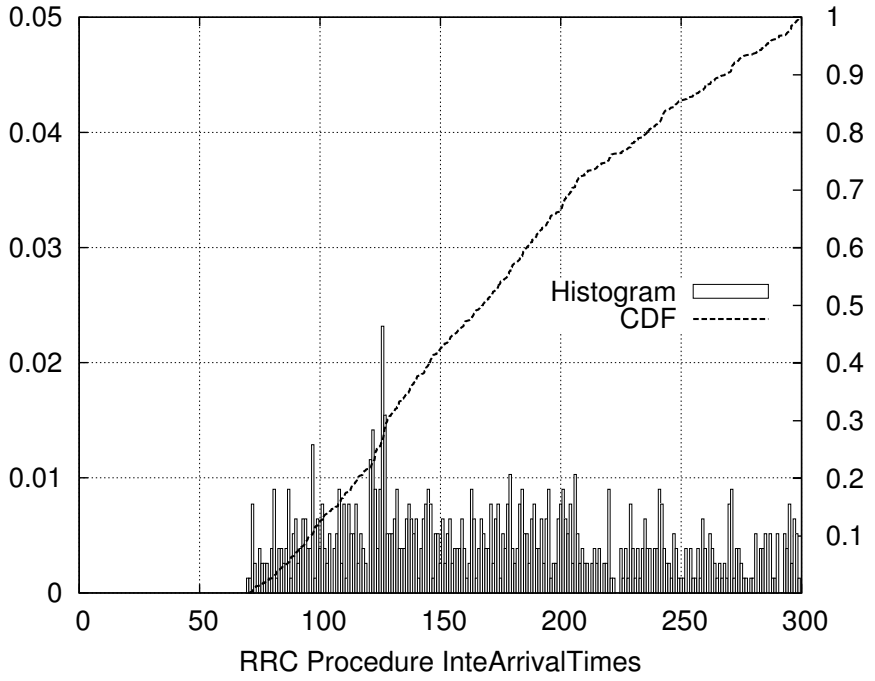

Fig. 4.1 Rate for RRC_{IT} = 70.554s

To detect any possible weaknesses for the eNodeB, we propose a worst case scenario in which we don't consider RRC_{IT} larger than 300s, that explains why the largest value for the histogram/cdf is 300s. Thus, the interval between two consecutive RRC request is included between the RRC_{IT} (we cannot trigger one RRC Connection Request until the connection itself has not been released by the eNodeB, of course) and 300s.

It is worth noticing how the trend of the probability density function (pdf) and the slope of the cumulative density function (cdf) are affected by the chosen inactivity value. In particular, as smaller is the value for the inactivity timer, the more is the probability of getting a low value for the inter arrival time. The main stats for all the selected RRC_{IT} are collected and displayed in table 4.1.

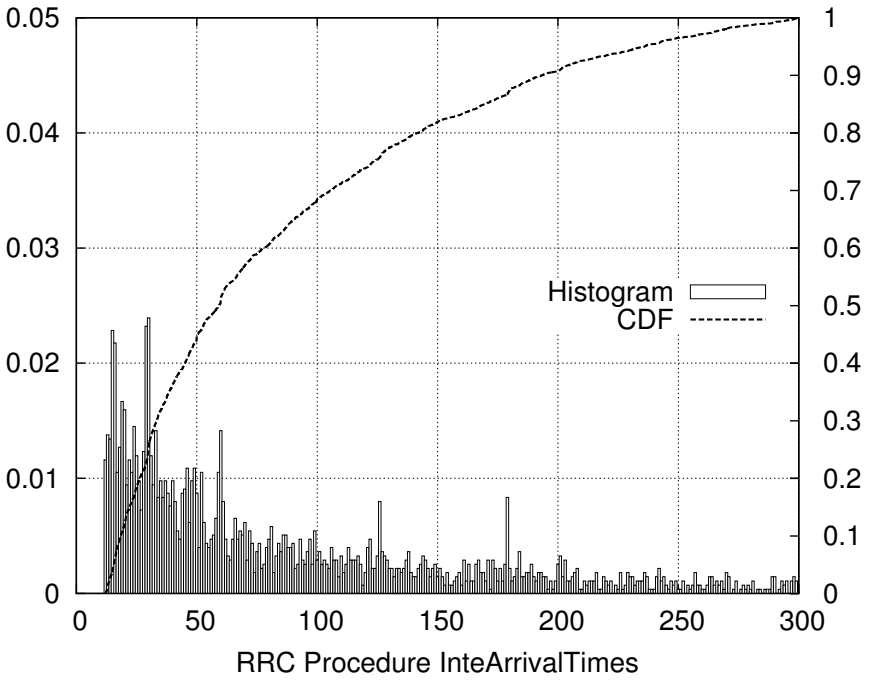

Fig. 4.2 Rate for RRC_{IT} = 12.154s

Table 4.1 Stats for different RRC_{IT}

RRC Inactivity Timer (sec)	70.554	61	12.154	3.275	2.035
Mean (sec)	173.359	163.212	84.536	42.805	34.348
Standard Deviation (sec)	61.574	63.695	69.973	59.519	55.144
90 Percentile (sec)	268.210	260.004	193.493	128.025	108.289
80 Percentile (sec)	235.808	228.0733	142.031	64.910	45.235
70 Percentile (sec)	204.673	198.038	105.422	33.866	22.229
60 Percentile (sec)	185.912	177.444	77.987	21.666	14.942

On IxLoad we have the constraint to use uniform random variables, that means that if we want to confer randomness on the RRC request generation, the only way is to generate uniformly distribute request over a customizable period of time. This approximation is acceptable for RRC_{IT} equals to 70.554 and 61 seconds (figure 4.1), but the other distributions are far away for being uniformly distributed (figures 4.2 and 4.3). We decide to limit the possible values for the RRC request in the range (RRC_{IT}, 60Percentile(RRC_{IT})], to generate the most probable values for these distribution. As explained before, our first purpose is to test eNodeB reaction under heavy load conditions, thus, even with the limitation imposed by our tools, we may reuse the statistical information inferred by live capture analysis to perform comparable fair tests.

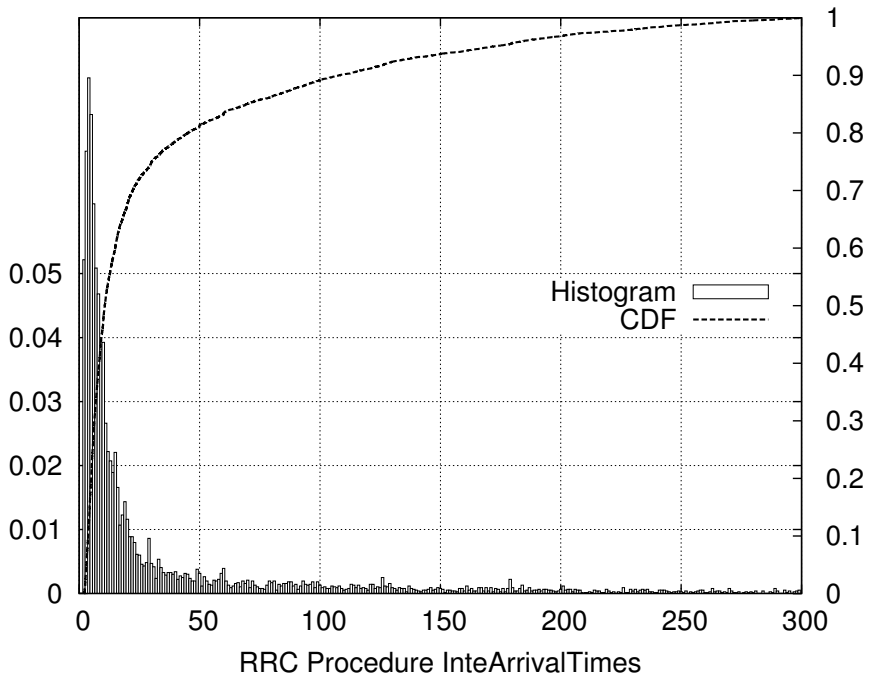

Fig. 4.3 Rate for RRC_{IT} = 2.134s

4.2.2 Test Configuration

Here we provide a more detailed discussion about our test setup. For each RRC_{IT} we perform the test as follows. On the eNodeB, the maximum number of UEs that can be RRC connected has been set. On Ixload, we set the number of UEs under coverage. Five UEs are always responsible for generating uplink and downlink UDP packets to fill the available bandwidth in both directions. We configure the eNodeB with 10MHz (50 Resource Block), thus the maximum throughput that we may theoretically achieve is 70Mbps in downlink e 25Mbps in uplink, with smartphone category 5. The Xcellon card in the Ixia Chassis handles the generation and incapsulation of all data units from Application to PDCP layer. PDCP layer is the most CPU hungry element and the load is proportional the throughput that have to be generated. In order to not overload the port CPU and obtain results that are affected by this, we configure the test with multiple port and maintain the average CPU load lower than 50%.

The left UEs are used to generate the HTTP requests and trigger the RRC Connection Requests. According to the RRC_{IT} set on the eNodeB, we change the rate with which the connection are randomly generated by the UEs camped on the base station, as explained in section 4.2.1. Once the UE is in connected, it creates a dummy HTTP Request, throughout a GET command for a file of 4KB size.

In IxLoad we can select the version of the HTTP protocol that you want to use in the test: 1.0 or 1.1. Under HTTP 1.0 without Keep-Alive, when a user clicks on a link for a web page, a TCP connection request is sent by the client to the server. When the server accepts the connection, the client sends an HTTP GET request to download the web page from the server. After making a single HTTP request, the client closes the TCP connection. If we check Keep-Alive box, the client adds the Connection: Keep-Alive header to its request and the server keeps the connection open after the first request. When the client sends another request, it uses the same connection. This will continue until either the client or the server decides that the session is over, and one of them closes the connection. If a client and server use HTTP 1.1, multiple HTTP requests can be sent by the client on a single TCP connection. This saves processing power, since fewer TCP connections need to be established. HTTP 1.1 also allows for persistent connections, enabling connections to stay up for (relatively) long periods of time. In HTTP 1.1, the server initiates the closing of the TCP connection by sending a FIN message.

As we saw in section 3.5, most modern browsers use HTTP 1.1. To be fair with our analysis, we choose HTTP 1.1 (both for client and server) and set the maximum number of transaction for each connection as maximum as possible. Upon the reception of the HTTP response, the UE doesn't perform traffic anymore and comes back in idle state.

The UE waits for a random value of time, that depends on the RRC_{IT}, and then performs another GET request, and so on. Each test lasts two hours.

At the end of each test we export the log .dct file from Ixload test repository folder and parse it with a perl script. The log looks like the picture in figure 4.4. The perl script check for the user id (UEid) if the RACH procedure is completed correctly (highlighted in red) and calculate the time needed to setup the RRC connection (highlighted in yellow).

```
00129.9794 {UE_163_162_149_25_3_1.1} rrc_r8_lte S Data_Req_UE Cell_LCId=[cell_Id=441 CCCH=[UE_Id=1]]
Data=[rrcConnectionRequest=[rrcConnectionRequest_r8=[randomValue=00000011010001100010011000100011101010111 establishmentCause=mo_Signalling spare=0]]]
00129.9806 {UE_163_162_149_25_3_1.3} cstats_ue_r8_lte R CSTATS_UE_Enable_Resp UE_Id=1 ConfigStatus=[ConfigResult=SUCCESS ConfigReason='PHY stats will become available after
initial RACH procedure completes']
00129.9808 {UE_163_162_149_25_3_1.4} cerr_ue_r8_lte R CERR_UE_Enable_Resp UE_Id=1 ConfigStatus=[ConfigResult=SUCCESS]
00129.9808 {UE_163_162_149_25_3_1.4} cerr_ue_r8_lte R CERR_UE_Enable_Resp UE_Id=1 ConfigStatus=[ConfigResult=SUCCESS]
00129.9808 {UE_163_162_149_25_3_1.5} clog_ue_r8_lte R CLOG_UE_RLC_Disable_Resp UE_Id=1 ConfigStatus=[ConfigResult=SUCCESS]
00129.9808 {UE_163_162_149_25_3_1.5} clog_ue_r8_lte R CLOG_UE_MAC_Disable_Resp UE_Id=1 ConfigStatus=[ConfigResult=SUCCESS]
00129.9808 {UE_163_162_149_25_3_1.5} clog_ue_r8_lte R CLOG_UE_MAC_Disable_Resp UE_Id=1 ConfigStatus=[ConfigResult=SUCCESS]
00129.9812 {Xair-80821} c l $>> INFO (inst 0) MAC:    [UE = 1]    RAPID 0 (GrpA 0-31, GrpB 32-39) selected
00129.9813 {Xair-80821} c l $>> INFO (inst 0) MAC:    MAC requesting RACHs on subframe 1 resource 0 - As=1 Bs=0 (UEIDs 1  from this instance)
00129.9816 {UE_163_162_149_25_3_2.11} ixcatprim R tcp_data_indication conn_id=14 address=[ip_text='10.0.3.2' port=32794] local_ip_text='10.0.3.2' local_port=5678
i=[DCT_Start=[protocol_version=2 transaction_id=2 subscriber_id=11 scenario_id=Attach apn_id=1 cell_id=441 attach_retry=0]]

00129.9863 {Xair-80821} c l $>> INFO (inst 0) MAC:    RACHs transmitted by PHY on RA-RNTI=2 RAPID_bitmask=1 (UEIDs 1  from this instance)
00129.9863 {Xair-80821} c l $>> INFO (inst 0) MAC:    Starting 3ms timers for RA-RNTI=2 4
00129.9903 {Xair-80821} c l $>> INFO (inst 0) MAC:    Awaiting RARs for RACHs transmitted by PHY on RA-RNTI=2 RAPID_bitmask=1 (UEIDs 1  from this instance)

00129.9920 {Xair-80821.1} mac_r8_lte R RAR metadata=[Rnti=2, CRC_Ok, Sysframe=607, Subframe=6, PhyData=[DCIFormat=1C resource_type=0 AL4 MCS=1 RV=0 dlReTx=0 RBLen=4 HarqID=0
NDI=0 TB1], Mac_pdu_len=7] RAR_header=[E=0 T=1 RAPID=0] RAR_BODY=[MAC_RAR=[TA=0 UL_Grant=49332 Temporary_C_RNTI=39897]]

00130.0032 {UE_163_162_149_25_3_1.1} rrc_r8_lte R Data_Ind_UE Cell_LCId=[cell_Id=441 CCCH=[UE_Id=1]] Data=[rrcConnectionSetup=[rrc_TransactionIdentifier=2
rrcConnectionSetup_r8=[radioResourceConfigDedicated=[srb_ToAddModList=[SRB_ToAddMod=[srb_Identity=1 am=[ul_AM_RLC=[t_PollRetransmit=ms100 pollPDU=pInfinity pollByte=kBinfinity
maxRetxThreshold=t16] dl_AM_RLC=[t_Reordering=ms50 t_StatusProhibit=ms0]] logicalChannelConfig_defaultValue]] explicitValue=[ul_SCH_Config=[maxHARQ_Tx=n5
periodicBSR_Timer=infinity retxBSR_Timer=sf320 ttiBundling=FALSE] timeAlignmentTimerDedicated=sf10240] physicalConfigDedicated=[pdsch_ConfigDedicated=[p_a=dB0]
pusch_ConfigDedicated=[betaOffset_ACK_Index=10 betaOffset_RI_Index=10 betaOffset_CQI_Index=8] uplinkPowerControlDedicated=[p0_UE_PUSCH=0 deltaMCS_Enabled=en0
accumulationEnabled=TRUE p0_UE_PUCCH=0 pSRS_Offset=7] cqi_ReportConfig=[cqi_ReportModeAperiodic=rm30 nomPDSCH_RS_EPRE_Offset=0 setup=[cqi_PUCCH_ResourceIndex=0
cqi_pmi_ConfigIndex=39 widebandCQI ri_ConfigIndex=1 simultaneousAckNackAndCQI=FALSE]] explicitValue=[transmissionMode=tm3 n2TxAntenna_tm3=11 release]
schedulingRequestConfig_setup=[sr_PUCCH_ResourceIndex=4 sr_ConfigIndex=15 dsr_TransMax=n64]]]]]]

00130.0045 {UE_163_162_149_25_3_1.1} cue_r8_lte S CUE_RBConfig_Req=[UE_Id=1 Add=[sRB_Id=1 PDCP_Info=[Port=3 RLC_Mode=AM CNF=No] AM_UL_RLC_MODE=[Timer_Poll_Retransmit=ms100
Poll_PDU=pInfinity Poll_Byte=kbInfinity Max_Retx_Threshold=t16] AM_DL_RLC_MODE=[Timer_Reordering=ms50 Timer_Status_Prohibit=ms0] MAC_Info=[LCID=1 Logical_channel_priority=1
P_Bit_Rate=infinity Bucket_Size_Duration=spare2 Logical_channel_group=0]] TrCh_config=[MAC_main_config=[maxHARQ_tx=n5 periodicBSRtimer=infinity Time_Alignment_Timer=sf10240
retxBSRtimer=sf320 tti_bundling=No]] PHY_Info=[CQI_reporting_info=[reporting_mode_aperiodic=rm30 indexed_periodic_reporting_info=[resource=0 cqi_pmi_config_index=39 widebandCQI
ri_config_index=1]] Indexed_Scheduling_Info=[resource=4 sr_config_index=15 dsr_TransMax=n64] PDSCH_info=[P_a=db0] PUSCH_info=[ack_index=10 ri_index=10 cqi_index=8]
Antenna_info=[transmissionMode=tm3] Uplink_Power_Control=[Po_ue_pusch=0 deltaMCSEnabled=en0 accumulationEnabled=on Po_ue_pucch=0 pSRS_Offset=7]]]

00130.0089 {UE_163_162_149_25_3_1.7} nas_rrc_r8_lte S EMMAS_DATA_REQ=[UE_Id=1 L3_PDU=[EMM_attach_request nas_key_set_identifier_value=[tsc_bit=0_security_context
nas_key_set_id=7] EPS_attach_type=EPS_Attach GUTI_or_IMSI=[IMSI_emm=[Identity_digits_IMSI='222017700012001']] UE_network_capability_lc=[EEA_0_to_7=224 EIA_1_to_7=224
UEA0_to_UEA7=128 UCS2_UIA_1_to_7=128 SRVCC_ISR=0] ESM_message_container_lv=[ESM_PDN_connectivity_request EPS_bearer_id=0 Procedure_transaction_id=1 PDN_type=IPv4
Request_type=initial_attach ESM_information_transfer_flag=ESM_information_transfer_required]]]
00130.0090 {UE_163_162_149_25_3_1.1} rrc_r8_lte S Data_Req_UE UE_Id_LCId=[UE_Id=1 sRB_Id=1] Data=[rrcConnectionSetupComplete=[rrc_TransactionIdentifier=2
rrcConnectionSetupComplete_r8=[selectedPLMN_Identity=1 dedicatedInfoNAS=$0741710829221077001002100 5E0E0808000000050201D011D1]]]
00130.0099 {UE_163_162_149_25_3_4.11} ixcatprim R tcp_data_indication conn_id=14 address=[ip_text='10.0.3.4' port=32794] local_ip_text='10.0.3.4' local_port=5678
i=[DCT_Start=[protocol_version=2 transaction_id=2 subscriber_id=31 scenario_id=Attach apn_id=1 cell_id=441 attach_retry=0]]
```

