Estudo da Viabilidade da Compressão de Dados em Rádios Definidos por Software

Paulo R. R. Dayrell[2], Diego G. Passos[2], Fernanda G. O. Passos[1]

[1]Departamento de Engenharia de Telecomunicações
Universidade Federal Fluminense (UFF)
Niterói, RJ - Brasil

fernanda@midiacom.uff.br

[2]Instituto de Computação
Universidade Federal Fluminense (UFF)
Niterói, RJ - Brasil

paulodayrell@midiacom.uff.br, dpasssos@ic.uff.br

Abstract. *Software-defined radios — or SDR — bring a series of benefits such as fast prototyping of new wireless communication techniques. One of the open questions in using these devices is how to facilitate the transfer, processing, and storage of samples provided by an SDR. This paper aims to study the feasibility of implementing a lossless data compression algorithm in the internal FPGA (Field Programmable Gate Array)commonly found on SDR in order to increase the sample throughput between the radio and the host processor, as well as reducing the storage space. The results show that it is possible to obtain an average compression ratio of 1.55 times, considering the limited capacity of existing SDR to compute such algorithms.*

Resumo. *Os Rádios Definidos por Software — ou SDR — trazem uma série de benefícios como a rápida prototipagem de novas técnicas de comunicação sem fio. Uma das questões em aberto na utilização destes equipamentos é como possibilitar a transferência, processamento e armazenamento das amostras fornecidas por um SDR. O objetivo deste trabalho é estudar a viabilidade de implementar um algoritmo de compressão de dados sem perdas na FPGA (Field Programmable Gate Array) comumente encontrada nas SDR com o objetivo de aumentar a taxa de transmissão de envio das amostras entre o rádio e o processador do hospedeiro, assim como reduzir o espaço de armazenamento. Os resultados mostram que é possível obter uma taxa de compressão de 1,55 vezes considerando a capacidade limitada de SDR existentes para computar tais algoritmos.*

1. Introdução

Historicamente, equipamentos de telecomunicações têm passado por constantes evoluções tecnológicas especialmente em relação à implementação de suas funcionalidades. Cada vez mais, grande parte das funcionalidades de comunicação em redes vem sendo implementada em processadores de propósito geral, ao invés de em *hardwares* dedicados. A exemplo, temos as populares interfaces de Ethernet e Wi-Fi, que seguem

este caminho executando parte de suas funcionalidades através de um *firmware*, tipo de *software* que fornece controle de baixo nível para o *hardware* específico de um dispositivo [Linux Wireless 2015].

Mais recentemente, o processo de migração de *hardware* para *software* nas tecnologias de comunicação de dados atingiu um novo patamar com o advento dos chamados *rádios definidos por software* [Dillinger et al. 2005] (ou SDR). Estes equipamentos proveem um conjunto mínimo de funcionalidades básicas implementadas em *hardware* para a digitalização de um sinal analógico, mas permitem que várias funcionalidades historicamente implantadas em *hardware* sejam delegadas para o *software* [Dillinger et al. 2005]. Apenas as conversões dos sinais entre os domínios analógico e digital são realizadas pelo *hardware*, enquanto as demais tarefas, como a aplicação de filtros e a demodulação, são executadas por um processador de um computador hospedeiro — ao qual o SDR está conectado — ou mesmo através de uma FPGA (*Field Programmable Gate Array*), componente de lógica digital configurável embutido em alguns equipamentos.

Do ponto de vista de um pesquisador que estuda mecanismos e características da camada física, um possível caso de uso para os SDRs seria a criação de *traces*, *i.e.*, a captura de amostras do espectro em determinada faixa de frequência para uso posterior. Há uma série de possíveis usos para este tipo de *trace*, incluindo finalidades de análise do espectro [McHenry et al. 2006] (*e.g.*, detectar os níveis de ocupação de uma determinada faixa de frequência de interesse) e a execução de simulações comparativas de mecanismos em um mesmo cenário [Breslau et al. 2000].

A obtenção e armazenamento de tais *traces*, no entanto, não é trivial. Em primeiro lugar, as taxa de transmissão necessárias para o envio das amostras obtidas pelo rádio para o computador que irá armazená-las ou processá-las é bastante elevada. Apenas como um exemplo numérico, a recente emenda IEEE 802.11ac [IEEE LAN/MAN Standards Committee 2013] pode utilizar canais de até 160 MHz. Segundo o Teorema de Nyquist-Shannon [Shannon 1949], para reproduzir perfeitamente as condições de um canal com esta largura de banda, é necessária uma taxa de amostragem de, ao menos, 320 milhões de amostras por segundo. Com amostras que ocupam 28 bits (como as geradas pelo modelo N200 da Ettus [Ettus Research 2019], por exemplo), são necessários 8,96 Gb/s de banda de comunicação entre o SDR e o hospedeiro. Nesta taxa, um *trace* de apenas 60 segundos ocuparia 67,2 GB de espaço de armazenamento.

O problema de armazenar, manipular e transmitir grandes volumes de dados não é novo em computação. Conteúdos multimídia, por exemplo, especialmente nos altos níveis de qualidade utilizados hoje, resultam em massas de dados consideráveis. Para viabilizar a utilização de aplicações multimídia, foi necessária a criação e ampla adoção de vários mecanismos de compressão de dados [Sayood 2012]. Tais métodos, que podem ser divididos em métodos com perda e sem perda de informação, possuem diferentes complexidades e resultam em diferentes níveis de compressão.

