Departamento de Ciências e Tecnologias da Informação

DEDUPLICAÇÃO SEGURA EM AMBIENTES MÓVEIS

Luis Filipe Mendes Marques

Dissertação submetida como requisito parcial para obtenção do grau de
Mestre em Software de Código Aberto

Orientador:
Doutor Carlos J. Costa, Professor Auxiliar,
ISCTE - Instituto Universitário de Lisboa

Outubro, 2013

Agradecimentos

Ao dar por concluído este trabalho, que constitui simultaneamente um processo de enriquecimento pessoal e de desenvolvimento de novas competências, gostaria de registar o meu profundo apreço a todos quantos, de diferentes formas, me apoiaram na sua concretização.

Em primeiro lugar, o meu reconhecimento é dirigido ao Professor Doutor Carlos Costa, pelo empenho com que orientou este trabalho nas diversas fases, pela sua disponibilidade, pelo rigor e pela oportunidade das críticas e sugestões.

À Luísa, João e Mafalda pelo constante apoio, motivação e paciência.

A todos os amigos e familiares que me cederam os seus dispositivos, sem os quais não me tinha sido possível realizar este trabalho, manifesto o meu agradecimento pela sua compreensão e disponibilidade.

Resumo

A utilização crescente de dispositivos móveis como *smart phones, PDAs* e *tablets* com capacidades de armazenamento e processamento a crescer rapidamente, permite aos utilizadores transportar um crescente volume de informação. O valor desta informação tem de ser protegido de perdas, roubos ou qualquer outro tipo de acidente que possa ocorrer com os dispositivos, tornando o processo de cópias de segurança um fator chave.

Nos dispositivos móveis a transmissão de dados tem um elevado custo, em preço e energia, e estes dispositivos dependem de baterias, otimizar a utilização de largura de banda é crucial. A deduplicação permite uma redução substancial do volume de dados a serem transmitidos sobre a rede em sessões de cópias de segurança, identificando blocos de dados semelhantes, transferindo-os e armazenando-os apenas uma vez.

Por outro lado, habitualmente os dispositivos móveis usam redes móveis, e o armazenamento na nuvem é uma solução viável para guardar dados de cópias de segurança. Por consequência é essencial a utilização de processos criptográficos para proteger a informação durante a transmissão, mas também no armazenamento.

Neste trabalho avalia-se a viabilidade da utilização de dispositivos móveis no processamento de cópias de segurança, empregando a deduplicação e criptografia simultaneamente.

Palavras chave

Cópias de segurança, Segurança, Deduplicação, Dispositivos móveis, Energia.

Abstract

An increased use of mobile devices such as smart phones, PDAs and tablet computers with storing and processing capacities is rapidly increasing ,allowing users to carry a bigger volume of data along with them. The value of this information must be protected from loss, theft or any kind of accident, which may occur with the devices, making the backup process a key factor.

In what concerns mobile devices, the data transmission has a high cost, not only in price, but also in terms of energy. On the other hand these devices depend on batteries, so the optimization of the bandwidth use is crucial. Deduplication allows substantial reductions in the data volume to be transmitted over network on backup sessions, by identifying common amounts of data, transferring and storing them only once.

On the other hand, mobile devices usually use mobile networks and cloud storage is a viable solution for data backup. Thus the use of cryptographic processes to protect data is essential, not only during the transmission, but also in what concerns storage.

This work evaluates the viability of the use of mobile devices, in what deals with the processing of backup, by using deduplication and encryption, simultaneously.

Keywords

Backup, Security, Deduplication, Mobile devices, Energy.

Índice

Índice de tabelas

Índice de gráficos

Índice de figuras

Lista de abreviaturas

% - Percentagem

AES - Advanced Encryption Standard

BTF - Blocos Tamanho Fixo

BTV - Blocos Tamanho Variável

CPU - Central Processing Unit

CTR - Counter

Et al. – e outros

FSH – Fixed Size Hashing

GB - GigaByte

Gb - Gigabit

GHz – GigaHertz

HSDPA - High Speed Downlink Packet Access

KB - KiloByte

KBs – KiloByte por segundo

MB - MegaByte

Mb - Megabit

MHz - MegaHertz

mAh - miliampere-hora

RAM - Random Access Memory

PDA - Personal Digital Assistant

SHA - Secure Hash Algorithm

SIM - Subscriber Identity Module

UML - Unified Modeling Language

VSH – Variable Size Hashing

WAN - Wide Area Network

WFH – Whole File Hashing

1 Introdução

1.1 Enquadramento e Motivação

O volume de informação utilizado quer nas organizações quer em termos individuais tem evoluído de uma forma acelerada, o que associado ao decréscimo do custo por byte dos dispositivos de armazenamento, tem provocado uma utilização do espaço de armazenamento por parte dos utilizadores como um recurso quase ilimitado. Por outro lado, a utilização crescente de equipamentos ultra-portáteis, como *smart phones*, *PDAs* e *tablets* com capacidades de armazenamento e processamento em rápido crescimento, tem permitido aos utilizadores transportarem um volume crescente de informação (Gantz et al., 2008). Acresce ainda a necessidade de desta informação ser salvaguardada em *backups*, o que faz com que a necessidade de espaço para cópias de segurança vá crescendo de forma exponencial.(Zhu, Li, & Patterson, 2008)

A utilização crescente de sistemas de *backup* baseados em discos faz com que se possam utilizar outros métodos de otimização dos dados armazenados, que não seriam viáveis em sistemas como *tapes*, para eliminar a redundância dos dados a arquivar.(Zhu et al., 2008)

Por outro lado, também se assiste a uma cada vez maior utilização de recursos deslocalizados para arquivo de dados, existindo um crescente número de serviços disponibilizados através da Internet para este fim. A utilização destes recursos externos levanta contudo a problemática da segurança, quer durante a transmissão dos dados, quer no seu armazenamento, bem como a utilização de largura de banda necessária para a transmissão dos dados a arquivar remotamente (Zhu et al., 2008). Este fator tem uma elevada importância quando se está a falar de dispositivos móveis, em que a utilização da rede móvel para transmissão de dados têm associado elevados custos.

A deduplicação é um processo que permite eliminar a redundância de dados, obtendo-se uma otimização do espaço necessário para o seu armazenamento(Park & Lilja, 2010). No caso das cópias de segurança é um fator com elevado peso dado a necessidade de retenção de dados em vários momentos temporais. Existe ainda normalmente uma elevada percentagem de dados que não é alterada entre os vários momentos em que são efetuadas as cópias de segurança. (Nie et al., 2010; Tan, Feng, Yan, & Zhou, 2010) Associado à menor necessidade de espaço

para arquivo está também uma necessidade inferior de transmissão de dados quando utilizados sistemas remotos de armazenamento.(Meister & Brinkmann, 2009)

Quando são utilizados serviços de armazenamento disponibilizados via Internet por terceiros (*cloud storage*), é necessário recorrer à encriptação dos dados a armazenar. Esta tem por propósito proteger a informação. Os processos criptográficos tem por objetivo ocultar por completo qualquer redundância dos dados produzindo a partir de um determinado bloco de dados um resultado aparentemente aleatório em relação aos dados originais (Storer, Greenan, Long, & Miller, 2008).

Tanto a deduplicação como a criptografia recorrem a sistemas matemáticos de maior ou menor complexidade, e consequente necessidade de processamento, o que em dispositivos móveis levanta dois problemas: a capacidade de efetuar o processamento em tempo útil para que possa ser utilizado num sistema de cópia de segurança, e a quantidade de energia necessária para efetuar esse processamento, dado que estes dispositivos são normalmente alimentados por baterias, portanto dispõem de uma capacidade limitada de energia.

1.2 Âmbito de Intervenção e Objetivos

A generalidade dos estudos efetuados sobre deduplicação baseia-se na eficiência do processo, na deteção de duplicação de dados e na performance da identificação de redundância.

Contudo quando se pretende estender estes processos a dispositivos móveis, fatores como o consumo energético, necessário para efetuar o processamento exigido na deduplicação e encriptação, assim como o volume de dados a ser transmitidos, assumem uma relevância extrema. Nos dias de hoje os conceito de poupança energética e tecnologias verdes merecem especial atenção, por isso a utilização de sistemas de arquivo de backups mais eficientes tem um valor muito elevado. (He, Li, & Zhang, 2010)

Nesta dissertação pretende-se dar resposta à seguinte questão:

Qual a melhor solução para deduplicação segura de dados em ambientes móveis?

Traçaram-se como objetivos específicos :

- Propor solução que permita realizar deduplicação segura.
- Avaliar qualidade dos diversos métodos de deduplicação.

Pretende-se testar a viabilidade da execução de processos de cópias de segurança recorrendo a métodos de deduplicação segura em dispositivos móveis, através da análise dos seguintes fatores :

- Consumo energético.
- Volume de dados transmitidos.
- Tempo de execução.
- Segurança dos dados arquivados (encriptação).
- Utilização do espaço para arquivo. (Meta-dados)

Pretende-se identificar qual o peso de cada fator, para cada um dos métodos de deduplicação e verificar a viabilidade da utilização para cada um dos métodos. É também pretendido detetar quais dos fatores que podem impedir ou invalidar a utilização de algum dos métodos de deduplicação.

1.3 Abordagem Metodológica

O trabalho de investigação iniciou-se com a revisão da literatura referente ao processo de deduplicação, métodos de duplicação e suas características, bem como da utilização conjunta de processos de deduplicação e criptografia. Esta revisão permitiu criar uma solução conceptual, onde se apresenta um modelo aplicacional para a execução de duplicação segura.

Com o desenvolvimento do protótipo valida-se a exequibilidade da proposta conceptual em ambientes móveis, enquanto que na avaliação é possível não só validar a qualidade da solução como também testar a execução dos diversos métodos de deduplicação segura, sobre conjuntos de ficheiros, com o objetivo da obtenção de informação sobre cada um dos métodos de duplicação.

De entre os sistemas disponíveis para dispositivos móveis foi selecionado o sistema Android, por estar bastante divulgado e presente numa grande diversidade de equipamentos, podendo por isso serem testados dispositivos com características diversas, para que os resultados obtidos tenham uma maior aderência à realidade.

A escolha do Android também se deveu ao facto de este ser um sistema Open-Source, logo no âmbito do mestrado para o qual esta dissertação foi realizada.

Para a execução dos testes existiu a necessidade de desenvolver duas aplicações. A primeira desenvolvida para ser executada em dispositivos com sistema Android, onde é efetuado o processamento de cópia de segurança recorrendo à deduplicação e criptografia. A segunda aplicação, a ser executada do lado do servidor foi desenvolvida em Java de modo a poder ser corrida em diversas plataformas. Foi ainda desenvolvido um protocolo de comunicação de modo a que as aplicações no cliente e no servidor pudessem comunicar entre si.

A aplicação cliente foi criada de modo a que se pudesse obter dados sobre o consumo de energia durante o processamento, os tempos gastos em cada uma das fases do processamento, assim como os volumes de dados processados, duplicados e transmitidos.

Do lado do servidor a aplicação recebe e valida os dados enviados pelo cliente, assim como arquiva a informação respetiva a cada processamento efetuado.

No final foi efetuado o tratamento dos dados recolhidos do modo considerado mais ajustado à análise pretendida.

1.4 Estrutura

O presente trabalho de investigação está estruturado da seguinte forma: introdução (capítulo 1); revisão da literatura (capítulo 2); proposta de solução conceptual (capítulo 3); desenvolvimento de protótipo (capítulo 4); avaliação preliminar (capítulo 5); conclusões (capítulo 6); trabalhos futuros e limitações(capítulo 7).

No capitulo 2 é apresentado o enquadramento teórico relativo ao sistema e métodos de deduplicação, processos que permitem otimizar o processamento da deduplicação, utilização de deduplicação e criptografia em simultâneo e consumo de energia na transmissão de dados em dispositivos móveis.

O modelo aplicacional a desenvolver para executar o processo de deduplicação segura, é proposto no capitulo 3

No capitulo 4 é descrito o modo como foram desenvolvidas as aplicações usadas, quais as tecnologias e procedimentos utilizados para dar corpo ao modelo apresentado no capitulo anterior.

No capitulo seguinte são apresentados os resultados obtidos nos vários processamentos efetuados.

As conclusões desta dissertação estão descritas no capítulo 6 e, no último capítulo, são apontadas pistas para trabalhos futuros.

2 Revisão da Literatura

O volume de informação arquivada digitalmente tem tido um elevado crescimento e a produção de informação digital tem um crescimento ainda maior, com um fator de incremento de 10 vezes em cada 5 anos, superando já a capacidade de armazenamento disponível. (Gantz et al., 2008)

Existindo elevado grau de duplicação e redundância nos dados arquivados, não só em ficheiros idênticos como em conteúdos que se repetem em diversos ficheiros. Assume especial relevância em sistemas de backup ou de controle de versões, o que requer uma enorme quantidade de espaço extra para armazenamento. (Liu et al., 2008)

Também o aumento da utilização de dispositivos móveis como telemóveis, *PDA's* e *Tablets*, com capacidade de armazenamento local, mas com a necessidade de se conectarem aos sistemas de armazenamento das organizações, é um dos fatores que contribui para que o volume de armazenamento necessário cresça 50% ao ano nas empresas. (Gantz et al., 2008)

Quando se associa esta utilização de sistemas remotos para arquivo recorrendo a redes públicas a necessidade de largura de banda é enorme. (Zhu et al., 2008)

2.1 Deduplicação

A deduplicação pode ser definida como um processo que identifica sequências idênticas de bytes nos dados a arquivar, guardando apenas uma instância desse conjunto de dados, e sempre que encontra uma sequência idêntica apenas é necessário guardar a posição do bloco de dados idêntico arquivado anteriormente. (He et al., 2010; Storer et al., 2008)

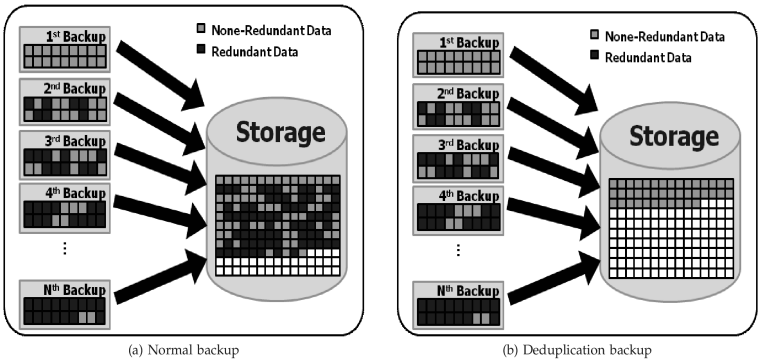


Figura 1: Deduplicação de dados (Min et al., 2011)

O principio de funcionamento deriva dos métodos de compressão de dados na eliminação de redundância, mas enquanto nos sistemas de compressão quando aplicados a ficheiros idênticos, estes são processados individualmente e são arquivadas as várias cópias, na deduplicação apenas uma cópia é arquivada. (He et al., 2010) É ainda possível através da deduplicação, remover dados redundantes de um modo eficiente quer inter-ficheiros quer intra-ficheiros, em grandes conjuntos de ficheiros. (Park & Lilja, 2010; Won et al., 2008)

Ao contrário do processo de *Delta encoding* que utiliza um ficheiro primário e para as alterações seguintes desse ficheiro guarda apenas as secções alteradas em relação ao ficheiro primário, processo utilizado em sistemas de controle de versões, obrigando ao acesso ao ficheiro original para poder obter as várias versões ulteriores, na deduplicação não existe tal necessidade. (Kulkarni, Douglis, LaVoie, & Tracey, 2004; Liu et al., 2008; Mandagere, Zhou, Smith, & Uttamchandani, 2008; You & Karamanolis, 2004)

Em sistemas de *backup* a maioria dos dados arquivados não é alterada entre as várias sessões do backup, o que conduz a uma elevada redundância dos dados a arquivar, a utilização de métodos de deduplicação permite minimizar o volume de informação a arquivar. (Nie et al., 2010) Também o volume de dados que é necessário transferir para os sistemas de arquivo é reduzido, fazendo portanto uma utilização mais racional da largura de banda (Meister & Brinkmann, 2009) e permitindo reduzir o tempo utilizado para efetuar cada *backup*. (Won et al., 2008)

Com a redução substancial do volume de dados a serem efetivamente arquivados em cada sessão de backup, a possibilidade de utilização de cópias para sistemas remotos através de

redes WAN, torna-se praticável, permitido obter cópias deslocalizadas para casos de catástrofe. (Zhu et al., 2008)

A deteção de duplicação de dados não pode ser efetuada por comparação byte a byte por motivos de performance do processo, em vez disso utiliza-se uma comparação por assinaturas, sendo a probabilidade de uma ocorrência de erro motivada por uma falha de hardware superior à probabilidade de uma falsa identificação de duplicação. (Meister & Brinkmann, 2009; Zhu et al., 2008)

Podem ser utilizadas várias funções *Hash* para gerar as assinaturas como MD5 (120bits), SHA-1 (160bits) ou SHA-256 (256bits), a resistência a colisões cresce consoante o maior tamanho da assinatura, mas também a necessidade de processamento é acrescida para assinaturas de maior dimensão. (Won et al., 2008)

A função SHA-1 é uma das mais utilizadas para a produção das assinaturas, dada a sua resistência a colisões e a sua eficiente utilização de processador. (Quinlan & Dorward, 2002)

Se considerarmos um volume de informação de 1 exabyte (18^{10} bytes) preenchido com blocos de 8 Kbytes, temos um número aproximado de 10^{14} de blocos, utilizando a função SHA-1 para calcular as assinaturas dos blocos temos uma probabilidade de 10^{-20} de ocorrer um erro de falsa duplicação de dados, o que se pode considerar como altamente improvável e como tal pode ser ignorado. (Quinlan & Dorward, 2002)

2.2 Métodos de deduplicação

O processo de deduplicação mais simples atua ao nível dos ficheiros, utiliza o ficheiro completo para verificar a existência de duplicação, no caso de já existir um ficheiro idêntico apenas guarda a localização desse ficheiro, este processo é também referenciado como *Whole-File Chunking*, *Whole File Hashing* ou *Unique Document Storage* (He et al., 2010; Kulkarni et al., 2004; Liu et al., 2008; Mandagere et al., 2008; Meister & Brinkmann, 2009).

O método de deduplicação ao nível de ficheiros utiliza um baixo nível de meta-dados, e quando é necessário recuperar um ficheiro, não necessita de o reconstruir previamente, ao contrário de outros métodos. (He et al., 2010) É também o método que menos recursos de processamento necessita. (Mandagere et al., 2008)

Outros processos recorrem a utilização de blocos ou secções (chunks) de ficheiros para identificarem semelhanças, arquivando apenas os blocos diferentes. Podem ser utilizados dois tipos de blocos, com tamanho fixo ou tamanho variável (Kulkarni et al., 2004; Liu et al., 2008; Mandagere et al., 2008; Meister & Brinkmann, 2009; Min, Yoon, & Won, 2011; Won et al., 2008)

Os métodos de duplicação que utilizam blocos no seu processamento, necessitam também de armazenar os meta-dados referentes a cada bloco arquivado, de modo a poderem reconstruir o ficheiro partindo dos blocos arquivados. (He et al., 2010)

A utilização de blocos bem como o tamanho dos blocos tem grande impacto em termos de tempo necessário para reconstruir os ficheiros, especialmente quando os ficheiros arquivados estão repartidos em muitos blocos. (Mandagere et al., 2008)

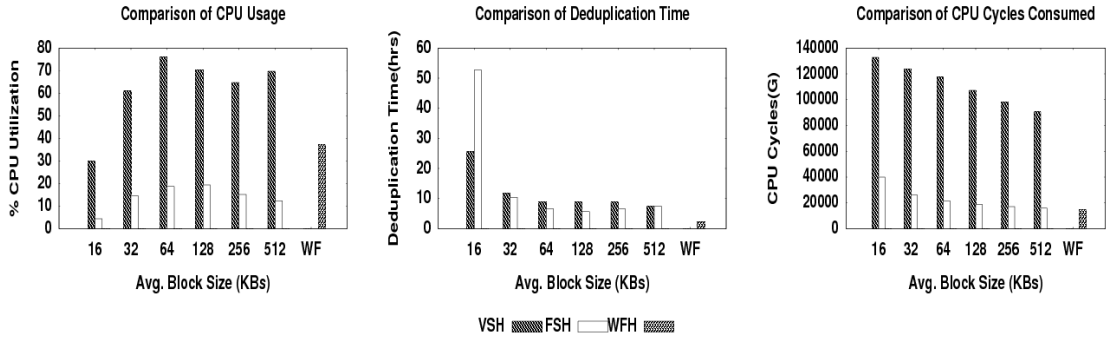


Figura 2: Comparação de consumo de CPU (Mandagere et al., 2008)

A utilização de blocos de tamanho fixo é conceptualmente mais simples e menos exigente em temos de recursos. Divide sempre os ficheiros em blocos do mesmo tamanho e verifica a existência de blocos idênticos, arquivando apenas blocos únicos. (Mandagere et al., 2008; Nie et al., 2010)

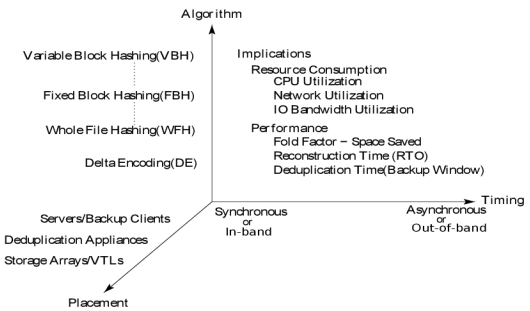


Figura 3: Espaço de conceção da deduplicação (Mandagere et al., 2008)

A utilização de blocos de tamanho fixo é contudo bastante afetada por pequenas alterações de edição nos ficheiros que implicam a movimentação do conteúdo, deste modo apesar do ficheiro poder ser bastante semelhante à versão anterior, os blocos em que vai ser dividido poderão ser todos diferentes dos gerados em versões anteriores. (Nie et al., 2010; Won et al., 2008)

Quando são utilizados blocos nos processos de deduplicação, para se obter uma otimização da utilização do espaço de armazenamento o tamanho dos segmentos deve ser igual ou múltiplo do tamanho dos blocos do dispositivo utilizado para armazenamento. (Nie et al., 2010; Zhu et al., 2008)

O sistema de blocos de tamanho variável gera uma série de identificadores para um segmento do ficheiro, de modo a identificar semelhanças e definir o tamanho do bloco a ser processado, este processo é também referenciado como *Contend-Defined Chunking* (CDC). (Meister & Brinkmann, 2009; Nie et al., 2010)

A utilização de sistemas baseados em blocos de tamanho variável é mais eficiente ao nível da deteção de redundâncias. (Meister & Brinkmann, 2009). Contudo este método de deduplicção é bastante mais exigente em termos de necessidade de processamento, gastando 70% a 80% do tempo de execução para otimizar a identificação de blocos de dados idênticos. (Won et al., 2008)

2.3 Processos de otimização da deduplicação

A utilização de blocos no processo de deduplicação permite um maior grau de eliminação de redundância. Quanto menores forem os blocos utilizados no processo maior é a deteção de redundância e é também maior o número de blocos arquivados assim como o volume de meta-dados necessários para guardar a informação de todos os blocos arquivados. (Anderson & Zhang, 2010; Wei, Jiang, Zhou, & Feng, 2010; Zhu et al., 2008) No caso dos métodos que recorrem à segmentação, a utilização de parâmetros específicos consoante o tipo de ficheiro, permite também aumentar a identificação de redundância. (Liu et al., 2008; Meister & Brinkmann, 2009; Nie et al., 2010)

Um dos processos de aumentar o desempenho do processo de deduplicação consiste em dispersar a informação arquivada, quer índices, quer blocos de dados, por vários dispositivos, beneficiando de maior velocidade de acesso aos dados. (Wei et al., 2010)

Quando é necessário verificar se existe uma determinada assinatura, o processo de deduplicação é obrigado a percorrer os índices dos blocos já arquivados para verificar a sua existência. Como não podem ser mantidos em memória todos os índices, isso implica operações de acesso ao disco, pelo que o modo como os índices são organizados tem um grande impacto no desempenho do processo.(Tan et al., 2010; Wei et al., 2010)

Existem vários estudos propondo sistemas otimizados para indexação de modo a obter uma melhor performance em processos de deduplicação.

No processo de *backup* a maioria de dados redundantes (94.7%) encontra-se entre as várias cópias de segurança de cada utilizador, (Tan et al., 2010), pelo que os autores citados propõem um sistema de índice hierarquizado quanto ao proprietário dos dados arquivados.

Quando se processam cópias de segurança, a sequência com que os dados são apresentados para serem arquivados é normalmente semelhante, assim se os índices forem organizados de modo a identificarem esta sequência, pode-se minimizar o número de consultas necessárias para encontar uma assinatura já existente. (Zhu et al., 2008)

Outro dos processos para melhorar o desempenho consiste na utilização do filtro *Bloom* (Bloom, 1970) evita acessos desnecessários ao índice, identificando de um modo rápido segmentos que não se encontram nas tabelas de índice. (Park & Lilja, 2010; Wei et al., 2010; Zhu et al., 2008)

Do lado do cliente também há propostas para otimizar o processo.

A utilização de índices do lado do cliente permite verificar alterações em relação à última sessão de backup sem ser necessário efetuar qualquer troca de informação com o servidor, (Anderson & Zhang, 2010) trazendo melhorias no tempo de execução e de utilização da largura de banda.

A utilização do algoritmo Modulo-K permite uma melhoria de aproximadamente 40% em termos de velocidade para identificar semelhanças em blocos de tamanho variável. (Won et al., 2008)

2.4 Deduplicação e segurança dos dados arquivados

A utilização de encriptação em conjunto com a deduplicação levanta alguns problemas dado que os dois processos têm objetivos opostos. Quando se utilizam chaves diferentes para encriptar o mesmos dados obtêm-se resultados diferentes, assim sendo não é possível identificar a sua semelhança. (Anderson & Zhang, 2010; Storer et al., 2008)

O modo de garantir a coexistência da encriptação e da deduplicação baseia-se na encriptação convergente, que consiste em gerar as chaves de encriptação a partir dos dados a serem encriptados. (Anderson & Zhang, 2010; Storer et al., 2008)

Deste modo para blocos de dados idênticos obtêm-se chaves idênticas, independentemente de onde se processa a encriptação, para um determinado bloco de dados resulta sempre num bloco cifrado semelhante.(Storer et al., 2008; Wang, Qin, Peng, & Wang, sem data)

A utilização de encriptação dos dados do lado do cliente garante o sigilo da informação durante a transmissão e no sistema onde os dados são arquivados mesmo que a segurança deste seja comprometida. (Storer et al., 2008)

2.5 Transmissão de dados e consumo energético em dispositivos móveis

A grande maioria dos dispositivos móveis tem capacidade de utilizar vários tipos de comunicação sem fios, (Wi-Fi, GSM, 3G) no entanto as velocidade de transmissão e o consumo energético para cada um destes é diferente.(Balasubramanian, Balasubramanian, & Venkataramani, 2009). Para além do tipo de rede o consumo energético nas comunicações sem fios está também dependente de fatores como, força do sinal, tempo de conexão e volume de dados transferidos. (Rahmati & Zhong, 2007)

O modo como a transmissão de dados são efetuadas também tem impacto no consumo de energia, não estando este consumo exclusivamente relacionado com o volume de dados transmitidos. Um pequeno volume de dados transmitido intermitentemente pode consumir

mais energia do que um grande bloco de dados, transmitido de uma só vez.(Balasubramanian et al., 2009)

Na comunicação 3G uma grande parte da energia é consumida porque o dispositivo mantém um estado de alta-energia por vários segundos após a conclusão da transferência de dados (*tail energy*), para manter a ligação estabelecida para comunicações futuras, e também, mas em menor escala, para passar para um estado de alta-energia antes de iniciar a comunicação (*ramp energy*). Estes consumos (*tail energy* e *ramp energy*) são constantes podendo ser amortizados em comunicações de grande volume de dados, ou em comunicações intermitentes frequentes. (Balasubramanian et al., 2009)

O sistema GSM exibe um cenário idêntico, mas apresenta tempos em estado de alta-energia bastante inferiores ao 3G. Contudo a velocidade de transmissão é muito inferior, o que implica um grande consumo de energia, em relação à quantidade de dados transferida. Com taxas de transferência mais elevadas o Wi-Fi tem o menor custo energético por Mb transferido, quando comparado com rede moveis. Apesar disso a energia gasta na busca de redes, ou na manutenção da conexão é alta. (Balasubramanian et al., 2009; Rahmati & Zhong, 2007)

Manter as comunicações Wi-Fi desligadas e só ativar quando é efetivamente necessário, aumenta substancialmente o tempo de utilização da carga da bateria nos dispositivos móveis. (Rahmati & Zhong, 2007)

2.6 Síntese

A cada vez maior utilização de dispositivos móveis bem como a sua cada vez maior capacidade de armazenamento faz com que a necessidade de salvaguardar essa informação seja cada vez mais premente.

O volume de dados alterados entre os momentos em que se efetuam as cópias de segurança, representa apenas uma pequena parte, em relação ao total da informação armazenada.

A utilização de processos de deduplicação na execução de cópias de segurança possibilita a deteção de dados duplicados de um modo eficaz, acrescentando um elevado grau de eficiência na sua execução.

A eficiência da deduplicação em sistemas de backup verifica-se ao nível da deteção de redundância de dados, o que tem influência na utilização de largura de banda e no tempo de execução destas tarefas, fatores que são elementos chave quando estamos a utilizar dispositivos moveis.

Podem ser utilizados vários métodos de deduplicação, tendo cada um destes diferentes impactos ao nível da deteção de redundância, necessidade de processamento, consumo energético e no volume de meta-dados.

A deduplicação ao nível de ficheiros é a menos eficiente na deteção de dados duplicados, mas tem uma necessidade de processamento baixa, e um muito pequeno volume de meta-dados.

A utilização de deduplicação por blocos de tamanho fixo em relação ao processamento por ficheiros apresenta uma maior taxa de reconhecimento de dados redundantes, mas necessita de um muito maior volume de meta-dados.

O método de deduplicação por blocos de tamanho variável apresenta o melhor resultado em relação à deteção de duplicações, em contrapartida tem um volume de meta-dados superior a todos os outros métodos e ainda uma necessidade de processamento bastante superior.

A utilização de serviços de armazenamento disponibilizados via Internet por terceiros, obriga a recorrer à encriptação do lado do cliente, de modo a ocultar a informação quer na transmissão quer no armazenamento.

A deduplicação e a encriptação têm objetivos inversos, o modo de garantir a sua utilização simultânea baseia-se na encriptação convergente, que consiste em gerar as chaves de encriptação a partir dos dados a serem encriptados.

Os dispositivos móveis utilizam vários tipos de comunicação sem fios, tendo cada um destes tipos características diferentes quer em relação à velocidade de transferência quer ao consumo energético de cada um deles. O modo como se executam as transmissões de dados também tem influencia na eficiência energética, podendo apresentar valores diferentes para transferências do mesmo volume de dados.

A realização de *backups* recorrendo a deduplicação e encriptação de dados utilizando dispositivos móveis, exige um sistema com capacidade de realização de todo o processamento

necessário para a deteção de duplicação, encriptação e transmissão de dados, utilizando apenas a energia disponibilizada pela bateria dos equipamentos.

3 Proposta de solução conceptual

Baseado nos conceitos apresentados na revisão da literatura foi criado o modelo aplicacional que permite a realização do procedimento de duplicação segura, em todas as suas fases de execução.

O modelo desenvolvido está repartido em 2 áreas, dispositivo móvel e servidor, sendo a primeira destas onde se realiza o procedimento de deteção de duplicação de dados, encriptação e envio. Enquanto no servidor é feita a receção, validação e arquivo dos dados.

O sistema de deteção de duplicações consiste na geração de uma assinatura dos dados em processamento seguido de uma verificação, no arquivo de meta-dados local, da existência de uma assinatura idêntica. (Meister & Brinkmann, 2009; Zhu et al., 2008) Quando não é encontrada nenhuma ocorrência da assinatura é efetuada a encriptação seguida do envio dos dados.

A geração de assinaturas e encriptação pode ser efetuada tanto sobre ficheiros como apenas sobre um segmento do ficheiro, permitido assim a utilização dos métodos de deduplicação por ficheiros ou por blocos de tamanho fixos e variável.

No servidor após a receção dos dados, enviados pelo cliente, é validada a coerência da informação recebida, quando considerados válidos os dados são arquivados. (Anderson & Zhang, 2010)

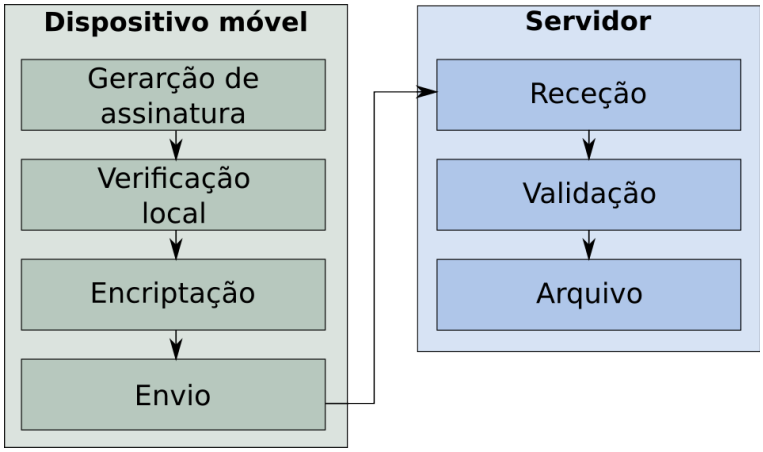


Figura 4: Modelo do protótipo

Com este modelo é possível realizar o procedimento de deduplicação em todos os métodos que se pretendem analisar. Garantido também a proteção dos dados durante a fase de transmissão e de arquivo no servidor, porque o processo de encriptação ocorre do lado do cliente. (Storer et al., 2008)

3.1 Aplicação para dispositivo móvel

A aplicação deve poder ser executada em qualquer dispositivo móvel, com sistema operativo Android, que disponha de capacidade de comunicação sem fios e capacidade de armazenamento suficiente para guardar o conjunto de dados a serem processados, assim como os respetivos meta-dados.

Esta aplicação terá capacidade de processar os vários métodos de deduplicação sobre o conjunto de ficheiros armazenados no dispositivo em pasta pré definida.

Deve ainda registar os dados obtidos durante a execução dos diversos processamentos relativos ao consumo energético, tempos de execução, volume de dados processados e volume de dados transferidos, após efetuar cada processamento deve enviar esses dados para o servidor.

3.1.1 Interação com o utilizador

O utilizador deve selecionar o tipo de processamento (método de deduplicação) a ser efetuado e iniciar o processamento. Após cada processamento pode decidir se pretende efetuar novo processamento ou terminar a aplicação. Para que possa ser efetuado o processamento o utilizador deve garantir a existência dos ficheiros, a processar, numa pasta pré definida.

Cabe ao utilizador ativar e parametrizar a rede sem fios que vai ser utilizada, quer seja uma rede Wi-Fi quer seja uma rede móvel, devendo a mesma estar conectada e pronta a comunicar, antes de ser iniciado qualquer processamento.

Previamente a qualquer processamento é necessário também que o utilizador configure o endereço do servidor com o qual vão ser efetuadas as transmissões, devendo o servidor no endereço designado estar em execução e à espera de conexões.