Fig. 4.4 Log file (.dct) for IxLoad

4.2.3 Test Results

To appreciate how much the Control Plane load affects the capabilities of the eNodeB we decide to measure the latency and the number of failures when establishing the RRC connection. Mainly, RRC connections can fail by three reasons.

The first one is because RACH procedure fails (RRC Connection NOT RACHED). Network knows when UE will send the RACH even before UE sends it because eNodeB tells UE when the UE is supposed to transmit the RACH. (If UE fails to decode properly the network information about the RACH, eNodeB will fail to detect it even though UE sends RACH). All the information related to the RACH procedure are sent within the System Information Block2 (SIB2) message. Details about the information element carried out in SIB2 messages can be found in 3GPP 36.331. When a UE transmit a RACH Preamble, it transmits with a specific pattern and this specific pattern is called a "Signature". The location of the RACH in the frequency/time resource grid is notified to the mobile via the downlink Broadcast Channel (BCH). In each LTE cell, total 64 preamble signatures are available and UE select randomly one of these signatures. The eNodeB set the maximum number of non-dedicated access preambles that can be used with a parameter called "Number Of RA-Preambles". When and where the UE is supposed to send the RACH is depending on a parameter called "PRACH Configuration Index". For example, if the UE is using "PRACH Configuration Idex = 0", it should transmit the RACH only in EVEN number SFN(System Frame Number), according to 3GPP specification TS36.211 - Table 5.7.1-2 (ADD REF). Table 4.2 displays the main parameter values in our setup.

Table 4.2 RACH Procedure Configuration

Number of RA-Preambles	40
PRACH Configuration Index	3
Root Sequence Index	12
RACH Response Window Size (ms)	10

Once the preamble is transmitted, the UE shall monitor the PDCCH for Random Access Response (RAR) identified by the RA-RNTI, in the RAR window wich starts at the subframe that contains the end of the preamble transmission plus three subframes and has the length of "ra-Response Window Size". In our case "ra-Response Window Size = 10" and this means that the maximum time difference between the end of RACH preamble and RACH Response is only 12 subframes (12 ms) which is pretty tight timing requirement. Thus, we can obtain a RACH failure if we don't receive the RAR in 12ms.

The establishment of the RRC connection may also fail when the T300 timer expiry. The RRC Connection Request is transferred using SRB 0 on the Common Control Channel

(CCCH) because neither SRB 1 nor a Dedicated Control Channel (DCCH) have been setup at this point. The uplink Resource Block allocation for the RRC Connection Request message is signalled by the RAR message. It includes a UE identity and an establishment cause. The UE starts the T300 timer after transmitting the RRC Connection Request message. The value of T300 is broadcast within SIB2. LTE uses the T300 timer to define how long the UE waits for a response to the RRC Connection Request message. The establishment procedure fails if T300 expires before receiving an RRC Connection Setup message. In our test T300 is set to 200ms. The UE proceeds to wait for an RRC Connection Setup message from the eNodeB. The PDCCH specifies the set of PDSCH Resource Blocks used to transfer the RRC Connection Setup message. The RRC Connection Setup message is transferred using SRB 0 on the CCCH. The RRC Connection Setup message contains configuration information for SRB1. This allows subsequent signalling to use the DCCH logical channel.

The eNodeB may also reject the connection establishment request as a result of congestion. Upon receiving an RRC Connection Reject message, the UE starts the T302 timer set with the wait time included in the reject message. The reject message is returned to the UE using SRB0 on the CCCH logical channel. The UE is not allowed to send another RRC Connection Request until T302 expires. In that case, the UE is permitted to send an RRC Connection Request to the new cell. We set T302 to be 3000ms.

Table 4.3 Test Results with 100 UE

Inactivity Timer	Request	Setup	Setup Complete	RACH Failure	T300 Failure	Rejected	Failure Percentage
70.554	5734	5734	5734	1	0	0	0.017%
61.000	6436	6436	6436	1	0	0	0.015%
12.154	19565	19565	19565	16	0	0	0.081%
3.275	58647	58647	58647	53	0	0	0.090%
2.035	91641	91641	91641	114	0	0	0.124%

The eNodeB allows us only to use integer value for the inactivity timer and the minimum value is 5 seconds. The minimum value is a vendor specific parameter. This represents a limitation for the test cases with RRC_{IT} lower then 5 seconds. Nevertheless, since we are mainly interested in the rate with which RRC connection request are generated, this limitation doesn't affect our results.

So for each test we set the inactivity timer according to the formula:

$$InactivityTimer = max(5, floor(RRC_{IT})) \tag{4.1}$$

Test with 100 UEs

Figure 4.5 show the histogram and cdf of the time required to accomplish the procedure for two different inactivity timer values. For the same values, figure 4.6 show the number of RRC connection attempts per second.

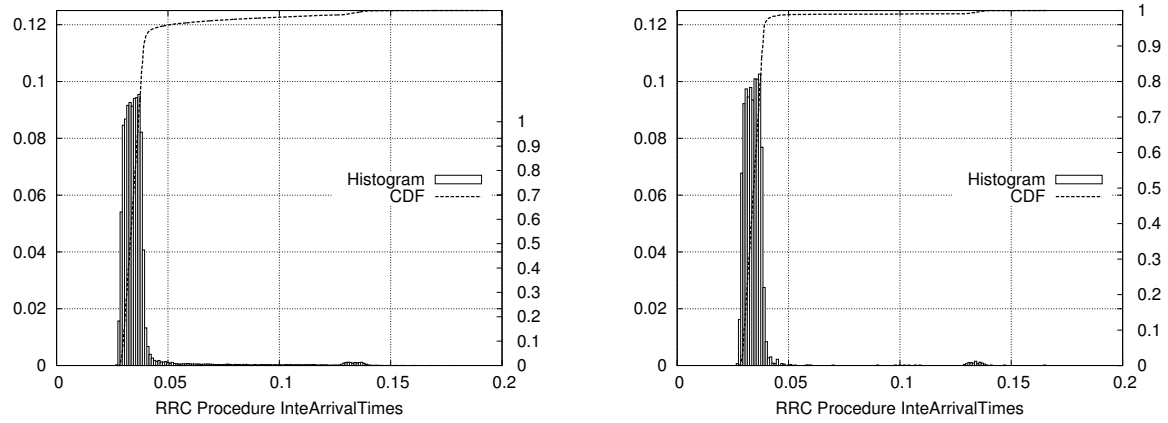

Fig. 4.5 Latency for RRC_{IT} = 70.544s (left) and 2.035s (rigth) - 100 UEs

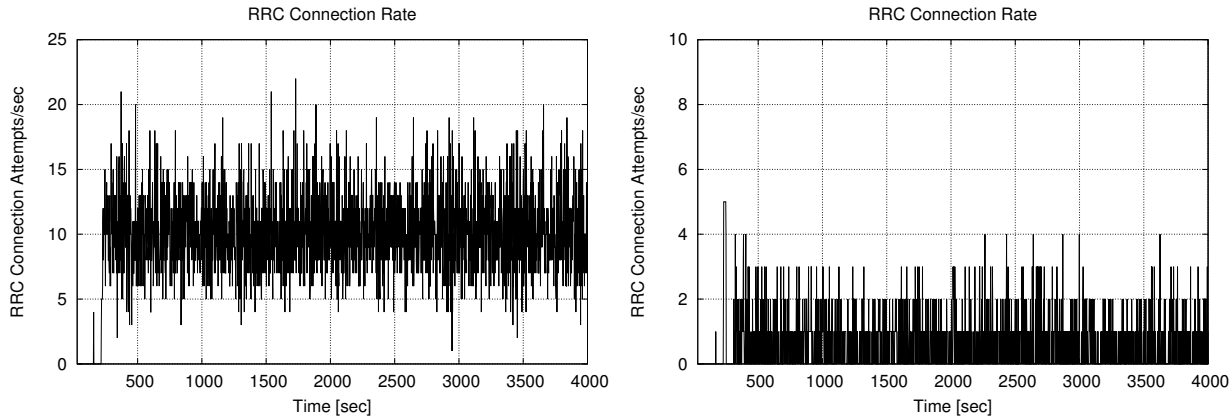

Fig. 4.6 Rate for RRC_{IT} = 70.544s (left) and 2.035s (rigth) - 100 UEs

Table 4.3 displays the main results for each test case. The first important outcome is that we never received a failure due to T300 timeout or RRC rejection message.

Figure 4.7 summarizes the latency-related results.

On each box, the central mark is the median, the edges of the box are the 25th (q_1) and 75th (q_3) percentiles, the whiskers extend to the most extreme data points not considered outliers, and outliers are plotted individually. Points are drawn as outliers if they are larger than q_3 + w*(q_3-q_1) or smaller than q_1 - w*(q_3 - q_1). We set w equal to 1.5. The default

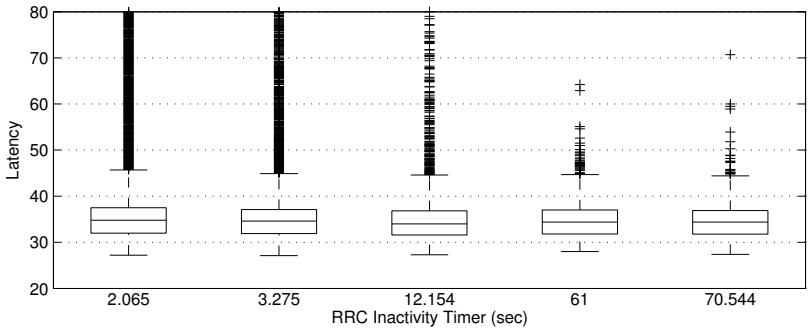

Fig. 4.7 Latency for different Inactivity Timer value - 100 UEs

of 1.5 corresponds to approximately +/-2.7*std and 99.3 coverage if the data are normally distributed.

As we can see, the results for 70.544 and 61 seconds are pretty similar: all the RRC procedures are completed within 0.15 ms, and we got only 1 RACH failure that represents the 0.017% and 0.015% off all attempts, respectively. The 75th percentile is pretty similar as well: 0.0369 and 0.0370 seconds. The mean values are 0.0354 and 0.0360 seconds.

Setting the RRC_{IT} equal to 12.154 seconds, the percentage of RACH failure due to RAR timeout increases. The mean time value for the procedure completion is 0.0359 that is slightly greater than the previous ones and this is due to the larger number of outliers, evidenced in the boxplot figure.

Further increasing the RRC request rate, involves an augmentation of the RACH failures, which represent the 0.09% and 0.125% of all the attempts for the RRC rate that we would have with an RRC_{IT} equal to 3.275 and 2.035, respectively. If UE doesn't receive RACH Response at the first trial, it just retries the procedure. The Backoff Indicator (BI) is a special MAC subheader (sent with RAR) that carries the parameter indicating the time delay between a PRACH and the next PRACH. We set the eNodeB with the BI parameter set to 2, that means that the UE can resend the preamble randomly between 0 and 20ms. See 3GPP 36.321 for more details.

In our tests one UE never experiences two consecutive RACH failures. This mean that, with an hypothetical inactivity timer of 2 seconds, in the worst case scenario that we need to reestablish the RRC connection immediately after realising it, if we get a RACH failure, we add a delay for packet transmission of 37.2ms (average latency) + 10ms (average backoff time). For a default bearer with QCI=9 this is an acceptable value, given that the standardized packet delay budget is 300ms (table 6.1.7 from TS 23.203).

Test with 400 UEs

We repeat the same tests increasing the number of UE that can be RRC connected at the same time. Now, the maximum number of allowed UE is set to 405. Each test lasts 45 minutes.

Table 4.4 Test Results with 400 UEs

Inactivity Timer	Request	Setup	Setup Complete	RACH Failure	T300 Failure	Rejected	Failure Percentage
70.554	5896	5896	5896	1	0	0	0.017%
61.000	6289	6289	6289	0	0	0	0.000%
12.154	18385	18385	18385	10	0	0	0.054%
3.275	52217	52217	52217	52	0	0	0.099%
2.035	73404	73404	73404	140	0	0	0.191%

In comparison with previous results, we obtained the same percentage of failure, approximately, and all the failures are due to RAR timeout. Only for the last test we experience a significant increase of 54%.

Looking at latency variations, we notice more evident differences among different test configurations. In figure 4.8 we limit the view at 100 ms, to appreciate these differences.

