SBRC 2019

Gramado | RS

XXXVII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos

6 a 10 de maio

Gramado | RS

ANAIS DO WTG 2019

sbrc2019.sbc.org.br

SBRC 2019

Gramado | RS

XXXVII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos

6 a 10 de maio

Gramado | RS

ANAIS DO WTG

sbrc2019.sbc.org.br

EDITORA
Sociedade Brasileira de Computação (SBC)

ORGANIZAÇÃO
Alberto Egon Schaeffer Filho (UFRGS)
Weverton Luis da Costa Cordeiro (UFRGS)
Stênio Fernandes (UFPE)
Antônio Meneguette (IFSP)
Lourenço Pereira (ITA)

REALIZAÇÃO
Sociedade Brasileira de Computação (SBC)
Instituto de Informática - Universidade Federal do Rio Grande do Sul (UFRGS)
Laboratório Nacional de Redes de Computadores (LARC)

Capa: Dórian Fogliatto
Produção Editorial: Carlos Raniery P. dos Santos (UFSM)

Cópias Adicionais:
Sociedade Brasileira de Computação (SBC)
Av. Bento Gonçalves, 9500- Setor 4 - Prédio 43.412 - Sala 219
Bairro Agronomia - CEP 91.509-900 - Porto Alegre - RS
Fone: (51) 3308-6835
E-mail: sbc@sbc.org.br

II Workshop de Trabalhos de Iniciação Científica e Graduação (2: 2019: Gramado, RS).

Anais / II Workshop de Trabalhos de Iniciação Científica e Graduação - WTG; organizado por Alberto Egon Schaeffer Filho, Weverton Luis da Costa Cordeiro, Stênio Fernandes, Rodolfo Meneguette, Lourenço Pereira - Porto Alegre: SBC, 2019

51 p. il. 21 cm.
Vários autores
Inclui bibliografias

1. Redes de Computadores. 2. Sistemas Distribuídos. I. Schaeffer-Filho, Alberto Egon II. Cordeiro, Weverton Luis da Costa III. Fernandes, Stênio IV. Meneguette, Rodolfo V. Pereira, Lourenço VI Título.

Sociedade Brasileira de Computação

Contato
Av. Bento Gonçalves, 9500
Setor 4 - Prédio 43.412 - Sala 219
Bairro Agronomia
91.509-900 – Porto Alegre RS
CNPJ: 29.532.264/0001-78
http://www.sbc.org.br

Laboratório Nacional de Redes de Computadores (LARC)

Diretor do Conselho Técnico-Científico
Paulo André da Silva Gonçalves (UFPE)

Vice-Diretora do Conselho Técnico-Científico
Rossana Maria de Castro Andrade (UFC)

Diretor Executivo
Ronaldo Alves Ferreira (UFMS)

Vice-Diretor Executivo
Danielo Gonçalves Gomes (UFC)

Membros Institucionais
SESU/MEC, INPE/MCT, UFRGS, UFMG, UFPE, UFCG (ex-UFPB Campus Campina Grande), UFRJ, USP, PUC-Rio, UNICAMP, LNCC, IME, UFSC, UTFPR, UFC, UFF, UFSCar, IFCE (CEFET-CE), UFRN, UFES, UFBA, UNIFACS, UECE, UFPR, UFPA, UFAM, UFABC, PUCPR, UFMS, UnB, PUC-RS, PUCMG, UNIRIO, UFS e UFU.

Contato
Universidade Federal de Mato Grosso do Sul (UFMS)
Faculdade de Computação
Caixa Postal 549
CEP 79.070-900 Campo Grande - MS - Brasil
http://www.larc.org.br

Organização do SBRC 2019

Coordenadores Gerais
Alberto Egon Schaeffer Filho (UFRGS)
Weverton Luis da Costa Cordeiro (UFRGS)

Coordenadores do Comitê de Programa
Antônio Jorge Gomes Abelém (UFPA)
Fabíola Gonçalves Pereira Greve (UFBA)

Coordenador de Palestras e Tutoriais
Ítalo Cunha (UFMG)

Coordenador de Painéis e Debates
Artur Ziviani (LNCC)

Coordenador de Minicursos
Miguel Elias Mitre Campista (UFRJ)

Coordenador de Workshops
Stênio Fernandes (UFPE)

Coordenador do Salão de Ferramentas
Leandro Villas (UNICAMP)

Coordenador do Concurso de Teses e Dissertações
Daniel Fernandes Macedo (UFMG)

Coordenadores do Hackaton
Raquel Lopes (UFMG)
Luis Carlos de Bona (UFPR)

Comitê de Organização Local
Avelino Zorzo (PUCRS)
Carlos Raniery Paula dos Santos (UFSM)
Cristiano Bonato Both (Unisinos)
Guilherme Rodrigues (IFSUL Charqueadas)
Jéferson Campos Nobre (UFRGS)
Juliano Wickboldt (UFRGS)
Marcelo Caggiani Luizelli (Unipampa)
Marcelo da Silva Conterato (PUCRS)
Rafael Pereira Esteves (IFRS)
Rodrigo Mansilha (Unipampa)
Tiago Ferreto (PUCRS)
Vinícius Guimarães (IFSUL Charquadas)

Comite Consultivo
Fabíola Gonçalves Pereira Greve (UFBA)

Paulo André da Silva Gonçalves (UFPE)
Fábio Luciano Verdi (UFSCAR)
Jó Ueyama (USP)
Antônio Jorge Gomes Abelém (UFPA)
Eduardo Cerqueira (UFPA)
Luiz Fernando Bittencourt (Unicamp)
Rossana Andrade (UFC)
Michele Nogueira (UFPR)
Edmundo Madeira (UNICAMP)

Mensagem dos Coordenadores Gerais

É com grande alegria e orgulho que, após 9 anos, estamos trazendo o Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC) novamente para a cidade de Gramado. O SBRC 2019 acontece em um ano que é especial por várias razões: primeiro, o aniversário de 50 anos da ARPANET, a primeira rede a implementar o conjunto de protocolos TCP/IP, que acabaram se estabelecendo mais tarde como os fundamentos da Internet; os 30 anos da invenção da World Wide Web pelo cientista Inglês Tim Berners-Lee; a comemoração dos 30 anos do .br (country code top-level domain), o qual tem atualmente mais de 4 milhões de domínios registrados; por fim, nesse ano comemora-se os 30 anos da Rede Nacional de Ensino e Pesquisa (RNP), que iniciou em 1989 como um projeto de pesquisa. É, portanto, um imenso privilégio termos a oportunidade de sediar o SBRC em um ano que carrega tanto significado para a área de redes de computadores e sistemas distribuídos.

A 37a edição do evento se apoia no histórico de sucesso do SBRC, que tradicionalmente inclui sessões técnicas, minicursos, painéis e debates, workshops, salão de ferramentas e palestras. Além de contar novamente com um Hackathon e com um Concurso de Teses e Dissertações, teremos muitas novidades no SBRC 2019, incluindo as reuniões de mentoria, o evento MUSAS e o workshop de estudantes latino-americanos. As reuniões de mentoria permitem oportunizar conversas 1-a-1 entre estudantes de pós-graduação e pesquisadores de excelência do Brasil e do exterior. O MUSAS (MUlheres em redeS de computadores e sistemAs diStribuídos) visa fomentar conexões entre mulheres atuando nas áreas de redes de computadores e de sistemas distribuídos, no Brasil e no exterior. Já o workshop de estudantes latino-americanos (LANCOMM) serve como um ponto de encontro para estudantes da região apresentarem e discutirem o andamento de suas pesquisas, e obterem feedback construtivo de pesquisadores estrangeiros de grande prestígio.

Neste ano, a trilha principal do SBRC recebeu 228 submissões completas e que entraram no processo de avaliação. Todos os artigos receberam pelo menos 3 revisões, e após um rigoroso processo seletivo, 80 artigos foram aceitos para publicação e organizados em 23 sessões técnicas. O Salão de Ferramentas selecionou 12 trabalhos que a serem demonstrados ao longo do SBRC 2019. Por sua vez, o Concurso de Teses e Dissertações selecionou 8 dissertações de mestrado e 8 teses de doutorado para apresentação durante o evento. Além disso, esse ano o evento conta com 4 palestrantes internacionais, 3 painéis, 1 tutorial, 5 minicursos e 11 workshops.

A organização de um evento com o porte do SBRC é um processo longo, que demanda muita energia e empenho. O evento desse ano só foi possível graças ao apoio e suporte incondicional de muitos grupos de pessoas e instituições. Agradecemos em particular o apoio da SBC, do LARC, do Comitê Consultivo do SBRC e da Comissão Especial de Redes de Computadores e Sistemas Distribuídos da SBC. O evento contou com o apoio do Comitê Gestor da Internet no Brasil (CGI.br) e do Núcleo de Informação e Coordenação do Ponto BR (NIC.br), da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), do Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), da Fundação de Amparo à Pesquisa do Estado do Rio Grande do Sul (FAPERGS) e da ACM SIGCOMM. Contamos novamente também com

o apoio institucional da Rede Nacional de Ensino e Pesquisa (RNP). Além disso, o SBRC 2019 recebeu o apoio de importantes patrocinadores, incluindo SAP, HUAWEI, DATACOM, Google, Bedu.tech, BRDigital e AdylNet. Também agradecemos à Universidade Federal do Rio Grande do Sul (UFRGS), e em especial ao Instituto de Informática (INF), pelo suporte incondicional.

Agradecemos a todas as pessoas que trabalharam diretamente na organização do evento. Obrigado pelo excelente trabalho de todos os membros do comitê de organização: Antônio Jorge Gomes Abelém/UFPA e Fabíola Gonçalves Pereira Greve/UFBA (Coordenadores do Comitê de Programa), Stênio Fernandes/UFPE (Coordenador de Workshops), Ítalo Cunha/UFMG (Coordenador de Palestras, Tutoriais e Mentoria), Artur Ziviani/LNCC (Coordenador de Painéis), Miguel Elias Mitre Campista/UFRJ (Coordenador de Minicursos), Leandro Villas/UNICAMP (Coordenador do Salão de Ferramentas), Daniel Fernandes Macedo/UFMG (Coordenador do Concurso de Teses e Dissertações), e Raquel Lopes/UFCG e Luis Carlos De Bona/UFPR (Coordenadores do Hackathon). Agradecemos também aos membros do nosso comitê local, que nos ajudaram na operacionalização de diversas tarefas relacionadas à organização: Avelino Zorzo/PUCRS, Carlos Raniery Paula dos Santos/UFSM, Cristiano Bonato Both/Unisinos, Guilherme Rodrigues/IFSUL Charqueadas, Jéferson Campos Nobre/UFRGS, Juliano Wickboldt/UFRGS, Marcelo Caggiani Luizelli/Unipampa, Marcelo da Silva Conterato/PUCRS, Rafael Esteves/IFRS, Rodrigo Mansilha/Unipampa, Tiago Ferreto/PUCRS, e Vinícius Guimarães/IFSUL Charqueadas. Agradecemos ao Luis Otávio Luz Soares, técnico administrativo do INF/UFRGS, pela imensa ajuda nos mais variados aspectos relacionados à organização do evento, assim como pelo apoio de Leandro Disconzi Vieira e Carlos Alberto da Silveira Junior, também técnicos administrativos da UFRGS. Por fim, agradecemos à nossa equipe local, formada por alunos de graduação, pós-graduação e pós-doutorando da UFRGS, pela dedicação e pronta ajuda sempre que precisamos: Arthur Selle Jacobs, Augusto Zanella Bardini, Bruno Dalmazo, Fernanda da Silva Bonetti, Guilherme Bueno de Oliveira, Guilherme Rotth Zibetti, Isadora Pedrini Possebon, Leonardo Lauryel, Libardo Andrey Quintero González, Lucas Bondan, Lucas Castanheira, Luciano Zembruzki, Mateus Saquetti, Jonatas Marques, Rafael Hengen Ribeiro, e Ricardo Parizotto.

Desejamos a todos um ótimo evento e uma semana produtiva e com muitas trocas de ideias em Gramado.

Alberto Egon Schaeffer Filho e Weverton Luis da Costa Cordeiro
Coordenadores Gerais do SBRC 2019

Mensagem do Coordenador de Workshops

Os Workshops do SBRC são uma importante oportunidade para aprofundamento no conhecimento de temas especializados ou emergentes na nossa comunidade científica. Como em anos anteriores, a coordenação manteve a *Chamada de Propostas de Workshops do SBRC*, que tem estimulado a comunidade brasileira de Redes e Sistemas Distribuídos a discutir a viabilidade de workshops com temas de pesquisa mais tradicionais, bem como estimular o debate para adoção de eventos que tratam de temas mais emergentes.

Tivemos uma diversidade salutar no Comitê de Avaliação em termos de localização geográfica e experiência, gerando, portanto, opiniões complementares sobre as propostas submetidas. Consideramos todas as propostas de altíssima qualidade, com temas, focos e escopo diversos. Nesta edição de 2019, buscamos equilibrar os benefícios trazidos aos participantes com a capacidade de alocação de espaços no evento. Desta forma, foram selecionados 10 Workshops de alta qualidade, além do tradicional Workshop da RNP (*WRNP*). Dentre as propostas aceitas, oito são reedições de workshops tradicionais do SBRC, a saber: **Gerência e Operação de Redes e Serviços** (*WGRS*), **Testes e Tolerância a Falhas** (*WTF*), **Pesquisa Experimental da Internet do Futuro** (*WPEIF*), **Segurança Cibernética em Dispositivos Conectados** (*WSCDC*), **Trabalhos de Iniciação Científica e Graduação** (*WTICG*), **Blockchain: Teoria, Tecnologias e Aplicações** (*WBlockchain*), **Clouds e Aplicações** (*WCGA*), e **Computação Urbana** (*CoUrb*). Como novidade, teremos dois novos workshops, a saber: o Workshop de **Teoria, Tecnologias e Aplicações de Slicing para Infraestruturas Softwarizadas** (*WSlice*) e o **Latin American Student Workshop on Data Communication Networks** (*LANCOMM Student Workshop*), que tem como público-alvo principal os alunos de pós-graduação e pesquisadores da América Latina, além de estreitar os relacionamentos da nossa comunidade com a Association for Computing Machinery (*ACM*).

Como coordenador dos Workshops do SBRC 2019, gostaria de agradecer a todos os envolvidos na seleção das propostas. Primeiramente aos coordenadores gerais do SBRC 2019, Weverton Cordeiro (UFRGS) e Alberto Egon Schaeffer Filho (UFRGS), pelo convite para a coordenação desta chamada de trabalhos, além de todo o apoio necessário para sua condução apropriada. Agradeço também a todos os membros do comitê de avaliação pelo imenso esforço nas revisões e discussões de alta qualidade de todas as propostas submetidas. Por fim, agradeço aos coordenadores dos workshops aceitos, pela dedicação no cumprimento dos prazos e na condução dos trabalhos internos de seleção dos artigos para seus respectivos eventos, mantendo a alta qualidade geral do SBRC. Os inscritos nos diversos workshops esperam ansiosamente pelas apresentações dos trabalhos e pelas frutíferas discussões que serão geradas.

Stênio Fernandes
Coordenador dos Workshops do SBRC 2019

Comitê de Programa

- Alberto Schaeffer-Filho (Federal University of Rio Grande do Sul)
- Alex Vieira (Universidade Federal de Juiz de Fora)
- André Drummond (Universidade de Brasília)
- Bruno Cesar (Universidade de São Paulo)
- Bruno Kimura (UNIFESP)
- Daniel Batista (IME - USP)
- Daniel Fernandes Macedo (Universidade Federal de Minas Gerais)
- Daniel Guidoni (Universidade Federal de São João del-Rei)
- Danielo G. Gomes (Universidade Federal do Ceará)
- Edmundo Madeira (UNICAMP)
- Geraldo Pereira (CiC - UnB)
- Jussara Almeida (DCC-UFMG)
- Kelvin Dias (UFPE)
- Lourenco Pereira Jr (Instituto Tecnológico de Aeronáutica - ITA)
- Luis Nakamura (IFSP)
- Luis Carlos De Bona (University Federal of Parana)
- Luiz Fernando Bittencourt (UNICAMP)
- Luiz Filipe Vieira (UFMG)
- Roberto Rigolin F Lopes (Fraunhofer FKIE)
- Roberto Sadao Yokoyama (Universidade Tecnológica Federal do Paraná)
- Robson De Grande (brocku University)
- Rodolfo Coutinho (UFMG)
- Rodolfo Meneguette (Instituto Federal de Educação, Ciência e Tecnologia de São Paulo)
- Rodrigo Miani (Universidade Federal de Uberlândia)
- Silvana Rossetto (UFRJ)
- Tiago Ferreto (PUCRS)

Sumário

II Workshop de Trabalhos de Iniciação Científica e Graduação
SBRC 2019
Sessão Técnica 1

Análise de Mecanismos de Serverless Computing em Ambientes de Nuvens Computacionais

Matheus N. da Silva, Marcus Carvalho

Departamento de Ciências Exatas – Universidade Federal da Paraíba (UFPB), Campus IV – Av. Santa Elizabeth, S/N, Centro – CEP 58297-000 – Rio Tinto – PB – Brazil

{matheus.nicolas, marcuswac}@dcx.ufpb.br

Abstract. *The serverless computing model is trending nowadays, because of its easy adoption and possibility of reducing costs. However, as users are not in charge of managing servers, they may face performance issues related to coldstarts, when the provider disables an idle application and new requests have to wait until it is deployed in a server and enabled again. The objective of this work is to analyze coldstarts in serverless computing, aiming to understand the time an application stays idle until the provider disables it, the overhead implied by coldstarts on response times and the impact of application's memory size on these metrics.*

Resumo. *O modelo de serverless computing tem se tornado tendência nos últimos anos, devido à facilidade de adoção e possibilidade de redução de custos. Porém, como usuários não têm controle sobre os servidores, eles podem ter problemas de desempenho relacionados a coldstarts, quando o provedor torna inativa uma aplicação ociosa e novas requisições precisam esperar até que ela seja implantada em um servidor e se torne ativa novamente. O objetivo deste trabalho é analisar o coldstart em serverless computing, buscando entender o tempo de ociosidade de uma aplicação até o provedor torná-la inativa, o overhead imposto pelo coldstart no tempo de resposta e se a quantidade de memória alocada para a aplicação afeta essas métricas.*

1. Introdução

A computação em nuvem permite a seus usuários reduzir despesas na aquisição de hardware e em sua manutenção, investindo por outro lado em serviços terceirizados pelos provedores. Inicialmente, ainda cabia ao usuário da nuvem realizar tarefas operacionais como: criar instâncias (servidores) e determinar sua capacidade (e.g. CPU, memória e disco); implantar a aplicação nos servidores; e decidir quando e quantas instâncias estarão em execução com base na demanda da aplicação [Savage 2018].

O *serverless computing* surgiu como alternativa a este modelo tradicional, no qual os usuários não precisam gerenciar os servidores que rodam suas aplicações. Neste novo modelo, o usuário pode focar de fato na sua aplicação, deixando para o provedor a implantação da aplicação em instâncias, escalabilidade e tolerância a falhas. O usuário paga apenas pelo uso de suas aplicações com base na demanda e não mais pelo tempo em que os servidores estiveram rodando [Fowler 2012].

Um dos principais modelos associados ao *serverless* é o *FaaS* (ou função como serviço, do inglês *Function as a Service*). Neste modelo, o usuário carrega o código da função no provedor, que é responsável por toda a gerência de servidores, implantação da função em um ambiente de execução e provisionamento automático de recursos para atender à demanda. A execução da função é acionada por eventos definidos pelo usuário no provedor – por exemplo, uma requisição HTTP que chega em um certo caminho pode acionar uma função que faz o seu processamento [Fowler 2012]. Provedores de *FaaS* da atualidade incluem Amazon AWS Lambda[1] e Google Cloud Functions[2], que dão

[1] https://aws.amazon.com/pt/lambda/

suporte a funções implementadas em linguagens de programação e ambientes de execução como: Node.js, Python, Go, Java, etc.

Um dos diferenciais do modelo *serverless* é que, para reduzir custos, o provedor pode desativar aplicações após um tempo de ociosidade (i.e. em períodos sem demanda). Porém, para processar requisições que chegam para uma aplicação inativa, o provedor precisa antes implantar esta aplicação em um servidor e iniciá-la em um ambiente de execução. Este tempo de inicialização de aplicações inativas em um ambiente *serverless* é chamado de *coldstart*, que pode impactar o desempenho das aplicações [Baldini et al 2016].

O objetivo geral deste trabalho é investigar o *coldstart* em um ambiente de *serverless computing,* analisando o seu impacto no tempo de resposta das aplicações e os fatores que podem afetar a sua frequência e duração. Para isto, foram realizados experimentos de medição em um provedor de *FaaS,* usando funções e cargas sintéticas. Como objetivos específicos, buscou-se responder às seguintes perguntas de pesquisa:

- Com quanto tempo de ociosidade uma função é desativada pelo provedor, tendo como consequência o *coldstart* em requisições que vierem a seguir?
- Qual o tempo adicional (*overhead*) que o *coldstart* impõe no tempo de resposta das requisições, comparado a requisições que não passam por *coldstart*?
- Qual o impacto da quantidade de memória alocada para a função no *overhead* do *coldstart* e no intervalo de ociosidade para que ocorra um *coldstart*.

