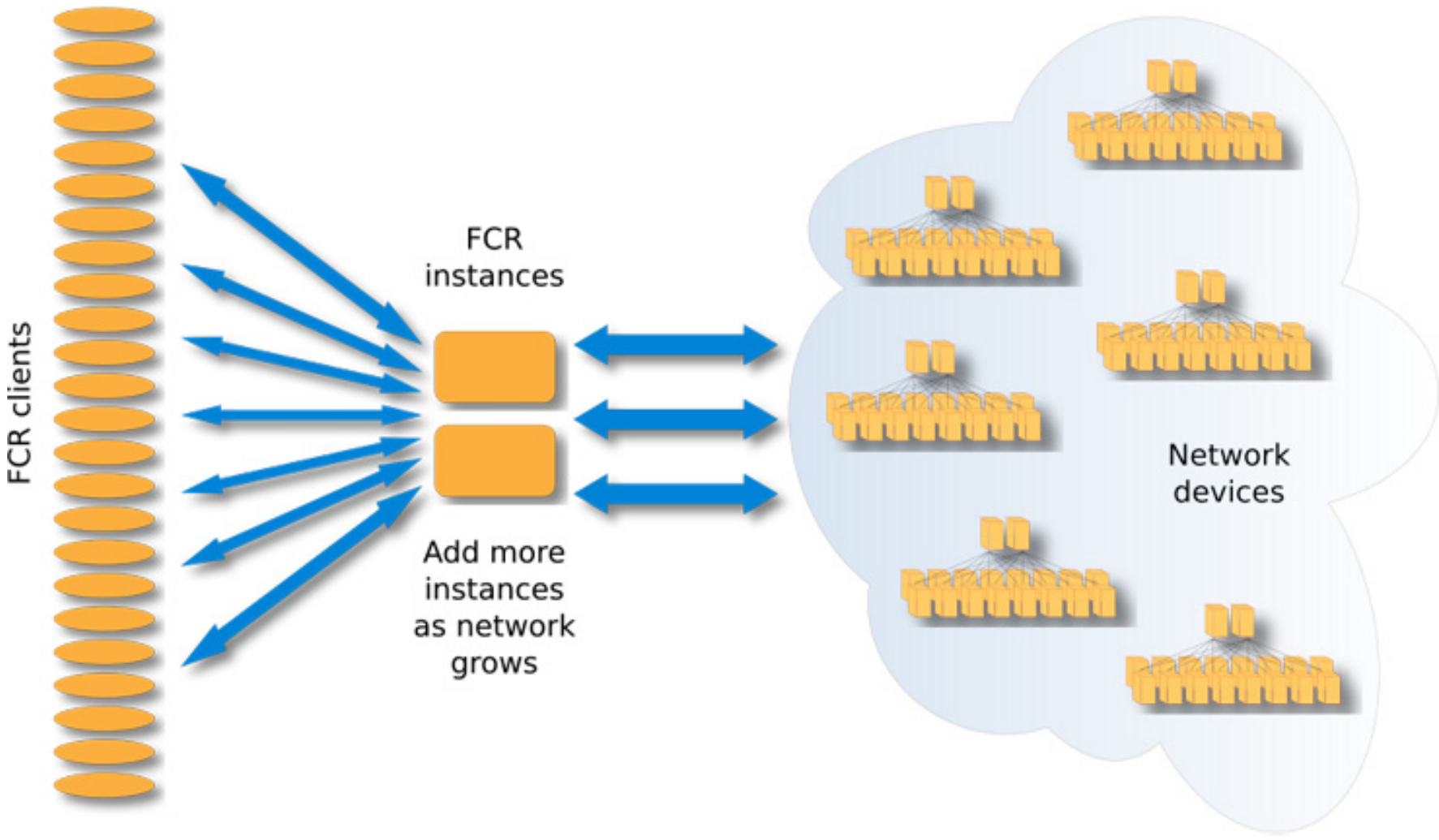
POSTED ON AUG 28, 2017 TO NETWORKING & TRAFFIC

FCR: Open source command runner for network devices



By Surinder Singh Starsky Wong Eric Yu-Wei Sung

As Facebook continues to grow, we are constantly looking for ways to improve our operations and drive greater efficiency. Operating at a global scale poses unique technical challenges, particularly in the network environment, where large amounts of data are always on the move.