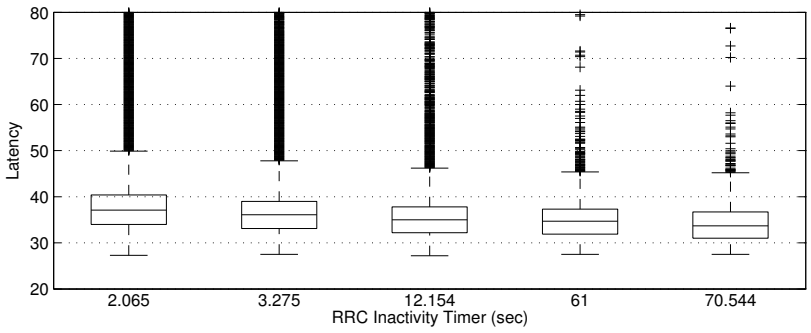

Fig. 4.8 Latency for different Inactivity Timer value - 400 UEs

As we can see, as we move from the highest to the lowest value, both the 75th percentiles and the upper whiskers are larger, indicating that in heavy load condition and with a larger number of UE the eNodeB requires, on average, more sub-frames to schedule in downlink the RRC Connection Setup and in uplink the RRC Connection Setup Complete. At this point, it is worth noting the following: as we said before the PDCCH specifies the set of PDSCH and PUSCH Resource Blocks used to transfer the RRC messages. Within the PDCCH the eNodeB use DCI1 message to tell one UE where are the PDSCH Resource Blocks addressed to him. But in the same sub-frame, the eNodeB could have to assign Uplink grants to a certain number of UEs, and this is made sending DCI0 in the PDCCH (e.g. to send the RRC

Setup Complete message). Therefore, the total number of UEs that can be addressed in one sub-frame is physically limited by the number of physical resource on the PDCCH. The dimension of the PDCCH in each sub-frame is determined by the information carried into the Physical Control Format Indicator Channel (PCFICH). Mapped to the first OFDM symbol in each of the downlink sub-frame, it carries the number of OFDM symbols used for PDCCH and PHICH. In our test PCFICH is set to 2, that means that 2 OFDM are used for PDCCH. In Annex A the way to calculate the maximum UE number that can be scheduled within one TTI is proposed.

That is not all: depending on vendor implementation, the eNodeB can be configured with the maximum number of DCIs that can be sent within one sub-frame. During our test, we use the vendor default configuration, that provides 9 maximum DCI in downlink, and 7 maximum DCI in uplink per TTI. Vendor default implementation could limit the number of possible DCI per TTI due to scheduling implementation, allowed load (e.g. we may not want to allocate 100% resources), the usage of GBR and the Semi-Persistent scheduling, that does not require transmission on DCI, but allow UE to reuse preallocate PRB on the PDSCH.

Every second, we collect the maximum number of DCI transmitted in one TTI, and the total number of DCI transmitted in the previous one thousand TTI. Figure 4.9 (up) shows the average value of the maximum number of DCI in one TTI per polling interval, across all the tests and all the inactivity timer value. Figure 4.9 (down) shows the average number of transmitted DCI per polling interval. The dotted and dashed lines distinguish UL and DL DCI, whereas the 'o' and 'x' markers refers to the tests with 100 and 400 UEs, respectively.

In the figures, it is evident that decreasing the inactivity timer, can occur to reach the maximum allowed number of DCI per TTI. This means that if the eNodeB has to schedule more UEs, for example more than 9 UEs in downlink, it needs more TTI to address all the UEs. The probability to postpone one DCI to the next TTI, is obviously larger if the number of DCI to be sent increases.

This analysis confirms our intuition that the congestion on physical channel is the main reason for the growth of the RRC latency.

In particular, in the worst test case with 400 UEs and inactivity timer of 2.054, the eNodeB generates on average 7 DL and 6.25 UL DCI each TTI, and in one second at least one TTI reaches the maximum number of allowable DCI. In this very congested scenario we notice the highest growth for the RRC latency, with the mean value to complete the RRC procedure equals to 40.7 ms, and about the 25% of all attempts needs more than 50 ms to setup the radio connection.

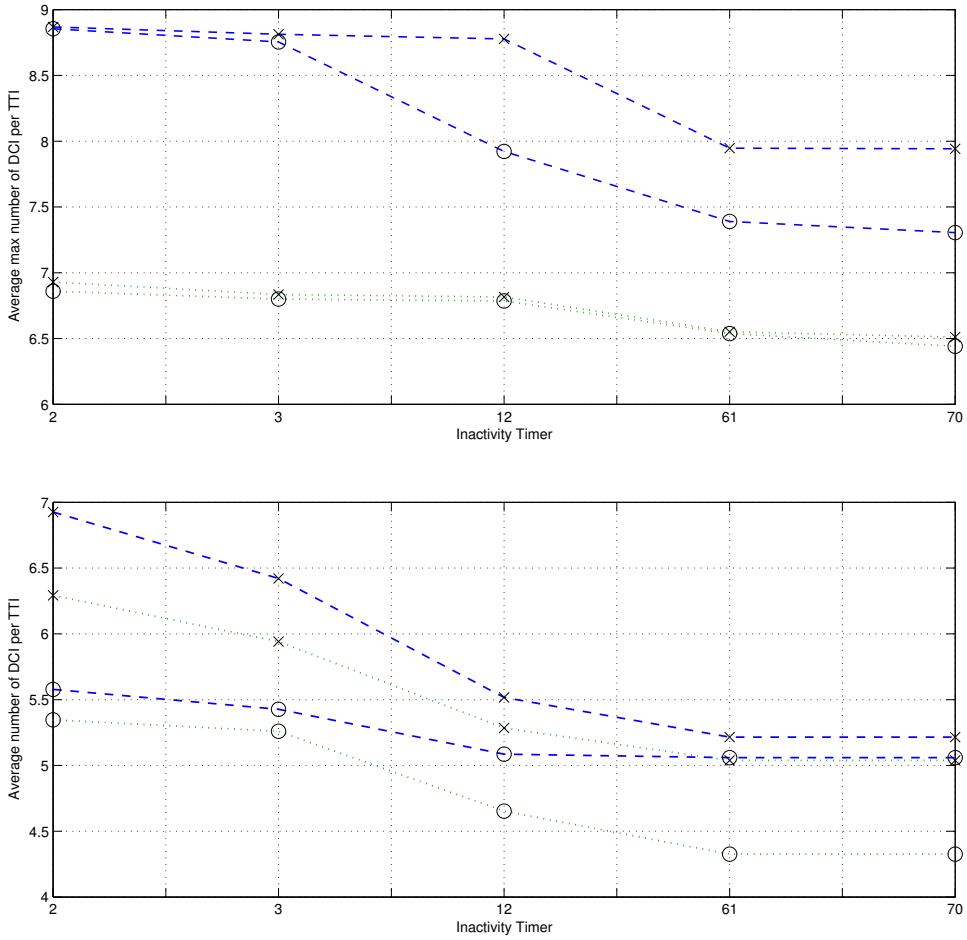

Fig. 4.9 Maximum number of DCI in one seconds (up) and average number of DCI per TTI (down)

4.2.4 Other Considerations

Here we present other problems that may come out and that are not addressed by this work, but that can be evaluated in future works.

First of all, during our test we configure the MME to perform NAS procedures only as part of the attach procedure. As we already pointed out in section 3.4, NAS procedures are optional and may occur depending on internal MME policy. Performing Security and Authentication procedures imply to load further the eNodeB and to occupy physical channel with additional control plane messages.

This paper focus on eNodeB performance only, nothing has been mentioned about MME. The MME is key control-node for the LTE access-network. It can be addressed by several eNodeB, basing on local policy and network topology. Diminishing the inactivity timer

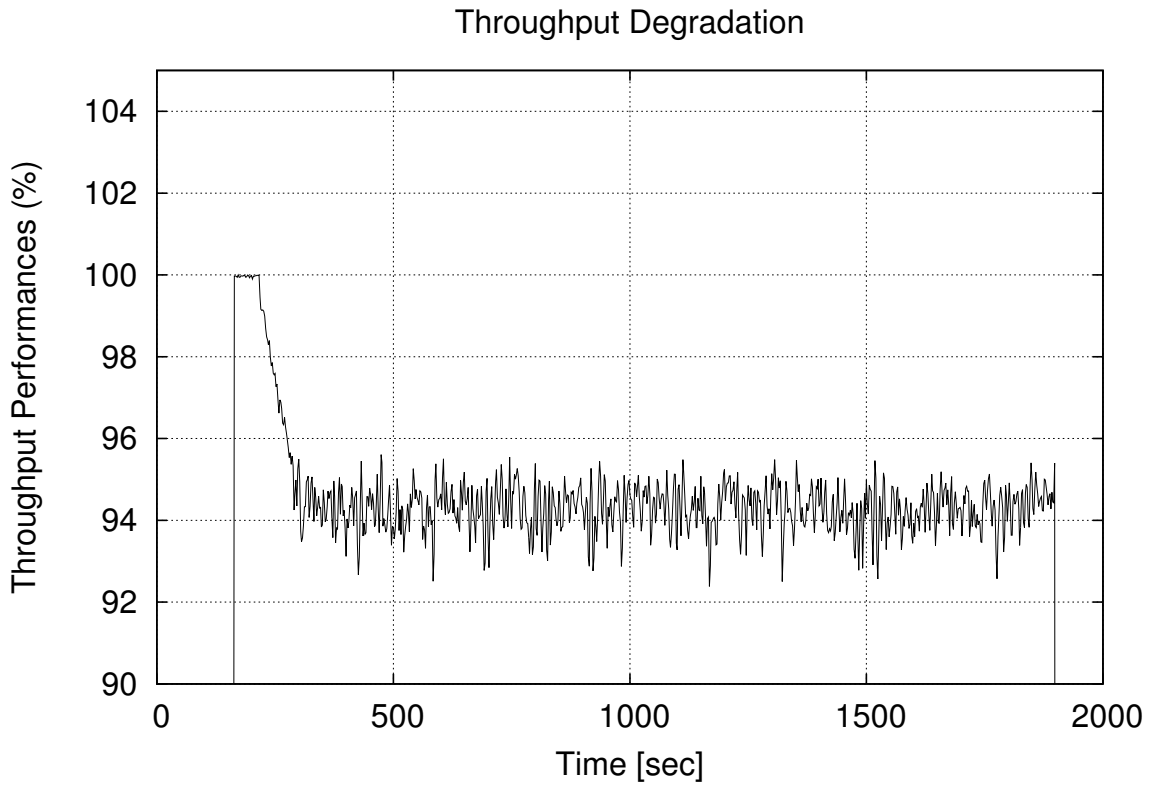

Fig. 4.10 Impact on UP throughput for RRC_{IT} = 2.035s - 400 UEs

impacts also the signalling load forwarded to the MME, that may suffer for a large number of request coming from all the eNodeB attached to it. According to vendor's evaluation, MME may represent the bottleneck that leads performance degradations.

Our work has been dedicated to evaluate the impact of signalling load on eNodeB performance. Nothing has been said about how control plane could impact data plane throughput performance. Here we'd like to provide some suggestion and results. According to 3GPP TS 36.322, paragraph 5.1.3.1.1, SRB has precedence over DRB at RLC level: the transmitting side of an Acknowledgment Mode (AM) RLC entity shall prioritize transmission of RLC control PDUs over RLC data PDUs. The transmitting side of an AM RLC entity shall prioritize retransmission of RLC data PDUs over tx of new AMD PDUs. This means, that in congested network, if there are buffered data that belong to different UEs, the eNode shall prioritize control plane traffic among data plane traffic. With the constraint of the maximum number of the allowed DCI in PDCCH, the eNodeB will delay data plane traffic transmission, decreasing the overall throughput.

Control plane and data plane share the PDSCH/PUSCH to transmit transport blocks, therefore increasing signalling load will automatically affect throughput performance. This is neglectable in normal conditions, but is not automatically true in heavy load conditions.

In figure 4.11, we show all the messages exchanged on the radio interface to setup/release the RRC connection. Some of the associated layer 1 control aspects are also included. MAC PDU sizes are shown without padding or sub-headers padding. CQI/CSI transmission may also be present after the RRC connection setup messages, but they are not explicitly shown in the picture. In bold, the messages that are transmitted on the PDSCH/PUSCH channels are highlighted. In particular, each RRC establishment/release procedure involves nine messages in downlink that use the PDSCH, and five messages in uplink that use the PUSCH. We assume that all messages are received without errors (i.e. no retransmissions).

We performed a test to measure the impact of CP traffic only, in our worst case scenario (i.e. minimum inactivity timer). Test design is the same as the one in 4.2.3 and 4.2.3, except for "HTTP-Users" which don't perform data plane traffic: they establish the radio connection and send a dummy Service Request without sending any packet. In background 5 UEs generate UDP traffic to fulfill the available bandwidth.

Figures 4.10 shows layer 7 throughput, normalized to the maximum value. After few seconds, as soon as the HTTP-Users start requesting the radio bearer, the throughput decrease on average of 6%. This means that we loose approximately 3 Mbps, for a system with 10MHz bandwidth.

4.3 RAN Overload: Machine-to-Machine and Human-to-Human Communication

Since LTE-Advanced evolves from LTE, it is still highly optimized and suitable for legacy H2H communications such as voice calls, video streaming, online gaming, social networking and web surfing. The requirements of H2H communications are high data rates, mobility, and human quality of service and experience. M2M communications desire a very different set of requirements than H2H communications because they are mainly characterized by a high device density in a cell, small amounts of payload, machine-originated communications and low traffic volumes per machine.

Laya et al. [38] described the main differences between H2H and M2M communications, which can be summarized as follow:

- Traffic Direction:

Step	↑ UL ↓ DL	Contents (of MAC PDU or L1 control)	MAC PDU Size (Bytes)	
			UL	DL
1	↑	Preamble	--	--
2	↓	Random Access Response (+PDCCH DL grant)	--	8
3	↑	RRC Connection Request	7	--
4	↓	PHICH ACK	--	--
5	↓	Contention Resolution CE (+PDCCH DL grant)	--	7
6	↑	PUCCH ACK	--	--
7	↓	RRC Connection Setup (+PDCCH DL grant)	--	30
8	↑	PUCCH ACK	--	--
9	↑	Scheduling Request	--	--
10	↓	PDCCH UL grant	--	--
11	↑	RRC Connection Setup Complete (inc. NAS Service Request) + PHR + short BSR	20	--
12	↓	PHICH ACK	--	--
13	↓	RLC Status PDU (+PDCCH DL grant)	--	3
14	↑	PUCCH ACK	--	--
15	↓	Security Mode Command (+PDCCH DL grant)	--	11
16	↑	PUCCH ACK	--	--
17	↑	Scheduling Request	--	--
18	↓	PDCCH UL grant	--	--
19	↑	Security Mode Complete + RLC Status PDU + PHR + short BSR	17	--
20	↓	PHICH ACK	--	--
21	↓	RLC Status PDU (+PDCCH DL grant)	--	3
22	↑	PUCCH ACK	--	--
23	↓	RRC Connection Reconfiguration (+PDCCH DL grant)	--	45
24	↑	PUCCH ACK	--	--
25	↑	Scheduling Request	--	--
26	↓	PDCCH UL grant	--	--
27	↑	RRC Connection Reconfiguration Complete + RLC Status PDU + PHR + short BSR	19	--
28	↓	PHICH ACK	--	--
29	↓	RLC Status PDU (+PDCCH DL grant)	--	3
30	↑	PUCCH ACK	--	--
31	↓	RRC Connection Release (+PDCCH DL grant)	--	10
32	↑	PUCCH ACK	--	--
33	↑	Scheduling Request	--	--
34	↓	PDCCH UL grant	--	--
35	↑	RLC Status PDU	3	--
36	↓	PHICH ACK	--	--
Total Bytes			**66**	**120**
Number of occupied subframes			**18**	**18**

Fig. 4.11 RRC Connection Setup/Release Sequence - Table 5.2.1-1 [7]

  - M2M involves mainly uplink data to report sensed information. For some applications, symmetric uplink and downlink capacity is needed in order to allow the dynamic interaction between sensors and actuators
  - H2H traffic is mostly downlink; although uplink traffic is increasing over the last year , human still download more than they upload

- Message Size
  - M2M: the size of the messages is generally very short (e.g. few nits as part of a reading meter)
  - H2H: the size of the messages is generally big and high variable, due to application such as multimedia, real time transmissions, including video streaming

- Connection and Access Delay
  - many M2M applications will be based on duty-cycling, for example having device sleeping and waking up from time to time to transmit data
  - Human-based traffic ten to be very demanding once the connection is established. However, although not desirable, longer connection delays are well tolerated

- Transmission Periodicity
  - this will be very variable for MTC
  - Human traffic is very random and asynchronous in nature, In addition, the frequent transmission of control information is required to ensure high throughput and good delay performance

- Mobility
  - for many M2M applications, mobility is not a major concern: some applications may not have mobility at all (e.g. sensors for metering)
  - mobility management and exchange of location information is required to ensure seamless connectivity

- Information Priority
  - some M2M may transmit critical information and thus require very high priority level
  - in general, there is no major differentiation between users in term of priority, but only on the applications running on their handset devices

- Amount of Device
  - hundreds or thousands of device per connection point
  - at most few hundreds of device per connection point

RAN overload has gained a lot interest in research communities, mainly due to the advent of M2M Type Communication (MTC). In TR 37.868 ([8]), 3GPP defines six key performance indicators to evaluate the goodness of the new proposals that intend to improve the operation of RACH. And these are the following:

- Collision probability, defined as the ratio between the number of occurrences when two or more MTC devices send a random access attempt using exactly the same preamble and the overall number of opportunities (with or without access attempts) in the period.
- Access success probability, defined as the probability to successfully complete the random access procedure within the maximum number of preamble transmissions.
- Statistics of number of preamble transmissions, defined as the CDF of the number of preamble transmissions to perform a random access procedure, for the successfully accessed MTC devices.
- Statistics of access delay, defined as the CDF of the delay for each random access procedure between the first RA attempt and the completion of the random access procedure, for the successfully accessed MTC devices.
- Statistics of simultaneous preamble transmissions (for UMTS FDD), defined as the CDF of the number of MTC devices that transmit preamble simultaneously in an access slot. This serves an indirect measure of Rise over Thermal (RoT).
- Statistics of simultaneous data transmissions (for UMTS FDD), defined as the CDF of the number of MTC devices that transmit pilot or pilot AND data simultaneously in an access slot. This serves an indirect measure of Rise over Thermal (RoT).