O restante do artigo está estruturado da seguinte forma: a seção 2 apresenta os trabalhos relacionados; a seção 3 apresenta a metodologia da avaliação; a seção 4 apresenta os resultados; e a seção 5 discute as conclusões e trabalhos futuros.

2. Trabalhos relacionados

Baldini et al. (2016) faz um levantamento das tendências e problemas em aberto na área de *serverless computing*. Um dos desafios apontados é exatamente o fenômeno do *coldstart*, pois ao mesmo tempo que pode-se reduzir custos escalando uma função para zero servidores, desativando-a, pode-se também piorar seu desempenho por causa do *coldstart*. Por isso, seu trabalho aponta como crítico o estudo de técnicas para minimizar o *coldstart,* ao mesmo tempo que aplicações ociosas ainda possam ser desativadas. Apesar da relevância do tema, encontramos poucas publicações acadêmicas que realizam análises voltadas ao *coldstart* durante nossa revisão da literatura. Desta forma, o trabalho corrente visa preencher um pouco esta lacuna, contribuindo com uma avaliação de desempenho de *coldstarts* em um ambiente de *serverless computing*.

Os trabalhos encontrados mais relacionados foram os de Cui (2017a) e Cui (2017b), que apesar de não terem sido publicações acadêmicas, apresentam uma metodologia clara e resultados bem descritos. Em seu primeiro estudo, Cui (2017a) propõe uma metodologia de detecção de *coldstarts* e analisa um provedor de *FaaS* (AWS Lambda) quanto tempo uma função fica ociosa até que o provedor torne-a inativa, resultando em seguida em um *coldstart*. Este estudo concluiu que o AWS Lambda tornou uma função inativa após aproximadamente 40 a 60 minutos de ociosidade, mas que eventualmente podiam ocorrer *coldstarts* antes desse período de ociosidade, para liberar recursos. Também observou-se que funções com mais memória alocada tendem a ser desativadas com um tempo menor de ociosidade. Com base nesta análise inicial, nosso trabalho buscou reproduzir os experimentos de Cui (2017a) para medir quando os *coldstarts* ocorrem e, além disso, expandir a análise para medir o *overhead* imposto pelo *coldstart* no tempo de resposta das funções, dependendo da quantidade de memória alocada para ela.

---

[2] https://cloud.google.com/functions/

Em outro estudo, Cui (2017b) mediu como a linguagem de programação, o tamanho do código e a quantodade de memória alocada afetam o *coldstart*. Em seus resultados, observou-se que: linguagens de programação diferentes podem apresentar diferenças significativas no *overhead* de *coldstarts*, sendo os *coldstarts* para *Python* e *NodeJS* bem menores que para *Java* e *C#*; quanto mais memória alocada para a função, menor a duração do *coldstart*; quanto maior o código, menor o *coldstart*. O nosso trabalho buscou fazer uma análise mais aprofundada da duração e frequência de *coldstarts* para diferentes tamanhos de memória, focando no framework *NodeJS*. Ao contrário do trabalho de Cui (2017b), observamos que a relação do tempo do *coldstart* com o tamanho da memória alocada não é linear, o que será discutido nos resultados.

3. Metodologia

Neste trabalho, foram realizados experimentos de medição para analisar a frequência e duração de *coldstarts* em um ambiente de *serverless computing*. O experimento foi baseado na metodologia usada por Cui (2017a) para criação de funções, medição de desempenho e geração de carga.

O ambiente de *FaaS* do AWS Lambda foi usado para a medição de desempenho e análise de *coldstarts*. Uma função lambda chamada *system-under-test* foi criada no ambiente de execução NodeJS, sendo configurada para executar ao chegarem requisições HTTP em um caminho específico. Esta função informa como retorno se a requisição sendo executada está se deparando ou não com um *coldstart* da seguinte forma: em seu primeiro acesso, a função sendo executada armazena uma *flag* em memória indicando que já foi acessada; se um acesso posterior à função não observar tal *flag* ativada, é porque a função passou por um *coldstart* e teve que ser iniciada novamente, perdendo seu estado prévio [Cui 2017a]. Com o intuito de avaliar o impacto da alocação de memória no *coldstart*, foram executados diferentes cenários de alocação de memória para a função *system-under-test*, com os seguintes valores: 128MB, 256MB, 512MB, 1024MB e 2048MB.

A geração de carga e medição de desempenho foi realizada utilizando o serviço AWS Step Functions[3], que possibilita a criação de um fluxo de execução de funções e a gravação de medições em serviços de monitoramento. A Figura 1, demonstra as funções criadas e o fluxo de execução para a geração de carga periódica dos experimentos.

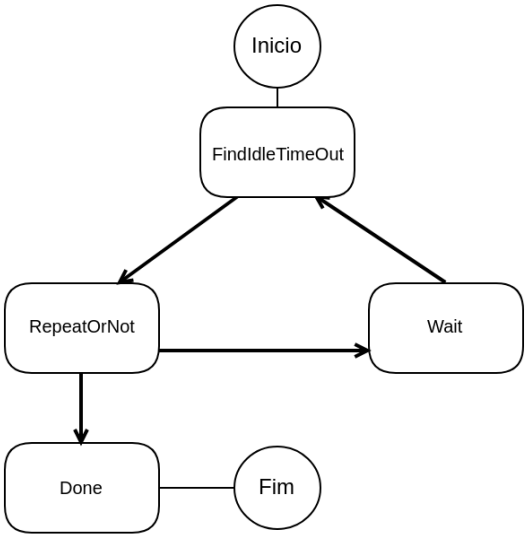


Figura 1. Fluxo de execução de funções de geração de carga dos experimentos.

[3] https://aws.amazon.com/pt/step-functions/

Durante a execução de um experimento, o fluxo de execução mantém uma máquina de estados com os três atributos abaixo:

- *target:* função alvo a ser chamada pelo gerador de carga e para a qual o desempenho será medido;
- *coldstarts:* contador da quantidade de *coldstarts* consecutivos encontrados*;*
- *wait*: intervalo de tempo entre requisições submetidas pelo gerador de carga.

A função *FindIdleTimeout* é responsável por gerar a carga e medir o tempo de resposta, enviando requisições periodicamente à função *system-under-test* informada no parâmetro *target*. Cada chamada à função *system-under-test* retorna o status do *coldstart* para aquela chamada: se houve *colstart*, o parâmetro *coldstarts* do experimento é incrementado. Em seguida, o fluxo do experimento passa para a função *RepeatOrNot*.

A função *RepeatOrNot* verifica se o experimento chegou ao fim, ou se ainda serão necessários mais testes. Além disso, ela pode ajustar o parâmetro *wait* para determinar o tempo de espera até a próxima requisição ser gerada. Um dos objetivos do experimento é saber após quanto tempo de ociosidade o provedor torna a função inativa – ou seja, quando os *coldstarts* começam a acontecer sistematicamente. Nos nossos experimentos, o parâmetro *wait* é iniciado com o valor de 10 minutos, sendo incrementado ao longo do experimento até que seja encontrado o tempo de ociosidade que causa *coldstarts* de forma sistemática. Como *coldstarts* inesperados podem acontecer mesmo com pouco tempo de ociosidade, considera-se que o tempo de ociosidade sistemático para tornar a função inativa é encontrado quando 10 *coldstarts* consecutivos são observados. Desta forma, se a função *RepeatOrNot* verificar que a variável *coldstarts* chegou a 10, o fluxo do experimento segue para o estado *Done* para ser finalizado. Caso contrário, se não houve *coldstart* na última requisição, tenta-se um tempo de ociosidade maior incrementando o atributo *wait* em 1 minuto e zerando o contador de *coldstarts* para este novo período. No fim, o fluxo passa para a função *Wait*.

A função *Wait* representa um simples estado de espera, onde a variável *wait* é observada para saber quantos minutos o fluxo de execução do experimento ficará aguardando naquele estado, até voltar novamente à função *FindIdleTimeout* para gerar mais uma requisição e continuar o ciclo.

Esse fluxo de execução do experimento foi rodado uma vez para cada cenário de alocação de memória, um cenário de cada vez. A execução de todos os cenários durou aproximadamente um dia para executar por completo, devido as várias rodadas necessárias para encontrar o tempo de ociosidade *(wait)* que cause 10 *coldstarts* consecutivos.

As métricas avaliadas no experimento foram:

- *tempo máximo ocioso*: tempo máximo que uma função permanece ociosa, sem receber requisições, até ser desativada pelo provedor. Essa métrica foi calculada como o maior valor da variável *wait*, para o qual foram observados 10 *coldstarts* consecutivos.
- *tempo de resposta das requisições*: tempo observado pelo gerador de carga entre a submissão de uma requisição à função *system-under-test* e o recebimento de sua resposta. Associada a esta medição também está a identificação da requisição enviada ter ou não passado por um *coldstart*.

A próxima seção apresenta os resultados da avaliação, discutindo as métricas para cada cenário descrito na metodologia.

4. Resultados

A Tabela 1 apresenta os resultados do tempo máximo ocioso para diferentes cenários de memória alocada para a função. Observa-se que, nos experimentos, o AWS Lambda desativou a função de forma sistemática após um tempo de ociosidade que variou de 30 a 42 minutos. O tempo máximo ocioso observado nos experimentos foram menores do que os encontrados por Cui (2017a), 1 ano antes, que reportou valores entre 48 e 63 minutos para os mesmos cenários de memória alocada. Este resultado sugere que o tempo de ociosidade para desativar uma função não é constante e que, além do tamanho da memória, ele pode depender de fatores externos como a carga do provedor. Apesar de Cui (2017a) sugerir que existe um padrão de quanto mais memória alocada, menor o tempo máximo ocioso, este fenômeno não foi observado nos nossos experimentos. O tempo máximo ocioso de fato variou ao mudar o total de memória alocada, mas não conseguimos observar nenhum padrão linear. Uma quantidade maior de experimentos devem ser realizados, em diferentes dias e horas, para tentar capturar apenas o efeito da alocação de memória no tempo máximo de ociosidade.

Tabela 1. Tempo máximo ocioso da função para o qual acontecem *coldstarts* sistemáticos, para diferentes cenários de memória alocada.

Total de memória alocada (MB)	Tempo máximo ocioso (minutos)
128	30
256	42
512	30
1024	30
2048	40

A Figura 2 apresenta o tempo de resposta das requisições ao variar o intervalo entre requisições *(wait)*, em diferentes cenários de memória alocada para a função. Para cada requisição, também é identificado se a mesma passou por um *coldstart* (em azul) ou se não passou por *coldstart* (em vermelho). Nota-se que sempre há um *coldstart* no intervalo inicial entre requisições, de 10 minutos. Isto acontece porque a primeira requisição sempre passa por um *coldstart*, pois ela que ativa a função pela primeira vez. Também pode-se notar claramente que o tempo de execução com *coldstart* é significativamente maior do que o tempo de resposta sem *coldstart*, reforçando a importância de minimizar ao máximo *coldstart*. Também se observa que os *coldstarts* tendem a aparecer aproximadamente após 25 minutos de ociosidade, mas nem sempre a função é desativada de forma sistemática em um intervalo fixo. Uma hipótese é que em momentos de sobrecarga no provedor, ele pode decidir desativar funções que estão ociosas a mais de 25 minutos, dando prioridade a funções que estão ociosas a mais tempo.

A função de maior alocação de memória (2048MB) foi a que mais apresentou *coldstarts* antes de atingir os 10 consecutivos, havendo: 5 *coldstarts* com ociosidade de 26 minutos; 2 *coldstarts* com ociosidade de 35 minutos; e 10 *coldstarts* com ociosidades de 10 minutos, quando considerou-se o tempo de ociosidade sistemática. Uma outra hipótese para estes *coldstarts* mais frequentes em intervalos mais curtos é que o provedor pode tentar desativar primeiro funções que estão ociosas por algum tempo e que possuem mais memória alocada, para liberar mais recursos em períodos de sobrecarga no servidor.

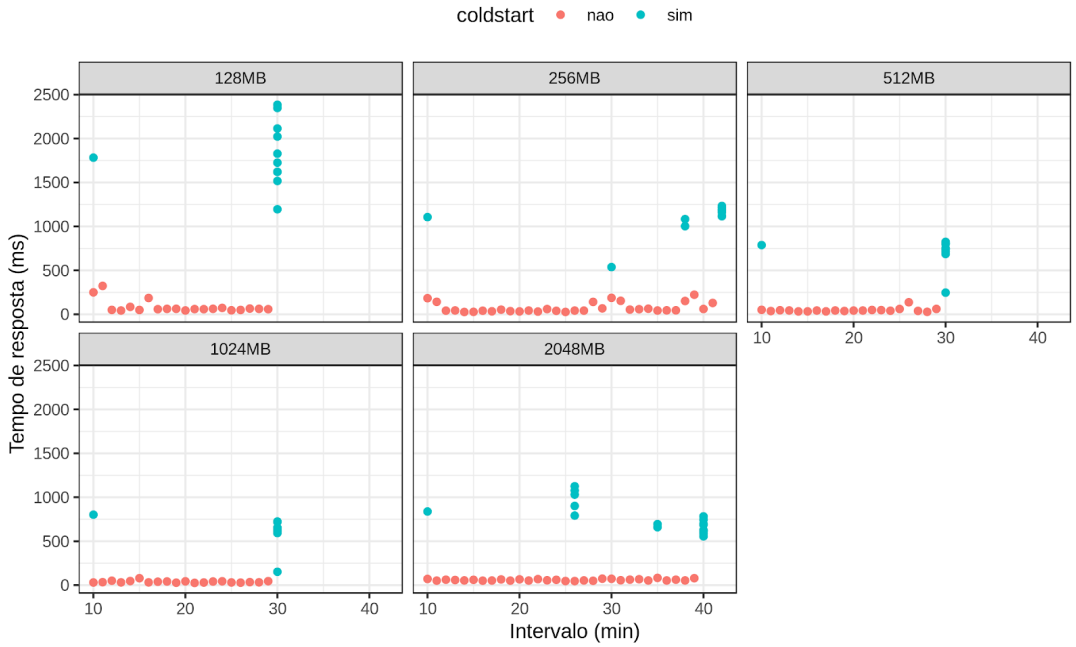


Figura 2. Tempo de resposta das requisições com e sem *coldstart*.

A Figura 3 apresenta o *boxplot* utilizado para analisar o tempo de resposta das requisições para todos os cenários. Podemos comparar as medianas e os quartis do tempo de resposta quando há ou não *coldstart.* Os pontos dispersos presentes na figura são chamados de *outliers*; são os valores discrepantes no nosso gráfico e a presença deles nos experimentos é a razão de fazermos a comparação através da mediana e não da média. Observa-se que o tempo de resposta com coldstart foi significativamente maior do que sem coldstart. A mediana do tempo de resposta sem *coldstart* foi de 51ms, enquanto com *coldstart* a media foi 802ms. Ou seja, o tempo adicional gasto com o *coldstart* foi de aproximadamente 751ms para os cenários avaliados.

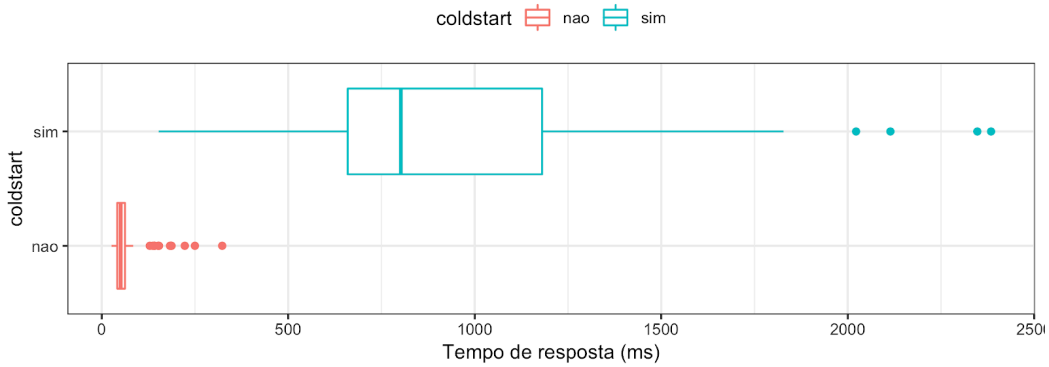


Figura 3. Boxplot do tempo de resposta total com e sem *coldstart*.

A Figura 4 apresenta o boxplot do tempo de resposta com e sem *coldstart*, agora dividindo para cada cenário de memória alocada. Observando o gráfico do tempo de execução quando ocorrem *coldstarts* (à direita), é possível identificar uma tendência: quanto mais memória alocada, menor é o tempo de *coldstart*. Este comportamento é inesperado, pois imaginava-se que conforme a alocação de memória fosse aumentando, maior seria o impacto do carregamento da função durante o *coldstart*.

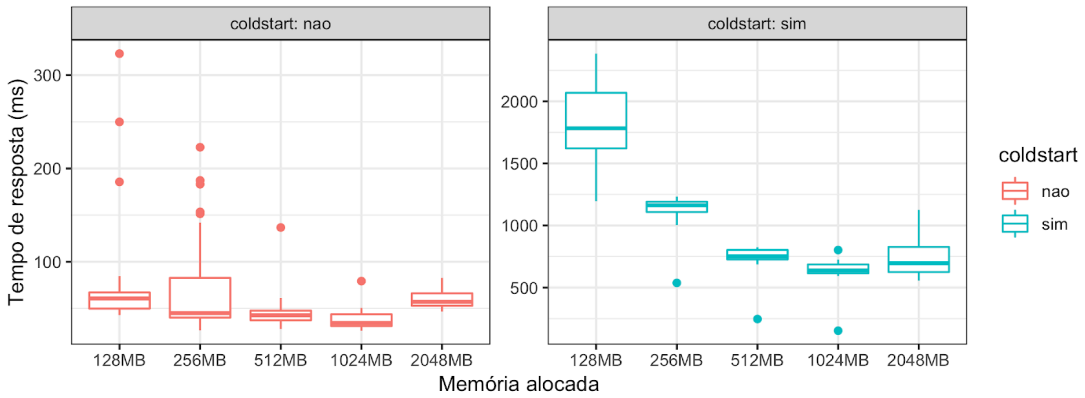


Figura 4. Boxplot do tempo de resposta com e sem *coldstart*, para diferentes cenários de memória alocada para a função.

Este fenômeno pode ser explicado através da documentação do AWS Lambda[4], que diz que a capacidade de CPU alocada para a função é proporcional à quantidade de memória alocada para a mesma. Portanto, uma hipótese é que funções que alocam mais memória, como também possuem maior capacidade de CPU, carregam a função e respondem às requisições mais rapidamente. Uma ressalva é que ao aumentar a memória alocada de 1024MB para 2048MB, não se observou uma diminuição no tempo de resposta; houve, na verdade, um pequeno acréscimo na mediana. Uma possível explicação, no entanto, é que a partir deste ponto, CPU deixa de ser um gargalo e não melhorar mais o desempenho do carregamento e execução da função, enquanto a quantidade maior de memória a ser alocada passa a fazer algum efeito no tempo de carregamento, mesmo que pouco significativo. Além disso, quando não há *coldstart* (gráfico há esquerda), o tempo de resposta apresenta mais *outliers* com valores maiores, mostrando que a capacidade menor de CPU também pode afetar o tempo de resposta da função mesmo sem *coldstart*.

5. Conclusão

O modelo de *serverless computing* está se tornando bastante atrativo para usuários de nuvem, devido à menor complexidade operacional por não precisar gerenciar servidores; da possível redução de custos ao ser cobrado apenas pelo uso da aplicação. Porém, ao desativar funções após um período de ociosidade para reduzir custos, o provedor gera um novo problema para a aplicação chamado *coldstart*, que é o tempo necessário para inicializar e implantar uma aplicação inativa ao chegar uma nova requisição. Neste trabalho, analisamos o *coldstart* em um provedor de *serverless computing*, medindo para diferentes cenários de memória alocada para a função: o tempo que uma função passa ociosa até que o provedor a torne inativa (havendo *coldstart*); o tempo de resposta das requisições com e sem *coldstart*; e o *overhead* que o coldstart impõe no tempo de resposta dependendo da quantidade de memória alocada.

Os resultados dos experimentos mostraram que: (1) *Coldstarts* aconteceram de forma sistemática após 30 a 42 minutos de ociosidade, dependendo da memória alocada para a função. Alguns *coldstarts* isolados também ocorrem com tempos de ociosidade menor, de 25 minutos, principalmente para quantidades maiores de memória alocada, o que supõe que em momentos de sobrecarga o provedor pode desativar funções que ocupam mais recursos mais cedo. (2) O *overhead* imposto pelo *coldstart* foi significativo para os experimentos realizados, sendo a diferença nas medianas do tempo de resposta com e sem *coldstart* de aproximadamente 751ms. (3) Quanto maior a

[4] https://docs.aws.amazon.com/lambda/latest/dg/resource-model.html

memória alocada, menor tende a ser o *overhead* do *coldstart*. A hipótese é que como o provedor aloca CPU proporcionalmente à memória, a função é inicializada e executada mais rapidamente quando há mais memória. Porém, quando se aumentou a memória de 1024MB para 2048MB, o *overhead* não diminuiu. A possível explicação é que a CPU deixou de ser um gargalo e aumentar sua capacidade já não afeta o tempo de resposta significativamente.