Facebook's global network has tens of thousands of network devices connected to hundreds of thousands of servers. To keep the network operational, our network devices — the connective tissue linking everything together — need to be continuously monitored to detect deviations from their desired state. This allows us to be more responsive to issues as they arise.

We monitor the state of the devices by actively collecting operational information through a variety of methods, including SNMP, XML, Thrift, and CLI. Once issues are detected, we need to mitigate them quickly, a process that may include making changes to the network devices. At Facebook's scale, we aim to automate information collection and failure mitigation as much as possible to save time and reduce errors.

Automation challenges

The command-line interface (CLI) is a popular mechanism for managing traditional vendor devices. It often provides read/write access to information that is not available using other methods. However, automation using CLI is not always straightforward, and it poses a few significant technical challenges. For example, each network device requires a separate connection that needs to be opened and maintained. As the network scale grows, latency also becomes an issue. More broadly, automation is constrained by the following:

- **Multiple vendors.** Network devices often come from a variety of vendors. Each vendor typically has different prompts and different setup commands for automated commands (such as setting terminal width

and height), and each may prefer a different session type (SSH, Telnet, Thrift). Although most vendors today support SSH access, some may provide a more programmable interface such as Thrift or gRPC to retrieve information from the devices. Vendors may also have multiple types of devices that require different setups.

- **Large number of devices.** Facebook's network has tens of thousands of network devices and continues to grow. These devices are spread among multiple regions around the globe. Further, each device may be accessible through multiple addresses, such as IPv4 or IPv6, management or in-band addresses. Some of the devices may be outside our core network and need special access. The local policies may prefer some addresses over others. All of this makes it challenging to run commands efficiently. The automation scripts need to be adjusted constantly to help them scale with the growth of the network. Overall, the system becomes non-deterministic and unreliable.
- **Number of commands.** For automation, the commands are run periodically to collect device information. The frequency of each command can vary from a couple of minutes to a few hours. As the number of commands grows, it becomes difficult to guarantee timely execution. To be able to scale, each application will need to distribute the load across multiple instances, which makes the application complex and prone to failures.

Each of these challenges poses problems for network reliability, whether for smaller networks or the large, scaling network we have at Facebook. As we encountered these issues, we knew there had to be a better, more scalable way to monitor network devices. Ultimately, we created FBNet Command Runner (FCR), which we're excited to open-source today at https://github.com/facebookincubator/FCR. FCR is a service that defines a Thrift interface for running commands on network devices. The service is independently scalable and allows people to focus on the business logic. FCR is part of our state-of-the-art Robotron system. We built Robotron's active monitoring features on top of FCR.

In addition to monitoring, FCR can handle provisioning and deployment on devices. Instead of simply managing the connections to devices, the provisioning and deployment tools use FCR to run provisioning commands and push configurations to the devices. With FCR, commands also can be run on devices manually, especially when they need to be run on a large set of devices. Making FCR available is part of a broader effort to open-source parts of Robotron to boost collaboration and improve efficiency across network operators.

FCR service architecture

FCR defines simple Thrift APIs that allow applications to connect and run commands on thousands of network devices without needing to worry about creating threads and managing multiple socket connections to these devices. FCR saves engineers time, since they don't need to worry about matching prompts or any other device-specific details.

We wanted to allow FCR users to write applications in any programming language. The FCR API was defined using Thrift, which allows the client application to be written in any programming language Thrift supports. This design gives users the same consistent API across languages.