O propósito deste trabalho é justamente avaliar a viabilidade e a eficiência de comprimir os dados das amostras obtidas através de um SDR para reduzir a banda necessária entre o rádio e o processador (o que, indiretamente, aumenta a largura de banda máxima com a qual um SDR pode trabalhar, dado um canal de comunicação de capaci-

dade fixa entre o rádio e o processador do hospedeiro) e reduzir o espaço necessário para o armazenamento de *traces* destas amostras. Aspirando futuramente construir um compressor de amostras em um SDR especializado, o estudo é conduzido através da análise de simulações de um protótipo e alguns parâmetros são variados a fim de possibilitar um estudo criterioso das possíveis limitações da FPGA embutida nesse tipo de rádio. Para tal, o algoritmo de compressão sem perda LZW (Lempel-Ziv-Welch) [Welch 1984] foi o escolhido para ser implementado na FPGA. Este algoritmo foi escolhido por ser um algoritmo altamente coeso e que se adapta naturalmente a alterações dinâmicas nas características dos dados. Outros algoritmos, como o Algoritmo de Huffman, tradicionalmente precisam enviar metadados junto aos dados comprimidos para possibilitar a descompressão. No LZW também há a vantagem do dicionário ser construído de forma independente nos lados do transmissor e do receptor.

Para entender melhor o problema e as características de um SDR, a Seção 2 trata da definição do conceito de SDR e de uma descrição do problema de transmissão e armazenamento de amostras. Na Seção 3, alguns dos métodos clássicos de compressão de dados sem perda são apresentados, especialmente o LZW que é o foco deste trabalho. A Seção 4 destaca os experimentos realizados para se obter uma melhor análise da implementação do LZW no rádio. A Seção 5 relata o estado da arte da compressão de dados sem perdas em FPGAs para diversos propósitos. Por último, são apresentadas as conclusões e trabalhos futuros.

2. SDR e o Problema da Transmissão e Armazenamento de Amostras

Em uma arquitetura SDR tradicional, durante a recepção, o *hardware* é responsável primariamente pela amostragem e quantização do sinal analógico em uma determinada banda de frequência. O processamento efetivo deste sinal, incluindo operações como a aplicação de filtros e a demodulação, são executados por um *software*. Dependendo do equipamento e de certas decisões de projeto, este *software* pode ser executado pela CPU de um computador hospedeiro — ao qual o SDR está conectado — ou mesmo através de uma FPGA, presente em algumas implementações de SDRs [Ettus Research 2019]. O processo de transmissão é análogo, com o sinal a ser transmitido sendo formatado por *software* no domínio digital.

A grande vantagens dos rádios definidos por *software*, em comparação com implementações tradicionais majoritariamente baseadas em *hardware*, está na sua flexibilidade. Uma interface de rede Wi-Fi, por exemplo, tem uma série de parâmetros de funcionamento disponíveis para o usuário — *e.g.*, taxas de transmissão variadas, a utilização ou não do mecanismo RTS/CTS (*Request-To-Send*/*Clear-To-Send*), limiares de fragmentação —, mas todas estas funcionalidades são limitadas pelo escopo do que é oferecido pelo padrão IEEE 802.11 (ou, mais especificamente, pelos requisitos da certificação da Wi-Fi Alliance [Wi-Fi Alliance 2019]). É impossível, por exemplo, utilizar uma modulação ou um esquema de FEC (*Forward Error Correction*) diferente dos previstos no padrão. O padrão também limita as bandas de frequência nas quais os equipamentos podem operar. Os SDRs, no entanto, não impõem tais limitações (ao menos, não tão rigidamente), provendo total controle ao usuário sobre esquemas de modulação, códigos de correção de erros, temporizações, e tantos outros aspectos inerentes às camadas física e de enlace. Além disso, estes equipamentos são comumente capazes de operar em diversas faixas de frequência.

Embora usuários finais — principalmente os leigos — não tenham, necessariamente, interesse em manipular estes aspectos de baixo nível dos rádios, esta flexibilidade é interessante tanto para fabricantes, quanto para a comunidade científica. Do ponto de vista dos fabricantes, erros e problemas na implementação de seus produtos poderiam mais frequentemente ser consertados através de simples atualizações de *software*. Já os pesquisadores da área tirariam proveito de uma plataforma altamente flexível que permite a rápida prototipagem e experimentação com novas técnicas e soluções. Além do estudo de técnicas básicas, como modulações e códigos de correção de erros, pesquisas na área de rádios cognitivos [Mitola III and Maguire Jr 1999, Rondeau et al. 2004] também têm utilizado SDRs como base.

De fato, há hoje várias plataformas de *hardware* [Ettus Research 2019, VANU 2019, WARP Project 2019] e *software* [GNU Radio 2019] que dão suporte ao desenvolvimento de SDRs. Em alguns casos, *hardwares* comuns, como de placas de som, prontamente disponíveis na maior parte dos computadores modernos, são utilizados como *front-ends* para SDRs [Dascal et al. 2013]. Por outro lado, plataformas de *hardware* especializadas oferecem maior capacidade e mais funcionalidades, como FPGAs integradas para processamentos especializados.

As plataformas SDR especializadas, em particular, são capazes de realizar a amostragem dos sinais analógicos a grandes taxas e com grande resolução. O modelo N200 da Ettus [Ettus Research 2019], que será usado neste trabalho, é capaz de obter mais de 100 milhões de amostras por segundo. Cada amostra é constituída de dois componentes numéricos — I e Q — de até 14 bits de resolução cada, resultando em uma taxa bruta de dados de até 2,8 Gb/s a serem processados por *software*. O volume de dados gerado por este tipo de equipamento impõe severos requisitos de capacidade de processamento e armazenamento, além da própria transferência dos dados entre o rádio e o hospedeiro.