As specified in that documentation, a large number of MTC devices are expected to be deployed in a specific area, thus the network has to face increased load as well as possible surges of MTC traffic. Network congestion including Radio Network Congestion and Signalling Network Congestion may happen due to massive concurrent data and signaling transmission. This may cause intolerable delays, packet loss or even service unavailability. Mechanisms to guarantee network availability and help network to meet performance requirements under such MTC load need to be investigated. In reference to this document, for

the collision probability we consider either the definition provided in section 6.3 ($P_c^{(RAO)}$) either in ANNEX B ($P_c^{(HTC)}$).

As part of the improvement process for the random access procedure, the following enhancements have been identified :

- Optimized MAC: aiming at applications where M2M devices transmit small amounts of data, the authors in [22] suggest removing the need to connect to the network to transmit data. The key idea is to transmit data embedded into the access process by attaching data in one of the messages involved in the Random Access procedure

- Access Class Barring: ACB is the actual mechanism exploited by LTE and LTE-Advanced system to control access to hte air interface. ACB is based on the idea that certain access classes, which are indicated by means of network broadcasting information, are not allowed to access the network in some PRACH opportunities. As specified by 3GPP a different number of classes could be defined, depending on the granularity of the control policy that is intended to deploy. The possible solutions are:

  - Individual ACB: in order to achieve more control granularity, the network shall signal how individual devices or groups of devices will sale the barring parameters
  - Extended ACB: the basic idea is that devices that belong to delay-tolerant applications are not permitted to access the network in the case of congestion, leaving the contention for devices that are delay-sensitive
  - Dynamic Access Barring: in this method the eNodeB continuously monitors the loading state of the network in order to control the number of preamble transmission in each slot. In case of RAN overload new attempts by MTC are delayed.

- Separation of RA resources: the separation of resources can be obtained either by splitting the available preambles between HTC and MTC or by allocating different slots to HTC and MTC [23]

R. Cheng et al. in [24] use the result presented in 3GPP TR 37.868 and present an analytical model to derive collision and success probability for both RAO and MTC. The analytical model has been validated by numerical results. Nevertheless, neither the analytical model nor the simulation result take into account the backoff indicator, that is used after a collision event, and the time spent in connected state by the device, if the access procedure is successfully completed.

The technical report signed as FFS (For Further Study) the impacts of H2H traffic. Indeed, for the purpose of RACH capacity evaluation, all RACH attempts are assumed to be initiated by MTC devices with no background noise caused by H2H UEs. The introduction of separate Access Class(es) for MTC devices legitimates this assumption, because it allows the network to separately control the access from these MTC, in addition to access control for other devices. Depending on the granularity of the control needed among MTC devices, either one or several Access Classes can be introduced. By the way it is worth mentioning that this is not mandatory, thus without access class control MTC and H2H devices share the RACH resource and experience the same access collision probability. Separate RACH resources can be also provided for the H2H and MTC devices: for LTE the separation of resources can be done by either splitting the preambles into H2H group(s) and MTC group(s) or by allocating PRACH occasions in time or frequency to either H2H or MTC devices. In any case, coexistence of MTC and H2H handset device is an open problem that needs to be addressed for future (LTE-Advanced, 5G) networks deployments.

M. Condoluci et al. [25] proposed a 3GPP-compliant architecture that absorbs MTC traffic via home evolved eNodeB, allowing to reduce congestion and overloading of radio access and core networks. Throughout this novel design, MTC should not affect the performance of classical H2H communication, at the cost of introducing new network devices able to communicate with existing core network devices. Conversely from this approach, our intend is to evaluate network performance in terms of random access collision probability in existing network architecture. Lo et al. [40] investigate the impact of massive M2M communication and proposed a self-optimizing overload control mechanism that can tune RACH resources according to current load status. Even in this case, they considered separately the background H2H traffic and the RA attempt generated by MTC devices.

From HTC point of view, lot of researchers have addressed the problem of energy consumption, to figure out how to increase battery lifetime. Indeed, battery lifetime represents the first limitation on smartphone usage today. In LTE systems, Discontinuous Reception represents a strategy for power savings. Jha et al. [36] and Koc et al. [37] studied how to find the best trade-off between latency and power saving through optimum DRX configuration set. Power savings can be obtained by reducing the Radio Resource Control Inactivity Timer (RRC_{IT}), which leads the User Equipment (UE) to stay in the RRC Idle state, where its RF circuitry has a near zero power consumption. On the other hand, low values of RRC_{IT} may imply a high rate of access requests to the Random-Access CHannel (RACH), which may increase the collision probability.

This section proposes an analytical model to evaluate the collision probability on the RACH as a function of the number of UEs, the number of available preambles and the

Interarrival times of the RACH Requests (IRR) of the average user. The model for the IRR of the average user is obtained from real traffic data captured at the eNodeB of a mobile operator, and is derived by emulating the RRC state machine for different RRC_{IT} settings. The output of the emulator allows reconstructing the timeseries associated with the interarrival times of RACH requests produced by the average UE. Mixture modelling is then applied to such timeseries and used to analytically estimate the RACH collision probability.

4.4 RACH Procedure

The random access procedure is detailed in 3GPP TS 36.300 [10]. Current specification states that the procedure can be triggered by any of the following events: (i) initial access from RRC Idle; (ii) RRC Connection Re-establishment procedure; (iii) handover; (iv) DL/UL data arrival during RRC Connected requiring random access procedure; (v) for positioning purpose during RRC Connected requiring random access procedure;

Furthermore, the random access procedure takes two distinct forms:

- Contention based: applicable to first four events;
- Non-contention based: applicable to only handover, DL data arrival, positioning.

This work focused on initial access from RRC idle, so from now on we'll talk about contention based RACH procedure. When a UE in the RRC Idle receives/sends a packet, a state transition from RRC Idle to RRC Connected is needed. At this point, the UE initiates the random access procedure by sending the random access channel preamble (RACH Preamble). When a UE transmits a PRACH Preamble, it transmits with a specific pattern and this specific pattern is called "signature". In each LTE cell, total 64 preamble signatures are available and UE select randomly one of these signatures.

Network knows when UE will send the RACH even before UE sends it because eNodeB tells UE when the UE is supposed to transmit the RACH. If UE fails to decode properly the network information about the RACH, eNodeB will fail to detect it even though UE sends RACH. All the information related to the RACH procedure are sent within the System Information Block2 (SIB2) message. Details about the information element carried out in SIB2 messages can be found in 3GPP 36.331 [13].

The location of the RACH in the frequency/time resource grid is notified to the mobile via the downlink Broadcast Channel (BCH).

The eNodeB set the maximum number of non-dedicated access preambles that can be used with a parameter called "Number Of RA-Preambles". The other "signatures" can be used for contention free procedure, e.g. during handover.

When and where the UE is supposed to send the RACH is depending on a parameter called "PRACH Configuration Index" (*PracConfigIndex*). For example, if the UE is using "PRACH Configuration Idex = 0", it should transmit the RACH only in EVEN number SFN(System Frame Number), according to 3GPP specification TS36.211 - Table 5.7.1-2 [9]. One LTE SFN lasts 10ms, so one UE can use the RACH channel every 20 ms. The PRACH Configuration Index and the number of preambles reserved for contention based procedure are responsible for the number of Random Access Opportunity (RAO) per second. If we defined L as the total number of RAO per second, with a PRACH Configuration Idex = 0 and 20 available preambles, we'll have L = 20x50 RAO/second.

Being a contention based procedure, this procedure can fail if two or more UEs attempt to access the RACH channel using the same RAO. In this case, both of the UE will receive the same TC-RNTI and resource allocation in the RAR (Random Access Response) messages. As a result, both UE would send the RRC connection request message through the same resource time/frequency location, to the eNodeB. At this point, one possibility is that these two signal act as interference to each other and the eNodeB decode neither of them. In this case, none of the UE would have any response (HARQ ACK) from the network and they all think that RACH process has failed and start and procedure from the beginning. The other possibility would be that the eNodeB successfully decodes the message from only one UE and failed to decode it from the other UE. In this case, just one UE will get the HARQ ACK from Network. This HARQ ACK process is called "contention resolution" process.

If a collision occurs, meaning that the UE sent a PRACH but didn't get a RAR for some reason, or UE sent a PRACH and got RAR, but the RAPID in the RAR is not for the UE, that UE has to repeat the procedure. The Backoff Indicator is a special MAC subheader that carries the parameter indicating the time delay between a PRACH and the next PRACH. Figure 4.12 shows the mac header/subheader structure carrying the backoff indicator information.

The bit field shown below, is made up of 4 bits, implying that it can carry the value from 0 to 15. Each of these value maps to a specific time value as shown in table 4.13 taken from 3GPP TS 36.321 [12]. For example, if the BI field value is 10, Backoff Parameter value is 320 ms. This means UE can send PRACH any time in between 0 and 320 ms from now. This is used to reduce the collision probability, dispersing random access across multiple successive slots.

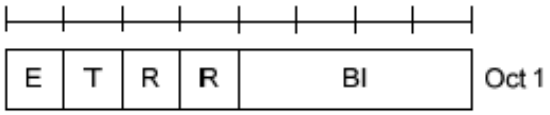

Fig. 4.12 MAC sub-header for Backoff Indicator

Index	Backoff Parameter value (ms)
0	0
1	10
2	20
3	30
4	40
5	60
6	80
7	120
8	160
9	240
10	320
11	480
12	960
13	Reserved
14	Reserved
15	Reserved

Fig. 4.13 Map for Backoff Indicator and specific time values

4.5 Modelling inter–RACH times

The first goal of our analysis is to derive a statistical model for the interarrival times of the UE RACH requests. These observations depend on the configuration of the eNodeB (specifically, in terms of RRC_{IT}) and on the traffic generated by end-users. Our starting point is the traffic data acquired by passive measurements carried out at a commercial eNodeB of a mobile operator. Upon the necessary anonymization procedures and after stripping out privacy–sensitive data, the acquired traffic dump consisted of a list of entries each one reporting the IP source and destination addresses of each observed packet along its timestamp.

To obtain the interarrival times of RACH requests for each user, a simulator of the RRC state machine has been developed. For each user (say, i), the input of the simulator is the set of the traffic interarrival times associated with it. By emulating the RRC state machine and by taking into account the RRC_{IT}, the output of the simulator is the timeseries representing the "Interarrival times between successive RACH Requests" (IRR) of the user i. We emulated the RRC state machine for different RRC_{IT} values, namely 2, 5 and 10 s. The timeseries IRR_j, $j = 2,5,10$, is obtained by merging the IRR obtained for all users i when the RRC_{IT} is set to j. The timeseries IRR_j represent the observations of interarrival times of the triggered RACH for the average user. Some statistical properties of these timeseries are shown in Table 4.5. Obviously, having the same input traffic, the emulator of the RRC state machine generates higher numbers of triggered RACHs (and lower mean IRR values) for lower RRC_{IT} values.

Table 4.5 IRR_i : Statistical Parameters

Dataset	Size	Mean (s)	Covariance (s^2)	Coef. Var.
IRR_2	22654	47.46	1.98e+04	2.96
IRR_5	13424	80.03	3.16e+04	2.22
IRR_{10}	9106	117.73	4.48e+04	1.80

The coefficient of variation (CV), which represents a standardized measure of dispersion of a probability distribution, is defined as the ratio of the standard deviation to the mean. The CV values higher than 1, as shown in the Table, suggest to consider a model based on a mixture of exponential distribution, since the exponential distribution has $CV = 1$. Hence, the considered timeseries have been modeled by i.i.d. random variables with density equal to a mixture of exponentials:

$$f(x) = \sum_{c=1}^{C} \alpha_c \lambda_c e^{-\lambda_c x} \tag{4.2}$$

where C refers to the number of components in the mixture, α_c is the mixing coefficients (with $\sum_{c=1}^{C} \alpha_c = 1$), and λ_c is the parameter of the c-th exponential component.

For each considered RRC_{IT} value, the Bayesian model selection approach has been applied to estimate the mixture parameters of timeseries IRR_j, with $j = 2, 5, 10$.

In particular, we referred to the algorithm presented in [52], where the prior distribution assumes that α_c and λ_c are independent. The symmetric Dirichlet distribution is used as a prior of α_c, whereas the gamma distribution $\Gamma(a_0, b_0)$ is selected as the conjugate prior distribution on λ_c. As suggested in [52], a_0 is set to a small value (we set $a_0 = 0.1$) and b_0 is chosen in such a way that the prior mean is matched to the mean of the data, $\overline{Y}$, i.e. $b_0 = a_0 * \overline{Y}$. The Markov Chain Monte Carlo (MCMC) algorithm described in [32] has been used for the estimation of the model parameters for different number of components.

Different techniques can be used to estimate how to choose the number of components C, such as those based on Akaike information criterion (AIC) or its Bayes Information Criterion (BIC) variant, as well as other different classification–based information criteria, which are minimized for the optimal model among a set of potential models (see [32] for details).

Any of these criteria should be based on the maximum likelihood (ML) estimator, and require iterative runs of the algorithm used for the estimation of model parameters. In figure 4.14, the BIC values evaluated at the ML estimator for diverse C are shown for the three sequences IRR_j. The figure suggests that $C = 3$ is the optimal value for IRR_5 and IRR_{10}, whereas 4 components are needed for the IRR_2. The estimated parameters are summarised in table 4.6. The Table shows that in all timeseries, more than 74% of samples can be modelled

Table 4.6 Estimated model parameters

Dataset	α_c	λ_c
IRR_2	0.7782, 0.0955, 0.0102, 0.1160	0.0804, 0.0192, 0.0013, 0.0046
IRR_5	0.1935, 0.0149, 0.7916	0.0043, 0.0013, 0.0334
IRR_{10}	0.0291, 0.2269, 0.7441	0.0015, 0.0039, 0.0185

with exponential distribution with a mean value around 50 s or less (λ_c are around 0.02 or higher), whereas the remaining observations are in the range of 250 s (i.e., λ_c around 0.004) and 750 s ((i.e., λ_c around 0.0013). Figure 4.15 shows the quantile–quantile (QQ) plot for the different timeseries. Each subfigure reports the QQ curves obtained by comparing the actual dataset to the "Exponential" model, and to the "Mixture" model. In the first case, the parameters are set according to the mean values reported in Table I, while in the second case according to Table II. To immediately visualize the quality of the fitting results, the reference "Best Fitting" curve is also reported. The figure shows that for the IRR2 and IRR5 timeseries, the points of the curve "Mixture" lay very close to the "Best Fitting" curve, whereas they deviate for high quantile values in the case of IRR10. Conversely, we observe that the points of the "Exponential" curve are always far from the "Best Fitting" curve.

