Seminar: Distributed Computer Systems SS23

On the Reliability of LTE Random Access: Performance Bounds for Machine-to-Machine Burst Resolution Time

Leonard Kleinberger
lkleinbe@rhrk.uni-kl.de

***Abstract*—Cellular networks are becoming increasingly important. Currently, the predominant technology for setting up mobile networks is 4G LTE which is used extensively to server a large number of users. That is why performance of LTE Networks is of high importance. For setting up connections, the RACH Procedure is the bottleneck, reducing the number of devices that can connect at the same time and increasing the time, until a burst arrival of devices is resolved. In the Paper TBD the authors analyse the performance limits of the RACH Procedure and the impact of different access barring policies. In this Seminar work the necessary background is presented and the findings are summarized. The theoretical analysis is further validated via large scale simulations run in CPython using Numpy.**

I. Introduction

Cellular Networks serve a high number of users over large areas. The most used technology for mobile communication networks is 4G Long Term Evolution (LTE), while the usage of 5G NR networks is steadily increasing. In Germany 97.1% of the surface is atleast served by one LTE Provider **??**. These Systems also supply a large number of devices thereby holding a challenge from a performance standpoint. In Germany over 74.5 Million active SIM Cards were served by roughly 85,000 LTE Base Stations. In Addition to that over 58.3 Million Machine to Machine(M2M) SIm Cards were operating without active users. One common occurrence in LTE systems is the arrival of many UEs in a short period of time which can be characterized as burst arrival. One scenario of a burst arrival are power outages. Base Stations are usually not supplied with emergency power leading the base station to shut down during power outages. On power restoration and reboot, most devices in the cell will try to reconnect in a short time leading to a burst arrival. Another scenario of burst arrival are emergency situations during which many users will either seek help, information or just call a relative. All these actions will lead to a burst arrival because many devices will be switching from idle mode to a connected State. Also, sensor networks can cause burst arrivals simply by reporting data at the same time.

To all burst arrivals of devices, the Random Access Channel(RACH) Procedure is the main bottleneck increasing the burst resolution time exponentially as the number of devices approaches the cell capacity. However, in emergency situations large delays are unacceptable for some users. In industrial M2M networks devices even need guarantees for reaching a connected state. For that reason the authors of **??** analyze the performance of LTE Networks under burst arrivals. The Authors use deterministic methods as well as probabilistic methods to derive methods, which can compute the probability for burst resolution by a given time.

The standardized mechanism to deal with burst arrivals in LTE is the usage of Access Class Barring Policies. These Policies reduce the number of admitted UEs thereby avoiding congestion. The Authors present such Policies and analyse their impact on the burst resolution probability.

The necessary background for this analysis is presented in this seminar work and the results of the authors are summarized. Further large Scale Simulations are implemented and compared to the findings of **??**

II. LTE System Background

This section will provide the necessary background to understand the LTE startup procedure, abstract from the physical layer and to analyse the Access Barring Policies

A. LTE Startup procedure

LTE end-to-end systems are structured into 2 Parts, the UMTS Terrestrial Radio Access Network (E-UTRAN) and the Evolved Packet Core (EPC). The E-UTRAN consists of all base stations(eNodeB), their respective cells and user equipment (UE). UEs are in many cases mobile phones. However, they can also be sensor networks, cars or other smart devices, which either do not have a wired connection to a service provider or have a roaming behaviour. An UE will first connect to an eNodeB, which provides a connection to the EPC and the Internet. After the connection with the eNodeB is fully established, the UE will be in the Radio Resource Control (RRC) Connected State, which allows it to send data via the eNodeB . This connection is only in the E-UTRAN . Typically, the UE will follow up with an attachment procedure with the EPC and some other procedures, which in turn will allow the ue to access the internet. Nevertheless, the Startup Procedure until the RRC Connected State will be the first bottleneck, which the UE will encounter. After turning on, the UE will first search for the Reference Signal, the Primary Synchronization Signal(PSS) and the Secondary Synchronization Signal(SSS). Evaluating these signals will allow the UE to compensate frequency offsets and synchronize with the eNodeB . After synchronization the UE will be able to receive the Master Information Block (MIB) and the System Information Blocks(SIBs), of which SIB1 and SIB2 are the most important for this work. MIB and SIB1 will contain important cell parameters, such as the coding scheme, the current Frame Number and the scheduling of further SIBs.

