1. Introduction to OpenDaylight and the OpenFlow Manager (OFM)

This section explores the architectural framework of Software-Defined Networking (SDN) through the lens of the **OpenDaylight (ODL)** controller and its specialized application, the **OpenFlow Manager (OFM)**[1].

1.1 Conceptual Framework: Software-Defined Networking (SDN)

Software-Defined Networking is an architectural paradigm where an application interacts with a network of domain-specific devices to simplify operations or enable programmable services.

- **The Controller:** A centralized logical entity positioned between the application layer and the physical network.
  Southbound Interface (SBI): The mechanism through which the controller interacts with network elements, such as switches, using diverse protocols.
    - **OpenFlow (OF):** A standard communications protocol that gives access to the forwarding plane of a network switch or router over the network.
  - **Northbound Interface (NBI):** The abstraction layer that presents the network state to applications, typically through common REST APIs.
    - **REST API:** An application programming interface that conforms to the constraints of REST architectural style and allows for interaction with RESTful web services.

1.2 The OpenDaylight (ODL) Platform

OpenDaylight (ODL) is a modular, open-source application development and delivery platform designed for SDN environments. It serves as the "vehicle" for network applications by providing the necessary infrastructure to manage multivendor hardware and diverse protocols.

1.3 The OpenFlow Manager (OFM) Application

The **OpenFlow Manager (OFM)** is a specialized application built to run on top of the ODL controller. Its primary engineering objectives include:

- **Topology Visualization:** Providing a graphical representation of OpenFlow network structures[11].
- **Path Programming:** Enabling the manual or automated configuration of data paths across the OpenFlow fabric.
- **Statistics Aggregation:** Gathering telemetry and operational data from OpenFlow-enabled devices.

1.4 Project Metadata and Development Context

The development of OFM represents a collaborative effort within the SDN ecosystem. The following table outlines the project's technical identity and history.

Attribute	Detail	Significance for Engineers
Project Kick-Off	January 2014	Marks the early adoption phase of production-grade SDN[1515].
Current Status	Enhanced version available with Cisco Open SDN Controller	Indicates the transition from pure open-source to enterprise-grade integration[1717].
Primary Language	JavaScript	Utilized for the development of the front-end user interface[1919].
Key Protocols	OpenFlow, RESTCONF	Defines the communication standards between the app, controller, and hardware.

The data above illustrates a project that began during the initial surge of SDN innovation (2014) and has since matured into enterprise-level deployments[22222222]. For a network engineer, the reliance on **JavaScript** and **RESTCONF** suggests a shift from traditional Command Line Interface (CLI) management toward web-based, API-driven orchestration[232323].

> **Modern Insight (2026):** While OFM and ODL focus on OpenFlow, modern SDN architectures are increasingly adopting **P4 (Programming Protocol-independent Packet Processors)**. P4 allows engineers to define exactly how the forwarding plane processes packets, moving beyond the fixed

"Match/Action" sets of standard OpenFlow. Additionally, many controller functions are now being deployed as **CNFs (Cloud-native Network Functions)** within Kubernetes environments for better scalability.

1.5 Technical Skillsets and Core Components

The OFM project integrates several advanced technologies that must be mastered:

- **YANG:** A data modeling language used to model configuration and state data manipulated by network management protocols.

- **RESTCONF API:** A HTTP-based protocol that provides a programmatic interface for accessing data defined in YANG, using the datastores defined in NETCONF.

- **NeXt:** [Technical Definition: A specialized HTML5/JavaScript toolkit designed for visualizing complex network topologies.

---

2. OFM System Architecture and Control Plane Logic

This section details the architectural layers of the OpenFlow Manager (OFM) and the mechanisms by which it orchestrates data flow across a Software-Defined Networking (SDN) fabric.

2.1 The OFM Architectural Stack

The OFM system follows a tiered approach to decouple the user interface from the complexities of the network protocols.

- **Application Layer:** The top tier where the OpenFlow Manager resides, providing the UI for path programming and topology visualization.
- **Abstraction Layer (ODL):** The middle tier utilizing **RESTCONF** and **MD-SAL** to translate application requests into protocol-specific commands.
    - **RESTCONF API:** [Technical Definition: A HTTP-based protocol that provides a programmatic interface for accessing data defined in YANG, using the datastores defined in NETCONF.
    - **MD-SAL (Model-Driven Service Adaptation Layer):** [Technical Definition: A central infrastructure in OpenDaylight that uses YANG models to define data structures and automatically generate RESTCONF APIs for application interaction.
    - **OpenFlow: You all know this by now.**
- **Infrastructure Layer:** The bottom tier consisting of a network of OpenFlow-enabled switches.

