УДК 004.41 DOI: 10.17587/it.25.223-228

**И. В. Филиппов,** инженер по разработке программного обеспечения, e-mail: ili.filippov@gmail.com,
**А. Ф. Мелик-Адамян,** канд. техн. наук, архитектор инженерной разработки,
e-mail: areg.melik-adamyan@intel.com,
**В. А. Сухомлинов,** инженер по разработке программного обеспечения,
e-mail: vadim.sukhomlinov@gmail.com,
Intel Corporation US, США

# Размещение сетевых функций в облачных инфраструктурах

*Рассматривается задача выполнения сетевых функций (Network Functions, NF) в облачных инфраструктурах. Предлагается концепция представления сетевых функций в виде микросервисов. Для входа и выхода из системы используется граничная функция, преобразующая входящие пакеты к внутреннему представлению и распределяющая их по микросервисам. Обсуждается реализация системы, в которой полная цепочка микросервисов для каждого пакета выполняется на одном сервере по принципу "run-to-completion" с использованием возможности построения графа обработки пакетов системой NFF-GO.*

***Ключевые слова:*** *сетевые функции, облачная инфраструктура, микросервисы, виртуализация, контейнеризация, NFF-GO*

## Введение

Компьютерная сеть включает в себя не только оборудование для передачи пакетов, но и многочисленные компоненты, реализующие дополнительную функциональность, связанную с промежуточной обработкой пакетов. В качестве примеров такой функциональности можно привести трансляцию сетевых адресов — NAT, фильтрацию трафика — Firewall, операции инкапсуляции и декапсуляции, шифрования и дешифрования, выставления счетов, проверки следования (Service Level Agreements, SLA) соглашениям на предоставление сервиса и т. д.

Такая функциональность называется сетевыми функциями. Изначально сеть строилась по принципу прозрачности — все пакеты, посланные в сеть, пересылались до адресатов без изменений, сетевые функции были интегрированы в приложения и исполнялись на оконечных устройствах вместе с ними. По мере усложнения сетевых функций они были вынесены в сеть, нарушая принцип прозрачности: помимо доставки пакетов сеть стала выполнять функции обработки пакетов.

Как только сетевые функции были вынесены с оконечных устройств, возникла задача их размещения и развертывания. Требования по производительности для сетевой функции, находящейся внутри сети, многократно возрастают по сравнению с оконечным устройством из-за большего трафика, так как функция обслуживает множество оконечных устройств одновременно.

С течением времени объем сетевого трафика растет за счет большого числа новых приложений. Это приводит к росту требований к пропускной способности и необходимости снижения задержек обработки (например, в связи с введением стандарта связи 5G). Реализация сложной функциональности требует проектирования с возможностью масштабирования сетевой функции как в рамках одного сервера, так и, если он не справляется, в многомашинном окружении. В данной статье будет предложена схема развертывания сетевых функций в облачной инфраструктуре, позволяющая достичь многомашинного масштабирования, эффективного управления, легкости развертывания и поддержания отказоустойчивости.

### Существующие подходы

Стандартное (универсальное) оборудование и стандартные операционные системы (ОС) не подходят для высокопроизводительной обработки пакетов. Это связано с особенностями реализации, включающими большое число копирований памяти, с задержками, вносимыми механизмом прерываний, переключениями между пользовательским и системным пространством/режимом работы и другими факторами, делающими возможной универсальную обработку, но снижающими производительность.

По этой причине хронологически первым вариантом размещения сетевых функций является их размещение на специализированном оборудовании. Такое оборудование предназначено для оптимизированного решения одной конкретной задачи. Под каждую сетевую функцию создается отдельное устройство — middle-box.

Со временем появилась возможность создания высокопроизводительных сетевых функций на стандартном оборудовании. Появились специализированные библиотеки, предоставляющие собственные драйверы (например, DPDK), в ОС были добавлены возможности отображения памяти и запуска драйверов в режиме опрашивания, сетевые карты начали предоставлять RSS (Receive Side Scaling) — масштабирование на стороне приема и offloading — исполнение некоторых самых распространенных операций на сетевой карте. Эти и другие изменения сделали возможным второй вариант размещения сетевых функций — на стандартном оборудовании со специальными библиотеками, драйверами и сетевыми картами.