O rádio N200 da Ettus ainda apresenta uma FPGA integrada, modelo Spartan 3A-DSP 1800 da fabricante Xilinx, que possibilita tratar os dados recebidos pelo rádio diretamente do conversor analógico-digital. Com este dispositivo, o circuito lógico poderia ser reconfigurado sempre que necessário, tornando a implementação de camadas de baixo nível de transmissão e recepção de dados mais flexível e adaptável.

Como ferramenta de desenvolvimento de um SDR, o *software* GNU Radio pode ser instalado no hospedeiro. Este *software* permite integração imediata com o *hardware* do rádio através de *drivers* especialmente desenvolvidos para ele [Reis et al. 2012]. Através do GNU Radio e de uma conexão Ethernet entre o rádio e o computador, as amostras de radiofrequência podem ser coletadas pelo computador conforme descrito previamente.

3. Métodos de Compressão de Dados

Compressão de dados é o processo de converter um fluxo de dados de entrada F em outro fluxo F' de tamanho menor quando comparado aos dados originais. O processo de descompressão é o inverso: dado um fluxo de dados comprimido F', o algoritmo gera um fluxo de dados reconstruído R [Sayood 2002]. Tais processos são apresentados na Figura 1. De acordo com o esquema de reconstrução adotado, os algoritmos de compressão podem ser divididos em dois tipos: com perda, quando F e R são potencialmente diferentes, e sem perda, quando F e R são necessariamente iguais [Sayood 2012].

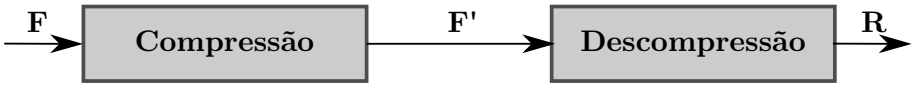

Figura 1. Fluxo de dados no processo de compressão e descompressão.

Algoritmos de compressão com perdas são comumente utilizados por aplicações tolerantes a perda de dados — como telefonia IP, *streaming* de vídeos e armazenamento de imagens/vídeos — e, em geral, são mais eficientes em termos de taxa de compressão. Alguns métodos populares de compressão com perdas incluem o JPEG (imagem), o MP3 (áudio) e o MPEG (vídeo). Já em aplicações nas quais o dado reconstruído deve ser exatamente igual ao original, como em compressão de textos e arquivos executáveis, o padrão é utilizar algoritmos de compressão sem perda.

Para a transmissão e armazenamento de amostras em um SDR, a compressão sem perdas é mais indicada porque ter acesso às amostras originais exatas pode ser um requisito importante para as diversas aplicações possibilitadas por esse tipo de dispositivo. Dentre os métodos de compressão sem perda, há os algoritmos que operam por aproximação de entropia e aqueles baseados em dicionários [Sayood 2002]. Os métodos de aproximação de entropia — ou estatísticos — calculam as probabilidades de ocorrência dos símbolos no fluxo de dados e geram códigos de acordo com esses valores.

O algoritmo de Huffman [Huffman 1952] é um exemplo de método de compressão sem perdas que faz uso da aproximação de entropia. A versão tradicional gera uma estrutura de árvore baseada na probabilidade dos símbolos no fluxo de dados e precisa utilizá-la igualmente tanto na compressão quanto na descompressão. Isso requer que, em uma transmissão, a árvore seja enviada para o receptor para que ele consiga recuperar os dados originais. No entanto, existem versões adaptativas que permitem construir a árvore no transmissor e receptor independentemente, sem a necessidade do envio desta estrutura de dados.

3.1. Algoritmo LZW

O algoritmo LZW [Welch 1984] é um exemplo de método de compressão sem perdas baseado em dicionário. Este algoritmo é simples e trabalha gerando um dicionário que associa os símbolos do fluxo de entrada de dados a um código geralmente de tamanho menor. O LZW é naturalmente adaptativo, pois tanto o compressor quanto o descompressor geram o mesmo dicionário independentemente, o primeiro para realizar a codificação, enquanto o segundo para a decodificação.

O LZW é simples, pois é definido em etapas resumidas em verificar a existência de um símbolo no dicionário, a adição de um novo elemento no dicionário e a geração de um código de acordo com uma entrada no dicionário. Para a implementação em uma FPGA, a simplicidade dos métodos a serem descritos em forma circuitos lógicos é fundamental para viabilizar a compressão de dados embutida no SDR. Na Seção 5, são apresentados alguns trabalhos que tratam da implementação de algoritmos de compressão em FPGAs.

Usualmente, o processo de construção do dicionário utilizado por algoritmos de compressão em geral é classificado como sendo: estático, semi-adaptativo ou adaptativo [Sayood 2002].

Quando construído estaticamente, o dicionário é preenchido por completo antes do

processo de compressão com subconjuntos do domínio da entrada. O método de seleção dos subconjuntos deve ser definido da mesma maneira no algoritmo de compressão e descompressão. Dicionários semi-adaptativos também são construídos por completo antes de iniciar a compressão. Os subconjuntos adicionados ao dicionário são selecionados por um processo de inspeção dos dados de entrada. Como os dados de entrada não estão disponíveis no processo de descompressão, o algoritmo precisa enviar de alguma maneira o dicionário utilizado pelo algoritmo de compressão para o algoritmo de descompressão.

Dicionários adaptativos são construídos no decorrer do processo de compressão e devem ser reconstruídos da mesma maneira no algoritmo de descompressão. Dicionários adaptativos costumam conter um sub-conjunto estático chamado *dicionário de arranque*, que por sua vez também possui um método próprio de construção, definido da mesma maneira nos algoritmos de compressão e descompressão.