    - **OpenFlow Agent:** [Technical Definition: A software component running on a physical or virtual switch that communicates with the SDN controller using the OpenFlow protocol.

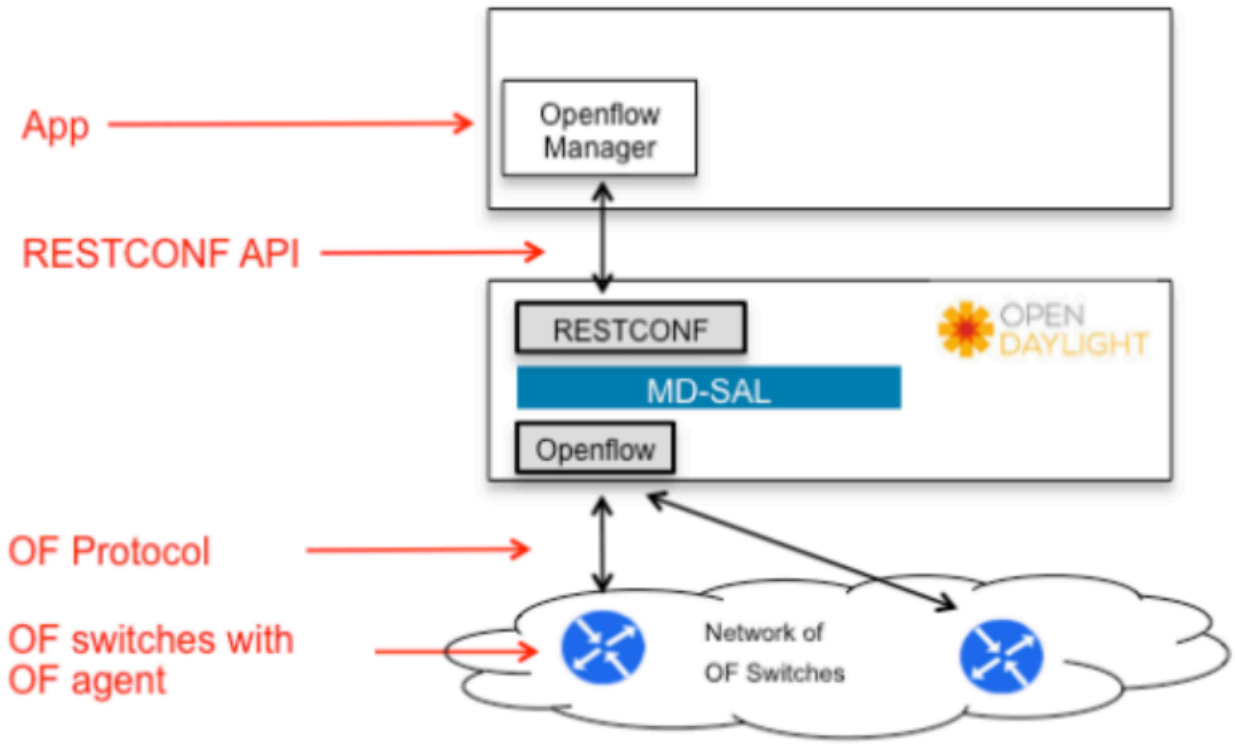


FIGURE 1: OFM System Architecture

Figure 1 in the source materials depicts the vertical communication flow. The **App** communicates downward via the **RESTCONF API** to the **MD-SAL**. The **MD-SAL** then interacts with the **OpenFlow** plugin to send instructions over the **OF Protocol** to the physical or virtual switches.

2.2 Control Plane Protocols and Topology Discovery

To maintain a real-time view of the network, the system utilizes several advanced routing and control protocols:

- **BGP-LS (Border Gateway Protocol - Link State):** [Technical Definition: An extension to BGP that allows the distribution of link-state topology information—traditionally held by OSPF or IS-IS—to external entities like an SDN controller.
    - One router in the network acts as a BGP-LS speaker to transport the topology database to the ODL controller.

- **PCEP (Path Computation Element Protocol):** [Technical Definition: A communication protocol used between a Path Computation Element (PCE) and a Path Computation Client (PCC) to request and send path computation results for MPLS-TE tunnels.
    - ODL uses PCEP to instruct a source router to establish an MPLS traffic-engineered path to a destination.
- **IGP (OSPF/IS-IS):** These protocols maintain the link-state database (LSDB) distributed among all routers in the network.