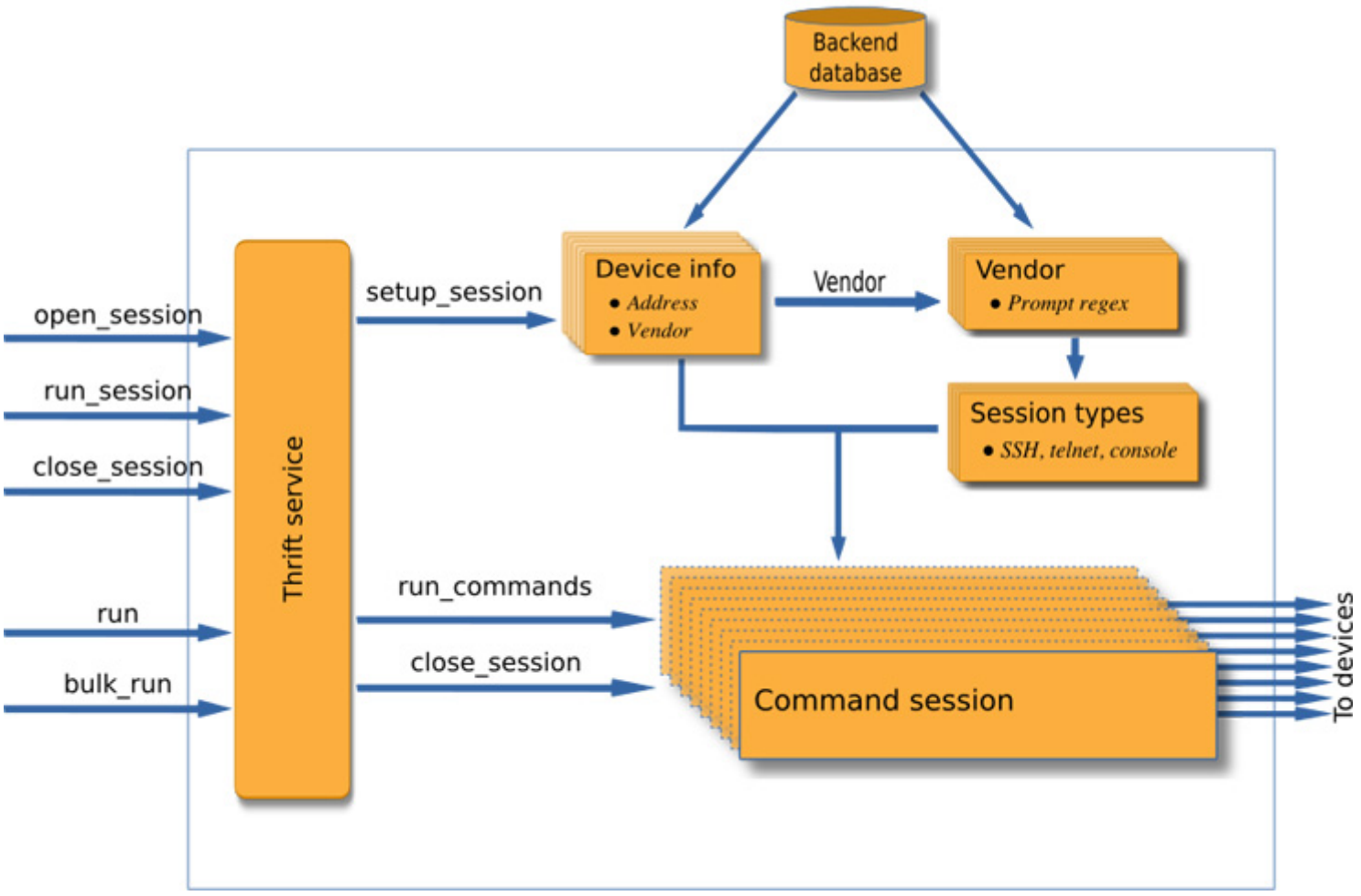
To remove the dependencies on vendors, we abstracted the vendor information, which allows us to adapt quickly as new vendors are added or if a vendor changes the way a device is accessed. Users just need to know what commands to run; they no longer have to worry about the vendor-specific details. They can run commands across devices from multiple vendors seamlessly, rather than worrying about different prompts.

