Gerência do Ciclo de Vida de VNFs e Implementação de Serviços Distribuídos na Rede

Giovanni Venâncio[1]
Rogério Turchetti (Coorientador)[2]**, Elias P. Duarte Jr. (Orientador)**[1]

[1]Universidade Federal do Paraná (UFPR) – Curitiba – PR – Brasil

[2]Universidade Federal de Santa Maria (UFSM) – Santa Maria – RS – Brasil

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

Abstract. *Network Function Virtualization provides a flexible and cost-effective alternative to design and manage network services. In this work, a VNF Manager (VNFM) architecture was proposed to manage the VNF lifecycle. The proposed VNFM simplifies management operations and enables interoperability between platforms. The second contribution proposes VNF-Consensus, a VNF that consistently synchronizes the distributed control plane in an SDN network. The VNF decouples the synchronization mechanisms, avoiding the overhead on the controllers. Finally, we introduce AnyBone, a virtual backbone that provides broadcast services that are deployed on the network itself. AnyBone ensures the order of the messages by using a sequencer which was implemented as a VNF.*

Resumo. *A Virtualização de Funções de Rede oferece uma alternativa flexível para projetar e gerenciar serviços de rede. Neste trabalho foi proposta a arquitetura de um VNF Manager (VNFM) para o gerenciamento do ciclo de vida de VNFs. O VNFM simplifica as operações de gerência e permite a interoperabilidade entre plataformas NFV. A segunda contribuição propõe a VNF-Consensus, uma VNF que sincroniza consistentemente o plano de controle distribuído em uma rede SDN. A VNF desacopla os mecanismos de sincronização, evitando a sobrecarga nos controladores. Por fim, a última contribuição é o AnyBone, um backbone virtual com serviços de difusão confiável e ordenada de mensagens implementados na própria rede. O AnyBone garante a ordem das mensagens através de um sequenciador, também implementado como uma VNF.*

1. Introdução

A Virtualização de Funções de Rede (*Network Function Virtualization* - NFV) surge como uma alternativa para projetar e gerenciar serviços de rede, aplicando tecnologias de virtualização para implementar funções de rede em hardware de prateleira [Mijumbi et al. 2016]. Dessa forma, os serviços de rede são disponibilizados através de Funções Virtualizadas de Rede (*Virtualized Network Functions* - VNFs), simplificando a inclusão e alteração de novos serviços [Mijumbi et al. 2016].

Este trabalho apresenta um conjunto de contribuições que é classificado em duas partes: *(i)* gerenciamento de VNFs; *(ii)* implementação e execução de serviços distribuídos na rede. O VNF *Manager* (VNFM) é responsável principalmente pela gerência do ciclo de vida das VNFs [ETSI 2016]. As plataformas atuais que implementam o

VNFM exigem uma grande quantidade de procedimentos, tornando-se complexas em sua utilização. Além disso, são pouco flexíveis, dificultando a integração entre diferentes tecnologias NFV. A primeira contribuição deste trabalho propõe a arquitetura de um VNFM para suprir estas deficiências, simplificando e flexibilizando as operações de gerência através de APIs que permitem gerenciar de maneira completa o ciclo de vida das VNFs. O VNFM proposto é utilizado na plataforma FENDE (http://coral.ufsm.br/gt-fende), uma *Marketplace* para a distribuição e execução de VNFs e SFCs (Service Function Chains).

As redes SDN (*Software-Defined Network*) separam o plano de controle do plano de dados [Kreutz et al. 2015]. Em geral, o plano de controle é centralizado, o que traz desafios em termos de disponibilidade, escalabilidade e desempenho [Canini et al. 2015]. Um dos maiores desafios é garantir a consistência das operações realizadas na rede que precisam ser consistentemente sincronizadas entre múltiplos controladores. A segunda contribuição deste trabalho é a *VNF-Consensus*, uma VNF responsável por garantir a consistência de operações em um plano de controle distribuído. Esta abordagem desacopla os controladores das tarefas de sincronização, diminuindo a sua carga de trabalho.

