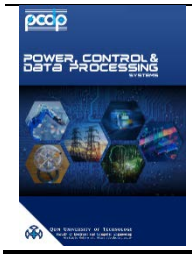
DOI: 10.30511/pcdp.2025.2066687.1037 - ISSN: 3060-8376 - Vol 2(4), 2025



POWER, CONTROL AND DATA PROCESSING SYSTEMS

Available Online at: https://pcdp.qut.ac.ir/

A Comprehensive Survey on Methodologies of Implementing Software-Defined Networking

ARTICLE INFO

Article Type
Original Research

Authors
N.Farajipour Ghohroud[1]

[1] Department of computer engineering, e-Institute of Higher Education, Iranians University, Tehran, Iran,
n.farajipour@iranian.ac.ir

**** Correspondence***
Address: Department computer engineering, e-Institute of Higher Education, Iranians University
Tehran, Iran, Postal Code: 1457896351
n.farajipour@iranian.ac.ir

Article History
Received: July 22, 2025
Accepted: August 04, 2025
ePublished: December 01, 2025

ABSTRACT

Software-Defined Networking (SDN) revolutionizes network architecture by decoupling the control plane from the data plane, which allows for centralized control, enhanced programmability, and remarkable network agility. This paradigm facilitates more straightforward management and dynamic policy enforcement across a variety of devices and environments. The main methodologies explored include OpenFlow-based architectures, which emphasize strict protocol-driven communication between the controller and the data plane switches, enabling direct programmability. Another approach involves API-driven control of legacy or traditional network devices, leveraging standard interfaces like SNMP, CLI, or REST APIs to integrate them into an SDN framework without full replacement. The hypervisor-based overlay virtualization networks method creates virtual networks atop physical infrastructures, enhancing flexibility and isolation, while hybrid SDN deployments combine legacy and SDN elements to balance innovation with operational continuity.

Each methodology presents unique benefits and limitations regarding scalability, security, hardware dependencies, and deployment complexity. For instance, OpenFlow architectures provide high programmability but require compatible hardware, whereas API-driven SDN extends SDN benefits to existing infrastructure at the cost of some openness. Current research challenges addressed include scalability in controller design, robust security mechanisms to prevent attacks such as denial-of-service and spoofing, and standardization to ensure interoperability among diverse vendors and platforms. This comprehensive review assists both researchers and network practitioners in choosing appropriate SDN implementation strategies compatible with specific operational requirements and future-proofing networks through this adaptive technology.

Keywords: Software-Defined Networking, SDN Implementation, OpenFlow, API-based SDN, Overlay Networks, Hybrid SDN, Network Virtualization.

1 Introduction

The explosive growth of cloud computing, virtualization, and dynamic network demands has exposed the limitations of legacy network architectures. Traditional networks tightly couple the control and data planes within switches and routers, resulting in operational complexities, slow configuration changes, and vendor lock-in. Software-Defined Networking (SDN) emerged as a transformative approach to address these challenges by separating network control logic from the forwarding hardware. This logical centralization enables programmable and agile network management, fostering innovation and rapid deployment of new services.

However, the effectiveness and adoption of SDN largely hinge on the choice of implementation methodology. Various approaches—ranging from the hardware-dependent OpenFlow protocol to API-driven control of existing devices and software overlays—offer different trade-offs in scalability, interoperability, and deployment complexity.

This survey categorizes and analyzes the principal SDN implementation methodologies, providing a critical examination of their architecture, advantages, and limitations. The contributions are:

- Presenting a taxonomy of SDN implementation strategies.
- Detailing architectural and operational characteristics of each method.
- Comparing their suitability for different deployment environments.
- Discussing future research directions to address existing SDN challenges.

2 Fundamental SDN Architecture

Writing a conclusion for your research paper can be difficult. Concluding paragraphs should be clear and sum up what you have presented in your research without sounding redundant. An effective concluding paragraph can also add impact to what you have presented in your paper. In this article, you will learn the importance of writing a strong concluding paragraph, how to write one and some tips to help you write the conclusion for your research paper.

Software-Defined Networking (SDN) divides network control into three fundamental components (Controller, Switches and Forwarding Devices and Interfaces) as shown in figure 1, each with distinct roles and responsibilities.

At the core is the **Controller**, a software entity or platform that maintains a comprehensive, centralized view of the network. The controller is responsible for making intelligence-driven decisions about how traffic should be routed and managed across all devices within the network, allowing for coordinated updates and global optimization of resources [1].

The **Switches and Forwarding Devices** operate at the data plane and are tasked with the hands-on job of moving packets through the network. These devices do not make independent forwarding decisions; instead, they implement the instructions received from the controller. By relying on centralized guidance, switches become simpler and more efficient, focusing solely on the execution of forwarding actions as specified by the controller, which reduces complexity at the hardware level.