O Algoritmo 1 mostra o processo de compressão do LZW. Antes de, de fato, começar o processo de compressão, o LZW inicializa dois atributos: uma variável w que é utilizada durante a compressão para o agrupamento de símbolos e é inicializada com símbolo vazio ϵ (Linha 2); e o dicionário adaptativo (Linha 1) conforme já mencionado. Após inicializados estes atribuídos, a partir da Linha 3, para cada palavra k lida do fluxo de dados de entrada F, na Linha 4 do Algoritmo 1 é verificado se a concatenação entre w e k já está presente no dicionário. Em caso positivo, o valor de w é atualizado para wk na Linha 5. Caso contrario, wk é inserido no dicionario, w recebe o valor de k e o código referente ao símbolo k no dicionário é escrito no fluxo de saída F' (Linhas 7, 8 e 9, respectivamente).

Algoritmo 1: Compressão LZW.

```
   Entrada: F
   Saída: F'
1  Inicializar o dicionário
2  w ← ϵ
3  Para cada k ∈ F faça
4      se wk ∈ dicionário então
5          w ← wk
6      senão
7          Inserir wk no dicionário
8          w ← k
9          c ← CODIGO(k)
10         Escrever c em F'
11     fim
12 fim
```

O Algoritmo 2 apresenta a descompressão do LZW. Assim como na compressão, o processo de descompressão começa com a inicialização do dicionário de arranque (Linha 1) e de uma variável w que é utilizada para o agrupamento de símbolos (Linha 2). Após estes atributos serem inicializados, o algoritmo lê a primeira palavra c' em F' e escreve em R o símbolo k' associado a c' no dicionário (Linhas 3,4 e 5). A partir da Linha 6, para cada palavra c lida dos dados de entrada restante o algoritmo verifica se c já está presente no dicionário (Linha 7). Caso esteja, uma variável s, também utilizada para

o agrupamento de símbolos, recebe o símbolo apontado por c (Linha 8). Caso contrário, s recebe a concatenação entre o símbolo de c' e w (Linhas 10 e 11). Logo em seguida, na Linha 13, o valor de s é escrito no fluxo de dados de saída R, w é atualizado com o primeiro símbolo da variável s (Linha 14) e c' atualizado com o valor c na Linha 15. Após as atualizações serem feitas, a concatenação entre k' e w é inserida no dicionário (Linha 16).

Algoritmo 2: Descompressão LZW.

Entrada: F'
Saída: R

```
1  Inicializar o dicionário
2  w ← ϵ
3  Ler c' em F'
4  k' ← SIMBOLO(c')
5  Escrever k' em R
6  Para cada c ∈ F' faça
7      se c ∈ dicionário então
8          s ← SIMBOLO(c)
9      senão
10         s ← SIMBOLO(c')
11         s ← sw
12     fim
13     Escrever s em R
14     w ← PRIMEIRO-SIMBOLO(s)
15     c' ← c
16     Inserir k'w no dicionário
17 fim
```

3.2. Implementação do LZW

O algoritmo especificado por Welch [Welch 1984] possui tamanho de palavra de entrada de 8 bits e utiliza 12 bits para representar os códigos associados às entradas do dicionário. Nessas condições, o dicionário possui o limite de 2^{12} possíveis valores. A depender da aplicação, outros tamanhos de código e palavra podem ser utilizados para melhorar o desempenho em um determinado contexto ou simplesmente atender a requisitos de *hardware*. Estas questões serão avaliadas nos Experimentos 2 e 3 da Seção 4.

Existem diferentes versões do LZW de acordo com a forma pela qual o dicionário é utilizado. Duas delas que serão exploradas neste trabalho são:

- LZ78: utiliza dicionário de arranque nulo.
- LZW: utiliza dicionário de arranque sendo todas as combinações de palavras de 8 bits (256 possíveis valores no total).

Para a aplicação em estudo nesse trabalho, onde os dados são fluxos de bits quaisquer, o dicionário de arranque é por padrão construído com todas as permutações de bits proporcionais ao tamanho da palavra definida pela implementação do algoritmo (*e.g.*, 8 bits, 16 bits). Além disso, serão considerados dicionários adaptativos, uma vez que os dispositivos utilizados devem apresentar limitação de armazenamento.

4. Experimentos

Os experimentos foram realizados em computador com processador Intel(R) Core(TM) i7-860 de 2,8 GHz e 32 GB de memória principal. O tamanho de palavra de entrada utilizado nos testes é de 8 bits e o tamanho do código gerado é de 16 bits, o que limita o tamanho máximo do dicionário a 2^{16} valores. Os *traces* do rádio Ettus N200 foram gerados através de capturas em um canal de 20 MHz na faixa ISM de 2,4 GHz, em um ambiente com várias redes, durante aproximadamente 10 segundos, utilizando o GNUradio, resultando no total de 128 MiB. Nesse mesmo *software* é possível redirecionar o fluxo de amostras para um arquivo e tornar os experimentos mais simples de serem efetuados.

O objetivo principal dos experimentos descritos nessa seção é avaliar a taxa de compressão de algumas implementações dos algoritmos tradicionais da literatura sobre dados de amostras reais de um SDR. Para isso, algumas diferentes configurações de dados de entrada e tamanho de dicionário (para o LZW) foram testadas, com três objetivos:

1. comparar os algoritmos mais tradicionais: Huffman, LZW e a versão do LZW sem o dicionário de arranque (LZ78);
2. avaliar o desempenho do LZW limitando o tamanho do dicionário e reiniciando cada vez que alcançar o limite (cenário mais provável, já que a FPGA possui limitação no armazenamento do dicionário);
3. entender a influência do tamanho do dicionário para o LZW.