Additionally, we maintain our own device database that lets us gather device attributes without needing to do a time-intensive database query or name lookup. This lets us scale the service efficiently to tens of thousands of network devices. This database is kept in sync with FBNet, allowing us to quickly adapt to any changes to network devices (such as a device being replaced with a different vendor). We can also detect when a device is down and rapidly fail the commands instead of waiting for timeouts. We also use the device availability information to determine the best address to use for each device.

FCR helps us with load balancing across multiple instances. New instances can be added as needed to support large-scale networks, which means client applications no longer have to worry about the network scale. They can easily support a large number of devices using FCR APIs.

Users don't need to be concerned with rebuilding and redeploying their application/services, nor with the scale of the network. They can run requests on a large number of devices without creating threads or maintaining multiple connections. From the user's point of view, these changes are handled transparently, which allows them to focus on the business logic.

The diagram below shows the FCR service architecture.



API design

FCR service defines simple Thrift APIs that allow an application to run commands on thousands of devices. The client applications can be written in any language Thrift supports. You can find the definitions of these APIs at https://github.com/facebookincubator/FCR/blob/master/if/CommandRunner.thrift.

- **run**: runs a command on a device. Multiple commands can be joined together using newline.
- **bulk_run**: runs commands on multiple devices. The commands are run in parallel, with no need to create multiple threads and manage multiple connections in-application. For large requests, the service may further split the request and forward to other service instances. Using this API, we can quickly run commands on thousands of devices.
- **open_session**: runs commands interactively on a device when needed. It allows users to create a session that can be used for this.
- **run_session**: runs a command in open_session.
- **close_session**: closes a session to the device.