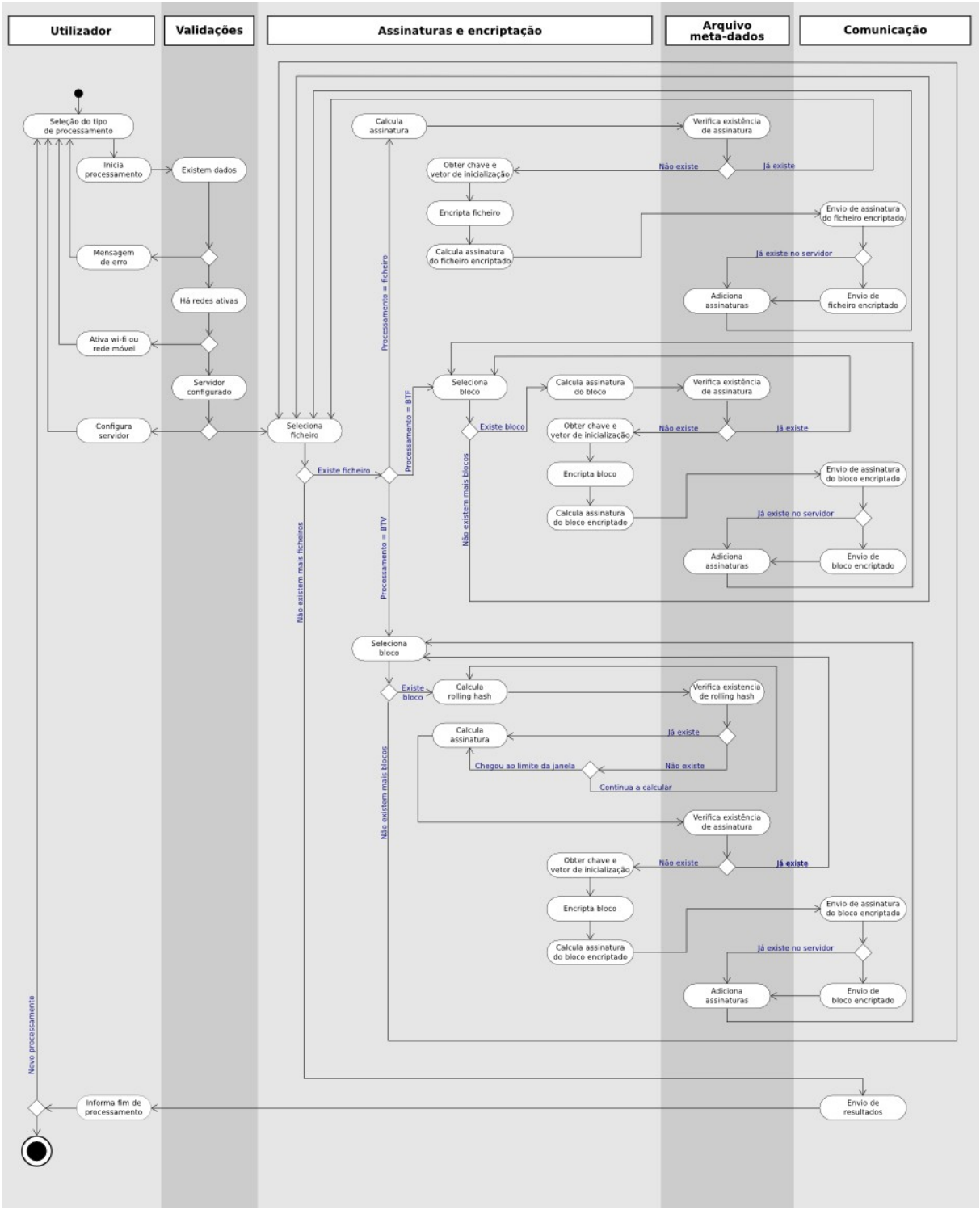


Figura 5: Diagrama de atividades utilizando UML Aplicação cliente

3.1.2 Validações

A aplicação valida se cada uma das condições necessárias para funcionamento se encontram válidas, no caso de alguma das condições não estar correta, informa o utilizador de qual o parâmetro incorreto e devolve o controlo ao utilizador.

3.1.3 Assinaturas e encriptação

O processamento é iniciado com a seleção de cada um dos ficheiros existentes na pasta predefinida e prossegue consoante o tipo de método de deduplicação selecionado.

No processamento por ficheiros é calculada a assinatura do ficheiro através de um algoritmo de *hash*, após a validação no arquivo de meta-dados da sua não existência, é obtida a chave e o vetor de inicialização a partir da assinatura anteriormente gerada. (Anderson & Zhang, 2010; Storer et al., 2008) No caso da assinatura gerada já existir no arquivo de meta-dados é selecionado um novo ficheiro, se existir, para processamento.

Segue-se a encriptação do ficheiro utilizando-se a chave e o vetor de inicialização obtidos anteriormente, sobre o ficheiro encriptado é calculada nova assinatura. A assinatura do ficheiro encriptado e o próprio ficheiro encriptado são enviados para a fase de comunicação.

Quando selecionado o processamento por blocos de tamanho fixo o ficheiro selecionado é dividido em blocos de um tamanho pré definido, para cada um dos blocos é calculada a assinatura, (Kulkarni et al., 2004; Mandagere et al., 2008; Nie et al., 2010) se for verificado a não existência da assinatura pela fase de arquivo de meta-dados, o bloco é encriptado utilizando-se a chave e vetor de inicialização obtidos da assinatura anteriormente gerada, sobre o bloco encriptado é criada uma nova assinatura. O bloco encriptado e a respetiva assinatura são enviados para a fase de comunicação.

Repetindo-se o processo para cada um dos blocos do ficheiro, quando terminado este processamento de blocos, é selecionado um novo ficheiro.

No processamento em blocos de tamanho variável, é selecionado um bloco cujo tamanho é igual à soma do valor pré definido utilizado no processamento em blocos de tamanho fixo mais o comprimento de janela também predefinido.

É calculada uma assinatura “fraca” utilizando um sistema de *rolling hash* para cada posição.

Cada um destes valores é verificado pela fase de arquivo de meta-dados, em caso de ser localizada uma assinatura "fraca" idêntica é então calculada a assinatura do respetivo bloco, para despistar falsos positivos. (Meister & Brinkmann, 2009; Nie et al., 2010)

Quando se atinge o limite da janela, sem ser localizada nenhuma assinatura idêntica, é selecionado o bloco de tamanho pré definido e sobre o qual é obtida uma assinatura. Após a obtenção da assinatura a sequência seguinte de processamento é idêntica ao processamento em blocos de tamanho fixo a partir do ponto onde é calculada a assinatura do bloco.

3.1.4 Arquivo de meta-dados

Nesta fase valida-se a existência ou não de assinaturas enviadas pelo processamento, assim como se arquivam as novas assinaturas após a comunicação de dados para o servidor. (He et al., 2010; Kulkarni et al., 2004; Liu et al., 2008; Mandagere et al., 2008; Meister & Brinkmann, 2009)

3.1.5 Comunicação

Na comunicação são recebidos os dados encriptados e a respetiva assinatura. Em primeiro lugar a assinatura é comunicada ao servidor, que valida a sua existência e comunica o resultado, caso a resposta do servidor seja negativa, quanto à existência da assinatura, são transferidos para o servidor os dados encriptados.

3.1.6 Recolha de dados

Não fazendo parte do processamento de deduplicação segura terá também a aplicação de efetuar a recolha de dados, para posterior análise, que deverá ser efetuada sobre a evolução do consumo da bateria durante o decorrer de todo o teste, sobre os tempos gastos parcelarmente nas várias fases do processamento e sobre o volume de dados transmitidos incluído a sobrecarga do protocolo. Os dados assim obtidos deverão ser enviados para o servidor no final de cada processamento.

3.2 Aplicação para servidor

A aplicação para servidor deve poder ser executada em qualquer sistema com ligação de rede que possa ser acedido por rede local via Wi-Fi, mas também estar disponível via Internet para poder ser conectado a partir de dispositivos móveis por via de redes móveis.

Deve também o sistema servidor ter capacidade de armazenamento para albergar os dados recebidos dos clientes móveis durante a execução do processo de deduplicação, para arquivar os respetivos meta-dados necessários ao processamento e ainda arquivar os dados relativos a cada processamento.

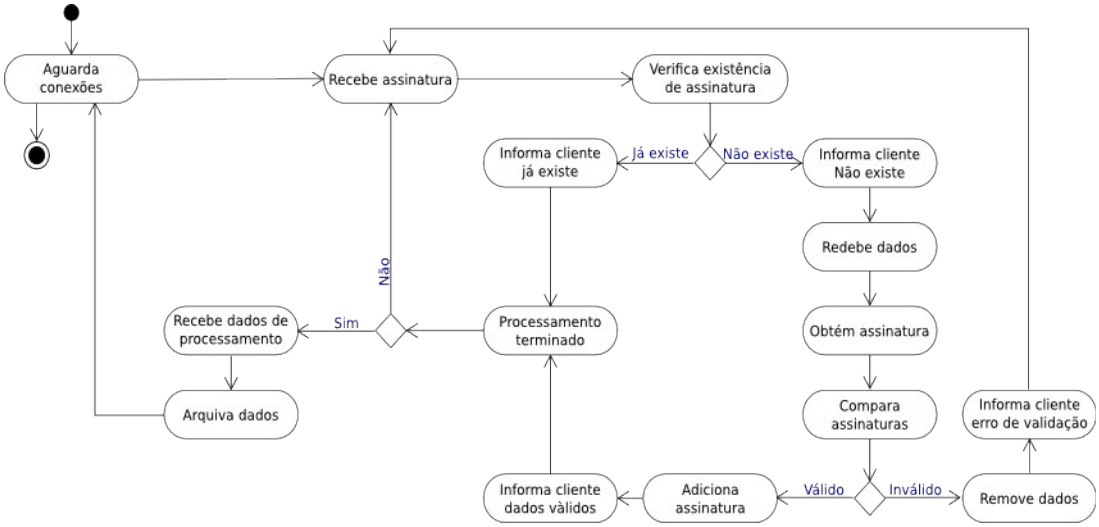


Figura 6: Diagrama de atividades utilizando UML Aplicação servidor

A aplicação servidor deve aguardar por ligações por parte dos clientes, quando estabelecida a conexão por parte do cliente, recebe a assinatura dos dados encriptados a serem enviados, verificada a existência da assinatura informa o cliente da necessidade de envio dos dados encriptados.

Quando há transferência de dados encriptados por parte do cliente é gerada assinatura sobre os dados recebidos. Após comparação com a assinatura recebida anteriormente informa o cliente da validade dos dados recebidos, (Anderson & Zhang, 2010) quando esta validação é positiva adiciona a assinatura ao arquivo de meta-dados, caso contrário descarta os dados recebidos.

3.3 Criptografia convergente e validação

No processo de encriptação de dados recorre-se à criptografia convergente, que consiste na utilização da chave gerada a partir dos dados a serem encriptados (Anderson & Zhang, 2010; Storer et al., 2008). Para concretizar esse principio, a aplicação gera uma assinatura de 160 bits , utilizando o algoritmo SHA-1 (Bhagwat, Eshghi, Long, & Lillibridge, 2009), utilizando os primeiros 128 bits como chave e os últimos 128 bits como vetor de inicialização. Para proceder à encriptação dos dados é utilizado o algoritmo AES no modo CTR/NoPadding, uma vez que este modo é considerado seguro e ainda tem a vantagem de não arredondar o volume de dados encriptado, não aumentando assim o volume de dados a transferir na fase de comunicação.

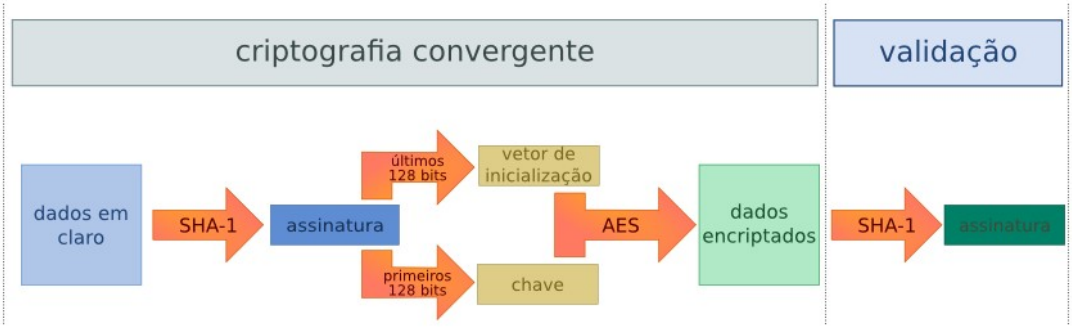


Figura 7: Criptografia convergente e validação

A assinatura dos dados encriptados que vai servir para validação no servidor, deve igualmente ser gerada utilizando o mesmo algoritmo que foi utilizado anteriormente. Esta validação serve para evitar ataques ao servidor, através do envio de assinaturas que não correspondam aos dados enviados,(Anderson & Zhang, 2010) deste modo o servidor pode gerar uma assinatura sobre os dados recebidos e comparar com a assinatura recebida.

O modelo apresentado enquadra o conceito de utilização de deduplicação segura nas suas várias fases, apresentando também o procedimento de recolha de dados necessários para efetuar a avaliação dos processamentos.

4 Desenvolvimento de protótipo

Com desenvolvimento do protótipo pretende-se validar o modelo apresentado no capitulo anterior, apresentando as soluções utilizadas para concretizar o procedimento de deduplicação segura e também para a recolha de dados dos processamentos.

4.1 Desenvolvimento das aplicações

O desenvolvimento das aplicações será feito em Java tanto para o cliente como para o servidor.

Para o processamento criptográfico e de assinaturas será utilizada a biblioteca Bouncy Castle, já que esta não apresenta qualquer restrição de utilização, ao contrário da biblioteca criptográfica incluída no Java.

4.1.1 Interface de utilizador

A aplicação permite parametrizar o endereço do servidor para onde vai ser realizada a transmissão de dados, assim como selecionar qual o método de deduplicação a utilizar.

É também mostrado na interface do utilizador parâmetros da rede que está a ser utilizada e informação sobre a identificação do dispositivo e versão de Android utilizada.

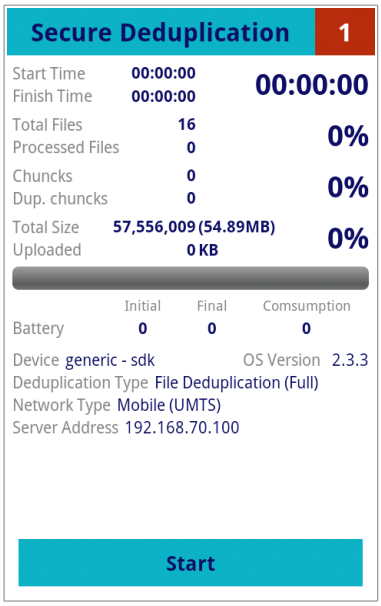


Figura 8: Interface do utilizador da aplicação cliente

Durante a execução é mostrado na interface a evolução do processamento em termos de dados processados, tempo decorrido e consumo de bateria.

Deve ainda esta aplicação após cada processamento receber e arquivar os dados relativos às condições de execução, consumo energético, tempos de execução, volume de dados processados e volume de dados transferidos durante o processamento.

4.1.2 Meta-dados

Para arquivo dos meta-dados nos dispositivos moveis, será utilizado o sistema de base de dados *sqlite* que é suportado nativamente pelo sistema Android. No servidor será utilizado o mesmo sistema de base de dados.

4.1.3 Recolha de dados

Para a recolha de informação quanto ao consumo de bateria durante o processamento, deverá ser utilizada a informação disponibilizada pelo "*BatteryManager*" do sistema Android, e através de um "*BroadcastReceiver*" desencadear um procedimento de cada vez que existam alterações na carga da bateria. Deste modo é possível registar o momento em que ocorreram as variações de carga de bateria durante o processamento.

```
private BroadcastReceiver batInfoReceiver = new BroadcastReceiver(){
	@Override
	public void onReceive(Context context, Intent i) {
		int level = i.getIntExtra(BatteryManager.EXTRA_LEVEL, -1);
		int scale = i.getIntExtra(BatteryManager.EXTRA_SCALE, -1);
		int batStatus = i.getIntExtra(BatteryManager.EXTRA_STATUS, -1);
		batCharging = (batStatus == BatteryManager.BATTERY_STATUS_CHARGING);
		if (batCharging) {
			tvCharging.setVisibility(View.VISIBLE);
		} else {
			tvCharging.setVisibility(View.INVISIBLE);
		}
		int batteryPct = (Math.round((level  / (float)scale)*100));
	if (batteryPct != batLevel) {
		batLevel = batteryPct;
		if  (processing){
			tvBatEnd.setText(String.valueOf(batteryPct));
			long runSecs = Math.round((float)(SystemClock.elapsedRealtime() -
chrono.getBase()) / (float) 1000);
			results.addBatStatus(batLevel, runSecs);
			int batIni = Integer.parseInt(tvBatStart.getText().toString());
			tvBatConsumption.setText(String.valueOf(batIni-batLevel));
		}
		}
	}
};
```

O sistema Android disponibiliza também um valor em milissegundos desde o último arranque do sistema podendo-se obter valores precisos para o registo de tempo de execução das várias fases do processamento. Para que esta recolha de informação seja o menos intrusiva possível no processamento, pode ser feita a leitura do valor disponibilizado pelo sistema Android no início e no fim de cada fase do processamento que se pretende avaliar, sendo o tempo consumido o diferencial dos dois valores.

4.1.4 Fases do processamento

Para a obtenção de dados de tempos necessários para a execução das diversas fases de processamento são consideradas as seguintes fases:

Assinatura (claro) – geração da assinatura sobre o ficheiro ou bloco de dados em processamento, assim como a computação do *rolling hash* utilizado na deduplicação por blocos de tamanho variável.

Encriptação – procedimento de obtenção do ficheiro ou bloco encriptado

Assinatura (encriptado) – obtenção de assinatura efetuada sobre o ficheiro ou bloco de dados encriptado.

Base de dados – operações de inserção ou busca de assinaturas na base de dados.

Transmissão – procedimento de comunicação efetiva de informação para o servidor, não incluindo a criação ou análise de pacotes enviados e recebidos.

Outros – diferença entre somatório de todas as outras fases de processamento e o valor do tempo total necessário para a execução do respetivo processamento.

4.1.5 Processamento por blocos

No processamento da deduplicação recorrendo a blocos de tamanho fixo, o valor a utilizar para o comprimento de cada bloco será de 4096 Bytes (4KB). No método de blocos de tamanho variável será utilizado o mesmo valor para o tamanho base do bloco e será utilizado um tamanho de janela flutuante de 32 Bytes.

Na deduplicação por blocos de tamanho variável a função para gerar assinaturas “fracas” será desenvolvida utilizando um processo baseado no sistema *rolling hash* de Rabin Karp, que pode ser apresentado com a seguinte fórmula:

$$A(b) = b[0]a^{n-1} + b[1]a^{n-2} + \ldots + b[n-1]$$

Este sistema permite calcular assinaturas de uma forma muito rápida e leve em termos de processamento, já que não é necessário recalcular a assinatura para cada posição dentro da janela flutuante, bastando adicionar o novo elemento e subtrair o primeiro valor do bloco.

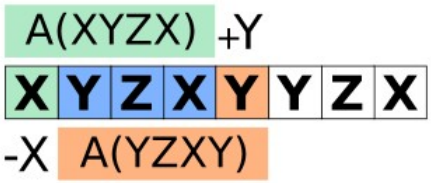


Figura 9: Rolling Hash

O processamento do *rolling hash* é efectuado por uma classe que é inicializada com os valores do tamanho de bloco e tamanho da janela e disponibiliza dois métodos públicos que devolvem a assinatura do bloco. O método *getBlockHash* devolve a assinatura do bloco inicial enquanto o método *updateHash* avança a janela e devolve a nova assinatura.

```
public class RollingHash {
        final static int RP = 370248451;
        final static int R = 256;
        private int blkSize;
        private long Z;
        private long h;
        private byte[] blk;
        private int blkPoiter;

        public RollingHash (int blockSize, int slideSize){
                blkSize = blockSize;
                blk = new byte[blockSize+slideSize];
                Z = 1;
                for (int i = 1; i <= blkSize -1; i++){
                        Z = (Z * R) % RP;
                }
        }

        public long getBlockHash (byte[] block){
                h = 0;
                for (int i = 0; i < blkSize; i++){
                        h = (h * R + ((int)block[i]&0xFF)) % RP;
                }
                System.arraycopy(block, 0, blk, 0, blk.length);
                blkPoiter=0;
                return h;
        }

        public long updateHash (){
                h = (h + RP - Z *  ((int)blk[blkPoiter]&0xFF) % RP) % RP;
                h = (h * R + ((int)blk[blkSize+blkPoiter]&0xFF)) % RP;
                blkPoiter++;
                return h;
        }

}
```

Na transmissão de dados quando utilizada a deduplicação por ficheiros cada pacote deve também conter 4KB de informação respeitante ao ficheiro. Deste modo as condições utilizadas na transferência de dados serão o mais equiparadas possível entre os vários métodos de deduplicação.

4.1.6 Comunicação de resultados

Após a realização de cada teste, a aplicação envia para o servidor os dados recolhidos durante o processamento.

Para armazenamento dos dados obtidos durante o processamento é utilizada uma classe constituída apenas por atributos e os respetivos *getters* e *setters*, durante o processamento são evocados os vários *setters* para armazenar os dados.

O método que envia os dados para o servidor recorre a essa classe para obter os valores armazenados, evocando os vários *getters*, e procede ao seu envio.

```
public void sendResults(SecDedupResults results, long[] timers) throws IOException {
        byte bBufer = 0;
        dos.writeByte(SDProto.RESULTS);
        dos.writeInt(deviceID);
        dos.writeLong(results.getTotalTime());
        dos.writeLong(results.getTotalSize());
        dos.writeInt(results.getProcFiles());
        dos.writeInt(results.getProcChuncks());
        dos.writeInt(results.getDupChuncks());
        dos.writeLong(results.getTotalUpload());
        dos.writeInt(results.getDedupLevel());
        dos.writeInt(results.getNetworkType());
        dos.writeByte(timers.length);
        for (int i = 0; i < timers.length; i++){
                dos.writeLong(timers[i]);
        }
        int b = results.getBatSatusCount();
        dos.writeByte(b);
        for (int i = 0; i < b; i++){
                BatteryStatus bs = results.getBatStatus(i);
                dos.writeInt(bs.getBatteryLevel());
                dos.writeLong(bs.getElapsedTime());
        }
        dos.flush();
        bBufer = dis.readByte();
        if (bBufer == SDProto.RESULTS_OK){
                dos.writeByte(SDProto.PROC_END);
                dos.flush();
                out.close();
                in.close();
                s.close();
        }
}
```

4.2 Síntese

O protótipo é composto por duas aplicações, uma a ser executada nos dispositivos a testar e a outra a ser executada no servidor. Para ambas as aplicações é utilizado o Java como linguagem de desenvolvimento, o *sqlite* como sistema de armazenamento de dados, e é utilizada a biblioteca *Bouncy Castle*, nos procedimentos de geração de assinaturas e criptografia.

A aplicação cliente para além de executar o procedimento de deduplicação segura, recolhe dados do consumo de bateria durante a execução dos processamentos e os tempos de execução para as várias fases do processo de deduplicação. Estes dados são transmitidos para o servidor no final de cada teste e aí são arquivados.

Esta aplicação tem capacidade de executar os métodos de deduplicação por ficheiros, por blocos de tamanho fixo e variável. Para realizar o último método é utilizado um sistema de *rolling hash*.

5 Avaliação preliminar

O contexto em que decorreram os testes em termos de dispositivos utilizados, ambiente de teste e a caracterização dos conjuntos de dados utilizados, são de seguida apresentados. Também são apresentados e discutidos os resultados obtidos nos diferentes processamentos

5.1 Dispositivos

Para a execução dos vários processamentos deverão ser utilizados diversos dispositivos móveis com sistema Android de modo a testar os processamentos em variadas condições relativas à capacidade de processamento.

Para que o consumo de energia possa ser avaliado o mais corretamente possível, antes de ser iniciado qualquer processamento, cada um dos dispositivos dever ser ligado à corrente até o dispositivo assinalar que a bateria está completamente carregada.

Devem também ser desativadas, nos dispositivos a testar, todas as aplicações e serviços que for possível, para que os dados obtidos não sejam influenciados no tempo de execução, consumo energético e utilização de largura de banda.

5.1.1 Dispositivos móveis utilizados

Para realização dos testes foram utilizados vários equipamentos cujas características permitiram conseguir realizar todos os testes, apresentando dados coerentes.

ID	Descrição	Processador	Bateria	Versão do Android
U8510	Huawei U8510	Qualcomm MSM 7227 600 MHz	1200 mAh	2.3.3
A983	Alcatel one touch 983	Broadcom 21552G 1.00 GHz	1300 mAh	2.3.7
I9100	Samsung Galaxy SII I9100	Exynos Cortex A9 dual-core 1.2 GHz	1650 mAh	2.3.3

Tabela 1: Equipamentos utilizados nos testes

5.1.2 Dispositivos móveis excluídos

Foi tentado utilizar um maior número de equipamentos de modo a que os dados obtidos tivessem uma maior aderência à realidade, contudo foram encontrados problemas nos

dispositivos que invalidaram a sua utilização, tendo sido dois os fatores que anularam a sua utilização.

- Falta de espaço de armazenamento interno
- Inconsistência nos dados de consumo da bateria.

Em alguns dos equipamentos o espaço de armazenamento interno é bastante pequeno, sendo agravado o problema por colocação de software não removível por parte dos fabricantes e também por parte das operadoras. Como o sistema Android, por defeito, armazena as bases de dados na memória interna, não foi possível realizar os testes que utilizam o processamento de deduplicação por blocos de tamanho fixos e variáveis, devido ao tamanho atingido pela base de dados dos meta-dados.

A obtenção do estado de carga da bateria por parte do sistema Android não é muito rigorosa pelo que os dados obtidos em alguns equipamentos apresentavam incongruências, como quedas abruptas ou subidas na carga da bateria, mesmo realizando vários testes não foi possível obter um conjunto de dados coerentes.

5.2 Servidor

A aplicação servidor foi executada em todos os processamentos num computador com o sistema operativo Fedora Linux *release* 17 (*Beefy Miracle*) a correr com o *kernel* 3.9.10-100.fc17.x86_64, com o Java instalado no sistema OpenJDK 64-Bit Server, version 1.7.0_25

O hardware utilizado no servidor é composto por um processador AMD Phenom II X6 1090T, 16GB de memória RAM, o armazenamento foi efetuado num disco SSD Intel 520.

5.3 Rede

O equipamento para execução da aplicação de servidor está ligado numa rede local estruturada a 1Gb, que por sua vez está conectada à Internet através de uma ligação com 30Mb/1Mb de Download/Upload de largura de banda, para utilização nos testes a efetuar em que os dispositivos móveis utilizam redes móveis.

Para os testes a realizar sobre Wi-Fi, é utilizado um equipamento TP-Link TL-WR741N dedicado, ligado à rede estruturada, devendo ser realizados os testes sequencialmente de modo a que a largura de banda disponível pudesse ser utilizada sem qualquer restrição e sem ser partilhada.

Em todos os testes os dispositivos deverão ser colocados na mesma localização de modo que as condições de sinal das redes sem fios sejam idênticas. No caso dos processamentos sobre redes móveis também deverá ser sempre utilizado o mesmo cartão SIM, de operador, de modo que também se verifiquem condições idênticas, na realização dos testes.

A quando da realização dos testes na rede não pode existir qualquer outro tráfego a utilizar a largura de banda, o mesmo princípio deve ser utilizado em relação à utilização da Internet a quando da realização de testes em que os dispositivos móveis utilizam redes móveis, evitando deste modo a adulteração em termos de utilização da largura de banda e consequentemente dos tempos de execução, nos dados obtidos.

5.4 Conjuntos de dados

Para efetuar os testes foram criados dois conjuntos de dados com ficheiros retirados de vários dispositivos, como normalmente o tipo de ficheiros mais vulgarmente encontrados em dispositivos móveis são ficheiros multimédia, foi dado grande peso nas amostras para processamento a estes tipos de ficheiros.

Os ficheiros utilizados foram classificados em cinco categorias :

Áudio - ficheiros do tipo MP3 e Flac.

Documentos - ficheiros do tipo PDF

Email – Arquivo de emails recebidos.

Fotos - ficheiros do tipo jpg, tendo sido selecionadas fotografias de várias dimensões e resoluções, produzidas nos diversos dispositivos utilizados.

Vídeo – Ficheiros do tipo mp4, curtas sequencias de vídeo criadas nos dispositivos utilizados nos testes.

Os conjuntos de dados devem ser arquivados no mesmo cartão microSD, a ser utilizado em todos os dispositivos, para que a velocidade de leitura dos dados durante o processamento seja idêntica em todos os equipamentos testados.

5.4.1 Conjunto de dados 1

Este grupo de ficheiros contém exclusivamente ficheiros multimédia, composto por 52 ficheiros com aproximadamente 200MB, sobre este conjunto de ficheiros será realizado um processamento completo nos diversos modos de deduplicação.

Tipo de ficheiro	**Ficheiros**	**Tamanho** (bytes)
Áudio	28	162.219.701
Fotos	23	22.400.875
Vídeo	1	25.237.299
Total	**52**	**209.857.875** (~ 200MB)

Tabela 2: Ficheiros de conjunto de dados 1

5.4.2 Conjunto de dados 2

No segundo conjunto de ficheiros para processamento será utilizado um maior volume de dados, mas serão considerados dois momentos de execução de modo a simular um processamento diferencial.

Tipo de ficheiro	**Momento 0**		**Momento 1**		**Diferença**	
	Ficheiros	**Tamanho** (bytes)	**Ficheiros**	**Tamanho** (bytes)	**Ficheiros**	**Tamanho** (bytes)
Áudio	44	220.207.664	48	230.163.342	4	9.955.678
Documentos	6	722.538	6	722.538	0	0
Fotos	58	139.750.047	66	155.605.250	8	15.855.203
Email	1	14.286.949	1	14.598.628	0	311.679
Vídeo	2	70.796.685	2	70.796.685	0	0
Total	**111**	**445.763.883** (~ 425MB)	**123**	**471.886.443** (~ 450MB)	**12**	**26.122.560** (~ 25MB)

Tabela 3: Ficheiros de conjunto de dados 2

A diferença do conjunto de dados entre os dois momentos de execução deve-se a adição de 12 novos ficheiros e também a alterações efetuadas no arquivo de Email, de onde foram eliminadas várias mensagens e arquivadas novas mensagens. Isto faz com que o ficheiro tenha um conteúdo alterado, apresentando uma diferença de tamanho de aproximadamente 304KB.

O processamento relativo ao momento 0 será realizado previamente, de modo a obter os meta-dados relativos a um processamento completo sobre os 111 ficheiros. Para cada um dos métodos de deduplicação, o volume total de informação processada é de aproximadamente 425MB.

Para a execução dos testes no momento 1 deverá ser inserido em cada dispositivo a base de dados com os meta-dados produzidos anteriormente e serem efetuados os vários processamentos sobre o conjunto de 123 ficheiros com aproximadamente 450MB.

5.5 Apresentação dos dados obtidos em cada processamento.

Para cada dispositivo utilizado nos testes obteve-se uma tabela com os tempos consumidos em cada uma das fases de processamento em termos absolutos de tempo e em termos relativos, considerando o tempo total de execução de cada teste. Para poderem ser relacionados de um modo mais fácil, os valores dos tempos de execução nas fases de processamento são também apresentados em forma de gráfico.

Em cada processamento foram ainda obtidos os valores de alteração da carga da bateria ao longo tempo de execução do teste, estes valores são apresentados em forma de gráfico.

Durante a execução da aplicação foi inibida a capacidade de desligar o ecrã do dispositivo. Embora isto implique o aumento do consumo de energia, permite que as condições sejam idênticas em todos os dispositivos, já que o tempo em que o ecrã se encontra ligado varia entre os dispositivos utilizados nos teste, ou mesmo durante a execução de um processamento, quando o nível da bateria desce abaixo de um determinado valor.

Nos pontos seguintes são apresentados os dados obtidos no processamento dos vários métodos de deduplicação, com os dispositivos a utilizarem rede Wi-Fi.

5.6 Processamento do conjunto de dados 1

Após o processamento do conjunto de dados 1 foi possível obter os valores gerais sobre cada método de deduplicação.

Tipo de deduplicação	Ficheiro	BTF	BTV
Blocos processados	52	51.259	51.259
Blocos repetidos	0	19	2.161
Volume enviado (bytes)	210.321.338	211.727.171	202.872.143
Tamanho meta-dados (bytes)	13.312	6.232.064	6.338.560

Tabela 4: Valores obtidos no processamento do conjunto de dados 1

O número de blocos processados no caso do processamento de deduplicação ao nível de ficheiros é igual ao número de ficheiros processados, já que cada ficheiro é considerado um bloco. Para o processamento por blocos de tamanho fixo como de tamanho variável os ficheiros foram partidos em 51.259 blocos.

Como não existiam no conjunto de dados ficheiros iguais, o processamento ao nível de ficheiros não consegue identificar dados repetidos, como apontado por estudos anteriores a deduplicação por blocos consegue detetar redundância de dados mesmo quando não são analisados ficheiros iguais. Os dados obtidos confirmam também estudos anteriores no que diz respeito a eficiência na deteção de dados duplicados quando se utiliza processos de deduplicação por blocos de tamanho variável.

Como demonstram os resultados obtidos foram detetados 19 blocos repetidos quando se efetuou o processamento por blocos de tamanho fixo, subindo o valor para 2161 repetições quando da utilização de blocos de tamanho variável.

O arquivo das assinaturas de cada um dos blocos implica que o tamanho da base de dados aumente em consequência do número de assinaturas guardadas, como é demonstrado pelos resultados. Isto causa um tamanho de base de dados muito superior nos processamentos por blocos de tamanho fixo e variável, situação que é crítica devido ao reduzido espaço para armazenamento interno disponível nos dispositivos móveis, impossibilitando mesmo em alguns casos a sua utilização.

Apesar do número de blocos analisados ser igual nos dois processamentos por blocos, o número de duplicações encontrada é superior no método de blocos de tamanho variável pelo que o número de assinaturas a guardar é menor. No entanto este método implica o arquivo das assinaturas geradas pelo sistema de *rolling hash*, para além das assinaturas guardadas nos outros casos, o que justifica o diferencial de tamanho da base de dados gerada nos dois processamentos.

Tipo de deduplicação	Volume de dados	Diferencial	%
Ficheiros	210.321.338	+463.463	+0,22%
BTF	211.727.171	+1.869.296	+0,89%
BTV	202.218.971	-6.985.732	-3,33%
Dados originais	209,857.875		

Tabela 5: Volume de dados transmitido

Neste conjunto de dados pretendia-se testar a execução de uma cópia de segurança completa, pelo que a relação entre o volume de dados transferido e o volume de dados processado revela a eficiência do sistema de duplicação, mas também o peso do protocolo de transmissão.

No processamento em que se utiliza o método de deduplicação por ficheiro, como já foi analisado, não foi detetada qualquer redundância, pelo que o diferencial entre os dados enviados e o volume de dados processados se deve exclusivamente à informação suplementar enviada pelo protocolo, relativa à identificação de cada pacote transferido. Como se optou por um tamanho de carga efetiva por pacote idêntico ao tamanho do bloco utilizado nos outros métodos de deduplicação, o número de pacotes enviados foi aproximado. O peso relativo do protocolo representa apenas 0,22% dos dados transferidos, neste processamento.

Quando foi utilizado o processamento por blocos de tamanho fixo o peso do protocolo foi ligeiramente superior à situação anterior devido à necessidade de envio de uma assinatura por cada bloco transferido, ainda assim o acréscimo é de apenas 0,89%.

No caso do processamento utilizando blocos de tamanho variável, apesar de também ser necessário o envio de uma assinatura por cada bloco, dada a redução de blocos a enviar

devido à deteção de redundâncias, o volume relativo de dados realmente enviado foi 3.33% inferior ao montante total de dados processados.

5.6.1 Resultados do equipamento Huawei U8510

Fase	Ficheiro	%	BTF	%	BTV	%
Assinatura (claro)	00:03:03	17,41%	00:06:44	5,14%	00:15:48	6,58%
Encriptação	00:05:40	32,26%	00:14:45	11,26%	00:15:17	6,37%
Assinatura (encriptado)	00:03:06	17,63%	00:07:47	5,94%	00:06:50	2,85%
Base de dados	00:00:02	0,22%	00:46:54	35,79%	02:08:14	53,40%
Transmissão	00:05:42	32,40%	00:49:14	37,58%	00:47:56	19,96%
Outros	00:00:00	0,07%	00:05:37	4,29%	00:26:02	10,85%
Total [1]	**00:17:36**	**100,00%**	**02:11:02**	**100,00%**	**04:00:08**	**100,00%**

Tabela 6: Tempos de execução - Huawei U8510 - Conjunto de dados 1

A Tabela 6 e o Gráfico 1 apresentam os dados obtidos sobre os tempos de execução absolutos e relativos para os diversos métodos de deduplicação, quando utilizado o dispositivo Huawei U8510.

