Um Protocolo Multicast para Comunicação por Luz Visível (VLC)

Samuel P. C. Sena[1], **Luiz F. M. Vieira** [2], **Marcos A. M. Vieira** [2], **Alex B. Vieira**[3], **Edelberto F. Silva**[3], **José Augusto M. Nacif**[1]

[1]Instituto de Ciências Exatas e Tecnológicas – Universidade Federal de Viçosa (UFV)
[2]Departamento de Ciência da Computação – Universidade Federal de Minas Gerais (UFMG)
[3]Departamento de Ciência da Computação – Universidade Federal de Juiz de Fora (UFJF)

{samuel.sena, jnacif}@ufv.br, {lfvieira, mmvieira}@dcc.ufmg.br,
{alex.borges, edelberto.franco}@ufjf.br

Abstract. *Currently, the demand for Wireless Internet resources has grown exponentially. However, the wireless communication technologies available today may not be enough to keep up with the growing demand for bandwidth. In this context, Visible Light Communication (VLC) is a valid and promising alternative, given the high data transfer rate, the use of a lighting infrastructure already widely disseminated today, in addition to a wide spectrum available and not reserved for use. Therefore, this work aims to present an implementation of the multicast network protocol for visible light communication.*

Resumo. *Atualmente, a demanda por comunicação por redes sem fio tem crescido exponencialmente. Contudo, as tecnologias de comunicação sem fio disponíveis hoje podem não ser suficientes para acompanhar a crescente demanda por banda. Nesse contexto, a Comunicação por Luz Visível (VLC) consiste em uma possibilidade válida e promissora, tendo em vista a alta taxa de transferência de dados, a utilização de uma infraestrutura de iluminação já amplamente difundida na atualidade, além de um largo espectro disponível e não reservado para o uso. Logo, este trabalho tem como objetivo apresentar uma implementação do protocolo de rede Multicast na Comunicação Por Luz Visível.*

1. Introdução

A demanda por taxas de transferências maiores em redes sem-fio cresce a cada dia. Grande parte dessa evolução deve-se ao crescente volume de dados consumidos por dispositivos portáteis [Matheus et al. 2017]. Esses dispositivos estão cada dia mais velozes e requisitam uma maior quantidade de recursos de trocas de dados. Entretanto, a disponibilidade destes recursos não acompanha a crescente demanda na atual frequência das ondas eletromagnéticas e assim, sobrecarrega o uso de frequências das ondas utilizadas na tecnologia WiFi. Não obstante, a comunicação por luz visível é uma opção viável para acompanhar a demanda crescente por comunicação, com velocidades de transmissão altas. Por exemplo, pesquisas com VLC já obtiveram resultados extremamente expressivos neste aspecto, chegando a velocidades superiores a 100 Gbps [Gomez et al. 2015]. Além disso, com o IP *multicast* é possível realizar uma espécie de desobstrução do canal de comunicação. Este consiste em uma tecnologia de conservação de largura de banda, reduzindo o tráfego ao fornecer um único fluxo de informações simultaneamente para potencialmente milhares de usuários [Williamson 2000].

Tendo em vista a crescente demanda por recursos de trocas de dados e situações nas quais um conjunto de *hosts* demandam pelo mesmo fecho de pacotes, o *multicasting* se torna uma excelente opção para otimizar a sobrecarga de recursos sem fio. Contudo, no atual cenário da comunicação por luz visível não há um grande volume de soluções *multicast* bem difundidas e disponíveis para a comunidade que atua nesta linha de pesquisa. O principal trabalho similar existente é trazido por [Sales et al. 2021], em seu artigo é apresentado o protocolo MCAST-VLC. Este consiste em um protocolo completo para roteamento e comunicação em *multicast*, capaz de realizar gerenciamento de grupo, isolamento de tráfego e descoberta de rotas multi-hop.

Neste artigo é apresentado uma implementação de um protocolo *multicast* para comunicação por luz visível. Foi considerado a realização de uma implementação simples, com ferramentas de rede comuns e facilmente encontradas. Além disso, através da utilização de testes executados sobre o protocolo, foram realizadas aferições que conseguem trazer resultados a respeito do comportamento do protocolo. É esperado que tais resultados possam motivar e servir como base para novas implementações de protocolos que tenham o intuito de melhorar ainda mais a eficiência na comunicação por luz visível.

Ao contrário do trazido por [Sales et al. 2021], esta implementação prezou pela simplicidade e pelo foco central em permitir que *n hosts* clientes recebam o mesmo fluxo de pacotes enviados por um *host* servidor, sem a utilização de subgrupos *multicast* e sem uma inteligência capaz de nomear novos *hosts* como gerentes de subgrupos ou traçar novas rotas. Dessa forma, foi possível implementar um protocolo de fácil interpretação e alta legibilidade, além de utilizar ferramentas como o paralelismo de rotinas, armazenamento de endereços de maneira persistente e utilização de diferentes tipos de *sockets* de rede, visando a vantagem de cada um em cada determinado contexto.

A estruturação do restante do artigo segue como o descrito: A Seção 2 define e apresenta a configuração do contexto da implementação do protocolo. A Seção 3, detalha o funcionamento do algoritmo proposto. A Seção 4 descreve como foram realizadas as configurações de rede para execução do algoritmo proposto. Na Seção 5 é explicado toda a execução do algoritmo em cada um dos *hosts*. Na Seção 6 os resultados são apresentados graficamente e discutidos. A Seção 7 exibe as considerações finais e trabalhos futuros e, por fim, o Apêndice A apresenta pseudocódigos dos principais módulos presentes na implementação do nosso protocolo.