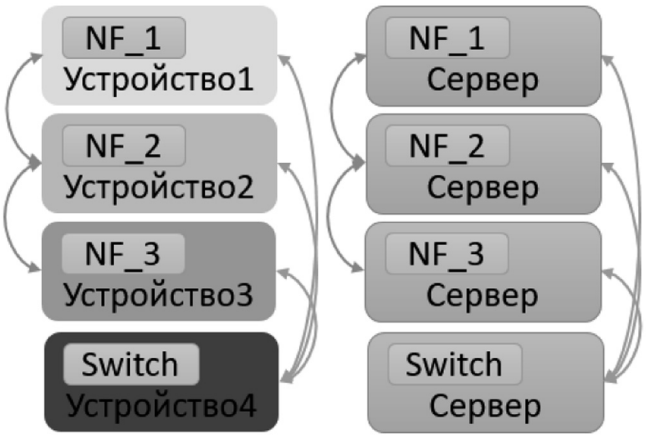


**Рис. 1. "Bump-in-the-wire" вариант размещения сетевых функций, выполняющихся на специализированных устройствах или стандартных серверах (NF — сетевая функция, switch — центральный коммутатор)**

Оба рассмотренных варианта развертывания объединяет жесткое закрепление каждой функции на исполняющем ее оборудовании (отдельном сервере или middle-box), которое связано физическим проводом с центральным коммутатором системы, — так называемый bump-in-the-wire подход (рис. 1). В сетях интернет-провайдеров находятся десятки специализированных middle-box или стандартных серверов, последовательно обрабатывающих пакеты.

Bump-in-the-wire подход имеет существенные недостатки. Набор функций тяжело изменить, так как необходимо физическое перемещение оборудования. Он тяжело масштабируется, так как резервное оборудование простаивает вне пикового времени. Необходимо физически перенастраивать конфигурацию оборудования для обработки отказов.

Эти проблемы привели к появлению третьего варианта развертывания — концепции виртуализации сетевых функций (NFV, Network Function Virtualization) [1, 2]. Основная идея NFV — исполнение сетевых функций на стандартном оборудовании с использованием механизма виртуализации, позволяющего изолировать функции друг от друга. Предполагается, что на одном сервере будут находиться несколько виртуальных машин, каждая из которых будет исполнять одну сетевую функцию [3]. Виртуальные машины будут связаны виртуальным коммутатором, который будет реализовывать последовательную обработку каждого пакета всеми необходимыми функциями (рис. 2, см. третью сторону обложки).

Похожая схема применяется при четвертом варианте размещения — контейнеризации [4] — технологии виртуализации, которая, однако, использует ядро физической ОС (рис. 3, см. третью сторону обложки). Плюсами контейнеров является снижение накладных расходов (память, реализация виртуализации) и быстрый старт и удаление функций.

В случае виртуализации функции находятся под управлением гипервизора, в случае контейнеризации — под управлением одного из оркестраторов (например, Docker).

Для снижения накладных расходов, вносимых виртуальным коммутатором и разделяемым между функциями ресурсом — сетевой картой, были созданы технологии, повышающие производительность сервера с виртуальными функциями: SR-IOV (Single Root I/O

Virtualization) — использование PCI порта одновременно несколькими виртуальными машинами; PCI passthrough — пересылка пакета в виртуальную машину напрямую из PCI устройства, минуя уровень гипервизора; полная или частичная реализация виртуального коммутатора в сетевой карте — offloading.

Концепция NFV является сложной в реализации. Коммутатор должен знать адреса всех сетевых функций, которые задаются ему через управляющий протокол (например, OpenFlow). Кроме того, он же должен выполнять роль балансировщика нагрузки, запуская дополнительную виртуальную машину, если какая-то виртуальная машина не справляется, и выгружая функциональность на отдельный сервер, если текущий сервер полностью перегружен. При этих действиях необходима автоматическая замена сетевых адресов сетевых функций.

Производительность страдает из-за того, что виртуальные машины не могут общаться напрямую и вынуждены каждый раз передавать пакет в коммутатор при реализации последовательной обработки пакета цепочкой функций, что приводит к лишним разборам каждого пакета.

## Предлагаемое решение

♦ ***Сетевые функции как микросервисы***

Одним из популярных шаблонов для создания облачных приложений является применение микросервисов [5]. Шаблон подразумевает разбиение функциональности приложения на небольшие отдельные подзадачи, разворачивающиеся отдельно и поддерживающие связь друг с другом с помощью удаленных вызовов функций (Remote Procedure Call, RPC).

Преимуществами микросервисного подхода является отсутствие необходимости заново развертывать все приложение при изменении отдельных частей, высокая обособленность отдельных частей, которые могут быть написаны на разных языках, разными разработчиками.