Sendo este dispositivo o que tinha o processador mais fraco de todos os equipamentos utilizados nos testes, é normal que os valores obtidos para as fases de assinatura e encriptação sejam mais elevados que noutros dispositivos, chegando a ocupar 32,26% do tempo total de processamento na fase de encriptação quando utilizado o método de deduplicação por ficheiros.

Nos processamentos que recorrem à utilização de blocos de tamanho fixo e variável as fases de base de dados e de transmissão ocupam a maioria do tempo de processamento utilizado. Quando utilizado o método de deduplicação por ficheiros a transmissão também representa uma parcela considerável do tempo de processamento, no entanto a fase de base de dados tem um impacto muito pequeno no tempo de processamento, o que é justificado pelo pequeno número de interações necessárias com a base de dados. Apenas é feita uma inserção de

1 Os valores apresentados nas linhas foram arredondados, pelo que a soma das linhas pode apresentar diferenças para o total

assinatura e uma busca para verificar se já existe uma assinatura idêntica por cada ficheiro processado.

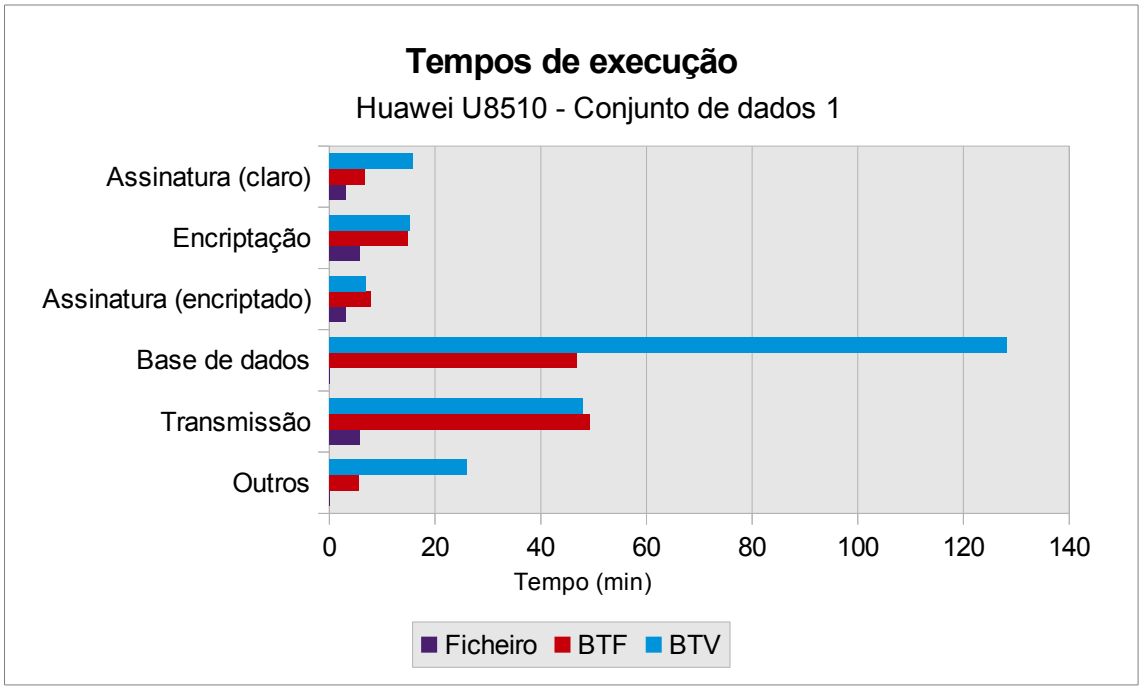


Gráfico 1: Tempos de execução - Huawei U8510 - Conjunto de dados 1

Tomando como base o tempo de processamento utilizando o método de duplicação por ficheiros, na Tabela 7 relacionam-se os tempos totais de execução para os vários tipos de processamento.

	Ficheiro	**%**	**BTF**	**%**	**BTV**	**%**
Tempo total de execução	00:17:36	100%	02:11:02	744%	04:00:08	1364%

Tabela 7: Comparação de tempos de execução – Huawei U8510 - Conjunto de dados 1

A execução do teste utilizando o processamento por ficheiro apresenta um valor absoluto bastante inferior a todos os outros valores de tempo total de execução, que é de 17 minutos e 36 segundos. O valor para o processamento por blocos de tamanho fixo é aproximadamente 7,5 vezes superior, enquanto para o processamento por blocos de tamanho variável temos uma relação de aproximadamente 13,5 vezes.

	Ficheiro	BTF	BTV
Taxa de processamento (KBs)	194,03	26,07	14,22

Tabela 8: Taxa de processamento – Huawei U8510 - Conjunto de dados 1

Na Tabela 8 é apresentado a taxa de processamento para o dispositivo Huawei U8510, que consiste no rácio entre o volume de dados processados e o tempo total de execução para cada método de duplicação testado.

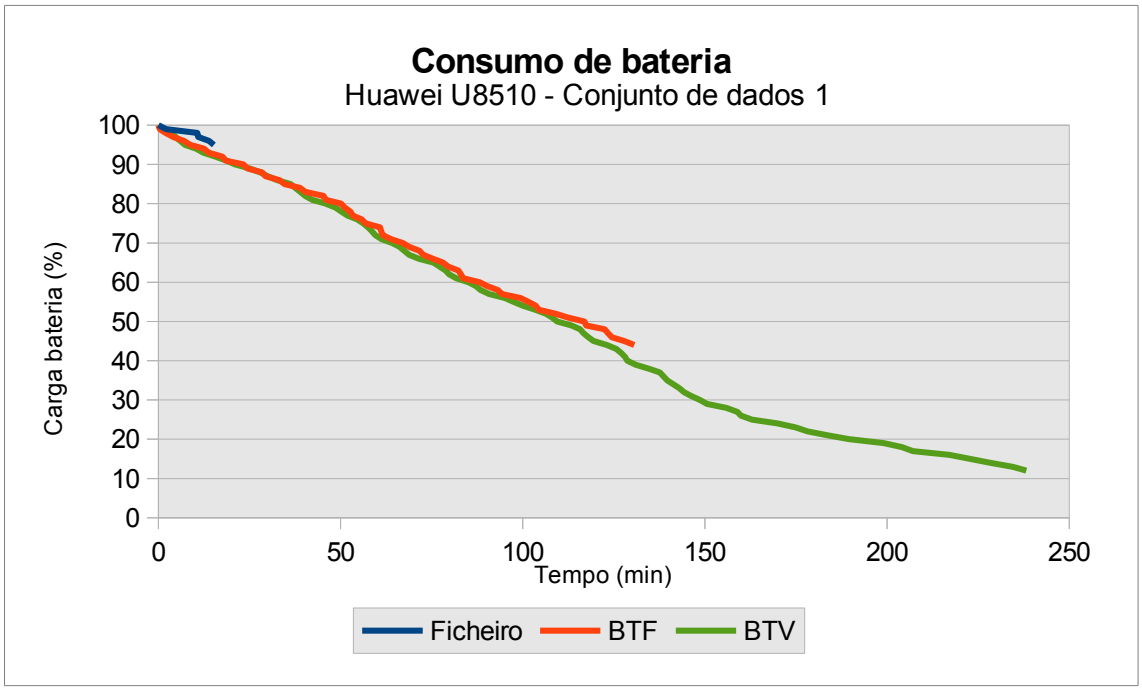


Gráfico 2: Consumo de bateria – Huawei U8510 - Conjunto de dados 1

O consumo de bateria neste dispositivo está próximo de constante durante a execução dos testes, apresentando no entanto mais irregularidades que noutros dispositivos.

É claramente visível no Gráfico 2 um atenuar do declive da linha representativa do processamento por blocos de tamanho variável, na fase final da execução. Este efeito tem como causa fato do dispositivo durante o teste ter atingido um baixo nível de bateria e por isso ter entrado em modo de economia de energia.

O consumo dos processamentos por blocos de tamanho fixo e variável são bastante idênticos sendo por isso representados no gráfico por linhas sobrepostas ou muito próximas.

5.6.2 Resultados do equipamento Alcatel A983

Fase	Ficheiro	%	BTF	%	BTV	%
Assinatura (claro)	00:01:34	14,14%	00:03:46	4,15%	00:10:18	5,31%
Encriptação	00:02:36	23,52%	00:06:22	7,01%	00:05:56	3,06%
Assinatura (encriptado)	00:01:37	14,60%	00:04:04	4,48%	00:03:32	1,82%
Base de dados	00:00:01	0,16%	00:22:43	25,00%	01:44:11	53,63%
Transmissão	00:05:16	47,50%	00:48:58	53,91%	00:47:27	24,43%
Outros	00:00:01	0,08%	00:04:56	5,44%	00:22:48	11,74%
Total	**00:11:06**	**100,00%**	**01:30:51**	**100,00%**	**03:14:15**	**100,00%**

Tabela 9: Tempos de execução - Alcatel A983 - Conjunto de dados 1

Na Tabela 9 e no Gráfico 3 estão representados os tempos de execução obtidos para a várias fases de processamento utilizando os diversos métodos de deduplicação, recolhidos após a execução dos respetivos processamentos no equipamento Alcatel A983.

Pode verificar-se que as fases onde existe uma utilização do processador mais intensa, geração de assinaturas e encriptação, representam em todos os testes um tempo de processamento muito inferior às fases de transmissão. Com exceção do processamento por ficheiros onde o número de inserções e pesquisas na base de dados é bastante mais baixo, o tempo consumido na fase de base de dados representa também um peso relativo, elevado em relação ao tempo total de processamento.

Apesar do número de blocos processados ser idêntico nos dois processamentos por blocos, o tempo consumido para gerar assinaturas sobre os blocos em claro é superior no processamento por blocos de tamanho variável, devido à necessidade suplementar do sistema de *rolling hash.*

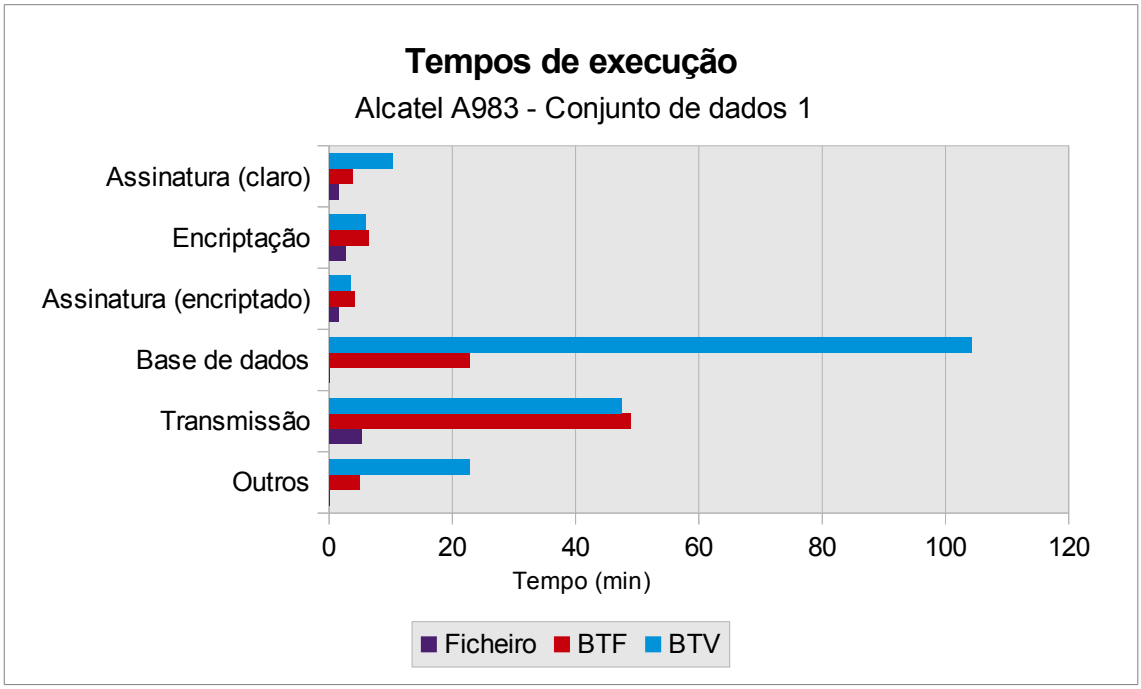


Gráfico 3: Tempos de execução - Alcatel A983 - Conjunto de dados 1

O tempo total de execução dos vários tipos de processamento apresenta grandes diferenças consoante o método de deduplicação utilizado, na Tabela 10 relacionam-se os valores do tempo total de execução aplicando-se o tempo de execução do processamento por ficheiros, que é o mais curto, como base de comparação.

	Ficheiro	**%**	**BTF**	**%**	**BTV**	**%**
Tempo total de execução	00:11:06	100%	01:30:51	817%	03:14:15	1748%

Tabela 10: Comparação de tempos de execução - Alcatel A983 - Conjunto de dados 1

A execução do teste utilizando o processamento por ficheiro apresenta um valor absoluto de 11 minutos e 6 segundos que é bastante inferior a todos os outros valores de tempo total de execução, o valor para o processamento por blocos de tamanho fixo é mais de 8 vezes superior, enquanto para o processamento por blocos de tamanho variável temos uma relação de aproximadamente 17,5 vezes.

	Ficheiro	BTF	BTV
Taxa de processamento (KBs)	307,41	37,59	17,58

Tabela 11: Taxa de processamento - Alcatel A983 - Conjunto de dados 1

Considerado o volume de dados que é processado neste conjunto de dados e como foi efetuado um processamento completo sobre os dados, podemos obter uma taxa de processamento conseguida por este dispositivo, para cada um dos métodos de deduplicação, como é apresentado na Tabela 11.

O Gráfico 4 demonstra o consumo da bateria ao longo do processamento nos vários modos de deduplicação, para o dispositivo Alcatel A983, quando utilizado o conjunto de dados 1.

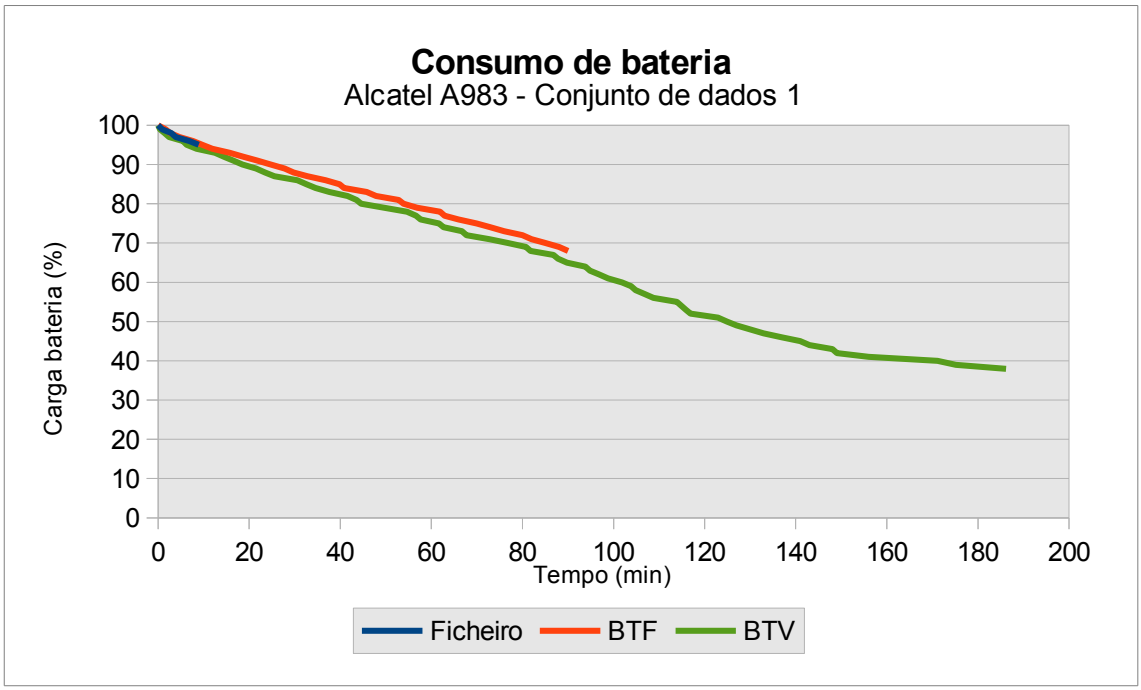


Gráfico 4: Consumo de bateria - Alcatel A983 - Conjunto de dados 1

Neste equipamento o consumo de bateria foi praticamente constante durante todo o processamento tendo uma evolução de consumo aproximada em todos os tipos de processamento.

5.6.3 Resultados do equipamento Samsung I9100

Fase	Ficheiro	%	BTF	%	BTV	%
Assinatura (claro)	00:00:46	9,61%	00:02:02	1,58%	00:04:38	2,93%
Encriptação	00:01:03	13,06%	00:04:46	3,71%	00:05:07	3,23%
Assinatura (encriptado)	00:00:40	8,36%	00:02:53	2,25%	00:02:04	1,30%
Base de dados	00:00:04	0,83%	01:05:52	51,13%	01:30:05	56,76%
Transmissão	00:05:29	68,10%	00:51:17	39,81%	00:48:12	30,38%
Outros	00:00:00	0,04%	00:01:57	1,52%	00:08:34	5,40%
Total	**00:08:04**	**100,00%**	**02:08:51**	**100,00%**	**02:38:43**	**100,00%**

Tabela 12: Tempos de execução - Samsung I9100 - Conjunto de dados 1

Os resultados obtidos nos processamentos efetuados no dispositivo Samsung I9100 nos vários métodos de deduplicação são apresentados na Tabela 12 e no Gráfico 5.

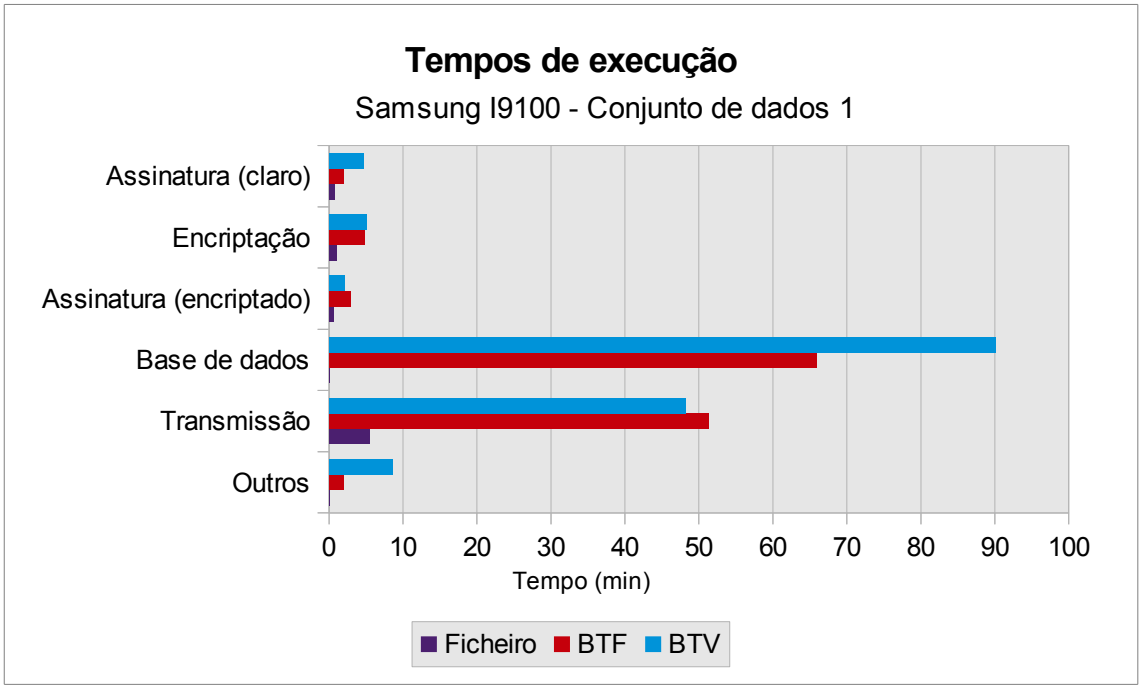


Gráfico 5: Tempos de execução - Samsung I9100 - Conjunto de dados 1

No processamento por ficheiros a maioria do tempo é consumida na fase de transmissão, que representa 68,1% do tempo total de processamento, por sua vez a fase de base de dados apenas ocupa 0,83% do tempo de processamento.

Também neste equipamento quando utilizados os métodos de deduplicação por blocos de tamanho fixo e variável a fase de base de dados tem um grande peso relativo no tempo de processamento representado 51,13% e 56,76% respetivamente.

Nas fases de utilização intensiva de processador, produção de assinaturas e encriptação, este dispositivo apresenta os melhores valores absolutos em comparação aos outros dispositivos testados, por ser o equipamento que possui o melhor processador.

Tomando como base o tempo de processamento utilizando o método de duplicação por ficheiros, na Tabela 13 relaciona-se os tempos totais de execução para os vários tipos de processamento.

	Ficheiro	**%**	**BTF**	**%**	**BTV**	**%**
Tempo total de execução	00:08:04	100%	02:08:51	1596%	02:38:43	1966%

Tabela 13: Comparação de tempos de execução – Samsung I9100 - Conjunto de dados 1

A execução do teste utilizando o processamento por ficheiro apresenta um valor absoluto bastante inferior a todos os outros valores de tempo total de execução, que é de 8 minutos e 4 segundos, os valor para o processamento por blocos de tamanho fixo e variável são mais próximos que nos outros dispositivos testados e são respetivamente 7,5 e X vezes superiores.

	Ficheiro	**BTF**	**BTV**
Taxa de processamento (KBs)	423,16	26,51	21,52

Tabela 14: Taxa de processamento – Samsung I9100 - Conjunto de dados 1

Na Tabela 14 é apresentado a taxa de processamento para o dispositivo Samsung I9100, que consiste no rácio entre o volume de dados processados e o tempo total de execução para cada método de duplicação testado.

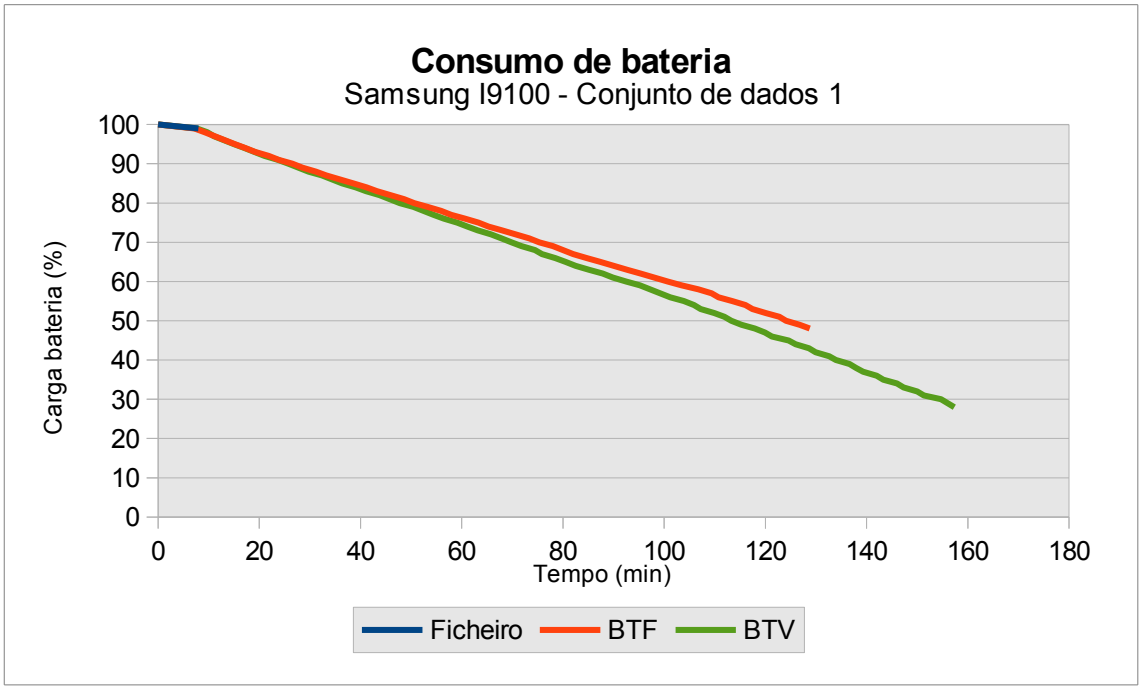


Gráfico 6: Consumo de bateria - Samsung I9100 - Conjunto de dados 1

Os dados de consumo de bateria do dispositivo Samsung I9100 apresentados no Gráfico 6 demonstram um consumo constante durante todo o processamento sendo representados no gráfico por uma linha quase reta nos vários processamentos, apenas variando no declive. Indica, o gráfico, que quando é utilizado o método de deduplicação por blocos de tamanho variável o consumo de bateria é superior ao consumo quando se utiliza o processamento por blocos de tamanho fixo.

Este facto pode ser justificado por este método necessitar de efetuar mais operações que requerem uma maior utilização do processador devido à utilização do sistema de rolling hash.

5.6.4 Comparação dos dados obtido nos vários dispositivos

Fase	Ficheiro			BTF			BTV		
	U8510	A983	I9100	U8510	A983	I9100	U8510	A983	I9100
Assinatura (claro)	00:03:03	00:01:34	00:00:46	00:06:44	00:03:46	00:02:02	00:15:48	00:10:18	00:04:38
Encriptação	00:05:40	00:02:36	00:01:03	00:14:45	00:06:22	00:04:46	00:15:17	00:05:56	00:05:07
Assinatura (encriptado)	00:03:06	00:01:37	00:00:40	00:07:47	00:04:04	00:02:53	00:06:50	00:03:32	00:02:04
Base de dados	00:00:02	00:00:01	00:00:04	00:46:54	00:22:43	01:05:52	02:08:14	01:44:11	01:30:05
Transmissão	00:05:42	00:05:16	00:05:29	00:49:14	00:48:58	00:51:17	00:47:56	00:47:27	00:48:12
Outros	00:00:00	00:00:00	00:00:00	00:05:37	00:04:56	00:01:57	00:26:02	00:22:48	00:08:34
Total	**00:17:36**	**00:11:06**	**00:08:04**	**02:11:02**	**01:30:51**	**02:08:51**	**04:00:08**	**03:14:15**	**02:38:43**

Tabela 15: Tempos de processamento - Conjunto de dados 1

A Tabela 15 exibe os valores obtidos no processamento do conjunto de dados 1 pelos dispositivos móveis utilizados, nas várias fases de processamento, assim como o tempo total de processamento, para os métodos de deduplicação testados.

Deste conjunto de resultados pode constatar-se que nas 3 primeiras fases, que são as que necessitam de maior capacidade de processamento, em todos os métodos de deduplicação testados os resultados apontam para que quanto melhor é o processador do dispositivo mais reduzidos são os tempos de processamento necessários para gerar as assinaturas e proceder à encriptação dos dados.

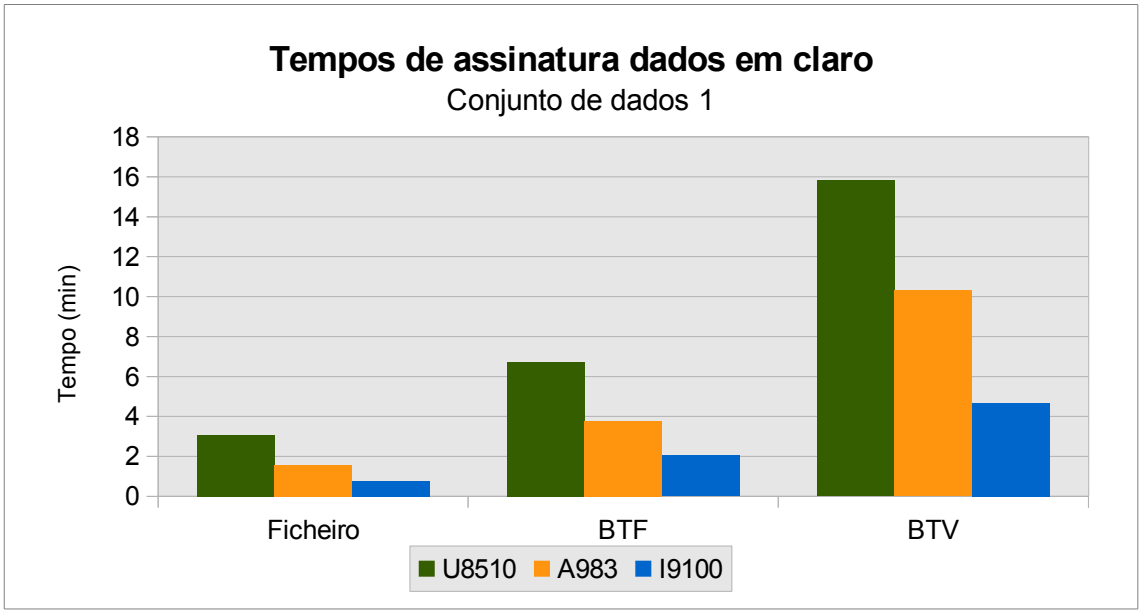


Gráfico 7: Tempos de assinatura de dados em claro - Conjunto de dados 1

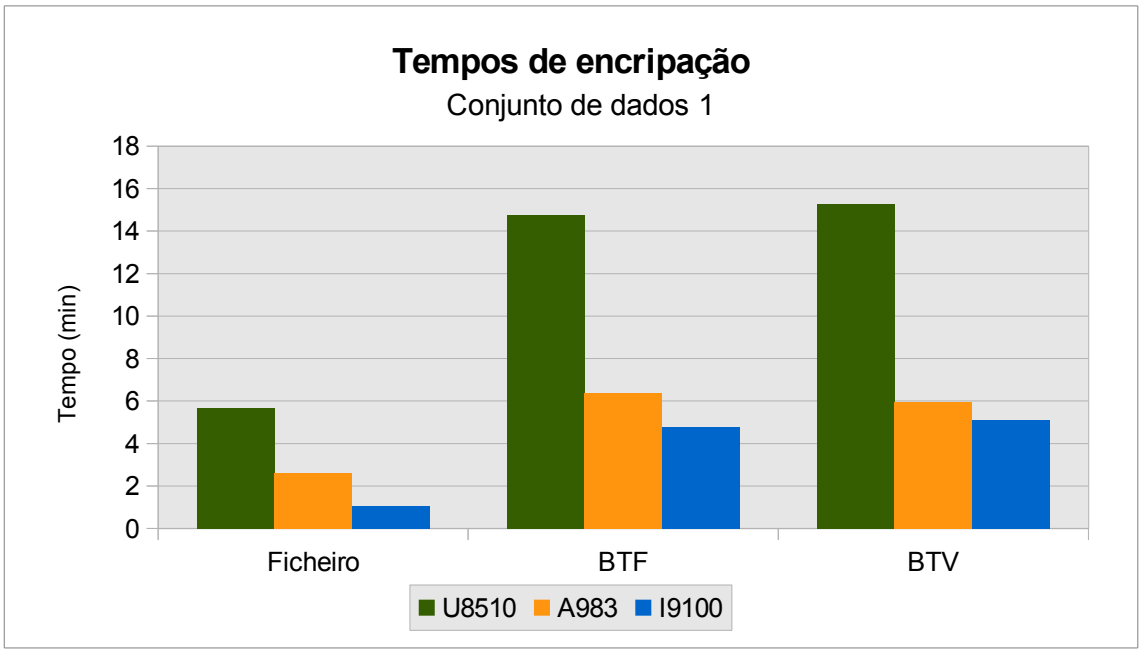


Gráfico 8: Tempos de encriptação - Conjunto de dados 1

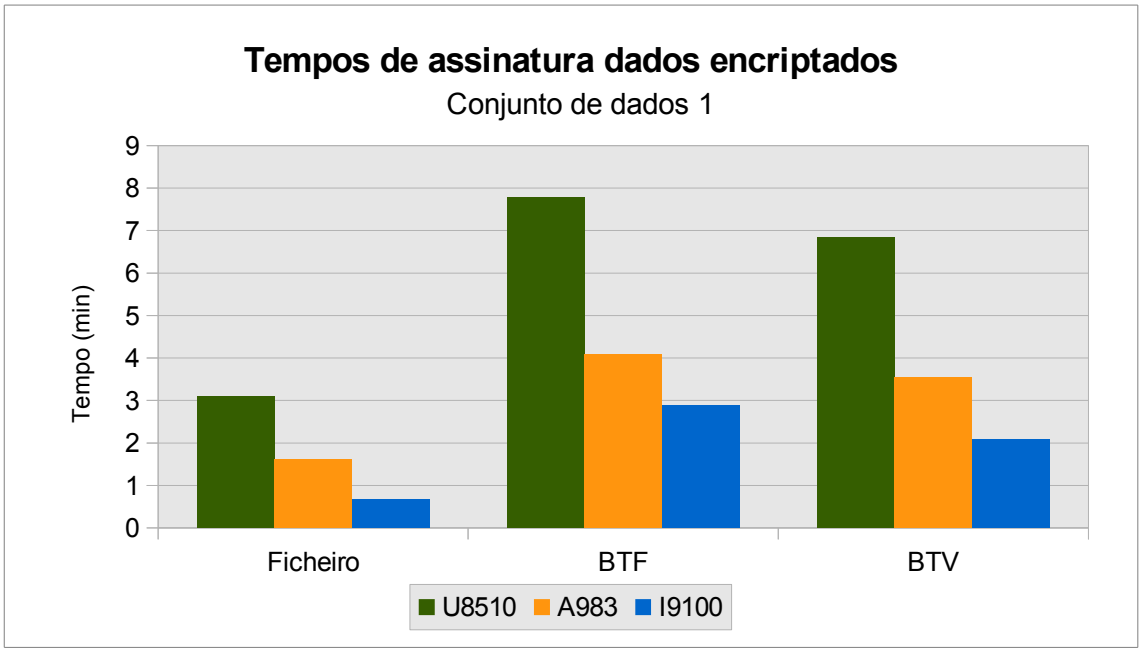


Gráfico 9: Tempos de assinatura dados encriptados - Conjunto de dados 1

Verifica-se ainda que os tempos de assinatura dos dados em claro e dos dados encriptados, quando utilizado o método de duplicação por ficheiros, são bastante aproximados. Quando é

utilizado o processamento por blocos de tamanho fixo aumentam as diferenças entre as tempos para as duas fases. O diferencial ainda é maior no processamento por blocos de tamanho variável, verificando-se este fenómeno em todos os dispositivos.

Este efeito deve-se à deteção de duplicações de dados, no primeiro tipo de processamento não são detetadas duplicações, assim o volume de dados utilizado para gerar as assinaturas sobre os dados encriptados, e sobre os dados em claro são iguais.

Como é referido na Tabela 4 com a utilização do processamento por blocos de tamanho fixo já são encontradas duplicações, crescendo este valor no processamento por blocos de tamanho variável, o que implica uma diminuição do volume de dados a processar na fase de assinatura de dados encriptados e a respetiva diminuição de tempo de processamento.

No processamento por ficheiros o número de operações de inserção e busca na base de dados é muito pequeno, o que implica um tempo de execução bastante reduzido. Para os processamentos por blocos, apesar de o número de operações na base de dados ser idêntica para todos os dispositivos, os valores apontam um tempo de execução diverso para cada um dos equipamentos. Este facto indica que o tipo dispositivo tem um grande impacto na execução da fase de base de dados.

