Utilizando NFV para Implementar a Difusão Confiável e Ordenada de Mensagens na Rede

Rogério C. Turchetti[1], Giovanni Venâncio[2], Elias P. Duarte Jr.[2]

[1]CTISM - Universidade Federal de Santa Maria (UFSM)
Avenida Roraima, 1000 – 97105-900 – Santa Maria – RS – Brasil

[2]Departamento de Informática – Universidade Federal do Paraná (UFPR)
Caixa Postal 19.081 – 81.531-980 – Curitiba – PR – Brasil

turchetti@redes.ufsm.br, gvsouza@inf.ufpr.br, elias@inf.ufpr.br

Abstract. *This work presents a virtual backbone based on NFV (Network Function Virtualization) that provides broadcast services implemented within the network. Broadcast services are often required to build fault-tolerant distributed applications. Our solution provides a reliable broadcast primitive and three ordered broadcast primitives: FIFO (First-In First-Out), Causal and Atomic. Reliable broadcast ensures the delivery of messages transmitted over a SDN (Software-Defined Network) by all the correct processes. Ordered broadcast strategies ensure that all messages are delivered in the same order (atomic), according to the order in which they were sent by the origin (FIFO) or following the causal order. The virtual backbone guarantees the order of the messages through a sequencer, also implemented as a VNF, called VNF-Sequencer. The strategy proposed has been implemented and experimental results are presented.*

Resumo. *Este trabalho apresenta um backbone virtual baseado em NFV (Network Function Virtualization) que oferece serviços de difusão de mensagens implementados na própria rede. Os serviços de difusão são frequentemente utilizados para a construção de aplicações distribuídas tolerantes a falhas. Esta solução oferece uma primitiva de difusão confiável e três primitivas de difusão ordenada de mensagens: FIFO (First-In First-Out), Causal e Atômica. A difusão confiável garante a entrega das mensagens transmitidas em uma rede SDN (Software-Defined Network) por todos os processos corretos. As estratégias de difusão ordenada garantem que as mensagens são entregues todas na mesma ordem (atômica), de acordo com a ordem em que foram enviadas pela origem (FIFO) ou obedecendo a ordem causal. O backbone virtual garante a ordem das mensagens através de um sequenciador, também implementado como uma VNF, denominada de VNF-Sequencer. A estratégia proposta foi implementada e resultados experimentais são apresentados.*

1. Introdução

Uma abstração importante para o desenvolvimento de aplicações distribuídas e tolerante a falhas é a difusão confiável (*reliable broadcast*). Informalmente, a difusão confiável garante que as mensagens enviadas para um conjunto de processos seja entregue por todos os processos corretos [Défago et al. 2004]. Além disso, um processo pode exigir que todas as mensagens sejam entregues em uma determinada ordem por todos os demais

processos. Neste caso, existem diversos tipos de ordens que podem ser definidos. Se todos os processos corretos devem entregar todas as mensagens exatamente na mesma ordem, a difusão é denominada atômica (*atomic broadcast*). Se a ordem for determinada a partir do processo emissor, então a difusão é denominada FIFO (*First-In First-Out*). Quando as mensagens devem ser entregues de acordo com a precedência causal [Lamport 1978], então a difusão é denominada causal (*causal broadcast*). Na prática, a implementação da difusão confiável e ordenada não é uma tarefa trivial. Se este serviço é oferecido na própria aplicação, a complexidade para o seu desenvolvimento é significativamente maior [Li et al. 2016]. Outra possível alternativa é a utilização de *middlewares* específicos que executam nas máquinas dos próprios usuários.

Em [Ekwall and Schiper 2007] os autores argumentam que o desempenho dos algoritmos de difusão confiável são influenciados pelo número de passos na comunicação e pelo número de mensagens necessárias para alcançar uma decisão. No entanto, além do usuário levar em consideração aspectos de desempenho, também é necessário escolher um algoritmo que se adapte melhor às características específicas da rede onde é executado. A nova alternativa explorada neste trabalho retira esta preocupação do usuário, pois move a difusão confiável da máquina do usuário (onde é normalmente executada como uma aplicação ou *middleware*) para a própria rede.

Neste trabalho é proposta a implementação de um *backbone* virtual que oferece difusão confiável utilizando Virtualização de Funções de Rede (*Network Function Virtualization* ou NFV) em uma Rede Definida por Software (*Software-Defined Network* ou SDN). Este *backbone* virtual aproveita as vantagens das tecnologias de virtualização para habilitar a rede a oferecer diversos serviços de difusão confiável e ordenada. De fato, a ordem total das mensagens é garantida através do uso de um sequenciador que também está presente dentro da própria rede, enquanto que as primitivas de difusão são executadas através de uma API (*Application Programming Interface*) acessível pela aplicação distribuída. O sequenciador é implementado como uma Função Virtualizada de Rede (*Virtualized Network Function* ou VNF), denominada de *VNF-Sequencer* e a API é denominada de *RBCast*, onde são oferecidos diversos tipos de difusão: difusão confiável, difusão atômica, difusão atômica FIFO e difusão atômica causal. Portanto, a *VNF-Sequencer* é responsável por gerenciar toda a troca de mensagens de forma a garantir as propriedades de comunicação definidas na aplicação distribuída.

O *backbone* virtual foi implementado e resultados experimentais são descritos, incluindo a latência e vazão da difusão em diversos cenários. Por fim, os resultados demonstram que a estratégia proposta consegue garantir a difusão confiável e ordenada de mensagens de maneira simples e eficiente.

O restante deste trabalho está organizado da seguinte forma. A Seção 2 descreve trabalhos relacionados, apresentando também conceitos básicos de difusão de mensagens e virtualização de funções de rede. A Seção 3 apresenta as estratégias de ordenação e a *VNF-Sequencer*. Os resultados experimentais são apresentados na Seção 4 e as conclusões são apresentadas na Seção 5.