2. Fundamentos

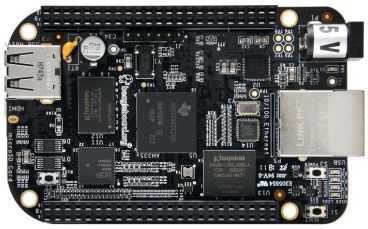
2.1. Visible Light Communication

A Comunicação por Luz Visível (Visible Light Communication - VLC) é o nome atribuído para o processo de comunicação em que os dados são enviados através da modulação de ondas eletromagnéticas que fazem parte do espectro visível da luz. Logo, qualquer método de transferência de informações que utilize a luz no espectro visível pode ser classificado como comunicação por luz visível. Neste trabalho, foi utilizado a plataforma OpenVLC na versão 1.0 para implementação e testes da proposta.

2.2. OpenVLC

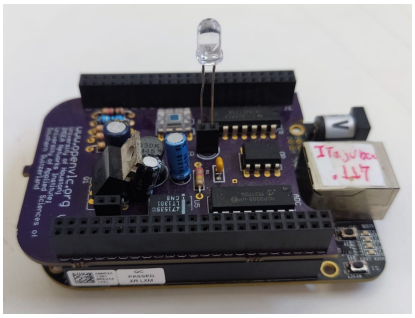
A plataforma escolhida para desenvolvimento da aplicação em comunicação por luz visível foi a OpenVLC. Como apresentado por [Matheus et al. 2017,

Wang et al. 2015], a OpenVLC consiste em uma plataforma de código aberto que é acoplada ao BeagleBone Black (BBB), sendo este um computador de placa única que opera com a arquitetura ARM de fácil utilização e baixo custo, conforme mencionado por [Nayyar and Puri 2016]. A plataforma OpenVLC é considerada uma placa de expansão contendo componentes responsáveis pela modulação e demodulação de sinais, além do envio e recebimento de ondas eletromagnéticas no espectro visível. A recepção dos sinais pode ser feita através de um LED ou de um fotodiodo e a transmissão através de um LED de alto brilho ou um LED de alta potência. A Figura 1 ilustra o computador de placa única BBB utilizado e a placa OpenVLC devidamente acoplada a este.



(a) BeagleBone Black [a]

[a]beaglebone.org



(b) OpenVLC acoplado a BBB

Figura 1. Equipamento utilizado

O *software* que compõe a plataforma OpenVLC é implementado nas camadas de rede física e enlace. Seu carregamento é realizado no sistema através de um módulo para o Kernel Linux. Dessa forma, a comunicação é intermediada entre a pilha de rede do Linux e a placa de expansão. Logo, as principais vantagens na utilização da OpenVLC são a facilidade do uso devido a integração com a pilha de rede do Linux, o baixo custo e flexibilidade.

2.3. Multicast

O IP *multicast* consiste em uma tecnologia de conservação de largura de banda que reduz o tráfego ao fornecer um único fluxo de informações simultaneamente para potencialmente milhares de usuários [Williamson 2000]. O roteamento *multicast* permite que um *host* origem envie pacotes para um grupo de *hosts* destino utilizando um endereço de IP de um grupo, chamado grupo *multicast*.

Como afirmado por [Williamson 2000], a utilização de redes com IP *multicast* permitem que o tráfego de dados seja reduzido através da transmissão de um único fluxo de informações para diversos clientes. Diversas tecnologias podem tirar vantagem desse tipo de transmissão, dentre elas se destacam: streaming de vídeo; ensino a distância; distribuição de *softwares*; divulgação de notícias. As redes IP *multicast* possuem a capacidade de suportar milhares de clientes simultaneamente. Clientes que possuem o interesse de receber pacotes endereçados a um grupo *multicast* podem realizar uma requisição através de protocolos para ingressarem no respectivo grupo. Um grupo *multicast* consiste em um conjunto de destinatários que possuem o interesse em comum de receber um determinado fluxo de dados. A *Internet Assigned Numbers Authority* (IANA), responsável pela numeração do IPv4 (*Internet Protocol version 4*), reservou a faixa de endereços 224.0.0.0 até 239.255.255.255 para atribuição aos grupos de *multicast* ao redor do mundo.

3. Arquitetura Proposta para Multicast VLC

A implementação de um protocolo IP *multicast* na plataforma OpenVLC 1.0 foi realizada através do uso da linguagem de programação Python 3. O algoritmo em questão conta com as funcionalidades de lista de clientes *multicast* e endereço de grupo *multicast* armazenado em memória persistente, além de operar com o uso de *threads* para realizar envios de mensagens ao grupo *multicast* e receber requisições de entrada e saída do grupo *multicast*. O algoritmo é composto por três módulos, sendo um destinado exclusivamente para ser executado na BeagleBone que opera como servidor *multicast*. Os dois módulos restantes são destinados para serem executados exclusivamente em clientes, o primeiro tem a funcionalidade de realizar requisições de entrada em grupo *multicast* e receber mensagens advindas deste, já o segundo arquivo tem a finalidade de realizar uma requisição de saída do grupo *multicast* ao servidor.

A memória persistente no módulo servidor tem a funcionalidade de armazenar a lista de clientes que compõem o grupo *multicast* para execuções futuras. Já no módulo cliente, ela é responsável por armazenar o endereço do grupo *multicast* adquirido em uma execução anterior, dessa forma, é evitado mais de uma requisição de entrada em grupo para o mesmo cliente em diferentes momentos de execução.

A tecnologia de *Sockets* de rede presentes nas bibliotecas padrão da linguagem Python foi utilizada como base para implementação do protocolo. Contudo, o envio de mensagens entre servidor e clientes *multicast* foi implementado de duas maneiras: para o envio de mensagens textuais é utilizado *sockets* UDP, devido a não necessidade de ordenação de pacotes e uma baixa quantidade de perda de pacotes obtida em testes, além desta abordagem possuir o melhor desempenho; e para o envio de arquivos é utilizado *sockets* TCP, devido a facilidade em transferir arquivos de diversos tamanhos preservando a integridade da ordem de pacotes recebidos e realizando o reenvio automático de eventuais pacotes perdidos ou corrompidos durante a transferência. Em paralelo a isso, em ambos os tipos do protocolo é executado inicialmente uma conexão TCP entre cliente e servidor, responsável pela troca de mensagens para ingresso no grupo *multicast*, além da troca de informações a respeito do endereço do grupo *multicast* e porta utilizada.