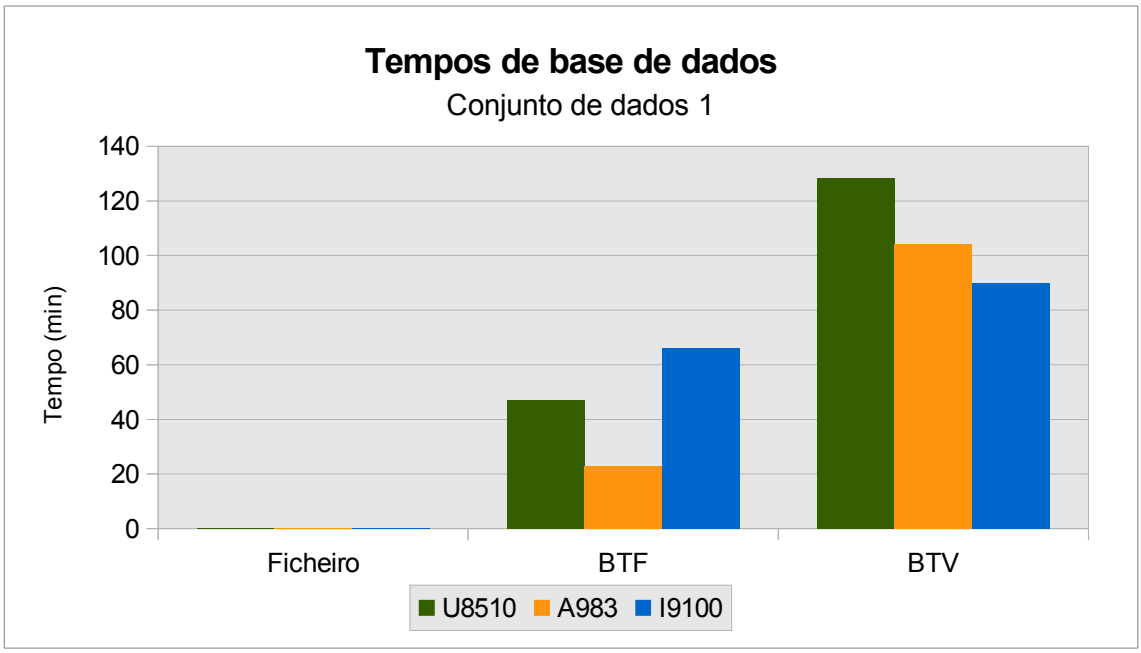


Gráfico 10: Tempos de base de dados - Conjunto de dados 1

Os tempos de transmissão dentro do mesmo método de deduplicação apresentam valores próximos mas variam entre os vários métodos de duplicação testados, como se pode verificar no Gráfico 11, o que indica que este fator está pouco dependente dos dispositivos, mas fortemente dependente do tipo de processamento.

Considerando o volume de dados efetivamente transmitido nos processamentos efetuados sobre o conjunto de dados 1, apresentados na Tabela 5 pode-se também verificar facilmente que os tempos de execução da fase de transmissão não é diretamente proporcional ao volume de dados transmitidos.

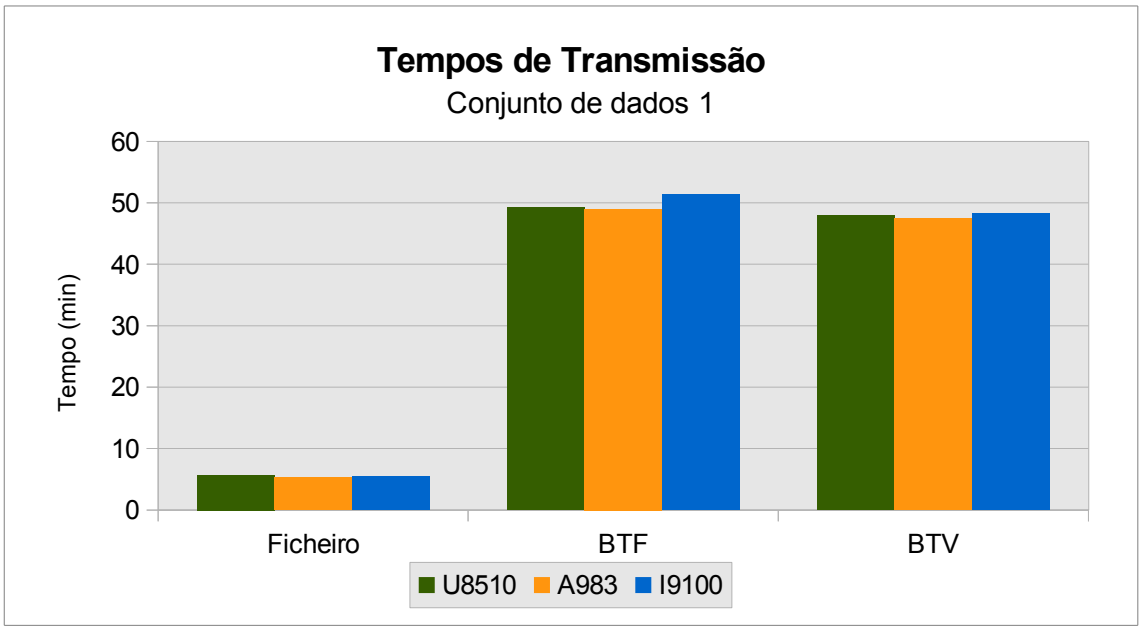


Gráfico 11: Tempos de transmissão - Conjunto de dados 1

Os testes decorreram em ambiente controlado de modo a que as condições de rede Wi-Fi fossem idênticas para todos os processamentos realizados, mas a alterações nas condições de acesso à rede por parte dos dispositivos pode implicar grandes alterações nos tempos de transmissão, mesmo com dispositivos e volumes de dados idênticos.

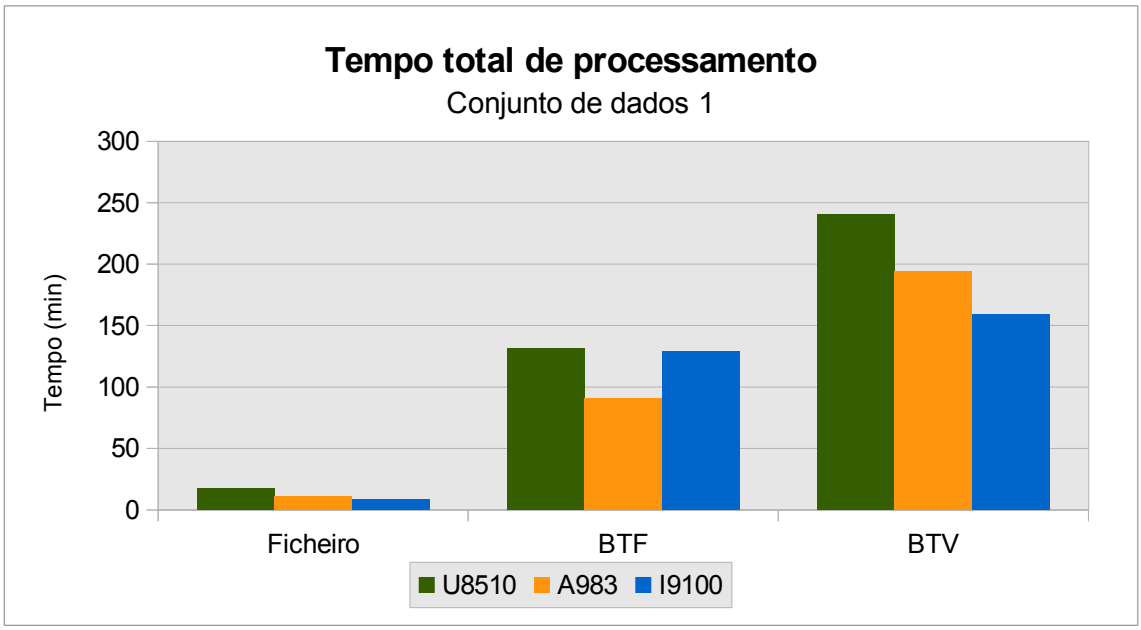


Gráfico 12: Tempo total de processamento - conjunto de dados 1

Como se pode verificar também no Gráfico 12 quando se utiliza o processamento por ficheiro o tempo de execução em todos os dispositivos é sempre bastante inferior a qualquer outro dos processamentos. Também se verifica que é o processamento onde as diferenças de tempo de execução entre os vários dispositivos é a menor.

Já se tinha verificado que a evolução do consumo de bateria era próxima de constante e não apresentava grandes diferenças entre os diversos tipos de processamento efetuados em cada um dos dispositivos utilizados nos testes.

Os gráficos 13, 14 e 15 mostram que também entre os vários dispositivos utilizados não se registam grandes diferenças na evolução do consumo de bateria durante a execução dos testes. Verificam-se alterações quando os testes são mais longos e os níveis de bateria atingem valores muito baixos o que obriga os dispositivos a entrarem em modo de poupança de energia, como é visível nos resultados do processamento por blocos de tamanho variável na fase final do processamento.

Pode-se assim verificar que a evolução do consumo de bateria nos processamentos está diretamente relacionado com o tempo que os mesmos processamentos demoram na sua execução.

Tanto o tipo de dispositivo como o tipo de processamento tem pouca influência na evolução do consumo de bateria, mas já quanto ao consumo total de bateria estes fatores tem um forte impacto.

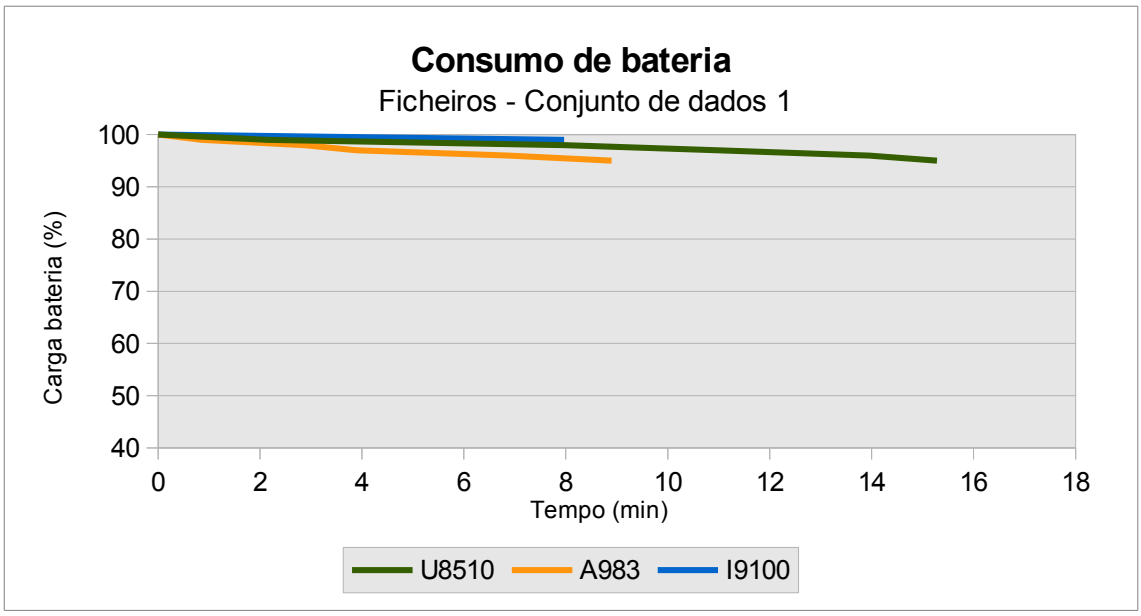


Gráfico 13: Consumo de bateria - Processamento ficheiros - Conjunto de dados 1

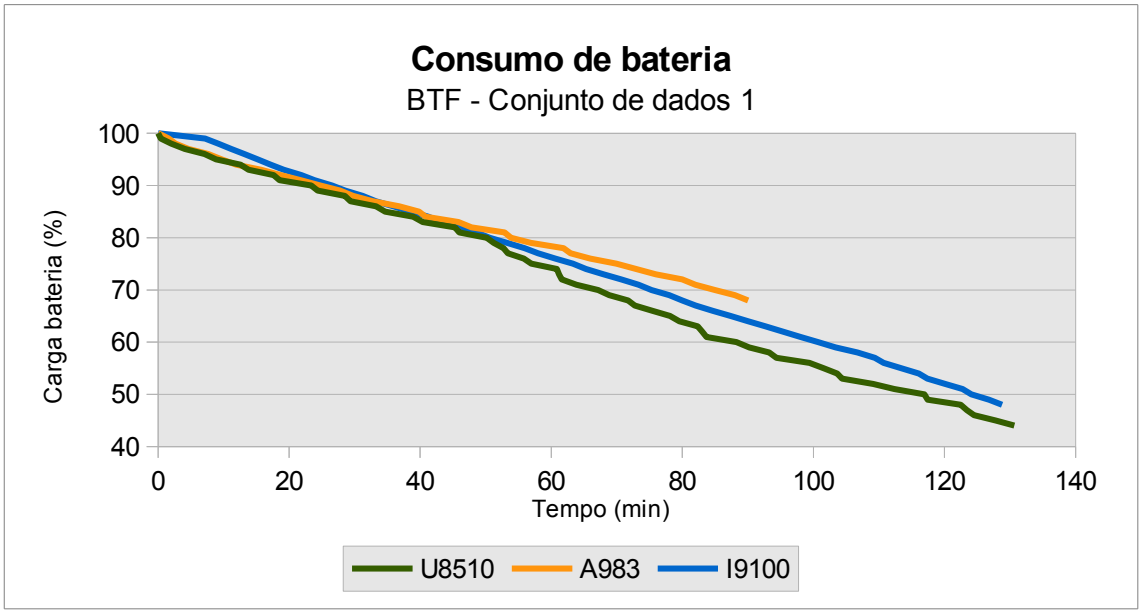


Gráfico 14: Consumo de bateria - Processamento BTF - Conjunto de dados 1

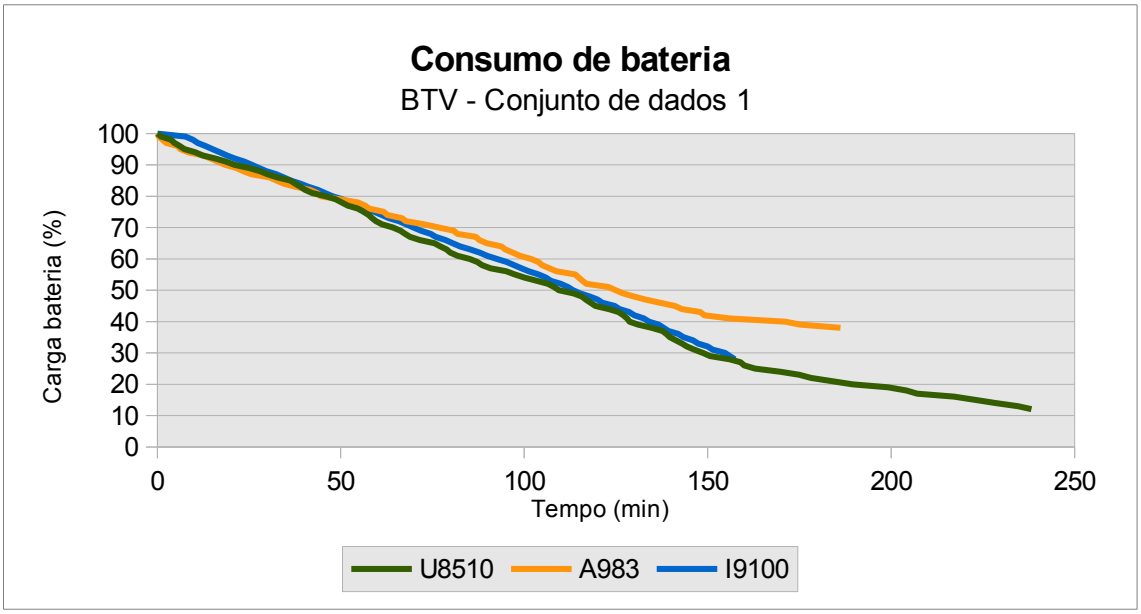


Gráfico 15: Consumo de bateria - Processamento BTV - Conjunto de dados 1

5.7 Processamento do conjunto de dados 2

Após o processamento do conjunto de dados 2 foi possível obter o valores gerais sobre cada método de deduplicação.

Tipo de deduplicação	Ficheiro	BTF	BTV
Blocos processados	123	115.262	115.261
Blocos repetidos	110	105.452	107.458
Volume enviado (bytes)	40.498.899	40.524.241	32.223.129
Tamanho meta-dados inicial (bytes)	19.456	13.496.320	13.467.648
Tamanho meta-dados final (bytes)	21.504	14.687.232	14.502.912

Tabela 16: Valores obtidos no processamento do conjunto de dados 2

No processamento por ficheiros o número de blocos processado é idêntico ao número de ficheiros do conjunto de dados 123, sendo detetados 110 ficheiros repetidos, o que

corresponde ao número ficheiros existentes no momento 0, com exceção do ficheiro que foi alterado entre os dois momentos de execução.

Os ficheiros do conjunto de dados 2 foram repartidos em 115.262 blocos e 115.261 blocos respetivamente para o método de blocos de tamanho fixo e variável, apresentado um valor muito próximo de blocos processados. No entanto o valor de blocos detetados como duplicados foi superior quando utilizado o processamento com blocos de tamanho variável, 107.458 contra os 105.452 para o processamento com blocos de tamanho fixo.

Como aconteceu com o conjunto de dados 1 o volume de dados transmitido foi próximo para o método de deduplicação por ficheiros e de blocos de tamanho fixo, apresentado um valor bastante inferior quando utilizado o método de blocos de tamanho variável, devido ao efeito da maior deteção de duplicações que este método permite.

A utilização do método de deduplicação por ficheiros apenas necessita de guardar uma assinatura por cada ficheiro, criando por isso uma base de dados com um tamanho bastante inferior às situações em que se utilizam os outros métodos de deduplicação.

No processamento por blocos de tamanho variável, mesmo com a necessidade de guardar mais informação por cada bloco, este fator é compensado pelo menor número de assinaturas que necessita de arquivar devido à deteção de mais duplicações, apresentando assim um tamanho final da base de dados inferior ao da base de dados criada pelo processamento por blocos de tamanho fixo.

5.7.1 Resultados do equipamento Huawei U8510

Fase	Ficheiro	%	BTF	%	BTV	%
Assinatura (claro)	00:05:54	70,42%	00:13:05	27,29%	00:31:57	30,62%
Encriptação	00:00:58	11,73%	00:02:45	5,76%	00:02:47	2,67%
Assinatura (encriptado)	00:00:30	6,04%	00:01:14	2,60%	00:00:59	0,94%
Base de dados	00:00:00	0,17%	00:10:58	22,87%	00:47:44	45,74%
Transmissão	00:00:57	11,47%	00:08:31	17,75%	00:07:27	7,14%
Outros	00:00:00	0,18%	00:11:23	23,73%	00:13:27	12,89%
Total	**00:08:22**	**100,00%**	**00:47:59**	**100,00%**	**01:44:23**	**100,00%**

Tabela 17: Tempos de execução - Huawei U8510 - Conjunto de dados 2

A Tabela 17 e o Gráfico 16 apresentam os dados obtidos sobre os tempos de execução absolutos e relativos para os diversos métodos de deduplicação, quando utilizado o dispositivo Huawei U8510.

No processamento por ficheiros a fase de assinatura dos dados em claro representa uma fatia considerável do tempo total de processamento. A necessidade de gerar as assinaturas para todos os ficheiros processados para validar se são duplicados em relação ao momento de processamento anterior, justifica o peso atingido por esta fase. Em todas todas as outras fases o processamento só é realizado sobre os ficheiros que não são duplicados, o que em termos de volume de dados representa apenas uma pequena fração do volume total de dados, como é mostrado na Tabela 3. Sendo este dispositivo o que tem menos capacidade de processamento, é por isso também onde se verifica o maior peso relativo desta fase de processamento.

Este fator também influencia os processamentos por blocos de tamanho fixo e variável, implicando um peso relativo superior à fase de assinatura dos dados em claro, quando comparado com os dados obtidos para o processamento do conjunto de dados 1.

A fase de base de dados no processamento utilizando o método de deduplicação por blocos de tamanho variável também é fortemente influenciada pelo mesmo fator, já que são efetuadas buscas à base de dados para verificação de todas as assinatura “fracas” geradas pelo sistema de *rolling hash*.

Como em todos os outros testes verifica-se que a fase de transmissão é bastante mais rápida quando utilizado o processamento por ficheiros.

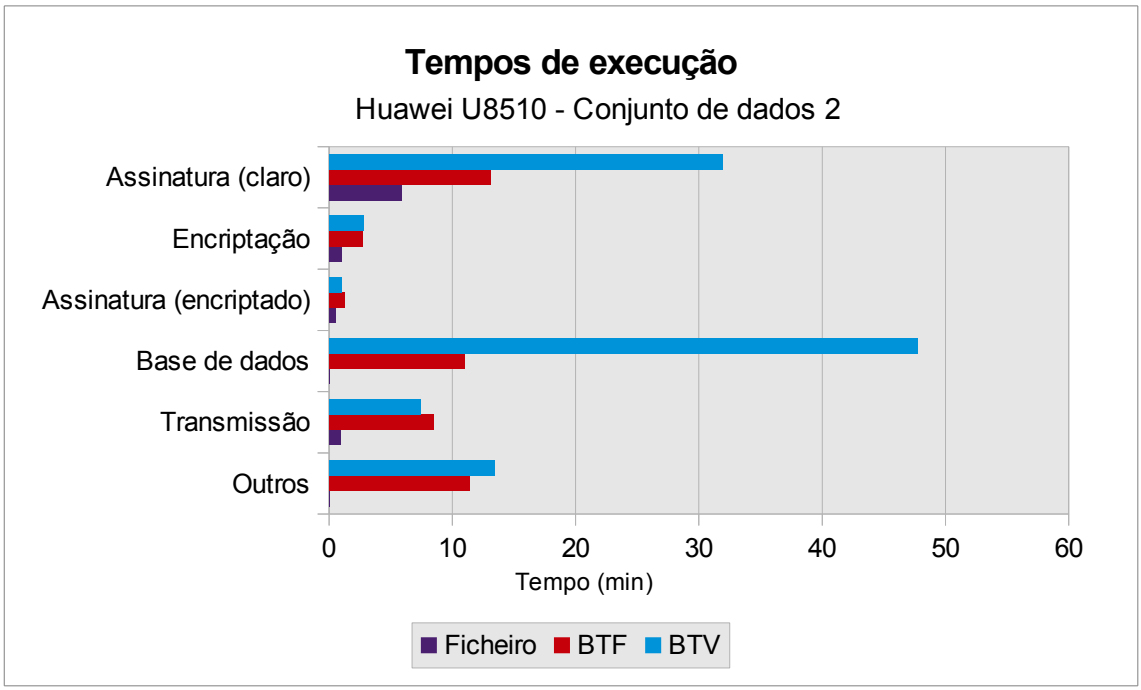


Gráfico 16: Tempos de execução - Huawei U8510 - Conjunto de dados 2

O consumo de bateria representados no Gráfico 17 é constante durante todos processamentos e apresenta consumos aproximados para os vários métodos de deduplicação testados.

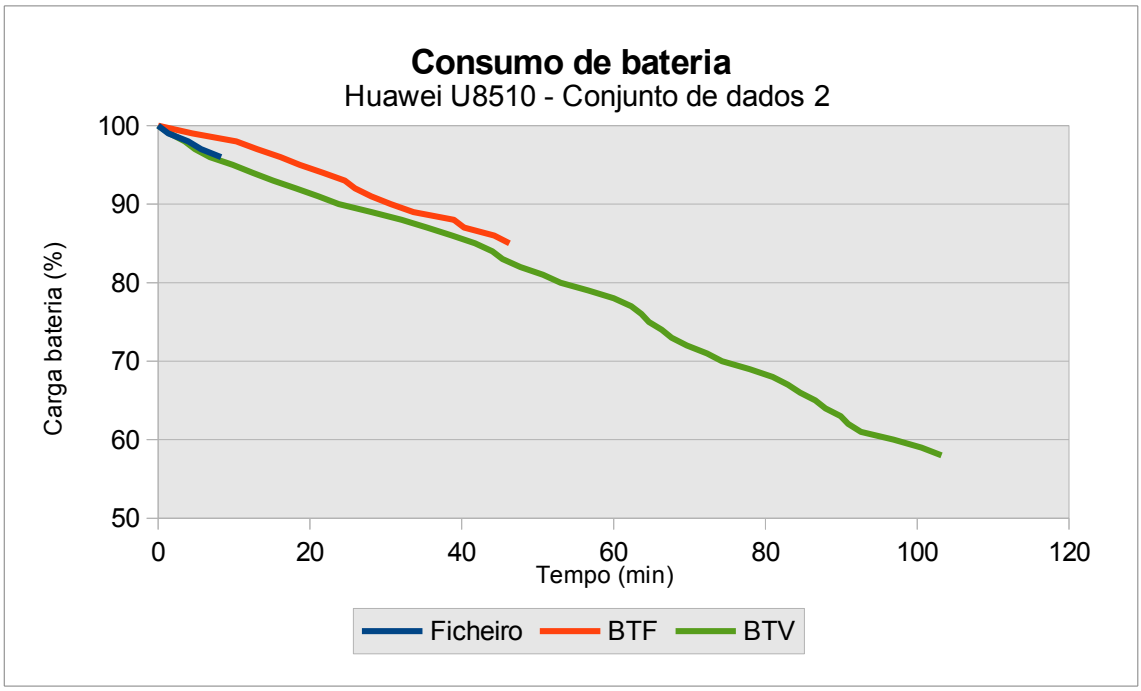


Gráfico 17: Consumo de bateria - Huawei U8510 - Conjunto de dados 2

5.7.2 Resultados do equipamento Alcatel A983

Fase	Ficheiro	%	BTF	%	BTV	%
Assinatura (claro)	00:03:35	64,62%	00:08:35	16,54%	00:16:29	26,32%
Encriptação	00:00:35	10,77%	00:01:40	3,22%	00:01:20	2,15%
Assinatura (encriptado)	00:00:18	5,47%	00:00:46	1,49%	00:00:35	0,93%
Base de dados	00:00:00	0,23%	00:04:25	8,51%	00:25:47	41,19%
Transmissão	00:01:01	18,49%	00:08:39	16,67%	00:07:10	11,45%
Outros	00:00:01	0,42%	00:27:50	53,58%	00:11:14	17,96%
Total	00:05:33	100,00%	00:51:58	100,00%	01:02:38	100,00%

Tabela 18: Tempos de execução - Alcatel A983 - Conjunto de dados 2

Dos testes efetuados com o dispositivo Alcatel A983, utilizando os vários métodos de deduplicação sobre o conjunto de dados 2 obtiveram-se os dados constantes na Tabela 7 e no Gráfico 18.

O facto de este teste representar o processamento diferencial implicando a geração de assinaturas para todos os ficheiros ou blocos do conjunto de dados, sendo prosseguido o processamento apenas com os ficheiros ou blocos não duplicados, o que implica um maior peso na fase de assinatura dos dados em claro em todos os tipos de processamento.

Neste dispositivo os valores obtidos para a fase de base de dados, quando utilizado o processamento por blocos de tamanho fixo são bastante mais baixos do que os registados nos outros dispositivos testados. Por outro lado este tipo de processamento apresenta um valor muito elevado no diferencial entre o tempo total de processamento e o somatório das fases analisadas que na Tabela 18 é identificado por “Outros”, que para este teste atinge os 53,58% do tempo total de processamento. Isto leva a considerar a hipótese de neste dispositivo existir algum tipo de *cache* que para este teste retardava de algum modo as operações de escrita efetiva na base de dados, não sendo estas apanhadas pela aplicação dentro da janela de aquisição de tempo para a fase processamento da base de dados. Este facto foi detetado durante a execução dos testes, tendo por esse motivo sido efetuados repetidos testes, inclusive com outros conjuntos de dados diferentes dos utilizados para a realização desta investigação. Todos os valores obtidos foram coerentes com os aqui apresentados.

O elevado peso atingido na fase de base de dados quando utilizado o método de deduplicação por blocos de tamanho variável, é consequência do elevado número de consultas necessário para verificação de todas as assinatura obtidas pelo sistema de *rolling hash*, que é efetuado sobre a totalidade dos blocos.

Como em todos os outros testes a fase de transmissão apresenta um valor bastante inferior quando utilizado o processamento por ficheiros, seguido por o processamento por blocos de tamanho variável, devido à maior deteção de blocos duplicados e consequente menor volume de dados transferidos.

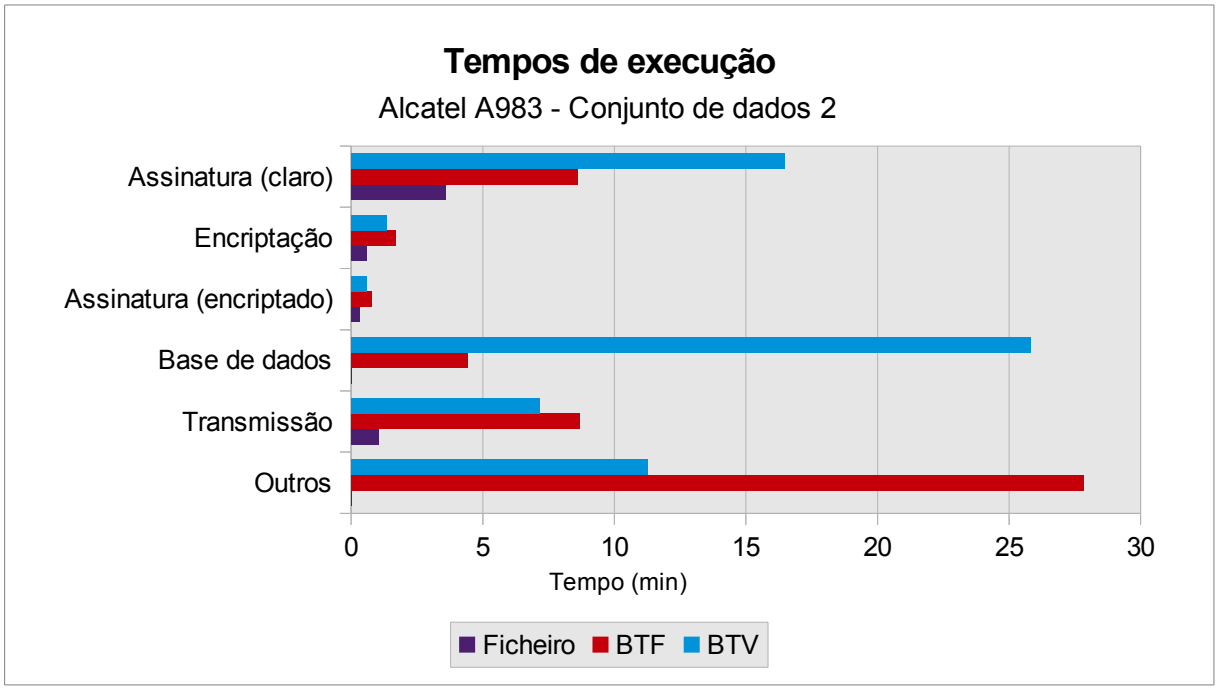


Gráfico 18: Tempos de execução - Alcatel A983 - Conjunto de dados 2

Conforme é possível constatar no Gráfico 19 também nestes testes o consumo de bateria é constante durante toda a execução dos testes e bastante aproximado para os vários métodos de deduplicação utilizados.

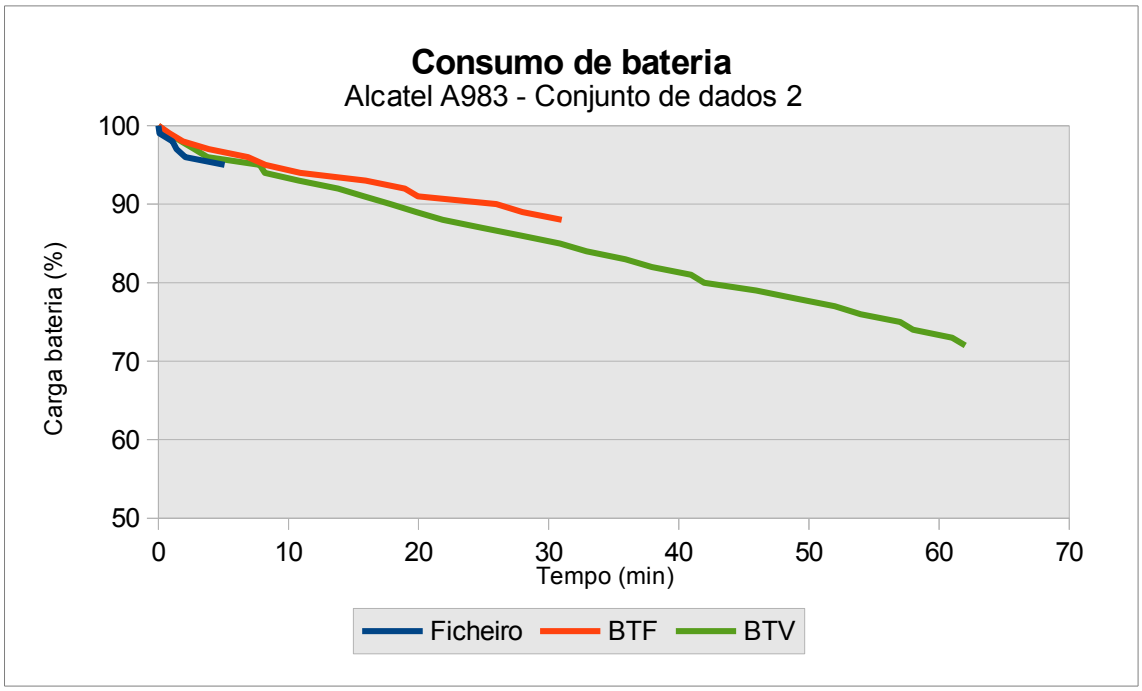


Gráfico 19: Consumo de bateria - Alcatel A983 - Conjunto de dados 2

5.7.3 Resultados do equipamento Samsung I9100

Fase	Ficheiro	%	BTF	%	BTV	%
Assinatura (claro)	00:01:36	55,66%	00:02:25	9,25%	00:05:00	14,97%
Encriptação	00:00:12	7,09%	00:00:46	2,94%	00:00:47	2,37%
Assinatura (encriptado)	00:00:07	4,49%	00:00:27	1,74%	00:00:19	0,97%
Base de dados	00:00:01	0,64%	00:12:11	46,41%	00:16:55	50,57%
Transmissão	00:00:55	32,00%	00:08:17	31,52%	00:06:32	19,55%
Outros	00:00:00	0,11%	00:02:08	8,14%	00:03:52	11,57%
Total	**00:02:53**	**100,00%**	**00:26:17**	**100,00%**	**00:33:28**	**100,00%**

Tabela 19: Tempos de execução - Samsung I9100 - Conjunto de dados 2

Na Tabela 19 e no Gráfico 20 estão representados os tempos de execução obtidos para a várias fases de processamento utilizando os diversos métodos de deduplicação, recolhidos após a execução dos respetivos processamentos utilizando o equipamento Samsung I9100 sobre o conjunto de dados 2.

Como já se verificou em todos os outros testes apresentados também neste dispositivo o processamento por ficheiros apresenta um tempo de execução muito mais reduzido que os outros tipos de processamentos.

Sendo realizado neste teste um processamento diferencial, o número de deteções de dados duplicados é maior que no casos dos processamentos efetuados sobre o conjunto de dados 1 em que é feito um processamento completo, o que tem por consequência um maior peso da fase de assinatura dos dados em claro, que nos testes com este dispositivo também se verifica.

A fase de base de dados volta a ser a que tem maior peso no tempo total de processamento quando se utilizam os métodos de duplicação que recorrem à utilização de blocos, em especial no caso do processamento por blocos de tamanho variável, onde é ocupado aproximadamente metade do tempo total de execução do teste.