Dentre os algoritmos selecionados, o mais complexo seria o método de Huffman por ser necessário construir uma estrutura de árvore usando alguma linguagem de descrição de *hardware* por necessitar enviá-la ao receptor (embora exista uma versão adaptativa do método que permite construir as árvores independentemente no transmissor e no receptor). No entanto, este método foi considerado nos experimentos mesmo sabendo dessas dificuldades. A Tabela 1 resume as configurações dos experimentos.

Tabela 1. Configurações de dados de entrada e tamanhos de dicionário

Experimento	Tamanho da entrada (*trace*)	Tamanho do dicionário	Algoritmos avaliados
1	128 KiB	64 KiB	LZW, LZ78 e Huffman
2	6 MiB	64 KiB	LZW
3	128 KiB e 128 MiB	4KiB a 80 KiB	LZW

4.1. Experimento 1

O objetivo do primeiro experimento é comparar a eficiência de compressão do algoritmo adaptativo LZW implementado utilizando dicionário de arranque padrão, com o mesmo algoritmo implementado com o dicionário completamente vazio (o LZ78) e com o algoritmo não adaptativo de Huffman conforme apresentado na Seção 3. A eficiência do algoritmo é medida através da taxa de compressão, definida como a razão entre o tamanho do arquivo de entrada e do arquivo de saída.

Como entradas do algoritmo de compressão, foram utilizados 20 arquivos de entrada de 128 KiB cada. Estes arquivos foram obtidos dividindo-se um *trace* maior de amostras de dados coletada pelo rádio em blocos de 128 KiB. O tamanho máximo do dicionário para as implementações do LZW e LZ78 é de 64 KiB. Uma vez alcançado

esse limite, o dicionário para de ser atualizado até o final do processamento do fluxo de entrada.

A Figura 2 apresenta um gráfico que mostra a taxa de compressão obtida pelos algoritmos LZW (com dicionário de arranque), LZ78 (sem dicionário de arranque) e Huffman. O valor médio da taxa de compressão com o LZW foi de 1,78, para o LZ78 foi de 1,52 e para o de Huffman foi de 1,25.

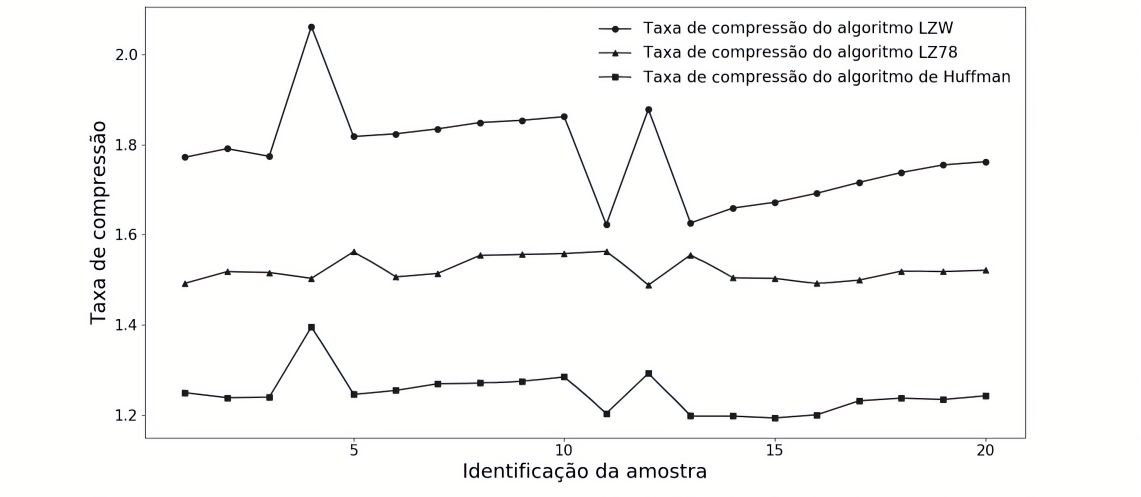

Figura 2. Taxa de compressão para o algoritmo de LZW, LZ78 e Huffman para blocos de amostra de 128 KiB.

Para os dados utilizados neste experimento, o algoritmo de compressão LZW utilizando dicionário de arranque padrão foi mais eficiente que a implementação utilizando dicionário completamente vazio em todos os grupos de amostra. Além disso, tanto o LZW quanto o LZ78 obtiveram taxas de compressão maiores que a do algoritmo de Huffman. Por esta razão, os próximos experimentos irão considerar apenas o LZW. Outro motivo é que o algoritmo LZW é mais simples de ser implementado em *hardware* comparado ao de Huffman que apresenta uma estrutura de dados de árvore que é mais complexa em relação ao dicionário do LZW e LZ78, além de trabalhar com números fracionários (para o cálculo da entropia).

4.2. Experimento 2

Um dos maiores desafios da utilização de dicionários adaptativos é a manutenção do dicionário. Como o dicionário é constantemente incrementado durante a compressão/descompressão, em algum momento a execução pode se deparar com limitações físicas (como quantidade de memória disponível) ou mesmo limitações do próprio algoritmo (como a quantidade de representações possíveis definidas pelo tamanho do código de palavra gerado).

Uma maneira de tratar este problema é definir o tamanho máximo, em bits, do código de saída de forma que, no pior caso, o dicionário não ocupe uma quantidade de memória maior que a disponível e o dicionário seja reconstruído sempre que se atingir

o limite da capacidade determinado ao invés de manter o mesmo dicionário para todo o fluxo de entrada. Desta maneira, o algoritmo fica adaptável[1] ao fluxo de dados de entrada.