2. Fundamentação e Trabalhos Relacionados

Nesta seção são apresentados os principais conceitos utilizados, incluindo estratégias relacionadas à difusão de mensagens e tecnologia NFV e trabalhos relacionados.

2.1. Difusão Confiável e Ordenada de Mensagens

Em sistemas distribuídos, um caso especial de comunicação ocorre quando um processo deseja transmitir uma mensagem que precisa ser entregue, garantidamente, a todos os demais processos corretos do sistema [Défago et al. 2004]. Essa comunicação é denominada difusão confiável (*reliable broadcast*) e é fundamental para a construção de aplicações distribuídas tolerantes a falhas [Pedone and Schiper 2003].

A difusão confiável pode ser definida em termos de duas primitivas: *broadcast(m)* e *deliver(m)*, onde m é uma mensagem. A primitiva *broadcast(m)* é executada pelo transmissor para disseminar m para todos os processos. A primitiva *deliver(m)* causa a entrega da mensagem m por cada processo correto. Apesar da difusão confiável garantir a entrega das mensagens, ela não impõe nenhuma restrição quanto a ordem em que as mensagens são entregues. Há vários tipos de ordem que podem ser utilizadas para a sequência de mensagens entregues pelos processos do sistema.

A difusão atômica requer que todos os processos corretos entreguem todas as mensagens na mesma ordem, denominada de ordem total. Em geral, o algoritmo garante a ordem total para uma sequência de mensagens que é globalmente acordada. Em outras palavras, a ordem total requerida pela difusão atômica implica que processos corretos entreguem a mesma sequência de mensagens. Existe ainda a possibilidade da aplicação exigir outras restrições com relação à ordem das mensagens. Duas das possíveis propriedades de ordenação são descritas a seguir, denominadas de ordem FIFO (*First-In First-Out*) e de ordem causal.

Em algumas aplicações, o contexto de uma mensagem só faz sentido tendo em vista as outras mensagens previamente transmitidas pela mesma origem. A difusão que obedece esta ordem é denominada FIFO. Neste trabalho a propriedade FIFO foi implementada em conjunto com a difusão atômica, de forma a garantir que a ordem total obedeça também à ordem FIFO.

Por outro lado, uma mensagem m pode também depender das mensagens que o transmissor de m entrega, antes de transmitir m. Neste caso, a ordem FIFO não é suficiente. No cenário descrito é necessário um tipo de ordenação que leva em consideração eventos com precedência causal. A noção de causalidade no contexto de sistemas distribuídos foi formalizada por Lamport [Lamport 1978]. Um exemplo em que a ordem causal é necessária ocorre em redes sociais, quando uma mensagem causa outras, que devem ser recebidas posteriormente. Se as aplicações necessitarem de uma ordenação total respeitando eventos com precedência causal, faz-se necessário o uso da difusão atômica causal. Em outras palavras, a difusão atômica causal é uma difusão confiável que satisfaz as propriedades de ordem causal e de ordem total.

2.2. *Network Function Virtualization*

A Virtualização de Funções de Rede (*Network Function Virtualization*–NFV) surge como uma tecnologia para facilitar o gerenciamento de serviços de rede, uma vez que estes serviços são normalmente implementados como *middleboxes* e, em geral, necessitam de hardware específico [Cotroneo et al. 2014]. A tecnologia NFV utiliza técnicas de virtualização para disponibilizar estes serviços através de dispositivos virtuais que executam em hardware genérico (e.g., arquitetura x86). Dessa forma, a inclusão de novos servi-

ços, alteração e/ou atualização de serviços já existentes, não necessitam da instalação de equipamentos de hardware especializados.

Existem diversas vantagens para utilizar a tecnologia NFV. Além de reduzir as despesas para a aquisição e manutenção de equipamentos, há vantagens como a economia de energia e de espaço físico [Han et al. 2015]. Além disso, a flexibilidade do gerenciamento da rede também melhora, na medida em que serviços de rede são facilmente instanciados, utilizados e removidos. Por fim, Funções Virtualizadas de Rede – *Virtualized Network Functions* (VNFs) – podem ter seus recursos ajustados de maneira automática, aumentando ou diminuindo de acordo com a demanda necessária, fazendo uso eficiente dos recursos do sistema.

O *backbone* virtual proposto neste trabalho é disponibilizado através de funções virtualizadas de rede, tirando proveito do fato que a tecnologia NFV traz vantagens em oferecer serviços arbitrários através de dispositivos virtuais. Os detalhes de implementação deste *backbone* são apresentados na Seção 3.

2.3. Trabalhos Relacionados

Nesta seção são descritas plataformas e protocolos que oferecem primitivas de difusão atômica. A grande diferença entre estas estratégias e a solução proposta é que o *backbone* virtual utiliza funções virtualizadas de rede, sendo possível a implantação das primitivas de comunicação dentro da própria rede.

Em [Reed and Junqueira 2008] os autores propõe Zab, um protocolo de difusão atômica utilizado pelo serviço ZooKeeper [Hunt et al. 2010]. O protocolo Zab é utilizado para manter réplicas dos dados em cada servidor ZooKeeper. O protocolo Zab considera um sequenciador fixo, denominado de líder, eleito a partir de um algoritmo de eleição de líder. Cada processo que deseja realizar uma difusão atômica deve enviar sua mensagem ao líder. Para a entrega das mensagens aos demais processos, o líder executa um algoritmo semelhante ao *two-phase commit*, onde é feita uma requisição, uma coleta de votos e posteriormente o *commit*. Para garantir a ordem FIFO, todas as comunicações utilizam conexões TCP entre todos os pares de processos do sistema.