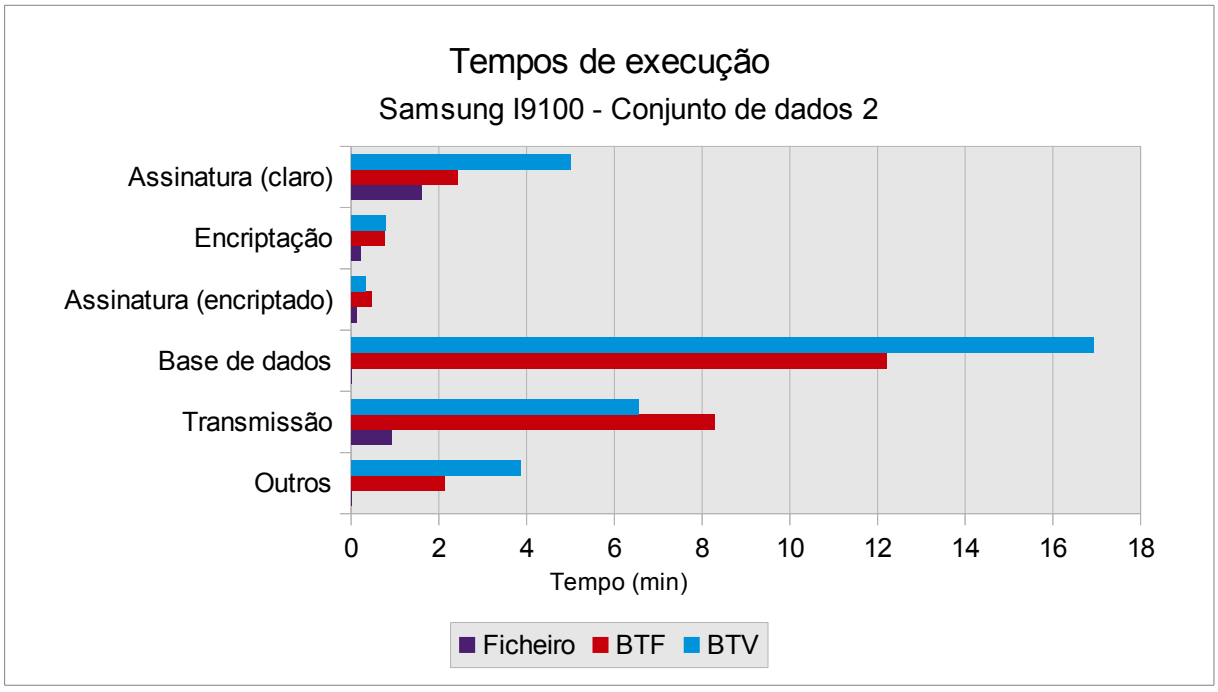


Gráfico 20: Tempos de execução - Samsung I9100 - Conjunto de dados 2

Na fase de transmissão o processamento por ficheiros é uma vez mais o mais eficiente em termos de tempo necessário para efetuar o envio de dados, apesar do volume de dados enviados ser aproximado ao valor de dados a enviar pelo processamento por blocos de tamanho fixo, o tempo de execução da fase de transmissão é aproximadamente 8 vezes

inferior. No processamento por blocos de tamanho variável o resultado do tempo de execução é inferior ao do método de deduplicação por blocos de tamanho fixo, devido à maior deteção de dados duplicados e consequente menor volume de dados a transferir.

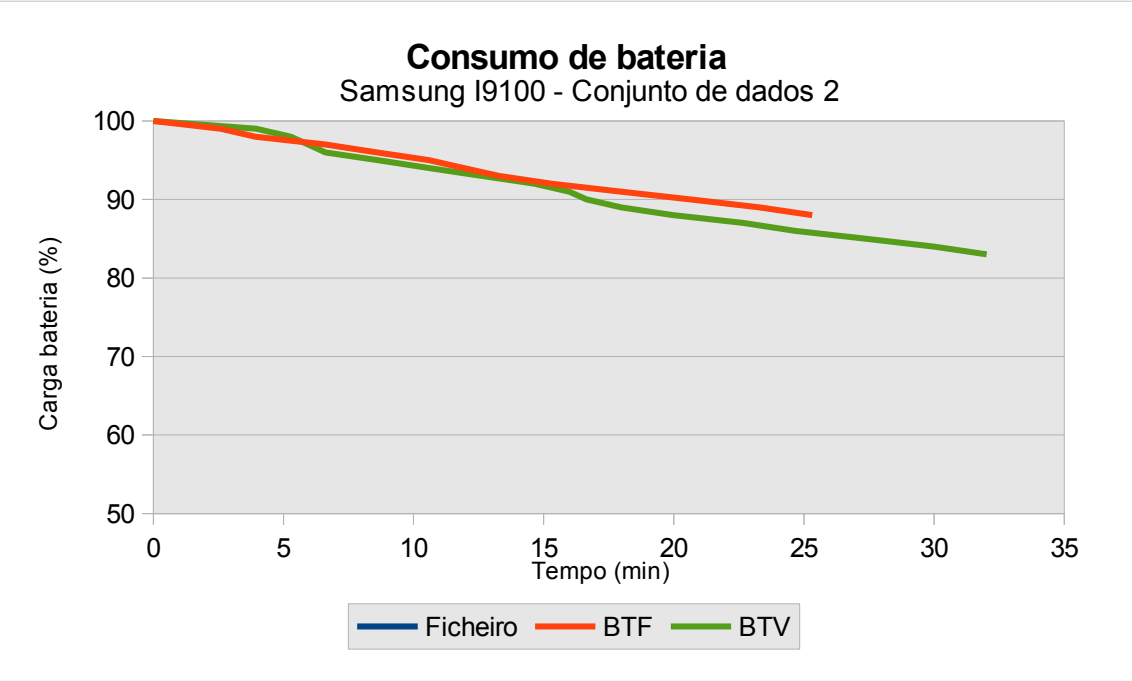


Gráfico 21: Consumo de bateria - Samsung I9100 - Conjunto de dados 2

Este dispositivo completou o teste utilizando o método de deduplicação por ficheiros com um consumo de bateria inferior a 1%, para os outros tipos de processamento apresenta um consumo constante durante toda a execução do teste, apresentando também uma evolução do consumo da carga da bateria aproximado nos dois tipos de processamento.

5.7.4 Comparação dos dados obtido nos vários dispositivos

Fase	Ficheiro			BTF			BTV		
	U8510	A983	I9100	U8510	A983	I9100	U8510	A983	I9100
Assinatura (claro)	00:05:54	00:03:35	00:01:36	00:13:05	00:08:35	00:02:25	00:31:57	00:16:29	00:05:00
Encriptação	00:00:58	00:00:35	00:00:12	00:02:45	00:01:40	00:00:46	00:02:47	00:01:20	00:00:47
Assinatura (encriptado)	00:00:30	00:00:18	00:00:07	00:01:14	00:00:46	00:00:27	00:00:59	00:00:35	00:00:19
Base de dados	00:00:00	00:00:00	00:00:01	00:10:58	00:04:25	00:12:11	00:47:44	00:25:47	00:16:55
Transmissão	00:00:57	00:01:01	00:00:55	00:08:31	00:08:39	00:08:17	00:07:27	00:07:10	00:06:32
Outros	00:00:00	00:00:01	00:00:00	00:11:23	00:27:50	00:02:08	00:13:27	00:11:14	00:03:52
Total	00:08:22	00:05:33	00:02:53	00:47:59	00:51:58	00:26:17	01:44:23	01:02:38	00:33:28

Tabela 20: Tempos de processamento - Conjunto de dados 2

Os valores obtidos em termos de tempos totais e dos tempos de cada uma das fases dos processamentos efetuados sobre o conjunto de dados 2 pelos dispositivos móveis utilizados, são apresentados na Tabela 20.

Tratando-se este teste de um processamento diferencial, a execução foi iniciada com uma base de dados que continha já uma série de meta-dados respeitantes a um processamento efetuado previamente, o que implica que nas fases de assinatura dos dados em claro e base de dados o processamento incida sobre a totalidade dos dados, enquanto que nas outras fases apenas se processam os dados não duplicados.

Pelo que seria de esperar que comparando com resultados do processamento do conjunto de dados 1 estas fases apresentassem tempos de execução superiores, até porque o volume de dados do conjunto 2 é 2,25 vezes superior.

No entanto comparando os dados da Tabela 15 e da Tabela 20 verifica-se que para a fase de assinatura de dados em claro existe de facto um aumento generalizado dos tempos de execução, mas para a fase de base de dados pelo contrário existe uma redução dos tempos de execução.

No processamento de dados do conjunto 1 o valor de dados duplicados é nulo ou muito baixo quando comparado com o valor de dados duplicados no processamento do conjunto de dados 2. No conjunto de dados 1 existe um elevado número de operações de inserção de novos registos em base de dados, enquanto que quando é utilizado o conjunto de dados 2 existe um maior número de operações de procura de assinaturas na base de dados. O que leva à

conclusão de que o impacto em termos de tempo das operações de escrita é superior ao das operações de leitura.

Tal como se verificou em relação ao processamento do conjunto de dados 1, também no conjunto de dados 2 pode constatar-se que nas 3 primeiras fases, que são as que necessitam de maior capacidade de processamento, em todos os métodos de deduplicação testados os resultados apontam para que a maior capacidade de processamento dos dispositivos está diretamente relacionada com a redução do tempo de execução para as fases de assinatura, como se pode ver no Gráfico 22 e no Gráfico 24 e para a encriptação como pode ser verificado no Gráfico 23

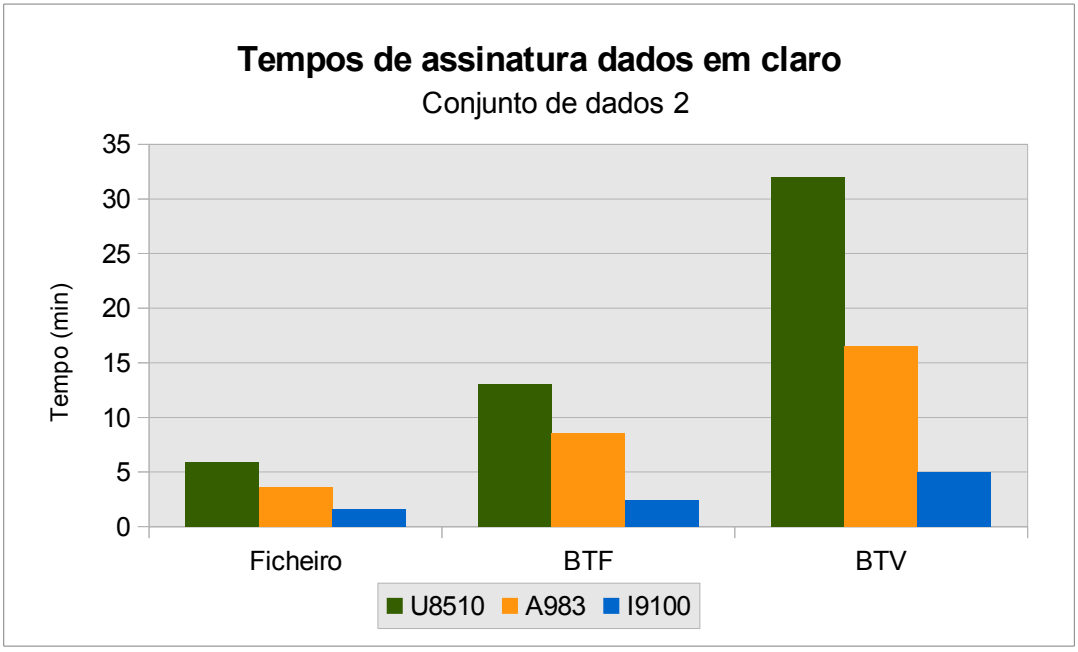


Gráfico 22: Tempos de assinatura de dados em claro - Conjunto de dados 2

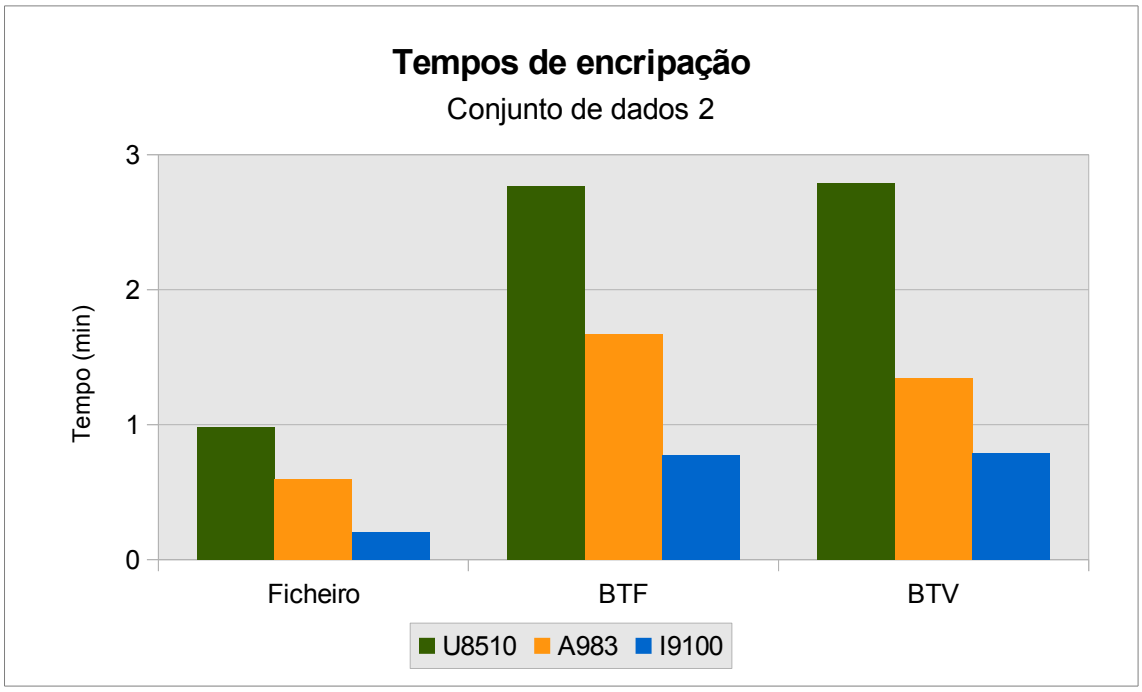


Gráfico 23: Tempos de encriptação - Conjunto de dados 2

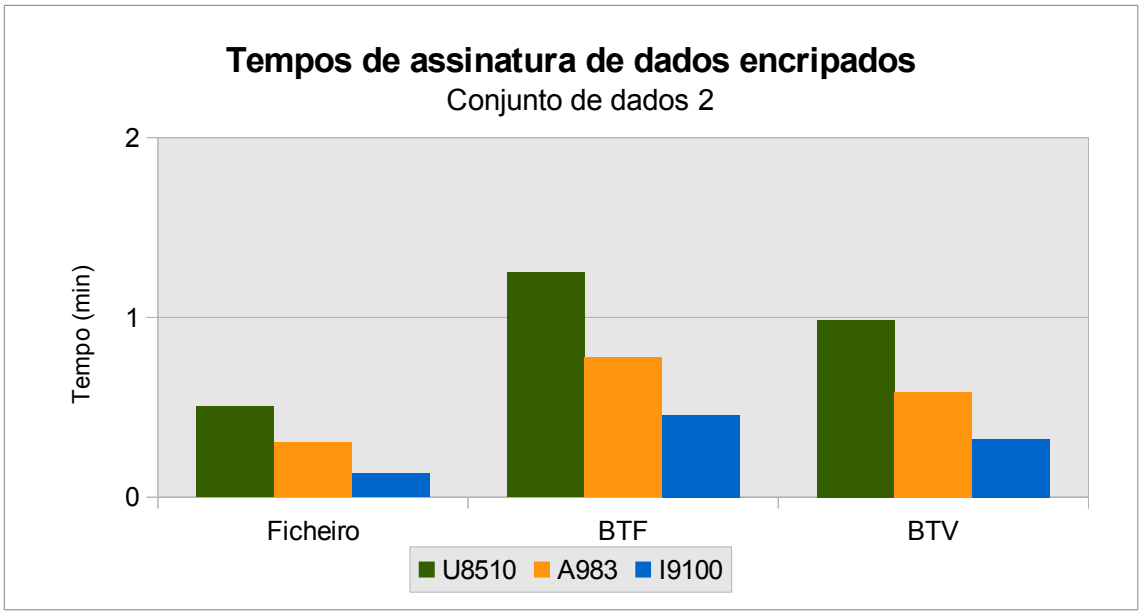


Gráfico 24: Tempos de assinatura dados encriptados - Conjunto de dados 2

A fase de base de dados no processamento por ficheiros apresenta valores muito baixos devido ao pequeno número de interações necessárias, apenas uma busca por cada ficheiro processado e uma inserção por cada ficheiro não duplicado.

O caso da fase de base de dados, no dispositivo Alcatel, já foi descrito na análise dos resultados do equipamento, o que justifica o baixo valor de tempo apresentado por este equipamento no método de duplicação por blocos de tamanho fixo.

No processamento por blocos de tamanho variável os diferentes dispositivos apresentam valores bastante dispares para um processamento idêntico.

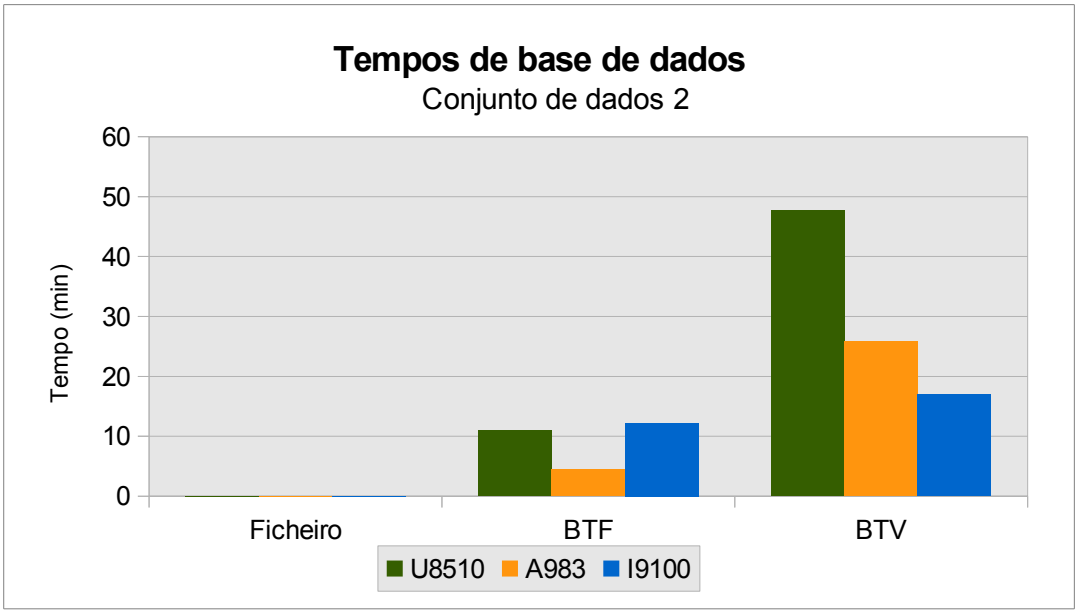


Gráfico 25: Tempos de base de dados - Conjunto de dados 2

Para a fase de transmissão o tipo de processamento utilizado tem maior influência nos resultados do que o equipamento utilizado.

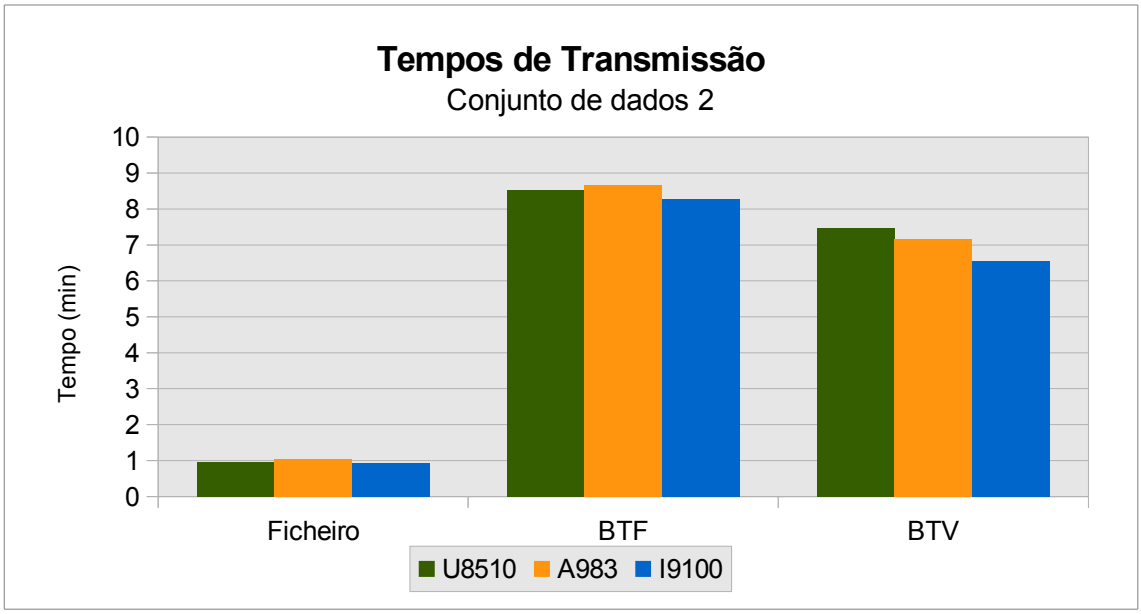


Gráfico 26: Tempos de transmissão - Conjunto de dados 2

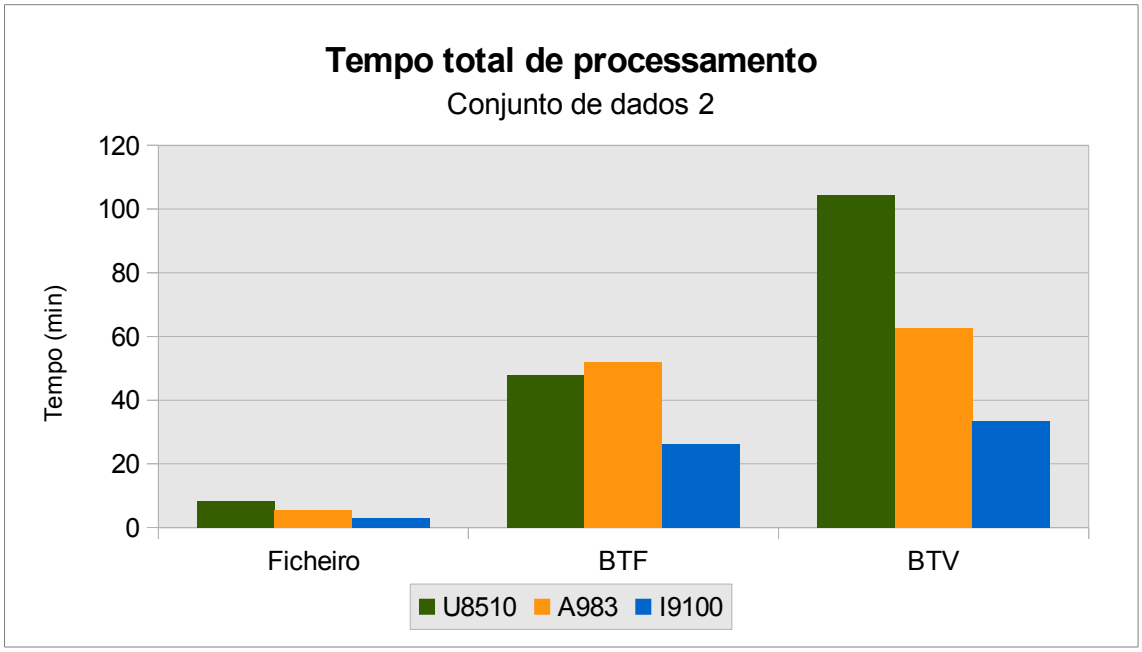


Gráfico 27: Tempo total de processamento - conjunto de dados 2

Tal como acontece para o conjunto de dados 1, também aqui, quando se utiliza o processamento por ficheiro, o tempo de execução em todos dispositivos é sempre bastante inferior a qualquer outro dos processamentos. Verifica-se igualmente que é o processamento onde as diferenças de tempo de execução entre os vários dispositivos são mais reduzidas, como se pode observar no Gráfico 27.

Já se tinha verificado, em todos os valores apresentados anteriormente neste estudo, que a evolução do consumo de bateria era aproximadamente constante e não apresentava grandes diferenças entre os diversos tipos de processamento efetuados em cada um dos dispositivos utilizados nos testes.

Também durante os processamentos efetuados sobre o conjunto de dados 2, os valores obtidos entre os vários dispositivos utilizados não se registam grandes diferenças na evolução do consumo de bateria, apresentando contudo maiores diferenças, como é observável no Gráfico 28, no Gráfico 29 e no Gráfico 30

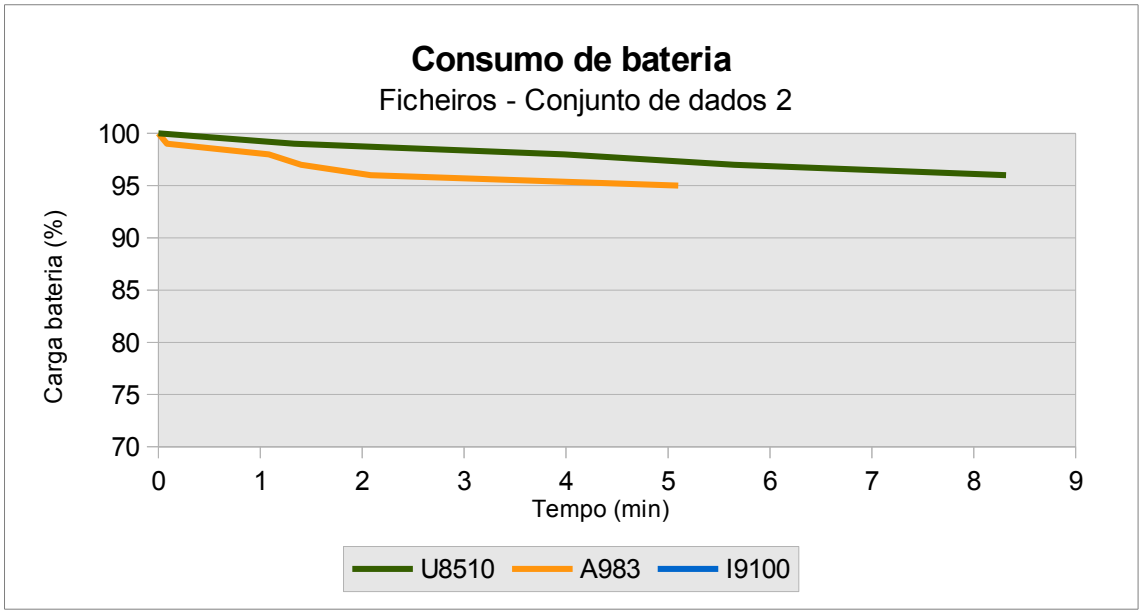


Gráfico 28: Consumo de bateria – Processamento por ficheiros - Conjunto de dados 2

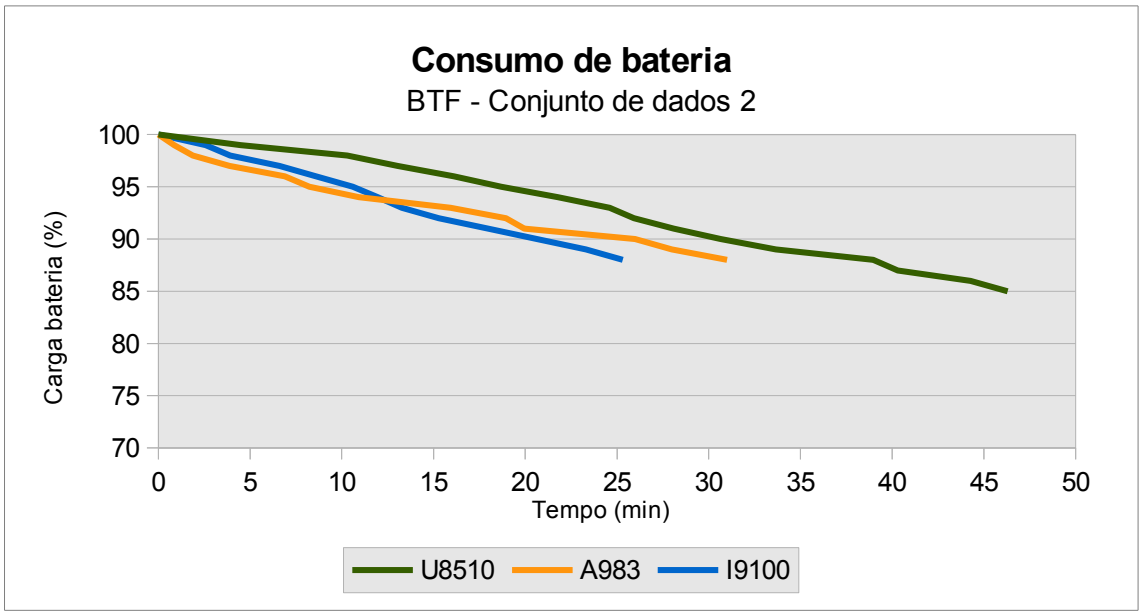


Gráfico 29: Consumo de bateria - Processamento BTF - Conjunto de dados 2

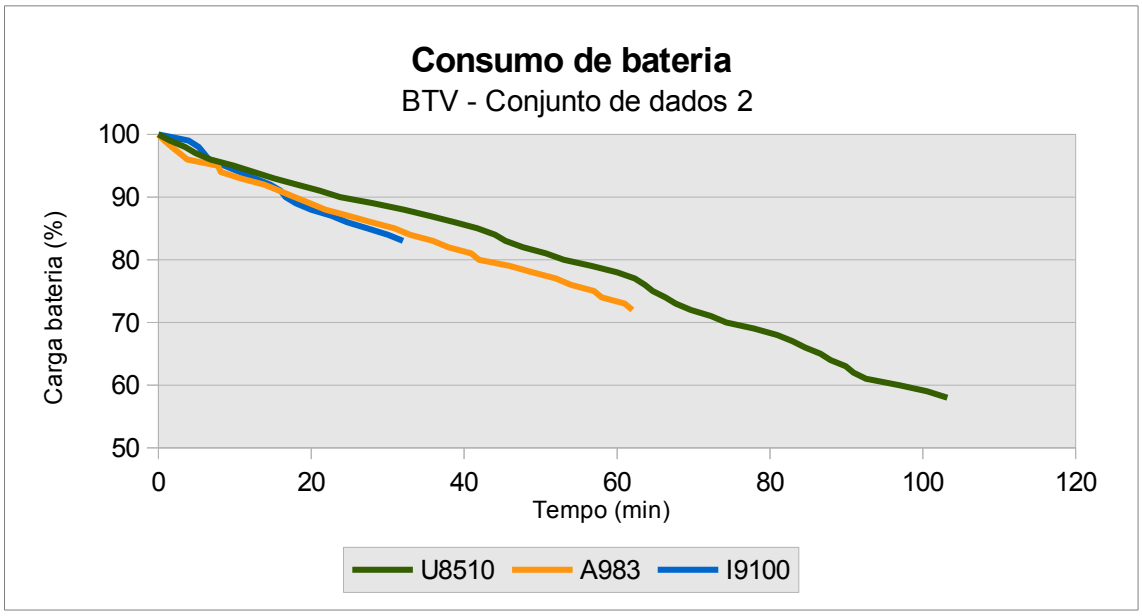


Gráfico 30: Consumo de bateria - Processamento BTV - Conjunto de dados 2

5.8 Utilização de redes móveis

Muitos dos dispositivo com sistema Android têm capacidade de comunicação através de redes móveis, era o caso de todos os equipamentos utilizados nos testes, pelo que é importante também obter informação sobre os vários tipos de processamentos utilizando redes móveis.

Como a utilização de redes móveis está normalmente associada a custos relacionados com o volume de tráfego utilizado, a situação mais normal será a execução de processamentos diferenciais, que tem normalmente um volume de tráfego inferior aos dados a processar.

Foi por isso utilizado o conjunto de dados 2 para os testes realizados sobre redes móveis.

Todos os testes foram realizados utilizando tecnologia HSDPA e de modo a terem o menor congestionamento por parte do operador foram efetuados em horas de baixo tráfego.

Fase	**Ficheiro**	**%**	**BTF**	**%**	**BTV**	**%**
Assinatura (claro)	00:01:36	4,44%	00:03:19	3,12%	00:05:55	6,21%
Encriptação	00:00:12	0,58%	00:01:54	1,79%	00:00:53	0,93%
Assinatura (encriptado)	00:00:08	0,38%	00:01:00	0,94%	00:00:23	0,42%
Base de dados	00:00:01	0,07%	00:12:36	11,85%	00:18:11	19,04%
Transmissão	00:34:25	94,53%	01:20:42	75,93%	01:03:48	66,80%
Outros	00:00:00	0,01%	00:06:45	6,36%	00:06:18	6,60%
Total	**00:36:24**	**100,00%**	**01:46:17**	**100,00%**	**01:35:31**	**100,00%**

Tabela 21: Tempos de execução - Samsung I9100 - Rede móvel - Conjunto de dados 2

A Tabela 21 e o Gráfico 31 mostram os resultados obtidos na execução dos testes utilizando os vários métodos de deduplicação sobre redes moveis, utilizando o dispositivo Samsung I9100.

Como era expectável não existem grandes diferenças nos resultados dos testes efetuados com o mesmo dispositivo utilizando Wi-Fi, apresentados na Tabela 19, e este teste que utilizou redes moveis, excetuando na fase de transmissão.

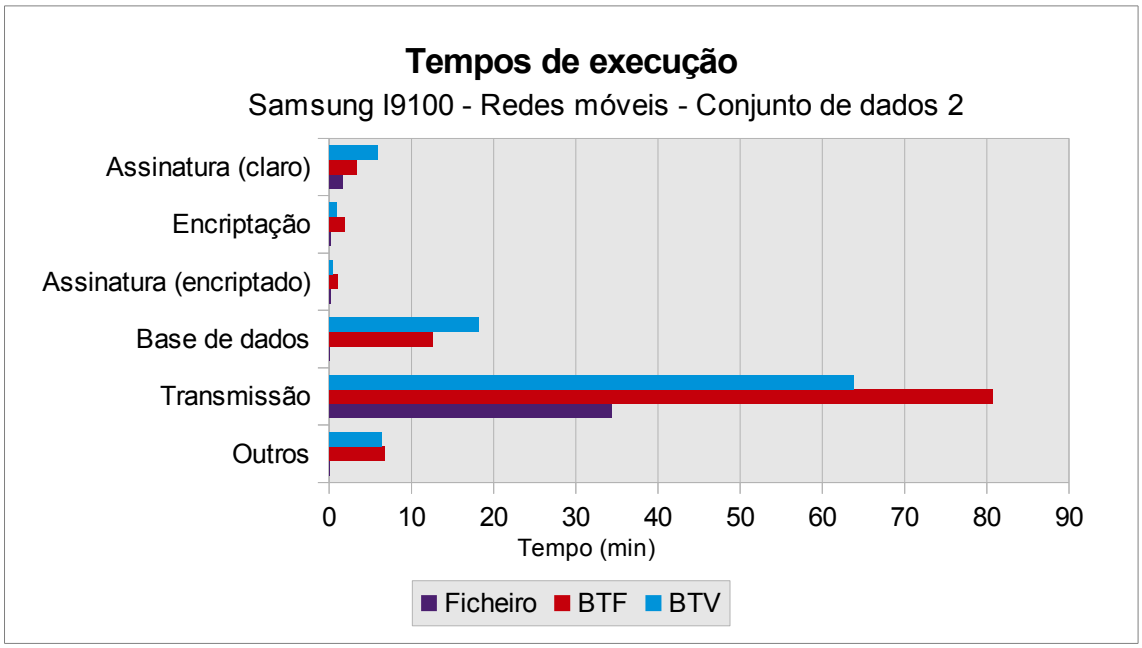


Gráfico 31: Tempos de execução - Samsung I9100 - Redes móveis - Conjunto de dados 2

Na fase de transmissão os tempos de execução foram bastante superiores aos dos testes idênticos, realizados recorrendo a comunicações por Wi-Fi.