O objetivo do segundo experimento foi verificar como esse processo de reinicialização do dicionário quando esse alcança seu tamanho limite impacta na taxa de compressão do LZW. Para isso, o tamanho do dicionário foi limitado a 64 KiB correspondendo a um tamanho de código máximo de 16 bits — um valor que funciona bem para os tamanhos de entrada avaliados (ver o experimento na Seção 4.3). A entrada utilizada nesse experimento foi um *trace* de 6 MiB. As taxas de compressão foram coletadas em intervalos regulares de 1 KiB. Isto é, a cada 1 KiB de dados de entrada consumidos, a taxa de compressão foi calculada pela razão entre o tamanho total do fluxo comprimido e pela quantidade total de dados consumidos no fluxo de entrada até aquele ponto. A Figura 3 apresenta a taxa de compressão obtida pelo algoritmo LZW ao longo da execução.

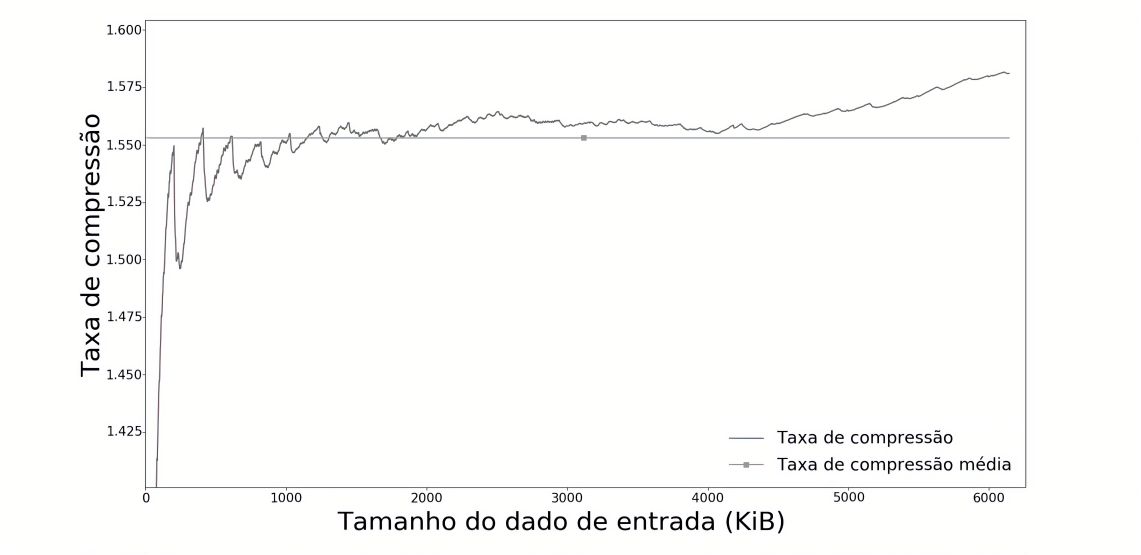

Figura 3. Taxa de compressão acumulada para o algoritmo LZW limitando o tamanho do dicionário a 64 KiB.

É importante mencionar que, para os dados utilizados neste experimento, os algoritmos implementados com dicionário completamente vazio e com dicionário de arranque padrão mantiveram taxas de compressão similares durante toda a execução. Os momentos em que o dicionário atinge o limite de tamanho e é reconstruído podem ser observados pelos pontos de máximo local do gráfico. À medida que o processo de compressão ocorre, a diferença entre os pontos de máximo e mínimo locais diminuem e a taxa de compressão converge para 1,55, aproximadamente 14,83% a menos que a média do experimento reportado na Seção 4.1.

4.3. Experimento 3

Para entender a influência do tamanho do dicionário na compressão de dados, a implementação do algoritmo LZW foi avaliada variando-se o limite de tamanho do dicionário.

[1]Diferente de *adaptativo*, que consiste em ambos os lados da transmissão apenas dependerem do fluxo de dados de entrada

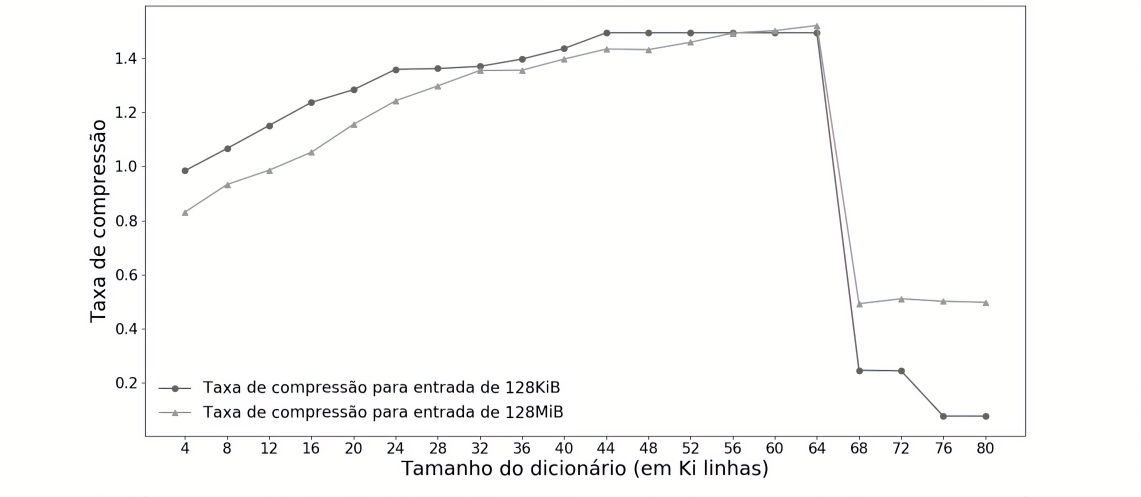

Figura 4. Taxa de compressão do LZW variando o tamanho do dicionário para tamanhos da entrada de 128 KiB e 128 MiB.