Dos dois tipos de envios implementados, o mais utilizado para documentação foi o de envios de mensagens textuais, devido a sua maior simplicidade e foco na proposta deste trabalho. No entanto, a versão implementada para envio de arquivos ainda conta com a funcionalidade de verificação de integridade de arquivo recebido pelo cliente. Para isso, é utilizado o algoritmo de *hash* apresentado por [Eastlake 3rd and Jones 2001] SHA-1 (*Secure Hash Algorithm 1*) sobre o arquivo no servidor e no cliente. O servidor realiza o envio de uma tupla com três itens: uma *string* do nome do arquivo enviado, o conteúdo do arquivo em binário e uma *string* do hash SHA-1 do arquivo. O cliente, ao acabar de receber todos os *dataframes* e reconstruir o arquivo, computa o *hash* e compara o resultado. Caso o resultado seja igual, um *match* é informado, caso contrário o arquivo é dado como corrompido.

3.1. Servidor

O funcionamento do módulo servidor é dividido em três partes principais: A primeira, que sempre é executada no início da execução do módulo, é responsável pela recuperação ou criação de uma nova lista de clientes. A segunda, executada de maneira

paralela a terceira, é responsável pelo atendimento de requisições de clientes para entrar e sair do grupo *multicast*. E a terceira responsável pelo aguardo de entrada destinada ao envio para todos os clientes presentes no grupo *multicast*. A Figura 2 ilustra de maneira prática a ordem das ações presentes no módulo servidor e os Algoritmos 1, 2, 3 e 4 presentes no apêndice A descrevem as rotinas executadas no módulo servidor.

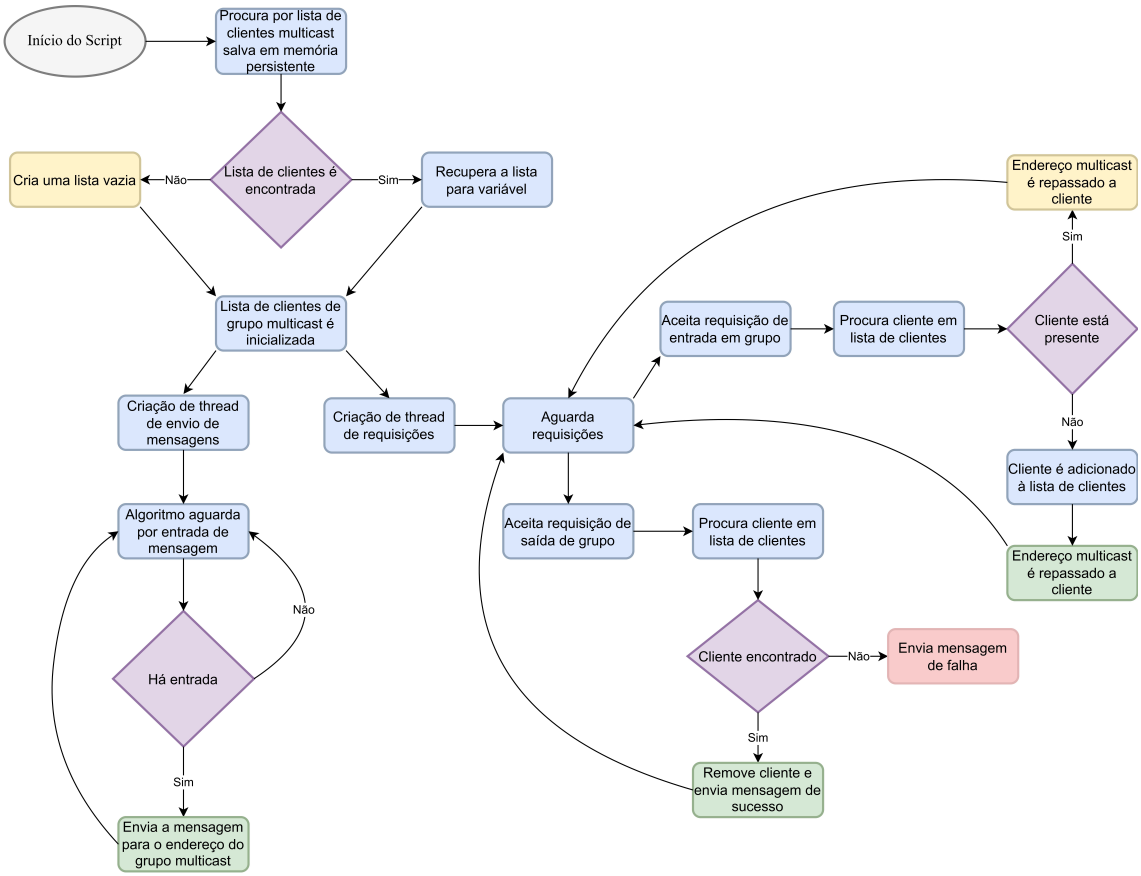


Figura 2. Fluxo do algoritmo Servidor

3.2. Cliente

O funcionamento do módulo cliente é relativamente mais simples em comparação ao módulo servidor. Este possui apenas duas partes principais executadas de maneira totalmente sequencial: A primeira é responsável por recuperar o endereço de grupo *multicast* armazenado, e, caso não encontre, o mesmo realiza uma nova solicitação de entrada em grupo ao módulo servidor. Em seguida, a segunda parte do módulo cliente entra em em uma espécie de modo "ouvinte", permitindo que assim o cliente seja passível de receber mensagens do servidor a qualquer instante. A Figura 3 apresenta um fluxograma do funcionamento e os Algoritmos 5 e 6 presentes no apêndice descrevem as rotinas presentes no módulo cliente.