Acquiring this information enables the UE to receive SIB2, which announces the beginning of Random Access Channel (RACH) contention phases and the expected receiving signal power. If the UE wants to connect to the eNodeB, it has to wait for the beginning of a RACH contention phase. During a RACH contention phase the UE will select one RACH preamble out of a given Number of Preambles M and send it to the UE on the RACH . The eNodeB will respond with RACH Response, which contains a grant for each selected Preamble. This grant assigns the UE some resources in the network. The UE can use these resources to send an RRC Connection Request to the eNodeB on a different frequency at a scheduled timeslot. The RRC Connection Request will contain the required parameters to sign up the UE at the eNodeB . The eNodeB will respond with an RRC Connection Setup message, assigning an identifier for the ue and providing information about the channels, which the UE is now allowed to use. The UE will confirm this message with an RRC Connection Setup Complete message and enter the RRC Connected State.

B. LTE Physical Layer and collisions

LTE uses Orthogonal Frequency Division Multiplexing(OFDM). This means that lte signals consist of multiple sub-carriers, which are orthogonally placed into the frequency spectrum. The sub-carriers are therefore independent of each other and can send symbols without interference of other sub-carriers.

In LTE time is divided into Frames with a duration of $10ms$. LTE Frames are further divided into sub-frames with a duration of $1ms$ each containing 2 Symbols. These Symbols are then modulated onto sub-carrier using Amplitude Modulation. Physical Resource Blocks (PRB) are then defined as symbols mapped to each sub-carrier. PRBs can be visualized in a Resource Grid and are the smallest unit of assignable resources in LTE Networks. The eNodeB will give grants to the UE, which allow the UE to use specific PRBs to contact the eNodeB . In the same way, the eNodeB will reserve some PRBs for Messages send by the eNodeB .

Usually each transmission is 1 subframe long but this does not mean that only 2 Symbols can be sent. 1 option to transmit large messages is obviously to partition the message into multiple smaller messages. As an alternative, multiple sub-carriers can be used together to send multiple symbols at the same time. Finally different modulation and coding schemes can be applied to compress more data into these 2 symbols.

The PRACH Configuration is broadcast by the eNodeB and therefore is equal for all UE implying each UE participating in the contention phase will use the PRB. The expected receiving Signal power is also broadcast and timing requirements are very strict. If multiple UEs send the same Message to the UE, the messages will overshadow each other and the eNodeB can decode the message successfully. On the other Hand if different messages are send in the same PRB, the messages will therefore collide most likely destroying the messages. In rare cases the eNodeB might be able to decode one of the different messages, e.g. when one UE uses more sending power than broadcast. However, because UEs are typically mobile and run on battery power, they usually want to save as much power as possible and will not use a higher signalling power. RACH Preambles are an exception to this rule because they are by design not destructible. This implies that multiple UEs can send different RACH Preambles to the eNodeB in the same PRB, which will not interfere with each other. In this case, multiple UEs will be assigned the same grant for the RRC Connection Request. Again this will lead to multiple UEs sending a Connection Request Message in the same PRBs. Because the Connection Request Message contains UE specific parameters, these messages will be different from each other. RRC Connection Request Messages will interfere with each other in contrast to RACH Preambles, so there will be a collision destroying all RRC Connection Requests.

In summary if multiple UEs choose the same Preamble, the RACH Procedure will not complete because the RRC Connection Request Messages will collide destructively.