Com base nos resultados obtidos no experimento, sugerimos que: aplicações que passam muito tempo ociosas podem reduzir custos por não serem cobradas nestes períodos. Porém, se estes períodos forem superiores a 25 minutos e a aplicação for sensível à latência, precisando de tempos de resposta inferiores a 1 segundo para uma grande porcentagem das suas requisições, o *overhead* imposto pelo *coldstart* pode afetar a qualidade de serviço da aplicação, sendo mais adequado um serviço tradicional baseado em máquinas virtuais ou containers, ou estratégias para que funções não sejam desativadas. Por outro lado, se as aplicações se mantém ativas e raramente apresentam longos períodos de atividade, a facilidade de uso do modelo de *serverless computing* pode ser um grande atrativo para os usuários executarem e escalarem suas aplicações sem precisar gerenciar servidores.

Como possíveis trabalhos futuros, pretende-se investigar o impacto no *coldstart* adotando diferentes provedores de serviços, diferentes linguagens de programação e com experimentos mais exaustivos em dias da semana e horas diferentes para eliminar efeitos externos da análise.

Referencias

Baldini, I., Castro, P., Chang, K., Cheng, P., Fink, S., Ishakian, V., Mitchell, N., Muthusamy, V., Rabbah, R, Slominski, A. and Suter, P. (2017) "Serverless Computing: Current Trends and Open Problems". In: Research Advances in Cloud Computing. Edited by Chaudhary S., Somani G. and Buyya R, Springer, Singapore.

Cui, Y. “Finding coldstarts: how long does AWS Lambda keep your idle functions around?” (2017a) in: https://theburningmonk.com/2017/06/finding-coldstarts-how-long-does-aws-lambda-keep-your-idle-functions-around/. Acesso em: 28 set. 2018.

Cui, Y “How does language, memory and package size affect cold starts of AWS Lambda?” (2017b) in: https://read.acloud.guru/does-coding-language-memory-or-package-size-affect-coldstarts-of-aws-lambda-a15e26d12c76/. Acesso em: 22 out. 2018

Fowler, M. “Serverless Architectures” (2018). In: martinfowler.com. Disponível em: https://martinfowler.com/articles/serverless.html. Acesso em: 14 ago. 2018.

Gancarz, R. “Serverless Still Requires Infrastructure Management” (2018). In: InfoQ. Disponível em: https://www.infoq.com/articles/serverless-infrastructure-management. Acesso em: 14 ago. 2018.

Ken, F. "Why The Future Of Software And Apps Is Serverless" (2012). In: readwrite. Disponível em: https://readwrite.com/2012/10/15/why-the-future-of-software-and-apps-is-serverless. Acesso em: 08 ago. 2018.

Mell, P., Grance, T. (2011). The NIST definition of cloud computing.

McGrath, G., Brenner, P.R. Serverless Computing: Design, Implementation, and Performance, IEEE 37th International Conference on Distributed Computing Systems Workshops, 2017

Perez, C. “Serverless e AWS Lambda” (2018). In: Elo7 Tech. Disponível em: https://engenharia.elo7.com.br/serverless/. Acesso em: 20 ago. 2018.

Savage, N. (2018). Going serverless. Communications of the ACM, 61(2), 15-16.

II Workshop de Trabalhos de Iniciação Científica e Graduação
SBRC 2019
Sessão Técnica 2

Análise Preliminar da Detecção de Ataques Ofuscados e do Uso de Hardware de Baixo Custo em um Sistema para Detecção de Ameaças*

Lucas Seiki Oshiro[1], **Daniel Macêdo Batista**[1]

[1]Departamento de Ciência da Computação – IME
Universidade de São Paulo (USP) – São Paulo, SP, Brasil

lucas.oshiro@usp.br, batista@ime.usp.br

Abstract. *Network traffic transmitted at high rates results in the need for more efficient security mechanisms, since analyzing package by package before taking an action becomes a costly task in terms of processing. One way to solve this problem is with the development and deployment of threat detection systems that use machine learning mechanisms to anticipate attacks. This paper presents the preliminary results obtained in an attempt to improve such a system by analyzing automated attacks that employ obfuscation and by evaluating the performance of a Raspberry Pi unit that can be used as a processing node in the improved system.*

Resumo. *Tráfego de rede transmitido a altas taxas traz como consequência a necessidade de mecanismos de segurança mais eficientes, já que analisar pacote por pacote antes de tomar uma ação torna-se uma tarefa custosa em termos de processamento. Uma forma de resolver esse problema é com o desenvolvimento e implantação de sistemas de detecção de ameaças que utilizem mecanismos de aprendizado de máquina para antecipar os ataques. Este artigo apresenta os resultados preliminares obtidos na tentativa de melhorar um sistema como esse por meio da análise de ataques automatizados que empregam ofuscação e por meio da avaliação de desempenho de uma unidade Raspberry Pi que poderá ser usada como nó de processamento no sistema melhorado.*

1. Introdução

,Com a obtenção de quantidades massivas de dados de tráfego e de aplicações de rede, surge a necessidade de métodos mais "inteligentes" para buscar incidentes de segurança, principalmente porque passa a ser possível encontrar informações novas por meio da correlação das informações e também porque uma análise de força-bruta levaria muito tempo para ser finalizada. Essa necessidade já vem sendo discutida há alguns anos tanto na academia [Brown et al. 2015] quanto na indústria, que já fornece diversos serviços para clientes dos mais diversos tamanhos [Splunk 2019]. Entretanto, as implementações dos métodos não seguem uma arquitetura que seja eficiente para todos os tipos de organizações. De fato, tem-se notado que cada vez mais as arquiteturas precisam ser

*Este artigo resume os resultados parciais do trabalho de conclusão de curso que encontra-se em desenvolvimento por Lucas Seiki Oshiro e que pode ser acompanhado em `https://linux.ime.usp.br/~lucasoshiro/mac0499/`. Último acesso em 22 de Março de 2019.

otimizadas e personalizadas para cada tipo de usuário [Feth 2015]. Além disso, é desejável que o sistema a ser desenvolvido seja capaz de antecipar, ao máximo possível, incidentes em tempo real e que o mesmo não tenha um custo elevado tanto em termos financeiros para adquirir equipamentos, quanto em termos de consumo de energia. Uma forma de antecipar incidentes é com a utilização de mecanismos baseados em aprendizado de máquina. Uma forma de reduzir os custos é com a utilização de clusters baseados em hardware de baixo custo.

No escopo do projeto **GT-BIS – Mecanismos para Análise de Big Data em Segurança da Informação** [GT-BIS 2018], foi desenvolvido um sistema capaz de detectar ataques a partir da análise de grandes volumes de logs de servidores web e de servidores de banco de dados, por meio de técnicas de aprendizado de máquina. No momento é necessário melhorar o sistema com a adição da detecção de novos tipos de incidentes de segurança, como por exemplo aqueles causados por ataques automatizados que usam ofuscação e que burlam mecanismos mais tradicionais de proteção.

Nesse sentido, esse artigo apresenta resultados parciais obtidos com a análise de diferentes ataques contra aplicações web e com a análise de desempenho de uma unidade Raspberry Pi 3 Model B em cenários de uso intensivo de CPU. Como resultado foi possível identificar algumas características dos ataques que podem ser úteis para o treinamento de um modelo de detecção de ameaças. Já os resultados da análise de desempenho da Raspberry Pi mostraram que ela apresenta boa vazão de tráfego de rede e temperatura dentro de limites seguros mesmo quando submetida a altas cargas de processamento.

As próximas seções estão organizadas da seguinte forma. A Seção 2 resume a arquitetura do sistema considerado e os ataques estudados no trabalho. A Seção 3 descreve as ferramentas que foram utilizadas na criação de um ambiente de experimentação para reprodução e detecção dos ataques. A Seção 4 descreve os experimentos realizados e os resultados obtidos. A Seção 5 finaliza o artigo com as conclusões e os próximos passos.

2. Conceitos Básicos

Este artigo apresenta resultados visando melhorar o sistema de detecção de ameaças ilustrado na Figura 1. Os números na figura descrevem os dados que são passados entre cada componente do sistema: **1)** logs de serviços e de sistemas de segurança. **2)** dados normalizados, filtrados e enriquecidos. **3)** fluxos de dados organizados em filas para serem processados. **4)** informações brutas e de dados já processados (p. ex. alertas); comandos de consultas. **5)** dados para serem persistidos ou recuperados. **6)** alertas de ameaças cibernéticas compartilhados pelo sistema central ou parceiros. **7)** dados da interação do administrador com a interface Web. **8)** informações disponíveis nos componentes de Processamento (alertas, dados brutos, dados processados); comandos de consultas.

`Logstash`, `Kafka`, `Spark` e `ElasticSearch`, apresentadas na Figura 1, são ferramentas escaláveis e paralelizáveis de software livre para, respectivamente, filtrar e normalizar dados de logs, implementar um sistema produtor/consumidor para acesso a filas de dados, processar dados empregando por exemplo mecanismos de aprendizado de máquina e armazenar e visualizar dados. Todas essas ferramentas podem ser replicadas em diferentes nós em *clusters* a fim de tornar o funcionamento escalável em função da carga de dados de entrada e é nesse sentido que propomos a utilização de unidades de Raspberry Pi em cada um desses *clusters*.

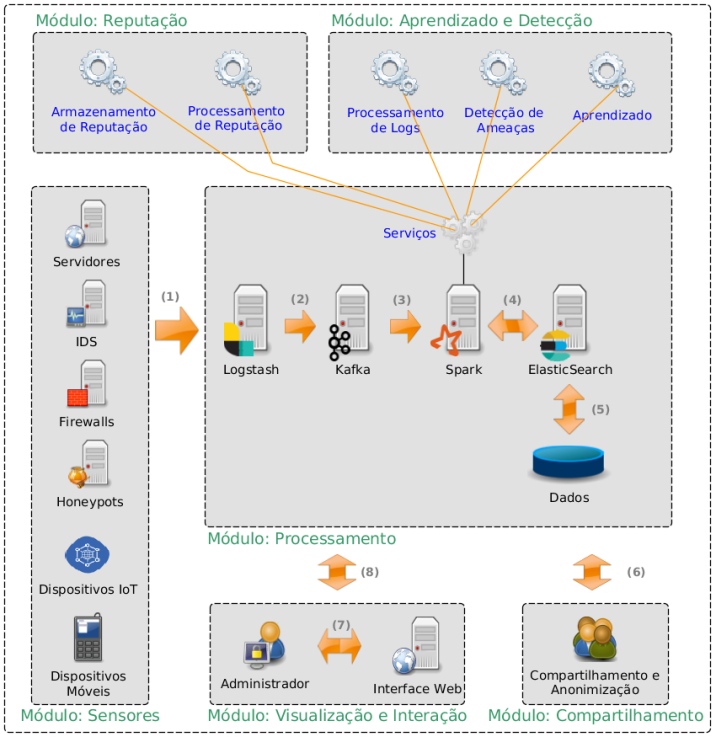


Figura 1. Arquitetura do sistema para detecção de ameaças por meio da análise de logs [GT-BIS 2018]

2.1. Ataques Contra Aplicações Web

Ataques contra redes de computadores ocorrem por conta de vulnerabilidades em diversas camadas da arquitetura Internet. Especificamente no caso de aplicações web, aquelas projetadas para serem acessadas principalmente por meio de um navegador web tendo o protocolo HTTP como base, falhas podem levar à negação do serviço, ao acesso indevido a informações ou mesmo à execução de comandos arbitrários do lado do servidor.

Pelo fato do protocolo HTTP ser um protocolo sem estado, boa parte das aplicações web requer a utilização de algum Sistema Gerenciador de Banco de Dados (SGBD). Tais sistemas costumam ser alvos de atacantes em busca de falhas que possam afetar a infraestrutura de uma aplicação web. Outro alvos para a exploração de falhas é a construção de URLs maliciosas.Alguns exemplos de ataques desses tipos são: **Injeção de SQL** – inserção de dados que, se não tratados, realizam ações indevidas no banco de dados; **XSS Refletido** – inserção de código HTML e JavaScript malicioso em uma URL, que é executado quando um site é aberto por meio daquela URL; **XSS Persistido** – envio de código HTML e JavaScript malicioso, que fica armazenado e é executado no navegador de todos os usuários que acessarem o site.

A distinção de ataques como os listados acima no meio de acessos legítimos pode ser dificultada caso o atacante utilize técnicas de ofuscação facilitadas por ferramentas que automatizam a exploração desses ataques.

2.2. Plataformas de Computação de Baixo Custo

Muitas aplicações para Internet das Coisas [Singh and Kapoor 2017] dependem de placas e sistemas embarcados que funcionem tanto como sensores, obtendo informações

do mundo real, quanto como processadores, analisando os dados capturados e enviando comandos de atuação. A necessidade de implantação desses elementos em larga escala justifica a utilização de hardware de baixo custo. Dentre as várias opções de hardware para esse objetivo destacam-se o Arduino [Arduino 2019] e a Raspberry Pi [Raspberry Pi Foundation 2018]. O Arduino é uma plataforma de código aberto para computação física baseada em entradas e saídas simples [Banzi 2011], enquanto que a Raspberry Pi, embora também seja um dispositivo em placa única, é um computador e possui maior poder de processamento que o Arduino. O grande sucesso na utilização desses dispositivos em aplicações de Internet das Coisas tem levado à sua utilização em outros domínios de aplicação. No caso da Raspberry Pi, vários projetos tem utilizado diversas unidades conectadas via rede criando assim um *cluster* de baixo custo [Pahl et al. 2016]. No Brasil, uma unidade Arduino UNO pode ser adquirida por cerca de R$50,00, enquanto uma unidade Raspberry Pi 3 Model B pode ser adquirida por cerca de R$200,00.

Antes de considerar a utilização dessa plataforma como nó de processamento em algum dos componentes de um sistema de detecção de ataques como o ilustrado na Figura 1 é importante analisar o seu desempenho em situações de uso intensivo de CPU.

3. Ambiente de Experimentação

Dois conjuntos de experimentos foram realizados. O primeiro conjunto teve por objetivo a análise dos logs gerados nos diversos servidores envolvidos na execução de uma aplicação web em situações de ataques. O segundo conjunto teve por objetivo avaliar uma unidade Raspberry Pi em situações de uso intensivo da CPU.

3.1. Simulação de Ataques

Optou-se por utilizar uma aplicação web já existente, a *Damn Vulnerable Web Application* (DVWA)[1]. A DVWA é uma aplicação Web disponibilizada para profissionais de segurança testarem suas ferramentas e habilidades. Cada página disponibilizada na DVWA possui uma vulnerabilidade.

A utilização da DVWA tinha por objetivo garantir que o servidor web instalado teria uma aplicação com as vulnerabilidades esperadas, de modo que o sucesso, ou insucesso, dos ataques pudesse ser de fato avaliado a partir dos logs do servidor e do comportamento da aplicação. A DVWA versão 1.10 foi instalada em uma máquina virtual (MV) gerenciada pelo VirtualBox versão 5.2. As configurações de hardware da MV eram: 4GB de memória RAM, 10GB de armazenamento, usando apenas 1 núcleo do processador. As configurações de hardware da máquina física utilizada eram: 8GB de memória RAM, 200GB de disco rígido e processador Intel Core i5, com 2 núcleos e 4 *hyperthreads* a 2,7GHz. As configurações de software da MV eram: Ubuntu 16.04 LTS, servidor web Apache 2.0, SGBD MySQL 14.14 e PHP 7.0. As configurações de software da máquina física eram: sistema operacional Manjaro Linux 17.1 Hakoila, com kernel Linux 4.14 LTS. O MySQL precisou ser configurado para produzir um arquivo de log com os acessos feitos a ele. Essa configuração não vinha por padrão no software.

Os ataques realizados contra a DVWA foram realizados em uma MV gerenciada no VirtualBox na mesma versão e na mesma máquina física. As configurações de hardware da MV eram: 3GB de memória RAM, 20GB de armazenamento, usando apenas

[1] `http://www.dvwa.co.uk/`. Último acesso em 22 de Março de 2019.

1 núcleo do processador. O sistema operacional usado nessa máquina foi o Kali Linux versão 4.15. O Kali Linux é uma distribuição Linux baseada no Debian, contendo software especializado para a execução de testes de penetração e auditoria de segurança. As ferramentas utilizadas para a realização dos ataques foram o SQLMap versão 1.2.4 para automatizar os ataques de injeção de SQL, o XSSer versão 1.7 para automatizar os ataques de *Cross-Site Scripting* (XSS) refletido e persistido e o Burp Suite versão 1.7.33 para interceptar as informações transmitidas pelo navegador utilizado nos experimentos. A interconexão entre as MVs foi feita no modo rede exclusiva de hospedeiro no VirtualBox.

3.2. Análise de Desempenho da Raspberry Pi

A Raspberry Pi utilizada foi do modelo Raspberry Pi 3 Model B. Seu processador é um Broadcom BCM2837 64bit, com 4 núcleos a 1,2GHz, e sem dissipador de calor; sua memória RAM tem 1GB; o sistema operacional utilizado foi o Raspbian 2018-06-27. Para auxiliar as análises, foi utilizada a mesma máquina hospedeira das MVs usadas para os ataques. As máquinas estavam conectadas fisicamente à Raspberry Pi via Fast Ethernet com um roteador Linksys WRT54G.

4. Resultados

4.1. Injeção de SQL

Os ataques de injeção de SQL foram feitos conforme o fluxo de trabalho da Figura 2. Ao término das ações foram obtidos os nomes de todos os usuários da base de dados e os *hashes* de suas senhas. Através de força bruta baseada em dicionário com o software SQLMap, foi possível descobrir todas as senhas.

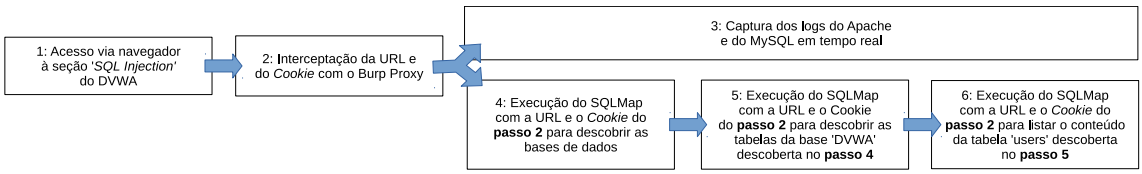


Figura 2. Passo-a-passo da realização dos ataques de injeção de SQL

Com a coleta dos logs realizada no passo 3 foi possível observar as seguintes características no log do Apache: URLs de acesso com grande quantidade (em geral, mais de 5) de caracteres em notação hexadecimal que são representados pelo caracter `%` seguido de um número hexadecimal; Grande uso de palavras reservadas da linguagem SQL como `SELECT`, `AND`, `OR`, `NOT`, `WHERE`, `BOOLEAN`; Intervalo de tempo entre acessos de um mesmo endereço IP muito curto (foram mais de 30 acessos por segundo); Embora seja algo fácil de modificar, o agente de usuário de todos os acessos registrados foi `sqlmap`.

No log do MySQL foi observada uma grande frequência de comparações do tipo "numero1=numero2" na cláusula `WHERE` das consultas, sendo ambos os números escritos explicitamente. Essas comparações vêm acompanhadas dos booleanos `AND` e `OR`.

Considerando a criação de um modelo baseado nas características listadas acima, a presença de apenas uma delas não significa a certeza de um ataque. No caso de caracteres hexadecimais por exemplo, isso pode ocorrer caso o caracter sendo passado na URL não possa ser representado na notação ASCII. Já no caso de vários acessos de um mesmo

IP, isso pode ocorrer por conta da utilização de NAT. Considerar apenas uma das características pode levar a uma alta taxa de falsos positivos, entretanto, como eles estiveram sempre presentes nos ataques realizados pelo SQLmap, eles representam potenciais candidatos de *features* a serem analisadas por algoritmos de aprendizado de máquina. Claro que o comportamento normal dos logs deve ser estudado para avaliar o peso dado a cada uma das *features* reduzindo a chance de falsos positivos e de falsos negativos.

4.2. XSS Refletido

Os ataques de XSS refletido foram feitos conforme o fluxo de trabalho da Figura 3. Ao término das ações foi possível acessar com sucesso a DVWA por meio de URLs contendo código HTML malicioso inserido pelos ataques.

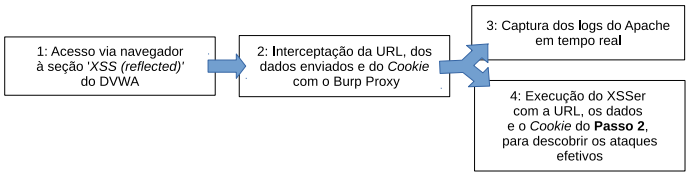


Figura 3. Passo-a-passo da realização dos ataques de XSS Refletido

Com a coleta dos logs realizada no passo 3 foi possível observar as seguintes características no log do Apache: URLs de acesso com código HTML inserido; Intervalo de tempo entre acessos de um mesmo endereço IP muito curto (foram cerca de 300 acessos por segundo); Embora seja algo fácil de modificar, o agente de usuário de todos os acessos registrados foi `Googlebot/2.1 (+http://www.google.com/bot.html)`.

Vale observar a similaridade com algumas das características presentes durante os ataques de Injeção de SQL. Os mesmos comentários anteriores sobre a consideração dessas características para a criação de um modelo continuam pertinentes.

4.3. XSS Persistido

Os ataques de XSS persistido foram feitos de forma similar ao fluxo de trabalho da Figura 3, com a diferença de que a seção acessada na DVWA foi a *XSS (store)* (Passo 1) e que os logs do MySQL também foram monitorados (Passo 3). Apesar do XSSer não identificar nenhum ataque como bem sucedido, a página da DVWA referente ao XSS persistido sempre redirecionou para uma página diferente, efeito desse ataque.