2.3 The OFM User Interface (UI) Panels

The OFM is accessed through a web browser (e.g., Chrome) and utilizes a modern stack including **AngularJS** and **NeXt**.

Table 2: OFM Functional Tabs and Operations

Tab Name	Primary Function	Engineering Utility
Basic View	Topology and host mapping.	Provides real-time visualization of OpenFlow devices and connected hosts.
Flow Management	CRUD operations on flows.	Used to view, add, modify, or delete flow entries on specific devices.
Statistics	Telemetry aggregation.	Monitors data rates and packet counts for flows and device ports.
Hosts	End-device inventory.	Displays summary information for all discovered host devices.

The structural arrangement of these tabs allows a network engineer to move from Observation (Basic View) to Action (Flow Management) and Validation (Statistics). This workflow simplifies operations by replacing per-device CLI management with an aggregated,

intent-based interface.

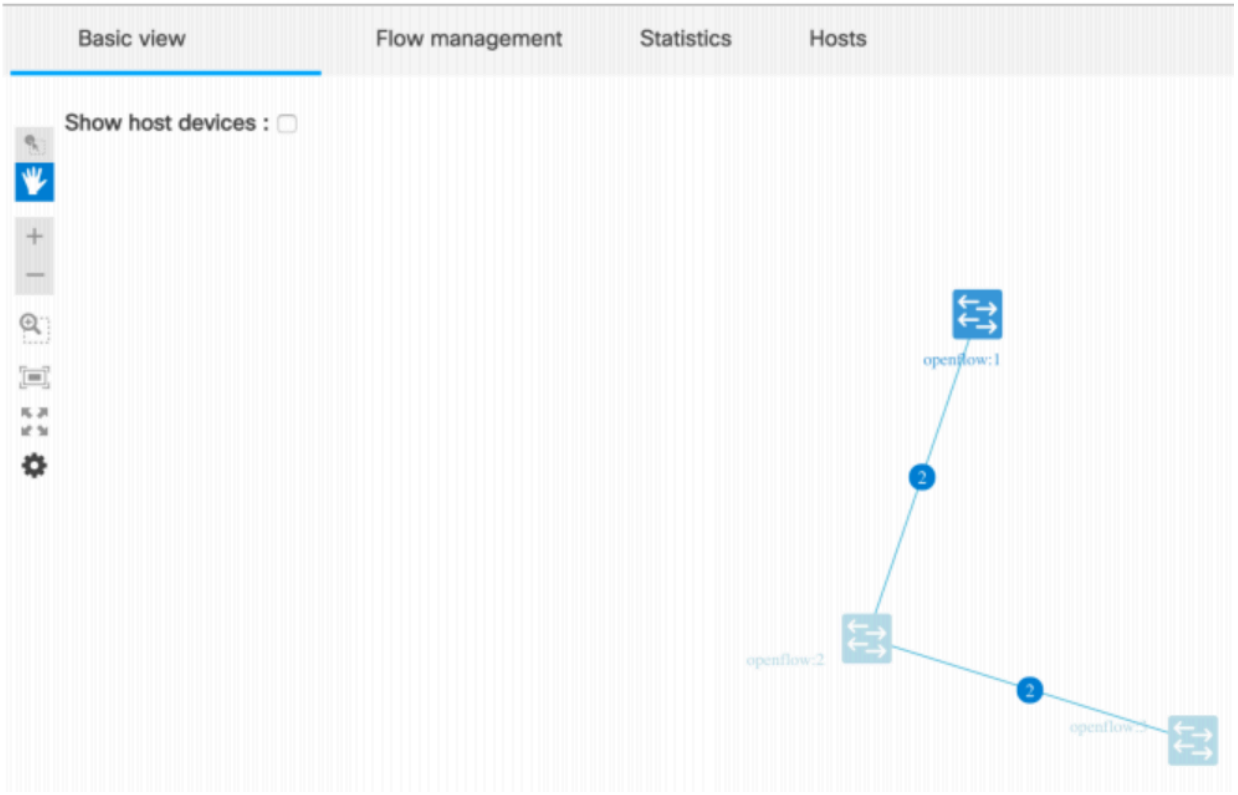


Figure 2 Main OFM panel

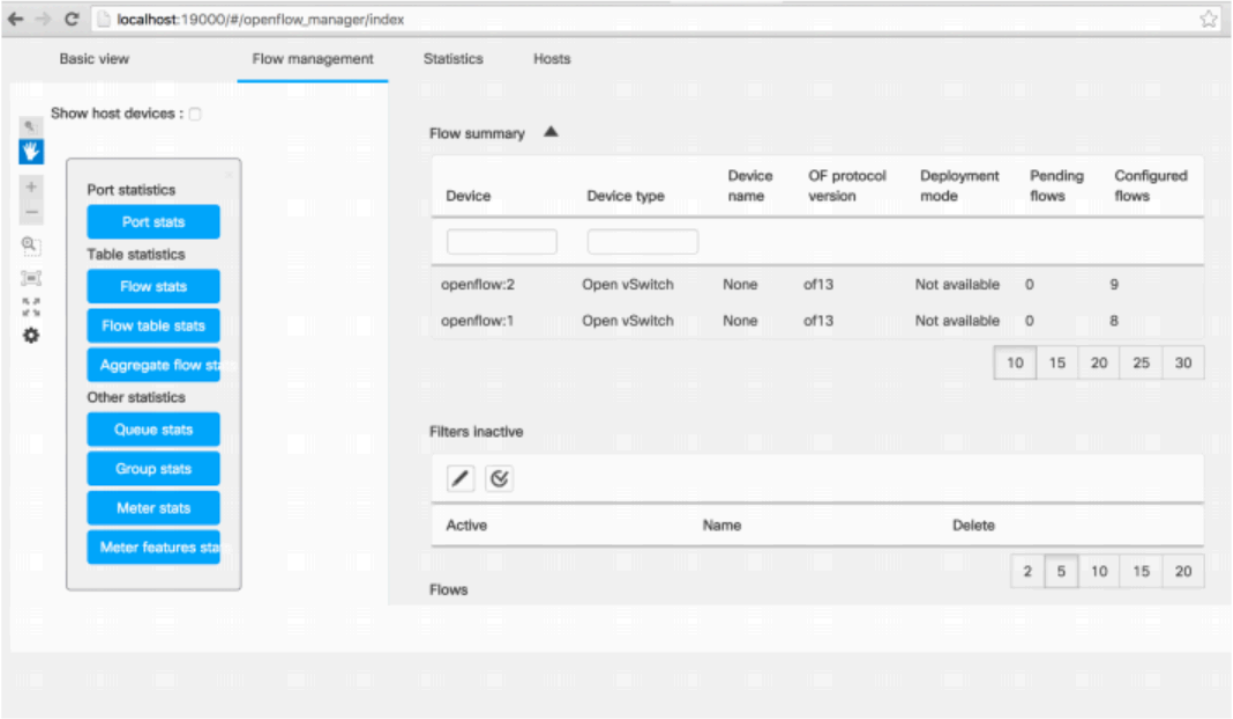


Figure 3 OFM Flow Management panel

And finally the user might want to gather stats from the OF devices as depicted in Figure 4