3.3. Algoritmo de Roteamento

Para melhorar a eficiência e auxiliar em possíveis problemas com rotas e visibilidade da iluminação de uma OpenVLC a outra, é possível utilizar algum algoritmo de roteamento arbitrário em conjunto com o protocolo apresentado. O protocolo *multicast*

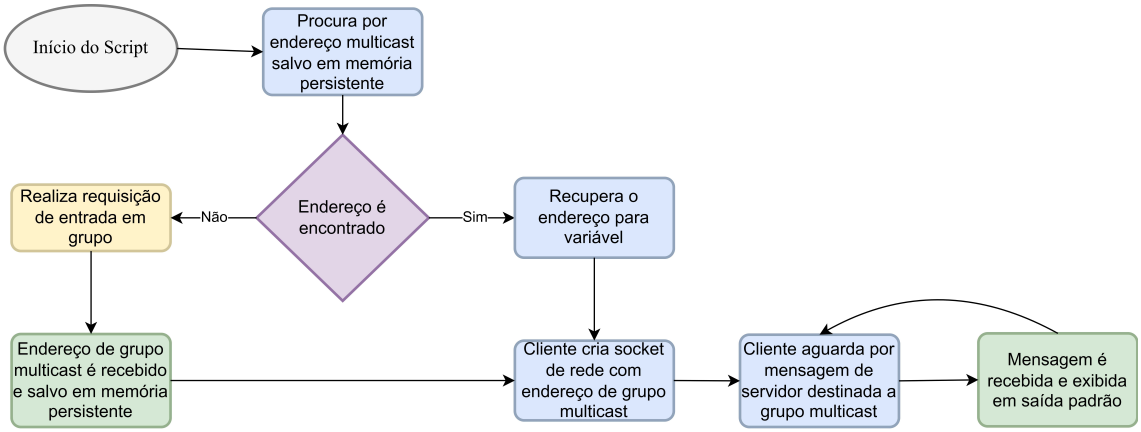


Figura 3. Fluxo do algoritmo Cliente

apresentado por esse trabalho é implementado para executar na camada de aplicação, dessa forma, é possível a execução paralela de algum algoritmo de roteamento em camadas inferiores a esta. O algoritmo escolhido para permitir um melhor gerenciamento de rotas foi o apresentado por [Matheus et al. 2019], chamado de DYRP-VLC (*dynamic routing protocol for visible light communication networks*). Neste, rotas são dinamicamente criadas à medida em que novos nós ingressam na rede, além disso, caso um nó não possua comunicação direta com outro, nós terceiros se encarregam de retransmitir pacotes entre origem e destino.

4. Preparação do Ambiente

Para preparar o ambiente de execução do protocolo implementado foi separado três BeagleBones Black devidamente acopladas a placas OpenVLCs na versão 1.0. A BeagleBone servidor foi configurada para operar no IP 192.168.0.1, já as duas outras BeagleBone foram configuradas para operar nos IPs 192.168.0.2 e 192.168.0.3. Além disso, todas foram configuradas para operar na frequência de 50 Hz, enviar dados por um LED de alto brilho da cor azul (as cores azul e vermelho possuem um melhor desempenho de transmissão com diferentes distâncias na OpenVLC 1.0) e receber dados através do fotodiodo. Por último, as BeagleBone foram postas de perfil, sendo que as duas configuradas como clientes foram postas de frente para a configurada como servidor, de forma que os sinais ópticos enviados atinjam ambas máquinas cliente. A Figura 4 ilustra a disposição das BeagleBone acopladas a uma OpenVLC.

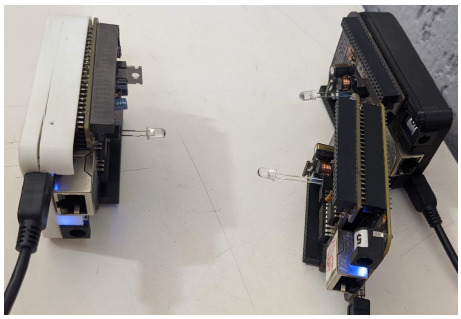


Figura 4. Alinhamento das placas

O protocolo foi configurado para trabalhar com *dataframes* com um *payload* de 1024 *bytes*, o protocolo opera com a configuração de apenas um grupo *multicast* com o endereço definido em 224.1.1.5, além das portas 5060 e 50001 configuradas para o *socket* de requisição TCP de entrada e saída do grupo e envio de pacotes UDP para o grupo *multicast*, respectivamente. Tais configurações são arbitrárias, podendo ser alteradas sem comprometimento das funcionalidades do protocolo.

5. Metodologia de execução

A execução do protocolo consistiu na inicialização dos módulos através do interpretador da linguagem de programação Python em cada placa. A BeagleBone configurada como servidor executa o módulo implementado especialmente para ela e as duas BeagleBone clientes executam o mesmo módulo destinado para clientes.

O módulo servidor, ao iniciar, verifica se existem informações sobre a lista de clientes armazenadas de maneira persistente resultantes de uma execução anterior. Caso sejam encontradas tais informações, as mesmas são restauradas para uma variável de execução, caso contrário, uma nova lista de clientes vazia é inicializada. Em seguida, é iniciado uma *thread* responsável por aguardar requisições de entrada e saída de clientes no grupo *multicast*. Em paralelo a isso, outra *thread* é inicializada, esta permite que mensagens textuais sejam enviadas através da entrada padrão.