C. Access Barring Policies

A Network Operator will try to avoid that multiple UEs select the same RACH Preamble, because this leads to collision and wasted resources, as described in section II-B. Additionally, because the UEs did not finish the RACH Procedure successfully, they will participate in the next RACH contention phase, effectively using even more resources. This can lead to a very long connection time or even deadlocks, if the number of UEs trying to connect N is much larger than the Number of Preambles: M

$$N >> M$$

The most straightforward solution to this problem is to increase the number of Preambles M. Because Preambles are designed to be non-destructive between each other, there is a natural limit of 64 Preambles

$$M \leq 64$$

which can not be exceeded. Also in reality the number of Preambles is further limited by the quality of the physical channel and the already present traffic reducing the number of grants. As a result the network operator is usually not able to allow more PRACH Preambles. Access Barring Policies provide a different solution to the deadlock Problem. In SIB2 the eNodeB can specify an Access Barring Policy(ACB). This ACB specifies a probability p that a UE can participate in the RACH Procedure. If probabilty $p < 1$ is set, the UE will a run random throw on a binomial distribution with weight p, whether it is admitted to this specific PRACH contention window. In case to many UEs try to use the RACH Procedure at the same time, the eNodeB can therefore lower p reducing the number of admitted UEs and lowering the required time, until all ues are connected. The expected Number of admitted UEs then be:

$$E[N] = Np$$

SIB2 even allows setting the access probability differently for different UE Classes. This enables the eNodeB to give preference to emergency services or put sensor networks at a disadvantage.

The most basic Access Barring Policy is obviously just setting

$$p = 1$$

This implies that Access Barring is essentially not used.

An improvement over that is to use the static access barring policy. The intuition of this policy is to just set

$$p = x \ , \ x < 1$$

and leave it for the entire run. Static access barring policy can therefore reduce the number of admitted UEs by a factor x. On the downside static access barring policy is insensitive to the number of UEs trying to connect which might lead to p not being optimal. In general p has to lowered if many UEs are active and increased if only a few UEs are active. So if many UEs are active, p will probably be too high and if few UEs are active p will be too low. In Fact it is possible to set p always to the optimal solution The Authors of TBD showed that the optimal access barring policy is to set

$$p_i = \min\left\{1, \frac{M}{B(i)}\right\}$$

where p_i is the access probability and $B(i)$ is the access probability in contention window i. Optimal Access Barring policy assumes, that the backlog $B(i)$ is known to the eNodeB . However, in reality the eNodeB does not know the backlog and has only access to statistics about past contention windows and channel quality. That is why the authors of TBD propose dynamic access barring policy, which is based on the past number of idle preambles and the number of successful completed PRACH Procedures.

Algorithm 1 Dynamic Access Barring with estimation

1: Initialize $v_0 = M$ and $p_0 = 1/M$
2: **if** the number of idle preambles is r **then**
3: $\quad \Delta v = 0.582 \cdot M - 1.582 \cdot r$
4: $\quad v_{t-1} = v_{t-1} + \Delta v$
5: $\quad a_t = \max(0, \Delta v)$
6: $\quad v_t = v_{t-1} + a_i - s_i$
7: **end if**
8: $v_t = max(M, v_t)$ and $p_t = \min\left(1, \frac{M}{v_t}\right)$

The dynamic access barring policy is shown in Algorithm 1.

III. Analysis

This sections present the results of the Paper **??**

A. Assumptions

Because the perdiodicly of PRACH contention windows is depending on the eNodeB, cell conditions and already active users, time is partitioned into discrete time steps t. At t_0 the burst arrival occurs. $t_1..t_n$ are then the PRACH contention windows following t_0 An LTE System can then be characterized as a tuple

$$sc = (M, N, a_i, acb)$$

where

M is the number of Preambles
N is the number of UEs arriving at t_0
a_i is the number of UEs arriving at t_i
acb is the used ACB Policy

The required Quality of Service can then be defined as a tuple

$$qos = (b^\epsilon, t, \epsilon)$$

with b^ϵ being the required backlog at time t with a probability $1 - \epsilon$. So ϵ is the probability of violating the QoS requirement For many Simulations or analysis results some of these parameters will be fixed and the missing parameters are calculated such that the QoS Requirement can be satisfied by the system. For the analysis presented, it is further assumed that all steps of the PRACH procedure occur before the next PRACH contention windows. That implies that every UE knows whether its last PRACH procedure was a success. Successful connected UEs will therefore not try to connect again. Further this implies, that it is sufficient to evaluate the number of times each preamble has been chosen to get the number of successful PRACH Procedures. However, not connected UEs will all try the PRACH Procedure (depending on ACB) in the next PRACH contention window. For dynamic access barring policies, which change the admission probability p regulary, it is additionally assumed, that p can be reconfigured between each timestep With the above assumptions the RACH Procedure is essentially Modelled as an M-channel slotted ALOHA protocol where each Preamble is mapped to a channel. This enables the usage of performance analysis on M-channel slotted AlOHA instead.