Diferente dos experimentos anteriores, o servidor web precisou ser configurado para registrar nos logs os dados enviados pelo método `POST` do HTTP. Esse procedimento, porém, não é indicado uma vez que podem ser fornecidos dados muito grandes através desse método. Além disso, a quantidade de entradas no *log* é muito grande. No experimento foram executados 558 envios de formulários, e a partir deles foram geradas 46497 entradas no *log*, sendo que a maior parte possui informações pouco relevantes.

Com a coleta dos logs realizada no passo 3 foi possível observar as mesmas características observadas durante os experimentos de XSS refletido no log do Apache, com a diferença de que as *tags* HTML estavam incompletas. Como os dados fornecidos são armazenados no banco de dados, as inserções no banco de dados observadas no log do MySQL também contém essas *tags*.

4.4. Análise de Desempenho da Raspberry Pi

A fim de avaliar de forma preliminar a eficiência da Raspberry Pi como nó de um cluster para análises em busca de ameaças de segurança, ela teve seu processador estressado pela ferramenta `stress` com diferentes números de processos "pesados". Cada um desses processos é chamado de *worker*. Avaliou-se vazão da rede e temperatura da placa. A primeira métrica foi avaliada para verificar se a unidade era capaz de receber dados mesmo se estivesse com utilização elevada da CPU, simulando uma situação em que dados de logs para análise estejam constantemente sendo enviados em paralelo à análise desses dados. A ferramenta `iperf` foi utilizada para esse fim, enviando dados com o protocolo UDP e com o protocolo TCP. A segunda métrica foi avaliada para verificar se a placa era capaz de operar em situações extremas mesmo em um local que não tivesse uma infraestrutura preparada para resfriamento de dispositivos computacionais de alto desempenho, algo que aumentaria o custo da solução. A ferramenta `vcgenmod` foi utilizada para esse fim.

A Figura 4 apresenta a vazão média retornada pelo `iperf` em função de diferentes quantidades de `workers`. A dispersão dos dados foi omitida do gráfico por ter ficado baixa em todos os experimentos (o maior desvio padrão foi 0,37Mbps). Pelos resultados obtidos, o uso da CPU não afetou a capacidade da placa em receber dados via rede.

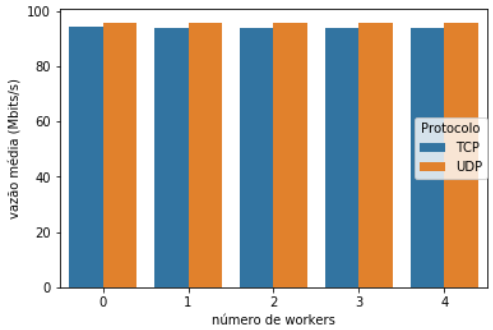


Figura 4. Vazão de rede da Raspberry Pi com CPU estressada

A Figura 5 apresenta a temperatura retornada pelo `vcgenmod` em função do tempo. Antes da execução do `stress`, foram amostrados 40 valores de temperatura por 20 segundos; durante a execução do `stress`, foram amostrados 1200 valores de temperatura durante 600 segundos; após a execução do `stress`, foram amostrados 2400 valores de temperatura durante 1200 segundos. Desta forma, foi possível verificar o tempo que o processador levou para esquentar e esfriar. A placa continuou em pleno funcionamento durante todos os experimentos, suportando a carga de trabalho imposta.

5. Conclusões e Próximos Passos

Este artigo apresentou os resultados parciais de um trabalho de conclusão de curso que encontra-se em andamento e que tem por objetivo detectar ataques ofuscados utilizando hardware de baixo custo. Os resultados mostraram diversas características observadas durante os ataques que podem ser utilizadas para a criação de modelos de aprendizado de máquina e a boa eficiência de uma unidade Raspberry Pi 3 Model B tanto em termos de vazão de rede quanto em termos de temperatura em situações de uso intensivo da CPU. Os próximos passos são a criação de modelos baseados nas características observadas e a implantação de unidades Raspberry Pi no sistema baseado nesses modelos.

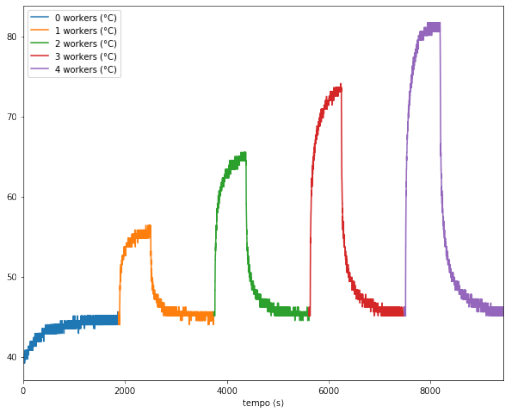


Figura 5. Temperatura da Raspberry Pi com CPU estressada

Agradecimentos

Esta pesquisa é parte do INCT da Internet do Futuro para Cidades Inteligentes financiado pelo CNPq proc. 465446/2014-0, Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Código de Financiamento 001, FAPESP proc. 14/50937-1, e FAPESP proc. 15/24485-9.

Referências

Arduino (2019). Arduino - Home. `https://www.arduino.cc/`. Último acesso em 22 de Março de 2019.

Banzi, M. (2011). *Getting Started with Arduino*. Make: projects. O'Reilly Media.

Brown, S., Gommers, J., and Serrano, O. (2015). From Cyber Security Information Sharing to Threat Management. In *Proceedings of the 2Nd ACM WISCS'15*, pages 43–49.

Feth, D. (2015). User-centric Security: Optimization of the Security-usability Trade-off. In *Proceedings of the 10th ESEC/FSE 2015*, pages 1034–1037.

GT-BIS (2018). GT-BIS – Mecanismos para Análise de Big Data em Segurança da Informação. `http://gtbis.ime.usp.br/`. Último acesso em 22 de Março de 2019.

Pahl, C., Helmer, S., Miori, L., Sanin, J., and Lee, B. (2016). A Container-Based Edge Cloud PaaS Architecture Based on Raspberry Pi Clusters. In *4th IEEE FiCloudW*, pages 117–124.

Raspberry Pi Foundation (2018). Raspberry Pi – Teach, Learn, and Make with Raspberry Pi. `https://www.raspberrypi.org/`. Último acesso em 22 de Março de 2019.

Singh, K. J. and Kapoor, D. S. (2017). Create Your Own Internet of Things: A survey of IoT Platforms. *IEEE Consumer Electronics Magazine*, 6(2):57–68.

Splunk (2019). SIEM, AIOps, Application Management, Log Management, Machine Learning, and Compliance — Splunk. `https://www.splunk.com/en_us`. Último acesso em 22 de Março de 2019.

LabSensing: Um Sistema de Sensoriamento para Laboratórios Científicos com Computação Inteligente nas Bordas

Kaylani Bochie e Miguel Elias M. Campista *

[1]Grupo de Teleinformática e Automação (GTA)
PEE/COPPE-DEL/Poli
Universidade Federal do Rio de Janeiro (UFRJ)

{kaylani,miguel}@gta.ufrj.br

Resumo. *Este trabalho propõe o LabSensing, um sistema de sensoriamento distribuído para monitoramento de laboratórios científicos. O LabSensing utiliza uma estratégia de comunicação assíncrona para transferência de dados e reduz o tráfego injetado na rede através de políticas de eliminação de dados redundantes nas bordas. As motivações técnicas que levaram à escolha das ferramentas, assim como o processo de desenvolvimento do sistema a partir da arquitetura proposta são discutidos. Os resultados experimentais mostram o funcionamento do sistema e a efetividade obtida com a computação nas bordas para redução da carga de dados na rede.*

Abstract. *This paper proposes LabSensing, a distributed sensing system for scientific laboratory monitoring. LabSensing employs an asynchronous communication strategy to transfer data and reduces the traffic injected in the network using policies to eliminate redundant data at the edges. The technical motivations leading to tools selection, as well as the design and development of the system taking into account the proposed architecture are discussed. Experimental results show the operation of the system and the impact obtained using edge computing to reduce the network data load.*

1. Introdução

A explosão de dispositivos eletrônicos tem impulsionado o paradigma de Internet das Coisas (*Internet of Things* – IoT), ao possibilitar diversas aplicações com base em sensoriamento. A popularização dessas aplicações leva a um aumento do número de dispositivos conectados à Internet e ao crescimento do volume de dados gerado e encaminhado pelas redes de comunicação. Diversos conceitos e abordagens vem sendo aplicados no desenvolvimento de ferramentas e sistemas nesse contexto, tais como a Computação na Borda (*Edge Computing*) [Shi et al., 2016]. Apesar de ambos os conceitos, IoT e Computação na Borda, costumarem ser aliados a soluções baseadas em nuvens computacionais de larga escala, os mesmos princípios podem ser aplicados a redes locais. A ideia é igualmente distribuir e melhor utilizar o poder computacional disponível em dispositivos embarcados e sistemas distribuídos.

*O presente trabalho foi realizado com apoio do CNPq; da FAPERJ; da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior Brasil (CAPES), Código de Financiamento 001; e da Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP), processos n^o 15/24494-8 e 15/24490-2.

Em redes locais, um dos problemas de implementação do sensoriamento é a má utilização dos recursos computacionais dos nós de borda. Essa prática gera sobrecarga no nó central da rede, responsável pela recepção dos dados gerados, contrastando com a subutilização dos nós de sensoriamento. Ao invés de cumprir um papel meramente de sensores, esses nós podem ser usados para verificar os dados coletados e, apenas se necessário, enviá-los ao nó central. A ideia é distribuir o processamento pelas bordas da rede, agregando inteligência para: (i) reduzir o volume de dados enviados até o nó central e (ii) aumentar a escalabilidade do sistema mesmo em ambientes locais. Outra vantagem do deslocamento do processamento para as bordas da rede é a redução do tempo de resposta, que se torna útil se os sensores esperarem alguma forma de retorno do sistema.

Na literatura existem trabalhos que também propõem sistemas de monitoramento. Por exemplo, Kodali e Gorantla discutem escolhas de software e hardware para uma rede de sensores sem fio [Kodali e Gorantla, 2017]. Também são analisadas as possíveis aplicações para redes de baixo custo e os méritos da tecnologia utilizada. Alqinsi et al. constroem um sistema de monitoramento através do protocolo MQTT (*Message Queue Telemetry Transport*) [Alqinsi et al., 2018]. Saha e Majumdar desenvolvem um sistema de monitoramento para centros de dados baseado em nuvem [Saha e Majumdar, 2017]. O sistema proposto neste trabalho segue a tendência usando, além das técnicas encontradas na literatura de sistemas IoT, conceitos utilizados em computação na borda aplicados a redes locais, tais como distribuição de poder computacional nos dispositivos de sensoriamento. Isso tem como consequência o aumento de confiabilidade do sistema, devido à independência de serviços remotos e agentes externos.

Este artigo propõe o LabSensing, um sistema para sensoriamento de dados em laboratórios científicos complementar a trabalhos anteriores [Kodali e Gorantla, 2017, Alqinsi et al., 2018, Saha e Majumdar, 2017]. O LabSensing reúne conceitos de IoT e de Computação na Borda e os aplica a sistemas implementados em redes locais. Essa característica elimina o uso das nuvens computacionais comerciais, reduzindo os custos da operação do sistema. Dessa forma, tanto os sensores quanto os nós de borda e o servidor central são instalados no próprio laboratório. A visualização dos dados é feita a partir de uma interface *web*, acessível tanto internamente quanto externamente ao laboratório. O sistema LabSensing permite o monitoramento do laboratório a partir de nós sensores diversos e evita que todos os dados sejam enviados ao servidor central através de processamento nas bordas. O LabSensing é construído com hardware de baixo custo e software livre, além de permitir a adição de componentes de forma modular, desde que estes possam implementar o MQTT, adotado por permitir a troca de mensagens em sistemas locais com restrições de hardware [Oasis, 2014]. O sistema proposto é também capaz de variar as políticas de processamento nas bordas através de parâmetros de controle no código de cada um dos nós. Os resultados experimentais mostram que o pré-processamento realizado nas bordas da rede alivia o volume de dados transmitido. Essa redução do tráfego, aliada ao baixo custo dos dispositivos, faz com que o sistema seja ideal para utilização em redes locais, porém ainda extensível para ambientes de maior escala.

Este trabalho está organizado da seguinte forma: a Seção 2 descreve a arquitetura do LabSensing e apresenta as ferramentas utilizadas para a sua implementação. A Seção 3 apresenta os experimentos práticos conduzidos e a análise dos resultados obtidos. Por fim, a Seção 4 conclui este trabalho e aponta as direções futuras.

2. Arquitetura e Implementação do LabSensing

Esta seção descreve inicialmente a arquitetura do LabSensing e, em seguida, as opções de implementação.

2.1. Arquitetura

A arquitetura do LabSensing segue uma divisão em camadas típica de sistemas de IoT. Ele é composto por uma camada de *percepção*, uma de *comunicação* e uma de *armazenamento e exibição dos dados*. A camada de percepção, responsável pelo sensoriamento do ambiente monitorado, é composta por nós sensores distribuídos, chamados de nós de percepção. Esses sensores são implementados em dispositivos capazes de reduzir a carga de dados injetada na rede, segundo políticas de filtragem predeterminadas. Essas políticas podem ser usadas para atribuir "inteligência" aos nós de borda. Já a camada de comunicação, responsável pela troca de mensagens entre nós produtores e consumidores de dados, utiliza um substrato composto por redes sem fio e cabeadas. Os sensores utilizam redes sem fio devido a facilidades de instalação física, mas essa decisão não representa uma limitação da arquitetura. A camada de comunicação ainda possui um Nó de Concentração conectado ao Broker através de uma interface local (*localhost*). Os dados enviados através da rede são armazenados em um nó central que executa a camada de armazenamento e exibição dos dados do sistema de IoT. O Nó Concentrador da arquitetura executa clientes que recebem, processam e armazenam os dados da base de dados disponível. Por fim, um *dashboard* é adicionado para visualização em tempo real dos dados por parte dos usuários. Essa possibilidade de visualização é uma alternativa a consultas diretas a base de dados. A Figura 1 apresenta uma visão geral da arquitetura, na qual as setas representam a direção do fluxo de dados desde os nós de percepção até a visualização em um *dashboard*.

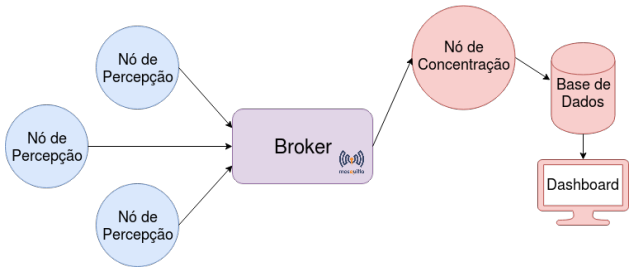


Figura 1. Arquitetura do LabSensing composta por nós de percepção (nós sensores), interconexão entre os componentes e um servidor central responsável pelo armazenamento, processamento e exibição dos dados.

2.2. Implementação

A implementação do LabSensing prioriza o emprego de hardware de baixo custo e software livre. Esta seção apresenta o hardware utilizado, o software e a implementação da inteligência nas bordas da rede. A apresentação da implementação está organizada a partir da camada de comunicação, uma vez que uma decisão importante da arquitetura foi a comunicação assíncrona entre os nós consumidores e produtores de dados. Todos os códigos utilizados para programar os microcontroladores podem ser encontrados em um repositório GitHub [Bochie, 2019].

Camada de comunicação: É utilizada para transferência de mensagens. Nesse sentido, o protocolo MQTT é selecionado para proporcionar a comunicação entre os nós sensores e o nó central de forma assíncrona. O MQTT foi escolhido por ser um protocolo baseado no paradigma *Publish/Subscribe* para comunicação entre dispositivos de rede com limitações de banda e poder computacional. A operação típica do MQTT envolve agentes do tipo cliente e Broker. O papel do Broker é receber as mensagens dos clientes produtores e redirecioná-las aos clientes consumidores interessados. As mensagens são publicadas em tópicos pelos clientes produtores e consumidas apenas por aqueles que estejam inscritos nos tópicos correspondentes. Isso possibilita a comunicação Dispositivo a Dispositivo (D2D) através do Broker. O protocolo MQTT também suporta três níveis de QoS (*Quality of Service*): no QoS 0 (*At Most Once*) a entrega das mensagens não é garantida; no QoS 1 (*At Least Once*) o Broker garante que os clientes inscritos no tópico recebam no mínimo uma cópia da mensagem; no QoS 2 (*Exactly Once*), o Broker garante que os clientes inscritos no tópico recebam apenas uma cópia da mensagem enviada.

Os sensores utilizam redes Wi-Fi, por questões de facilidade de instalação, enquanto o Nó de Concentração foi implementado pelo software Node-RED na mesma máquina onde o Broker foi instalado. O nó utiliza a interface local (*localhost*) para se conectar ao Broker. Sobre a configuração da rede, diversos pontos de acesso Wi-Fi são espalhados pelo laboratório utilizando *Raspberry Pis*, que aumentam a cobertura de rede e permitem uma melhor distribuição dos sensores pelo ambiente de monitoramento. Para implementações em redes locais, é possível que o número de endereços disponíveis seja insuficiente para um número elevado de sensores. Este problema, porém, pode ser contornado por pontos de acesso Wi-Fi que utilizem NAT (*Network Address Translation*).

O software Mosquitto [Eclipse, 2019] foi escolhido para implementação do Broker por ser livre e ter extenso suporte na comunidade acadêmica. Os clientes MQTT podem ser dispositivos diversos, enquanto o Broker deve ser capaz de lidar com um grande número de conexões. Estas escolhas não impedem a adição de nós atuadores ao sistema.

Camada de percepção: É responsável por produzir os dados de sensoriamento vindos de dispositivos diversos. A implementação desta camada se deu a partir de nós baseados no módulo NodeMCU, clientes produtores de dados enviados através do MQTT.

O NodeMCU é uma plataforma IoT de software livre baseada no SoC (*System on Chip*) Wi-Fi ESP8266. O hardware é baseado no módulo ESP-12. Apesar do termo NodeMCU se referir apenas ao firmware ao invés da placa de desenvolvimento, ele é comumente utilizado para descrever toda a plataforma de prototipagem, envolvendo hardware e software. Além disso, cada nó de sensoriamento é composto pelo módulo NodeMCU e por um *array* de sensores. A quantidade e tipo de cada sensor dependem da função do nó no ambiente de sensoriamento. A implementação atual do LabSensing possui quatro nós de sensoriamento equipados com os sensores listados na Tabela 1.Esses nós foram distribuídos pelo laboratório para monitoramento diverso, sendo que dois deles foram implementados de forma a possibilitar a análise de desempenho do sistema, são eles:

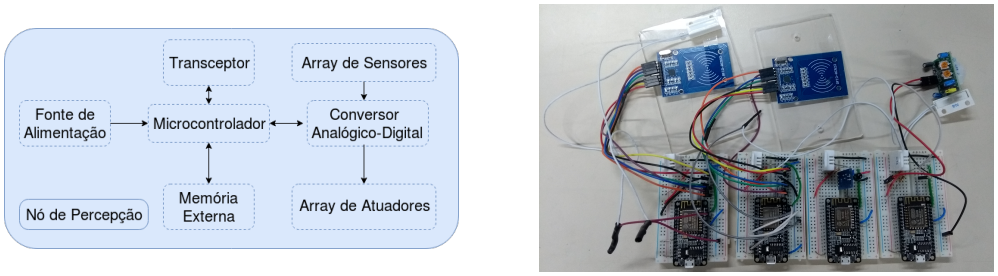
- **Nó Síncrono:** envia todos os dados coletados em um intervalo fixo de tempo, definido por um parâmetro no arquivo de configuração do sistema. Entre intervalos de transmissão, o nó assume um estado de *deep sleep* para economia de energia.
- **Nó Assíncrono:** envia os dados coletados apenas quando um evento pré-programado é detectado, ou seja, é orientado a eventos. Os eventos escolhidos

foram a mudança de estados detectada pelo sensor de porta, identificação de novo cartão RFID e mudanças abruptas de temperatura.

Tabela 1. Sensores em cada um dos quatro nós de sensoriamento da rede.

Nós	Sensores				
	Leitor RFID (MFRC522)	**Temperatura e Umidade (DHT22)**	**Chama**	**Movimento PIR**	**Porta *Reed Switch***
Nó 1 - Assíncrono	√	√			√
Nó 2 - Síncrono	√	√			√
Nó 3 - Dados		√	√		√
Nó 4 - Professor		√		√	

A Figura 2(a) mostra o projeto geral dos nós sensores com os elementos necessários de cada um, enquanto a Figura 2(b) mostra de fato os quatro nós sensores implementados. O transceptor de todos os nós deve suportar comunicação Wi-Fi, sendo que nem todos os nós possuem o mesmo *array* de sensores. A troca de sensores, porém, é simples e pode ser realizada caso haja necessidade. Na Figura 2(b), da esquerda para a direita, observa-se o Nó Assíncrono, o Nó Síncrono, o nó posicionado na sala do centro de dados do laboratório (Nó Dados) e o nó da sala de um professor (Nó Professor).



(a) (b)

Figura 2. (a) Esquema geral de um nó de sensoriamento. (b) Nós implementados e utilizados nos experimentos.