O módulo cliente, também possui suporte a memória persistente, este, caso não encontre um endereço de grupo *multicast* armazenado de uma execução anterior, realiza uma requisição de entrada em grupo para o servidor. O servidor por sua vez verifica se o cliente já não se encontra presente em sua lista de clientes do grupo *multicast*, caso esteja, o cliente não é adicionado novamente e apenas o endereço do grupo é repassado ao cliente solicitante. Caso contrário, o cliente é armazenado na lista de clientes e o endereço do grupo também é repassado ao cliente solicitante.

Uma vez que o módulo cliente possui o endereço do grupo *multicast* armazenado em variável de execução ou em memória persistente, o mesmo entra em um modo de monitoramento contínuo de pacotes advindos do servidor com destino a membros do grupo *multicast*. Com isso, todas as mensagens textuais enviadas pelo servidor são exibidas em todos os clientes que aguardam pela chegada.

Se em algum dado momento, um cliente desejar evadir do grupo *multicast*, um segundo módulo implementado para esse propósito deverá ser executado. O cliente realiza uma requisição para o servidor e solicita a sua remoção da lista de clientes do grupo *multicast*. O servidor, caso encontre o respectivo cliente em sua lista, realiza a remoção e informa ao cliente o sucesso da operação. Por último, o módulo realiza a limpeza da memória persistente que armazena o endereço do grupo *multicast* no cliente.

6. Resultados

Com a realização da execução dos módulos em servidor e clientes, é possível observar os processos de requisição de entrada e saída do grupo *multicast*, além do envio e recepção de mensagens textuais entre servidor e clientes. Ainda, o envio de pacotes que transportavam arquivos também foi testado, no entanto, a eficiência desse método de envio foi bastante reduzida, uma vez que a velocidade de transferência entre as plataformas OpenVLC na versão 1.0 é bastante reduzida (cerca de 15 Kbps a 50 Hz).

Em seguida, para realizar a medição de métricas relativas à operação do protocolo, foram feitas modificações especialmente para permitir o monitoramento e registro de informações referentes a perda de pacotes, vazão e atraso, sem interferir no funcionamento normal do protocolo. Infelizmente, os resultados trazidos por [Sales et al. 2021] se concentram em gráficos de ocupação de largura de banda e, por isso, não puderam ser comparados diretamente com os resultados trazidos por este trabalho.

6.1. Perda de pacote

Tendo em vista que o protocolo de envio de mensagens textuais para o grupo *multicast* utiliza pacotes enviados por um *socket* que utiliza o protocolo UDP, não há um suporte a um mecanismo de correção de perda de pacote ativo. Tendo isso em vista, um método para a medição de perda de pacotes foi implementado. Nesta, são realizadas 30 iterações de envio de 30 mensagens textuais, totalizando 900 mensagens enviadas. Em caso de alguma destas mensagens não serem recebidas por algum cliente, uma contabilização é feita e salva em arquivo de texto. Foi observado que caso as mensagens adjacentes sejam enviadas de maneira contínua e sem aguardar um período de tempo, o *buffer* de recebimento da OpenVLC é cheio e uma grande quantidade de pacotes é perdido (*dropped*).

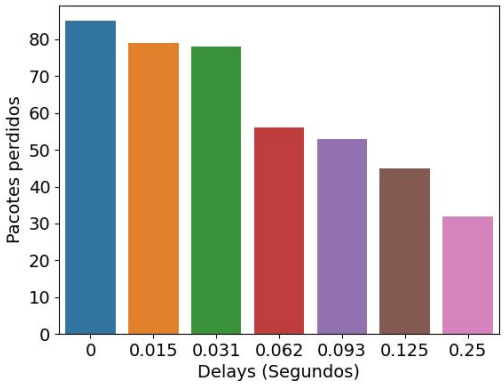


Figura 5. Gráfico de Perda de Pacote

É possível perceber pela Figura 5 que, quando o intervalo de tempo entre um envio e outro de mensagem é muito pequeno, a quantidade de mensagens perdidas é muito grande devido a incapacidade da OpenVLC de acompanhar a velocidade de recebimento das mensagens (*packet drop*). Foi observado que para intervalos superiores a 250 ms a perda de pacote para 900 mensagens se manteve dentro de 30 pacotes. Logo, os resultados de menores perdas de pacote são referentes a erros de comunicação física na plataforma OpenVLC. A partir disso, temos que com o devido intervalo entre envios de mensagens adjacentes não há prejuízos no envio de mensagens utilizando o protocolo *multicast*.

6.2. Vazão

Para o teste de vazão de *download* dos clientes na rede *multicast* foi utilizado a implementação com *sockets* que utilizam protocolo TCP (envio de arquivo), tendo em vista o controle de ordenação e correção de perda de pacotes presentes de maneira nativa na implementação. Contudo, neste cenário o desempenho final para enviar um arquivo para todos os clientes presentes no grupo *multicast* é drasticamente reduzido, uma vez

que para cada um dos clientes, uma conexão TCP deverá ser estabelecida. A figura 6 faz uma comparação da vazão atingida em cada uma das BBBs que executaram os testes para as distâncias de 10 cm, 20 cm, 40 cm, 80 cm e 100 cm. Para distâncias superiores, a comunicação da OpenVLC 1.0 se torna muito instável e, devido a isto, a distância máxima foi estabelecida em 100 cm.

Através da análise dos *boxplots*, para cada distância apresentada, é possível observar, que a medida em que a distância aumenta, a velocidade diminui na distribuição de maneira gradual. Também é possível observar que a distribuição dos dados possui uma menor amplitude para a distância de 10 cm (Figura 6a), algo que aponta que para essa distância a distribuição de resultados apresentou uma menor variação. Vale ressaltar que, apesar das diferentes performances na transferência em cada uma das distâncias avaliadas, os dados enviados foram recebidos de maneira satisfatória em todas as ocasiões, tendo sua integridade avaliada através do algoritmo de *hash* SHA-1 de [Eastlake 3rd and Jones 2001].