B. Deterministic Analysis

The success of a given Preamble m in timeslot i is defined as

$$s_{i,m} = \begin{cases} 1 & \text{if chosen by 1 UE} \\ 0 & \text{otherwise} \end{cases}$$

Using this notation the number of successful connected UEs s_i in Timeslot i can be defined as

$$s_i = \sum_{m=0}^{M} s_{i,m}$$

The Backlog at Timeslot i can then be recursively defined as

$$B(i) = B(i-1) + a_i - s_i$$

The intuition of this formula is that the Backlog at time i depends on the Backlog at time$i-1$. Then some UEs may arrive additionally and some UEs might connect successful. In case ACB is used one can compute the expected number of admitted ues b' with

$$b' = (B(i-1) + a_i) \cdot p$$

Again the intuition is simply to compute the expected value for all UEs participating having a success with probability p, which is defined by the ACB

C. Probabilistic Analysis

Assuming a System with backlog b' the probability distribution for s_i can be described as:

$$P[s_i = k, b'_i = x] = \binom{x}{k}\binom{M}{k}\frac{k!}{M^x}$$
$$\times \sum_{j=1}^{\min(m-k,x-k)} (-1)^j \binom{M-k}{j}\binom{x-k}{j} j!(M-k-j)^{x-k-j}$$

With that, the service can be described as Moment Generating Function as follows:

$$\overline{M}_S(\theta) = \left(\sum_{k=0}^{M}\sum_{x=0}^{N}\frac{M^x e^{-m}}{x!}P[s_i = k, b'_i = x]e^{-\theta k}\right)$$

One can use this to compute the burst resolution violation probability as follows

$$\epsilon(b^\epsilon, t) = \inf_\theta \left\{ e^{-b^\epsilon\theta}\cdot(e^{\theta N}\overline{M}_S(\theta)^t + \overline{M}_S(\theta)\frac{1-\overline{M}_S(\theta)^{t-1}}{1-\overline{M}_S(\theta)}) \right\}$$

This formula enables a fast computation of the burst resolution violation probabilty compared to the recursive formula in III-B. However, this formula is only accurate for $b^\epsilon >> M$ and becomes very conservative when $b^\epsilon \to M$. For that reason this formula can not be used directly. The usage of the formula is restricted to $B(i) \geq cM$ with the following formula:

$$\begin{aligned} P[t_1 + t_2 \geq t] &= P[t_1 \geq t - x]P[t_2 = x] \\ &\leq \epsilon(cM, t-x)P[t_2 = x] \end{aligned}$$

The intuition of this formula is, that time is partitioned into 0 to t_1 and t_2. The Backlog is also partioned into N to cM and cM to 0 where $c > 1$ is a new variable, which is set before calculation. This allows to compute the Backlog probability to reach Backlog cM with formula 24 which is fast. Then only the backlog probability to reduce the remaining Backlog cM has to be computed using the recursive Formula in III-B. Because the Backlog has been significantly reduced to $cM << N$, this will be fast in comparison to calculating the whole error probability recursively. The Authors further showed, that their proposed solution get results close to real world examples by running simulations in OMNeT++, which is a framework for simulating large communication networks. OMNeT++ does also simulate physical properties therefore not relying on assumptions about the physical layer. However the authors do not explain, whether reaction times and schedules are simulated or if the assumptions from III-A are used.