A difusão confiável é uma abstração importante para o desenvolvimento de aplicações distribuídas [Défago et al. 2004]. A difusão confiável não garante a ordem em que os processos entregam as mensagens, tornando-se necessária a difusão confiável e ordenada. Em geral, são implementadas na própria aplicação, aumentando a complexidade para o seu desenvolvimento [Li et al. 2016]. Assim, a última contribuição deste trabalho propõe o *AnyBone*: um *backbone* virtual baseado em NFV que oferece primitivas de difusão para garantir a entrega confiável e ordenada das mensagens. O *AnyBone* define a ordem das mensagens através de um sequenciador, disponibilizado através de uma VNF.

O restante deste trabalho está organizado da seguinte forma. A Seção 2 apresenta a especificação de um VNFM para o gerenciamento de VNFs. A Seção 3 descreve a *VNF-Consensus* e a Seção 4 detalha o *AnyBone*. As conclusões são apresentadas na Seção 5.

2. Simplificando o Gerenciamento do Ciclo de Vida de VNFs

O padrão NFV do ETSI (*European Telecommunications Standards Institute*) [ETSI 2016] define o NFV-MANO (NFV *MANagement and Orchestration*), responsável por realizar a gerência das VNFs. Um dos principais módulos do bloco NFV-MANO é o VNF *Manager* (VNFM), responsável principalmente pela gerência do ciclo de vida das VNFs, que consiste primariamente da instanciação, remoção e atualização.

As plataformas que implementam o VNFM proposto pela ETSI [Tacker 2017, OpenBaton 2017, ETSI 2017] são complexas, uma vez que demandam múltiplos procedimentos para operar o ciclo de vida das VNFs. Além disso, tais soluções utilizam requisições REST, diminuindo a flexibilidade e dificultando a integração com outras tecnologias NFV, visto que cada solução implementa uma API diferente. Por fim, estas soluções não gerenciam as VNFs de maneira completa, demandando o gerenciamento manual das funções ou a utilização de ferramentas adicionais. Apesar da diversidade de plataformas que implementam as responsabilidades do NFV-MANO, nenhuma é totalmente adequada ao conjunto de operações previstas para este bloco.

Este trabalho propõe uma especificação para o VNFM com o objetivo de simplificar o gerenciamento do ciclo de vida das VNFs, abstraindo diversas operações e

permitindo a interoperabilidade entre diferentes plataformas NFV. A Figura 1 mostra os principais componentes da arquitetura do VNFM proposto.

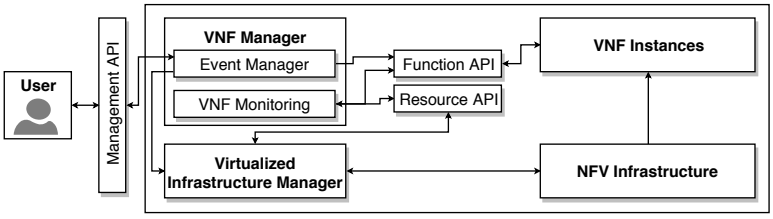


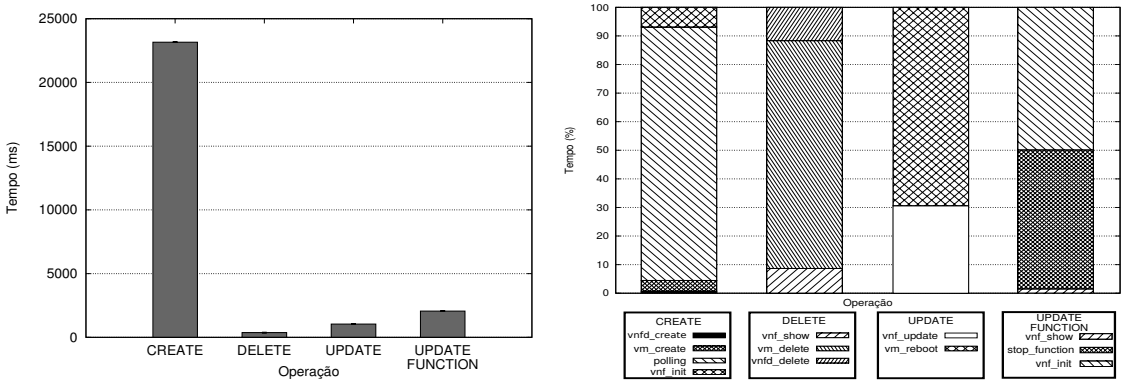
Figura 1. Arquitetura onde o VNFM está inserido.