Processamento	**Móvel**	**Wi-fi**	**%**
Ficheiro	00:34:25	00:00:55	2,66%
BTF	01:20:42	00:08:17	10,26%
BTV	01:03:48	00:06:32	10,24%

Tabela 22: Tempos de transmissão em Wi-Fi e Rede móvel

Como se pode verificar pelos resultados apresentados na Tabela 22 e no Gráfico 32 a maior diferença ocorreu no processamento por ficheiros onde o tempo utilizado na rede Wi-fi representa apenas 2,66% do tempo que foi necessário para efetuar a mesma transferência de dados utilizando redes móveis, apesar disso este tipo de processamento apresenta um tempo de execução bastante inferior aos outros métodos de deduplicação.

Nos processamentos utilizando blocos de tamanho fixo e variável a relação entre os tempos obtidos nos dois tipos de rede é semelhante.

Também se verifica que o facto do processamento por blocos de tamanho variável beneficiar de um número superior de deteções de duplicação e consequente redução do volume de dados a transmitir, implica um menor tempo de transmissão em relação à utilização de método de deduplicação por blocos de tamanho fixo.

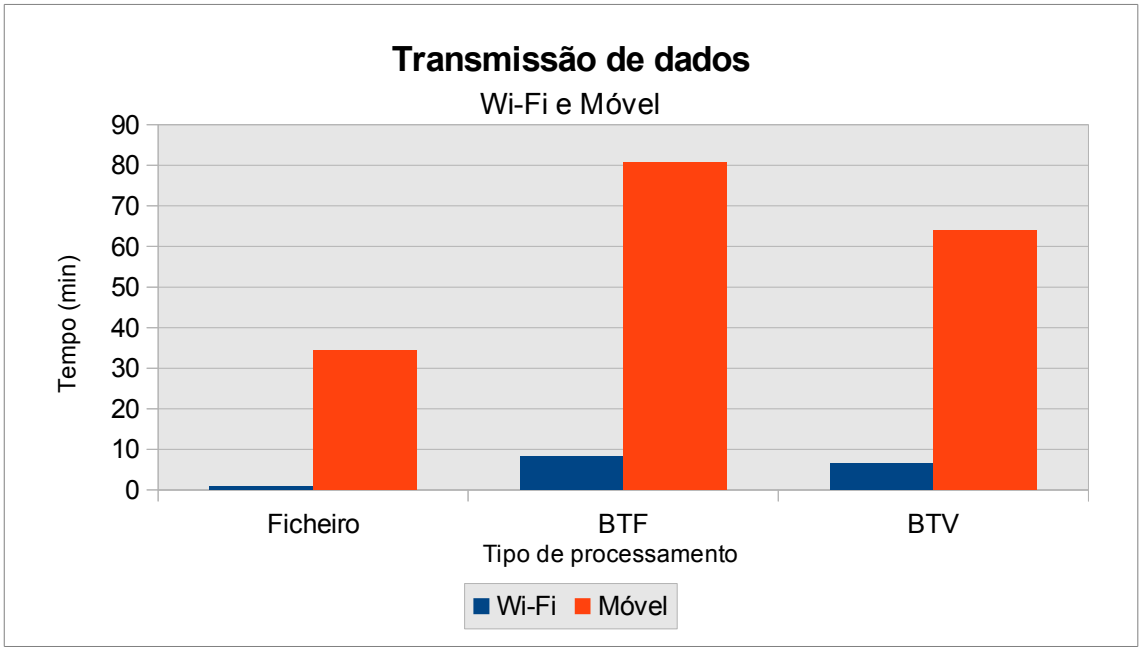


Gráfico 32: Transmissão de dados - Wi-Fi e rede móvel

O Gráfico 33 mostra que a evolução do consumo de bateria durante a execução dos teste manteve-se constante e bastante aproximada para os processamentos por blocos, mas ligeiramente inferior para o processamento por ficheiros.

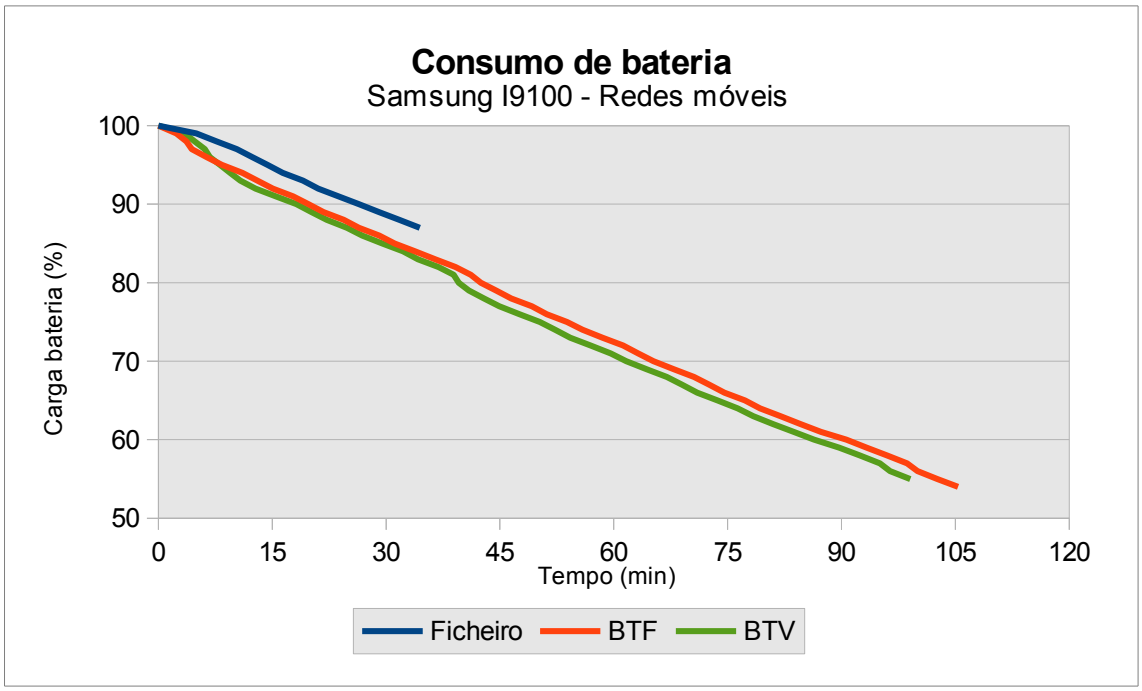


Gráfico 33: Consumo de bateria - Samsung I9100 - Redes móveis - Conjunto de dados 2

5.9 discussão dos resultados

De acordo com os testes realizados pode verificar-se que os resultados apresentados, nos processamentos por ficheiros o número de deteções de dados duplicados foi a menor e em contrapartida no processamento por blocos de tamanho variável foi superior aos outros métodos de deduplicação de acordo com outros estudos. (Meister & Brinkmann, 2009) Existindo também neste último método de deduplicação uma maior ocupação do tempo de processamento nas fase de assinatura de dados em claro. (Mandagere et al., 2008; Won et al., 2008)

Verificou-se também que o tamanho do arquivo de meta-dados é bastante reduzido quando se utiliza a deduplicação por ficheiros (He et al., 2010), enquanto nos outros métodos com a necessidade de arquivar informação sobre cada um dos blocos em que os ficheiros foram divididos, o tamanho do arquivo apresentou valores muito superiores. (Mandagere et al., 2008) Esta maior necessidade de armazenamento de meta-dados causou um aumento dos tempos de execução total devido à especificidade dos dispositivos que apresentam uma baixa velocidade de escrita na memória interna.

6 Conclusões

Foi possível a criação de um modelo para a execução de duplicação segura baseado nos conceitos de trabalhos anteriores, analisados na revisão da literatura. Partindo desse modelo conseguiu-se criar o prototipo com capacidade de executar os métodos de deduplicação segura em ambientes móveis e obter dados sobre cada um dos métodos.

Dos dados obtidos em todos os testes verifica-se que o consumo de bateria é constante durante todo o processamento e bastante semelhante para todos os métodos de deduplicação e dispositivos utilizados. A limitação da bateria está apenas relacionada com o tempo de execução do processamento.

A utilização dos procedimentos de geração de assinaturas e encriptação de dados, que são operações de uso intensivo de processador, foram executadas sem grandes custos de tempo em todos os dispositivos, mesmo quando utilizado o método mais exigente que é o de duplicação por blocos de tamanho variável. Este facto indica que em termos de capacidade de processamento é viável a utilização de sistemas de deduplicação segura em ambientes móveis.

O volume de armazenamento dos meta-dados é bastante dependente do método de deduplicação utilizada, exigindo um volume de espaço muito superior no processamento por blocos de tamanho fixo e variável quando comparado ao processamento por ficheiros. (He et al., 2010; Mandagere et al., 2008) Devido às características dos dispositivos móveis, que disponibilizam um baixo volume de armazenamento, a utilização dos dois métodos de duplicação que recorrem a blocos e o armazenamento local dos meta-dados pode ser inviabilizada. Este motivo impediu a utilização de alguns equipamentos nos testes realizados nesta investigação.

O armazenamento dos meta-dados no dispositivo também é bastante influenciado pela velocidade de leitura e particularmente da escrita no armazenamento interno de cada equipamento. Os métodos de deduplicação por blocos necessitam de guardar um volume considerável de meta-dados. A operação de inserção de elementos na base de dados é penalizada pela velocidade de escrita nestes dispositivos, sendo por isso a utilização destes métodos é bastante prejudicada em termos de tempo de execução.

Na fase de testes desta investigação foi explorada a utilização da memória externa para armazenamento dos meta-dados. Esta solução apresenta alguns problemas em relação à integridade dos dados, mas em contrapartida resolve a questão de falta de espaço de armazenamento. Os resultados obtidos apontaram para tempos de execução superiores aos da utilização da memória interna, pelo que foi abandonada tal solução.

Apesar de terem sido utilizados conjuntos de dados relativamente pequenos comparado com a capacidade de armazenamento externo que está disponível para utilização nestes diapositivos, os tempos de execução nos processamentos por blocos de tamanho fixo e variável chegaram a atingir as várias horas. Se extrapolarmos estes tempos de execução para volumes de informação na ordem dos vários GB, iríamos ter tempos de execução na ordem dos dias, o que mesmo considerando uma execução com o dispositivo ligado à corrente, ultrapassa os limites do razoável para o tempo de execução de uma cópia de segurança.

Se for tomado em consideração que nos testes realizados os ganhos em termos de deteção de dados duplicados, nos processamentos por blocos relativamente ao processamento por ficheiros, (Mandagere et al., 2008; Meister & Brinkmann, 2009; Won et al., 2008) quando comparado com o enorme aumento no tempo de execução, leva a desaconselhar a utilização destes métodos de processamento por blocos nas condições testadas.

Por outro lado a utilização de deduplicação por ficheiros apresentou em todos os testes tempos de execução mais baixos, utiliza um volume muito inferior de armazenamento para os meta-dados (He et al., 2010) e apresenta as melhores taxas de transferência de dados, apesar da limitação imposta ao tamanho de pacote para os testes realizados. Pode-se considerar por isso como um método viável para a utilização de processos de cópias de segurança em dispositivos móveis, para as condições testadas.

Assim em resposta à questão que se colocou no início deste trabalho, qual a melhor solução para deduplicação segura de dados em ambientes móveis ?. A análise dos resultados obtidos indica que o método de deduplicação por ficheiros se apresenta como a solução mais indicada.

7 Trabalhos futuros e limitações

Tendo em vista futuros desenvolvimentos da investigação realizada, poder-se-á testar a utilização de outros métodos de arquivo dos meta-dados, com utilização de outros sistemas de base de dados ou utilização do servidor para arquivar estes meta-dados. A redução de tempo gasto e de espaço utilizado na memória interna dos dispositivos móveis com o arquivo dos meta-dados, poderá viabilizar a utilização de métodos de deduplicação utilizando blocos de tamanho fixo e variável.

O tamanho da base de dados de meta-dados e o número de interações necessárias para a execução de processamentos que utilizam a segmentação dos ficheiros, é também influenciado pelo tamanho dos blocos. Poder-se-á analisar qual o impacto, no desempenho global, da utilização de tamanhos de blocos superiores nestes métodos de deduplicação.

Outra área onde se pode aprofundar a investigação é a segurança. Neste trabalho verificou-se que tanto na geração de assinaturas como na encriptação de dados, todos os dispositivos efetuaram os processamentos sem grande “esforço”. Poder-se-á testar a utilização de processos de assinatura de maior comprimento, a utilização de chaves maiores e de outros métodos criptográficos, com vista a obter uma maior segurança em todo o processo.

A otimização da transmissão de dados poderá também ser pesquisada. Os dados obtidos nos testes realizados durante esta investigação mostram taxas de transferência de dados bastante inferiores à largura de banda disponível. É certo que no caso do processamento por ficheiros foi imposto um pré-requisito do tamanho do pacote, idêntico ao tamanho dos blocos dos outros tipos de processamento, não existindo essa limitação, os valores de utilização de largura de banda seriam, nesse caso, superiores. Poder-se-á contudo testar outras soluções na transmissão, como a compressão de dados, ou aglutinação de blocos, de modo a obter uma melhor utilização da largura de banda e redução do tempo de transferência de dados.

Neste trabalho era pretendido obter uma medição do tempo de execução para cada uma das fases do processamento dos métodos de deduplicação, não tendo por isso utilizado processamento paralelo das várias fases, para que não existisse “contaminação” do tempo de execução entre elas. A utilização de várias *threads* poderá conduzir a um melhor desempenho global dos processamentos

Numa fase avançada do desenvolvimento do projeto foi tomado conhecimento de uma vulnerabilidade apresentada pelo sistema de criptografia convergente utilizado (Xu, Chang, & Zhou, 2013), pelo que já não foi possível proceder às alterações necessárias para a eliminação da vulnerabilidade. Pela avaliação feita dos trabalhos efetuados sobre o assunto, o impacto nos tempos de execução não deveriam ser significativos, afetando todos métodos de duplicação de forma idêntica. Deste modo qualquer trabalho futuro deverá tomar em consideração esta vulnerabilidade, alterando o procedimento de criptografia convergente.

8 Referências Bibliográficas

Anderson, P., & Zhang, L. (2010). Fast and secure laptop backups with encrypted de-duplication. Em *Proceedings of the 24th international conference on Large installation system administration* (pp 1–8).

Balasubramanian, N., Balasubramanian, A., & Venkataramani, A. (2009). Energy consumption in mobile phones: a measurement study and implications for network applications. Em *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference* (pp 280–293).

Bloom, B. H. (1970). Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, *13*, 422–426. doi:http://doi.acm.org/10.1145/362686.362692

Gantz, J. F., Reinsel, D., Chute, C., Schlichting, W., McArthur, J., Minton, S., … Manfrediz, A. (2008). The diverse and exploding digital universe: An updated forecast of worldwide information growth through 2011. *IDC white paper*.

He, Q., Li, Z., & Zhang, X. (2010). Data deduplication techniques. Em *Future Information Technology and Management Engineering (FITME), 2010 International Conference on* (Vol 1, pp 430–433). doi:10.1109/FITME.2010.5656539

Kulkarni, P., Douglis, F., LaVoie, J., & Tracey, J. M. (2004). Redundancy elimination within large collections of files. Em *Proceedings of the annual conference on USENIX Annual Technical Conference* (pp 5–5). Berkeley, CA, USA: USENIX Association. Obtido de http://portal.acm.org/citation.cfm?id=1247415.1247420

Liu, C., Lu, Y., Shi, C., Lu, G., Du, D. H. C., & Wang, D.-S. (2008). ADMAD: Application-Driven Metadata Aware De-duplication Archival Storage System. *Storage Network Architecture and Parallel I/Os, IEEE International Workshop on*, 29–35. doi:http://doi.ieeecomputersociety.org/10.1109/SNAPI.2008.11

Mandagere, N., Zhou, P., Smith, M. A., & Uttamchandani, S. (2008). Demystifying data deduplication. Em *Proceedings of the ACM/IFIP/USENIX Middleware '08 Conference Companion* (pp 12–17). New York, NY, USA: ACM. doi:http://doi.acm.org/10.1145/1462735.1462739

Meister, D., & Brinkmann, A. (2009). Multi-level comparison of data deduplication in a backup scenario. Em *Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference* (pp 8:1–8:12). New York, NY, USA: ACM. doi:http://doi.acm.org/10.1145/1534530.1534541

Min, J., Yoon, D., & Won, Y. (2011). Efficient Deduplication Techniques for Modern Backup Operation. *IEEE Transactions on Computers*, *60*, 824–840. doi:http://doi.ieeecomputersociety.org/10.1109/TC.2010.263

Nie, X., Qin, L., Zhou, J., Liu, K., Zhu, J., & Wang, Y. (2010). Optimization for data de-duplication algorithm based on file content. *Frontiers of Optoelectronics in China*, 1–9.

Park, N., & Lilja, D. J. (2010). Characterizing datasets for data deduplication in backup applications. Em *Workload Characterization (IISWC), 2010 IEEE International Symposium on* (pp 1–10).

Quinlan, S., & Dorward, S. (2002). Venti: a new approach to archival storage. Em *Proceedings of the Conference on File and Storage Technologies* (pp 89–101).

Rahmati, A., & Zhong, L. (2007). Context-for-wireless: context-sensitive energy-efficient wireless data transfer. Em *Proceedings of the 5th international conference on Mobile systems, applications and services* (pp 165–178).

Storer, M. W., Greenan, K., Long, D. D. E., & Miller, E. L. (2008). Secure data deduplication. Em *Proceedings of the 4th ACM international workshop on Storage security and survivability* (pp 1–10). New York, NY, USA: ACM. doi:http://doi.acm.org/10.1145/1456469.1456471

Tan, Y., Feng, D., Yan, Z., & Zhou, G. (2010). DAM: A DataOwnership-Aware Multi-layered De-duplication Scheme. *Networking, Architecture, and Storage, International Conference on*, 403–411. doi:http://doi.ieeecomputersociety.org/10.1109/NAS.2010.57

Wang, C., Qin, Z., Peng, J., & Wang, J. (sem data). A novel encryption scheme for data deduplication system. Em *Communications, Circuits and Systems (ICCCAS), 2010 International Conference on* (pp 265–269).

Wei, J., Jiang, H., Zhou, K., & Feng, D. (2010). MAD2: A scalable high-throughput exact deduplication approach for network backup services. Em *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on* (pp 1–14).

Won, Y., Kim, R., Ban, J., Hur, J., Oh, S., & Lee, J. (2008). PRUN : Eliminating Information Redundancy for Large Scale Data Backup System. *Computational Science and its Applications, International Conference*, 139–144. doi:http://doi.ieeecomputersociety.org/10.1109/ICCSA.2008.46

Xu, J., Chang, E.-C., & Zhou, J. (2013). Weak leakage-resilient client-side deduplication of encrypted data in cloud storage. Em *Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security* (pp 195–206). Obtido de http://dl.acm.org/citation.cfm?id=2484340

You, L. L., & Karamanolis, C. (2004). Evaluation of efficient archival storage techniques. Em *Proceedings of the 21st IEEE/12th NASA Goddard Conference on Mass Storage Systems and Technologies* (Vol 4, pp 227–232).

Zhu, B., Li, K., & Patterson, H. (2008). Avoiding the disk bottleneck in the data domain deduplication file system. Em *Proceedings of the 6th USENIX Conference on File and Storage Technologies* (pp 18:1–18:14). Berkeley, CA, USA: USENIX Association. Obtido de http://portal.acm.org/citation.cfm?id=1364813.1364831

9 Anexos

Anexo A

Paper produzido na fase inicial da investigação e apresentado em "OSDOC '11: *Workshop on Open Source and Design of Communication*"

Secure deduplication on mobile devices

Luis Marques
Department of Science and Information Technology
ISCTE-Instituto Universitário de Lisboa
lfmms@iscte.pt

Carlos Costa
Department of Science and Information Technology
ISCTE-Instituto Universitário de Lisboa
carlos.costa@iscte.pt

ABSTRACT
An increased use of mobile devices such as smart phones, PDAs and tablet computers with capabilities of storing and processing is rapidly growing. It allows users to carry an increasing volume of data, the value of this information must be protected from loss, theft or any kind of accident which may occur with the devices, making the backup process a key factor.

In mobile devices the data transmission has a high cost, in price and energy, and this devices depend on batteries, optimizing the bandwidth use is crucial. Deduplication allows substantial reductions in the data volume to be transmitted over network on backup sessions, by identifying common chunks of data, transferring and storing them only once.

On the other hand, usually mobile devices communications use mobile networks, and cloud storage is a viable solution for data backup.

Consequently it is essential the use of cryptographic processes to protect data, during the transmission but also in storage.

This paper proposes a model and a process for measuring the best performance of a safe and energy efficient backup system.

Keywords
Backup, Security, Deduplication, Mobile devices, Energy.

1. INTRODUCTION
The volume of information digitally stored is growing at hight rate and production of digital information is growing even more. A growth rate of 10 times every five years, exceeding already the storage capacity available. And the increased use of mobile devices like mobile phones, PDAs and tablet computers, with local storage capacity - but with the need to connect to networked storage across an increasingly connected world - is one of the factors that contributes to the storage volume required to grow 50% year in enterprises [1].

The increased use of backup systems based on random access devices as hard discs, makes available new methods that could be used to optimize the stored data, like deduplication.

Deduplication is a process that eliminates data redundancy, allowing an optimization of the space required for storage, and associated with less need for storage space there is also a reduced need for data transmission. This, in the case of backups as high impact, given the need for data retention at various time points, because there is usually a high percentage of data that is not changed between backup sessions, the redundancy is massive in archival sets [2].

Apart from reducing the storage space and bandwidth required to perform backups, deduplication also reduces the time used in performing each backup [3].

On the other hand, it is possible to witness a growing use of outsourced resources for data storage, and, there is a growing number of services available through the Internet for this purpose.

The use of external resources, however, raises the issues of security, both during data transmission and storage,[4] as well as, the use of bandwidth needed for transmission of data to be stored remotely. This is a factor which has a high importance when we talking about mobile devices, where the use of the mobile network for data transmission has high costs associated.

Both encryption and deduplication rely on mathematical systems with more or less complexity, hence processing is required, which in mobile devices raises two issues:

- The ability to make all process in an acceptable time window so it can be used to backup the data in device.
- The amount of energy required to perform such processing, as these devices are generally powered by batteries, thus they have limited energy.

When it intends to extend the process of secure deduplication to mobile devices, factors as the energy needed to perform the processing required for deduplication and encryption as well as the bandwidth usage, are of the most importance, however there is the possibility of using these principles in other more conventional systems.

The concepts of energy saving and green technologies deserve in present times special attention, the use of more efficient backups systems has a very high value[5].

2. RELATED WORK
There is an high degree of duplication and redundancy in stored data, not only in equal files, but also identical contents that are repeated in different files, especially in backup systems or version control systems.

This requires a huge amount of extra space for storage, [6] and when associated with remote storage systems using public networks the need for bandwidth can be huge [7].

Deduplication provides an efficient way to remove redundant data either inter-files or intra-files, [3], [8] this process identifies identical chunks of data saving only one instance of this sequence and, whenever it finds an identical sequence, it saves only the position of the sequence already saved [4], [5].

The detection of duplicate data can not be effected by comparing byte by byte due to performance of the process, instead it uses a comparison of fingertips, which are hash signatures computed by some hash function, [7], [9] like MD5 (120bits), SHA-1 (160bits) or SHA-256 (256bit).

The resistance to collisions grows as the size of the signature increases, but also the need of processing is increased for larger signatures [3].

In related works the SHA-1 is one of the most used for the production of signatures, given its resistance to collisions and efficient utilization of processor.

Hypothetically, the probability of collision is real, but if we consider a volume of information from one exabyte (10^{18} bytes) filled with blocks of 8 Kbytes, we have a number of approximately 10^{14} blocks. Using the SHA-1 function to calculate the signatures of the blocks we have a probability of 10^{-20} of an error of false data duplication, which could be classified as highly unlikely and as such can be ignored [10].

The probability of occurrence of an error triggered by a hardware failure exceeding a false identification of duplication.

2.1 Deduplication methods

The simplest process of deduplication acts at the file level, using the entire file to check for data duplication, where there is already an identical file it only saves a link to the previously stored file. This process is also referred to as the Whole File Chunking (WFC), Whole File Hashing (WFH) or Unique Document Storage [5], [6], [9], [11], [12].

This method needs a low level of supplementary information (meta-data), it only needs a hash signature per file, when the recovery of a file is requested, it doesn't need to rebuild the file from chunks in advance, unlike other methods [5].

Being also the method that requires less processing resources, [12] it is also the less efficient method in detection of redundancy, a simply change of a bit on a file, involves copying the entire file in the next backup session.

Other deduplication methods are based on the use of blocks or chunks of files to identify similarities, saving only the different blocks, there can be used two types of blocks, with fixed or variable size, [3], [6], [9], [11-13] this process also needs to keep the meta-data for each archived file, so they can rebuild the file from stored chunks [5].

The use of blocks and the block size has great impact on the time needed to rebuild the files, especially when the stored files are divided into a high number of blocks, [2], [12] decreasing the block size increases the detection of redundancy, but also increases the volume of meta-data.

The use of fixed-size blocks is conceptually simpler, faster and less demanding in terms of resources. It always divides files into blocks of same size, computes hash signatures for every chunk and checks for an exact match [3], [12].

The use of fixed blocks, however, is greatly affected by small changes in the files that involve the shift of the file contents, so, although the file may be quite similar to the previous version, it generates an entirely different set of chunks, in this case [3].

The variable-sized blocks systems use series of fingerprints for a file chunk, generated from a rolling checksum scheme as the Rabin algorithm [14], in order to identify similarities and define the block the boundaries [2], [9], and then processed as in fixed-size block method.

The use of variable-sized blocks is more efficient in detecting redundancies [9]. However this deduplication process requires greater processing power, spending 70% to 80% of run time to optimize the identification of blocks of identical data [3].

2.2 Deduplication optimization

There are also some works related to the deduplication optimization, to improve performance at server level, [2], [6-9], [15], [16] which are outside the scope of this work.

But also at the client side there are proposals for improvement.

The use of local indexes on the client side allows to check data changes compared to the last backup session, without requiring any data exchange with the server, [17] bringing improvements in runtime and use of bandwidth.

Using the Modulo-K algorithm allows an improvement of approximately 40% over the Rabin algorithm in terms of speed to identify similarities in variable-sized blocks [3].

2.3 Secure deduplication

Since it is intended to ensure the security of information, encryption must be used in conjunction with deduplication, but using encryption simultaneously with deduplication brings some problems as it has opposed objectives, when using different keys to encrypt the same data it gives different results, making it impossible to identify their similarity [4], [17].

The way to ensure the coexistence of encryption and deduplication is based on convergent encryption, which generates the encryption keys from the data to be encrypted [4], [17].

Since identical plain-text result in the use of identical keys, regardless of who does the encryption, for a given data chunk always result in the same cipher-text [4], [18].

When encryption of data occurs on the client side, it ensures the confidentiality of information during transmission, even if the system where data is stored is compromised [4].

2.4 Data transmission and energy consumption in mobile devices

The vast majority of mobile devices are capable of using various types of wireless networks - Wi-Fi, 3G and GSM - power consumption and transmission rate for each one is different.

The energy consumption it is not exclusively related to the data volume transmitted, but it is also very dependent on how the transmission is performed [19].

A small data volume transmitted intermittently can consume more energy than transferring a much larger block in one shot.

In 3G an great part of that energy is wasted because the system keeps a high-energy state several seconds after the transfer is completed, (tail energy) to maintain the link enabled for future communications, and also, but on a smaller scale, to move to a high-energy state before starting a communication (ramp energy).

These consumptions (tail and ramp) are constant and can be amortized in communications of large data blocks, or in frequent intermittent communications [19].

GSM have a similar scenario, but time in a high-energy state is much lower than in 3G. However the transfer rate is much lower, which implies a high energy consumption, in relation to the amount of information getting transferred [19].

With the highest transfer rates the Wi-Fi has the lowest energy cost per Mb transferred, compared with the mobile networks. However the energy used in searching for networks, or in maintaining the connection is high [19], [20].

Keep communications Wi-Fi turned off and only activate when needed, increases substantially the utilization time of battery charge in mobile devices [20].

The power consumption in wireless communications is dependent on factors such as network type, signal strength, connection time and volume of data transmitted [20].

3. PROPOSAL

Mobile devices depend on batteries, so when the battery is discharged the device becomes useless.

In the development of applications for such devices, energy consumption must be the primary concern. With this in mind we develop our proposal.

The objective of this work is to backup data in the device, this means data transmission, and it is precisely in the transmission that these devices need more energy.

As already described, deduplication can reduce substantially the volume of data that is transmitted when already exists at least one backup session. It is assumed that the first backup session has already been carried out in unusual conditions, with the device connected to an external power source and preferably using a Wi-Fi network to allow a shorter execution time.

3.1 Data hierarchy

At the first place, is it essential to backup all the data ? The data in the device has not all the same "value", there are various types of data with different importance.

The data is unique, or there is a copy of the same data in a server, with respective backup policies.

So it is important to create a hierarchy to the existing information that an interface must be available to a user in which it can assign various levels of importance to the data.

Obviously, when an operation, of any level, runs, data tagged with higher levels is also included in the session.

In our work we define 4 levels of importance to categorize the information, which are :

- Very Important Data (VID) - the crucial data, must be in all backup sessions.
- Less Important Data (LID) - excluded only in extreme conditions like poor mobile network signal.
- Ordinary Data (OD) - only when the optimal conditions are present, they are included in backup.
- No Need Backup (NNB) - data tagged with this level is never sent to backup.

The system must keep track of the date on which it was made the last backup session on each level, if the time elapsed since the last backup is long the system should perform a temporary upgrade of the data in each level, to prevent a large data set without backup on the mobile device

3.2 User Interface

The classification must be performed by the user, and can't be predefined because the same information can have different value for different users.

For example, the contacts and appointments, can be tagged as VID for one user and NNB to another, because it is connected to an Groupware server, so if he loses his mobile device simply sets up another device to access the server and synchronize the data and all the information is restored no new device.

Other settings that should be left to the discretion of the user, such as, if it can be performed a VID level operation when the device uses a roaming service, and if it can be carried out an operation level OD on mobile networks.

Such options may lead to high costs and depend on the contract the user has with the mobile networks service operator.

3.3 File types with specific processing.

There are some files types that it is not expected the occurrence of changes, and even in some cases where changes to a file are very undesirable, as is the case of the application binary files.

Multimedia content such as movies or pictures, are often found in this type of device, but are rarely changed. In this case can be avoided the additional processing required by the block-based deduplication.

In the case of applications can be detected unwanted changes in the file by action of viruses or any other malfunctions occurred. Being able to notify the user of these potentially unwanted changes

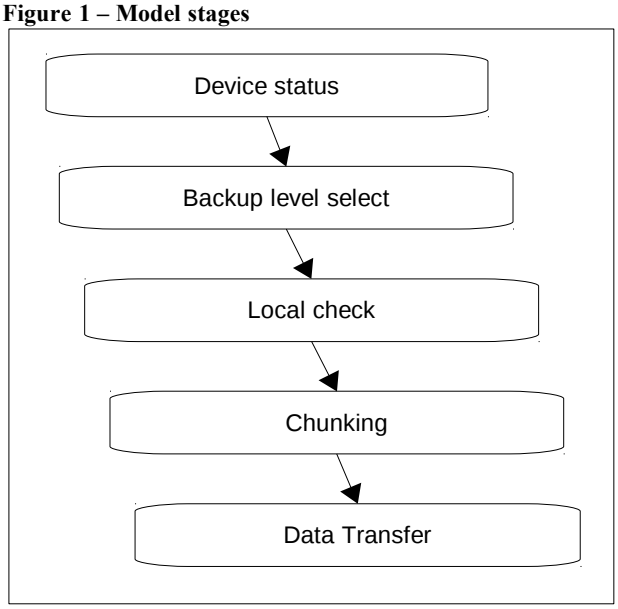
For this particular file types the process of deduplication should always be done at the file level, which has the smallest processing needs. Even when processing block-based deduplication.

This specific treatment not only saves energy in processing but also in the transmission, since all data is sent at one time, instead in an intermittent communication of smaller blocks.

3.4 Stages

In our model the process is divided in 5 stages as shown in figure 1

Figure 1 – Model stages



3.4.1 Device Status

To select the type of data that will be covered in each backup session it is essential to detect the mobile device status before proceeding with the backup process, so, the first stage in our process acquires the battery charge status and the types of available networks, if the battery is too low the process must enter into a wait state until the device is charged again.

As mobile devices they can be moved from one place to another and the state of available networks can be changed while running the backup, and so the next stage should change according with the new status of the device.

Obviously if the device is in airplane mode or there is no kind of network signal, the backup procedure cannot be performed.

3.4.2 Backup level select

With the available network status obtained previously it can be selected the level of backup to perform.

Table 1 – Backup level for available network

Backup level	Wi-Fi	3G	GSM	Roaming
VID			●	✣
LID		●		
OD	●	✣		

● - Standard execution ✣ - executed if it is allowed by user

A wi-fi network is available, so it is possible to perform an operation that includes all levels of information that require backup (OD level). If the device is using a roaming service it only allows a VID level backup if it is permitted by the user, or else, if only a low quality mobile network is present just a VID Level backup is processed. In the presence of a good quality mobile network, if allowed by user, it is executed a OD level backup else a LID level backup session is carried out.

3.4.3 Local check

To decrease the communications to the minimum possible it is maintained an index with the signatures generated from each file in the last backup session and stored in the device itself, this allows to validate the existing changes in the files since the last backup, without the need for data transmission. For each new file or changed file the information is transmitted to the next stage for processing

3.4.4 Chunking

In this stage, and again using the local index, file chunks that have been altered in relation to the last backup are searched. When different chunks are found the signature is transmitted to the server if no match is found, in the server, the chunk is submitted to the next stage for processing.

3.4.5 Data transfer

This stage process the encryption, compression and data transmission to the server.

In order to use a convergent encryption a key is generated based on the block that is being processed, then it encrypts the chunk with the previously generated key and it packs it with the hash signatures of the chunk.

The encryption process is described with more detail in section 3.5

Finally, the package is compressed and sent to the server.

All the transfers made in this stage are buffered to optimize communication and energy consumption.

It is also possible to send data in a random sequence, rather than sequentially as it is processed, making more difficult the reconstruction of information to an attacker that can intersect the transmission.

After a successful transmission of an entire file the new meta-data stored in the local index is committed.

3.5 Security

At the transmission process all the data must be encrypted. But, the sever needs to know some information in order to be able to perform data deduplication.

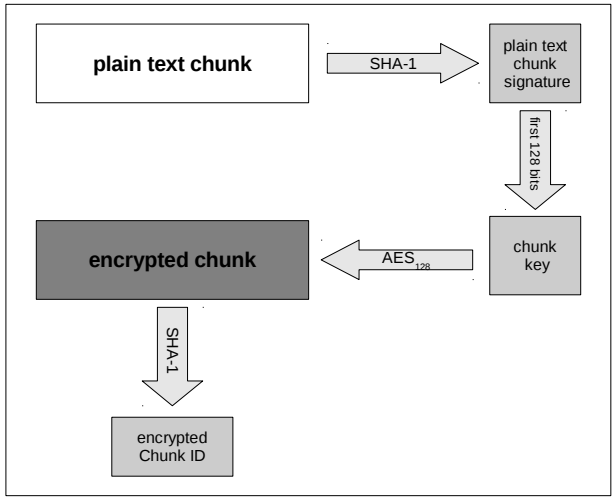
It will then be necessary to use two types of encryption :

- Symmetric cryptography, used in all the data that the server shall not be aware, but it is necessary in order to recover the stored data.
- Asymmetric cryptography, used in the data that the server needs to carry out storage processing, but it is necessary to protect during transmission.

Each ciphered chunk must assigned to a identifier, in order to further checks for matches. In our system, this identifier is the hash value of the encrypted chunk.

To avoid targeted collision attacks as described by Storer et al. [4] it is necessary to produce a identifier that can be verified by the storage system.

Figure 2 – Chunk encryption process



With this model it is possible in the server, check if the encrypted chunk ID corresponds to the content of the chunk.

All the encrypted chunk ID are maintained in a table, in order to perform quick searches for ID matches.