D. Discussion

Most of the Assumptions in III-A make sense in respect to real world system. One might criticise, that in many cases the UE will not have finished the RACH Procedure before the next RACH contention window begins. The maximum response time for RACH is between $3ms$ and $12ms$ as is specified in SIB2. In practice, the response time will be more towards the upper end of that spectrum because decoding, reacting, and creating the response takes some time. In previous experiments using an NUC11, an Intel® Core™ i7-1165G7 Processor, SrsRAN and USRP B210 I measured a minimum achievable reaction time of 7ms. This leads me to believe that reaction times will rather be in the range of $7ms$ to $12ms$ than lower. This also implies that the RACH Procedure will not be finished in most cases, because almost all RACH contention window schedules have a lower periodicity. However, this will only defer the burst resolution by a few frames, because this simply means, that UEs with a successful RACH procedure might leave one or 2 contention windows later.

IV. Simulation

For this seminar I implemented a simulation on the recursive backlog formula in section III-B. To improve the performance the formula was implemented as an Iterator instead of a recursive Function. Using an Iterator removes unnecessary function calls and minimizes the memory profile therefore allowing large scale simulations in a parallel execution. The behaviour of the iterator is shown in algorithm 2 Generating

Algorithm 2 Simulation Loop

```
Initialize N and p_0 = 1/M
while B > b^ε or t_c < t do
    t_c += 1
    draw B samples from a binomial distribution to get admitted ues a
    draw a random integer samples from a uniform distribution [0..M]
    count the number of occurences of each preamble
    s_i = number of preambles chosen exactly once
    B -= s_i
    update p for dynamic ACB policies
end while
if B > b^ε then
    return success
else
    return failure
end if
```

large samples is done using Numpy in order to use the performance of the underlying C++ framework as much as possible. Simulations over ranges of parameters are implemented using a Process Worker Pool. Each task is characterized by a tuple

$$(sc, qos, nr\_simulations)$$

and submitted to the worker pools. sc and qos are implemented similar to their definition in III-A. From there multiple workers can take executed Task and run it independently. This requires each task to be completely independent of each other. By design simulations should not interfere with each other. However, extra care has to be taken, that no shared objects are used which would slow down calculation and would invalidate

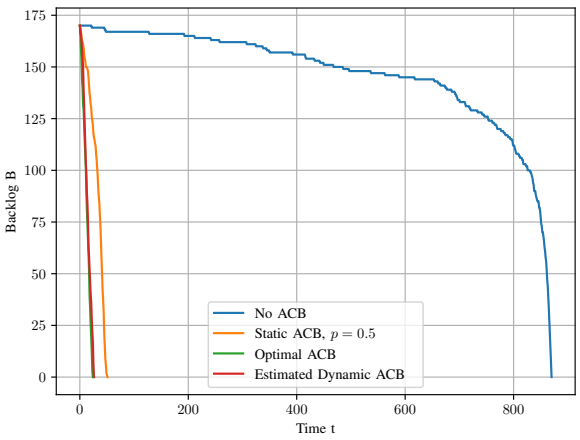

Fig. 1. Backlog over Time for different ACB, M = 20, N = 170

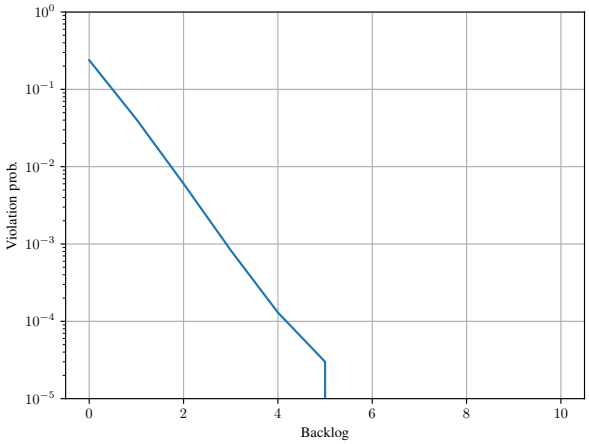

Fig. 2. violation Probability vs. Backlog, $M = 30$, $N = 100$, $t = 15$, static ACB $p = 0.5$, 100000 Samples