A Figura 4 mostra esse comportamento para uma entrada de tamanho de 128 KiB e outra de 128 MiB. Para tamanhos de dicionário pequenos, a taxa de compressão é baixa e, conforme vai se aproximando de 64 Ki linhas, a taxa de compressão se aproxima de 1,5. Após esse valor, o tamanho do código precisa de ser de 32 bits (que antes era de 16 bits), fazendo com que o tamanho do dado comprimido seja maior que a entrada para a maioria dos dados tanto para entradas pequenas quanto para grandes. Deste modo, o resultado da taxa de compressão fica em torno de 0,5 para entrada de 128 MiB e abaixo de 0,3 para entrada de 128 KiB, valores que indicam que não há diminuição dos dados de saída mas um aumento.

5. Estado da Arte da Compressão Sem Perdas em FPGA

Alguns trabalhos na literatura lidam diretamente com o problema de compressão sem perda em FPGA [Qiao et al. 2018, Zhou et al. 2016, Fowers et al. 2015, Bartík et al. 2015, Cui 2007, Rigler et al. 2007, Dandalis and Prasanna 2005], que é o foco deste trabalho. Um conjunto deles se preocupa com a aceleração da implementação do algoritmo [Qiao et al. 2018, Zhou et al. 2016, Fowers et al. 2015] enquanto que outros lidam com eventuais limitação do dispositivo, como a capacidade de armazenamento e número de elementos lógicos utilizados [Bartík et al. 2015, Rigler et al. 2007, Dandalis and Prasanna 2005]. Ainda existem aqueles que investem em melhorar a taxa de compressão para tipos específicos de dados com no trabalho em [Cui 2007].

O artigo [Qiao et al. 2018] afirma aumentar a vazão de transmissão em relação ao estado da arte [Fowers et al. 2015]. Ele apresenta uma implementação paralela nas FPGAs Startix V e Arria 10, ambas do fabricante Xilinx, que corresponde a uma junção dos codificadores LZ77 e Huffman (chamado de DEFLATE), base do mais utilizado compactador de arquivos, o GZIP. O LZ77 é um algoritmo de compressão sem perdas não patenteado (diferentemente do LZW) que trabalha com janelas deslizantes sobre o fluxo de dados de entrada e gera tuplas que associam a posição de sequências repetidas nessa janela. Esta implementação integra as plataformas CPU e FPGA, usando barramento de

comunicação de alta capacidade e baixa latência. Todos os experimentos foram realizados com dados de um *benchmark* com diversos tipos de arquivos.

Com a intenção de acelerar o processo de compressão, os autores em [Zhou et al. 2016] descrevem uma implementação da compressão do LZW em uma FPGA. Experimentos usando arquivos de imagens mostraram que sua proposta, avaliada em uma FPGA Virtex 7 da Xilinx, é 23,5 vezes mais rápida que a implementação sequencial do LZW em uma CPU.

Sobre os trabalhos que lidam com as características dos dispositivos de *hardware* configurável, os autores em [Bartík et al. 2015] implementaram o compressor LZ4 que é um derivado do padrão LZ77 focado na velocidade de compressão (e sacrificando a taxa de compressão). Testes foram realizados em FPGAs da Xilinx, Virtex 6 e Virtex 7, comparando o uso de recursos do dispositivo com a compressão LZRW, uma versão industrial semelhante ao LZ4. Similarmente, o trabalho em [Rigler et al. 2007] apresenta o processo de desenvolvimento, em uma linguagem de descrição de *hardware*, a VHDL, dos algoritmos de compressão LZ77 e Huffman (base do GZIP) e mostra o uso de recursos em uma placa da Altera DE2.

6. Conclusões e Trabalhos Futuros

Este artigo traz uma análise de alguns métodos e parâmetros de compressão de dados sem perda de modo a viabilizar a implementação de um destes métodos no dispositivo de *hardware* configurável (FPGA) de um SDR. Tal experimentação preliminar foi desempenhada em um processador utilizando um fluxo de amostras reais de rádio-frequência em canais Wi-Fi através do rádio modelo N200 da fabricante Ettus. Algoritmos clássicos da literatura, o LZW, o LZ78 e o algoritmo de Huffman — que trabalham com dicionário —, foram escolhidos para serem testados uma vez que são simples e, portanto, mais propícios de serem implementados na FPGA do rádio.

O algoritmo LZW se demonstrou o mais eficiente na compressão dos dados do SDR, com taxa de compressão de 1,78, comparado ao LZ78 (igual ao LZW, mas sem o dicionário de arranque) e ao algoritmo de Huffman. O algoritmo de Huffman foi considerado nos experimentos mesmo sabendo das dificuldades de implementação em *hardware* na FPGA devido a precisar construir uma estrutura de dados complexa de árvore e por necessitar enviá-la ao receptor. Além disso, pensando em um cenário real para uma FPGA com pouca memória disponível para armazenar o dicionário, limitamos o tamanho do dicionário e fizemos com que ele seja reiniciado cada vez que se tornasse completo. Essa solução mostrou-se uma alternativa bastante promissora para o algoritmo LZW e LZ78, tendo uma compressão média de 1,55 sobre um fluxo de entrada de 6 MiB. Deste modo, estes algoritmos poderiam ser implementados com o dicionário sendo reiniciado toda vez que chegar ao limite de memória disponível na FPGA.