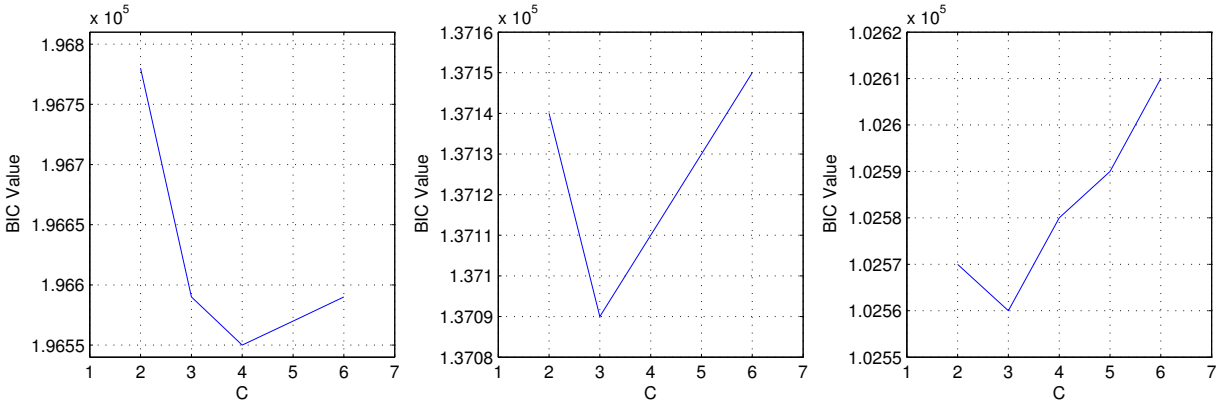

Fig. 4.14 BIC vs C for $RRC_{IT} = 2, 5, 10s$ (Left to Right)

4.6 RACH Collision Probability: Analytical Model

In this section we derive the analytic expression of the probability of collision for RACH procedures initiated by different stations under a quite general RACH request model. The result is then specialized for the mixture model (4.2) and used next in the performance evaluation section 4.6.1.

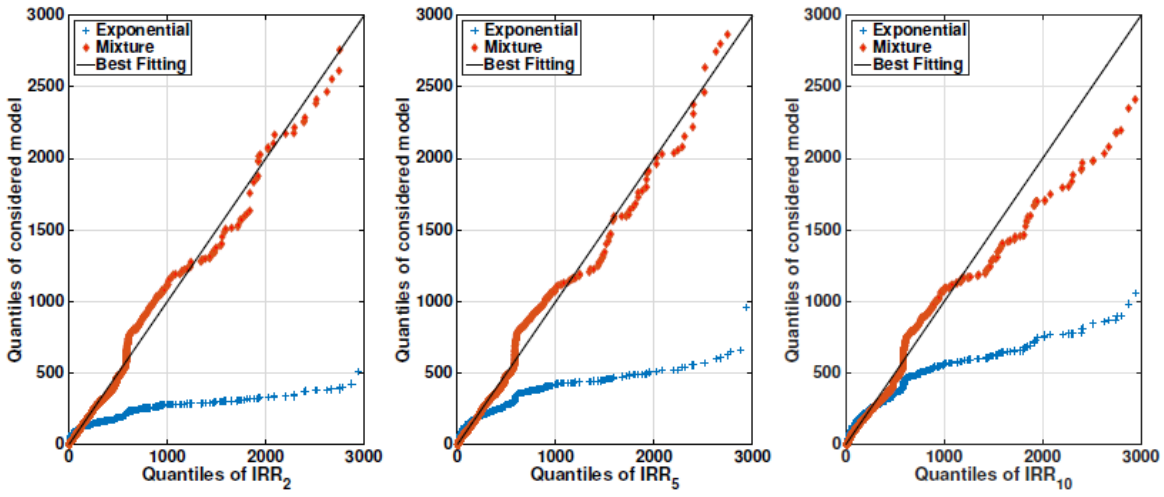

Fig. 4.15 QQplot for Inactivity Timer = 2,5,10 (Left to Right)

To this aim, let us first indicate the number of stations in the RRC Idle state with N and the number of available preambles with k. In addition, let T be the duration of each timeslot. At each timeslot, any station i (with $i \leq N$) in RRC Idle state may place a RACH request by requesting a given preamble j, with $j \leq k$.

The problem of successfully select a transmission preamble is "nearly" equivalent to that of successfully switching a cell in an unbuffered $N \times k$ crossbar switch with Bernoulli arrivals and uniform routing. The analogy is not perfect, however, in that multiple cells colliding on the same switch outlet result in the transmission of a cell only, whereas, in the RACH case, no transmission occurs.

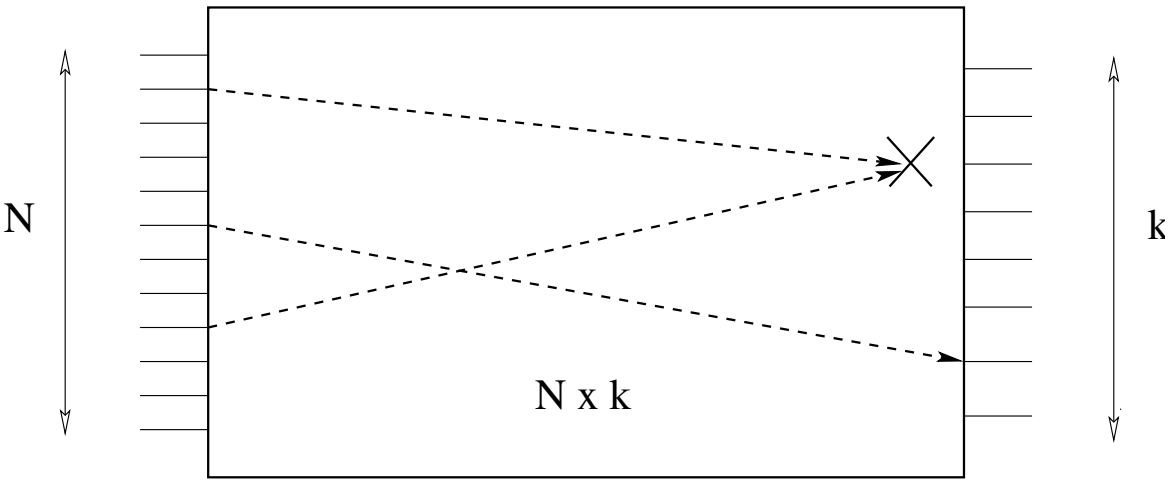

Fig. 4.16 Switching analogue to RACH request operations

Figure 4.16 visualizes the equivalent system: RACH procedure requests arrive at the N inlets of the switch by requesting for a preamble, represented by one of the k outlet of the switch.

Let $A_i(n) \in \{0,1\}$ be the number of requests placed by station i at time nT and with $A_{i,j}(n) \in \{0,1\}$ the number of requests placed by station i by selecting preamble j.

Assume now that stations in RRC Idle state may each independently place a request with probability p, i.e. $\Pr\{A_i(n) = 1\} = p$ and that preambles are selected uniformly at random with probability $1/k$. As such: $\Pr\{A_{i,j}(n) = 1\} = \frac{p}{k}$. Consider now the total number of requests $X_j(n)$ to preamble j at time nT: $X_j(n) = \sum_{i=1}^{N} A_{i,j}(n)$. At each time n, and for each preamble, RACH collision occurs whenever $X_j(n) \geq 2$. The random variable $X_j(n)$ has binomial distribution with parameter p/k, and the number of RACH request failures $L_j(n)$ is:

$$L_j(n) = \begin{cases} 0 & \text{if } X_j(n) < 2 \\ X_j(n) & \text{otherwise} \end{cases} \tag{4.3}$$

Note that, since we assume the system at the statistical equilibrium, the dependence on time can be safely omitted. Hence, the mean number of RACH collisions to each single preamble is:

$$\begin{aligned} \mathbb{E}\left[L_j\right] &= \sum_{n=2}^{N} n \Pr\{X_j = n\} \\ &= \sum_{n=2}^{N} n \binom{N}{n} \left(\frac{p}{k}\right)^n \left(1\frac{p}{k}\right)^{N-n} \\ &= \frac{Np}{k}\left(1 - \left(1 - \frac{p}{k}\right)^{N-1}\right) \end{aligned} \tag{4.4}$$

and the overall RACH collision probability $P_C^{(HTC)}$ is:

$$\begin{aligned} P_C &= \frac{\mathbb{E}\left[L_j\right]}{\mathbb{E}\left[X_j\right]} \\ &= \frac{\frac{Np}{k}\left(1 - \left(1 - \frac{p}{k}\right)^{N-1}\right)}{\frac{NP}{k}} \\ &= 1 - \left(1 - \frac{p}{k}\right)^{N-1} \end{aligned} \tag{4.5}$$

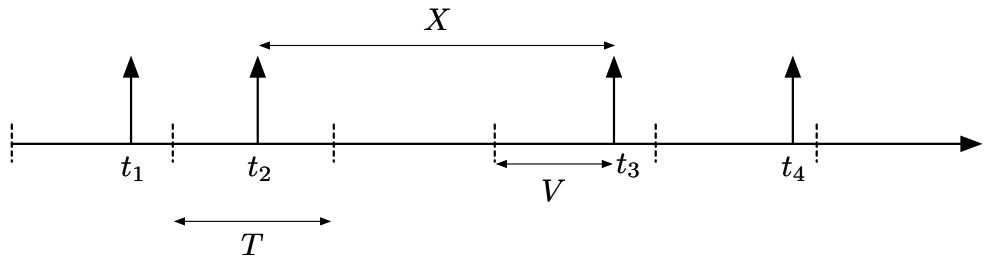

Fig. 4.17 RACH requests arrival process

From the point of view of an external observer, instead, it is easy to prove that the collision probability of a generic RAO is given by:

$$\begin{aligned} P_C^{(RAO)} &= 1 - \frac{\text{\# of idle RAOs in } \tau \text{ s.}}{\text{\# of RAOs in } \tau \text{ s.}} + \\ &- \frac{\text{\# of successful RAOs in } \tau \text{ s.}}{\text{\# of RAOs in } \tau \text{ s.}} \\ &= 1 - \left(1 - \frac{p}{k}\right)^N - \frac{Np}{k}\left(1 - \frac{p}{k}\right)^{N-1} \end{aligned} \tag{4.6}$$

Equations (4.5) and (4.6) assume the probability p of a RACH request for any idle station at each timeslot is known. Hence, the next step in the derivation is to analytically find the value of p.

In the previous section, we assumed the process of RACH requests to be a point process (figure 4.17) $\{t_1, t_2, \dots\}$ in which the inter–arrival times $\{X_1, X_2, \dots\}$ form a sequence of i.i.d. random variables each of them with probability density function given by (4.2). More generally, by indicating the common pdf with $f_X(x)$ and the common probability distribution with $F_X(x)$, the overall arrival process is a *renewal process* [26] and the arrival events are called *renewals*.

Under this assumption, finding the value of p is equivalent to compute the probability that one renewal occurs in the generic time interval $((n-1)T, nT]$. Notice that, depending on the renewal distribution, the inter–arrival times can be arbitrarily small so that more than one renewal event may occur in the same timeslot. However, this phenomenon is not possible in practice and, as it will be shown, its probability becomes analytically negligible for physically reasonable model parameters and small timeslots.

In order to compute p, let us consider a generic timeslot that begins at time t. As the beginning of the timeslot is asynchronous with the arrival process, the first renewal following time t occurs after time V, while the second, third, etc. occur after time $V + X_2$, $V + X_2 + X_3$, $V + X_2 + X_3 + \dots$. The time V is called *forward recurrence time*, while $X_2, X_3, \dots\dots$ are usual

renewal times and the resulting renewal process is referred to as *modified*. When t is large enough so that the system has been running for long, it can be proven [26] that the forward recurrence time has pdf $f_V(x) = \dfrac{1-F_X(x)}{\mathbb{E}[X]}$.

Under the previous hypothesis, the overall process is called *equilibrium renewal process* and the probability p of at least one arrival in the timeslot of length T is:

$$p \quad = \quad 1 - \Pr\{V > T\} \tag{4.7}$$

The RACH collision probability and the RAO collision probability are then readily obtained by substituting (4.7) in (4.5), and (4.6) respectively.

Finally, expression (4.7) can be specialized when the pdf of renewals is that of (4.2). Indeed, in this case,

$$p = 1 - \frac{\sum_{c=1}^{C} \dfrac{\alpha_c}{\lambda_c} e^{-\lambda_c T}}{\sum_{c=1}^{C} \dfrac{\alpha_c}{\lambda_c}} \tag{4.8}$$

So far, the value of p has always been computed as the probability of the complementary event of observing zero arrivals in a timeslot. For the sake of completeness, the probability of having *exactly* one arrival in a timeslot is given by:

$$\begin{aligned} &\Pr\{N(T)=1\} = \Pr\{V \le T\} - \Pr\{V + X \le T\} = \\ &= \frac{\left(T \sum_{c=1}^{C} \alpha_c^2 e^{-\lambda_c T} + \sum_{c=1}^{C} \sum_{k \neq c}^{C} \dfrac{\alpha_c \alpha_k \left(e^{-\lambda_k T} - e^{-\lambda_c T} \right)}{\lambda_c - \lambda_k} \right)}{\sum_{c=1}^{C} \dfrac{\alpha_c}{\lambda_c}} \end{aligned} \tag{4.9}$$

and it is easy to prove that, under practical parameter configuration such as the one of table 4.6, the difference between equation (4.9) and (4.8) is in the order of 10^{-4}.

4.6.1 Performance Evaluation

The accuracy of the proposed model is assessed by simulating a cell with 1000 UEs, each one independently placing RACH requests with inter–arrival times picked randomly from the IRR_j timeseries. Simulation runs are carried out with different configuration of the RRC_{IT} value (namely, 2, 5, and 10s), and the performance parameters, $P_C^{(HTC)}$ and $P_C^{(RAO)}$, are estimated after an observation time of 1000 s.

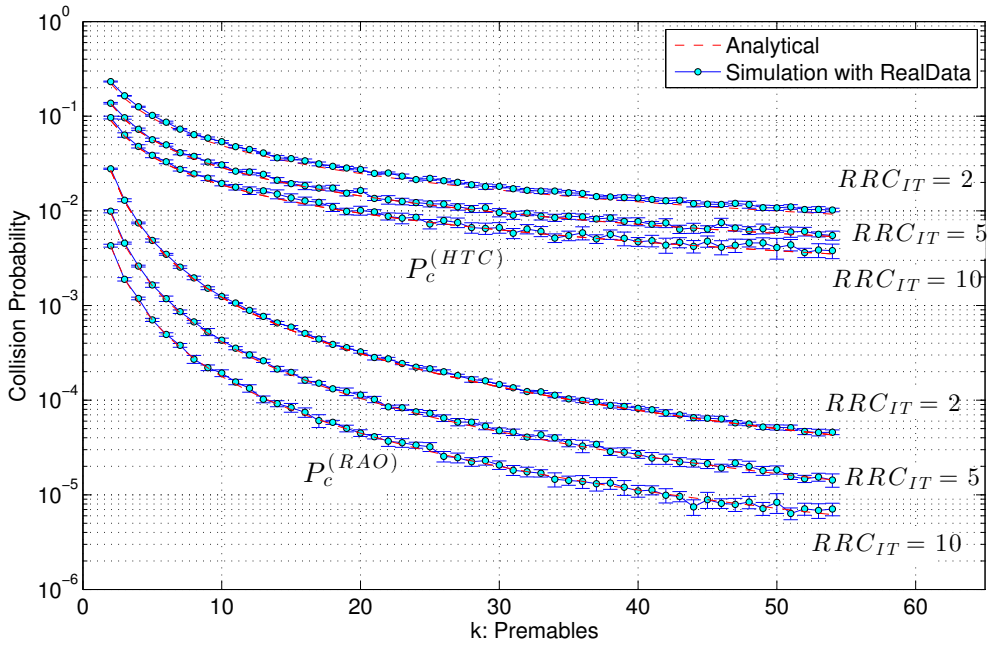

Fig. 4.18 Analytical model vs. simulation results - $P_C^{(HTC)}$ and the $P_C^{(RAO)}$ for different RACHP values, k

Figure 4.18 shows the average values of the $P_C^{(HTC)}$ and the $P_C^{(RAO)}$ along with the 95% confidence interval estimated over 10 simulation runs for the three RRC_{IT} configurations.