6.3. Atraso

O atraso entre envio e recebimento de mensagens textual para servidor e clientes foi medido em milissegundos em cerca de 30 iterações de 30 envios de mensagens, totalizando 900 envios para cada distância avaliada. Em seguida, um gráfico de Função de distribuição empírica (eCDF) foi elaborado a partir da distribuição de resultados de atrasos de cada distância (Figura 7).

Através da análises destes gráficos, é possível perceber que a distribuição de atrasos se concentra principalmente nos valores entre 0.5 e 0.6 segundos (cerca de 90% dos resultados). Contudo, à medida que a distância aumenta, os resultados que ficam com atrasos superiores a 0.6 segundos tem seu valor em atraso acrescido.

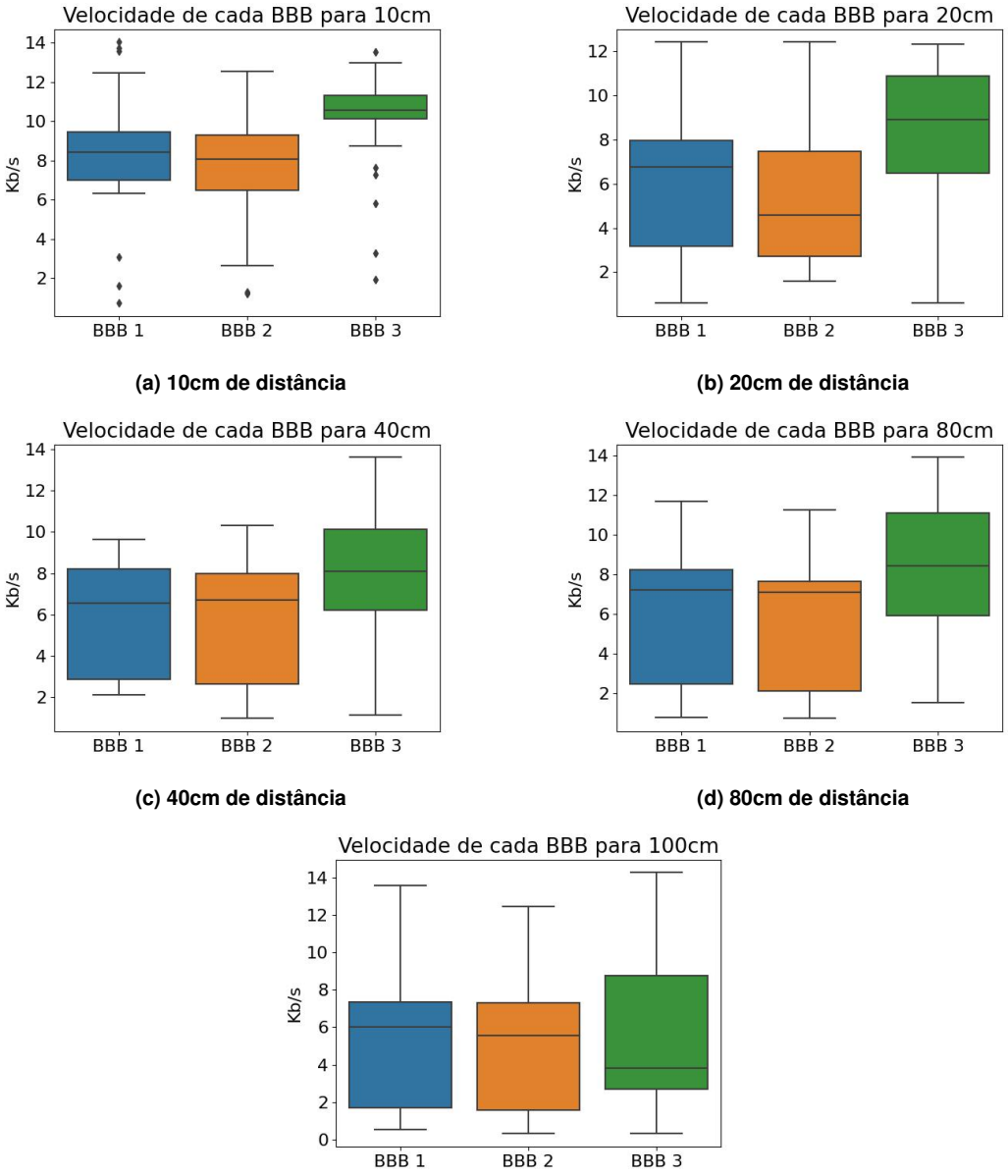
A medição de atraso utiliza o envio de mensagens textuais no grupo *multicast* através de *sockets* de rede UDP. Logo, é possível concluir que o aumento do atraso ocorre de maneira gradativa, à medida que a distância entre as BBBs aumenta, o que acarreta em uma maior distância necessária para a informação viajar e atingir o cliente. Situação semelhante ao que ocorreria em um cenário de comunicação *unicast* (ou seja, de 1 pra 1).

7. Conclusão

Este trabalho apresentou uma solução de um protocolo *multicast* para VLC, implementado na camada de aplicação. A solução apresentada consiste em uma abordagem minimalista e com o foco em apenas garantir que a comunicação de um para *n* ocorresse de maneira satisfatória, além de trazer códigos com alta legibilidade, paralelismo e conceitos de alto nível.

A utilização do protocolo baseado em *multicasting* reduziu a quantidade de envios necessários pelo servidor para atingir todos os clientes pretendidos. Com isso, é possível concluir que em cenários onde alguma redução na quantidade de transmissões diminua a concorrência pelo recurso, aumente a velocidade ou ainda diminua a latência, seria muito interessante possuir suporte a tal regime de transmissão.