No trabalho apresentado em [Li et al. 2016], com o objetivo de garantir a replicação consistente em *data centers*, os autores utilizam uma solução que divide as tarefas entre a rede e a aplicação. A rede garante a ordem das mensagens, enquanto que o protocolo de replicação garante a entrega das mensagens. O protocolo utilizado é denominado de NOPaxos (*Network-Ordered Paxos*). NOPaxos é executado somente quando necessário, evitando a sincronização constante entre os processos. Em outras palavras, quando ocorrem imprevistos na comunicação (e.g., mensagens perdidas), o protocolo NOPaxos é executado. A ordem das mensagens é garantida pela implementação de um sequenciador localizado dentro da rede. Os autores mostram que quando o sequenciador é implementado diretamente nos *switches*, a replicação ocorre com baixa latência e alta vazão, fornecendo uma replicação com baixo custo.

Mais antigo, o ISIS [Birman and Joseph 1987] é um sistema distribuído clássico que fornece mecanismos de tolerância a falhas. A primitiva ABCAST do ISIS fornece a difusão atômica de mensagens. Além disso, são oferecidas outras formas de ordenação, como a ordem causal e atômica causal. Os autores consideram grupos de processos e a ordem global das mensagens é garantida mesmo entre grupos sobrepostos (i.e., um ou

mais processos em grupos diferentes). Uma vez que garantir a ordem total das mensagens pode ser uma tarefa custosa, o sistema ISIS também fornece primitivas mais fracas de ordenação em troca de um melhor desempenho. A aplicação do usuário determina qual primitiva irá utilizar.

Em [Kaashoek and Tanenbaum 1991], os autores propõe Amoeba, um sistema operacional distribuído com primitivas de difusão de mensagens. Em especial, Amoeba fornece primitivas de difusão confiável e ordenada para grupos de processos. Pode-se assumir um número variado de grupos e um processo pode pertencer a mais de um grupo. Além disso, as mensagens disseminadas por um processo são enviadas apenas aos processos pertencentes do grupo do processo emissor. O protocolo também assume falhas de comunicação e falhas nos processos. O sistema Amoeba executa o protocolo dentro do *kernel* do sistema operacional. Dessa forma, o hardware de cada processo deve ser idêntico, executar o mesmo *kernel* e utilizar a mesma aplicação. O protocolo utiliza um sequenciador centralizado, onde um dos membros do grupo recebe o papel de sequenciador.

Em geral, as plataformas e protocolos propostos executam nas máquinas dos usuários. Isso é feito através da utilização de hardware ou software especializado. Além disso, nem todas as soluções apresentadas implementam todas as primitivas importantes de difusão ordenada. A proposta deste trabalho é utilizar uma função virtualizada de rede para a implementação, na própria rede, de diversas primitivas de difusão confiável e ordenada. Dessa forma, a complexidade para construção de aplicações distribuídas é menor, uma vez que basta utilizar um serviço oferecido pela própria rede. Por fim, a estratégia baseada em NFV não requer nenhum hardware específico, *middleware* ou modificações na aplicação para executar os diversos tipos de difusão.

3. Um *Backbone* Virtual com Difusão Confiável e Ordenada de Mensagens

O *backbone* virtual proposto neste trabalho implementa uma VNF que fornece primitivas de comunicação para a transmissão de mensagens via difusão confiável e ordenada. Esta abordagem assume canais de comunicação confiáveis e implementa vários algoritmos clássicos para garantir a entrega e a ordem das mensagens. O sistema subjacente é assíncrono, dotado de detector de falhas, que deve apresentar completude forte [Chandra and Toueg 1996]. Os algoritmos são executados de forma modular, em camadas. Por exemplo, para a entrega confiável das mensagens, é necessário sempre executar o algoritmo para a difusão confiável, ao passo que, para garantir também a ordem total das mensagens, é necessário executar também, no topo da difusão confiável, o algoritmo de difusão atômica.

O *backbone* implementa o algoritmo básico de difusão confiável que utiliza um detector de falhas [Chandra and Toueg 1996]. Este algoritmo garante a entrega das mensagens da seguinte forma: o processo origem que dispara a difusão, envia a mensagem m para todos os processos. Quando um processo recebe a mensagem m pela primeira vez, ele só retransmite m para todos os processos se o processo que originou a mensagem for suspeito de ter falhado. Sempre que um processo é suspeito de ter falhado, as mensagens que originou são retransmitidas pelos demais. Cada mensagem, na difusão confiável, tem informações sobre o emissor da mensagem e o conteúdo do pacote. É importante notar que toda mensagem possui um campo, denominado $local\_seq$, composto pelo identifica-

dor do processo e o número de sequência desta mensagem. Dessa forma as mensagens podem ser identificadas unicamente e o sequenciador pode ordenar mensagens vindas concorrentemente de múltiplos processos emissores.

As próximas seções apresentam detalhes de como o *backbone* define a ordem total das mensagens transmitidas por difusão para serem entregues às aplicações, bem como a arquitetura e lógica de implementação.

3.1. Garantindo a Ordem das Mensagens

Em um sistema em que é necessário manter a ordem total das mensagens, é fundamental resolver o problema de como garantir a ordenação. Na literatura existem algumas alternativas para a implementação da ordem total dos eventos em um sistema distribuído. Uma delas é a utilização de um módulo sequenciador responsável por receber todas as mensagens e encaminhá-las aos destinatários, garantindo então a ordem total das mensagens. O sequenciador pode ser fixo, móvel, baseado em privilégios, entre outros [Défago et al. 2004]. Este trabalho utiliza um sequenciador fixo para garantir a ordem das mensagens.

Nesta abordagem, a ordem total é construída da seguinte forma: para uma mensagem m ser transmitida por difusão, em primeiro lugar o emissor de m encaminha a mensagem para o sequenciador; quando m é recebida pelo sequenciador a mensagem obtém um valor único de sequência; m é transmitida por difusão para todos os destinatários; por fim, os processos receptores entregam m de acordo com o valor de sequência atribuído pelo sequenciador.