Moreover, the figure reports the analytic collision probability curves, $P_C^{(HTC)}$ and the $P_C^{(RAO)}$, derived in (4.5) and (4.6) respectively, using the values of p computed for the renewal times IRR_j modelled as shown in section 4.5 (i.e., p is set according to relation (4.8)). The results show an excellent adherence of the theoretical model with respect to the simulation outcomes as the analytical curves almost always lay within the 95% C.I. of simulations for all RRC_{IT} configurations.

4.7 Guidelines for RACH preamble serparation between HTC and MTC

In LTE and LTE–Advanced systems the rate of requests on the Random Access CHannel (RACH) can be high. Indeed, the Machine Type Communication (MTC) implies to have a high number of devices that need to request radio resources for transmitting small amount of data. Furthermore, reducing the time in which radio resources are allocated to Human Type Communications (HTC) for energy savings purposes, may lead to radio access network

overload as well. In this framework, this section aims at providing a set of guidelines for the resource allocation task in the RACH. In particular, the study investigates the impact of both the backoff indicator scheme and the maximum number of retransmissions on the RACH performance parameters. The rate of RACH requests associated with the HTC traffic is modelled by inferring their statistical properties starting from a dataset acquired in an operational eNodeB. The estimation of the average delay and the average number of maximum retransmissions gives insights on how many preambles should be reserved for HTC in order to meet the target performance, and provides suggestions on the configuration of the backoff indicator.

4.7.1 Simulation Design: MTC, HTC and RAO definition

The content-based operation of the RACH is based on ALOHA-type acces, that means "transmit the request in the first available opportunities". As we explained before the maximum number of RAO opportunities is determined by the *PracConfigIndex* and the number of available preambles.

Two definitions of collision probability are given in TR 37.686.

In ANNEX B, the so colled "MTC collision probability" has been defined as the collision probability of an MTC device which transmits a preamble. This is a conditioned probability, and is calculated as:

$$\frac{\#\ of\ collided\ Preambles\ in\ T\ seconds}{\#\ of\ transmitted\ Preambles\ in\ T\ seconds}$$

We apply the same definition, for HTC. The different between MTC and HTC is given by their statistical behavior (e.g. the rate of random access attempts).

In section 6.3, the "RAO collision probability" is defined as the ratio between the number of occurrences when two or more MTC/HTC devices send in the same RAO, meaning in the same timeslot and with the same preambles, and the total number of RAO, no matter if there was or not a random access. This is an unconditioned probability, and it is given by

$$\frac{\#\ of\ collided\ Preambles\ in\ T\ seconds}{\#\ of\ RAO\ in\ T\ seconds}$$

In our simulation, it is assumed that the network reserves L RAOs per second for MTC devices to transmit their preambles. We considered 1000 MTC and/or HTC devices under coverage. Already deployed eNodeBs allow to host few thousand of devices (MTC and HTC), depending on their configuration. This is to say that we don't have to wait for 5G networks to solve and analyze this kind of problem.

Time is dived in time slot defined by the currently set *PracConfigIndex*. In this study, it has been considered that 10 out of the 64 available preambles are always reserved for contention free procedure. The MTC/HTC devices which wake up between two slots shall wait and transmit at the beginning of the next available RACH slot, using one of the available preambles. Two possible scenarios could happen.

The random access is completed without collision. The device is supposed to transmit data and remains in connected state until the RRC inactivity timer elapses. Theoretically, if new data have to be transmitted during the connected state, the MTC/HTC device has no need to perform the random access again. Taking into account the time spent in connected state, it lowers the number of devices that at a given time can compete on a given resource. In other words, this methodology allows us to obtain a tighter bound for the collision probability estimation.

The random access is not completed, due to a collision. When two or more UEs use the same RAO we assume that the signal always act as interference to each other, and every UEs will experience a collision. Each one of these UEs, will retransmit the PRACH at any time between 0 and the given backoff indicator time. This procedure is repeated until all UEs successfully perform the random access. There is no maximum value for RACH retransmissions.

4.7.2 Simulation results: MTC traffic

The first analysis aims at evaluating the impact of *PrachConfigIndex* on the collision probability. The different parameter setups associated with each *PrachConfigIndex* are reported in [9].

Across all simulations we modify the random access intensity for a specific RAO, to be fair in our results. The access attempts generated at every timeslot follow a Poisson process. The rate, Λ, is set to obtain the same access intensity in all simulations. For example, with *PrachConfigIndex* equal to 0 or 1 or 2, the Poisson rate $\Lambda = 10$ (as in [24]), whereas with *PrachConfigIndex* equal to 3 or 4 or 5, the value of Λ is set to 5. It is worth noticing that evaluating results for configuration indexes up to 14 is completely exhaustive for all 64 possible values. Our results consider the first 11 *PrachConfigIndex*.

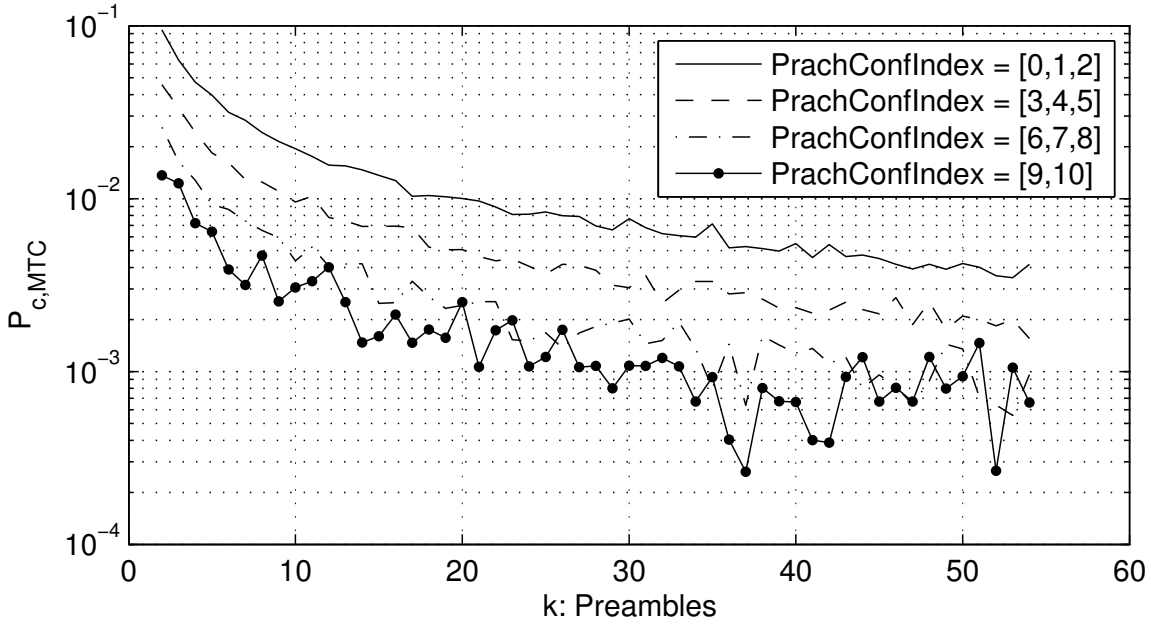

Fig. 4.19 $P_c^{(MTC)}$ for different *PrachConfigIndex* - $P_c^{(MTC)}$ is defined in AnnexB of [8]

We compute the collision probability by varying the number of available preambles from 2 to 54. For each possible value we repeat the simulations 10 times. Each simulation lasts for 10^6 timeslots.

As expected, the larger the granularity of RAO, the lower the collision probability (see figures 4.19 and 4.20). In addition, by increasing too much such granularity (*PrachConfigIndex* from 6 to 11) does not provide significant improvements when the access rate is not so high. Conversely, this leads to wasting unnecessary resources. Thus, the next presented results are obtained by assuming *PrachConfigIndex* 0, which represents our worst case scenario.

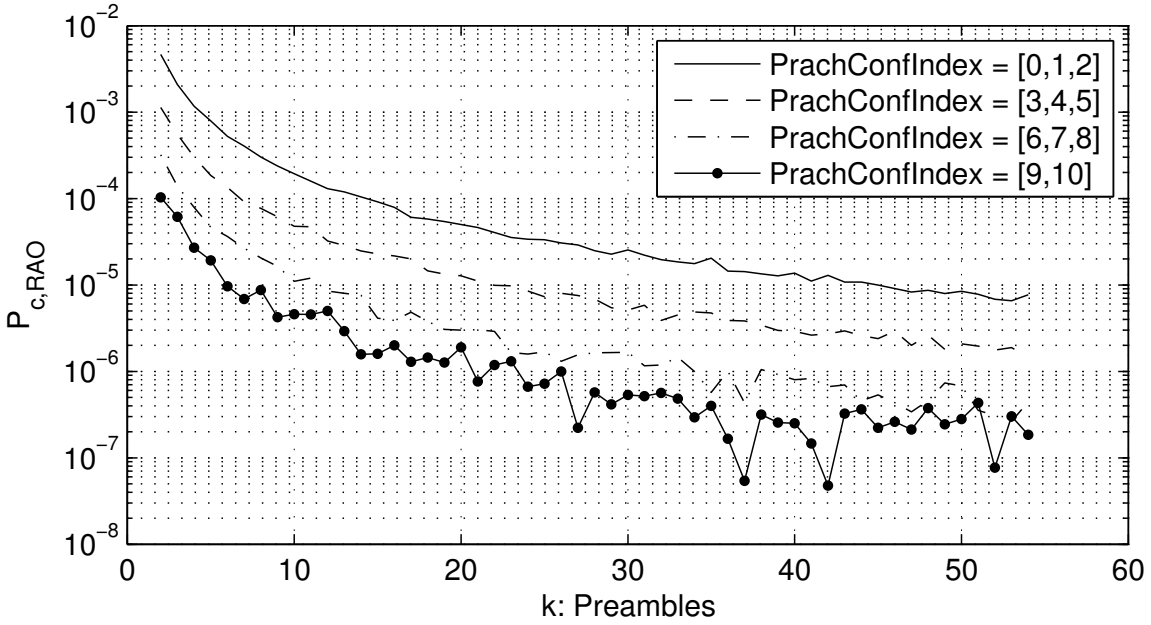

Fig. 4.20 $P_c^{(RAO)}$ for different *PrachConfigIndex* - $P_c^{(RAO)}$ is defined in Section 6.3 of [8]

Previous studies do not consider the effect of the backoff timer on the performance parameters. They assume that if two or more MTCs fail during the initial RACH procedure, no successive attempts will be made, leading to an under estimation of the collision probability. To overcome this issue, instead of considering a Poisson process to generate the aggregate number of RACH requests, we consider 1000 MTC devices individually, each one generates random access attempts with i.i.d. exponential interarrival times. We carried out two simulation runs, one with mean inter–arrival times $\lambda_1 = 100$s , and the other with $\lambda_2 = 10$s. The backoff indicator varies from 20ms to 320ms and a maximum number of 10 retransmissions are allowed.

Figures 4.21 shows the results. Using low backoff allows MTC to reduce the time needed for a new attempt, lowering the overall latency for RACH procedure. Depending on M2M application class, it could be important not to excessively delay retransmissions. It is worth recalling that the setting *PrachConfigIndex* equal to 0, and Backoff Indicator (BI) equal to 20ms implies "to retransmit the next slot", whereas a BI of 160ms uniformly spreads the probability of retransmission over the next 8 time slots.

When the traffic load is limited (λ_1), numeric results demonstrate that introducing the backoff scheme has low impact, and the $P_{c,MTC}$ has a similar trend as the case without the backoff. Indeed, except for the first preambles (up to 5), $P_{c,MTC}$ can be evaluated considering the analytical model provided in [24]. With such a low rate, even when few preambles are disposed to MTC devices the radio interface does not become highly congested and the

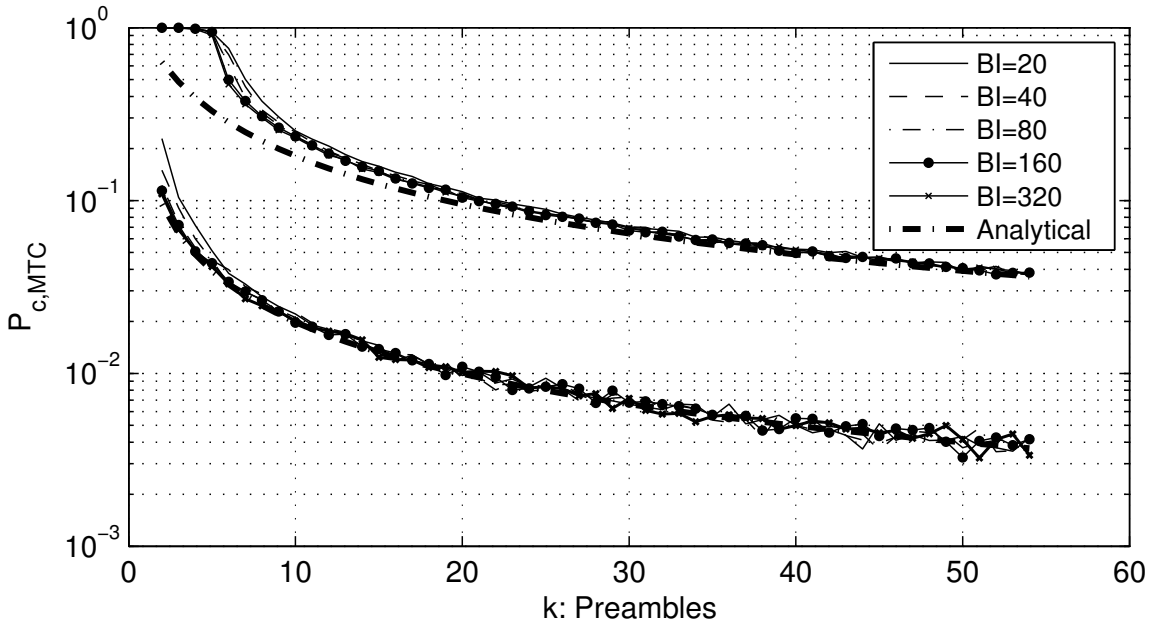

Fig. 4.21 $P_c^{(MTC)}$ evaluated with $\lambda_2 = 10$ (upper curves) and $\lambda_1 = 100$ (lower curves)

probability of correct transmission gets immediately close to one. Due to the low rate of requests, the benefit obtained from the backoff timer is hardly perceptible.

Instead, when random access rate is higher and equal to $1/\lambda_2$ per second, the $P_{c,MTC}$ deviates from the performance expected by the analytical curve. This analysis proves that the analytical formula for $P_{c,MTC}$ is not accurate enough in presence of backoff timer and maximum number of transmissions, especially when a small number of preambles is reserved. Indeed, in order to observe acceptable performance (e.g. in the order of 10^{-1}) at least 20 preambles must be reserved.