Como mencionado anteriormente, o LabSensing reduz a quantidade de dados injetados na rede através de processamento nas bordas. Para tal, são definidas duas políticas para a transmissão de dados que são simplesmente o envio periódico de mensagens e o envio de mensagens baseado em limiar. A segunda política apenas envia dados quando uma alteração acima de um limiar predefinido é detectada, levando em conta os dados gerados no ambiente sensoreado. A política é implementada diretamente no código do nó pertinente. No caso de teste, o nó busca detectar quaisquer mudanças de estado em seus sensores: uma porta abrindo ou fechando, um usuário introduzindo um cartão RFID no leitor ou uma variação significativa de temperatura. A ideia do envio baseado em limiar é reduzir de forma simples a carga na rede.

Camada de armazenamento e exibição dos dados: Executada em um nó central, esta camada necessita de um cliente MQTT para obter os dados produzidos na borda, um banco de dados para garantir persistência dos dados recebidos e uma ferramenta para visualização. Assim, o Node-RED, ferramenta de programação visual criada com o intuito de agir como intermediário entre diferentes protocolos, APIs (*Application Programming Interface*) e hardwares, é usado como nó de concentração. No LabSensing, o cliente

Node-RED está inscrito nos tópicos onde os nós de percepção publicam suas leituras. O nó de concentração ainda processa a informação recebida para posteriormente escrever na base de dados. Os dados são recebidos em formato JSON (*JavaScript Object Notation*) com campos do tipo *string* e são passados em formato JSON com tipos corretos para a base de dados. A implementação do *flow* Node-RED desenvolvido pode ser encontrada em um repositório GitHub [Bochie, 2019].

A ferramenta também pode ser utilizada para exibir as informações recebidas em um painel acessado via navegador *web*. Dessa forma, uma ferramenta de visualização é utilizada para permitir que um usuário do sistema analise quaisquer alterações no ambiente. Caso algum sensor detecte situações que necessitem de atuação de um administrador, alarmes podem ser disparados.

Os dados coletados para posteriormente serem exibidos são armazenados em uma base de dados especializada em dados temporais. A base de dados InfluxDB, desenvolvida pela InfluxData [InfluxData, 2019], é usada por ser uma plataforma aberta, desenvolvida para recuperação rápida de dados. Essas características a habilitam para o uso em aplicações de IoT. Já o *dashboard*, definido na arquitetura, é outra ferramenta desenvolvida pela InfluxData, chamada Grafana, desenvolvida para realizar consultas a bases de dados temporais. Uma página *web* é usada para visualizar o conteúdo de cada base de dados. O Grafana também possibilita a alteração dos intervalos temporais de interesse, o que facilita a detecção de erros na transmissão de mensagens e na base de dados.

3. Resultados Experimentais

Dois nós de sensoriamento (Nó Assíncrono e Nó Síncrono) foram projetados para medir a diferença de desempenho do sistema com as duas políticas de transmissão implementadas. Em ambos os casos, os nós possuem o mesmo hardware, mas operam de forma descrita na Seção 2. Os dados foram coletados ao longo de 24 horas no laboratório GTA (Grupo de Teleinformática e Automação) da Poli/COPPE, UFRJ. Porém, os dados gerados pelo Nó Assíncrono não foram coletados em intervalos regulares. As políticas de transmissão dos nós sensores tiveram seus parâmetros alterados para análise do impacto de tais políticas na redução dos dados coletados nas bordas.

A Tabela 2 mostra o número de mensagens inseridas na base de dados no servidor central em função das políticas de transmissão. O Nó Síncrono foi programado para entrar em modo *deep sleep* em intervalos regulares definidos por um parâmetro no código do nó. As medidas exibidas na Tabela 2 foram coletadas durante o mesmo período de 2 horas em dias úteis no laboratório. Já o Nó Assíncrono foi programado para enviar mensagens caso detecte eventos e com o limiar de variação de temperatura ajustado para 1^oC. O número de mensagens injetadas na rede é estimado a partir do número de entradas na base de dados no servidor que, para o Nó Síncrono, resulta em um valor exato já que o nó envia dados em intervalos conhecidos. Caso haja um intervalo sem a inserção de uma nova entrada na base de dados, isso pode ser assumido como uma perda na transmissão já que as mensagens usam QoS 0. No caso do Nó Assíncrono, o número de mensagens injetadas na rede é uma estimativa, já que a base de dados apresenta apenas as mensagens enviadas com sucesso e não há como saber quando uma nova mensagem é gerada. Para detectar perdas, seria necessário registrar também as mensagens enviadas no nó transmissor, o que ainda não é feito neste trabalho. Assume-se que isso não é impactante já que a rede não

apresenta perdas relevantes em sua operação normal.

A comparação dos resultados entre os Nós Síncrono e Assíncrono mostra que o uso da política de redução usada no Nó Assíncrono permite uma redução significativa no número de entradas na base de dados. Em relação ao *deep sleep* usado no Nó Síncrono, percebe-se que há uma redução esperada conforme aumenta-se o intervalo de economia de energia. Vale ressaltar que a mudança no intervalo do *deep sleep* do Nó Síncrono não tem nenhuma relação influência no Nó Assíncrono. Os números diferentes listados na tabela relativos ao Nó Assíncrono são uma mera consequência do número de eventos observados durante os experimentos.

Tabela 2. Número de entradas na base de dados para as duas políticas de transmissão.

Nó Síncrono	Nó Assíncrono
198 (5 segundos em *deep sleep*)	9
165 (10 segundos em *deep sleep*)	23
129 (15 segundos em *deep sleep*)	16

A Figura 3 apresenta os dados coletados pelo Nó Assíncrono e pelo Nó Síncrono durante 4,5 horas em uma tarde de funcionamento regular do laboratório. O intervalo foi escolhido por mostrar bem como os nós refletem os dados medidos durante um período de variação de temperatura. Nota-se o nível de granularidade superior do Nó Síncrono, o que é esperado, já que a política empregada por este nó envia todas as mensagens geradas. Entretanto, mesmo com um número inferior de mensagens, o Nó Assíncrono consegue acompanhar as medidas do Nó Síncrono com um número bem menor de mensagens enviadas na rede. As duas políticas, porém, não são excludentes já que a decisão entre qual utilizar depende da aplicação sensoreada.

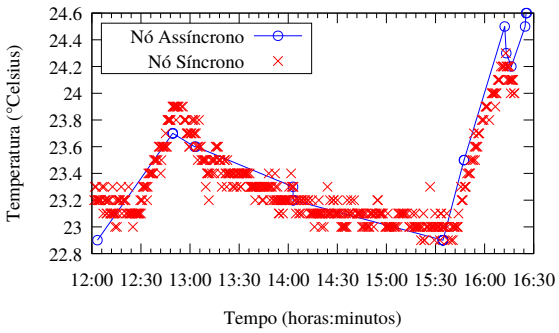


Figura 3. Mensagens inseridas na base de dados pelos Nós Assíncrono e Síncrono.

O consumo de energia dos dois nós foi monitorado por 24 horas durante o funcionamento regular do laboratório. O Nó Síncrono consume 795 miliwatts-hora em 24 horas. Este valor foi obtido enquanto o nó opera em estado *deep sleep* por 15 segundos entre transmissões e em estado normal por 7 segundos durante coleta de dados e transmissões. O Nó Assíncrono consumiu 1750 miliwatts-hora em 24 horas. A diferença de consumo entre os dois nós para os parâmetros selecionados é de 955 miliwatts-hora por dia. Observa-se que o consumo do Nó Assíncrono é maior devido a necessidade de monitorar o ambiente sem intervalos para dormir. Essa política de transmissão se justifica

caso seja necessário detectar eventos e processá-los sem risco de perda. Uma política híbrida, síncrona e assíncrona, pode ser vislumbrada dependendo das características do ambiente monitorado, por exemplo, o sensor pode usar a política assíncrona durante o dia e a síncrona durante a noite.

4. Conclusão e Trabalhos Futuros

Este trabalho apresentou o LabSensing, um sistema para monitoramento de laboratórios científicos. O sistema proposto é suficientemente genérico para ser utilizado em diferentes cenários e aplicações. O sistema é composto por nós sensores, encaminhamento de mensagens usando a abordagem *Publish/Subscribe* e armazenamento e exibição de dados em servidor centralizado.

A escalabilidade do sistema é concluída, visto que *Brokers* MQTT podem suportar milhares de conexões concorrentes e que o sistema proposto reduz o número de conexões feitas ao *Broker*. Os resultados avaliaram duas políticas de transmissão implementadas nos nós sensores localizados nas bordas da rede. As políticas implementadas foram: enviar mensagens periódicas até o nó consumidor ou enviar mensagens apenas se o valor medido ultrapassar um limiar predefinido. As medidas indicaram que a quantidade de mensagens enviadas é de fato reduzida pela transmissão por limiar, mas que se esses nós não forem capazes de dormir, o seu consumo de energia se torna superior. Os trabalhos futuros permitirão testar outras políticas, tais quais intervalos estimados por algoritmos de regressão linear implementados nos nós de sensoriamento e políticas estimadas por técnicas de *Machine Learning* implementadas no servidor, além de expandir a rede de testes já implementada.

Referências

Alqinsi, P., Matheus Edward, I. J., Ismail, N. e Darmalaksana, W. (2018). IoT-Based UPS monitoring system using MQTT protocols. Em *2018 4th International Conference on Wireless and Telematics (ICWT)*, p. 1–5.

Bochie, K. (2019). Labsensing. `https://github.com/kaylani2/labsensing`. Acessado em 19/03/2019.

Eclipse, F. (2019). Eclipse Mosquitto. `https://mosquitto.org/`.

InfluxData (2019). Influxdata. `https://www.influxdata.com/`. Acessado em 19/03/2019.

Kodali, R. K. e Gorantla, V. S. K. (2017). Weather tracking system using MQTT and SQLite. Em *2017 3rd International Conference on Applied and Theoretical Computing and Communication Technology (iCATccT)*, p. 205–208.

Oasis (2014). MQTT Version 3.1.1. Relatório técnico, Oasis.

Saha, S. e Majumdar, A. (2017). Data centre temperature monitoring with ESP8266 based Wireless Sensor Network and cloud based dashboard with real time alert system. Em *2017 Devices for Integrated Circuit (DevIC)*, p. 307–310.

Shi, W., Cao, J., Zhang, Q., Li, Y. e Xu, L. (2016). Edge computing: Vision and challenges. *IEEE Internet of Things Journal*, 3(5):637–646.

HyPER: Heurística de deposição de infraestruturas auxiliares para Redes Veiculares

Pedro H. Souza[1], Massilon L. Fernandes[1], Thiago S. Gomides[1], Fernanda S.H. Souza[1], Cristiano M. Silva[2,3] e Daniel L. Guidoni[1]

[1]Departamento de Ciência da Computação (DCOMP/UFSJ)
[2]Departamento de Tecnologia pela UFSJ (DTECH/UFSJ)
Universidade Federal de São João Del Rei (UFSJ), MG – Brasil
[3]Departamento de Computação (DECOM/UFOP)
Universidade Federal de Ouro Preto (UFOP), MG – Brasil
{pedronhds,massilon,gomides,fsumika,cristiano,guidoni}@ufsj.edu.br

Abstract. *In this work the HyPER algorithm is presented. This algorithm consists of the combination of two combinatorial optimization techniques, GRASP and Path Relinking, to solve the communication infrastructure (RSU) deployment problem using the Deployment $\Delta\binom{\rho^1}{\rho^2}$ metric. This metric aims to establish communication guarantees based on the duration of contact between vehicles and the infrastructures. HyPER was compared to a greedy strategy and the results show that HyPER uses a smaller number of RSUs considering the same performance requirements.*

Resumo. *Neste trabalho é apresentado o algoritmo HyPER. Esse algoritmo consiste na combinação de duas técnicas de otimização combinatória, GRASP e Path Relinking, para resolver o problema de distribuição de unidades de comunicação (RSU) utilizando a métrica Deposição $\Delta\binom{\rho^1}{\rho^2}$. Essa métrica visa estabelecer garantias de comunicação a partir do tempo de duração do contato entre os veículos e as infraestruturas auxiliares de comunicação. O HyPER foi comparado com uma estratégia gulosa e os resultados mostram que o HyPER utiliza um número menor de RSUs considerando os mesmos requisitos de desempenho.*

1. Introdução

O agrupamento populacional e o desenvolvimento tecnológico são alguns dos fatores responsáveis pelo constante fluxo de migração das pessoas para as grandes cidades [Mabogunje 1970]. Este crescimento em conjunto com a falta de planejamento urbano são elementos prejudiciais ao bem-estar e a qualidade de vida das pessoas, tornando as cidades ambientes mais estressantes e menos produtivos [Ekblad 1993]. Estima-se, segundo [Van Audenhove et al. 2014], que no ano de 2014, 53% da população mundial estava concentrada nas grandes cidades, devido as melhores oportunidades. No entanto, a previsão é que em 2050 tenhamos um crescimento de 14%, prejudicando ainda mais a vida da população.

Nesse sentido, com objetivo de minimizar os reflexos da concentração populacional, a incorporação de tecnologias de sensoriamento, análise, controle e comunicação nos centros urbanos têm sido frequentemente discutidas [Bazzan and Klügl 2007]. Essas

tecnologias, têm por objetivo proporcionar melhorias na infraestrutura urbana e, principalmente, condições que tornem as grandes cidades ambientes mais eficientes e melhores para se viver. Assim, as cidades inteligentes – *Smart cities*, são alternativas tecnológicas que buscam minimizar os efeitos proporcionados pelo constante fluxo de migração populacional, com menores impactos na rotina das pessoas.

Nesse cenário, populariza-se nos grandes centros, a implantação dos Sistemas Inteligentes de Transporte (STI) que propõe alternativas capazes de promover melhorias na mobilidade urbana. Para isso, os STI, a partir das Redes Veiculares (VANETs), têm como objetivo desenvolver sistemas de gerenciamento de tráfego a partir de redes de comunicação altamente sofisticadas que interagem a partir de suas entidades [Silva et al. 2016], como: veículos, pedestres, sensores de mobilidade, semáforos e infraestruturas auxiliares de comunicação (*Roadside Unit* - RSU). As VANETs, podem ser classificadas de acordo com o tipo de comunicação utilizada: comunicação entre veículos (V2V), entre veículos e infraestruturas auxiliares (V2I) e o modelo de comunicação híbrido (V2X) que combina as abordagens V2V e V2I.

A comunicação infraestruturada – V2I, proporciona melhorias na conectividade da rede veicular, no entanto, possui um alto custo para sua implantação. Dessa maneira, este trabalho tem como objetivo propor uma técnica de otimização fundamentada no uso de heurísticas, capazes de minimizar a quantidade de infraestruturas necessárias para criar uma rede V2I. Nesse sentido, dado um cenário urbano, nossa proposta **HyPER**: **H**eurística de de**P**osição de infra**E**struturas auxiliares para **R**edes Veiculares, consiste na aplicação de duas técnicas de otimização combinatória, GRASP e Path Relinking, para resolver o problema de deposição de RSUs em Redes Veiculares Infraestruturadas. As seções deste artigo estão organizadas como segue. Os trabalhos relacionados são apresentados na seção 2. A métrica de garantias de comunicação, Deposição $\Delta\binom{\rho^1}{\rho^2}$, e a representação da malha rodoviária são expostas nas seções 3 e 4, respectivamente. A solução proposta é apresentada na seção 5 e a estratégia gulosa (VpV) na seção 6. A comparação dos algoritmos é apresentada na seção 7 e a conclusão do trabalho é apresentada na seção 8.

2. Trabalhos Relacionados

Na literatura para as redes veiculares, soluções heurísticas têm sido propostas para minimizar os efeitos dos problema de alocação mínima de infraestruturas auxiliares de comunicação [Sarubbi and Silva 2016], [Silva et al. 2016], e [Silva et al. 2017]. Nesse sentido, no desenvolvimento dessas soluções, os autores propõem métricas capazes de proporcionar garantia de comunicação em conjunto a redução do número de RSUs e, consequentemente, a redução do custo de implantação e manutenção da rede de comunicação.

Os autores em [Silva et al. 2016], propõem para a deposição de infraestruturas a métrica denominada Deposição $\Gamma_D\binom{\tau}{\rho^1}$ (Gamma Deployment). Nesse sentido, avalia-se a distribuição de infraestruturas de comunicação em relação ao tempo entre contatos dos veículos e as RSUs. Além disso, essa métrica, garante que a porcentagem ρ^1 de veículos estejam em frequente contato com as RSUs por no máximo τ segundos de diferença. Segundo os autores, o limite máximo entre contatos permite aos administradores da rede o monitoramento e a detecção de incidentes de trânsito com maior eficiência, além de permitir recomendações das condições do tráfego.

Em [Sarubbi and Silva 2016], os autores descrevem o problema de deposição de infraestruturas como o Problema de Localização de Máxima Cobertura (PLMC). Para isso, uma heurística gulosa de deposição de RSUs é construída sob o modelo de restrições denominado Deposição $\Delta\binom{\rho^1}{\rho^2}$ (Delta-Deployment). Nesse sentido, a deposição delta têm como objetivo garantir que a conexão entre ρ^2 porcento dos veículos estejam conectados às infraestruturas auxiliares durante uma porcentagem ρ^1 to tempo total de viagem. Os resultados obtidos indicam que a solução é capaz de diminuir o número de infraestruturas necessárias ao utilizar o tempo de contato relativo entre veículos e infraestrutura. Em [Silva et al. 2017], é proposto uma variação da métrica Delta-Deployment, onde é avaliado múltiplas regiões, permitindo garantir diferentes Qualidade de Serviços (QoS) para cada região.

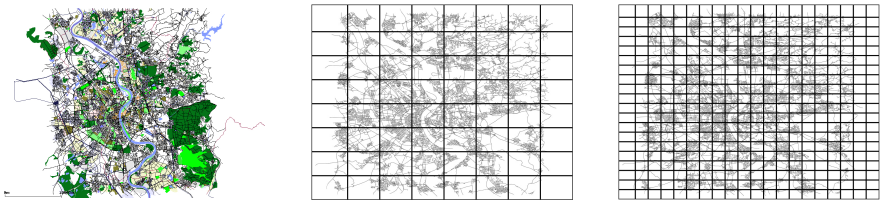
3. Deposição $\boldsymbol{\Delta\binom{\rho^1}{\rho^2}}$

Neste trabalho, para avaliar a distribuição das infraestruturas de comunicação, utiliza-se a métrica Deposição $\Delta\binom{\rho^1}{\rho^2}$ [Silva and Meira 2015] que, por princípio, estabelece garantias de comunicação a partir do tempo de duração do contato entre os veículos e às infraestruturas auxiliares. Essas garantias se dão por meio dos parâmetros ρ^1 e ρ^2. Dessa forma, uma alocação de infraestruturas satisfaz a Deposição $\Delta\binom{\rho^1}{\rho^2}$, se ρ^2 porcento dos veículos tem tempo de contato com as RSUs por no mínimo ρ^1 porcento dos seus tempos totais de viagem. A apresentação formal da Deposição $\Delta\binom{\rho^1}{\rho^2}$ é exposta na Definição 1.

Definição 1: *Seja M uma representação de uma malha rodoviária, V =* $\{v_1, v_2, ..., v_n\}$ *um conjunto dos veículos que viajam em M, e T =* $\{C_1, C_2, ..., C_n\}$ *a trajetória de cada veículo* $v_n \in V$. *Assim, para um determinado veículo* $v_n \in V$, *existe uma trajetória* $C_n \in T$. *Cada* $C_n \in T$ *representa um subconjunto de células urbanas* $C_n = \{u_1^n, u_2^n, ..., u_k^n\}$ *percorridas por um veículo* v_n *durante sua viagem. Seja* $V' \subset V$ *o subconjunto de veículos* v_n *no cenário que permanecem conectados com RSUs durante o tempo* $\geq \rho^1, \forall\ u \in C_n$. *Essa deposição é considerada* $\Delta\binom{\rho^1}{\rho^2}$ *se* $\frac{|V'|}{|V|} \geq \rho^2$.

4. Representação da Malha

Os ambientes urbanos possuem topologias complexas devido aos variados tamanhos e formatos das vias públicas, que são diferentes para cada cidade. Nesse sentido, são numerosas as possibilidades para a alocação das infraestruturas e, consequentemente, torna o problema de deposição de RSUs complexo e de alto custo computacional para resolução. Nesse sentido, para reduzir as regiões candidatas na alocação de infraestruturas, o particionamento da área urbana em um conjunto de ψ x ψ células urbanas retangulares de mesma dimensão é aplicado ao nosso trabalho. Essa solução é proposta em alguns trabalhos presentes na literatura como [Silva and Meira 2015, Rocha Silva et al. 2016]. Esta simplificação não representa a cobertura de comunicação das RSUs, no entanto, aplica-se esta técnica para reduzir as regiões com possibilidades de alocação. Dessa forma, se posicionarmos uma RSU em uma célula u, todos os veículos que passarem por u terão cobertura de comunicação. Além disso, o número de células é proporcional à precisão requisitada, assim, para maiores níveis de precisão, aloca-se um maior no número de células urbanas. Como exemplo, as Figuras 1(b) e 1(c) ilustram o particionamento em 8×8 e 20×20 células urbanas do mapa rodoviário da cidade de Colônia, Alemanha, sob uma região de aproximadamente 400km^2 (Figura 1(a)). Nesse sentido, as trajetórias de cada veículo apresentadas na seção 3 serão definidas por um conjunto de células urbanas.