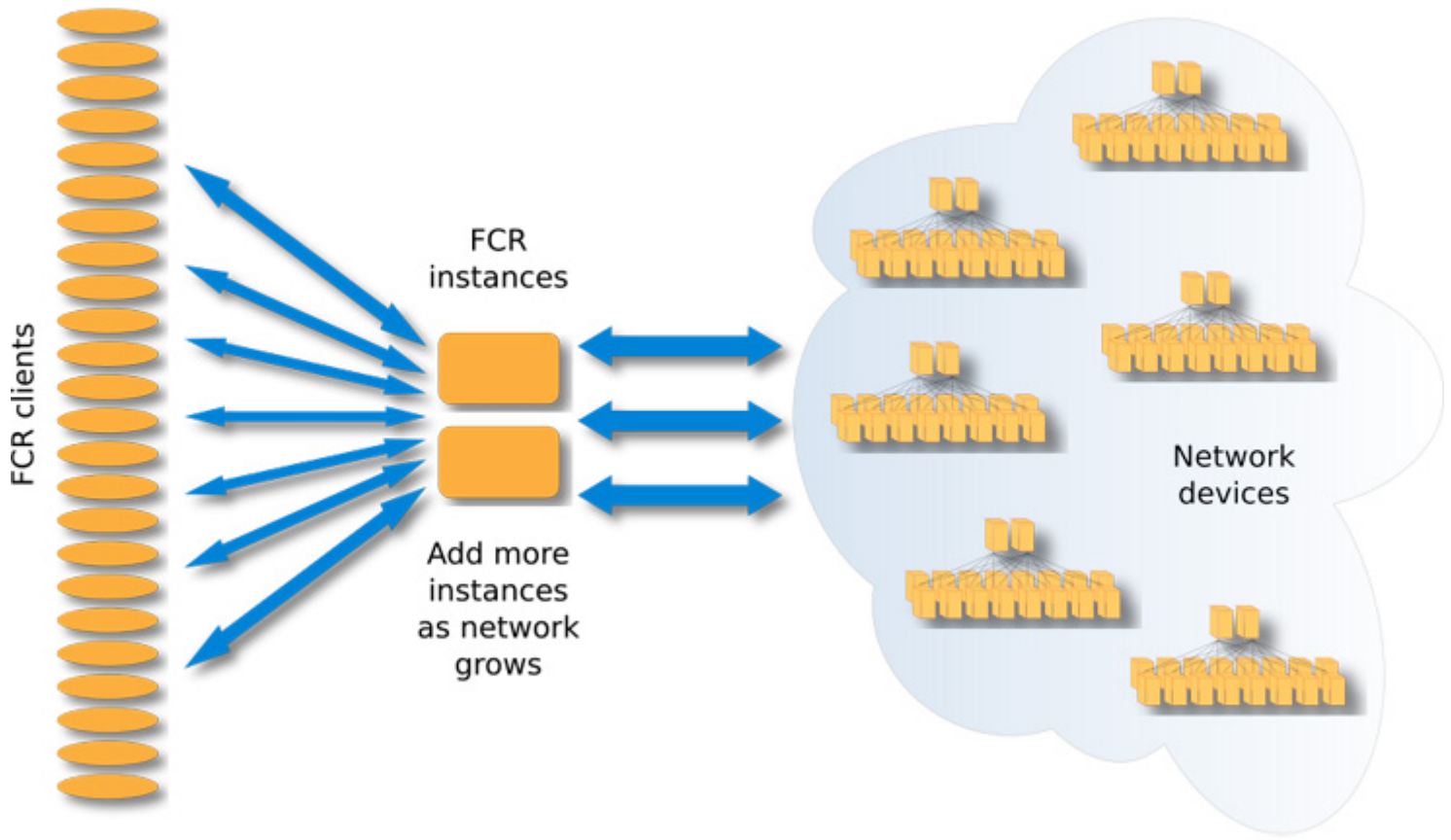
Setting up FCR service

To set up the FCR service, the first step is to customize the specific environment. FCR needs the following information to work effectively:

- Device vendor information: information about devices in your network, such as the vendor name and device prompts.

- Device database: for loading metadata from your data source; for example, names of devices, the device address, and device vendors.

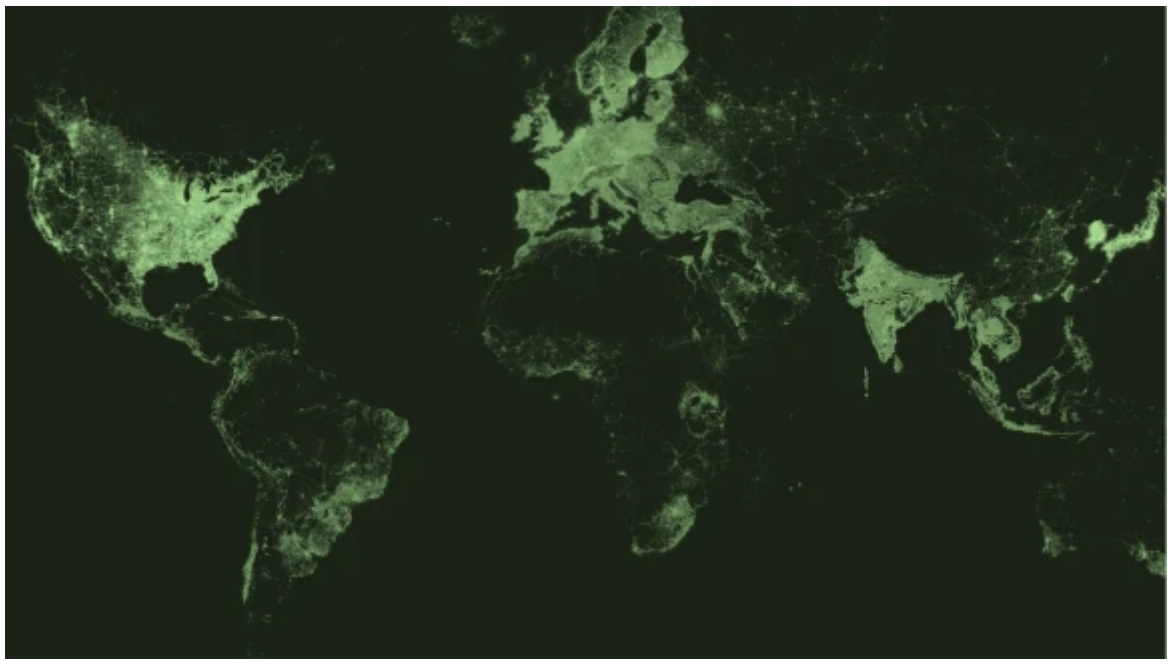
The diagram below shows a sample deployment of an FCR service.