It is worth looking at the average delay (figure 4.23) and the success probability (figure 4.22) to appreciate the influence of the BI, in the two cases. For λ_1, MTC devices always manage to correctly transmit data, whatever backoff scheme is used. For λ_2, the success probability suggests not to reserve less than 7–8 preambles, depending on the backoff which is used. The average delay is evaluated by considering only the correct transmissions. If we do not consider the very low k values (few units), for which the collision probability is close to one and the success probability (*goodputh*) gets close to zero, the results suggest the uselessness of using large BI. Indeed, setting the BI to 320ms does not allow to achieve high improvements from the *goodputh* point of view, though it produces larger access delay.

We analyze the average maximum number of attempts 4.24 needed to successfully establish the RRC connection, when λ_2 is used. The results have been evaluated for a subset of the available preambles, namely from 2 up to 30.

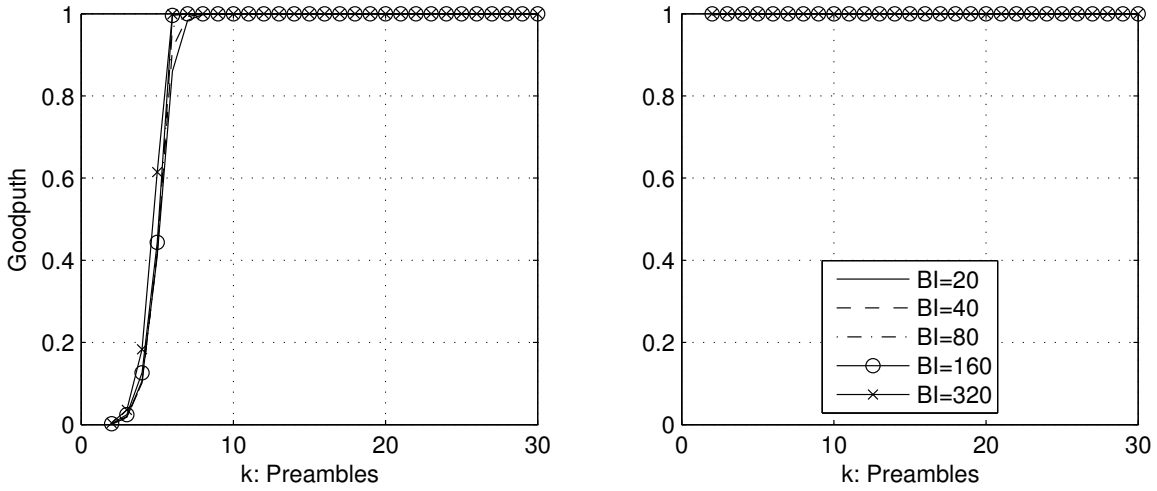

Fig. 4.22 Success Probability with $\lambda_2 = 10$ (left) and $\lambda_1 = 100$ (rigth)

Given that the maximum number of possible retransmissions is set to 10, the event "11-th attempts" in the graph means that the request will not be acknowledged by the network. The results, shown in figure 4.24, reflect the behavior described in figure 4.22 and 4.23.

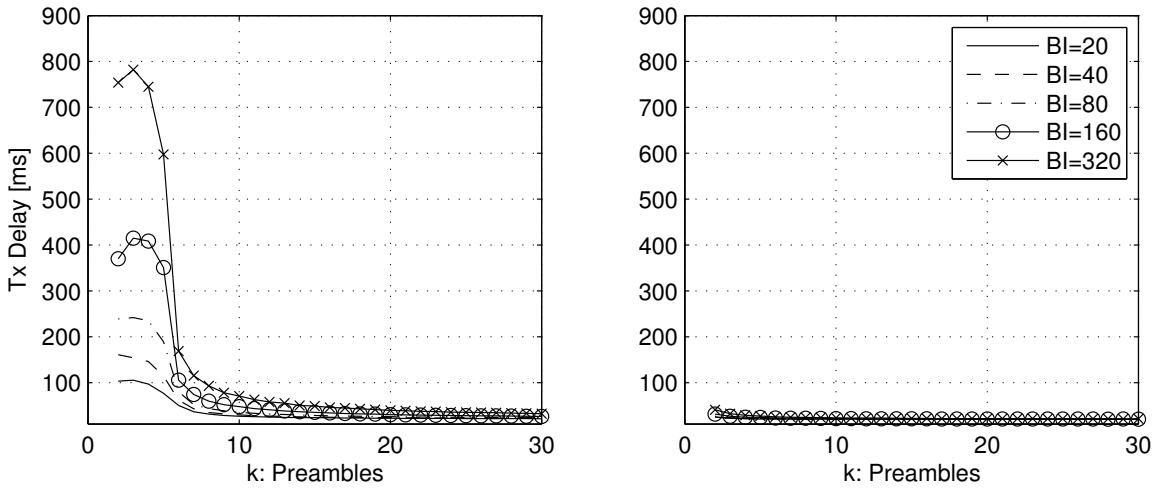

Fig. 4.23 Transmission Delay (from the 1st attempt to the successfull one), with $\lambda_1 = 100$ (left) and $\lambda_2 = 10$ (rigth)

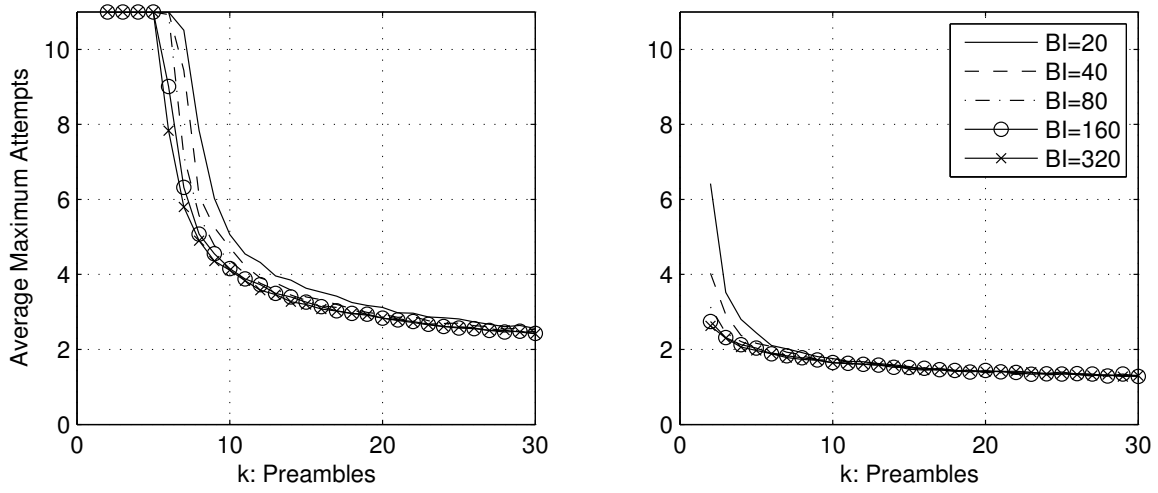

Fig. 4.24 Average maximum number of attempts per UE, with $\lambda_2 = 10$ (left) and $\lambda_1 = 100$ (rigth)

4.7.3 Simulation results: HTC traffic

For each considered RRC_{IT} value, the Bayesian model selection approach has been applied to estimate the parameters of the hyperexponential distribution.The estimated parameters are summarized in table 4.6.

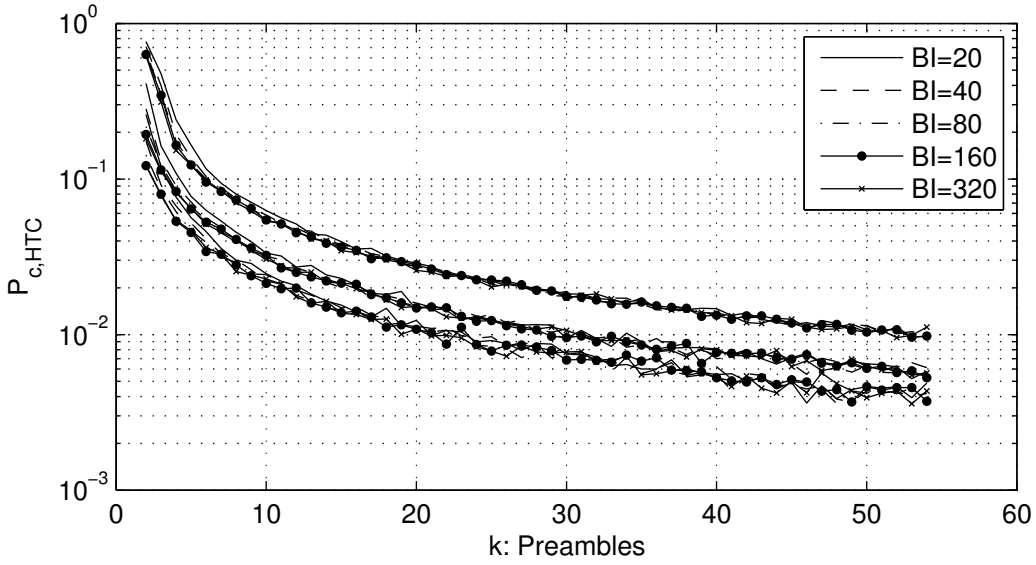

Fig. 4.25 $P_c^{(HTC)}$ evaluated with $RRC_{IT} = 2s$, $RRC_{IT} = 5s$ and $RRC_{IT} = 10s$

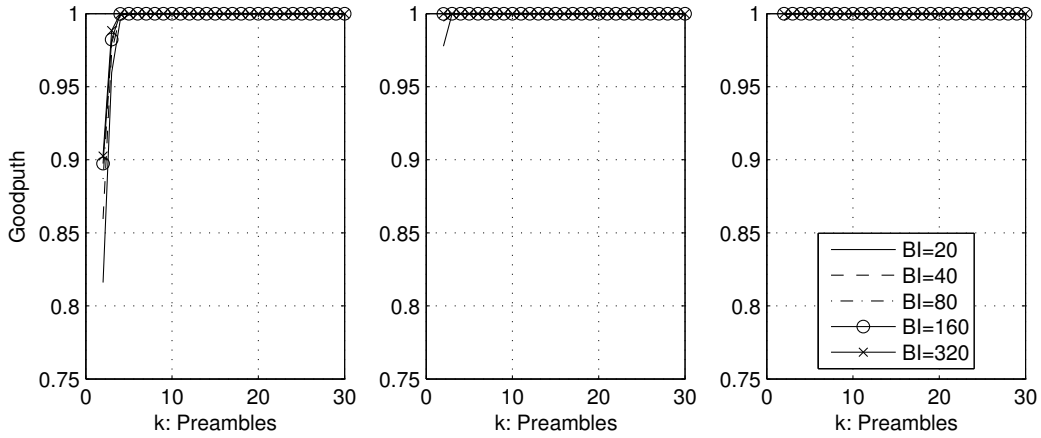

Fig. 4.26 Success Probability for HTC evaluated with $RRC_{IT} = 2s$ (left), $RRC_{IT} = 5s$ (center) and $RRC_{IT} = 10s$ () rigth

The first (expected) outcome is that, decreasing the RRC_{IT} values induces an increase of the $P_{c,HTC}$ (figure 4.25). Interestingly, if we used the analytical model provided in section 4.6, equations (4.5) and (4.6), which doesn't take into account the backoff scheme we'd obtain

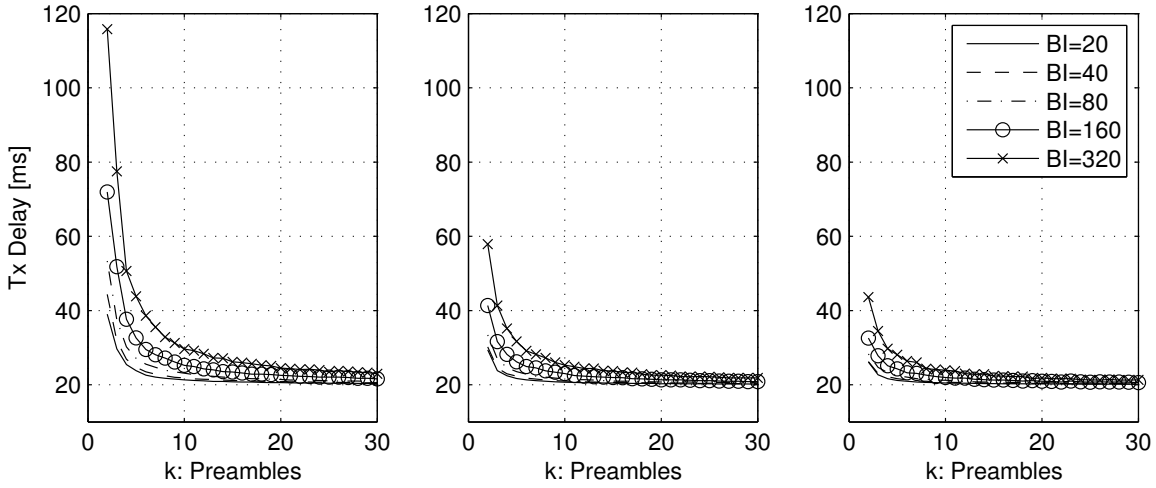

Fig. 4.27 Transmission delay for HTC, evaluated with $RRC_{IT} = 2s$ (left), $RRC_{IT} = 5s$ (center) and $RRC_{IT} = 10s$ () rigth

almost a perfect matching with simulation result. Thus, given the low rate of RACH request, even for the lowest RRC_{IT} value, we can predict the $P_{c,HTC}$ using the proposed formula when enough preambles are reserved for HTC. This interesting result may be really helpful if we'd like to determinate the number of preambles that have to be allocated given a target $P_{c,HTC}$. Furthermore, we observe that for larger numbers of preambles, the BI has no effect on the collision probability. By looking at the success probability (figure 4.26) we found out that few preambles are enough to assure that all RACH requests are acknowledged by the eNodeB.

To properly interpreter transmission delay results, it is worth reminding that in LTE/LTE-Advanced each bearer is associated to a QoS Class Identifier (QCI) with standardized delay budget. For example, the delay budget associated to QCI = 5 and QCI = 9 is 100ms and 300ms, respectively. Figure 4.27 shows the drawback of using large values of BI. Indeed, with large BI, we observe a slight decrease of the collision probability at the cost of increasing the access delay, almost achieving the maximum delay budget defined for some QCIs. Simulation results demonstrate that this is unnecessary, because few preambles has to be allocated to obtain the same performance for all the backoff values.

Numerical results point out that using an RRC_{IT} of 5 s with a BI of 40 ms guarantees that all RACH requests are delivered to the eNodeB. To attain a collision probability of 10^{-2} at least 30 preambles must be reserved. The average transmission delay is slightly above 20 ms, which is slightly higher than the minimum delay imposed by the *PrachConfigIndex*.

4.8 Conclusion

Evaluate the performance of hardware and software systems for the fourth-generation mobile network, as well as identify any possible weakness in the architecture, it is a complex job. A possible case study, is precisely to assess the robustness of the base station when it receives many requests for RRC connections, as effect of a decrease of the inactivity timer. In this regard, within the Testing LAB of Telecom Italia, we used IxLoad, a product developed by Ixia, as a load generator to test the robustness of one eNodeB. The tests consisted in producing a different load of RRC request on the radio interface, similar to those that would be produced by decreasing the inactivity timer to certain values. The statistical properties for the signalling traffic are derived from the analysis of real traffic traces. The main outcomes have shown that, even in the face of an high load of RRC requests only a small part (less than 1% in the most unfavorable of the cases) of the procedure fails. Therefore, even lowering the inactivity timer at values lower than 10 seconds is not an issue for the Base Station.

Finally, remains to be evaluated how such surge of RRC request impacts on users performance. If one of the users under coverage in the RRC Idle is paged for an incoming packet or need to send an uplink packet a state transition from RRC Idle to RRC Connected is needed. At this point, the UE initiates the random access procedure by sending the random access channel preamble (RACH Preamble). When two or more users attempt, simultaneously, to access the RACH channel, using the same preamble, the eNodeB may not be able to decipher the preamble. We presented a procedure to model the interarrival times of RACH requests by means of a mixture of exponential distributions. This procedure and the proposed analytical model provide an accurate estimation of the $P_C^{(HTC)}$ and the $P_C^{(RAO)}$ as a function of the number of UEs and the number of available preambles, for different RRC_{IT} settings.