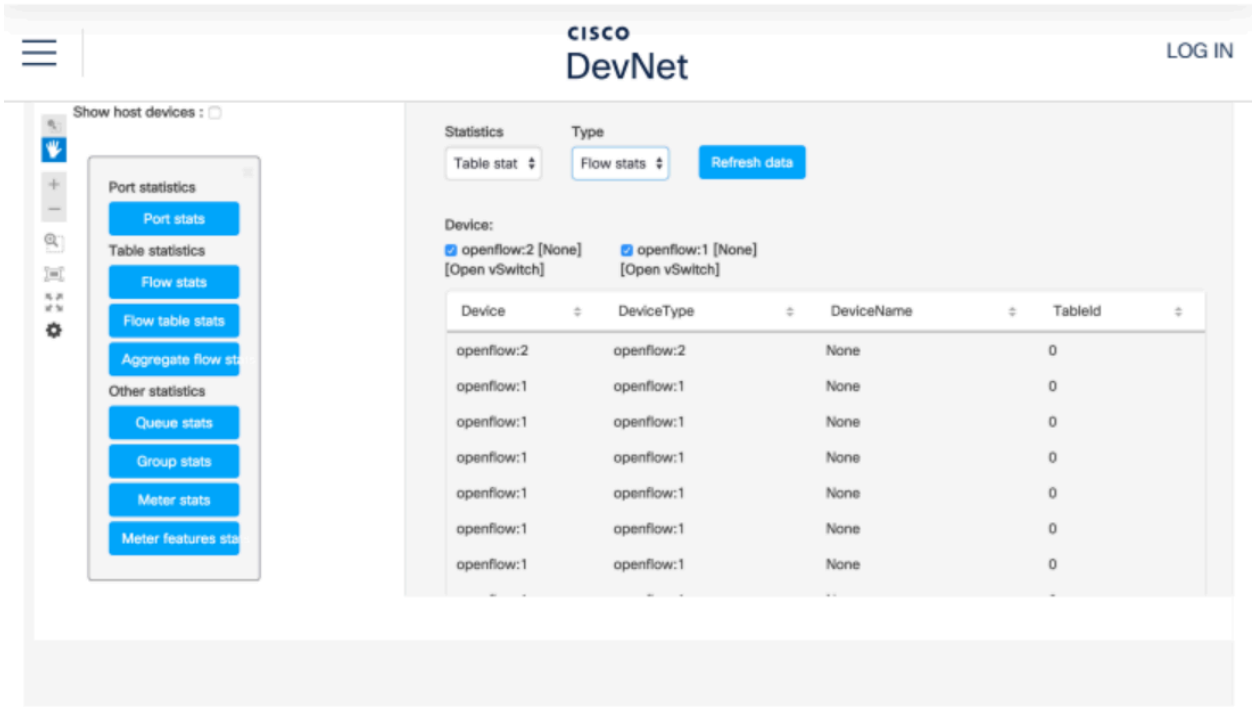


Figure 4 shows the OFM stats screen

2.4 Data Plane Execution

At the forwarding level, switches employ a **"Match/Action"** paradigm.

- **Logic:** When a packet arrives, the switch matches its header against the flow table.
- **Action:** If a match is found, the switch executes the programmed action (e.g., forward to port X, drop, or encapsulate in MPLS).

> **Modern Insight (2026):** In modern 2026 deployments, the "Match/Action" logic is being enhanced by **Stateful In-Network Processing**. While standard OpenFlow is largely stateless, newer **P4-based pipelines** allow switches to maintain state across packets, enabling more complex load balancing and security functions directly in the data plane, reducing the "punting" of packets to the controller.

---

3. Operational Interfaces and Lifecycle Management

This section examines the operational specificities of the OpenFlow Manager (OFM) user interface and the deployment methodologies required to establish a functional SDN laboratory environment.

3.1 The Flow Management Interface

The Flow Management panel is the primary administrative interface for manipulating the data plane. It provides an abstracted view of the flow tables residing on the OpenFlow switches.