Рассмотрим обеспечение необходимой производительности в микросервисах. Каждый сервис может быть масштабирован, т. е. скопирован и запущен на свободных ресурсах. Роль доставки пакетов часто выделяется в отдельный сервис, называемый sidecar proxy, в нем может проходить предобработка пакетов, балансировка нагрузки, динамическое определение маршрута посылки пакета и прочие вспомогательные задачи связи сервиса с сетью.

Задача организации приложения, использующего десятки микросервисов, каждый с десятками клонов, становится достаточно сложной, для ее решения используется специальный уровень инфраструктуры — service mesh, отвечающий за балансировку, отказоустойчивость и другие параметры приложения.

Приложения в основном работают на оконечных устройствах и находятся на уровне L7 сетевого стека. Однако сходную модель можно применить и внутри самой сети для организации сетевых функций. Действительно, как и в конечном приложении, в сетевой функции пакет должен пройти последовательно через различные стадии, которые можно назвать микросервисами. Как и в оконечном приложении, микросервисы можно масштабировать.

Представление сетевой функции или цепочки сетевых функций в виде микросервисов позволяет изменить семантику передачи пакетов между функциями от сетевой адресации к транспортному уровню, например, RPC вызовам или адресации точка-точка, что позволяет исключить виртуальный коммутатор и перейти к однократной коммутации. Кроме того, становится возможна более эффективная реализация передачи пакетов, например, с использованием механизмов общей памяти.

Между сетевой функцией, представленной в виде микросервисов, и остальной сетью необходимо использовать структуру, аналогичную sidecar proxy для отдельного микросервиса. Будем называть такую структуру *граничной функцией.*

♦ ***Граничная функция***

Для перехода от сетевого представления передачи пакетов к микросервисному необходима некоторая граница, которая бы преобразовывала входящий и исходящий трафик между пакетным представлением и внутренним представлением системы (рис. 4).


**Рис. 4. Предлагаемое размещение сетевых функций: F — различные сетевые функции; C — клоны сетевых функций, выполняющие ту же функциональность**

Для каждого входящего пакета граничная функция должна:

1) классифицировать пакет;

2) в результате классификации определить необходимую цепочку микросервисов для обработки пакета и дополнительные параметры для вызовов микросервисов;

3) выполнить вызовы микросервисов согласно определенной цепочке.

Так как сетевые функции обычно работают на уровне ниже L7, классификация пакета должна проводиться на низких уровнях. Например, это может быть разбиение на сетевые потоки по "5-tuple" признакам: двум L3 адресам, типу L4 протокола и двум L4 адресам. По "5-tuple" принципу работает, например, hash функция RSS сетевой карты. Другими примерами может быть классификация на основе MAC-адреса, MPLS-метки, или идентификатора туннеля в GTP. Если входной поток пакетов зашифрован, граничная функция может его расшифровывать или пересылать дальше как единый сетевой поток в зависимости от задачи.

Соответствие пакета и необходимой цепочки микросервисов является внешней настройкой граничной функции. Пакеты, принадлежащие различным сетевым соединениям, могут получать различные сервисы, отличающиеся как последовательностью функций (сервисных цепочек), так и конфигурацией отдельных микросервисов. Администратор сети должен иметь возможность настраивать граничную функцию под различные задачи, задавать правила классификации (например, с помощью протокола OpenFlow) и конфигурировать сервисные цепочки. Предполагается, что сами микросервисы для внешней настройки недоступны, они могут только получать параметры от граничной функции, тем самым реализуя принцип отсутствия внутреннего состояния (stateless), что является основой для масштабирования. Таким образом, реализуются принципы программно-конфигурируемых сетей (SDN — Software Defined Network) [6]. Уровень управления (Control Plane) представлен выбором микросервисов в зависимости от классификации пакета. Уровень данных (Data Plane) представлен микросервисами и отделен от управления.

Помимо управления граничная функция может осуществлять функцию мониторинга созданной системы и предоставлять пользователю различную статистику.

Выполнение найденной цепочки подразумевает несколько действий: инкапсуляцию пакета во внутреннее представление системы, содержащее контекст для исполнения цепочки и последующей декапсуляции к обычному представлению пакета для дальнейшей отправки по сети; передачу инкапсулированного пакета на исполнение с помощью транспорта конкретной реализации (это могут быть RPC вызовы функций, как в канонических микросервисах, либо передача как IP пакет с внутренней адресацией), использование заданных каналов и т. д.

После выполнения цепочки сервисов пакет должен быть возвращен в граничную функцию, преобразован к обычному сетевому представлению и отправлен далее по сети через выходные порты граничной функции.