Os resultados de perda de pacotes, vazão e atraso, demonstram que não há prejuízos significativos no envio de pacotes através do *multicasting*. Logo, com a utilização do protocolo em um cenário de comunicação por luz visível onde diversos clientes buscam



(a) 10cm de distância

(b) 20cm de distância

(c) 40cm de distância

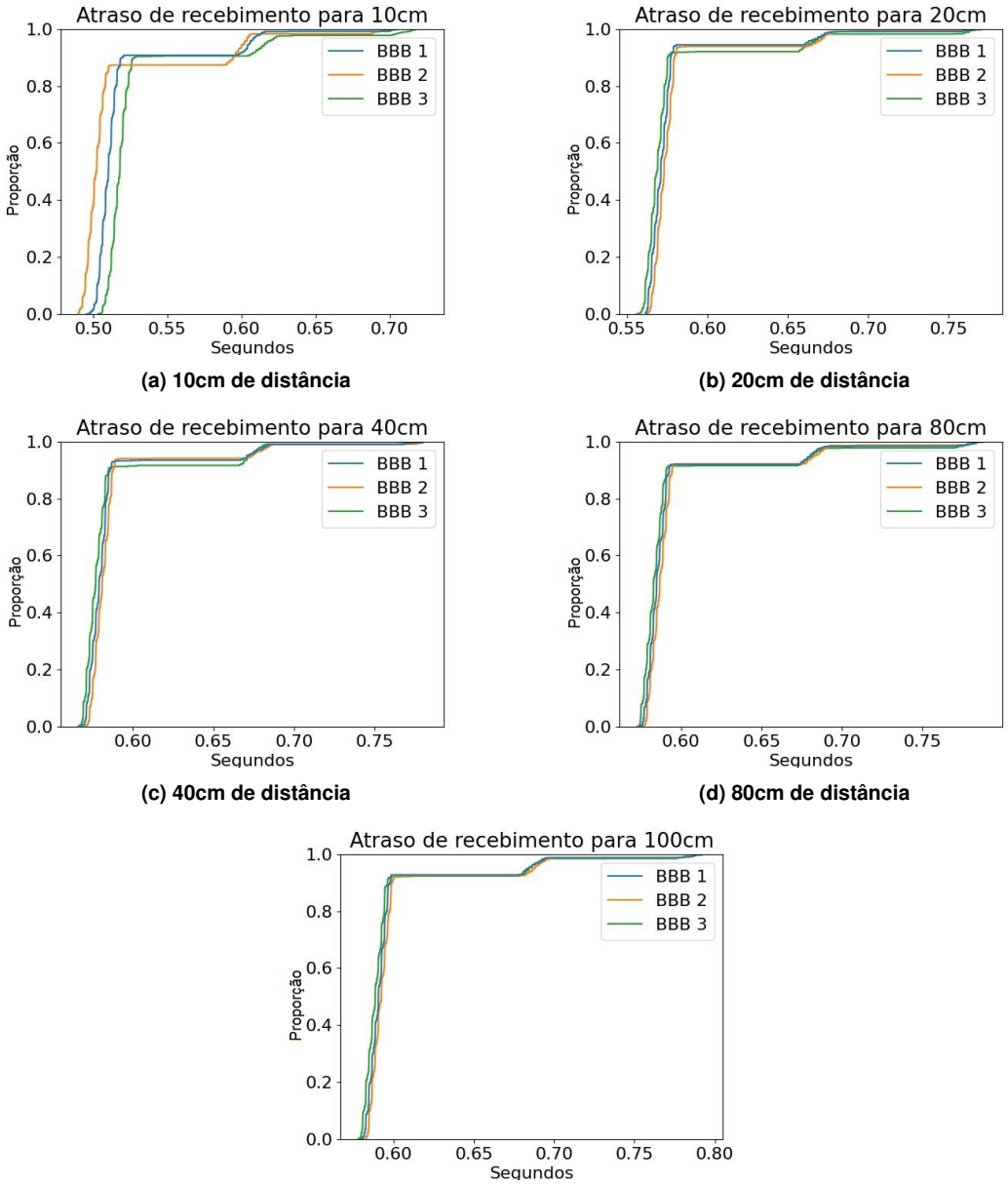
(d) 80cm de distância

(e) 100cm de distância

Figura 6. *Boxplots* de velocidades de transferência

pelo mesmo fecho de informações, o protocolo *multicast* ofereceria apenas vantagens, uma vez que otimizaria a largura de banda disponível, reduzindo a quantidade de envios necessários. Contudo, a desvantagem consiste em que para a utilização deste protocolo, é fundamental o contexto de clientes que necessitam das mesmas informações para operar.

Trabalhos futuros incluem a utilização de uma versão mais atual da plataforma OpenVLC. No presente trabalho, a solução necessitou ser implementada na versão 1.0 da plataforma OpenVLC devido a indisponibilidade de versões mais recentes. Sendo assim, caso uma nova implementação fosse criada em versões mais recentes como na OpenVLC 1.4, melhores resultados poderiam ser explorados, não apenas devido a maiores taxas de transferências e menor latência, mas também devido ao suporte de mais funcionalida-



(a) 10cm de distância

(b) 20cm de distância

(c) 40cm de distância

(d) 80cm de distância

(e) 100cm de distância

Figura 7. *eCDF* do atraso de transferência

des de rede disponíveis, tornando possível a implementação do protocolo em camadas inferiores da pilha de rede do protocolo TCP/IP.

Referências

Eastlake 3rd, D. and Jones, P. (2001). Us secure hash algorithm 1 (sha1). Technical report.

Gomez, A., Shi, K., Quintana, C., Sato, M., Faulkner, G., Thomsen, B. C., and O'Brien, D. (2015). Beyond 100-gb/s indoor wide field-of-view optical wireless communications. *IEEE Photonics Technology Letters*, 27(4):367–370.

Matheus, L., Borges, A., Freire, J., Vieira, L., Vieira, M., and Gnawali, O. (2017). *Comunicação por Luz Visível: Conceitos, Aplicações e Desafios*, pages 247–296.