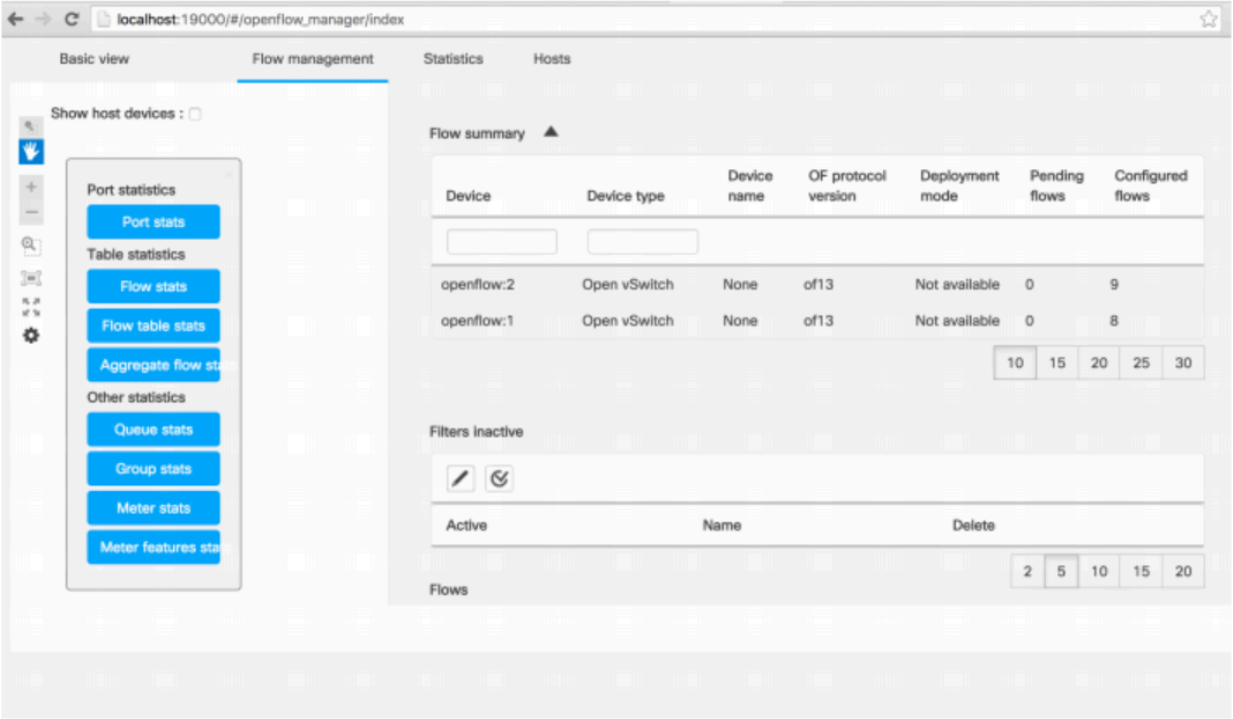


Figure 3 OFM Flow Management panel

- This figure illustrates the relationship between the **Device Inventory** (top) and the **Flow List** (bottom). The side-bar provides navigation to specialized statistics, such as **Queue stats** and **Meter stats**, allowing the engineer to pivot between high-level management and deep telemetry analysis.

Table 3: Flow Summary Data Fields

Field	Technical Description	Operational Significance
Device	Unique identifier (e.g., openflow:1).	Maps logical flow entries to specific physical or virtual hardware.
Device Type	Hardware/Software class (e.g., Open vSwitch).	Informs the engineer of the underlying forwarding capabilities.

OF Protocol Version	The version of OpenFlow being utilized (e.g., of13)[4].	Determines the available match fields and instruction sets (e.g., OpenFlow 1.3).
Configured Flows	Integer count of active flow entries[5555].	Indicates the complexity and current load of the switch's forwarding table.

The data fields in the Flow Summary table allow an engineer to verify that the controller is communicating correctly with the infrastructure. For instance, seeing 13 confirms that the switch supports OpenFlow 1.3, which includes support for multiple tables and group buckets. The Configured Flows column provides an immediate metric for auditing the network's current state.

3.2 Telemetry and Statistics Aggregation

The Statistics panel (Figure 4) allows for the granular monitoring of the network fabric, which is essential for performance validation and troubleshooting.