♦ ***Реализация как run-to-completion***

Ранее мы описали общее представление предлагаемой архитектуры, здесь будет предложена одна из реализаций. Главным недостатком микросервисной архитектуры являются накладные расходы при взаимодействии распределенных сервисов. В случае сетевой функции это пересылка пакетов между микросервисами внутри цепочки.

В качестве эффективного решения предлагается пересылать пакеты по сети только от граничной функции к началу цепочки. Дальнейшее перемещение пакета должно осуществляться внутри одного сервера с помощью разделяемой памяти. Такая архитектура удовлетворяет концепции "run-to-completion": когда пакет попадает на сервер для выполнения, он не выгружается на другие серверы до окончания процесса.

Для реализации всех микросервисов на одном сервере необходим специальный подход к их исполнению. Мы предлагаем использовать нашу систему NFF-GO [7]. Основная идея NFF-GO заключается в построении сетевой функции в виде графа обработки пакетов с помощью зацепления предопределенных, но конфигурируемых блоков обработки пакета, вызываемых для каждого пакета, проходящего по графу. Для реализации шаблона микросервисов достаточно создать граф в виде цепочек, где отдельные микросервисы будут представлены блоками. С помощью нотации загружаемых плагинов можно создать динамический граф, конфигурируемый перед обработкой, а также блоки, загружаемые в граф из сторонних компонентов сторонних производителей.

NFF-GO предоставляет возможность автоматического масштабирования каждого блока в рамках одного сервера на различное число ядер процессора.

Таким образом, общая топология предлагаемого решения сводится к граничной функции, находящейся на отдельном сервере, и множеству серверов, на которых выполняется граф обработки пакетов, созданный с помощью NFF-GO. На каждом сервере осуществляется автоматическое масштабирование в рамках сервера. Если свободных ядер процессора не остается, это означает невозможность обработать на этом сервере входящий поток. В этом случае граничная функция берет на себя функции балансировщика нагрузки, запускает граф обработки на свободном сервере и перераспределяет на него часть сетевых соединений.

Мы нигде не оговаривали, как должны размещаться и администрироваться микросервисы на серверах. В отличие от концепции NFV в нашей модели это не имеет принципиального значения. NFF-GO-граф может быть запущен в виде виртуальной машины, контейнера или другой сущности, администрировать которую предлагает провайдер облачной инфраструктуры. Вопросы запуска, завершения и надежности каждого NFF-GO-графа остаются в ведении оркестратора средств размещения, например, для контейнеров это Kubernetes.

Отметим, что сама граничная функция также представляет собой набор последовательных операций, а значит, может быть реализована с помощью NFF-GO как один из микросервисов, предшествующий попаданию пакетов в остальную систему. Общая модель размещения сетевой функции в облачной инфраструктуре показана на рис. 5 (см. третью сторону обложки).

## Результаты

Рассмотрим преимущества предложенного подхода. По сравнению с существующими способами реализации развертывания сетевых функций в многомашинном окружении, а именно, NFV, контейнеризацией, bump-in-the-wire подходом, предлагаемый метод:

- более простой в управлении, так как имеет один объект, с которым осуществляется взаимодействие, — граничную функцию и однократную коммутацию — требование один раз выбрать цепочку сервисов для каждого сетевого соединения;
- более универсальный, так как не оговаривает конкретные способы размещения сетевых функций на обрабатывающих серверах;
- более производительный, так как run-to-completion модель не имеет накладных расходов на пересылку пакета по сети, а однократная коммутация не предполагает разбора пакета перед каждым новым сервисом;
- более отказоустойчивый, так как на всех серверах находятся одинаковые графы, поэтому вычислительные мощности являются полностью идентичными;
- легче масштабируемый, так как в рамках одного сервера масштабирование осуществляется системой NFF-GO, что предполагает максимальное использование предоставляемых ресурсов, а на уровне многомашинной системы масштабирование осуществляет граничная функция один раз, а не балансировщик нагрузки, находящийся перед каждым этапом обработки.

### Список литературы


1. **ETSI //** Network Function Virtualization. White paper. 2012.

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

3. **Li X., Qian C.** A survey of network function placement // In Consumer Communications & Networking Conference (CCNC). 2016. 13th IEEE Annual. P. 948—953.

4. **Anderson J.** et al. Performance considerations of network functions virtualization using containers. Computing // Networking and Communications (ICNC), 2016 International Conference on IEEE. 2016.