Para prover a automação necessária foi proposto um módulo denominado *Event Manager* (EM), responsável por receber as requisições da camada de usuário através da *Management API*. As funcionalidades principais disponibilizadas pela *Management API* são: criação, remoção e atualização de VNFs. A partir destas requisições, o VNFM realiza, de maneira automática, todos os procedimentos para as operações de gerência. Para tanto, o EM comunica-se com duas outras APIs para simplificar a compatibilidade do VNFM. A *Resource API* define as funcionalidades para que o VNFM possa gerenciar os recursos das máquinas virtuais que irão hospedar as VNFs, enquanto que a *Function API* define as funcionalidades que o VNFM deve executar em nível de software.

Um protótipo da arquitetura foi implementado em Python e utiliza requisições do tipo REST para comunicação entre seus módulos. Neste trabalho, a *Resource API* foi implementada utilizando como base o Tacker [Tacker 2017], enquanto que para a *Function API* cada VNF disponibiliza uma interface REST.

Avaliação Experimental

Esta seção avalia as principais operações do protótipo executado em um servidor Dell PowerEdge com processador Intel Xeon(R) E3-1220v6 3.00GHz, com 4 núcleos, memória de 8GB e sistema operacional Ubuntu 16.04. Todos os testes foram realizados 30 vezes e os resultados são apresentados com um intervalo de confiança de 95%.



(a) Tempo total de execução das operações.

(b) Tempo de execução das suboperações.

Figura 2. Avaliação das operações de gerência.

A Figura 2(a) exibe o tempo das operações de gerência. A operação *Create* teve a maior duração devido ao tempo em que o sistema aguarda a criação da máquina virtual

(*polling*) para então configurar a função de rede (*vnf_init*), conforme mostra a Figura 2(b). Além disso, o protótipo utiliza um máximo de 1,5% de CPU e 0,25% de memória, demonstrando ser uma solução eficaz para o gerenciamento do ciclo de vida das VNFs.

3. Sincronização Consistente de um Plano de Controle SDN Distribuído

O objetivo das redes SDN é separar o plano de controle do plano de dados [Kreutz et al. 2015]. Em geral, o plano de controle é centralizado [Ho et al. 2016], afetando a disponibilidade, escalabilidade e desempenho das redes SDN [Canini et al. 2015].

Em geral, o plano de controle é distribuído aplicando redundância [Schiff et al. 2016]. Entretanto, é necessário garantir a consistência de operações na rede, já que ações do plano de controle entre múltiplos controladores precisam ser sincronizadas [Schiff et al. 2016]. As soluções que desenvolvem um plano de controle SDN distribuído empregam os próprios controladores para realizar as tarefas de sincronização [Canini et al. 2015, Ho et al. 2016] ou sincronizam as ações da rede no plano de dados [Schiff et al. 2016, Dang et al. 2015]. No entanto, tais soluções oferecem alto custo computacional ou exigem mudanças no protocolo SDN ou nos próprios *switches*.

A segunda contribuição deste trabalho propõe uma VNF para manter a consistência de um plano de controle distribuído, sem aumentar o custo computacional dos controladores e sem modificar o plano de dados ou o protocolo SDN. A VNF, denominada de *VNF-Consensus*, implementa um algoritmo de consenso para manter a consistência no plano de controle SDN. Informalmente, o consenso permite que um conjunto de processos proponham valores iniciais diferentes e entrem em acordo sobre um valor final. Em particular, a VNF proposta é baseada no algoritmo de consenso Paxos [Lamport 1998].