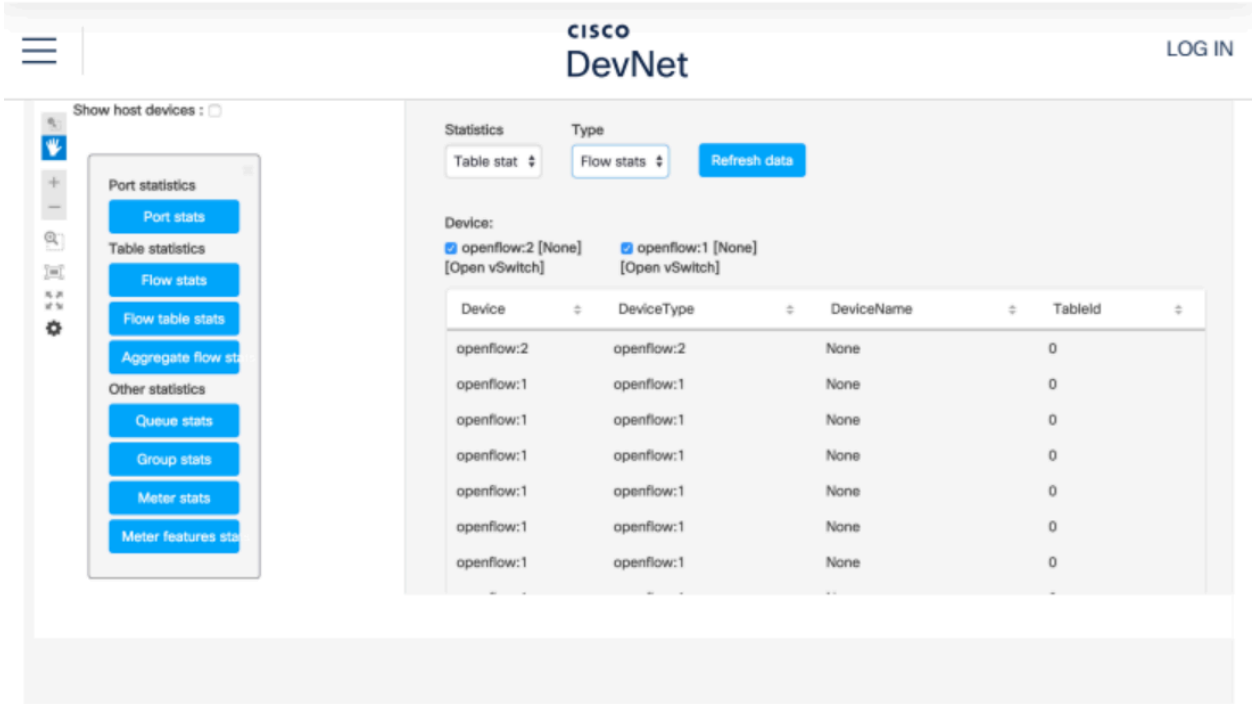


Figure 4 shows the OFM stats screen

- **Port Statistics:** Real-time data regarding packet ingress/egress, byte counts, and error rates on physical or virtual switch interfaces.
- **Flow Table Stats:** Metrics indicating the number of packets and bytes that have matched a specific flow entry.
- **Aggregate Flow Stats:** Provides a high-level summary of total traffic across all flows on a device.
- **Meter & Group Stats**: Statistics related to OpenFlow 'Meters' (used for rate limiting) and 'Groups' (used for multicast or fast-failover paths).

3.3 Deployment and Installation Methodologies

A key objective of SDN is to replace "cumbersome per-device configuration" with a centralized, web-based management approach. To achieve this, OFM supports two primary installation paths:

1. **Vagrant Method:** Utilizes an automated environment provisioner to deploy a pre-configured ODL and OFM instance.https://github.com/CiscoDevNet/OpenDaylight-Openflow-App/tree/Vagrant-setup/Vagrant
2. **Step-by-Step Installation:** Involves the manual configuration of ODL, OFM, and **Mininet**.
    - **Mininet:** A network emulator used to create a topology of virtual hosts, switches, and links on a single Linux kernel, supporting the OpenFlow protocol.
    - **Virtual Machine (VM) Backend:** A pre-built VM image containing Mininet is typically used to host the data plane for development. To run the backend for our app, you can use a pre-built VM image, including Mininet, downloadable from http://mininet.org/download/.

> **Modern Insight (2026):** While manual VM-based installation was the standard in 2014, current 2026 workflows prioritize **Containerized Orchestration**. Engineers now deploy SDN controllers like ODL as **Helm Charts** on Kubernetes. This allows the controller to scale as a **CNF (Cloud-native Network Function)**, providing high availability and automated recovery that was difficult to achieve with traditional VM-based deployments.

---

4. Integration, Configuration, and Environment

Orchestration

This section focuses on the technical integration of the OpenDaylight (ODL) controller with the OpenFlow Manager (OFM) application and the configuration of the underlying network emulator, Mininet.

4.1 OpenDaylight Feature Provisioning

To integrate ODL with the standalone app, you must enable these features:

- **odl-restconf-all**: A feature that enables the RESTCONF protocol, providing the Northbound Interface (NBI) necessary for applications to interact with the MD-SAL.
- **odl-openflowplugin-all**: The Southbound plugin that provides the OpenFlow implementation required to communicate with physical or virtual switches.
- **odl-l2switch-all**: A suite of applications that provide Layer 2 switching functionality, including host discovery and topology management.

4.2 Data Plane Emulation with Mininet

Mininet is used to create a virtualized OpenFlow network. The following command initializes a tree topology and connects it to the ODL controller:

sudo mn --topo tree --controller 'remote, ip=127.0.0.1, port=6653' --switch ovsk, protocols=OpenFlow13

- **--topo tree**: Defines a hierarchical tree structure for the network.
- **--controller 'remote, ip=127.0.0.1, port=6653'**: Points the switches to a remote controller located on the localhost at the standard OpenFlow port 6653.
- **--switch ovsk, protocols=OpenFlow13**: Specifies the use of Open vSwitch (OVS) and enforces the OpenFlow 1.3 protocol standard.

Operational Requirement: After the topology is active, engineers must execute the pingall command within the Mininet console. This generates traffic that allows the ODL controller to discover host nodes and populate the network-topology database.

4.3 Frontend Environment Configuration

The OFM application requires specific environmental parameters to establish a connection with the ODL Northbound Interface. That is, before running the OFM standalone application, it is important to configure the controller base URL, controller port, ODL username, and ODL password. All this information should be written in the env.module.js file located inthe directory ofm src/common/config.