5. **Balalaie Armin, Abbas Heydarnoori, Pooyan Jamshidi**. Microservices architecture enables devops: Migration to a cloud-native architecture // IEEE Software. 33.3 (2016): 42—52.

6. **Xia W., Wen Y., Foh C. H., Niyato D., Xie H.** A survey on software-defined networking // IEEE Communications Surveys & Tutorials**.** 2015. Vol. 17, N. 1. P. 27—51.

7. **Philippov I., Melik-Adamyan A.** Novel approach to network function development // *Proceedings of the 13th Central & Eastern European Software Engineering Conference in Russia* (CEE-SECR'17). ACM, New York, NY, USA, 2017. Article 17. 6 p. DOI: https://doi.org/10.1145/3166094.3166111

**I. V. Philippov,** Software Engineer, ili.filippov@gmail.com,
**A. F. Melik-Adamyan,** Software Architect, areg.melik-adamyan@intel.com,
**V. A. Sukhomlinov,** Software Engineer, vadim.sukhomlinov@gmail.com,
Intel Corporation US, USA

# Cloud Deployment of Network Functions

*The article discusses the problem of the deployment of network functions to a cloud infrastructure. We propose network function representation as a microservice and introduce a notion of the "Cloud Boundary Node" (CBN) for the boundary between functions system and another network. CBN classifies each incoming packet, determines a service chain for it, encapsulates packet to internal representation and sends it for processing, taking load balancing into account. Service chains are represented as a packet processing graph that should be implemented in a "run-to-completion" methodology — whole service chain is done at one server using shared memory technology. Packet processing graph can be built using NFF-GO system, which provides efficient automatic intra-node scaling. Graph structure is the same for all the nodes; however, the graph can be extended by using plugins, which are determined at the packet classification stage.*

*As a result, our architecture is easier to manage due to only one single management object — CBN, where the administrator can set the required service chains for every packet. Architecture can be better scaled and effectively handle various failures because all servers have the same processing graph, and CBN can simply rebalance traffic between them. Great performance is achieved by eliminating the costs of data transferring and commutation: only one commutation happens while CBN classification stage and only one data transfer happen between CBN and handling server.*

***Keywords:*** *Network functions, Cloud infrastructure, Microservices, Virtualization, Containerization, NFF-GO*

DOI: 10.17587/it.25.223-228

**References**

1. **ETSI.** "Network Function Virtualization" white paper, 2012.

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

3. **Li X., Qian C.** A survey of network function placement, *In Consumer Communications & Networking Conference (CCNC)*, 2016, 13th IEEE Annual, pp. 948—953.

4. **Anderson J.** et al. Performance considerations of network functions virtualization using containers. Computing, *Networking and Communications (ICNC), 2016 International Conference on IEEE*, 2016.

5. **Balalaie Armin, Abbas Heydarnoori, Pooyan Jamshidi.** Microservices architecture enables devops: Migration to a cloud-native architecture, *IEEE Software*, 33.3 (2016): 42—52.

6. **Xia W., Wen Y., Foh C. H., Niyato D., Xie H.** A survey on software-defined networking, *IEEE Communications Surveys & Tutorials*, 2015, vol. 17, no. 1, pp. 27—51.

7. **Philippov I., Melik-Adamyan A.** Novel approach to network function development, *Proceedings of the 13th Central & Eastern European Software Engineering Conference in Russia* (CEE-SECR'17), ACM, New York, NY, USA, 2017, Article 17, 6 p., DOI: https://doi.org/10.1145/3166094.3166111

УДК 519.876.5 DOI: 10.17587/it.25.228-233

**А. М. Пуртов,** канд. техн. наук, ст. науч. сотр., e-mail: andr.purtov@yandex.ru,
Омский филиал Института математики им. С. Л. Соболева СО РАН, г. Омск

# Использование образцов для выбора маршрутов в сетях передачи данных

*Рассмотрена возможность использования образцов для маршрутизации в сетях передачи данных. С этой целью разработан образец, построена на GPSSW имитационная модель фрагмента сети передачи данных, проведены имитационные эксперименты. Результаты экспериментов позволяют сделать вывод о целесообразности применения образцов для маршрутизации.*

***Ключевые слова:*** *образцы, принятие решений, графы, целенаправленное движение, маршрутизация, сеть передачи данных, имитационная модель, GPSSW, результаты экспериментов*

## Введение

Разработано много формальных и неформальных методов принятия решений в системах различного назначения, технических, экономических, социальных и др. Обучение и принятие решений по образцам часто используется в живой природе. Автор статьи разра-