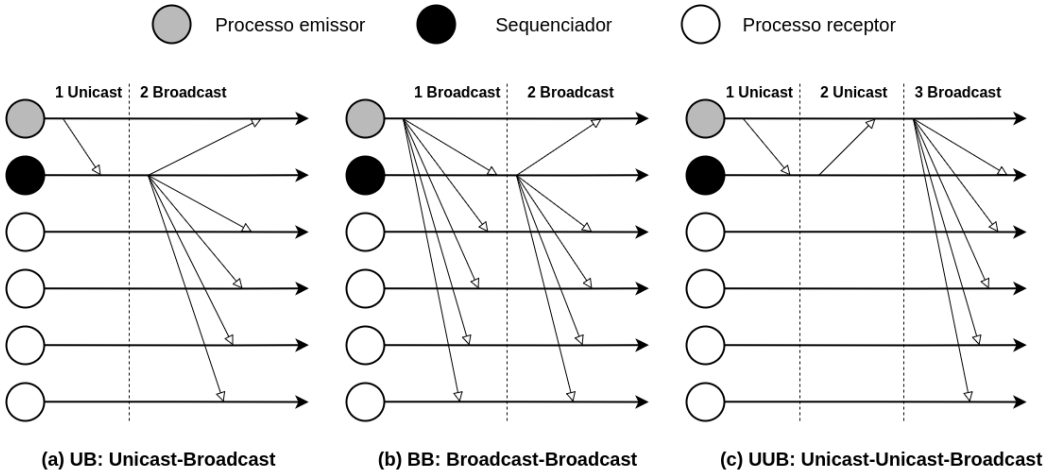

Figura 1. Três métodos para implementar o sequenciador fixo.

O sequenciador fixo pode ser implementado através de três métodos apresentados na Figura 1 [Défago et al. 2004]: UB (*Unicast-Broadcast*), BB (*Broadcast-Broadcast*) e UUB (*Unicast-Unicast-Broadcast*). A Figura 1(a) apresenta o método UB onde o processo que deseja transmitir uma mensagem por difusão (processo emissor) executa uma comunicação *unicast* (ponto-a-ponto) seguida de uma difusão executada pelo sequenciador. O sequenciador, ao executar a difusão, adiciona um número de sequência à mensagem. Ao receber a mensagem, o processo receptor entregará a mensagem para a aplicação respeitando a ordem indicada pelo sequenciador. Já no método BB apresentado na Figura

1(b), o processo emissor executa uma difusão para todos os destinatários, incluindo o sequenciador. O sequenciador adiciona um número de sequência na mensagem e executa uma difusão para todos os destinatários. Note que este método gera um número maior de mensagens do que o método UB. Por outro lado, o método BB facilita a implementação de um sequenciador tolerante a falhas. Por fim, no método UUB apresentado na Figura 1(c), o processo emissor solicita um número de sequência para o sequenciador. O próprio processo emissor inclui o número de sequência na mensagem e a transmite para todos os processos. Este último método (UUB) é menos comum que os demais e inclui um terceiro passo que o torna menos eficiente – já que os demais resolvem o problema em apenas dois passos. O sequenciador neste trabalho utiliza o método BB. A implementação do sequenciador, denominada de *VNF-Sequencer*, é descrita na Seção 3.2.

Neste trabalho, o sequenciador considera duas sequências de mensagens. A primeira respeita a ordem local de transmissão de cada processo, possibilitando por exemplo, que a ordem FIFO seja implementada. A outra é uma ordem global que possibilita garantir que a ordem de entrega será atômica. Na transmissão atômica, a ordem global é sempre a ordem em que os processos finais (receptores) consideram para a entrega das mensagens.

Algoritmo 1 Algoritmo para a construção da ordem global.

Sender:
1: **Init:**
2: $RBtype \coloneqq AtomicRB$ {Define o algoritmo}
3: local_seq $\coloneqq$ 1
4: **upon** broadcast(m) **do**
5: broadcast(m, local_seq, $RBtype$)
6: local_seq $\coloneqq$ local_seq + 1

Sequencer:
7: **Init:**
8: global_seq $\coloneqq$ 1 {Contador de mensagens global utilizado para entregar m}
9: **upon** receive (m, local_seq, $RBType$) **do**
10: broadcast(m, global_seq)
11: global_seq $\coloneqq$ global_seq + 1

Receiver:
12: **Init:**
13: nextMsg $\coloneqq$ 1
14: pendingMsg $\coloneqq \emptyset$
15: **upon** receive (m, global_seq) **do**
16: pendingMsg $\coloneqq$ pendingMsg $\cup \{m\}$
17: **while** ($\exists$ ($m' \in$ pendingMsg $\wedge$ global_seq = nextMsg)) **do**
18: deliver(m')
19: pendingMsg $\coloneqq$ pendingMsg $\setminus \{m'\}$
20: nextMsg $\coloneqq$ nextMsg + 1
21: **end while**

No Algoritmo 1 é apresentado o pseudo-código para a construção da ordem atômica entre os processos. No algoritmo existem três papéis distintos: *Sender* (processo emissor), *Sequencer* (sequenciador das mensagens) e o *Receiver* (processo receptor). O processo p_i que deseja transmitir uma mensagem m, por difusão atômica e confiável (*Sender*), inicialmente define o algoritmo a ser utilizado, por exemplo, difusão atômica,

difusão atômica e FIFO ou difusão atômica e causal. A mensagem m será transmitida por difusão carregando informações como o contador local de mensagens ($local\_seq$) e informando o tipo de algoritmo que será utilizado. O contador global é inserido pelo sequenciador na mensagem e incrementado sempre após a retransmissão de uma mensagem aos processos finais. Cada processo receptor (*Receiver*) que recebe uma mensagem m, adiciona esta mensagem ao conjunto de mensagens pendentes ($pendingMsg$). Logo após é feita a verificação se existe alguma mensagem m' que possui o contador igual ao identificador da próxima mensagem ($nextMsg$) e também se m' ainda está em $pendingMsg$. Quando estas duas condições são satisfeitas, então m' e todas as outras mensagens que satisfazem as condições indicadas são entregues para a aplicação.