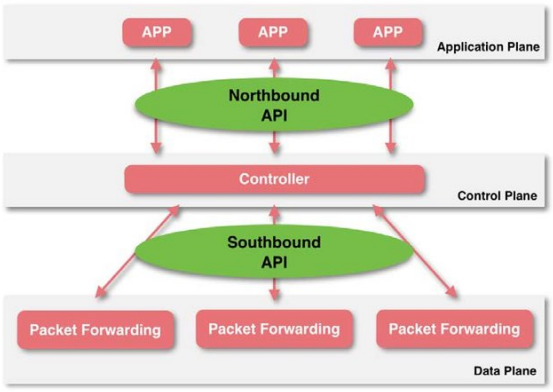


Figure 1: SDN Architecture — Controller, switches, southbound and northbound interfaces.

Interfaces play a crucial role in facilitating communication within the SDN architecture. The **Southbound Interfaces**—such as OpenFlow [2], NETCONF, or proprietary APIs—provide the communication protocols through which controllers interact with the underlying network devices. These interfaces ensure that instructions and configurations from the controller are accurately conveyed to and implemented by switches and routers in the data plane, bridging control and execution effectively.

Conversely, **Northbound Interfaces** are APIs exposed by the controller to higher-level applications and orchestration platforms. Through these APIs, network administrators and external applications can program, monitor, and optimize the network dynamically, without needing to manage individual devices directly. This well-defined separation of concerns in SDN facilitates greater flexibility and agility in network configuration and fosters rapid innovation in the development of advanced network control algorithms and applications.

3 Methodologies for Implementing SDN

3.1 Open SDN Implementation (OpenFlow-Based)

OpenFlow stands as the earliest and most widely endorsed southbound protocol within Software-Defined Networking (SDN). It enables direct, programmable interaction between an SDN controller and network hardware—most often Ethernet switches—by establishing standardized methods to define, install, and manage flow rules. This enables the decoupling of the network's control plane from the data plane, centralizing control logic in the SDN controller and leaving packet forwarding to the underlying hardware.

OpenFlow's architecture is layered and modular. The main components include:

- **OpenFlow Controller:** Acts as the network's "brain," maintaining a global view and directly issuing flow-modification messages to switches.
- **Switches (Physical or Virtual):** Forward packets according to rules (flow entries) set by the controller. Each switch contains flow tables where rules are stored and matched against incoming traffic.
- **Secure Channel:** All controller-switch communications occur over an encrypted or authenticated channel, employing well-defined OpenFlow messages to manage, monitor, and update network state.

The architecture allows flow tables to be dynamically updated, specifying how packets should be handled based on match fields (e.g., MAC address, IP, port) and prescribed actions (e.g., forward, drop, modify).

OpenFlow, as an open standard developed by the Open Networking Foundation (ONF) [3] [4], offers several significant advantages. Its vendor-neutral nature ensures broad compatibility among diverse hardware and software platforms, driving innovation across the networking industry. OpenFlow supports granular, rule-based management at the flow level, enabling real-time, fine-grained control over network traffic. This capability forms the basis for advanced networking features such as load balancing and virtualization. Additionally, by separating the control and data planes, OpenFlow streamlines policy definition, network orchestration, automation, and monitoring, resulting in centralized and simplified management.

Despite its strengths, OpenFlow faces certain limitations. One major challenge is hardware compatibility, as the protocol requires explicit support from switches and networking equipment, making deployment more complex in environments with legacy hardware. Furthermore, in large-scale or highly dynamic networks, the sheer volume of communications between controllers and switches can create performance bottlenecks. This scalability constraint demands careful planning of network architecture and operational strategies to ensure reliable and efficient performance.

OpenFlow proves particularly effective in modern data center deployments, research testbeds, and campus networks. In contemporary data centers, OpenFlow supports agile traffic management, virtualization, and dynamic workload optimization, provided that SDN-compatible switches are used. For researchers and testers, its programmability and wide vendor support enable rapid prototyping and experimentation with new network protocols and management strategies. In campus networks, OpenFlow's policy-driven automation and experimental flexibility make it attractive wherever new hardware can be installed. As OpenFlow continues to evolve, it remains a benchmark protocol driving innovation in network programmability and software-defined networking (SDN).

OpenFlow 1.6+ significantly improves upon the basic OpenFlow 1.0 by introducing enhanced flexibility and extensibility. It replaces the fixed match fields with a flexible TLV (type-length-value) format for richer flow specifications, supports extended and consistent actions and instructions, and expands multi-table pipeline processing for more sophisticated flow management. Port name length limits and supported protocol fields are increased, enabling better scalability and expressiveness. While OpenFlow 1.6 was not finalized as a formal ONF standard, it represents the latest proposed enhancements that refine OpenFlow's match/action model and multi-table architecture beyond the limitations of the single-table, fixed-structure design of OpenFlow 1.0.