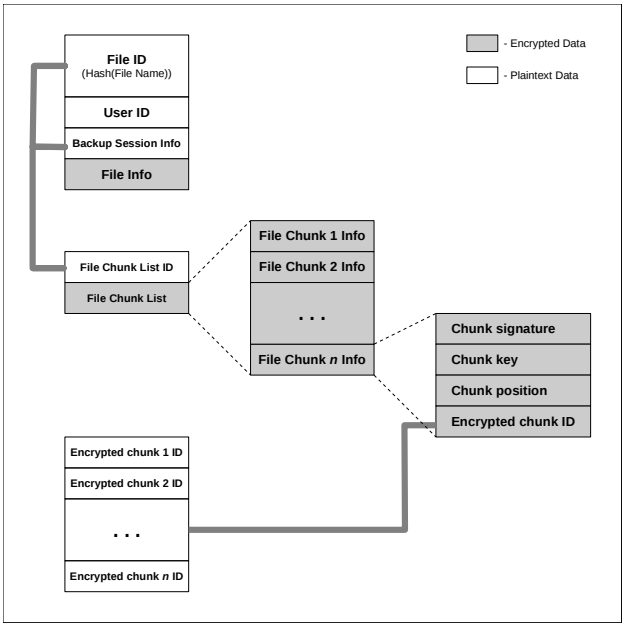
For each file version stored on the server will be required to maintain a list of chunks, but the server does not need to know the file name or the order of chunks within the file.

To hide the data from the server as file name, size or permissions is a calculated hash of the file name, which will serve as an identifier, all the other file information can be stored encrypted.

For every chunk it is necessary to save the chunk key, the encrypted chunk ID and the chunk position within the file. The information of all chunks of the file should be stored in a encrypted list of chunks and related the file identifier.

So even an attacker with root level on the server, only able to know the number of files stored for each user.

Figure 3 – Stored Meta-data structure



3.6 Devices and tools

The implementation of the proposed model should be done using available in open source technologies.

Based on the proposed model a Java application to run on Android systems is being developed. This will allow to carry out tests and acquire data on a wide range of equipment from several manufacturers.

The application can be tested not just in smart-phones but also in tablet computers, and even on netbooks running the Android x86

The server application will be implemented in a Linux server system. This server will be connected to de Internet using IP. The Android devices will be connected trough Wi-Fi or mobile network.

4. Evaluation

This is a work in progress, so in this chapter we propose the metrics to be used to evaluate the performance of the proposed conceptual model.

When looking at backup systems the parameters usually evaluated are the savings obtained in storage space, in amount of data transmitted or in run time.

As already mentioned there are several related works that indicate that the process of deduplication using variable-size blocks is the most effective in these parameters. But in a particular type of environment composed by mobile devices the power efficiency surpasses all other parameters.

Energy used in the backup process can be measured consulting the level of available energy in the battery at the beginning and at the end of the process. Because the energy level information that can be obtained by the software does not have high accuracy, tests should be done on a large data set to obtain reliable values. They must also be repeated several times and conducted on various and devices models from different manufacturers.

If succeed in filling a table like the following, registering the energy used for each stage with the specific conditions, it is possible to get some conclusions about the secure deduplication process on energy consumption perspective.

Table 2 – Energy consumption with local check

Network	Deduplication	Local check	Chunking	Data transfer	Over all
Wi-Fi	File Level				
	Fixed-size block				
	Variable-size block				
Mobile (3G)	File Level				
	Fixed-size block				
	Variable-size block				

The first two stages are not analyzed because the network used and the data set is predetermined in each situation.

The proposed model uses a local index to decrease the volume of data transmitted over the network to save energy, but on the other hand it involves the use of queries to find matches which also require energy for processing. It is intended to prove if the use of local Indexes effectively allows a reduction on energy consumption.

In the model without local check the Chunking stage must query the server by signatures and fingerprints of each block, in order to verify if they are identical.

To optimize communications when it starts to check a file should be made a pre-fetch from the server of all the meta-data for the file.

As mentioned above it is preferable to send a larger block at one time than doing many small communications.

Table 3 – Energy consumption without local check

Network	Deduplication	Chunking	Data transfer	Over all
Wi-Fi	File Level			
	Fixed-size block			
	Variable-size block			
Mobile (3G)	File Level			
	Fixed-size block			
	Variable-size block			

After obtaining the results and fill both tables, the assumption that the local check execution it is more efficient, can be verified in the values for the over all consumption in both situations

5. Discussion

At this point was achieved:

- A review of the literature related to deduplication, security and energy consumption in mobile devices transmissions.
- Proposed a model for implementing a system of secure deduplication on mobile devices.

- Some aspects of the user interface, and interactions required.
- Definition of evaluation metrics to verify the proposed model.

It is expected that energy consumption will increase as more complex processes are used in deduplication. But, then the volume of transmitted data is smaller and therefore less energy is needed. One effect compensates the other or the use of deduplication processes has a prohibitive cost in energy.

If on one hand the devices are increasing processing power, on the other it also has a greater storage capacity. It will be possible to backup all valuable information stored in devices, without requiring too much time and without spending all battery power.

Is it possible to use these secure and efficient procedures of backup on mobile devices already available on the market, all devices are ready or just the most recent, or it will be necessary to wait for the evolution of technology.

This work does not have answer to these questions. The implementation of the proposal model will allow, in future time, to answer the questions. As already said this is a work in progress, and the purpose and aim of this first step is just to create a model for future use.

6. REFERENCES

[1] J. F. Gantz et al., «The diverse and exploding digital universe: An updated forecast of worldwide information growth through 2011», *IDC white paper*, 2008.

[2] X. Nie, L. Qin, J. Zhou, K. Liu, J. Zhu, e Y. Wang, «Optimization for data de-duplication algorithm based on file content», *Frontiers of Optoelectronics in China*, p. 1–9, 2010.

[3] Y. Won, R. Kim, J. Ban, J. Hur, S. Oh, e J. Lee, «PRUN : Eliminating Information Redundancy for Large Scale Data Backup System», *Computational Science and its Applications, International Conference*, pp. 139-144, 2008.

[4] M. W. Storer, K. Greenan, D. D. E. Long, e E. L. Miller, «Secure data deduplication», in *Proceedings of the 4th ACM international workshop on Storage security and survivability*, New York, NY, USA, 2008, p. 1–10.

[5] Q. He, Z. Li, e X. Zhang, «Data deduplication techniques», in *Future Information Technology and Management Engineering (FITME), 2010 International Conference on*, 2010, vol. 1, p. 430–433.

[6] C. Liu, Y. Lu, C. Shi, G. Lu, D. H. C. Du, e D.-S. Wang, «ADMAD: Application-Driven Metadata Aware De-duplication Archival Storage System», *Storage Network Architecture and Parallel I/Os, IEEE International Workshop on*, vol. 0, pp. 29-35, 2008.

[7] B. Zhu, K. Li, e H. Patterson, «Avoiding the disk bottleneck in the data domain deduplication file system», in *Proceedings of the 6th USENIX Conference on File and Storage Technologies*, Berkeley, CA, USA, 2008, p. 18:1–18:14.

[8] N. Park e D. J. Lilja, «Characterizing datasets for data deduplication in backup applications», in *Workload Characterization (IISWC), 2010 IEEE International Symposium on*, 2010, p. 1–10.

[9] D. Meister e A. Brinkmann, «Multi-level comparison of data deduplication in a backup scenario», in *Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference*, New York, NY, USA, 2009, p. 8:1–8:12.

[10] S. Quinlan e S. Dorward, «Venti: a new approach to archival storage», in *Proceedings of the Conference on File and Storage Technologies*, 2002, p. 89–101.

[11] P. Kulkarni, F. Douglis, J. LaVoie, e J. M. Tracey, «Redundancy elimination within large collections of files», in *Proceedings of the annual conference on USENIX Annual Technical Conference*, Berkeley, CA, USA, 2004, p. 5–5.

[12] N. Mandagere, P. Zhou, M. A. Smith, e S. Uttamchandani, «Demystifying data deduplication», in *Proceedings of the ACM/IFIP/USENIX Middleware '08 Conference Companion*, New York, NY, USA, 2008, p. 12–17.

[13] J. Min, D. Yoon, e Y. Won, «Efficient Deduplication Techniques for Modern Backup Operation», *IEEE Transactions on Computers*, vol. 60, pp. 824-840, 2011.

[14] M. O. Rabin, *Fingerprinting by random polynomials*. Center for Research in Computing Techn., Aiken Computation Laboratory, Univ., 1981.

[15] Y. Tan, D. Feng, Z. Yan, e G. Zhou, «DAM: A DataOwnership-Aware Multi-layered De-duplication Scheme», *Networking, Architecture, and Storage, International Conference on*, vol. 0, pp. 403-411, 2010.

[16] J. Wei, H. Jiang, K. Zhou, e D. Feng, «MAD2: A scalable high-throughput exact deduplication approach for network backup services», in *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*, 2010, p. 1–14.

[17] P. Anderson e L. Zhang, «Fast and secure laptop backups with encrypted de-duplication», in Proceedings of the 24th international conference on Large installation system administration, 2010, p. 1–8.

[18] C. Wang, Z. Qin, J. Peng, e J. Wang, «A novel encryption scheme for data deduplication system», in *Communications, Circuits and Systems (ICCCAS), 2010 International Conference on*, p. 265–269.

[19] N. Balasubramanian, A. Balasubramanian, e A. Venkataramani, «Energy consumption in mobile phones: a measurement study and implications for network applications», in *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, 2009, p. 280–293.

[20] A. Rahmati e L. Zhong, «Context-for-wireless: context-sensitive energy-efficient wireless data transfer», in *Proceedings of the 5th international conference on Mobile systems, applications and services*, 2007, p. 165–178.