No caso da aplicação requerer ordenação não apenas atômica mas também FIFO, o sequenciador implementa a ordem de entrega de acordo com o Algoritmo 2. No algoritmo da difusão FIFO é necessário verificar se m, transmitida por p_i, possui o valor do contador esperado como valor da próxima mensagem daquele mesmo processo origem. Caso contrário, haverá uma pendência na ordem FIFO que precisa ser resolvida. Neste caso, m é inserida no conjunto de pendências ($F\_pendingMsg$). Em outras palavras, se por exemplo, p_i transmitir $m_3^{p_i}$ e $m_4^{p_i}$ (onde $m_3^{p_i}$ é uma mensagem transmitida por p_i e 3 é o valor de local_seq) e o sequenciador receber primeiro m_4, ela armazena e atrasa a transmissão de m_4 até que m_3 seja recebida.

Algoritmo 2 Algoritmo para a construção da ordem atômica e FIFO.

Sender:
1: **Init:**
2: $RBtype \coloneqq AtomicFIFO$
3: local_seq $\coloneqq$ 1
4: **upon** broadcast(m) **do**
5: broadcast(m, local_seq, $RBtype$)
6: local_seq $\coloneqq$ local_seq + 1

Sequencer:
7: **Init:**
8: nextMsg $\coloneqq$ 1
9: F_pendingMsg $\coloneqq \emptyset$ {Conjunto de mensagens pendentes}
10: **upon** receive (m, local_seq, $RBType$) **do**
11: **if** ($RBtype$ = $AtomicFIFO$) **then**
12: **if** (local_seq = nextMsg) **then**
13: broadcast(m, local_seq)
14: nextMsg $\coloneqq$ nextMsg + 1
15: **while** ($\exists$ ($m' \in$ F_pendingMsg $\wedge$ local_seq = nextMsg)) **do**
16: broadcast(m', local_seq)
17: nextMsg $\coloneqq$ nextMsg + 1
18: F_pendingMsg $\coloneqq$ F_pendingMsg $\setminus \{m'\}$
19: **end while**
20: **else**
21: F_pendingMsg $\coloneqq$ F_pendingMsg $\cup \{m'\}$
22: **end if**
23: **end if**

Seguindo o exemplo, quando a respectiva mensagem esperada (m_3) é recebida pelo algoritmo FIFO, ela e as mensagens que estão pendentes (m_4) já podem ser enca-

minhadas pelo sequenciador. A ordem de encaminhamento das mensagens pendentes respeita a ordem do contador local inseridas em $F\_pendingMsg$. Então, o sequenciador adiciona o valor do contador global à mensagem e transmite $m^s_{m'}$ e $m^s_{m''}$ por difusão confiável aos destinatários, onde $m' < m''$, no exemplo m'=m_3 e m''=m_4. Por fim, os receptores entregam as mensagens de acordo com o Algoritmo 1 (*Receiver*), isto é, enquanto houver mensagens pendentes (mensagens armazenadas em $pendingMsg$) e com o valor do contador global igual ao respectivo valor esperado ($nextMsg$), então as mensagens são entregues nesta ordem para a aplicação.

Observe que neste exemplo se um outro processo p_j transmitir $m^{p_j}_1$ (primeira mensagem de p_j), o *Sequencer*, ao receber esta mensagem, a processa e retransmite imediatamente, pois $m^{p_j}_1$ é independente das mensagens transmitidas por p_i quando considerada a ordem FIFO.

Na estratégia apresentada, como toda mensagem originada de uma difusão ordenada passa pelo sequenciador, a ordem causal é garantida ao garantir a ordem FIFO.

3.2. A Arquitetura e Implementação do *Backbone* Virtual

A arquitetura do *backbone* virtual é composta por componentes que estão operando dentro da própria rede SDN. Para disponibilizar as primitivas de comunicação às aplicações, é oferecida uma API na biblioteca denominada de *RBCast*. A arquitetura pode ser visualizada na Figura 2. Observe que a *RBCast* está, estrategicamente, nas pontas da comunicação, ou seja, junto aos processos emissores e receptores. A forma como os *hosts* se comunicam é descrita a seguir.

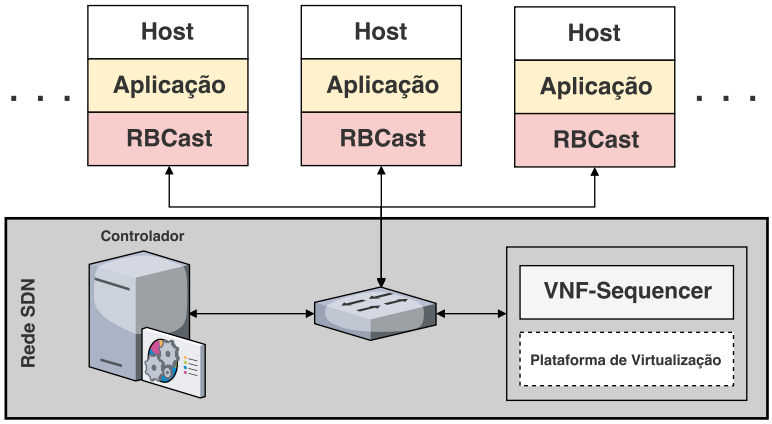

Figura 2. Arquitetura do *backbone* virtual.