(a) Mapa da cidade (b) *Grid* 8×8 (c) *Grid* 20×20

Figura 1. Mapa de células urbanas

5. HyPER

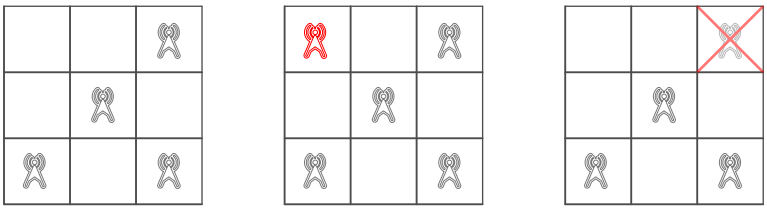
Nesta seção apresentamos a estratégia HyPER que, por objetivo, busca promover garantias de comunicação entre veículos e infraestruturas auxiliares em conjunto com a minimização dos custos associados à alocação de cada RSU. Nesse sentido, a heurística apresentada pode ser dividida em dois módulos, sendo esses, GRASP e *Path Relinking*. No primeiro módulo, para estabelecer garantias de comunicação, a construção de soluções iniciais viáveis é realizada a partir da métrica Deposição $\Delta\binom{\rho^1}{\rho^2}$, em seguida, aplica-se busca local para o seu refinamento e, consequentemente, melhora da solução. No segundo módulo, duas das melhores soluções do primeiro módulo são escolhidas e utilizadas para se obter uma outra solução com melhores resultados que as duas utilizadas (*Path relinking*). Os módulos apresentados são explicados com maiores atenções nas seções 5.1 e 5.2.

5.1. GRASP

O GRASP (*Greedy Randomized Adaptive Search Procedure*) é um procedimento de busca guloso, adaptativo e aleatório, desenvolvido para a resolução dos problemas de otimização combinatória a partir da construção de um conjunto de soluções promissoras. Nesse sentido, o GRASP constrói suas soluções por meio de um procedimento de execuções independentes, em outras palavras, a cada iteração uma nova solução é gerada. Além disso, a heurística é constituída de duas fases: (5.1.1) Fase de construção e (5.1.2) Busca Local. Na fase (5.1.1), são construídas soluções iniciais viáveis para o problema. No entanto, não existe garantias de que uma solução que respeita os critérios de viabilidade é uma boa solução. Nesse sentido, é aplicado na etapa (5.1.2) o refinamento das das soluções iniciais. Assim, para cada solução, a busca local cria soluções vizinhas na busca por ótimos locais. Além disso, o número de execuções do GRASP é definido previamente e, as duas fases descritas, são executadas a cada iteração.

5.1.1. Fase construtiva

Nesta fase, é realizada a construção de soluções parciais viáveis elemento por elemento considerando a Lista Restrita de Candidatos (RCL). Desta forma, a RCL é uma lista que contém os elementos de melhor benefício que podem ser adicionados à solução em construção. Este é um procedimento construtivo pseudo-guloso controlado pelo parâmetro de aleatoriedade α. Nesse sentido, o fator α, assume valores no intervalo $0 \leq \alpha \leq 1$, sendo 0 uma escolha totalmente gulosa e 1 uma escolha totalmente aleatória. Destaca-se que, para $(\alpha > 0)$, não existe garantias da escolha do melhor elemento, promovendo assim diversificação na construção da solução inicial. Além disso, a RCL é composta dos *elementos candidatos* ordenados de acordo com os benefícios associados.



(a) Solução inicial (b) Operação *ADD* (c) Operação *DROP*

Figura 2. Vizinhança da Busca local

Neste trabalho, os *elementos candidatos* são representados pelo conjunto de células urbanas que ainda não possuem RSU. Assim, o benefício associado na incorporação de uma célula à solução está relacionado ao número de veículos cobertos por ela. Dessa forma, quanto maior o número de veículos que passam por uma célula, melhor será o seu benefício para a solução e, quanto maior o tamanho da RCL, maior será o número de possibilidades para construção das soluções.

5.1.2. Busca Local

Dada uma solução inicial S_0, nesta etapa, o GRASP intensifica a exploração da vizinhança $N1$ de S_0 em busca dos ótimos locais. Destaca-se que, a vizinhança dessa solução consiste em conjuntos das células urbanas da vizinhança $N1$ que atendem as restrições determinadas pela Deposição $\Delta\binom{\rho^1}{\rho^2}$ [Sarubbi et al. 2017]. Como exemplo, na Figura 2, são apresentadas as operações *Add* e *Drop* utilizadas para intensificação da solução inicial. Assim, cada quadrante representa uma célula urbana que pode ou não conter uma RSU. As RSUs em cinza fazem parte da solução inicial depositadas pela fase construtiva, e a RSU em vermelho foi adicionada pela operação *Add*. Durante essa fase, a solução inicial (figura 2(a)) têm sua vizinhança $N1$ completamente explorada célula por célula a cada iteração. Além disso, nos casos que uma célula u não possui infraestrutura de comunicação, uma RSU é depositada, operação *Add* (Figura 2(b)). Por outro lado, se existe uma RSU em u, ela é removida a partir da operação *Drop* (Figura 2(c)).

Após cada operação (*Add* ou *Drop*), calcula-se a função objetivo (fo) da nova solução e, após o calculo da fo, a operação aplicada é revertida. A melhor solução da vizinhança de S_0 é armazenada devido a estratégia de melhor aprimorante. Além disso, a fo é definida pela minimização do número de RSUs em conjunto com a quantidade de veículos descobertos por essa deposição. Após as operações, para as soluções inviáveis, um fator de penalização é adicionado. Este procedimento, é feito para todas as células urbanas e, para cada operação, o valor da fo e a posição (x, y) dá célula modificada são armazenados. Ao fim das iterações, a operação que proporcionou os melhores ganhos para fo, é realizada de fato. Além disso, este processo é repetido até que o critério de parada seja atingido, ou seja, não exista mais melhoras para fo.

5.2. *Path Relinking*

O *Path-relinking* é uma estratégia de intensificação de busca que aprimora as soluções provenientes das etapas de busca local. Para isso, o *Path Relinking* utiliza duas soluções, S_1 e S_2, denominadas solução fonte e solução guia. O objetivo é partir da solução fonte e chegar na solução guia por meio da adição de atributos da solução guia à solução fonte, onde a solução guia possui um resultado melhor que a fonte. A figura 3 exemplifica como é o caminho feito pelo *Path Relinking*. Primeiro são tomadas duas soluções, a solução

fonte, em verde, e a solução guia, em vermelho. A partir da solução fonte, são geradas algumas possíveis soluções, em azul. Essas possibilidades são geradas através da adição de atributos da solução guia à solução fonte. Uma possibilidade é adicionar apenas 1 atributo à solução fonte, ou seja, para cada elemento diferente da solução guia aplicado à solução fonte é criada uma possibilidade diferente. As soluções apresentadas neste artigo são matrizes com RSUs depositadas em suas células, e a adição de atributos da solução guia se dá por meio da adição ou retirada de RSUs em determinadas células urbanas da solução fonte. Todos esses candidatos gerados a partir da solução fonte são avaliados. O melhor é então selecionado, em rosa, e este se torna a nova solução fonte. Todo esse procedimento é repetido até que um critério de parada ocorra. Um possível critério de parada é a solução fonte se tornar igual a solução guia. Outro eventual critério de parada é caso as soluções geradas na iteração i não forem melhores que a melhor solução da iteração $i-1$. A possibilidade de gerar soluções com melhores resultados que as duas soluções escolhidas, fonte e guia, é que torna interessante a aplicação desse algoritmo.

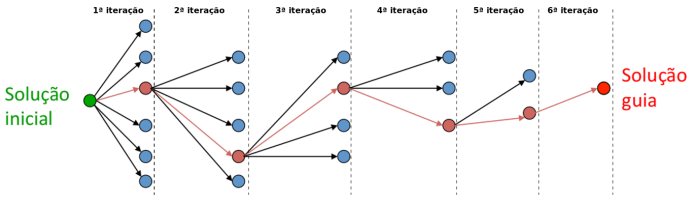


Figura 3. Caminho da solução fonte à solução guia.

6. Algoritmo base: VpV - Veículo por Veículo

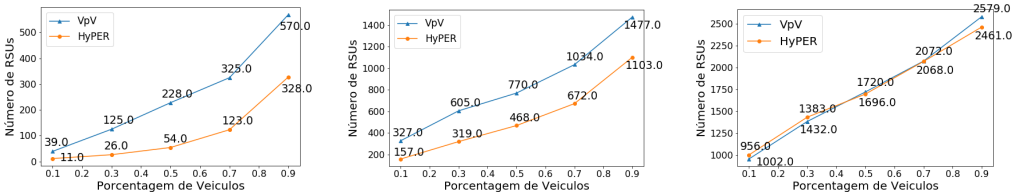
A ideia por trás da solução VpV consiste em iterativamente, de acordo com os parâmetros de entrada requeridos pela Deposição $\Delta\binom{\rho^1}{\rho^2}$, avaliar cada veículo separadamente e adicionar RSUs para cobrir o veículo em questão. Um ponto que se deve observar é que as RSUs utilizadas para um veículo atingir a cobertura miníma podem ser suficientes para outros veículos também ficarem cobertos pela deposição. Desse modo, após garantir a cobertura para um determinado veículo, os demais veículos são verificados com base nas novas RSUs adicionadas. O algoritmo termina quando é atingido a garantia miníma requerida pela deposição.

7. Análise de Resultados

Nesta seção, a comparação entre o HyPER e o algoritmo VpV é apresentada. Para isso, foi escolhido o *trace* (rastro) realístico da cidade de Colônia, Alemanha, contendo 75.515 veículos sob um trecho viário de 702 km^2. A mobilidade dos veículos, foi simulada a partir da ferramenta SUMO 0.30[1], bem como a localização de cada veículo durante todo o seu percurso. Esta localização, é obtida considerando as coordenadas cartesianas e, convertendo-as em células urbanas. Além disso, o cenário de Colônia foi particionado em um conjunto de 100×100 células urbanas, assim como apresentado nos trabalhos da seção 2, resultando em células de $270m \times 260m$.

O desempenho do Hyper é avaliado em relação à estratégia VpV, considerando a minimização do número de infraestruturas auxiliares que mantenham as garantias de comunicação definidas por $\Delta\binom{\rho^1}{\rho^2}$. As duas estratégias foram comparadas em relação ao

[1]http://sumo.dlr.de/



(a) $\rho^1 = 10\%$ (b) $\rho^1 = 50\%$ (c) $\rho^1 = 90\%$

Figura 4. O número de RSUs depositadas por HyPER e VpV. O eixo x representa a quantidade de veículos ρ^2, enquanto que o eixo y indica o número de infraestruturas alocadas.

número de RSUs que satisfaçam os parâmetros ρ^1, referente à porcentagem do tempo de viagem que os veículos devem permanecer conectados; e ρ^2, a porcentagem de veículos que devem estar conectados durante um intervalo maior ou igual a ρ^1. Para avaliar o comportamento dos algoritmos, considera-se a variação de ρ^1 e ρ^2 para os valores ρ^1: $\{0.1, 0.5 \text{ e } 0.9\}$ e ρ^2: $\{0.1, 0.3, 0.5, 0.7 \text{ e } 0.9\}$. No módulo 5.1, GRASP, foram executadas 10 iterações e, consequentemente, dez soluções considerando a Deposição $\Delta\binom{\rho^1}{\rho^2}$. As cinco melhores soluções foram selecionadas para aplicarmos a intensificação da busca *Path Relinking*, em uma combinação simples $C_{5,2}$, ou seja, dez soluções para o problema.

A figura 4 apresenta os resultados após a execução das duas fases do HyPER e da solução VpV. As figuras 4(a), 4(b) e 4(c) mostram os resultados para o conjunto de parâmetros ρ^1 igual à 10%, 50% e 90%, respectivamente. Para a figura 4(a), de tempo de contato igual à 10% durante as viagens, observamos que a nossa solução reduz consideravelmente o número de infraestruturas depositadas. Assim, para $\Delta\binom{0.1}{0.1}$ e $\Delta\binom{0.1}{0.9}$, o HyPER alocou, respectivamente, 71% e 42% menos RSUs que VpV. Além disso, ao considerarmos um tempo de contato de $\rho^1 = 50\%$, o HyPER para $\Delta\binom{0.5}{0.1}$, posiciona 52% menos infraestruturas comparado ao VpV e para $\Delta\binom{0.5}{0.9}$ posiciona 25% a menos também comparado ao VpV. Além disso, quando o cenário exige elevados tempos de contato, como para $\rho^1 = 90\%$, o HyPER aumenta em 4% considerando $\Delta\binom{0.9}{0.1}$ e, proporciona redução em 4% ao avaliarmos $\Delta\binom{0.9}{0.9}$, ambos em relação ao VpV. Em ambientes que exigem elevados tempos de contato como $\Delta\binom{0.9}{0.9}$, considera-se quase por completo o percurso dos veículos o que reduz as possibilidades para células senão quase o percurso inteiro dos veículos. O desempenho do HyPER em relação ao VpV é proveniente das otimizações combinatórias que o HyPER utiliza. A busca local do GRASP adiciona RSUs em células que maximizam o tempo de contato dos veículos e retira das células de baixo impacto para a cobertura do conjunto de veículos. Já o *Path Relinking* intensifica os ótimos locais encontrados pelo método GRASP por meio da incorporação de atributos de uma solução encontrada em outra, podendo então encontrar melhores soluções.

8. Conclusões

Neste trabalho foi apresentada a estratégia HyPER, que consiste na combinação de duas técnicas de otimização combinatória, GRASP e *Path Relinking* para resolver o problema de minimização da quantidade de RSUs para atender a métrica de Deposição $\Delta\binom{\rho^1}{\rho^2}$. Para comparar com o HyPER, foi utilizada a estratégia VpV, explicada na seção 6. Como apresentada na seção 7, nossa estratégia realiza a deposição de menos RSUs na maioria dos cenários em comparação com VpV para as instâncias apresentadas. Esse resultado é explicado devido ao modo como o HyPER funciona. A busca local do GRASP pro-

cura sempre encontrar ótimos locais ao procurar pela vizinhança de determinada solução viável, e o *Path Relinking* visa sempre procurar encontrar outros ótimos locais mudando o ponto de busca através da combinação de atributos de duas soluções que já são ótimos locais. Como trabalhos futuros, pretendemos avaliar a solução utilizando outros traces e cenários.

Agradecimentos

Os autores gostariam de agradecer a Fundação de Amparo à Pesquisa do Estado de Minas Gerais (FAPEMIG) sob concessão No.: APQ-03120-17 e a Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) sob concessão No.: 150545/2018-5. O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001.

Referências

Bazzan, A. L. and Klügl, F. (2007). Sistemas inteligentes de transporte e tráfego: uma abordagem de tecnologia da informação. *Anais das Jornadas de Atualização em Informática*, pages 2296–2337.

Ekblad, S. (1993). Stressful environments and their effects on quality ot life in third world cities. *Environment and Urbanization*, 5(2):125–134.

Mabogunje, A. L. (1970). Systems approach to a theory of rural-urban migration. *Geographical Analysis*, 2(1):1–18.

Rocha Silva, T., Sarubbi, J., Martins, F., and Silva, C. (2016). Algoritmos baseados na metaheurística grasp para implantação de unidades de comunicação em redes veiculares garantindo qualidade de serviço. *XLVIII Simposio Brasileiro de Pesquisa Operacional (SBPO), At Vitória, Espiríto Santo, Brazil*.

Sarubbi, J. F., Silva, T. R., Martins, F. V., Wanner, E. F., and Silva, C. M. (2017). A grasp based heuristic for deployment roadside units in vanets. In *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pages 369–376. IEEE.

Sarubbi, J. F. M. and Silva, C. M. (2016). Delta-r: A novel and more economic strategy for allocating the roadside infrastructure in vehicular networks with guaranteed levels of performance. In *Network Operations and Management Symposium (NOMS), 2016 IEEE/IFIP*, pages 665–671. IEEE.

Silva, C. M. and Meira, W. (2015). Design of roadside communication infrastructure with qos guarantees. In *2015 IEEE Symposium on Computers and Communication (ISCC)*, pages 439–444. IEEE.

Silva, C. M., Pitangui, C. G., Guidoni, D. L., Souza, F. S., and Sarubbi, J. F. (2016). Deposição gamma: Alocando infraestrutura de comunicação para redes veiculares garantindo o intervalo" entre contatos" de veículos com a infraestrutura de comunicação. *XXXIV Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos - SBRC*.

Silva, C. M., Silva, F. A., Sarubbi, J. F., Oliveira, T. R., Meira Jr, W., and Nogueira, J. M. S. (2017). Designing mobile content delivery networks for the internet of vehicles. *Vehicular communications*, 8:45–55.

Van Audenhove, F.-J., Korniichuk, O., Dauby, L., and Pourbaix, J. (2014). The future of urban mobility 2.0: Imperatives to shape extended mobility ecosystems of tomorrow.

Uma estratégia de leilões combinatórios para provisionamento de serviços virtualizados

Vanessa S. Vieira[1], **Andre L. L. Aquino**[1]

[1]Instituto de Computação – Universidade Federal de Alagoas (UFAL)

{vsv,alla}@ic.ufal.br

Abstract. *In this work, we study the use of a game theory-based auctioning strategy for provisioning virtualized services in mobile networks. This strategy aims to absorb the peak demands of mobile services clients. In this sense, we use a global strategy to assist the operators in choosing which nodes will be receiving the requested services; and a place to choose nodes that will serve as edge elements providing service to nodes that do not have access to the infrastructure. The results showed that our approach benefits the operator and the client by maintaining the equilibrium of the system.*

Resumo. *Neste trabalho, estudamos o uso de uma estratégia de leilões baseada em teoria dos jogos para provisionamento de serviços virtualizados em redes móveis. Essa estratégia visa absorver as demandas de pico de clientes de serviços móveis. Nesse sentido, utilizamos uma estratégia global para auxiliar as operadoras na escolha de quais nós estarão recebendo os serviços requisitados; e uma local para escolher nós que servirão de elementos de borda provendo serviço aos nós que não possuem acesso à infraestrutura. Os resultados mostraram que nossa abordagem beneficia a operadora e o cliente mantendo o equilíbrio do sistema.*

1. Introdução

A principal questão abordada em nosso trabalho envolve operadoras que desejam vender seus recursos de sobra e clientes que, em um cenário de alta demanda, desejam comprar esses recursos a um preço baixo e justo. A nossa solução utiliza uma estratégia de leilões para fornecimento de serviços em uma rede de telefonia móvel virtualizada [Han et al. 2015]. Nossa abordagem utiliza conceitos de teoria dos jogos e apresenta duas etapas, alocação de serviços global e local.

Na alocação global, alocamos os serviços virtualizados a vários clientes de várias operadoras de rede, aplicando um leilão combinatório. Este leilão determina os vencedores e seus respectivos preços com base nos lances ofertados. Cada lance é composto por um conjunto de serviços virtualizados, chamado de Cadeia de Funções de Serviço (do inglês, *Service Function Chains*, SFC) e um preço de lance. Cada SFC representa um pacote de serviços da rede móvel. Um cliente não pode ganhar um serviço virtualizado separado, mas apenas o SFC inteiro solicitado.

Na alocação de serviços locais, usamos os clientes vencedores da primeira etapa como elementos de borda para os perdedores mais próximos. Eles receberão os incentivos apropriados pelas operadoras permitindo que outros clientes utilizem a infraestrutura de

rede por meio deles. Para executar esta etapa, realizamos um segundo leilão combinatório entre os clientes vencedores do primeiro.

Ambos os leilões combinatórios são problemas considerados NP-difícil, ou seja, nem sempre podemos calcular a solução ótima em tempo polinomial devido à sua natureza exponencial. Assim, utilizamos uma heurística gulosa, que se mostra a melhor aproximação possível em tempo polinomial para o problema da determinação do vencedor, proposta por [Gonen and Lehmann 2000] e estendida por [Obadia et al. 2016]. O pior cenário possível para a soma do preço de lances é $\sqrt{m}$ se m é a soma máxima dos preços de lance atuais em uma solução ótima, sendo portanto a melhor aproximação possível em tempo polinomial [Gonen and Lehmann 2000]. Ajustamos essa heurística nas duas etapas de nossa abordagem.

Para avaliar nossa abordagem, no **Leilão Global de Alocação de Serviços**, inicialmente, comparamos nossa solução com a ótima, considerando um número reduzido de clientes. Em seguida, avaliamos a valoração do mercado, a receita da operadora, o número de clientes atendidos e o preço pago médio apresentado. No **Leilão Local de Alocação de Serviços**, avaliamos o número de clientes locais escolhidos para serem roteadores e a valoração de clientes roteadores variando o número de clientes. Os resultados mostram que nossa abordagem mantém o equilíbrio do sistema permitindo que o mercado funcione de maneira autorregulável, implementando conceitos importantes de teoria dos leilões. Os resultados também demonstram que nosso sistema é justo e um bom modelo de negócios para as operadoras móveis.

2. Trabalhos Relacionados

[Obadia et al. 2016] propôs uma abordagem de Teoria dos Jogos, utilizando Leilões Combinatórios para selecionar os clientes vencedores e o preço que eles pagarão pela cadeia de serviços de funções de rede virtualizadas (do inglês, Virtual Network Functions) solicitada numa rede de computadores tradicional. Já [Umrao et al. 2018] propôs um roteamento de dados móveis por meio de dispositivos inteligentes, explorando as comunicações Dispositivo-a-Dispositivo (do inglês, Device-to-Device, D2D). Nesse caso, a comunicação D2D pôde reduzir significativamente o congestionamento da rede e aprimorar a Qualidade de Serviço (do inglês, Quality-of-Service, QoS) a um custo menor.