We hope that you can use FCR as a starting point and build upon it to create your own scalable monitoring and deployment system. More details and tutorials on how to get started can be found on the GitHub page. We welcome any feedback that can help us make FCR even better. Thanks to many members from the Network Engineering, Edge and Network Services, and Net Systems teams at Facebook for making FCR a reality.

TAGS: INFRA OPEN SOURCE PERFORMANCE TOOLING

Related Posts

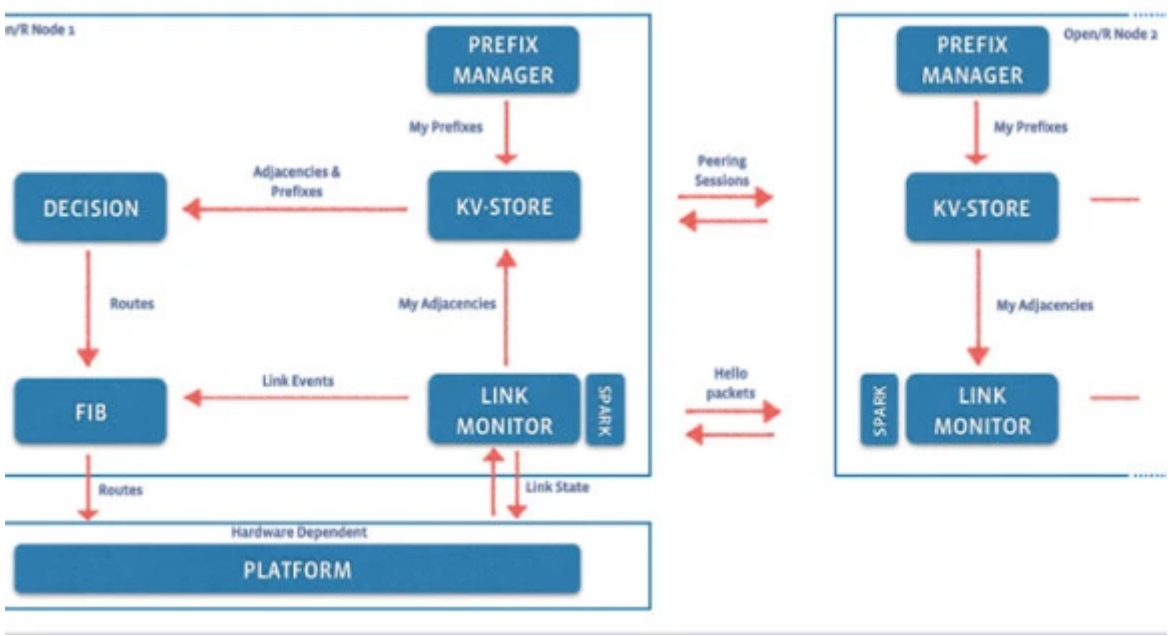


Aug 10, 2018

Announcing tools to help partners improve connectivity

May 21, 2018

Scaling the Facebook backbone through Zero Touch Provisioning



Nov 15, 2017

Open/R: Open routing for modern networks

Related Positions

Software Engineer (AI)

PARIS, FRANCE

System Test Engineering Architect, Portal

MENLO PARK, US

Solutions Engineer

MENLO PARK, US

Software Engineer, Ads Ranking

SEATTLE, US

Software Engineer, Machine Learning (IGTV)

SAN FRANCISCO, US

See All Jobs

Join Our Engineering Community

Available Positions

Software Engineer (AI)
PARIS, FRANCE
System Test Engineering Architect, Portal
MENLO PARK, US
Solutions Engineer

Stay Connected

Facebook Engineering

Like

Open Source

Facebook believes in building community through open source technology. Explore our latest projects in Artificial Intelligence, Data Infrastructure, Development Tools, Front End, Languages, Platforms, Security, Virtual Reality, and more.