Como trabalhos futuros pretende-se realizar mais experimentos semelhantes incluindo outras técnicas de compressão de dados sem perdas, como o DEFLATE (LZ77 + Huffman) bastante usada nos trabalhos relacionados, inclusive com implementações em FPGAs. Outra vertente a ser explorada é a reinicialização do dicionário em outros pontos da execução — *i.e.*, não apenas quando o tamanho limite for atingido — com o objetivo de expurgar códigos que deixam de ser frequentes ao longo do fluxo de dados de entrada. Esse processo teria o potencial de melhorar as taxas de compressão para fluxos de

entrada particularmente longos. Este trabalho visa também construir uma configuração de *hardware* possivelmente dos algoritmos LZW ou LZ78 na FPGA do rádio e observar as melhorias desta solução para o problema de transmissão de amostras de um SDR. Além disso, é importante considerar a capacidade de processamento da FPGA e, portanto, considera-se futuramente avaliar esta condição.

Referências

Bartík, M., Ubik, S., and Kubalik, P. (2015). LZ4 compression algorithm on FPGA. In *2015 IEEE International Conference on Electronics, Circuits, and Systems (ICECS)*, pages 179–182.

Breslau, L., Estrin, D., Yu, H., Fall, K., Floyd, S., Heidemann, J., Helmy, A., Huang, P., McCanne, S., Varadhan, K., et al. (2000). Advances in network simulation. *Computer*, 33(5):59–67.

Cui, W. (2007). New LZW data compression algorithm and its FPGA implementation. In *Proc. 26th Picture Coding Symposium (PCS 2007)*.

Dandalis, A. and Prasanna, V. K. (2005). Configuration compression for FPGA-based embedded systems. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 13(12):1394–1398.

Dascal, V., Dolea, P., Cristea, O., and Palade, T. (2013). Low-cost SDR-based ground receiving station for LEO satellite operations. In *2013 11th International Conference on Telecommunications in Modern Satellite, Cable and Broadcasting Services (TELSIKS)*, volume 02, pages 627–630.

Dillinger, M., Madani, K., and Alonistioti, N. (2005). *Software defined radio: Architectures, systems and functions*. John Wiley & Sons.

Ettus Research (2019). Especificações do modelo N200. Disponível em `https://www.ettus.com/content/files/07495_Ettus_N200-210_DS_Flyer_HR.pdf` (acessado em 01/12/2019).

Fowers, J., Kim, J.-Y., Burger, D., and Hauck, S. (2015). A scalable high-bandwidth architecture for lossless compression on fpgas. In *2015 IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines*, pages 52–59. IEEE.

GNU Radio (2019). Página oficial do projeto GNU Radio. Disponível em `http://gnuradio.org/` (acessado em 01/12/2019).

Huffman, D. A. (1952). A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101.

IEEE LAN/MAN Standards Committee (2013). 802.11ac-2013 IEEE standard for information technology – LAN/MAN – specific requirements – part 11: Wireless LAN medium access control (MAC) and physical layer (PHY) specification – amendment 4: Enhancements for very high throughput for operation in bands below 6 GHz.

Linux Wireless (2015). About mac80211. Disponível em `https://wireless.wiki.kernel.org/en/developers/documentation/mac80211` (acessado em 27/12/2019).

McHenry, M. A., Tenhula, P. A., McCloskey, D., Roberson, D. A., and Hood, C. S. (2006). Chicago spectrum occupancy measurements & analysis and a long-term studies proposal. In *Proceedings of the first international workshop on Technology and policy for accessing spectrum*, page 1. ACM.

Mitola III, J. and Maguire Jr, G. Q. (1999). Cognitive radio: making software radios more personal. *IEEE Personal Communications*, 6(4):13–18.

Qiao, W., Du, J., Fang, Z., Lo, M., Chang, M. F., and Cong, J. (2018). High-throughput lossless compression on tightly coupled CPU-FPGA platforms. In *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 37–44.

Reis, A. L. G., Barros, A. F., Lenzi, K. G., Meloni, L. G. P., and Barbin, S. E. (2012). Introduction to the software-defined radio approach. *IEEE Latin America Transactions*, 10(1):1156–1161.

Rigler, S., Bishop, W., and Kennings, A. (2007). FPGA-based lossless data compression using huffman and LZ77 algorithms. In *2007 Canadian Conference on Electrical and Computer Engineering*, pages 1235–1238.

Rondeau, T. W., Le, B., Rieser, C. J., and Bostian, C. W. (2004). Cognitive radios with genetic algorithms: Intelligent control of software defined radios. In *Software defined radio forum technical conference*, pages C3–C8.

Sayood, K. (2002). *Lossless compression handbook*. Elsevier.

Sayood, K. (2012). *Introduction to Data Compression, Fourth Edition*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 4th edition.

Shannon, C. E. (1949). Communication in the presence of noise. *Proceedings of the IRE*, 37(1):10–21.

VANU (2019). SDR: Software defined radio. Disponível em `http://www.vanu.com/documents/media/SoftwareDefinedRadio.pdf` (acessado em 01/12/2019).

WARP Project (2019). Wireless open access research platform. Disponível em `http://warpproject.org/trac/` (acessado em 01/12/2019).

Welch, T. A. (1984). A technique for high-performance data compression. *Computer*, 17(6):8–19.

Wi-Fi Alliance (2019). Página oficial da Wi-Fi Alliance. Disponível em `http://www.wi-fi.org/` (acessado em 01/12/2019).

Zhou, X., Ito, Y., and Nakano, K. (2016). An efficient implementation of lzw compression in the fpga. In Carretero, J., Garcia-Blas, J., Ko, R. K., Mueller, P., and Nakano, K., editors, *Algorithms and Architectures for Parallel Processing*, pages 512–520, Cham. Springer International Publishing.