Na literatura, alguns estudos propõem soluções sobre virtualização em um ambiente de telefonia celular, usando redes definidas por software (do inglês, Software Defined Network, SDN) por intermédio de uma transmissão multicast de dados móveis [Bukhari et al. 2018], ou controladores de orquestração para fornecer acesso ao cliente [Guerzoni et al. 2014]. Existem estudos sobre mecanismos de leilões para fornecimento de serviços nas redes de telefonia móvel virtualizada que apresentam tendências atuais e desafios abertos para mecanismos de leilão para virtualização em redes móveis [Habiba and Hossain 2018]. [Zhang et al. 2017] propõem um mecanismo de leilão estocástico on-line para provisionamento de cadeia de serviços sob demanda e determinação de preços em um provedor de NFV.

Os modelos de virtualização de rede existentes baseiam-se principalmente em acordos contratuais para regras de precificação e cobrança. Nossa abordagem, por outro lado, incorpora a alocação sob demanda de serviços móveis, onde os clientes podem adquirir capacidades de VNFs por um período limitado e um preço justo, considerando

o estado atual do mercado. Em contraste com as abordagens anteriores, nossa proposta também incentiva os clientes vencedores a compartilhar seus serviços para que os clientes perdedores possam usar a infraestrutura.

3. Leilão de Serviços de Redes Móveis

Nosso sistema de leilão possui três entidades principais: o *operador de infraestrutura*, que fornece serviços de rede na topologia composta por torres de telefonia celular (t) como nós e links de alta capacidade entre torres como enlaces; a *operadora de rede móvel*, que funciona como um nó de retransmissão entre os serviços oferecidos pela infraestrutura de rede e seus *clientes*, modelando os lances dos *clientes* em SFCs de serviços virtualizados e de banda larga; e o *cliente*, o usuário final que solicita serviços móveis.

Para o nosso problema utilizamos os *Leilões Combinatórios*. Os itens leiloados são serviços virtuais e de largura de banda ofertados por um *operador de infraestrutura*. Nós agrupamos esses serviços como SFCs e cada *cliente* tem seu próprio SFC. Com isso, temos um conjunto de serviços $S = \{s_1, s_2, \ldots, s_m\}$, e um conjunto de lances $\mathcal{B} = \{B_1, B_2, \ldots, B_n\}$, em que $B_i = (Q_i, b_i)$ é o lance enviado pelo *cliente* i. O preço apresentado pelo licitante i é denotado por b_i. Denotamos o conjunto de serviços solicitado pelo licitante i como $Q_i = \{q_i^{s_1}, q_i^{s_2} \ldots, q_i^{s_m}\}$, em que $0 \leq q_i^{s_j} \leq C_{s_j}$ indica a quantidade de serviço s_j requerida pelo *cliente* i e C_{s_j} é a capacidade máxima disponível de s_j. A valoração (v_i) é o valor máximo que o licitante i está disposto a pagar. A utilidade de um licitante i é $u_i = v_i - p_i$, onde p_i é o preço do martelo, ou seja, o preço a ser pago. Uma utilidade alta (u_i) para um *cliente* significa que o sistema é economicamente eficiente porque mostra que o preço pago pelo cliente (preço de martelo p_i) é menor que o preço apresentado. O melhor resultado para um licitante é enviar sua valoração correta como preço de oferta, pois esse preço é usado apenas para determinar se um cliente é vencedor ou não em nosso algoritmo.

O problema de encontrar uma alocação ótima em um leilão combinatório é NP-difícil [Gonen and Lehmann 2000]. Para o propósito deste trabalho, avaliamos uma heurística gulosa e um algoritmo exato, provando que nossa heurística trabalha para instâncias maiores e que, para todas as instâncias, a heurística dá uma estimativa muito aproximada do valor real. Podemos descrever este problema pela seguinte formulação de programação inteira binária:

$$\max \quad \sum_{i=0}^{n} x_i b_i \tag{1}$$

$$\text{sujeito a} \quad \sum_{i=1}^{n} x_i q_i^{s_j} \leq C_{s_j} \qquad \forall j \in [1..m] \tag{2}$$

$$x_i \in \{0, 1\} \qquad \forall i \in [1..n] \tag{3}$$

As variáveis binárias x_i assumem o valor 1 se e somente se a solução aceitar o lance i. A função objetiva (1) calcula a soma máxima dos preços de lance aceitos. Nas restrições (2), limitamos a quantidade alocada para o serviço s_j pela sua capacidade C_{s_j}.

Escolhemos uma heurística gulosa para implementação dos leilões combinatórios, proposta por [Gonen and Lehmann 2000] e refinada por [Obadia et al. 2016]. O Algoritmo 1 apresenta a heurística gulosa para determinar os vencedores e seus preços para o leilão global. Ele funciona reordenando os lances do cliente com a métrica dada de entrada ao algoritmo. Em seguida, o problema da determinação de vencedores é abordado, verificando se os lances reordenados, um por um, podem atender às suas solicitações na capacidade de serviço restante na infraestrutura. Depois disso, calculamos os preços usando a equação da linha 14 do Algoritmo 1, onde i é o índice do *cliente* que está tendo seu preço calculado e k é o índice do *cliente* que representa a classificação máxima que i poderia ter obtido enquanto ainda ganhava o leilão, como forma de obter o valor p_i mais justo, sempre menor ou igual à b_i.

Algoritmo 1 Heurística Gulosa para Leilão de Alocação Global de Serviços

Require: reordene os lances sequindo $\frac{b_1}{\sum_{j=1}^{m}\sqrt{q_1^{s_j}}} \geq \frac{b_2}{\sum_{j=1}^{m}\sqrt{q_2^{s_j}}} \geq \ldots \geq \frac{b_n}{\sum_{j=1}^{m}\sqrt{q_n^{s_j}}}, \forall j \in [1...m]$
$U(j) = 0$ representa quantas unidades de serviço disponíveis j foram usadas
Ensure: W_1: conjunto de lances vencedores para o Leilão Global de Alocação de Serviços; e $p_1, \ldots, p_n$: preço a ser pago por cada lance
{*Determinação de Vencedores: Parte 1*}
1: **for** $i = 1$ **to** n **do**
2: **if** $\forall j \in [1..m]\ q_i^{s_j} + U(j) \leq C_{s_j}$ **then**
3: $W_1 \leftarrow i$
4: $\forall j \in [1..m]\ U(j) = U(j) + q_i^{s_j}$
5: **end if**
6: **end for**
{*Computação de Preços: Parte 2*}
7: **for** $i \in W_1$ **do**
8: $\forall j \in [1..m]\ U(j) = 0$
9: **for** $k = 1$ **to** m, $k \neq i$ **do**
10: **if** $\forall j \in [1..m]\ q_k^{s_j} + U(j) \leq C_{s_j}$ **then**
11: $\forall j \in [1..m]\ U(j) = U(j) + q_k^{s_j}$
12: **end if**
13: **if** $\exists j$ tal que $q_i^{s_j} + U(j) \geq C_{s_j}$ **then**
14: $p_i = \frac{b_k \cdot \sum_{j=1}^{m}\sqrt{q_i^{s_j}}}{\sum_{j=1}^{m}\sqrt{q_k^{s_j}}}$
15: **break**
16: **end if**
17: **end for**
18: **end for**

O Algoritmo 2 apresenta a heurística gulosa para determinar os vencedores do leilão local de alocação de serviços. Se um *cliente* vence o leilão global, ele poderá participar do leilão de alocação local de serviços. Os lances serão reordenados usando a métrica dada como entrada no algoritmo. onde pp é o poder de processamento do dispositivo, como forma de avaliar se o dispositivo é capaz de suster a carga de processamento necessária para retransmissão de dados móveis, nn é o número de dispositivos vizinhos, sc é a capacidade de armazenamento (sc) em memória RAM, a fim de avaliar quantos dispositivos vizinhos poderão de fato ser contemplados, dt distância até a torre de celular mais próxima, como forma de avaliar a força do sinal.

De forma similar à demonstração de [Obadia et al. 2016], se usarmos o algoritmo quicksort temos uma complexidade de, na pior das hipóteses, $O(n^2)$ com n sendo o número total de *clientes*. O algoritmo então realiza operações $n \times r$ para verificar se o lance pode ser aceito em ambos os algoritmos e, apenas no caso do leilão global, para

Algoritmo 2 Heurística Gulosa para Leilão Local de Alocação de Serviços

Require: reordene os lances seguindo, $\frac{b_{1(pp)} + b_{1(bl)} + b_{1(nn)} + b_{1(sc)} + \sum_{j=1}^{m} q_1^{s_j}}{b_1 + b_{1(da)}} \geq \frac{b_{2(pp)} + b_{2(bl)} + b_{2(nn)} + b_{2(sc)} + \sum_{j=1}^{m} q_2^{s_j}}{b_2 + b_{2(da)}} \geq \cdots \geq \frac{b_{n(pp)} + b_{n(bl)} + b_{n(nn)} + b_{n(sc)} + \sum_{j=1}^{m} q_n^{s_j}}{b_n + b_{n(da)}}, \forall j \in [1..m]$.
$U(j) = 0$ representa quantas unidades de serviço disponíveis j foram usadas
Ensure: W_2: conjunto de lances vencedores para o Leilão Local de Alocação de Serviços
{*Atualização das Capacidades das Antenas com os dados do Leilão Global de Alocação de Serviços: Parte 1*}
{*Determinação de Vencedores: Parte 2*}
1: **for** $i = 1$ **to** n **do**
2: **if** $\forall j \in [1..m]\ q_i^{s_j} + U(j) \leq C_{s_j}$ **then**
3: $W_2 \leftarrow i$
4: $\forall j \in [1..m]\ U(j) = U(j) + q_i^{s_j}$
5: **end if**
6: **end for**

cada lance aceito, o algoritmo realiza operações $n \times r$ para encontrar o melhor preço de determinado cliente. Portanto, a complexidade será $O(n^2 \times r)$ no pior cenário possível.

O *operador de infraestrutura* usa uma topologia de rede modelada como um grafo $G = (V, E)$ onde V é o conjunto de torres de celular, e E é o conjunto de links de alta capacidade entre torres. Em cada torre existem cinco serviços VNF e em cada link existem dois serviços de largura de banda. Em nossa implementação, criamos cinco *operadoras de rede móvel*, cada uma com o mesmo número de *clientes* $\mathcal{B}$ capazes de participar do processo de leilão.

Para que o mercado se inicie, usamos os serviços solicitados por cada *cliente*, a capacidade de cada serviço e os preços de lance. Consideramos as seguintes etapas: (i) Para definir a cadeia de serviços de um lance, primeiro escolhemos uma torre de entrada aleatória e uma torre de saída aleatória da topologia de rede (G). Então, o algoritmo de Dijkstra é usado para encontrar o caminho mais curto entre esses dois nós. Todos os nós neste caminho mais curto são responsáveis por fornecer os serviços para determinado *cliente*. Usamos um número aleatório entre $[1, m]$ para definir o número máximo de serviços requeridos por cada *cliente*. Então, escolhemos, de forma aleatória, m serviços entre todos os serviços das torres de caminho mais curto. Por fim, escolhemos os serviços de largura de banda em todas as arestas. Para simplificar o processo, não distinguimos entre serviços de uma torre e serviços de largura de banda na cadeia de serviços; (ii) Para escolher a capacidade exigida, para cada serviço solicitado, escolhemos um número aleatório entre $[1, C_{s_j}]$. Sendo C_{s_j} a capacidade máxima para o serviço j; e (iii) Finalmente, para calcular o preço de lance, escolhemos um número aleatório entre 1 e $\sum_{j \in [1...M]} q_i^{s_j}$. Esse valor explica o fato de que, se um *cliente* solicitar mais serviços, mais seu preço de lance terá uma chance de ser maior.

4. Resultados e Discussões

Como não temos acesso a uma infraestrutura real de telefonia celular, usamos, como infraestrutura da *operadora de infraestrutura*, a topologia de rede Europeia GEANT, com $V = 27$ nós como torre de celular e $E = 38$ arestas como enlaces de alta capacidade entre as torres. O número total de serviços disponíveis (m) no sistema é $m = e\,m_e + t\,m_t = 211$, onde e é o número de arestas e t o número de torres de celular. Existem cinco *operadoras de rede móvel* com $m = 211$ serviços disponíveis. Os parâmetros utilizados nas simulações (com seus valores *default* em negrito) são: número de *clientes*

$n = \{\mathbf{1.400}, 2.800, \ldots, 11.200\}$; número de serviços solicitados por cada *cliente* i é um número aleatório entre $[1, m_i]$, em que $m_i = \{\mathbf{15}, 20, \ldots, 50\}$; e capacidade de serviço $C_{s_j} = \{\mathbf{250}, 300, \ldots, 500\}$ do serviço s_j.

Cada cenário apresentado nos resultados (exceto no algoritmo exato) foi executado com 20 simulações independentes. Usamos o valor médio dos resultados com o intervalo de confiança assintótico simétrico de 95%. Implementamos as simulações das heurísticas gulosas em Python 3.6 e o algoritmo exato em C++, usando o IBM CPLEX v12.5. O código da implementação das heurísticas está disponível em `https://github.com/vanessavieira/5G-auction`. As simulações gerais foram realizadas sob um computador Core i5 Memória 16 GB, HD 2 TB, 133 MHz DDR3 e macOS High Sierra v10.13.6. Para a comparação com algoritmo ótimo utilizamos um computador Core i7-7500U, Memória 16 GB, HD 2 TB, SSD 128 GB e sistema operacional Linux.

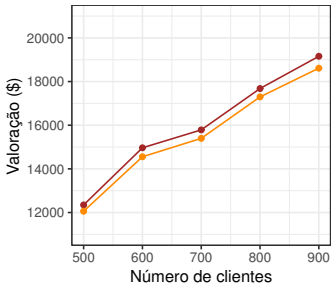
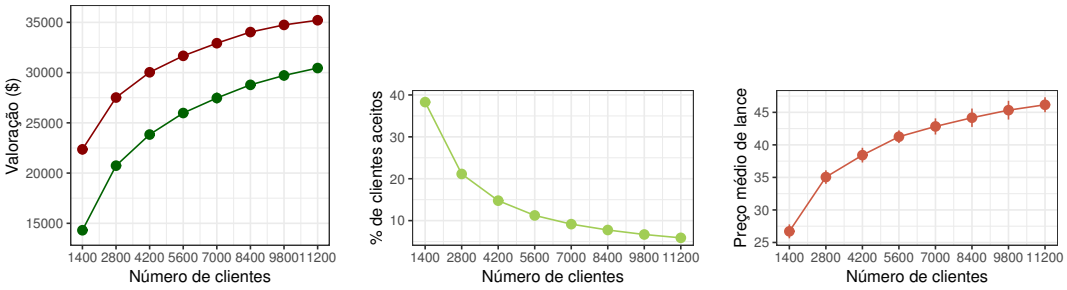


Figura 1. Em vermelho, a valoração do mercado calculada para o algoritmo exato. Em laranja, a valoração calculada para a heurística.

Para avaliar a qualidade de nossa estratégia de leilão, inicialmente, comparamos nossa solução com a ótima (figura 1). Como geramos as instâncias (conjunto de *clientes* e seus respectivos lances) artificialmente, cada instância associa-se a um nível de dificuldade diferente. Em particular, por exemplo, com 800 *clientes*, o tempo de execução da abordagem exata foi de 7h12min devido à lenta convergência do intervalo (gap). A instância com 900 *clientes*, por outro lado, demonstrou uma convergência mais rápida (59min). Os tempos de CPU em segundos para executar o algoritmo da heurística foram: 2,91; 3,38; 4,14; 4,50 e 4,94. Este resultado mostra que a nossa solução atende à restrição de tempo desse tipo de sistema pois resultados são obtidos em menos de 1 minuto, sendo essa a melhor aproximação possível em tempo polinomial, como provado anteriormente.

A primeira avaliação é a valoração de mercado e a receita do operador descrita na figura 2(a). Nesta figura, a linha vermelha indica a valoração de mercado, que é a soma de todas os valores de lance aceitos, e a verde indica a receita do operador, que é a soma de todos os preços pagos pelos *clientes* vencedores.

Esse resultado mostra a compatibilidade de incentivo entre o valor especulativo do mercado e a receita do *operador de infraestrutura*, pois o preço pago pelos *clientes* vencedores é sempre igual ou inferior aos preços de lance. Podemos notar que tanto o valor de mercado quanto a receita do operador crescem proporcionalmente, mantendo o equilíbrio do sistema e permitindo que o mercado funcione de maneira autorregulável. A figura 2(b) mostra a porcentagem de *clientes* vencedores. Quando o número de *clientes* aumenta, o número de vencedores permanece de forma semelhante, apenas com uma pequena variação. Com isso, percebemos o comportamento de decrescimento exponencial



(a) Em vermelho, a valoração de mercado. Em verde, a receita do operador.

(b) Porcentagem dos vencedores.

(c) Preço de lance por cliente.

Figura 2. Variação do número de clientes n.

no gráfico, que ocorre pois a capacidade de cada serviço e o número de serviços não varia. Na figura 2(b), o decrescimento da curva ocorre porque o número de *clientes* vencedores não está crescendo na mesma proporção que o número de *clientes* participantes. Na figura 2(c) os *clientes* começam a aumentar seus investimentos, ou seja, seus preços de lance por conta da competitividade do mercado.

De forma adicional, avaliamos a valoração de mercado e o preço por lance, variando o número máximo de serviços que podem ser solicitados pelos *clientes* para identificar o impacto do número de serviços usados na aplicação. Nessa avaliação, cujo os resultados foram comprimidos aqui, observamos que quando a quantidade de serviço aumenta de 15 até 50 (variando de 5 em 5), o preço médio de compra também aumenta linearmente de 26,71 até 115,7). Esse comportamento ocorre porque para requisitar mais serviços, o *cliente* deve aumentar seus investimentos. Outra avaliação considerou a capacidade máxima de um serviço para identificar o comportamento do sistema quanto as mudanças de capacidade. Observamos que, quando a capacidade aumenta de 250 até 500 (variando de 50 em 50), o preço médio de lance diminui de 26,71 até 14,5. Isso ocorre porque mais *clientes* podem participar do leilão com menos investimento.

Por fim, a figura 3 apresenta o investimento do operador para encorajar os *clientes* a servirem como nós de retransmissão. Especificamente, o investimento do leilão local é o mesmo independente do número de perdedores. Este resultado é significativo porque, nosso modelo considera, além do preço de lance, o nível de energia, localização e poder de processamento dos dispositivos. Assim, nosso sistema melhora o sistema móvel, uma vez que garantimos o equilíbrio e o uso justo dos recursos e serviços.

5. Conclusão

Este trabalho apresenta um sistema que combina conceitos de serviços virtuais com redes de telefonia móvel para criar uma estratégia de leilões para prover serviços em uma rede de telefonia móvel virtualizada em cenários de alta demanda de pico. Os resultados obtidos mostraram que o modelo proposto preserva o equilíbrio do sistema ao permitir que o mercado funcione de forma auto-reguladora em função da implementação dos conceitos da Teoria dos Leilões de eficiência econômica e compatibilidade de incentivos. Os resultados também demonstraram que o nosso sistema é justo para todos os *clientes* e um bom modelo de negócios para as *operadoras móveis*, ao mesmo tempo em que torna as

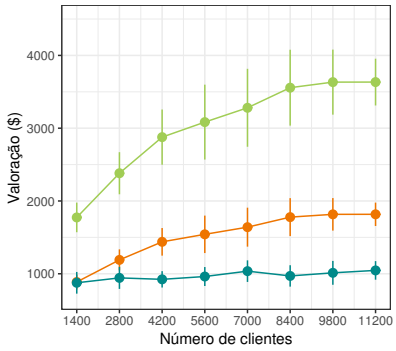


Figura 3. Em azul, o investimento para o segundo leilão. Em laranja, o investimento para o primeiro leilão. Em verde, a soma dos dois.

capacidades que até então não eram utilizadas, em capacidades de serviços virtualizados lucrativas em cenários de alta demanda de pico.

Como trabalho futuro, imaginamos simular a aplicação desse estudo em uma rede celular 5G para identificar como implantar adequadamente as funções de rede virtualizadas, além de estudar a melhor maneira de implementar a comunicação D2D. Por fim, os autores agradecem o apoio do **CNPq**, **FAPEAL** e **FAPESP**.

Referências

Bukhari, J., Park, J.-H., and Yoon, W. (2018). Providing multicast services over SDN-evolved LTE network: Architecture, procedures and performance analysis. *Computer Communications*, 127:131 – 145.

Gonen, R. and Lehmann, D. (2000). Optimal solutions for multi-unit combinatorial auctions: Branch and bound heuristics. In *2nd ACM Conference on Electronic Commerce*.

Guerzoni, R., Trivisonno, R., and Soldani, D. (2014). SDN-based architecture and procedures for 5G networks. In *1st International Conference on 5G for Ubiquitous Connectivity*.

Habiba, U. and Hossain, E. (2018). Auction mechanisms for virtualization in 5G cellular networks: Basics, trends, and open challenges. *IEEE Communications Surveys Tutorials*, 20(3):2264–2293.