In addition, we investigated the issue of radio access network overloading due to HTC and MTC by elaborating upon the impact of both the backoff indicator scheme and the maximum number of retransmissions in the RACH procedure. The performance evaluation is carried out by simulation under different number of available preambles. In the case of MTC, the results show that for high RACH request rate and small number of allocated preambles, the analytical model discussed in [24] losses accuracy in estimating the collision probability. In the case of HTC, the results suggest the use of low backoff values in that the negligible improvements in terms of collision probability obtained by increasing the backoff are obtained at the cost of a relatively high access delay. In particular, we recall that the estimated average delay in some cases almost achieves the maximum delay budget defined for some QCIs. The results prove that the adoption of an RRC_{IT} of a few seconds does not significantly affect the RACH performance, while still reducing power consumption.

References

[1] (2015). Layer-7. http://www.l7-filter.clearos.com.

[2] (2015). List of assigned port numbers. http://www.iana.org/assignments/port-numbers.

[3] (2015). Marketshare. http://www.marketshare.com.

[4] (2015). Ntop. http://www.ntop.org.

[5] (2015). Pace. http://www.ipoque.com/en/products/pace.

[6] (2015). Tstat. http://tstat.tlc.polito.it.

[7] 3GPP Standard; 3GPP TR 36.822 (2012). LTE RAN Enhancements for Diverse Data Applications. Technical report.

[8] 3GPP Standard; 3GPP TR 37.868 (2011). Technical Specification Group Radio Access Network; Study on RAN Improvements for Machine-type Communications. Technical report.

[9] 3GPP Standard; 3GPP TS 36.221 (2014). Evolved Universal Terrestrial Radio Access (E-UTRA); Physical channels and modulation. Technical specification.

[10] 3GPP Standard; 3GPP TS 36.300 (2014). Evolved Universal Terrestrial Radio Access (E-UTRA) and Evolved Universal Terrestrial Radio Access Network (E-UTRAN); Overall description; Stage 2. Technical specification.

[11] 3GPP Standard; 3GPP TS 36.304 (2014). User Equipment (UE) procedures in idle mode (Release 12). Technical specification.

[12] 3GPP Standard; 3GPP TS 36.321 (2014). Evolved Universal Terrestrial Radio Access (E-UTRA); Medium Access Control (MAC) protocol specification. Technical specification.

[13] 3GPP Standard; 3GPP TS 36.331 (2014). Evolved Universal Terrestrial Radio Access (E-UTRA); Radio Resource Control (RRC); Protocol specification. Technical specification.

[14] Aceto, G., Dainotti, A., De Donato, W., and Pescap, A. (2010). Portload: taking the best of two worlds in traffic classification. In *INFOCOM IEEE Conference on Computer Communications Workshops, 2010*, pages 1–5. IEEE.

[15] Alcock, S. and Nelson, R. (2012). Libprotoident: Traffic classification using lightweight packet inspection. *WAND Network Research Group, Tech. Rep*.

[16] Aucinas, A., Vallina-Rodriguez, N., Grunenberger, Y., Erramilli, V., Papagiannaki, K., Crowcroft, J., and Wetherall, D. (2013). Staying online while mobile: The hidden costs. In *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies*, pages 315–320. ACM.

[17] Bermudez, I. N., Mellia, M., Munafò, M. M., Keralapura, R., and Nucci, A. (2012). Dns to the rescue: discerning content and services in a tangled web. In *Proceedings of the 2012 ACM conference on Internet measurement conference*, pages 413–426. ACM.

[18] Bontu, C. S. and Illidge, E. (2009). Drx mechanism for power saving in lte. *Communications Magazine, IEEE*, 47(6):48–55.

[19] Bousia, A., Kartsakli, E., Antonopoulos, A., Alonso, L., and Verikoukis, C. (2013). Game theoretic approach for switching off base stations in multi-operator environments. In *Communications (ICC), 2013 IEEE International Conference on*, pages 4420–4424. IEEE.

[20] Bujlow, T., Carela-Español, V., and Barlet-Ros, P. (2015). Independent comparison of popular dpi tools for traffic classification. *Computer Networks*, 76:75–89.

[21] Cascarano, N., Este, A., Gringoli, F., Risso, F., and Salgarelli, L. (2009). An experimental evaluation of the computational cost of a dpi traffic classifier. In *Global Telecommunications Conference, 2009. GLOBECOM 2009. IEEE*, pages 1–8. IEEE.

[22] Chen, Y. and Wang, W. (2010). Machine-to-machine communication in lte-a. In *Vehicular Technology Conference Fall (VTC 2010-Fall), 2010 IEEE 72nd*, pages 1–4. IEEE.

[23] Cheng, J.-P., Lee, C.-h., and Lin, T.-M. (2011). Prioritized random access with dynamic access barring for ran overload in 3gpp lte-a networks. In *GLOBECOM Workshops (GC Wkshps), 2011 IEEE*, pages 368–372. IEEE.

[24] Cheng, R.-G., Wei, C.-H., Tsao, S.-L., and Ren, F.-C. (2012). Rach collision probability for machine-type communications. In *Vehicular Technology Conference (VTC Spring), 2012 IEEE 75th*, pages 1–5. IEEE.

[25] Condoluci, M., Dohler, M., Araniti, G., Molinaro, A., and Zheng, K. (2015). Toward 5g densenets: architectural advances for effective machine-type communications over femtocells. *Communications Magazine, IEEE*, 53(1):134–141.

[26] Cox, D. R., Cox, D. R., Cox, D. R., and Cox, D. R. (1962). *Renewal theory*, volume 4. Methuen London.

[27] Crotti, M., Dusi, M., Gringoli, F., and Salgarelli, L. (2007). Traffic classification through simple statistical fingerprinting. *ACM SIGCOMM Computer Communication Review*, 37(1):5–16.

[28] Deng, S. and Balakrishnan, H. (2012). Traffic-aware techniques to reduce 3g/lte wireless energy consumption. In *Proceedings of the 8th international conference on Emerging networking experiments and technologies*, pages 181–192. ACM.

[29] Falaki, H., Mahajan, R., Kandula, S., Lymberopoulos, D., Govindan, R., and Estrin, D. (2010). Diversity in smartphone usage. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, pages 179–194. ACM.

[30] Fiadino, P., Bar, A., and Casas, P. (2013). Httptag: A flexible on-line http classification system for operational 3g networks. In *Computer Communications Workshops (INFOCOM WKSHPS), 2013 IEEE Conference on*, pages 71–72. IEEE.

[31] Foremski, P., Callegari, C., and Pagano, M. (2014). Dns-class: immediate classification of ip flows using dns. *International Journal of Network Management*, 24(4):272–288.

[32] Frühwirth-Schnatter, S. (2006). *Finite mixture and Markov switching models*. Springer Science & Business Media.

[33] Fukuda, K. and Nagami, K. (2013). A measurement of mobile traffic offloading. In *Passive and Active Measurement*, pages 73–82. Springer.

[34] Huang, J., Qian, F., Guo, Y., Zhou, Y., Xu, Q., Mao, Z. M., Sen, S., and Spatscheck, O. (2013). An in-depth study of lte: Effect of network protocol and application behavior on performance. In *ACM SIGCOMM Computer Communication Review*, volume 43, pages 363–374. ACM.

[35] Huang, J., Xu, Q., Tiwana, B., Mao, Z. M., Zhang, M., and Bahl, P. (2010). Anatomizing application performance differences on smartphones. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, pages 165–178. ACM.

[36] Jha, S. C., Koc, A. T., and Vannithamby, R. (2012). Optimization of discontinuous reception (drx) for mobile internet applications over lte. In *Vehicular Technology Conference (VTC Fall), 2012 IEEE*, pages 1–5. IEEE.

[37] Koc, A. T., Jha, S. C., Vannithamby, R., and Torlak, M. (2014). Device power saving and latency optimization in lte-a networks through drx configuration. *Wireless Communications, IEEE Transactions on*, 13(5):2614–2625.

[38] Laya, A., Alonso, L., and Alonso-Zarate, J. (2014). Is the random access channel of lte and lte-a suitable for m2m communications? a survey of alternatives. *Communications Surveys & Tutorials, IEEE*, 16(1):4–16.

[39] Lin, P.-C., Lin, Y.-D., Lai, Y.-C., and Lee, T.-H. (2008). Using string matching for deep packet inspection. *Computer*, (4):23–28.

[40] Lo, A., Law, Y. W., Jacobsson, M., and Kucharzak, M. (2011). Enhanced lte-advanced random-access mechanism for massive machine-to-machine (m2m) communications. In *27th World Wireless Research Forum (WWRF) Meeting*, pages 1–5.

[41] Moore, A. W. and Zuev, D. (2005). Internet traffic classification using bayesian analysis techniques. In *ACM SIGMETRICS Performance Evaluation Review*, volume 33, pages 50–60. ACM.

[42] Nguyen, T. T. and Armitage, G. (2008). A survey of techniques for internet traffic classification using machine learning. *Communications Surveys & Tutorials, IEEE*, 10(4):56–76.

[43] Paul, U., Subramanian, A. P., Buddhikot, M. M., and Das, S. R. (2011). Understanding traffic dynamics in cellular data networks. In *INFOCOM, 2011 Proceedings IEEE*, pages 882–890. IEEE.

[44] Qian, F., Sen, S., and Spatscheck, O. (2013). Silent tcp connection closure for cellular networks. In *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies*, pages 211–216. ACM.

[45] Roma, S., Donatini, L., Foddis, G., Garroppo, R., Giordano, S., Procissi, G., and Topazzi, S. (2014a). Advances in lte network monitoring: A step towards an sdn solution. In *Mediterranean Electrotechnical Conference (MELECON)*, pages 339–343. IEEE.

[46] Roma, S., Foddis, G., Garroppo, R., Giordano, S., Procissi, G., and Topazzi, S. (2014b). Lte traffic analysis and application behavior characterization. In *Networks and Communications (EuCNC), 2014 European Conference*. IEEE.

[47] Roma, S., Foddis, G., Garroppo, R., Giordano, S., Procissi, G., and Topazzi, S. (2015). Traffic analysis for signalling load and energy consumption trade-off in mobile networks. In *International Conference on Communications (ICC)*. IEEE.

[48] Singh, H. (2015). Performance analysis of unsupervised machine learning techniques for network traffic classification. in advanced computing. In *In Advanced Computing & Communication Technologies (ACCT)*, pages 401–404. IEEE.

[49] Stea, G. and Virdis, A. (2014). A comprehensive simulation analysis of lte discontinuous reception (drx). *Computer Networks*, 73:22–40.

[50] Sundaresan, K. and Rangarajan, S. (2013). Energy efficient carrier aggregation algorithms for next generation cellular networks. In *Network Protocols (ICNP), 2013 21st IEEE International Conference on*, pages 1–10. IEEE.

[51] Valenti, S., Rossi, D., Dainotti, A., Pescapè, A., Finamore, A., and Mellia, M. (2013). Reviewing traffic classification. In *Data Traffic Monitoring and Analysis*, pages 123–147. Springer.

[52] Wagner, H. (2007). Bayesian analysis of mixtures of exponentials. *Journal of Applied Mathematics, Statistics and Informatics*, 3:165–183.

[53] Xu, Q., Erman, J., Gerber, A., Mao, Z., Pang, J., and Venkataraman, S. (2011). Identifying diverse usage behaviors of smartphone apps. In *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*, pages 329–344. ACM.

[54] Xue, Y., Wang, D., and Zhang, L. (2013). Traffic classification: Issues and challenges. In *Computing, Networking and Communications (ICNC), 2013 International Conference on*, pages 545–549. IEEE.

Appendix A

LTE Theoretical Limits

A.1 Maximum Number of UE per TTI

Here we presented a rough calculation for the maximum number of UEs that can be scheduled in one TTI. Each vendor defines this value, but there's a theoretical max limit for the number of UEs to be allocated within a single TTI. As we mentioned before, this strictly affects the maximum achievable throughput in the cell.

Roughly speaking, PDCCH is the bottleneck when we talk about max number of UEs can be scheduled in single TTI. Let's say we have the following settings: one cell with bandwidth size of 20MHz, 100 PRB, and 1 antenna ports. With this setting, we calculate the number of REG (Resource Element Group) that are available for PDCCH, when 3 OFDM symbols are dedicated to the channel. In the first symbol we have 2RE per PRB for RS (Reference Signals), whereas the second and third symbol don't have RS. Thus, the first symbol has 2 REGs/RB while the 2nd and 3rd symbols each has 3REGs/RB, therefore in 100 PRB channel, there are 100 x (2+3+3) = 800 REGs for PDCCH, PCFICH and PHICH.

Considering that:

- PCFICH consumes 4 REGs
- mimimum size for PHICH in 20 MHz = 3 * ceil ((1/6) * (100/8)) = 9 REGs
- Size of PDCCH in REGs = 800 -4 -9 = 787 REGs
- Size of PDCCH in CCEs = floor (787/9) = 87 CCEs

The number of CCEs used to transfer a DCI is called CCE aggregation level, and may be 1, 2, 4, or 8 consecutive CCEs (logical sequence), depending on the used PDCCH format. PDCCH format 0 uses 1 CCE, PDCCH format 1 uses 2 CCEs,and so on. Three main reasons

justify different aggregation level. First, PDCCH format is selected according to the size of the DCI: different type of DCI are used to improve resource utilization. Second, to accommodate different RF conditions. The ratio between the DCI size and the PDCCH size indicates the effective coding rate. With the DCI format fixed, higher aggregation levels provide more robust coding and reliability for the UEs under poor RF conditions. For a UE in good RF conditions, lower aggregation levels can save resources. Third, to differentiate DCIs for control messages and DCIs for UE traffic. Higher aggregation levels can be used for control message resource allocations to provide more protection.

Thus, if we use only PDCCH format 0 for allocating resources within this TTI, there are possible 87 DCI allocations i.e. 87 UEs can be effectively allocated resources. On the other extreme if we only use PDCCH format 3 (i.e. most robust scheme), then we have: floor(787 / 72) = 10 DCI allocations. Meaning 10 UEs can be allocated resources.

It is worth to be noted that that these DCI are used to allocate both DL and UL, and there are also some common messages using DCIs, such as System Info and Paging (that typically use higher formats, i.e. requires more robust PDCCH schemes, because they need to be received by users without knowing their channel conditions).

A.2 Maximum Downlink Throughput

You may hear it many times that the peak data rate of LTE is about 300Mbps. Let's estimate it in a simple way. Assume 20 MHz channel bandwidth, normal CP, 4x4 MIMO.

First, calculate the number of resource elements (RE) in a subframe with 20 MHz channel bandwidth: 12 subcarriers x 7 OFDMA symbols x 100 resource blocks x 2 slots= 16800 REs per subframe. Each RE can carry a modulation symbol.

Second, assume 64 QAM modulation and no coding, one modulation symbol will carry 6 bits. The total bits in a subframe (1ms) over 20 MHz channel is 16800 modulation symbols x 6 bits / modulation symbol = 100800 bits. So the data rate is 100800 bits / 1 ms = 100.8 Mbps.

Third, with 4x4 MIMO, the peak data rate goes up to 100.8 Mbps x 4 = 403 Mbps. Fourth, estimate about 25% overhead such as PDCCH, reference signal, sync signals, PBCH, and some coding. We get 403 Mbps x 0.75 = 302 Mbps.

Is there a way to calculate it more accurately? If this is what you look for, you need to check the 3GPP specs 36.213, table 7.1.7.1-1 and table 7.1.7.2.1-1. Table 7.1.7.1-1 shows the mapping between MCS (Modulation and Coding Scheme) index and TBS (Transport Block Size) index. Let's pick the highest MCS index 28 (64 QAM with the least coding), which is mapping to TBS index of 26. Table 7.1.7.2.1-1 shows the transport block size. It indicates

the number of bits that can be transmitted in a subframe/TTI (Transmit Time Interval). For example, with 100 RBs and TBS index of 26, the TBS is 75376. Assume 4x4 MIMO, the peak data rate will be 75376 x 4 = 301.5 Mbps.