Anexo B

Código fonte da aplicação desenvolvida e utilizada nos testes.

```
public class BatteryStatus {
        private int batteryLevel;
        private long elapsedTime;

        public BatteryStatus (int batLevel, long eTime){
                this.batteryLevel = batLevel;
                this.elapsedTime  = eTime;
        }

        public int getBatteryLevel() {
                return batteryLevel;
        }

        public long getElapsedTime() {
                return elapsedTime;
        }

}
```

```
import java.io.File;
import java.util.LinkedList;
import java.util.Queue;
import android.os.Environment;
import android.os.Handler;
import android.os.Message;

public class DataSetLocate extends Thread {

        private static final String BASE_DIR_NAME  = "SecDedup-Dataset";

        private static SecDedupActivity mActivity;
        private static String dsBaseDir;
        private static int dsTotalFiles;
        private static long dsTotalSize;

        private String dsFilesDir;

        public DataSetLocate (SecDedupActivity mainActivity, String filesDir){
                DataSetLocate.mActivity    = mainActivity;
                DataSetLocate.dsBaseDir = Environment.getExternalStorageDirectory().getAbsolutePath();
                DataSetLocate.dsTotalFiles = 0;
                DataSetLocate.dsTotalSize = 0;
                dsFilesDir = filesDir;
        }

        public void run () {
                boolean dsFounded = false;
                Queue<File> dq = new LinkedList<File>();
                dq.add(new File(dsBaseDir));
                while ((dq.size() > 0) & (!dsFounded)) {
                        File[] files = dq.poll().listFiles();
                        if (files != null) {
                                for (int f = 0, n = files.length; f < n; f++) {
                                        if(files[f].isDirectory()){
                                                if (files[f].getName().equals(BASE_DIR_NAME))  {
                                                        dsBaseDir = files[f].getAbsolutePath();
                                                        dsFounded = true;
                                                        dsBasedir.sendEmptyMessage(0);
                                                        break;
                                                } else {
                                                        dq.add(files[f]);

                                                }
                                        }
                                }

                        }
                }
                if (dsFounded){
                        Queue<File> fq = new LinkedList<File>();
                        fq.add(new File(dsBaseDir + File.separator + dsFilesDir));
                        while (fq.size() > 0){
                                File[] files = fq.poll().listFiles();
                                for (int f = 0, n = files.length; f < n; f++) {
                                        if(files[f].isDirectory()){
                                                fq.add(files[f]);
                                        } else {
```

```
                                                if (files[f].length() > 0) {
                                                        dsTotalFiles++;
                                                        dsTotalSize += files[f].length();
                                                        dsInfoHander.sendEmptyMessage(0);
                                                        try {
                                                                Thread.sleep(5);
                                                        } catch (InterruptedException e) {
                                                                e.printStackTrace();
                                                        }
                                                }
                                        }
                                }
                        }
                        dsLocTerminated.sendEmptyMessage(0);
                } else {
                        dsProcError.sendEmptyMessage(0);
                }
        }


        private static Handler dsProcError = new Handler() {
        @Override
        public void handleMessage(Message msg) {
        mActivity.showDialog(SecDedupActivity.DATASET_NOT_PRESENT);
        }
        };

        private static Handler dsBasedir = new Handler() {
        @Override
        public void handleMessage(Message msg) {
        mActivity.set_dsBaseDir(dsBaseDir);
        }
        };

        private static Handler dsInfoHander = new Handler() {
        @Override
        public void handleMessage(Message msg) {
        mActivity.updateValue(SecDedupActivity.TOTALFILES, dsTotalFiles);
        mActivity.updateValue(SecDedupActivity.TOTALSIZE,  dsTotalSize);
        }
        };

        private static Handler dsLocTerminated = new Handler() {
        @Override
        public void handleMessage(Message msg) {
        mActivity.showStart();
        }
        };

}
```

```
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.security.DigestInputStream;
import java.security.InvalidAlgorithmParameterException;
import java.security.InvalidKeyException;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.security.NoSuchProviderException;
import java.util.Arrays;
import java.util.LinkedList;
import java.util.Queue;
import javax.crypto.BadPaddingException;
import javax.crypto.Cipher;
import javax.crypto.CipherOutputStream;
import javax.crypto.IllegalBlockSizeException;
import javax.crypto.NoSuchPaddingException;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.SecretKeySpec;
```

```
import android.os.Handler;
import android.os.Message;

public class DatasetProcess extends Thread {

        static final int FBUFFERSIZE              = 1024*4;
        private static final int FIXEDBLOCKSIZE   = 1024*4;
        private static final int SLIDESIZE        = 32;
        private static final int KEYSIZE          = 16;

        private static SecDedupActivity mActivity;
        private SecDedupDataSource dataSource;

        private File dsPath;
        private File tmpPath;
        private static int  dsProcFiles;
        private static long dsProcChuncks;
        private static long dsDupChuncks;
        private static long dsProcSize;
        private static long dsUpload;
        private int procDedupLevel;
        private boolean canceled;
        private SecDedupIO io;
        private TimeControl timeCtrl;


        public DatasetProcess (SecDedupActivity mainActivity, String dsBaseDir, String dsFilesDir) {
                DatasetProcess.mActivity    = mainActivity;
                this.tmpPath             = new File(dsBaseDir + File.separator + "TEMP");
                this.dsPath              = new File(dsBaseDir + File.separator + dsFilesDir);
                this.procDedupLevel = DatasetProcess.mActivity.getDedupLevel();
                this.canceled       = false;
                DatasetProcess.dsUpload = 0;
                dataSource = new SecDedupDataSource(mainActivity, this.tmpPath.getAbsolutePath());
                timeCtrl = new TimeControl(DatasetProcess.mActivity );

        }

        public void run () {
                if (! dataSource.isOpen()){
                        dataSource.reset();
                }
                if (! canceled)  {
                        dsProcFiles   = 0;
                        dsProcChuncks = 0;
                        dsDupChuncks  = 0;
                        dsProcSize    = 0;
                        dsUpload      = 0;
                        Queue<File> fq = new LinkedList<File>();
                        dsProcFile.sendEmptyMessage(0);
                        dsProcUpload.sendEmptyMessage(0);
                        fq.add(dsPath);
                        dsProcStarted.sendEmptyMessage(0);
                        while (fq.size() > 0){
                                File[] files = fq.poll().listFiles();
                                for (int f = 0, n = files.length; f < n; f++) {
                                        if (canceled) break;
                                        if(files[f].isDirectory()){
                                                fq.add(files[f]);
                                        } else {
                                                if (files[f].length() > 0) {
                                                        if (procDedupLevel == 1) {
                                                                dsProcFile(files[f]);
                                                                dsProcChuncks++;
                                                                dsProcSize += files[f].length();
                                                        } else if (procDedupLevel == 2) {
                                                                dsProcFileFSB(files[f]);
                                                        } else if (procDedupLevel == 3) {
                                                                dsProcFileVSB(files[f]);
                                                        }
                                                        dsProcFiles++;
                                                        dsProcFile.sendEmptyMessage(0);

                                                }
                                        }
                                }
                        }
                }
                if (! canceled)  {
```

```
                        dataSource.close();
                        dsProcTerminated.sendEmptyMessage(0);
                }
        }

        private String toHex(byte[] digest){
                if (! canceled) {
                    StringBuilder sb = new StringBuilder(digest.length*2);
                    for(byte b: digest)
                      sb.append(Integer.toHexString(b+0x800).substring(1));
                        return sb.toString();
                } else {
                        return "";
                }
        }

        public void cancel() {
                canceled = true;
                dataSource.close();
                timeCtrl.resetTimers();
                io.procCancel();
                dsProcCanceled.sendEmptyMessage(0);
        }



/*
 * File deduplication level
 */

        private void dsProcFile(File file) {
                byte[] clearFileHash = null;
                byte[] encryptedFileHash = null;
                if (! canceled){
                        timeCtrl.startTimer(TimeControl.CLEAR_HASH_TIMER);
                        clearFileHash = getFileHash (file);
                        timeCtrl.stopTimer();
                        String hexDigest = toHex(clearFileHash);
                        timeCtrl.startTimer(TimeControl.DBASE_TIMER);
                        if (! dataSource.hashExists(hexDigest)){
                                timeCtrl.startTimer(TimeControl.CRIPTO_TIMER);
                                File encryptedFile = encryptFile(file,clearFileHash);
                                timeCtrl.startTimer(TimeControl.CRIPTO_HASH_TIMER);
                                encryptedFileHash = getFileHash(encryptedFile);
                                timeCtrl.startTimer(TimeControl.COMM_TIMER);
                                sendFile(encryptedFile,encryptedFileHash);
                                timeCtrl.stopTimer();
                                if (! canceled){
                                        timeCtrl.startTimer(TimeControl.DBASE_TIMER);

                                        dataSource.addHash(hexDigest);
                                        timeCtrl.stopTimer();
                                }
                        } else {
                                timeCtrl.stopTimer();
                                dsDupChuncks++;
                        }
                }
        }

        private void sendFile(File encryptedFile, byte[] encryptedFileHash) {
                if (! canceled){
                        try {
                                dsUpload += io.sendFile(encryptedFile, encryptedFileHash);
                                dsProcUpload.sendEmptyMessage(0);
                        } catch (IOException e) {
                                cancel();
                                dsProcIOError.sendEmptyMessage(0);
                        }
                }
        }

        private File encryptFile(File file, byte[] clearFileHash) {
                byte[] buffer = new byte[FBUFFERSIZE];
                byte[] fKey = new byte[KEYSIZE];
                byte[] fIV = new byte[KEYSIZE];
                int dataRead;
                InputStream fIn;
                OutputStream fOut;
```

```
            File encryptedFile = new File(tmpPath.getAbsolutePath() + File.separator +"encrypted");
            if (! canceled){
                    System.arraycopy(clearFileHash, 0, fKey, 0, KEYSIZE);
                    System.arraycopy(clearFileHash,clearFileHash.length-KEYSIZE, fIV,  0, KEYSIZE);

                    try {
                            fIn = new FileInputStream(file);
                            fOut = new FileOutputStream(encryptedFile);
                            SecretKeySpec key = new SecretKeySpec(fKey, "AES");
                        IvParameterSpec ivSpec = new IvParameterSpec(fIV);
                        Cipher cipher = Cipher.getInstance("AES/CTR/NoPadding", "BC");
                        cipher.init(Cipher.ENCRYPT_MODE, key, ivSpec);

                        CipherOutputStream cos = new CipherOutputStream(fOut, cipher);
                        dataRead = 0;
                        while (((dataRead = fIn.read(buffer)) != -1) & (! canceled)) {
                             cos.write(buffer, 0, dataRead);
                             }
                        cos.flush();
                        cos.close();
                             fIn.close();
                    } catch (NoSuchAlgorithmException e) {
                            cancel();
                            e.printStackTrace();
                    } catch (NoSuchProviderException e) {
                            cancel();
                            e.printStackTrace();
                    } catch (NoSuchPaddingException e) {
                            cancel();
                            e.printStackTrace();
                    } catch (InvalidKeyException e) {
                            cancel();
                            e.printStackTrace();
                    } catch (InvalidAlgorithmParameterException e) {
                            cancel();
                            e.printStackTrace();
                    } catch (FileNotFoundException e) {
                            cancel();
                            e.printStackTrace();
                    } catch (IOException e) {
                            cancel();
                            e.printStackTrace();
                    }
            }
            return encryptedFile;
    }

    private byte[] getFileHash (File f){
            byte[] buffer = new byte[FBUFFERSIZE];
            MessageDigest hashFile = null;
            byte[] digest = null;
            int dataRead;
            try {
                    hashFile = MessageDigest.getInstance("SHA1", "BC");
            } catch (NoSuchAlgorithmException e) {
                    e.printStackTrace();
            } catch (NoSuchProviderException e) {
                    e.printStackTrace();
            }
            InputStream fIn;
            try {
                    fIn = new FileInputStream(f);
                    fIn = new DigestInputStream(fIn, hashFile);
                    dataRead = 0;
                    do {
                            dataRead = fIn.read(buffer);
                    } while ((dataRead != -1) & (! canceled));
                    if (! canceled) {
                            digest = hashFile.digest();
                    }
                    fIn.close();
            } catch (FileNotFoundException e) {
                    cancel();
                    e.printStackTrace();
            } catch (IOException e) {
                    cancel();
                    e.printStackTrace();
            }
            return digest;
```

```
        }

/*
 * Fixed size block deduplication level
 */

        private void dsProcFileFSB(File file) {
                byte[] buffer = new byte[FIXEDBLOCKSIZE];
                byte[] clearBlockHash = null;
                byte[] encryptedBlockHash = null;
                int dataRead = 0;
                InputStream fIn;
                try {
                        fIn = new FileInputStream(file);
                        do {
                                dataRead = fIn.read(buffer);
                                if (dataRead != -1) {
                                        timeCtrl.startTimer(TimeControl.CLEAR_HASH_TIMER);
                                        clearBlockHash = getBlockHash (buffer,dataRead);
                                        timeCtrl.stopTimer();
                                        String hexDigest = toHex(clearBlockHash);
                                        dsProcChuncks++;
                                        if (! dataSource.hashExists(hexDigest)){
                                                timeCtrl.startTimer(TimeControl.CRIPTO_TIMER);
                                                byte[] encryptedBuffer  =
encryptBlock(buffer,clearBlockHash,dataRead);
                                                timeCtrl.startTimer(TimeControl.CRIPTO_HASH_TIMER);
                                                encryptedBlockHash = getBlockHash (buffer,dataRead);
                                                timeCtrl.startTimer(TimeControl.COMM_TIMER);
                                                sendBlock(encryptedBuffer,encryptedBlockHash);
                                                timeCtrl.stopTimer();
                                                if (! canceled) {
                                                        timeCtrl.startTimer(TimeControl.DBASE_TIMER);
                                                        dataSource.addHash(hexDigest);
                                                        timeCtrl.stopTimer();
                                                }
                                        } else {
                                                dsDupChuncks++;
                                        }
                                        dsProcSize += dataRead;
                                        dsProcFile.sendEmptyMessage(0);
                                }
                        } while ((dataRead != -1) & (! canceled));
                        fIn.close();
                } catch (FileNotFoundException e) {
                        cancel();
                        e.printStackTrace();
                } catch (IOException e) {
                        cancel();
                        e.printStackTrace();
                }
        }


/*
 * Variable size block deduplication level
 */

        private void dsProcFileVSB(File file) {

                byte[] buffer = new byte[FIXEDBLOCKSIZE+SLIDESIZE];
                byte[] toProc = null;
                long rollingHash;
                byte[] clearBlockHash = null;
                int dataRead = 0;
                int slide = 0;
                int shift = 0;
                boolean dupBlock = false;
                InputStream fIn;
                try {
                        fIn = new FileInputStream(file);
                        RollingHash rHash = new RollingHash(FIXEDBLOCKSIZE, SLIDESIZE);
                        do {
                                dataRead = fIn.read(buffer, shift, FIXEDBLOCKSIZE + SLIDESIZE - shift);
                                if (dataRead != -1) {
                                        if (dataRead >= FIXEDBLOCKSIZE) {
                                                timeCtrl.startTimer(TimeControl.CLEAR_HASH_TIMER);
                                                rollingHash = rHash.getBlockHash(buffer);
                                                timeCtrl.stopTimer();
```

```
                                                slide = 0;
                                                do {
                                                        if (! canceled){

        timeCtrl.startTimer(TimeControl.DBASE_TIMER);
                                                                if
(dataSource.rolHashExists(rollingHash)) {

        timeCtrl.startTimer(TimeControl.CLEAR_HASH_TIMER);
                                                                        clearBlockHash = getBlockHash
(Arrays.copyOfRange(buffer, slide, FIXEDBLOCKSIZE + slide),FIXEDBLOCKSIZE);
                                                                        timeCtrl.stopTimer();
                                                                        String hexDigest =
toHex(clearBlockHash);

        timeCtrl.startTimer(TimeControl.DBASE_TIMER);
                                                                        dupBlock =
(dataSource.hashExists(hexDigest));
                                                                }
                                                                timeCtrl.stopTimer();
                                                                if (! dupBlock){

        timeCtrl.startTimer(TimeControl.CLEAR_HASH_TIMER);
                                                                        rollingHash=rHash.updateHash();
                                                                        timeCtrl.stopTimer();
                                                                        slide++;
                                                                }
                                                        }
                                                } while ((! dupBlock) & (slide < dataRead + shift -
FIXEDBLOCKSIZE) & (! canceled));
                                                if (! canceled) {
                                                        if (dupBlock) {

                                                                byte[] headBlk = null;
                                                                if (slide > 0){
                                                                        headBlk =
Arrays.copyOfRange(buffer, 0, slide);
                                                                }
                                                                if ((toProc != null) && (toProc.length
== FIXEDBLOCKSIZE)) {

        dsProcessBlock(toProc,toProc.length, null, -1);
                                                                        toProc = null;
                                                                }
                                                                if (toProc != null){
                                                                        if (headBlk != null) {
                                                                                byte[] tmpBuffer =
Arrays.copyOfRange(toProc, 0, toProc.length);
                                                                                toProc = new
byte[tmpBuffer.length+headBlk.length];

        System.arraycopy(tmpBuffer, 0, toProc, 0, tmpBuffer.length);

        System.arraycopy(headBlk, 0, toProc, tmpBuffer.length , headBlk.length);
                                                                        }
                                                                        dsProcessBlock(toProc,
toProc.length, null, -1);
                                                                        toProc = null;
                                                                } else {
                                                                        if (headBlk != null) {
                                                                                dsProcessBlock(headBlk,
headBlk.length, null, -1);
                                                                        }
                                                                }

        dsProcessBlock(Arrays.copyOfRange(buffer, slide, FIXEDBLOCKSIZE + slide),FIXEDBLOCKSIZE,
clearBlockHash, rollingHash);
                                                                if (slide < SLIDESIZE){
                                                                        System.arraycopy(buffer,
FIXEDBLOCKSIZE + slide, buffer, 0, SLIDESIZE - slide);
                                                                }
                                                                shift = SLIDESIZE - slide;
                                                                slide = 0;
                                                        } else {
                                                                if (toProc != null){
                                                                        dsProcessBlock(toProc,
toProc.length, null, -1);
                                                                }
                                                                toProc = Arrays.copyOfRange(buffer, 0,
```

```
FIXEDBLOCKSIZE);
                                                                System.arraycopy(buffer, FIXEDBLOCKSIZE,
buffer, 0, SLIDESIZE);
                                                                shift = SLIDESIZE;
                                                                slide = 0;

                                                        }
                                                }
                                        } else {
                                                if (! canceled) {
                                                        if (toProc != null){
                                                                if (toProc.length == FIXEDBLOCKSIZE) {

        dsProcessBlock(toProc,toProc.length, null, -1);
                                                                        toProc = null;

        dsProcessBlock(Arrays.copyOfRange(buffer, 0, dataRead + shift), dataRead + shift, null , -1);
                                                                } else {
                                                                        byte[] tmpBuffer = new
byte[toProc.length + dataRead + shift];
                                                                        System.arraycopy(toProc, 0,
tmpBuffer, 0, toProc.length);
                                                                        System.arraycopy(buffer, 0,
tmpBuffer, toProc.length , dataRead + shift);
                                                                        dsProcessBlock(tmpBuffer,
tmpBuffer.length, null , -1);
                                                                        toProc = null;
                                                                }
                                                        } else {

        dsProcessBlock(Arrays.copyOfRange(buffer, 0, dataRead + shift), dataRead + shift, null , -1);
                                                        }
                                                }
                                        }
                                }
                                dsProcFile.sendEmptyMessage(0);
                        } while ((dataRead != -1) & (! canceled));
                        if (toProc != null){
                                dsProcessBlock(toProc, toProc.length, null , -1);
                        }
                        fIn.close();
                } catch (FileNotFoundException e) {
                        cancel();
                        e.printStackTrace();
                } catch (IOException e) {
                        cancel();
                        e.printStackTrace();
                }
        }

        private void dsProcessBlock(byte[] block, int blkSize, byte[] clearBlockHash2, long rollingHash2) {
                if (! canceled){
                        byte[] clearBlockHash = null;
                        byte[] encryptedBlockHash = null;
                        long rollingHash = -1;
                        if (clearBlockHash2 != null){
                                clearBlockHash = Arrays.copyOfRange(clearBlockHash2, 0,
clearBlockHash2.length);
                        } else {
                                timeCtrl.startTimer(TimeControl.CLEAR_HASH_TIMER);
                                clearBlockHash = getBlockHash (block,blkSize);
                                timeCtrl.stopTimer();
                        }
                        String hexDigest = toHex(clearBlockHash);
                        dsProcChuncks++;
                        dsProcSize += blkSize;
                        dsProcFile.sendEmptyMessage(0);
                        if (! dataSource.hashExists(hexDigest)){
                                if (blkSize == FIXEDBLOCKSIZE) {
                                        if (rollingHash2 > 0){
                                                rollingHash = rollingHash2;
                                        } else {
                                                RollingHash rHash = new RollingHash(FIXEDBLOCKSIZE, 0);
                                                rollingHash = rHash.getBlockHash(block);
                                        }
                                }
                                timeCtrl.startTimer(TimeControl.CRIPTO_TIMER);
                                byte[] encryptedBuffer  = encryptBlock(block,clearBlockHash,blkSize);
                                timeCtrl.startTimer(TimeControl.CRIPTO_HASH_TIMER);
```

```
                                encryptedBlockHash = getBlockHash
(encryptedBuffer,encryptedBuffer.length);
                                timeCtrl.startTimer(TimeControl.COMM_TIMER);
                                sendBlock(encryptedBuffer,encryptedBlockHash);
                                if (! canceled) {
                                        timeCtrl.startTimer(TimeControl.DBASE_TIMER);
                                        if (rollingHash < 0) {
                                                dataSource.addHash(hexDigest);

                                        } else {
                                                dataSource.add2Hash(hexDigest, rollingHash);
                                        }
                                }
                        } else {
                                dsDupChuncks++;
                        }
                        timeCtrl.stopTimer();
                }
        }

/*
 *      File block processing
 */

        private void sendBlock(byte[] encryptedBlock, byte[] encryptedBlockHash) {
                if (! canceled) {
                        try {
                                dsUpload += io.sendBlock(encryptedBlock, encryptedBlockHash);
                                dsProcUpload.sendEmptyMessage(0);

                        } catch (IOException e) {
                                cancel();
                                dsProcIOError.sendEmptyMessage(0);
                        }
                }
        }

        private byte[] encryptBlock(byte[] block, byte[] clearBlockHash, int blockSize) {
                byte[] bKey = new byte[KEYSIZE];
                byte[] bIV  = new byte[KEYSIZE];
                byte[] encryptedBlock = null;
                if (! canceled) {
                        System.arraycopy(clearBlockHash, 0, bKey, 0, KEYSIZE);
                        System.arraycopy(clearBlockHash,clearBlockHash.length-KEYSIZE, bIV,  0, KEYSIZE);

                        SecretKeySpec key = new SecretKeySpec(bKey, "AES");
                    IvParameterSpec ivSpec = new IvParameterSpec(bIV);
                    Cipher cipher;
                        try {
                                cipher = Cipher.getInstance("AES/CTR/NoPadding", "BC");
                            cipher.init(Cipher.ENCRYPT_MODE, key, ivSpec);
                            encryptedBlock = cipher.doFinal(block, 0, blockSize);
                        } catch (NoSuchAlgorithmException e) {
                                cancel();
                                e.printStackTrace();
                        } catch (NoSuchProviderException e) {
                                cancel();
                                e.printStackTrace();
                        } catch (NoSuchPaddingException e) {
                                cancel();
                                e.printStackTrace();
                        } catch (InvalidKeyException e) {
                                cancel();
                                e.printStackTrace();
                        } catch (InvalidAlgorithmParameterException e) {
                                cancel();
                                e.printStackTrace();
                        } catch (IllegalBlockSizeException e) {
                                cancel();
                                e.printStackTrace();
                        } catch (BadPaddingException e) {
                                cancel();
                                e.printStackTrace();
                        }
                }
                return encryptedBlock;
        }

        private byte[] getBlockHash(byte[] block, int blockSize) {
```

```
                MessageDigest hashBlock = null;
                try {
                        hashBlock = MessageDigest.getInstance("SHA1", "BC");
                } catch (NoSuchAlgorithmException e) {
                        cancel();
                        e.printStackTrace();
                } catch (NoSuchProviderException e) {
                        cancel();
                        e.printStackTrace();
                }
                hashBlock.update(block, 0, blockSize);
                return hashBlock.digest();
        }


/*
 * Message handlers
 */


        private static Handler dsProcFile = new Handler() {
        @Override
        public void handleMessage(Message msg) {
        mActivity.updateValue(SecDedupActivity.PROCFILES, dsProcFiles);
        mActivity.updateValue(SecDedupActivity.PROCCHUNCK, dsProcChuncks);
        mActivity.updateValue(SecDedupActivity.DUPCHUNCK, dsDupChuncks);
        mActivity.updateValue(SecDedupActivity.PROCSIZE, dsProcSize);
        }
        };

        private static Handler dsProcUpload = new Handler() {
        @Override
        public void handleMessage(Message msg) {
        mActivity.updateValue(SecDedupActivity.UPLOADED, dsUpload);
        }
        };

        private static Handler dsProcTerminated = new Handler() {
        @Override
        public void handleMessage(Message msg) {
        mActivity.terminateProcessing();
        }
        };

        private static Handler dsProcCanceled = new Handler() {
        @Override
        public void handleMessage(Message msg) {
        mActivity.cancelProcessing();
        }
        };

        private static Handler dsProcStarted = new Handler() {
        @Override
        public void handleMessage(Message msg) {
        mActivity.startProcessing();
        }
        };

        private static Handler dsProcIOError = new Handler() {
        @Override
        public void handleMessage(Message msg) {
        mActivity.showDialog(SecDedupActivity.IO_ERROR);
        }
        };

        public boolean isValid() {
                return dsPath.exists();
        }

        public void setIO(SecDedupIO sdIO) {
                this.io = sdIO;

        }

        public long[] getTimes() {
                return timeCtrl.getTimers();
        }
```

```
}
```

```
public class RollingHash {
        final static int RP = 370248451;
        final static int R = 256;
        private int blkSize;
        private long Z;
        private long h;
        private byte[] blk;
        private int blkPoiter;

        public RollingHash (int blockSize, int slideSize){
                blkSize = blockSize;
                blk = new byte[blockSize+slideSize];
                Z = 1;
                for (int i = 1; i <= blkSize -1; i++){
                        Z = (Z * R) % RP;
                }
        }

        public long getBlockHash (byte[] block){
                h = 0;
                for (int i = 0; i < blkSize; i++){
                        h = (h * R + ((int)block[i]&0xFF)) % RP;
                }
                System.arraycopy(block, 0, blk, 0, blk.length);
                blkPoiter=0;
                return h;
        }

        public long updateHash (){
                h = (h + RP - Z *  ((int)blk[blkPoiter]&0xFF) % RP) % RP;
                h = (h * R + ((int)blk[blkSize+blkPoiter]&0xFF)) % RP;
                blkPoiter++;
                return h;
        }

}
```

```
public interface SDProto {

        static final byte DEVICE_SIGN  = 1;
        static final byte DEVICE_ID    = 2;
        static final byte BHASH              = 3;
        static final byte FHASH              = 4;
        static final byte HASH_OK            = 5;
        static final byte HASH_DUP           = 6;
        static final byte FILE_CHUNCK  = 7;
        static final byte CHUNCK_OK    = 8;
        static final byte CHUNCK_ERROR = 9;
        static final byte FILE_OK      = 10;
        static final byte FILE_ERROR   = 11;
        static final byte BLOCK              = 12;
        static final byte BLOCK_OK           = 13;
        static final byte BLOCK_ERROR  = 14;
        static final byte RESULTS      = 97;
        static final byte RESULTS_OK   = 98;
        static final byte PROC_CANCEL  = 99;
        static final byte PROC_END     = 100;

}
```

```
import java.io.IOException;
import java.net.UnknownHostException;
import java.text.DecimalFormat;
import java.text.SimpleDateFormat;
import android.net.ConnectivityManager;
import android.net.NetworkInfo;
import android.net.wifi.WifiInfo;
import android.net.wifi.WifiManager;
import android.os.BatteryManager;
import android.os.Bundle;
import android.os.SystemClock;
```

```
import android.preference.PreferenceManager;
import android.app.Activity;
import android.app.AlertDialog;
import android.app.Dialog;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.DialogInterface;
import android.content.Intent;
import android.content.IntentFilter;
import android.content.SharedPreferences;
import android.telephony.TelephonyManager;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.Chronometer;
import android.widget.ProgressBar;
import android.widget.TextView;


public class SecDedupActivity extends Activity {

         protected static final int STARTPROC    = 0;
         protected static final int TOTALFILES   = 1;
         protected static final int PROCFILES    = 2;
         protected static final int PROCCHUNCK   = 3;
         protected static final int PROCSIZE     = 4;
         protected static final int DUPCHUNCK    = 5;
         protected static final int TOTALSIZE    = 6;
         protected static final int UPLOADED     = 7;
         protected static final int DATASET_PATH = 8;
         protected static final int SERVER_ADDR  = 9;

         static final int DATASET_NOT_PRESENT = 1;
         static final int BATTERY_CHARGING    = 5;
         static final int NO_NETWORK          = 6;
         static final int SERVER_NOT_FOUND    = 7;
         static final int IO_ERROR            = 8;
         static final int NO_SERVER                          = 9;

         private String FILES_DIR_NAME = "FILES1";
         private String serverAddress = null;
         private String runMode = "1";

         private int dedupLevel = 1;
         private int totalFiles = 0;
         private long totalSize = 0;
         private long procSize  = 0;
         private boolean processing = false;
         private boolean batCharging = false;
         private static int batLevel = -1;
         private String dsBaseDir;

         private DatasetProcess dsProc;
         private SecDedupIO io = null;
         private static Chronometer chrono;
         private SecDedupResults results;
         private SharedPreferences sPrefs;

         private TextView tvDedupLevel;
         private TextView tvStartTime;
         private TextView tvEndTime;
         private TextView tvTotalFiles;
         private TextView tvProcFiles;
         private TextView tvFilesPrct;
         private TextView tvChuncks;
         private TextView tvDupChuncks;
         private TextView tvChuncksPrct;
         private TextView tvTotalSize;
         private TextView tvTotalSizeUnit;
         private TextView tvUploaded;
         private TextView tvUploadedUnit;
         private TextView tvUploadPrct;
         private TextView tvCharging;
         private TextView tvBatStart;
         private TextView tvBatEnd;
         private TextView tvBatConsumption;
         private TextView tvDevice;
         private TextView tvOSVersion;
         private TextView tvDedupType;
```

```
        private TextView tvNetworkType;
        private TextView tvStart;
        private TextView tvServerAddr;
        private ProgressBar pbProcessed;
        TextView[] leds;

        @Override
        protected void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                setContentView(R.layout.secdedup_activity);
                sPrefs = PreferenceManager.getDefaultSharedPreferences(getBaseContext());
                sPrefs.registerOnSharedPreferenceChangeListener(prefListener);
                buildUI();

                chrono.setText(getResources().getString(R.string.strZeroTime));
                results = new SecDedupResults();
                registerReceiver(batInfoReceiver, new IntentFilter(Intent.ACTION_BATTERY_CHANGED));
                getNetworkType();

        new DataSetLocate(this,FILES_DIR_NAME).start();

        }

        private void buildUI() {
                tvDedupLevel     = (TextView)findViewById(R.id.tvDedupLevel);
                tvStartTime      = (TextView)findViewById(R.id.tvStartTime);
                tvEndTime        = (TextView)findViewById(R.id.tvEndTime);
                chrono           = (Chronometer)findViewById(R.id.chChrono);
                tvTotalFiles     = (TextView)findViewById(R.id.tvTotalFiles);
                tvProcFiles      = (TextView)findViewById(R.id.tvProcFiles);
                tvFilesPrct      = (TextView)findViewById(R.id.tvFilesPrct);
                tvChuncks        = (TextView)findViewById(R.id.tvChuncks);
                tvDupChuncks     = (TextView)findViewById(R.id.tvDupChuncks);
                tvChuncksPrct    = (TextView)findViewById(R.id.tvChuncksPrct);
                tvTotalSize      = (TextView)findViewById(R.id.tvTotalSize);
                tvTotalSizeUnit  = (TextView)findViewById(R.id.tvTotalSizeUnit);
                tvUploaded       = (TextView)findViewById(R.id.tvUploaded);
                tvUploadedUnit   = (TextView)findViewById(R.id.tvUploadedUnit);
                tvUploadPrct     = (TextView)findViewById(R.id.tvUploadPrct);
                tvCharging       = (TextView)findViewById(R.id.tvCharging);
                tvBatStart       = (TextView)findViewById(R.id.tvBatStart);
                tvBatEnd         = (TextView)findViewById(R.id.tvBatEnd);
                tvBatConsumption = (TextView)findViewById(R.id.tvBatConsumption);
                tvDevice         = (TextView)findViewById(R.id.tvDevice);
                tvOSVersion      = (TextView)findViewById(R.id.tvOSVersion);
                tvDedupType      = (TextView)findViewById(R.id.tvDedupType);
                tvNetworkType     = (TextView)findViewById(R.id.tvNetworkType);
                tvServerAddr      = (TextView)findViewById(R.id.tvServer);
                tvStart          = (TextView)findViewById(R.id.tvStart);
                pbProcessed                = (ProgressBar)findViewById(R.id.pbProcessed);

                pbProcessed.setProgress(0);
                if (android.os.Build.DEVICE.equals(android.os.Build.MODEL)) {
                        tvDevice.setText(android.os.Build.DEVICE);
                } else {
                        tvDevice.setText(android.os.Build.DEVICE + " - " + android.os.Build.MODEL);
                }

                tvOSVersion.setText(android.os.Build.VERSION.RELEASE);
                updateStr(SERVER_ADDR, sPrefs.getString("prefServer",
getResources().getString(R.string.strNoServer)));
                runMode = sPrefs.getString("prefRunMode", "1");
                updateDedupType();
                tvStart.setVisibility(View.INVISIBLE);

                leds    = new TextView[TimeControl.NR_TIMERS];
                leds[0] = (TextView)findViewById(R.id.led1);
                leds[1] = (TextView)findViewById(R.id.led2);
                leds[2] = (TextView)findViewById(R.id.led3);
                leds[3] = (TextView)findViewById(R.id.led4);
                leds[4] = (TextView)findViewById(R.id.led5);
        }

        @Override
        public boolean onCreateOptionsMenu(Menu menu) {
                getMenuInflater().inflate(R.menu.sec_dedup, menu);
                return true;
        }
```

```
        @Override
    public boolean onOptionsItemSelected(MenuItem item) {
                if (! processing) {
                switch (item.getItemId()) {
                case R.id.secdedup_settings:
                    Intent i = new Intent(this, SecDedupSettings.class);
                    startActivity(i);
                    break;
                }
                 }
        return true;
    }

        SharedPreferences.OnSharedPreferenceChangeListener prefListener =  new
SharedPreferences.OnSharedPreferenceChangeListener() {
            public void onSharedPreferenceChanged(SharedPreferences prefs,String key) {
                        if (key.equals("prefServer")){
                                updateStr(SERVER_ADDR, sPrefs.getString("prefServer",
getResources().getString(R.string.strNoServer)));
                }
                        if (key.equals("prefRunMode")) {
                                runMode = sPrefs.getString("prefRunMode", "1");
                                updateDedupType();
                        }
            }
        };

        private BroadcastReceiver batInfoReceiver = new BroadcastReceiver(){
                @Override
                public void onReceive(Context context, Intent i) {
                        int level = i.getIntExtra(BatteryManager.EXTRA_LEVEL, -1);
                        int scale = i.getIntExtra(BatteryManager.EXTRA_SCALE, -1);
                        int batStatus = i.getIntExtra(BatteryManager.EXTRA_STATUS, -1);
                        batCharging = (batStatus == BatteryManager.BATTERY_STATUS_CHARGING);
                        if (batCharging) {
                                tvCharging.setVisibility(View.VISIBLE);
                        } else {
                                tvCharging.setVisibility(View.INVISIBLE);
                        }
                        int batteryPct = (Math.round((level  / (float)scale)*100));
                if (batteryPct != batLevel) {
                batLevel = batteryPct;
                if  (processing){
                        tvBatEnd.setText(String.valueOf(batteryPct));
                        long runSecs = Math.round((float)(SystemClock.elapsedRealtime() -
chrono.getBase()) / (float) 1000);
                                results.addBatStatus(batLevel, runSecs);
                                int batIni = Integer.parseInt(tvBatStart.getText().toString());
                                tvBatConsumption.setText(String.valueOf(batIni-batLevel));
                        }
                        }
                }
        };

    public void onStartClick(View v){
        if (! processing) {
                dsProc = new DatasetProcess(this, dsBaseDir, FILES_DIR_NAME);
                if (validStatus()) {
                processing = ! processing;
                tvStart.setBackgroundColor(getResources().getColor(R.color.bakgroundColor2));
                tvStart.setTextColor(getResources().getColor(R.color.foregroundColor2));
                tvStart.setText(R.string.strCancel);
                dsProc.start();
                }
        } else {
                if (dsProc.isAlive()){
                        dsProc.cancel();
                        chrono.stop();
                }
                processing = ! processing;
                tvStart.setBackgroundColor(getResources().getColor(R.color.bakgroundColor1));
                tvStart.setTextColor(getResources().getColor(R.color.foregroundColor1));
                tvStart.setText(R.string.strStart);
        }
    }

        public void onLevelClick (View v){
        if (! processing) {
        dedupLevel++;
```

```
        if (dedupLevel > 3) {
                dedupLevel = 1;
        }
        results.setDedupLevel(dedupLevel);
        tvDedupLevel.setText(String.valueOf(dedupLevel));
        updateDedupType();
        }
    }

        private void updateDedupType() {
        String strDedupType;
        String strRunMode;
                if (dedupLevel == 1) {
                        strDedupType = getResources().getString(R.string.strDedup1);
        } else if (dedupLevel == 2) {
                strDedupType = getResources().getString(R.string.strDedup2);
                } else {
                        strDedupType = getResources().getString(R.string.strDedup3);
                }
                if (runMode.equals("1")) {
                        strRunMode = getResources().getString(R.string.strRunMode1);
                } else if (runMode.equals("2")) {
                        strRunMode = getResources().getString(R.string.strRunMode2);
                } else {
                        strRunMode = getResources().getString(R.string.strRunMode3);
                }
                tvDedupType.setText(strDedupType + " ("+ strRunMode + ")");
        }

        private boolean validStatus() {
                if ((serverAddress == null) | (serverAddress.equals(""))) {
                        showDialog(NO_SERVER);
                        return false;
                }
                ConnectivityManager cm = (ConnectivityManager)
getSystemService(Context.CONNECTIVITY_SERVICE);
                NetworkInfo netInfo = cm.getActiveNetworkInfo();
                if ((netInfo == null)  || (! netInfo.isConnected())) {
                        showDialog(NO_NETWORK);
                        return false;
                } else {
                        try {
                                io = new SecDedupIO(serverAddress);
                                if (io.getDeviceID((String)tvDevice.getText()+"-"+tvOSVersion.getText())
< 0){
                                        showDialog(IO_ERROR);
                                        return false;
                                } else {
                                        dsProc.setIO(io);
                                }
                        } catch (UnknownHostException e) {
                                showDialog(SERVER_NOT_FOUND);
                                return false;
                        } catch (IOException e) {
                                e.printStackTrace();
                                showDialog(IO_ERROR);
                                return false;
                        }
                }
                if (! dsProc.isValid()) {
                        showDialog(DATASET_NOT_PRESENT);
                        return false;
                }
                return true;
        }

        private void getNetworkType() {
                String conType = "";
                ConnectivityManager cm = (ConnectivityManager)
getSystemService(Context.CONNECTIVITY_SERVICE);
                NetworkInfo netInfo = cm.getActiveNetworkInfo();
                if ((netInfo != null) && (netInfo.isConnected())) {
                        if (netInfo.getType() == ConnectivityManager.TYPE_WIFI) {
                                WifiManager mainWifi =
(WifiManager)getSystemService(Context.WIFI_SERVICE);
                                WifiInfo wifiInfo = mainWifi.getConnectionInfo();
                                if(wifiInfo.getBSSID()!=null){
                                        int wifiSpeed=wifiInfo.getLinkSpeed();
                                        String units=WifiInfo.LINK_SPEED_UNITS;
```

```
                                        results.setNetworkType(1000+wifiSpeed);
                                        tvNetworkType.setText("WiFi ("+Integer.toString(wifiSpeed)
+units+")");
                                } else {
                                        results.setNetworkType(1000);
                                        tvNetworkType.setText("WiFi");
                                }

                        }
                        if(netInfo.getType() == ConnectivityManager.TYPE_MOBILE) {
                                int subType = netInfo.getSubtype();
                                results.setNetworkType(2000 + subType);
                                switch(subType){
                                case TelephonyManager.NETWORK_TYPE_1xRTT: // ~ 50-100 kbps
                                        conType = "1xRTT";
                                        break;
                                case TelephonyManager.NETWORK_TYPE_CDMA:  // ~ 14-64 kbps
                                        conType = "CDMA";
                                        break;
                        case TelephonyManager.NETWORK_TYPE_EDGE:  // ~ 50-100 kbps
                                        conType = "EDGE";
                                        break;
                        case TelephonyManager.NETWORK_TYPE_EVDO_0:// ~ 50-100 kbps
                                        conType = "EVDO 0";
                                        break;
                        case TelephonyManager.NETWORK_TYPE_EVDO_A:// ~ 600-1400 kbps
                                        conType = "EVDO A";
                                        break;
                        case TelephonyManager.NETWORK_TYPE_GPRS:  // ~ 100 kbps
                                        conType = "GPRS";
                                        break;
                        case TelephonyManager.NETWORK_TYPE_HSDPA: // ~ 2-14 Mbps
                                        conType = "HSDPA";
                                        break;
                        case TelephonyManager.NETWORK_TYPE_HSPA:  // ~ 700-1700 kbps
                                        conType = "HSPA";
                                        break;
                        case TelephonyManager.NETWORK_TYPE_HSUPA: // ~ 1-23 Mbps
                                        conType = "HSUPA";
                                        break;
                        case TelephonyManager.NETWORK_TYPE_UMTS:  // ~ 400-7000 kbps
                                        conType = "UMTS";
                                        break;
                        case TelephonyManager.NETWORK_TYPE_EVDO_B:// ~ 5 Mbps
                                        conType = "EVDO B";
                                        break;
                        case TelephonyManager.NETWORK_TYPE_IDEN:  // ~25 kbps
                                        conType = "IDEN";
                                        break;
/*                      case TelephonyManager.NETWORK_TYPE_EHRPD: // ~ 1-2 Mbps
                                        conType = "EHRPD";
                                        break;
                        case TelephonyManager.NETWORK_TYPE_LTE:   // ~ 10+ Mbps
                                        conType = "LTE";
                                        break;
                        case TelephonyManager.NETWORK_TYPE_HSPAP: // ~ 10-20 Mbps
                            conType = "HSPAP";
                                        break;*/
                                }
                             tvNetworkType.setText("Mobile ("+conType+")");
                             }
                }
        }

    private String humanReadeble(float value){
        int i = 0;
        DecimalFormat df = new DecimalFormat("###,###.00");
        String[] valUnit = {"B","KB","MB","GB","TB"};
        while (value > 1024){
                value = value / 1024;
                i++;
        }
                return "("+df.format(value)+valUnit[i]+")";
    }

    public void updateStr (int str2update, String str){
        if (str2update == SERVER_ADDR) {
                tvServerAddr.setText(str);
                serverAddress = str;
```

```
	}
}

	public void updateValue(int val2update, long value) {
		DecimalFormat intFormat = new DecimalFormat("###,###");
		if (val2update == TOTALFILES) {
			tvTotalFiles.setText(intFormat.format(value));
			totalFiles = (int) value;
		} else if (val2update == TOTALSIZE) {
			tvTotalSize.setText(intFormat.format(value));
			tvTotalSizeUnit.setText(humanReadeble(value));
			totalSize = value;
			results.setTotalSize(value);
		} else if (val2update == PROCFILES){
			tvProcFiles.setText(intFormat.format(value));
			tvFilesPrct.setText(intFormat.format(value*100/totalFiles)+"%");
			results.setProcFiles((int)value);
		} else if (val2update == PROCCHUNCK){
			tvChuncks.setText(intFormat.format(value));
			results.setProcChuncks((int)value);
		} else if (val2update == DUPCHUNCK) {
			tvDupChuncks.setText(intFormat.format(value));
			if (results.getProcChuncks() != 0) {

	tvChuncksPrct.setText(intFormat.format(value*100/results.getProcChuncks())+"%");
			}
			results.setDupChuncks((int)value);
		} else if (val2update == PROCSIZE){
			procSize = value;
			pbProcessed.setProgress((int)(value*100/totalSize));
		} else if (val2update == UPLOADED) {
			tvUploaded.setText(intFormat.format(value));
			tvUploadedUnit.setText(humanReadeble(value));
			if (procSize != 0){
				tvUploadPrct.setText(intFormat.format(value*100/procSize)+"%");
			} else {
				tvUploadPrct.setText("0%");
			}
			results.setTotalUpload(value);
		}

	}

	public void startProcessing(){
		tvStartTime.setText(new SimpleDateFormat("HH:mm:ss").format(System.currentTimeMillis()));
		tvEndTime.setText("");
		tvBatStart.setText(String.valueOf(batLevel));
		tvBatEnd.setText("");
		tvBatConsumption.setText("");
		results.resetBatStatus();
		results.addBatStatus(batLevel, 0);
		chrono.setBase(SystemClock.elapsedRealtime());
		chrono.start();
	}

	public void terminateProcessing(){
		results.setTotalTime(SystemClock.elapsedRealtime()-chrono.getBase());
		chrono.stop();
		tvEndTime.setText(new SimpleDateFormat("HH:mm:ss").format(System.currentTimeMillis()));
		processing = false;
		tvStart.setBackgroundColor(getResources().getColor(R.color.bakgroundColor1));
		tvStart.setTextColor(getResources().getColor(R.color.foregroundColor1));
		tvStart.setText(R.string.strStart);
		try {
			io.sendResults(results, dsProc.getTimes());
		} catch (IOException e) {
			showDialog(IO_ERROR);
		}
	}

	public void cancelProcessing(){
		chrono.stop();
		processing = false;
		tvStart.setBackgroundColor(getResources().getColor(R.color.bakgroundColor1));
		tvStart.setTextColor(getResources().getColor(R.color.foregroundColor1));
		tvStart.setText(R.string.strStart);
	}
```

```
        public int getDedupLevel() {
                return dedupLevel;
        }

        protected Dialog onCreateDialog(int id) {
        switch (id) {
        case DATASET_NOT_PRESENT:
        return new AlertDialog.Builder(SecDedupActivity.this)
                .setIcon(android.R.drawable.ic_dialog_alert)
                .setTitle(R.string.msg_DatasetNotPresent)
                .setPositiveButton("Ok", new DialogInterface.OnClickListener() {
                    public void onClick(DialogInterface dialog, int whichButton) {
                    }
                })
                .create();
        case BATTERY_CHARGING:
        return new AlertDialog.Builder(SecDedupActivity.this)
                        .setIcon(android.R.drawable.ic_dialog_alert)
                        .setTitle(R.string.msg_BatCharging)
                        .setPositiveButton("Ok", new DialogInterface.OnClickListener() {
                                public void onClick(DialogInterface dialog, int whichButton) {
                                }
                        })
                        .create();
        case NO_NETWORK:
        return new AlertDialog.Builder(SecDedupActivity.this)
                        .setIcon(android.R.drawable.ic_dialog_alert)
                        .setTitle(R.string.msg_NoNetwork)
                        .setPositiveButton("Ok", new DialogInterface.OnClickListener() {
                                public void onClick(DialogInterface dialog, int whichButton) {
                                }
                        })
                        .create();
        case SERVER_NOT_FOUND:
        return new AlertDialog.Builder(SecDedupActivity.this)
                        .setIcon(android.R.drawable.ic_dialog_alert)
                        .setTitle(R.string.msg_ServerNotFound)
                        .setPositiveButton("Ok", new DialogInterface.OnClickListener() {
                                public void onClick(DialogInterface dialog, int whichButton) {
                                }
                        })
                        .create();
        case IO_ERROR:
        return new AlertDialog.Builder(SecDedupActivity.this)
                        .setIcon(android.R.drawable.ic_dialog_alert)
                        .setTitle(R.string.msg_IO_Error)
                        .setPositiveButton("Ok", new DialogInterface.OnClickListener() {
                                public void onClick(DialogInterface dialog, int whichButton) {
                                }
                        })
                        .create();
        case NO_SERVER:
        return new AlertDialog.Builder(SecDedupActivity.this)
                        .setIcon(android.R.drawable.ic_dialog_alert)
                        .setTitle(R.string.msg_NoServer)
                        .setPositiveButton("Ok", new DialogInterface.OnClickListener() {
                                public void onClick(DialogInterface dialog, int whichButton) {
                                }
                        })
                        .create();
        }
        return null;
}

        void ledsCtrl(int ledNr){
                for (int i=0; i < TimeControl.NR_TIMERS; i++){
                        leds[i].setVisibility(View.INVISIBLE);
                }
                if ((ledNr >= 0) & (ledNr < TimeControl.NR_TIMERS)){
                                leds[ledNr].setVisibility(View.VISIBLE);
                }
        }

        public void set_dsBaseDir(String dsBaseDir) {
                this.dsBaseDir = dsBaseDir;
        }

        public void showStart() {
                tvStart.setVisibility(View.VISIBLE);
```

```
        }

        public String getServerAddress() {
                return serverAddress;
        }
}
```

```
import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.SQLException;
import android.database.sqlite.SQLiteDatabase;

public class SecDedupDataSource {

        private SQLiteDatabase db;
        private SQLDatabase dbHelper;
        private boolean isOpen;

        public SecDedupDataSource (Context context, String dbPath){
                isOpen = false;
                dbHelper = new SQLDatabase(context, dbPath);
        }

        public void open() throws SQLException {
                db = dbHelper.getWritableDatabase();
                isOpen = true;
        }

        public void reset() throws SQLException {
                db = dbHelper.getWritableDatabase();
                dbHelper.clearDatabase(db);
                isOpen = true;
        }

        public void close() {
                dbHelper.close();
                isOpen = false;
        }

        public void addHash (String hash){
                ContentValues values = new ContentValues();
                values.put(SQLDatabase.COLUMN_SESSION, 1);
                values.put(SQLDatabase.COLUMN_HASH, hash);
                db.insert(SQLDatabase.TABLE_HASHS, null, values);
        }

        public boolean hashExists (String hash) {
                if (isOpen) {
                        String[] Columns = {SQLDatabase.COLUMN_HASH};
                        String[] Values  = {hash};
                        Cursor c = db.query(SQLDatabase.TABLE_HASHS, Columns,
SQLDatabase.COLUMN_HASH+"=?", Values, null, null, null);
                        int rows = c.getCount();
                        c.close();
                        return rows > 0;
                } else {
                        return false;
                }
        }

        public boolean rolHashExists (long rolHash) {
                String[] Columns = {SQLDatabase.COLUMN_ROLHASH};
                String[] Values  = {String.valueOf(rolHash)};
                Cursor c = db.query(SQLDatabase.TABLE_HASHS, Columns, SQLDatabase.COLUMN_ROLHASH+"=?",
Values, null, null, null);
                int rows = c.getCount();
                c.close();
                return rows > 0;
        }

        public boolean isOpen() {
                return isOpen;
        }

        public void add2Hash(String hash, long rolHash) {
                ContentValues values = new ContentValues();
```

```
                values.put(SQLDatabase.COLUMN_SESSION, 1);
                values.put(SQLDatabase.COLUMN_HASH, hash);
                values.put(SQLDatabase.COLUMN_ROLHASH, String.valueOf(rolHash));
                db.insert(SQLDatabase.TABLE_HASHS, null, values);
        }

}
```

```
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.net.Socket;
import java.net.UnknownHostException;

public class SecDedupIO {

         private static final int TCP_PORT = 9999;
         private Socket s;
         private InputStream in;
         private DataInputStream dis;
         private OutputStream out;
         private DataOutputStream dos;
         private int deviceID = -1;

         public SecDedupIO(String serverAddress) throws UnknownHostException, IOException{
                 s = new Socket(serverAddress, TCP_PORT);
                 in = s.getInputStream();
                 dis = new DataInputStream(in);
                 out = s.getOutputStream();
                 dos = new DataOutputStream(out);
         }

         public int getDeviceID(String deviceSign) throws IOException {
                 byte bBufer;
                 dos.writeByte(SDProto.DEVICE_SIGN);
                 dos.writeInt(deviceSign.length());
                 dos.writeBytes(deviceSign);
                 dos.flush();
                 bBufer = dis.readByte();
                 if (bBufer == SDProto.DEVICE_ID){
                         deviceID = dis.readInt();
                         return deviceID;
                 }
                 return -1;
         }

         public int sendBlock (byte[] encryptedBlock, byte[] blkHash) throws IOException{
                 int uploadSize = 0;
                 byte bBufer = 0;
                 dos.writeByte(SDProto.BHASH);
                 dos.writeInt(deviceID);
                 dos.writeInt(blkHash.length);
                 dos.write(blkHash);
                 dos.flush();
                 bBufer = dis.readByte();
                 uploadSize = blkHash.length + 9;
                 if (bBufer == SDProto.HASH_OK){
                         dos.writeByte(SDProto.BLOCK);
                         dos.writeInt(deviceID);
                         dos.writeInt(encryptedBlock.length);
                         dos.write(encryptedBlock);
                         dos.flush();
                         bBufer = dis.readByte();
                         uploadSize += (encryptedBlock.length + 9);
                 }
                 return uploadSize;
         }

         public long sendFile (File encryptedFile, byte[] fileHash) throws IOException{
                 InputStream fIn;
                 byte[] buffer = new byte[DatasetProcess.FBUFFERSIZE];
                 long uploadSize = 0;
                 byte bBufer = 0;
```

```
            int dataRead;
            dos.writeByte(SDProto.FHASH);
                dos.writeInt(deviceID);
                dos.writeInt(fileHash.length);
                dos.writeInt(DatasetProcess.FBUFFERSIZE);
                dos.writeLong(encryptedFile.length());
                dos.write(fileHash);
                dos.flush();
                bBufer = dis.readByte();
                uploadSize = fileHash.length + 21;
                if (bBufer == SDProto.HASH_OK){
                        fIn = new FileInputStream(encryptedFile);
                        while ((dataRead = fIn.read(buffer)) != -1) {  // CANCELAR
                                dos.writeByte(SDProto.FILE_CHUNCK);
                                dos.writeInt(deviceID);
                                dos.writeInt(dataRead);
                                dos.write(buffer, 0, dataRead);
                                dos.flush();
                                bBufer = dis.readByte(); // CHUNCK_OK
                                uploadSize += (dataRead + 9);
                        }
                    fIn.close();
                    bBufer = dis.readByte(); // FILE_OK
                }
                return uploadSize;
        }

        public void sendResults(SecDedupResults results, long[] timers) throws IOException {
                byte bBufer = 0;
            dos.writeByte(SDProto.RESULTS);
                dos.writeInt(deviceID);
                dos.writeLong(results.getTotalTime());
                dos.writeLong(results.getTotalSize());
                dos.writeInt(results.getProcFiles());
                dos.writeInt(results.getProcChuncks());
                dos.writeInt(results.getDupChuncks());
                dos.writeLong(results.getTotalUpload());
                dos.writeInt(results.getDedupLevel());
                dos.writeInt(results.getNetworkType());
                dos.writeByte(timers.length);
                for (int i = 0; i < timers.length; i++){
                        dos.writeLong(timers[i]);
                }
                int b = results.getBatSatusCount();
                dos.writeByte(b);
                for (int i = 0; i < b; i++){
                        BatteryStatus bs = results.getBatStatus(i);
                        dos.writeInt(bs.getBatteryLevel());
                        dos.writeLong(bs.getElapsedTime());
                }
                dos.flush();
                bBufer = dis.readByte();
                if (bBufer == SDProto.RESULTS_OK){
                        dos.writeByte(SDProto.PROC_END);
                        dos.flush();
                        out.close();
                        in.close();
                        s.close();
                }
        }

        public void procCancel() {
                try {
                        dos.writeByte(SDProto.PROC_CANCEL);
                        dos.flush();
                        out.close();
                        in.close();
                        s.close();
                } catch (IOException e) {
                }
        }

}
```

```
import java.util.ArrayList;
```

```
public class SecDedupResults {

	private long totalTime;
	private int procFiles;
	private int procChuncks;
	private int dupChuncks;
	private long totalSize;
	private long totalUpload;
	private int dedupLevel;
	private int networkType;
	private ArrayList<BatteryStatus> batStatList;

	public SecDedupResults() {
		batStatList = new ArrayList<BatteryStatus>();
		this.dedupLevel = 1;
	}

	public long getTotalTime() {
		return totalTime;
	}

	public void setTotalTime(long totalTime) {
		this.totalTime = totalTime;
	}

	public int getProcFiles() {
		return procFiles;
	}

	public void setProcFiles(int procFiles) {
		this.procFiles = procFiles;
	}

	public int getProcChuncks() {
		return procChuncks;
	}

	public void setProcChuncks(int procChuncks) {
		this.procChuncks = procChuncks;
	}

	public int getDupChuncks() {
		return dupChuncks;
	}

	public void setDupChuncks(int dupChuncks) {
		this.dupChuncks = dupChuncks;
	}

	public long getTotalSize() {
		return totalSize;
	}

	public void setTotalSize(long totalSize) {
		this.totalSize = totalSize;
	}

	public long getTotalUpload() {
		return totalUpload;
	}

	public void setTotalUpload(long totalUpload) {
		this.totalUpload = totalUpload;
	}

	public int getDedupLevel() {
		return dedupLevel;
	}

	public void setDedupLevel(int dedupType) {
		this.dedupLevel = dedupType;
	}

	public int getNetworkType() {
		return networkType;
	}

	public void setNetworkType(int networkType) {
		this.networkType = networkType;
```

```
        }

        public void addBatStatus (int batLevel, long runSecs){
                batStatList.add(new BatteryStatus(batLevel, runSecs));
        }

        public void resetBatStatus () {
                if (batStatList.size() > 0){
                        batStatList.clear();
                }
        }

        public int getBatSatusCount(){
                return batStatList.size();
        }

        public BatteryStatus getBatStatus(int i) {
                return batStatList.get(i);
        }

}
```

```
import android.content.Context;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;

public class SQLDatabase extends SQLiteOpenHelper {


        private static final String DATABASE_NAME    = "SecDedup.db";
        private static final int    DATABASE_VERSION = 1;

        public static final String TABLE_HASHS    = "hashs";
        public static final String COLUMN_ID      = "_id";
        public static final String COLUMN_SESSION = "session";
        public static final String COLUMN_HASH    = "hash";
        public static final String COLUMN_ROLHASH = "rolling_hash";
        public static final String INDEX_HASH     = "hash_idx";
        public static final String INDEX_ROLHASH  = "rolhash_idx";

        private static final String DATABASE_CREATE  = "create table "
                    + TABLE_HASHS + "("
                        + COLUMN_ID + " integer primary key autoincrement, "
                        + COLUMN_SESSION + " integer not null, "
                    + COLUMN_HASH + " text not null, "
                        + COLUMN_ROLHASH + " integer);";
        private static final String CREATE_INDEX_HASH = "create unique index "
                        + INDEX_HASH + " ON "
                        + TABLE_HASHS + "(" + COLUMN_HASH + ")";
        private static final String CREATE_INDEX_ROLHASH = "create index "
                        + INDEX_ROLHASH + " ON "
                        + TABLE_HASHS + "(" + COLUMN_ROLHASH + ")";

        public SQLDatabase(Context context, String dbPath) {
                super(context, DATABASE_NAME, null, DATABASE_VERSION);
        }


        @Override
        public void onCreate(SQLiteDatabase db) {
                db.execSQL(DATABASE_CREATE);
                db.execSQL(CREATE_INDEX_HASH);
                db.execSQL(CREATE_INDEX_ROLHASH);
        }


        @Override
        public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
                db.execSQL("DROP INDEX IF EXISTS " + INDEX_ROLHASH);
                db.execSQL("DROP INDEX IF EXISTS " + INDEX_HASH);
            db.execSQL("DROP TABLE IF EXISTS " + TABLE_HASHS);
            onCreate(db);
        }

        public void clearDatabase (SQLiteDatabase db) {
                db.execSQL("DELETE FROM " + TABLE_HASHS + " WHERE "+COLUMN_SESSION+"=1");
        }
```

```
}
```

```
import java.util.Arrays;
import android.os.Handler;
import android.os.Message;
import android.os.SystemClock;

public class TimeControl {

	static final int CLEAR_HASH_TIMER  = 1;
	static final int CRIPTO_TIMER      = 2;
	static final int CRIPTO_HASH_TIMER = 3;
	static final int DBASE_TIMER       = 4;
	static final int COMM_TIMER        = 5;

	static final int NR_TIMERS = 5;
	private static SecDedupActivity mActivity;


	private long[] timers;
	private static int activeTimer;

	public TimeControl(SecDedupActivity mainActivity) {
		TimeControl.mActivity    = mainActivity;
		timers = new long[NR_TIMERS+1];
		resetTimers();
	}

	public void startTimer(int nrTimer){
		if (activeTimer != 0) {
			stopTimer();
		}
		long newTime = SystemClock.elapsedRealtime();
		timers[0] = newTime;
		activeTimer = nrTimer;
		tcLedOn.sendEmptyMessage(0);
	}

	public void stopTimer()  {
		if (activeTimer != 0) {
			long newTime = SystemClock.elapsedRealtime();
			timers[activeTimer] += (newTime - timers[0]);
		}
		timers[0] = 0;
		activeTimer = 0;
		tcLedOff.sendEmptyMessage(0);

	}

	public void resetTimers () {
		activeTimer = 0;
		for (int i=0; i <= NR_TIMERS; i++){
			timers[i] = 0;
		}
		tcLedOff.sendEmptyMessage(0);
	}

	public long[] getTimers(){
		return Arrays.copyOfRange(timers, 1, NR_TIMERS+1);
	}

	private static Handler tcLedOn = new Handler() {
	@Override
	public void handleMessage(Message msg) {
	mActivity.ledsCtrl(activeTimer-1);
	}
	};

	private static Handler tcLedOff = new Handler() {
	@Override
	public void handleMessage(Message msg) {
	mActivity.ledsCtrl(activeTimer-1);
	}
	};

}
```