Table 4: OFM Environment Parameters (`env.module.js`)

Parameter	Description	Engineering Purpose
Controller Base URL	The IP/Hostname of the ODL instance.	Defines the network destination for API calls.
Controller Port	Typically, port 8181 is used for RESTCONF.	Defines the specific TCP port used for HTTP communication.
ODL Credentials	Username and Password (default: `admin/admin`).	Provides the necessary authentication for API access.

These parameters are stored in the ofm/src/common/config/env.module.js file16. Proper configuration is critical; if the credentials or URL are incorrect, the OFM frontend will fail to retrieve inventory or program flows, as it cannot authenticate with the ODL MD-SAL17.

4.4 Web Server Deployment via Grunt

To run our standalone app on a local web server. If you do not have anyone, you can use a tool

Grunt, which is installable via npm, the Node.js package manager. Follow this tutorial

http://gruntjs.com/getting-started

This is because OFM is a standalone JavaScript application; it requires a web server for hosting. **Grunt** is used as the task runner to automate this process.

- **Grunt**: A JavaScript task runner used to automate repetitive development tasks, such as minification, compilation, and, in this context, starting a local web server.
- **`grunt-contrib-connect`**: A plugin that provides a static web server for the application.

A simplified `Gruntfile.js` configures the server to listen on port 9000 and serve files from the `ofm` directory. Once running, the engineer accesses the interface via `http://localhost:9000`.

> **Modern Insight (2026):** In modern 2026 environments, static task runners like Grunt have largely been replaced by **Vite** or **Next.js** development servers, which

offer Hot Module Replacement (HMR). Furthermore, the manual editing of `.js` config files is often replaced by **Environment Variables (.env)** or **ConfigMaps** in Kubernetes, allowing the same code to be deployed across different environments (Dev, Stage, Prod) without manual modification.

---

5. Ecosystem Collaboration and Community Integration

This final section outlines the community resources and legal frameworks governing the OpenFlow Manager (OFM) project within the Cisco DevNet ecosystem.

5.1 Code Exchange and Community Collaboration

The OFM project is hosted and maintained through the **Cisco DevNet Code Exchange**, a curated gallery of software projects designed to facilitate network automation and programmability.

- **Community Engagement:** The **Code Exchange Community** serves as a centralized hub for developers to collaborate, share code enhancements, and troubleshoot technical implementations.
- **Resource Access:** Users can access the source code directly via GitHub repository links provided on the platform.
- **Project Ownership and Contributions:** The project is owned by **CiscoDevNet** and includes contributions from multiple independent GitHub developers, reflecting the open-source nature of the ODL ecosystem.

5.2 Technical Categorization

The OFM application is classified based on its primary function and the programming environment required for its extension.

Table 5: Project Classification

Category	Detail	Technical Significance
Primary Category	Networking Tools	Identifies the software as an operational utility rather than a core protocol or driver.

Programming Language	JavaScript	Defines the skill set required for modifying the frontend UI or API call structures.
License	Other	Indicates that specific licensing terms (often open-source) apply to the redistribution of the code.

For a network architect, these categories define the project's "extensibility profile. The reliance on JavaScript ensures that frontend developers can easily integrate new visualization widgets using frameworks like NeXt or AngularJS.

5.3 Legal and Operational Disclaimers

When utilizing open-source or community-contributed code in a production environment, engineers must adhere to specific legal boundaries.

- **Support Status:** Cisco provides these samples for convenience and informational purposes only; they are provided without any form of official support.
- **Third-Party Governance:** Information and links to third-party websites (such as GitHub) are governed by their own separate terms and conditions.
- **Endorsement Policy:** Reference to a contributor or project on the DevNet platform does not imply an official endorsement or affiliation by Cisco.

> **Modern Insight (2026):** In 2026, the concept of "Community Support" has evolved into **AI-Augmented Collaborative Debugging**. Modern repositories often include **Copilot-ready configuration files** and **GitHub Action workflows** that automatically validate code against SDN controller updates. This ensures that even "unsupported" community code maintains a high level of reliability through automated CI/CD (Continuous Integration/Continuous Deployment) pipelines.

5.4 Summary of Resource Links

Beyond the Code Exchange, the DevNet platform provides a comprehensive learning ecosystem for you:

- **Sandbox:** Environment for testing code without physical hardware.
- **Learning Labs:** Structured modules for skill acquisition.

- **Sample Code:** Repository of pre-built automation scripts.