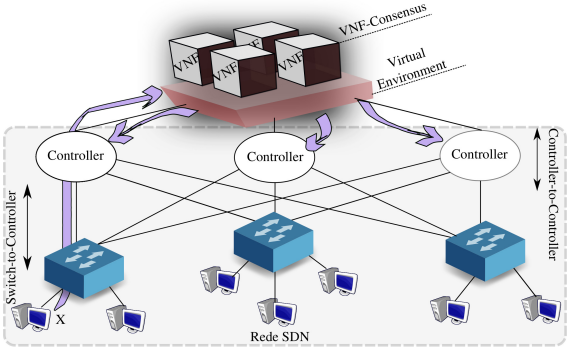


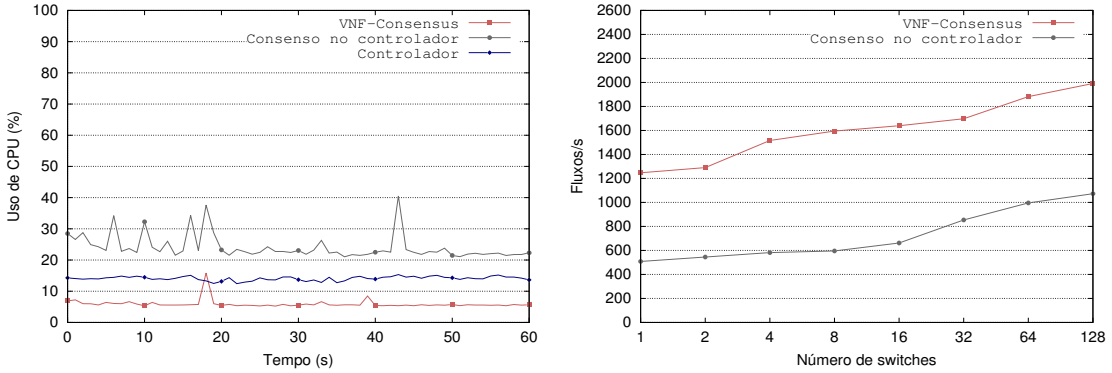
Figura 3. Uma rede SDN executando a *VNF-Consensus*.

A Figura 3 mostra a interação entre os controladores com a *VNF-Consensus*. Quando uma nova regra na rede precisa ser sincronizada, antes de ser instalada o respectivo controlador encaminha esta regra para a *VNF-Consensus*. A instância da *VNF-Consensus* recebe a regra, executa o algoritmo de consenso e retorna a decisão aos controladores, que por sua vez instalam a regra decidida. Note que este mecanismo não é bloqueante e o controlador continua atendendo novas requisições antes da decisão do consenso. Depois que uma decisão é tomada, todos os controladores recebem o resultado final e atualizam seu estado local, mantendo o plano de controle consistente.

Avaliação Experimental

Nesta seção são apresentados os resultados que avaliam a *VNF-Consensus* em um servidor com processador AMD FX-4300 3.8 GHz de 4 núcleos e sistema operacional Ubuntu 16.04. Para criar a rede SDN foram utilizados três controladores Ryu[1], três *switches* e o protocolo OpenFlow [McKeown et al. 2008]. A implementação do Paxos é baseada na libPaxos[2]. A *VNF-Consensus* possui três instâncias, de forma a tolerar uma falha. Para cada experimento três amostras foram coletadas.

A Figura 4(a) mostra a utilização de CPU para diferentes cenários. Quando o controlador executa a sincronização o uso de CPU é em média 27%. Ao utilizar a *VNF-Consensus* a média do uso de CPU do controlador diminui para 14%. Este experimento mostra claramente a vantagem de desacoplar a execução do consenso do controlador. Como consequência, a disponibilidade do controlador aumenta, uma vez que o custo de sincronização é realocado para uma entidade externa.



(a) Utilização de CPU. (b) Fluxos por segundo suportado pelo controlador.

Figura 4. Sincronização do plano de controle: avaliação do desempenho.

Devido a menor utilização de CPU, o experimento exibido na Figura 4(b) mostra que a *VNF-Consensus* aumenta o número de requisições no controlador em até 53%. Além disso, quando os controladores estão sobrecarregados com outras tarefas, a *VNF-Consensus* aumenta a vazão do fluxo de regras instalado em 3,6 vezes. O mesmo ocorre quando o tamanho da rede aumenta: para até 10 controladores, verificou-se que a latência do consenso diminui em média 54% e obtém vazão 2,8 vezes maior.

4. *AnyBone*

Uma abstração importante para o desenvolvimento de aplicações distribuídas é a difusão confiável, que garante a entrega das mensagens 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. Na prática, a implementação da difusão confiável e ordenada é feita na própria aplicação, consequentemente aumentando a complexidade para o seu desenvolvimento [Li et al. 2016].