Matheus, L., Vieira, A., and Vieira, M. (2019). Dyrp-vlc: Um protocolo de roteamento dinâmico para redes de comunicação por luz visível. In *Anaisdo XXXVII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, pages 105–112.

Nayyar, A. and Puri, V. (2016). A comprehensive review of beaglebone technology: Smart board powered by arm. *International Journal of Smart Home*, 10(4):95–108.

Sales, F., Silva, E., Vieira, A., Vieira, M., and Vieira, L. (2021). A proposal of a dynamic routing multicast protocol for visible light communication networks. In *Anais do I Workshop on Security, Privacy and Reliability on Wireless Sensing Networks*.

Wang, Q., Yin, S., Gnawali, O., and Giustiniano, D. (2015). Openvlc1. 0 platform for research in visible light communication networks. In *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*, pages 179–181.

Williamson, B. (2000). *Developing IP multicast networks*, volume 1. Cisco Press.

A. Apêndice

A.1. Pseudocódigos

Algoritmo 1 Script Principal do Servidor

```
1: função MAIN()
2:     EnderecoHost ← 192.169.0.1
3:     PortaTCP ← 5060
4:     EnderecoMulticast ← 224.1.1.5
5:     PortaMulticast ← 50001
6:     Endereco_Grupo_Multicast ← EnderecoMulticast : PortaMulticast
7:     PayloadSize ← 1024
8:     se ListaClientes presente em persistencia então
9:         Recupera ListaClientes
10:    senão
11:        Inicia nova ListaClientes
12:    fim se
13:    NovaThread(Recebe_Requisicoes())
14:    NovaThread(MCAST_Server())
15:    enquanto True faça
16:        se ListaClientes seja alterada então
17:            Imprime ListaClientes
18:        fim se
19:    fim enquanto
20: fim função
```

Algoritmo 2 Função de envio multicast

```
1: função MCAST_SERVER()
2:     MCAST_Socket ← NovoSocket(Tipo ← SocketDatagrama)
3:     enquanto msg != "pare" faça
4:         msg ← EntradaPadrao()
5:         MCAST_Socket.Envia(msg, Endereco_Grupo_Multicast, PayloadSize)
6:     fim enquanto
7:     MCAST_Socket.Fecha()
8: fim função
```

Algoritmo 3 Recebe Requisições

```
1: função RECEBE_REQUISICOES()
2:     TCPSocket ← NovoSocket()
3:     enquanto True faça
4:         conexao ← TCPSocket.AceitaConexao()
5:         Action, Cliente ← conexao.Receber()
6:         se Action ==' E' então
7:             NovaThread(ConectaRequisicaoEntrar())
8:         senão Action ==' Q' então
9:             NovaThread(ConectaRequisicaoSair())
10:        fim se
11:    fim enquanto
12: fim função
```

Algoritmo 4 Funções de Requisições de grupo

```
1: função CONECTAREQUISICAOENTRAR()
2:     JaCadastrado ← False
3:     se Cliente presente em ListaClientes então
4:         JaCadastrado ← True
5:     fim se
6:     conexao.Envia(Endereco_Grupo_Multicast)
7:     enquanto True faça
8:         Message ← conexao.Recebe()
9:         se Message == 'OK' e Not JaCadastrado então
10:            ListaClientes.Adiciona(Cliente)
11:            pare enquanto
12:        senão
13:            pare enquanto
14:        fim se
15:    fim enquanto
16:    conexao.Fecha()
17:    Thread.Finaliza()
18: fim função
19: função CONECTAREQUISICAOSAIR()
20:    se Cliente not presente em ListaClientes então
21:        conexao.Envia("Erro!")
22:        conexao.Fecha()
23:        Thread.Finaliza()
24:    fim se
25:    ListaClientes.Remove(Cliente)
26:    conexao.Envia("Sucesso!")
27:    conexao.Fecha()
28:    Thread.Finaliza()
29: fim função
```

Algoritmo 5 Script principal do Cliente

```
1: função EnviaRequisicaoEntrar()
2:     se EnderecoMulticast presente em persistencia então
3:         Recupera EnderecoMulticast
4:     senão
5:         SocketCliente ← NovoSocket()
6:         conexaoCliente ← SocketCliente.Conecta(EnderecoHost, PortaTCP)
7:         conexaoCliente.Envia("E")
8:         EnderecoMulticast ← conexaoCliente.Recebe()
9:         se EnderecoMulticast então
10:            conexaoCliente.Envia("OK")
11:            Salva Persistente EnderecoMulticast
12:        fim se
13:    fim se
14: fim função
15: função Main()
16:    EnderecoHost ← 192.169.0.1
17:    PortaTCP ← 5060
18:    PayloadSize ← 1024
19:    EnviaRequisicaoEntrar()
20:    SocketClienteMulticast ← NovoSocket(Tipo ← SocketDatagrama)
21:    SocketClienteMulticast.LigueA(EnderecoMulticast)
22:    enquanto msg != "pare" faça
23:        msg ← SocketClienteMulticast.Recebe(PayloadSize)
24:        Imprime msg
25:    fim enquanto
26:    SocketClienteMulticast.Fecha()
27: fim função
```

Algoritmo 6 Script de saída de grupo do Cliente

```
1: função EnviaRequisicaoSaida()
2:     EnderecoHost ← 192.169.0.1
3:     PortaTCP ← 5060
4:     SocketCliente ← NovoSocket()
5:     conexaoCliente ← SocketCliente.Conecta(EnderecoHost, PortaTCP)
6:     conexaoCliente.Envia("Q")
7:     se conexaoCliente.Recebe() == "Sucesso" então
8:         Apaga EnderecoMulticast persistente
9:         Imprime Saida efetuada!
10:    senão
11:        Apaga EnderecoMulticast persistente
12:        Imprime Erro ao Sair! Memoria persistente apagada.
13:    fim se
14: fim função
```