simulation results. A Process Pool worker is used instead of a Thread Pool Worker to avoid the Global Interpreter Lock (GIL). GIL is a mechanism, that restricts the usage of the python interpreter to one thread at the time. From a programming standpoint, GIL can be seen as a kind of mutex, essentially halting all other threads until the current thread is finished. The purpose of GIL is to guarantee consistency of python objects and prevent side effects or nondeterministic behaviour, which might occur because of parallel execution. Since the Simulations tasks are independent of each other, GIL is not needed and should be prevented. Spawning of multiple Processes however costs extra resources. For that reason, each Task consists of a given number of simulations. This avoids spawning a new Process for every simulation instance, which would again slow execution down. The typical use case for Thread Pools is usually IO limited tasks, justifying their existence for different types of large scale operations.

V. Simulation Results

Figure 1 shows the typical course of the Backlog over Time for the ACB Policies described in II-C. We observe that the given System is at its limit when using no ACB, which has exponentially increased the burst resolution time. This can be solved using ACB. The Estimated dynamic ACB Policy is also very close to the optimal solution. Next we can analyse the violation probability ϵ for different allowed backlogs b^ϵ Again we observe exponential behaviour as the system reaches the maximum QoS it can deliver. The vertical line at Backlog = 5 implies that for $b^\epsilon \geq 5$ the violation probability was always 0: $\epsilon = 0$ This simulation results supports the findings in **??**. However, the authors forget to explicitly state that $t = 15$ which made finding this configuration difficult.

Figure 3 then shows the maximum Number of UEs a cell with 10 Preambles can support, for different violation probabilities. We can observe exponential behaviour here aswell implying that with higher number of UEs the violation

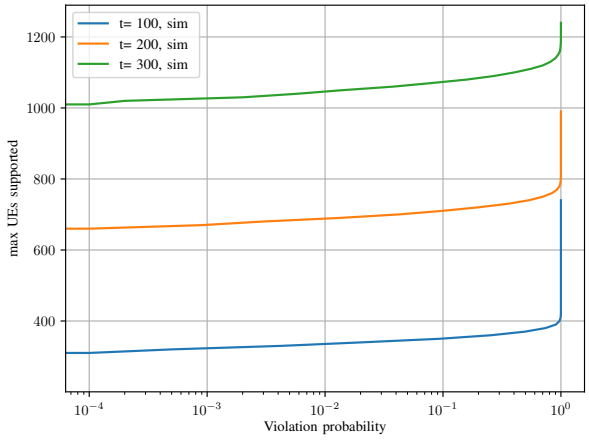

Fig. 3. maximum number of supported UEs vs total burst resolution violation probability for M = 10

probability for a given QoS will increase exponentially. This simulation uses the optimal ACB policy therefore the Results are only theoretical. Nevertheless, estimated dynamic access barring has performed very well compared to the theoretical optimum and can therefore reach Values close to the theoretical upper limit

VI. Conclusion

In this Paper, I presented the necessary background to understand the LTE Startup Procedure and analyse the RACH Procedure. For Burst Arrivals of large number, I showed how one can compute the Backlog over time and how this can implemented in a simulation. Further I summarized the results of **??**. The simulations done in this work support the findings of the paper in every aspect. For one case, the missing

parameters were calculated. Future work can further improve the analysis method into finding a framework, which allows for even tighter computation of performance bounds. Addition work is also necessary to find good Values for c which is essential for the analysis of the violation probability of QoS Requirements. Finally, there exist more techniques than ACB, which can improve burst resolution time. These additional techniques still need to be added into the analysis framework to give complete insights into the possible performance bounds for the RACH Procedure.

REFERENCES

ABBREVIATIONS AND ACRONYMS

ACB	Access Barring
eNodeB	evolved NodeB (Basestation)
LTE	Long Term Evolution
MIB	Master Information Block
PRACH	Physical Random Access Channel
PRB	Physical Resource Block
PSS	Primary Synchronization Signal
RRC	Radio Resource Control
SIB	System Information Block
SSS	Secondary Synchronization Signal
UE	User Equipment