Han, B., Gopalakrishnan, V., Ji, L., and Lee, S. (2015). Network function virtualization: Challenges and opportunities for innovations. *IEEE Communications Magazine*, 53(2):90–97.

Obadia, M., Bouet, M., Conan, V., Iannone, L., and Rougier, J.-L. (2016). Elastic network service provisioning with VNF auctioning. In *28th International Teletraffic Congress*.

Umrao, S., Roy, A., Saxena, N., Singh, S., and Jung, J. (2018). Mobile network operator and mobile user cooperation for customized D2D data services. *Journal of Network System Management*, 26(4):878–903.

Zhang, X., Huang, Z., Wu, C., Li, Z., and Lau, F. C. M. (2017). Online stochastic buy-sell mechanism for VNF chains in the NFV market. *IEEE Journal on Selected Areas in Communications*, 35(2):392–406.

II Workshop de Trabalhos de Iniciação Científica e Graduação
SBRC 2019
Sessão Técnica 3

Effectiveness of Implementing Load Balancing via SDN*

Leonardo C. F. P. Aguilar[1], **Daniel Macêdo Batista**[1]

[1]Department of Computer Science – IME
University of Sao Paulo (USP) – Sao Paulo, SP, Brazil

leonardo.aguilar@usp.br, batista@ime.usp.br

Abstract. *Software-Defined Networking (SDN) is an architecture that allows the creation, management and customization of the network through programmable switches and centralized controllers via a well-defined protocol. Despite the wide dissemination of general advantages in using SDN, it is always important to evaluate the real advantages considering specific network applications. In line with this, the purpose of this work is to analyze the effectiveness of using SDN for load balancing by developing a balancer, made available as free software, that can execute three different algorithms, giving to the administrator the possibility to choose, at run time, which will be used as well as their configurations, and the possibility to implement new algorithms.*

1. Introduction

According to [Moharana et al. 2013], load balancing in computer networks is an important technique aiming the optimization of existing resources for better distribution of data packets, avoiding equipment overload and decreasing of applications response time. The technique is based on choosing the most appropriate equipment to respond a particular request and it can be done in several ways. The most common is the deployment of a specific "black-box" hardware that aims to forward the data packets to the equipment in best current condition [Gandhi et al. 2014]. The definition of the best equipment is made by a distribution algorithm that can consider several static and dynamic attributes. However, an implementation based on this approach can cause several types of limitations, such as lack of flexibility in the choice of the algorithm and the impossibility to increase the number of equipment controlled in the pool, since these two aspects are locked by manufacturer and model. Besides, the cost of this solution is quite high, reaching US$ 2,914.99 for instance[1].

On the other hand, conforming to [Esteve Rothenberg et al. 2010], with the dissemination of the Software-Defined Networking (SDN) architecture in several environments, the data traffic within the network does not need to depend only on the hardware. This ensures greater customization and adaptation that can vary according to several network conditions (for example available servers, packets meta-data, etc.) in contrast to the previous rigid model. The use of SDN takes place in several aspects, ranging from the academic scope, to the development and testing of new network protocols, as well as in the commercial scope, to reduce costs and assure greater control over the network. Based

*This paper summarizes the final results of the capstone project developed by Leonardo C. F. P. Aguilar available at https://linux.ime.usp.br/~lcfpadilha/mac0499/. Accessed March 21, 2019.

[1]The Barracuda Load Balancer ADC 240 was chosen as the price parameter. Available at http://www.zones.com/site/product/index.html?id=100713342. Accessed March 21, 2019.

on this, it is possible to assume that the application of the SDN architecture to implement load balancing may be a more flexible alternative, and with a lower implementation cost, when compared to the use of specific "black-box" hardware.

The objective of this paper is to evaluate the effectiveness of implementing load balancing via SDN. Implementation specifics of a prototype and the results of experiments in an emulated network are presented to attest the advantages of the approach. It was possible to observe that the response time of a server farm using an SDN based balancer is within the user acceptance range (less than 0.1 second). To guarantee the reproducibility of the research and to encourage the spread of the approach, all the code developed on the scope of this project is available as free software at `https://github.com/lcfpadilha/pox` under Apache 2.0 license.

The rest of this paper is organized as follows: Section 2 presents some basic concepts about load balancers and SDN. Section 3 describes the main contribution: an implementation of a load balancer based on SDN. Section 4 presents the experiment planning. The results obtained from the experiments are presented on Section 5 and conclusions are presented on Section 6.

2. Basic Concepts

2.1. Load Balancers

According to [Moharana et al. 2013], load balancing is a widely used mechanism to better leverage all available resources, optimizing the use of them and the execution time of the tasks. Its application is either to distribute the workload between specific resources of a computer, such as hard disks, or between network servers in order to ensure that the best machine, i.e. the one with the greatest availability, responds to a particular request.

Several methods can be used to ensure the correct balance, and it is possible to use different elements of computer networks, such as DNS servers, to achieve this objective. One very common method is the use of a physical balancing device (the load balancer) connected to the available servers. This load balancer is responsible for listening to customers and, when a request arrives, it has the responsibility to redirect the request to one of the servers in the "farm". Because the load balancer is a single point that connects customers with the application, clients do not have direct access to the servers, which improves the security level of the service being accessed.

In addition, to ensure a better quality of load balancing between servers, it is common for load balancers to use algorithms that go beyond a simple random choice to select the next server that will respond the next request, being able to apply methods that can take into account various network parameters. There is no algorithm that can be considered preferable, since each one has its peculiarity, being able to act better in different scenarios. For this reason, some manufacturers use a combination of more than one algorithm in their devices, increasing the scenarios in which the distribution is most efficient. However, this has a great impact on cost of the devices, and can cause them to reach more than US$ 10,000.00 [KEMP Technologies nd].

2.2. Software-Defined Networking

Software-Defined Networking (SDN) is an architecture based on the decoupling of the data plane (the layer that, in fact, forwards the data) from the control plane (the layer

responsible for take decisions about the routing of the data) in such a way that it is possible to define the routing decisions using standardized and customized software not-specific to a manufacturer.

For years, the architecture of the networks was defined only by physical interconnection elements, like routers and switches, where the data plane and the control plane were indivisible. The routing and the forward of packets in these elements were managed by a layer of control created by the manufacturer, without the possibility of change by the network administrators. This control plane was built into the element itself. Thus, a network with n interconnection elements could have n different control plane rules, one for each device, not necessarily compatible with each other. With SDN, the control plane rules can stay on a dedicated server with high throughput and this server is responsible for adding rules for packets on each network device. Several rules are possible, such as allowing the flow of a particular IP address, blocking a particular port, etc.

The SDN architecture divides the networks into three distinct layers, superimposed conceptually one above the other, which have specific tasks and communicate only with their adjacent layer, as can be seen in Figure 1.

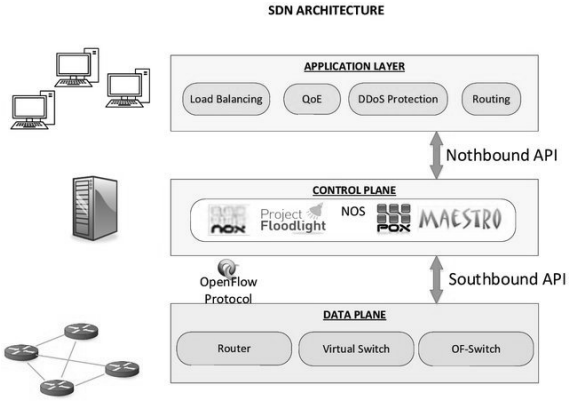


Figure 1. SDN Architecture.[Fernández et al. 2018, p.5]

The first layer, which is at the topmost level, is the **application** layer. The applications are software able to communicate with the layer below in order to define the behavior of network flows. The load balancer to be defined throughout this paper will act on this layer. The middle layer is called the control plane, commonly referred to as the **SDN controller**. It is characterized by communicating with both the level below and the level above, giving the upper layer an abstraction of the routing devices and allowing their flows to be modified by it. Several SDN controllers are available as free software. Figure 1 presents the names of some of these controllers. The layer below is the **data** plane, or layer of the network devices, whose function is to receive the data packets, perform some action with them based on the instructions passed by the controller, and update their internal counters which serve as general statistics. Communication between the application layer and the controller (or Northbound API) usually provides abstract views of the network and allows the direct expression of behavior and network requirements [Open Network Foundation 2013]. Communication between the layers of lower level with the controller (also known as Southbound API) is made through a protocol like the **OpenFlow**, that allows the control of the flows and devices' flow table.

3. SDN for Load Balancing

As explained on previous sections, the use of a physical "black-box" device to perform load balancing has a very high implementation cost and does not allow flexibility, since the administrator can not have full control of the balancing parameters. To solve this problem, this work proposes a SDN application to act as a load balancer, reducing the cost and allowing the administrator greater flexibility when compared to the fixed model determined by the manufacturers of balancers.

The proposed load balancer performs 3 different algorithms for load balancing (random, round-robin and least-bandwidth). It allows the administrator to select not only the algorithm but also the load ratio that each server will receive. The interface with the load balancer is made via CLI at run time.

In the proposed approach, a switch OpenFlow receives packets to be directed to one of the farm servers. The decision about what server will be selected is based on rules delivered by the controller, configured by the administrator. To implement the load balancer, the POX controller version 0.2.0, a controller written in Python, which allows a fast and easy development, widely used for prototyping projects in SDN, was used.

The source code of the POX controller comes with a sample module called `ip_loadbalancer` that implements a simple load balancer. This module has two classes: the `MemoryEntry`, an abstraction of the flows that are in the switches, and also the `ipbl` class, which contains the balancer logic itself. This class stores the servers' IP addresses, the controller's IP address, the flows that are active and the methods for selecting servers and for handling packets arriving at the controller.

The proposed module was implemented on the top of the sample POX module. It currently receives the IP address on which the balancer will be used and the server address list. Code 1 illustrates the execution of the load balancer. In the illustration, the load balancer address was set to `10.0.1.1` and the available servers were defined by the addresses `10.0.0.1`, `10.0.0.2` and `10.0.0.3`.

```
./pox.py misc.ip_loadbalancer ip=10.0.1.1 \\
servers=10.0.0.1,10.0.0.2,10.0.0.3
```

Code 1. Example of running the ip_loadbalancer component.

The controller sends ARP messages to the servers periodically to find out if there has been a crash on a server. This act like a heartbeat checking because if a machine doesn't respond to the message for a long time, it probably crashed and is removed from the list.

The balancing of the proposed module is done considering TCP segments, that is, only packets of this protocol will be balanced between the servers. Other protocols can be considered as long as they are supported by OpenFlow. When a packet carrying a TCP segment is received at the balancer, the controller will be responsible for sending the IP address of one of the active servers to the balancer, and when doing so, a copy of the flow is created in the controller by a period of time. Figure 2 exemplifies how balancing is done using the proposed module considering the execution illustrated on Code 1.

The original `ip_loadbalancer` module already implements the random algorithm and we extended it by including the round-robin and the least-bandwidth algo-

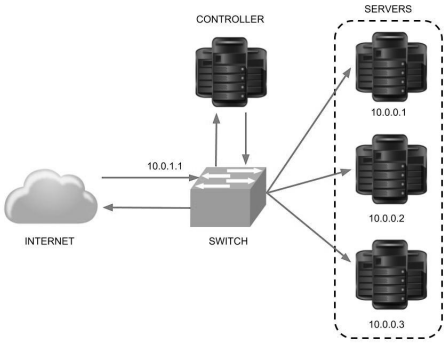


Figure 2. Exemplification of the ip_loadbalancer component.

rithms, both supporting the weights per server, and allowing the administrator to change the algorithms at run time. The round-robin algorithm works by associating the servers with a queue and when there is a request, the one that is at the beginning of the queue is used, removing it and re-positioning it to the end. The least-bandwidth algorithm selects the server with the lowest network traffic consumption to respond the next request.

4. Experiments

To evaluate the effectiveness of the proposed load balancer, it will be subjected to a load test and its response time will be measured for each one of the three algorithms and for a varying number of servers connected to the balancer.

The software used for the tests was the Apache Benchmark (`ab`) version 2.3, a program provided by the Apache Software Foundation that allows a comparative evaluation of HTTP servers, evaluating how many requests per second the servers can serve optimally. It allows the execution of requests concurrently, similar to a real scenario where more than one user can request a service from the server. In this way, this tool was used to simulate a scenario where 10 users try to access some page of a web application. The simulation also made 1000 requisitions at the same URL to thus analyze whether the balancer distributes the load efficiently and maintains an acceptable average response rate.

Thus, the test was based on executing the command displayed in Code 2 50 times and calculating the arithmetic mean of the response time of each of the tests.

```
ab -n 1000 -c 10 http://10.0.1.1
```

Code 2. Running the Apache Benchmark for testing, assuming the IP address of the balancer is 10.0.1.1.

The Mininet emulator was used to emulate a network with the default Mininet topology: a switch, which is the balancer, connected to other hosts, which can be both servers and clients. The type of the emulated network is Local Area Network (LAN) and it had $n+1$ hosts, where n represents the number of servers that one wants to test, leaving 1 host to serve only as client of the application, where the tests will be started.

The host number 1, whose IP address defined by Mininet is always 10.0.0.1, is the client, while the others, whose IP addresses range from 10.0.0.2 to 10.0.0.n+1, were the

servers of the application. Figure 3 exemplifies how the network topology was for a test with 4 servers and how each of the functions was separated.

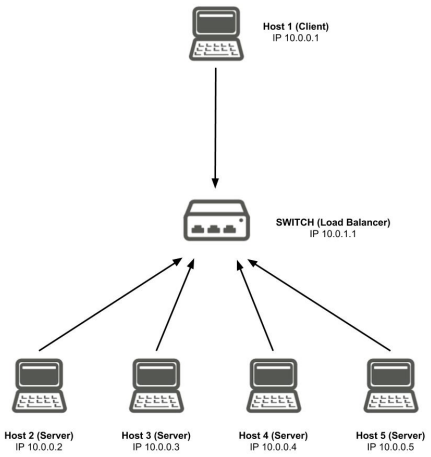


Figure 3. Exemplification of the emulated network for testing with 4 servers.

The tests were divided into two parts, using, in both, 9 different pages with sizes of 100KB, 250KB, 500KB, 1MB, 2.5MB, 5MB, 10MB, 25MB and 50MB. In the first one, the performance of the balancer, through the response time, was evaluated for 2, 3, 4, 5, 6, 7 and 8 servers, where each one delivered, in each request, a random page with the same probability to be chosen, similar to what happens on a real server, where the page size is not fixed for each request. The second part aims to evaluate how the balancer behaves when servers deliver a single page of defined size. For this test, a server farm with 8 servers was used and the tests were executed with each one of the pages and the response time for each of these scenarios were measured.

The tests were performed on an Ubuntu 14.04.4 virtual machine with 4096MB of reserved RAM running on a Macbook Pro with Intel Core i5 2.3 GHz processor, 8 GB RAM and MacOS High Sierra v10.13.6 operating system. Python 2.7.6, Mininet 2.2.2 and POX 0.2.0 (carp) were also used.

5. Results

The tests with different number of servers, whose results can be seen in Figure 4, showed that the controller behaves as expected, that is, when we add more servers, the response time tends to fall. In addition, the average response time for the tests was quite acceptable. The maximum measured was 28 milliseconds which, according to [Nielsen 1993], is within the limit where the user feels that the system is reacting instantaneously. Another consideration that can be made regarding the results is that the response time is halved when we compare a server-farm with two servers and one with three servers, but the result remains almost constant with the other number of servers.

Figure 5 presents the results of the second part of the tests. It can be seen that the balancer also has an expected behavior in the sense that the response time increases while increasing the size of the resource offered by the servers. Further analysis shows that the difference between response times for relatively small pages (between 100KB and 5MB)

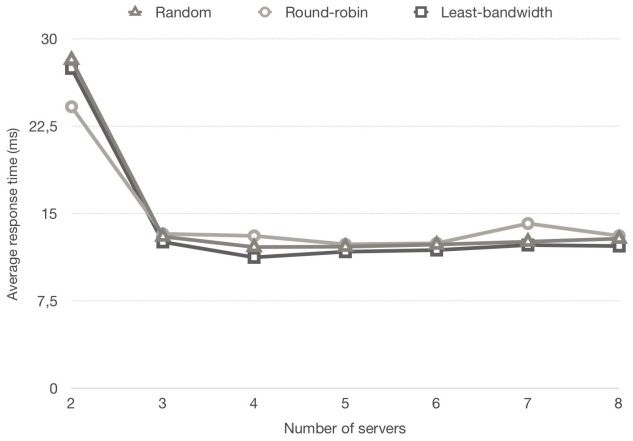


Figure 4. Average response time in relation to the number of servers. Scatter data has been suppressed since it does not exceed 2 seconds.

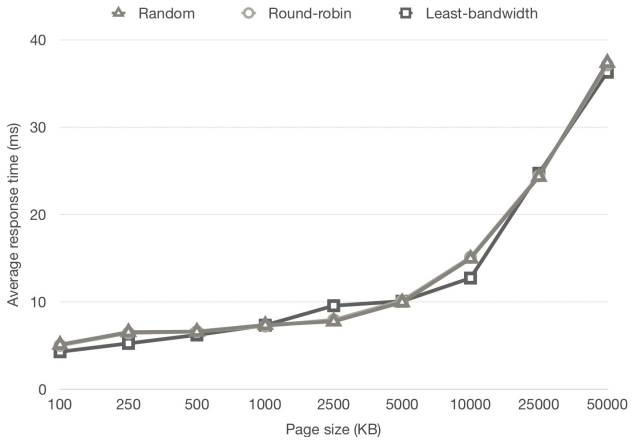


Figure 5. Average response time in relation to the pages size (Test performed with a server-farm of 8 servers). Scatter data has been suppressed since it does not exceed 1 second.

is quite small, reaching 5 milliseconds. For larger pages (between 10MB and 50MB) the difference is much greater, reaching 23 milliseconds. However, average response time remains acceptable for both small and large pages, with the maximum being 38 milliseconds, which is still within what is expected of an instant response.

In a similar way to the first test, we noticed that the differences between the algorithms are very small, especially when we compare round-robin with the random (mean difference less than 1 ms). Although the small difference, it is perceived that the least-bandwidth method has a generally low response time when compared to other algorithms.

6. Conclusions

Despite the advantages of using SDN, specific deployments of SDN applications are important to evaluate the real advantages of the architecture. This paper presented the results obtained in a capstone project which evaluated the effectiveness of using SDN for load balancing by developing a balancer made available as free software.

Load tests showed that the response time of a server farm using the proposed

SDN-based balancer is within the user acceptance range (less than 0.1 second), exposing how efficient a load balancer can be using SDN. In addition, the balancer elaborated in this work has the characteristic of being more flexible when compared to a physical device. Lastly, one of the main advantages when using SDN balancing lies in the price of specific and optimized hardware. While the cost of implementing a physical load balancer is around thousands of dollars to achieve quality balancing and with different balancing algorithms, the estimated cost for implementing a balancer like the one elaborated in this work is limited by the price of the *switch* used to forward the packages to the servers. An example of a good-quality OpenFlow switch is the HP Aruba 2920 24G PoE+ Switch, whose price is no more than U$1200.00[2], has a throughput close to 128Gbps and allows the connection of more than 20 servers.

Acknowledgments

This research is part of the INCT of the Future Internet for Smart Cities funded by CNPq proc. 465446/2014-0, Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Finance Code 001, FAPESP proc. 14/50937-1, and FAPESP proc. 15/24485-9.

References

Esteve Rothenberg, C., Nascimento, M., Salvador, M., and Magalhaes, M. (2010). OpenFlow e Redes Definidas por Software: um Novo Paradigma de Controle e Inovação em Redes de Pacotes. *Cad. CPqD Tecnologia*, 7:65–76.

Fernández, J., García Villalba, L., and Kim, T.-H. (2018). Software Defined Networks in Wireless Sensor Architectures. *Entropy*, 20:225.

Gandhi, R., Liu, H. H., Hu, Y. C., Lu, G., Padhye, J., Yuan, L., and Zhang, M. (2014). Duet: Cloud Scale Load Balancing with Hardware and Software. In *Proceedings of the 2014 ACM conference on SIGCOMM*, pages 27–38.

KEMP Technologies (n.d.). Comparison of the KEMP LoadMaster with F5 Big-IP LTM and Citrix Netscaler MPX Hardware Load Balancers and ADCs. https://kemptechnologies.com/compare-kemp-f5-big-ip-citrix-netscaler-hardware-load-balancers/. Last access at March 21, 2019.

Moharana, S. S., Ramesh, R. D., and Powar, D. (2013). Analysis of Load Balancers in Cloud Computing. *International Journal of Computer Science and Engineering (IJCSE)*, 2:101–108.

Nielsen, J. (1993). Response Times: The 3 Important Limits. https://www.nngroup.com/articles/response-times-3-important-limits/. Last access at March 21, 2019.

Open Network Foundation (2013). OpenFlow Switch Specification Version 1.4.0. https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.4.0.pdf. Last access at March 21, 2019.

[2]HP Aruba 2920 24G PoE+ Switch. Available at https://www.hpe.com/br/pt/product-catalog/networking/networking-switches/pip.aruba-2920-switch-series.5354494.html. Accessed March 21, 2019.

Realização

Organização

Apoio Institucional

Patrocínio

Diamante

Ouro

Prata

Bronze