Uma vez que algum *host* deseja realizar uma difusão na rede, é primeiro escolhido o algoritmo de difusão e as mensagens são encaminhadas a um *switch* da rede SDN. Do funcionamento padrão do protocolo *OpenFlow* [McKeown et al. 2008], toda mensagem que não possui uma entrada na tabela de fluxos é encaminhada ao controlador. Se o algoritmo escolhido for a difusão confiável, as mensagens não precisam ser enviadas ao sequenciador e são entregues de acordo com o algoritmo descrito em [Chandra and Toueg 1996]. Caso contrário, o controlador SDN é configurado para que as regras instaladas nos *switches* encaminhem o fluxo de pacotes para a própria *VNF-Sequencer*, sempre que a difusão for ordenada. Dessa forma, a *VNF-Sequencer* recebe

todo o fluxo de mensagens gerado a partir de uma difusão. Ao receber o fluxo de pacotes, a *VNF-Sequencer* executa o algoritmo especificado pelo processo emissor e envia as mensagens aos processos receptores de acordo com os algoritmos especificados na Seção 3.1.

Em geral, o sequenciador é um processo único no sistema, assim o *backbone* assume que o sequenciador não falha. O sequenciador é implementado como uma função virtualizada de rede denominada de *VNF-Sequencer*. A *VNF-Sequencer* explora as funcionalidades da rede SDN, isto é, como há um controlador SDN responsável por gerenciar toda a comunicação da rede, ela recebe do controlador todas as comunicações referentes às mensagens encaminhadas através da biblioteca *RBCast*. *RBCast* é uma biblioteca que possui a implementação das primitivas para a difusão confiável, bem como uma interface para a comunicação com as aplicações. Quando um processo deseja transmitir uma mensagem m por difusão confiável e atômica, a ordem total é construída através de dois passos, denominados de *Process-to-Sequencer* (primeiro passo) e *Sequencer-to-Process* (segundo passo), ambas detalhadas a seguir.

Process-to-Sequencer: o emissor de uma mensagem m define qual algoritmo será utilizado na transmissão da mensagem. Para isso ele acessa a primitiva de comunicação disponível pela interface *RBCast* e realiza uma difusão dentro da rede SDN. Este processo é similar ao primeiro passo do método BB apresentado na Figura 1(b). Do comportamento padrão do protocolo *OpenFlow*, as primeiras mensagens de cada fluxo são encaminhadas para o controlador, para a criação das respectivas regras no *switch*. Neste caso, o controlador é configurado para encaminhar todas as mensagens oriundas da *RBCast* para a *VNF-Sequencer*. Essa estratégia é utilizada para tornar transparente o local de execução da *VNF-Sequencer* (i.e., o emissor não necessita do endereço IP do sequenciador). Alternativamente, pode ser utilizada uma comunicação *unicast* que faz a conexão entre o processo emissor com o sequenciador, de maneira explicita. Por fim, antes da mensagem ser transmitida, m recebe um valor de sequência local atribuído pelo próprio emissor.

Sequencer-to-Process: quando a mensagem m é recebida pela *VNF-Sequencer*, o sequenciador seleciona o algoritmo para difusão definido no passo *Process-to-Sequencer*, executando todos os procedimentos necessários. Quando m é retransmitida pela *VNF-Sequencer* aos processos receptores, a mensagem recebe um valor global de sequência atribuído pelo próprio sequenciador. Por fim, os processos receptores entregam m de acordo com o valor de sequência atribuído pela função de rede.

A próxima seção descreve os resultados experimentais executados para a avaliação do funcionamento do *backbone* virtual em diversos cenários de execução.

4. Avaliação Experimental

Nesta seção são executados experimentos a fim de avaliar o desempenho da *VNF-Sequencer* em uma rede SDN. Além disso, é analisada a viabilidade de mover os serviços de difusão para dentro da rede, de maneira a garantir todas as propriedades de maneira eficiente.

O sistema foi executado em uma máquina física com processador Intel Core i5-7200U@2.50GHz com 4 núcleos, 8 GB de memória RAM e sistema operacional Ubuntu

16.04. Para a geração de pacotes, foi desenvolvido um *script* que realiza difusão de mensagens de maneira contínua para as demais aplicações do sistema. As aplicações, a *VNF-Sequencer* e o controlador SDN são executados em *containers* da plataforma *Docker*[1]. O controlador SDN utilizado é o Ryu[2]. São apresentados experimentos para avaliar o custo da *VNF-Sequencer* em termos de vazão e latência.

4.1. Avaliação da Latência da *VNF-Sequencer*

Na Seção 3 foi mostrado que o objetivo da *VNF-Sequencer* é garantir a entrega atômica das mensagens em uma rede SDN. A utilização de um sequenciador é, na verdade, a maneira mais eficiente de garantir a ordem total das mensagens. Por outro lado, uma vez que o sequenciador é um único processo que centraliza todas as comunicações, existe um custo computacional decorrente da necessidade de processar todos os pacotes originados das diversas difusões. Neste sentido, o primeiro experimento tem como objetivo medir o impacto na latência ao realizar uma difusão com sequenciador (difusão atômica) e sem sequenciador (difusão confiável).

No experimento cujos resultados são apresentados na Figura 3, a topologia consiste de um *switch*, um controlador SDN e um número variado de processos que participam da difusão confiável. Um dos processos participantes do experimento implementa uma aplicação cliente responsável por inicializar o envio das mensagens que são, inicialmente, repassadas ao *RBCast* local. Quando o *RBCast* dos processos receptores recebe o pacote, a mensagem é entregue para a aplicação. Neste experimento medimos a latência para a entrega das mensagens, que vai do instante de tempo em que uma mensagem foi transmitida por difusão confiável até o instante em que o último processo tenha entregado a mensagem para a aplicação. Para cada experimento são executadas 1000 difusões com pacotes de 1 KB. Os dados apresentados são valores médios de 10 amostras, utilizando um intervalo de confiança de 95%.