Em geral, as plataformas e protocolos existentes que implementam a difusão confiável e ordenada executam nas máquinas dos usuários ([Hunt et al. 2010,

[1]https://osrg.github.io/ryu/
[2]https://bitbucket.org/sciascid/libpaxos

Li et al. 2016]). Neste contexto é proposto o *AnyBone*, um *backbone* virtual que oferece difusão confiável e ordenada utilizando NFV em uma rede SDN. Em especial, é oferecida a difusão confiável, atômica, atômica FIFO e atômica causal.

Difusão Confiável de Mensagens pelo *AnyBone*

O algoritmo de difusão confiável implementado pelo *AnyBone* é descrito em [Chandra and Toueg 1996]. Para as difusões ordenadas é utilizado um sequenciador que 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 a ordem global que garante que a ordem de entrega será atômica.

Algoritmo 1 Algoritmo para construção da ordem total das mensagens.

Sender:
1: **Init:**
2: $RBtype \coloneqq AtomicRB$
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
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. Uma mensagem m é transmitida por difusão carregando o contador local de mensagens e o tipo da difusão. O contador global é inserido pelo sequenciador na mensagem e incrementado sempre após a retransmissão de uma mensagem aos processos finais. Cada processo 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' do processo emissor que possui o contador igual ao identificador da próxima mensagem ($nextMsg$) e também se m' ainda está em $pendingMsg$.

Arquitetura e Implementação do *AnyBone*

No *AnyBone* a ordem total das mensagens é garantida através de um sequenciador implementado como uma VNF, denominada de *VNF-Sequencer*, responsável por gerenciar toda a comunicação de forma a garantir as propriedades de comunicação definidas na aplicação distribuída. O *AnyBone* assume que o sequenciador nunca falha e considera-se um canal confiável. Para disponibilizar as primitivas de comunicação às aplicações, é oferecida uma API na biblioteca denominada de *RBCast*.

A arquitetura do *AnyBone* pode ser visualizada na Figura 5. Do funcionamento padrão do protocolo *OpenFlow*, toda mensagem que não possui 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

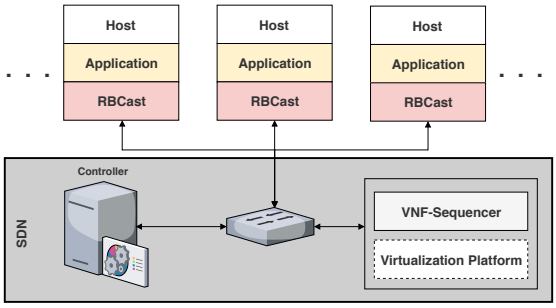
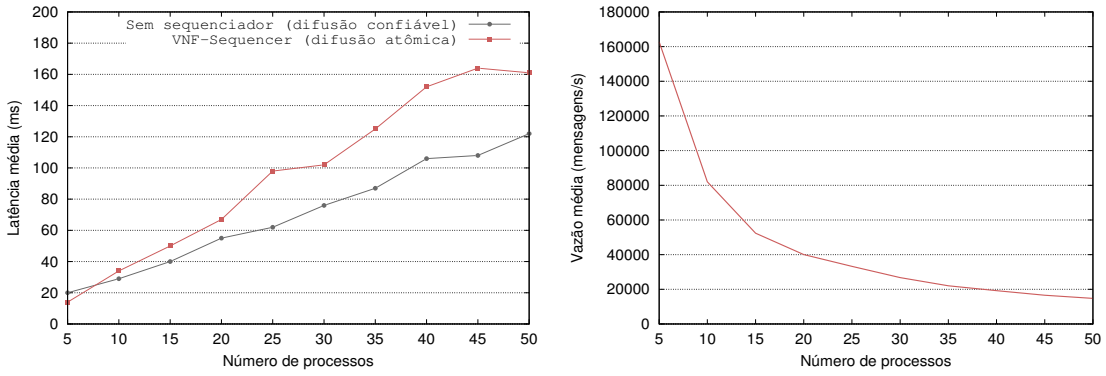


Figura 5. Arquitetura do *AnyBone*.

configurado para que o fluxo de pacotes seja encaminhado para a própria *VNF-Sequencer*. Ao receber o fluxo de pacotes, a *VNF-Sequencer* executa o algoritmo especificado pelo processo emissor e retransmite as mensagens aos processos receptores.