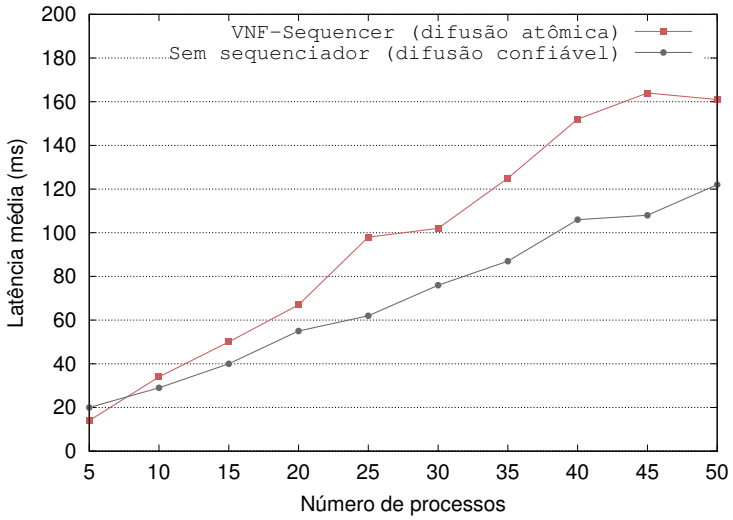

Figura 3. Comparação da latência para a entrega das mensagens.

[1]https://www.docker.com/
[2]https://osrg.github.io/ryu/

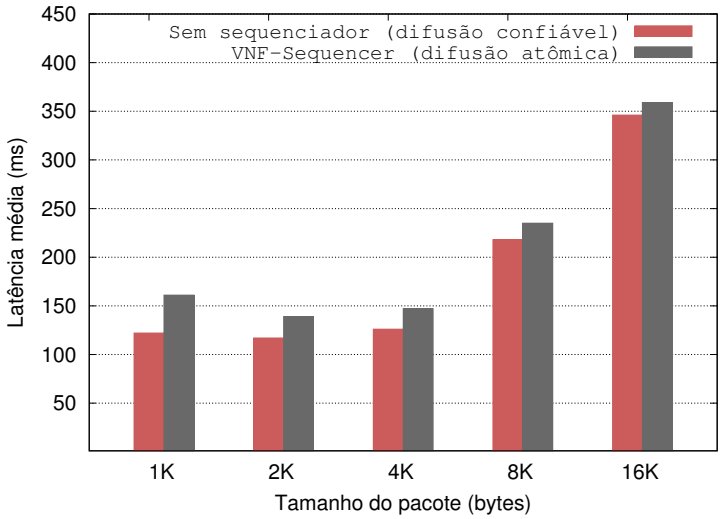

Figura 4. Comparação da latência para diferentes tamanhos de pacote.

No gráfico da Figura 3, o eixo Y apresenta o valor da latência em milissegundos para a entrega das mensagens para as aplicações, ao passo que o eixo X descreve o número de processos participantes da difusão confiável. A comunicação ocorre com sequenciador (curva *VNF-Sequencer*) e sem sequenciador (curva *Sem sequenciador*). Como previsto, é possível notar que a latência é maior com o uso da *VNF-Sequencer*. Entretanto, o objetivo é verificar quais são os valores deste custo, em termos de tempo. Neste caso, observa-se que os valores da latência ao utilizar um sequenciador é, em média, aproximadamente 32.1% maior. É possível notar também que a latência cresce para ambos os casos conforme o número de processos aumenta. Até 20 processos, o aumento na latência é aproximadamente 20%. Em outras palavras, em um sistema com até 20 processos, o custo para garantir a ordem total das mensagens em todos os processos é relativamente pequeno com o uso de um sequenciador. Para mais do que 20 processos, o custo na latência aumenta de 34% para até 56%.

Foi avaliado também como o tamanho do pacote influencia no desempenho das difusões. Neste experimento, é analisada a latência para 50 processos, onde um processo cliente envia pacotes com tamanho de 1 KB até 16 KB. A Figura 4 mostra a média obtida a partir de 10 amostras, onde cada amostra consiste de 1000 difusões.

Considerando estes resultados, a latência média aumenta em 15.7% com o uso da *VNF-Sequencer*. É importante notar que o aumento na latência diminui conforme o tamanho dos pacotes aumenta. Por exemplo, para pacotes de 1 KB, o aumento na latência é aproximadamente 31.7% maior com o uso do sequenciador. Por outro lado, para pacotes de 16 KB, a diferença cai para apenas 3.8%. Em outras palavras, ao aumentar o tamanho do pacote de 1 KB para 16 KB a latência aumenta em 224 ms sem o uso do sequenciador, ao passo que com sequenciador o aumento é de apenas 198 ms (i.e., 11.6% menor). Portanto, é possível concluir que conforme o tamanho do pacote aumenta o impacto causado na latência pelo uso da *VNF-Sequencer* é menor.

Diante das análises comparativas, é importante notar que apesar de existir um custo significativo, existe também uma vantagem: a *VNF-Sequencer* implementa a ordem

total utilizada na difusão atômica. Por outro lado, a difusão confiável (sem o sequenciador) não garante nenhuma ordem na entrega das mensagens para as aplicações. Portanto, conclui-se que os benefícios proporcionados pelo uso do sequenciador compensam a latência obtida nesta abordagem.

4.2. Avaliação da Vazão da *VNF-Sequencer*

O próximo experimento tem por objetivo avaliar a vazão da *VNF-Sequencer*, variando o número de processos participantes da difusão. No experimento, cada execução tem duração de três minutos e são apresentados dados médios de três execuções. Ao passo em que o número de processos aumenta, o experimento é executado novamente. É computado o número de pacotes processados por segundo no sequenciador. Dessa forma é calculado, para um intervalo de tempo, quantos pacotes foram executados e repassados para os destinatários. Foi avaliada a vazão considerando de 5 até 50 processos, onde um deles é o processo cliente que gera os pacotes. A difusão neste experimento é atômica.

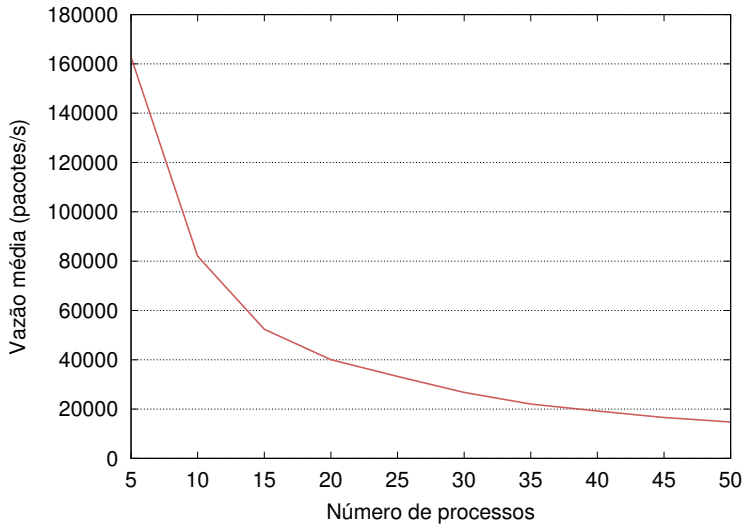

Figura 5. Vazão da *VNF-Sequencer* para a difusão atômica.

Na Figura 5 é possível observar a variação na vazão conforme o número de processos participantes da difusão aumenta. Uma vez que o processo emissor envia pacotes a uma taxa máxima, a vazão da *VNF-Sequencer* é determinada pelo tempo necessário para enviar os pacotes para todos os demais processos. Destaca-se que a latência da difusão aumenta quando aumenta o número de processos. Como consequência, a vazão diminui conforme o número de processos participantes aumenta. Em especial, para até 50 processos, a *VNF-Sequencer* não mostrou ser um gargalo, alcançando uma vazão de aproximadamente 15000 pacotes por segundo.

Os resultados experimentais demonstrados nesta seção permitem concluir que é possível mover os serviços de difusão para a rede de maneira eficiente e que garanta a entrega confiável e ordenada de mensagens para as aplicações distribuídas.

5. Conclusão

Neste trabalho propomos um *backbone* virtual baseado em NFV que oferece as primitivas de difusão confiável para garantir a entrega confiável e ordenada das mensagens

transmitidas na rede. Para realizar tal função, são utilizados dois componentes principais: *VNF-Sequencer* que é um sequenciador que gerencia as transmissões e entrega as mensagens ordenadas aos processos; e *RBCast* que oferece uma API para as aplicações trocarem mensagens utilizando as primitivas de difusão confiável. A principal contribuição é transferir para a rede a implementação e disponibilização dos serviços de difusão confiável e ordenada.

Resultados experimentais mostram que a solução proposta cumpre as expectativas de desempenho. O custo da latência foi apresentado em diferentes cenários, ou seja, variando o número de participantes da difusão e o tamanho das mensagens transmitidas. Por fim, foi medida a vazão aumentando o número de processos participantes na difusão atômica. No experimento observou-se que para até 50 processos a *VNF-Sequencer* não demonstrou ser um gargalo. Uma alternativa para melhorar o desempenho é distribuir a carga em sequenciadores móveis [Défago et al. 2004] ou, até mesmo, implementar sua lógica dentro de dispositivos dedicados como em *switches* proposto em [Li et al. 2016]. Trabalhos futuros também planeja-se implementar a construção da ordem total das mensagens de forma distribuída utilizando consenso.

Referências

[Birman and Joseph 1987] Birman, K. P. and Joseph, T. A. (1987). Reliable communication in the presence of failures. *ACM Transactions on Computer Systems (TOCS)*, 5(1):47–76.

[Chandra and Toueg 1996] Chandra, T. D. and Toueg, S. (1996). Unreliable failure detectors for reliable distributed systems. *Journal of ACM*, 43(2).

[Cotroneo et al. 2014] Cotroneo, D., De Simone, L., Iannillo, A., Lanzaro, A., Natella, R., Fan, J., and Ping, W. (2014). Network function virtualization: Challenges and directions for reliability assurance. In *Software Reliability Engineering Workshops (ISSREW), 2014 IEEE International Symposium on*, pages 37–42. IEEE.

[Défago et al. 2004] Défago, X., Schiper, A., and Urbán, P. (2004). Total order broadcast and multicast algorithms: Taxonomy and survey. *ACM Comput. Surv.*, 36(4):372–421.

[Ekwall and Schiper 2007] Ekwall, R. and Schiper, A. (2007). Modeling and validating the performance of atomic broadcast algorithms in high latency networks. In *13th International Euro-Par Conference*.

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

[Hunt et al. 2010] Hunt, P., Konar, M., Junqueira, F. P., and Reed, B. (2010). Zookeeper: Wait-free coordination for internet-scale systems. In *USENIX annual technical conference*, volume 8, page 9. Boston, MA, USA.

[Kaashoek and Tanenbaum 1991] Kaashoek, M. F. and Tanenbaum, A. S. (1991). Group communication in the amoeba distributed operating system. In *Distributed Computing Systems, 1991., 11th International Conference on*, pages 222–230. IEEE.

[Lamport 1978] Lamport, L. (1978). Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565.

[Li et al. 2016] Li, J., Michael, E., Sharma, N. K., Szekeres, A., and Ports, D. R. K. (2016). Just say no to paxos overhead: Replacing consensus with network ordering. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*.

[McKeown et al. 2008] McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., and Turner, J. (2008). Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74.

[Pedone and Schiper 2003] Pedone, F. and Schiper, A. (2003). Optimistic atomic broadcast: a pragmatic viewpoint. *Theoretical Computer Science (Elsevier)*, 291(1):79–101.

[Reed and Junqueira 2008] Reed, B. and Junqueira, F. P. (2008). A simple totally ordered broadcast protocol. In *proceedings of the 2nd Workshop on Large-Scale Distributed Systems and Middleware*, page 2. ACM.