Avaliação Experimental

Esta seção avalia o desempenho da *VNF-Sequencer* em uma rede SDN. O sistema possui processador Intel Core i5-7200U@2.50GHz com 4 núcleos e sistema operacional Ubuntu 16.04. Para cada experimento são executadas 1000 difusões e os dados apresentados são valores médios de 10 amostras, utilizando um intervalo de confiança de 95%.



(a) Latência para a entrega das mensagens.

(b) Vazão para a difusão atômica.

Figura 6. *VNF-Sequencer*: Latência e Vazão.

A Figura 6(a) mostra que conforme o número de processos aumenta, a latência aumenta em média 32% com a *VNF-Sequencer*. Essa sobrecarga ocorre devido a *VNF-Sequencer* garantir a ordem total para todos os processos. Já o experimento da Figura 6(b) mostra que o *AnyBone* atinge uma vazão de 15000 mensagens/s na difusão atômica.

5. Conclusão

Este trabalho propôs um conjunto de contribuições no contexto de NFV, abordando a gerência do ciclo de vida das VNFs e a implementação de serviços distribuídos na própria rede. A primeira contribuição propõe a arquitetura de um VNFM que simplifica o gerenciamento do ciclo de vida das VNFs, de forma completa e a permitir o uso de diferentes plataformas. O VNFM proposto está sendo utilizado em um ambiente de produção na plataforma FENDE, uma *Marketplace* para a distribuição e execução de VNFs.

A segunda contribuição descreve a *VNF-Consensus*, uma VNF que sincroniza um plano de controle distribuído em uma rede SDN. Essa VNF implementa o Paxos para evitar que os próprios controladores executem a sincronização. Por fim, é descrito o *Any-Bone*, um *backbone* virtual que implementa uma VNF para oferecer múltiplos serviços de difusão. Destaca-se que este trabalho tem como contribuição original a implementação dentro da rede de serviços normalmente executados na camada de aplicação.

Referências

Canini, M., Kuznetsov, P., Levin, D., and Schmid, S. (2015). A distributed and robust SDN control plane for transactional network updates. In *IEEE Conference on Computer Communications (INFOCOM)*.

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

Dang, H. T., Sciascia, D., Canini, M., Pedone, F., and Soulé, R. (2015). Netpaxos: Consensus at network speed. In *Symposium on Software Defined Networking Research, (SOSR'15/SIGCOMM)*.

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

ETSI (2017). Open source mano. `https://osm.etsi.org/`. Accessed: 2017-11-21.

ETSI (Available at http://www.etsi.org/technologies-clusters/technologies/nfv, Accessed on October 02, 2016). Etsi gs nfv 002: Architectural framework.

Ho, C. C., Wang, K., and Hsu, Y. H. (2016). A fast consensus algorithm for multiple controllers in software-defined networks. In *18th International Conference on Advanced Communication Technology (ICACT)*.

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.

Kreutz, D., Ramos, F. M., Verissimo, P. E., Rothenberg, C. E., Azodolmolky, S., and Uhlig, S. (2015). Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 103(1):14–76.

Lamport, L. (1998). The part-time parliament. *ACM Transactions on Computer Systems (TOCS)*, 16(2).

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, 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.

Mijumbi, R., Serrat, J., Gorricho, J.-L., Bouten, N., De Turck, F., and Boutaba, R. (2016). Network function virtualization: State-of-the-art and research challenges. *IEEE Communications Surveys & Tutorials*, 18(1):236–262.

OpenBaton (2017). Openbaton. `https://openbaton.github.io/`. Accessed: 2017-11-21.

Schiff, L., Schmid, S., and Kuznetsov, P. (2016). In-Band Synchronization for Distributed SDN Control Planes. *SIGCOMM Comput. Commun. Rev.*, 46(1).

Tacker (2017). Tacker. `https://wiki.openstack.org/wiki/Tacker`. Accessed: 2017-11-21.