3.2 API-Based SDN Implementation

To facilitate incremental adoption, many Software-Defined Networking (SDN) controllers provide support for device management through standardized or vendor-specific application programming interfaces (APIs) such as REST, NETCONF, or SNMP extensions. This architectural approach enables controllers to interact with traditional network devices using these APIs instead of relying solely on protocols like OpenFlow. By leveraging existing infrastructure, organizations can reduce both the cost and complexity associated with a full hardware overhaul. Additionally, this method offers greater flexibility for integrating SDN capabilities with heterogeneous equipment from different vendors, making it easier to adopt SDN principles gradually across networks that may contain a mix of legacy and modern devices.

However, this approach is not without its limitations. One significant challenge is the limited standardization of APIs, as many vendors offer proprietary extensions that can reduce interoperability and complicate portability between different systems. This fragmentation means that network operators may encounter difficulties when attempting to manage multi-vendor environments, as a unified management strategy may not be fully attainable. Moreover, not all device features are guaranteed to be exposed programmatically via these APIs, potentially restricting the scope of automation and programmability. Despite these constraints, this method is particularly attractive for use cases such as enterprise campus networks and carriers seeking a gradual evolution toward SDN without a wholesale refresh of existing hardware, thus providing a practical pathway to modernizing network management.

Recent advances in API-based Software Defined Networking (SDN) have prominently featured enhancements in YANG data modeling and NETCONF protocol versions. YANG continues to evolve as the key vendor-neutral data modeling language enabling unified, programmable network configurations, addressing ambiguities and extending capabilities based on operational feedback from major open-source projects like OpenDaylight. NETCONF 1.1 has seen updates focused on improved event notifications and secure, efficient session management. Moreover, the adoption of RESTCONF as a simplified REST-like interface for YANG data has grown, providing a developer-friendly option

alongside NETCONF for network automation. These developments enable seamless integration with automation tools (Ansible, Puppet), multi-vendor device programmability, and support dynamic network orchestration in 5G/6G environments. Frameworks such as ETSI's TeraFlowSDN leverage YANG/NETCONF to unify control of packet-optical networks, while initiatives like the Common API Framework (CAPIF) standardize API management across network layers, enhancing discoverability, authentication, and real-time service orchestration at the edge. Overall, the progress since 2023 reflects a strong industry movement towards mature, open, and automated SDN ecosystems driven by these API-centric standards and models [5] [6].

3.3 Hypervisor-Based Overlay SDN Networks

Overlay networks are designed to provide network virtualization by using tunneling protocols such as VXLAN and NVGRE. These protocols abstract the underlying physical network topology, allowing virtual networks to be created and managed independently of the existing hardware infrastructure. This decoupling of the virtual and physical layers enables organizations to build flexible and scalable network architectures atop their current infrastructure without the need for disruptive hardware changes.

The architecture of overlay networks revolves around the use of Virtual Tunnel Endpoints (VTEPs). VTEPs are responsible for encapsulating and decapsulating network traffic, effectively creating logical networks that overlay the physical underlay. Centralized controllers play a critical role in orchestrating these overlays, managing network policies, and ensuring tenant isolation within multi-tenant environments. This architecture streamlines network management and improves the ability to automate policy enforcement, making deployment and scaling more straightforward.

Overlay networks offer several important advantages. Because they operate independently of the physical hardware, they require no modifications to existing devices, significantly reducing the complexities and costs of upgrades. This approach simplifies VM mobility across the data center and facilitates secure multi-tenancy by isolating different tenants' network traffic. However, overlay networks are not without their challenges. The additional encapsulation introduces overhead, potentially impacting network efficiency. Furthermore, the added layer of abstraction can complicate network troubleshooting and performance monitoring. Overlay networks are widely used in large cloud data centers and multi-tenant environments, where their benefits in scalability, agility, and security are crucial.

Recent innovations in overlay SDN have prominently featured the integration of Segment Routing over IPv6 (SRv6) with VXLAN, enabling highly scalable, programmable, and simplified network architectures. SRv6 eliminates the need for multiple legacy protocol layers by using IPv6 for segment routing, which enhances end-to-end service delivery across diverse network domains, including underlays and overlays, data centers, and cloud environments. This integration offers improved load balancing, source path control, and stateless policy enforcement, all critical for dynamic and scalable cloud workloads. Additionally, AI-optimized overlay orchestration is advancing rapidly, leveraging machine learning and intelligent automation to dynamically manage cloud workload placement, resource allocation, and scaling. These AI-driven orchestration systems improve operational efficiency, reduce costs by minimizing over-provisioning, and enhance performance through continuous monitoring and adaptive control, thus enabling more responsive and resilient multi-cloud deployments. Together, SRv6-based overlays and AI-enhanced orchestration represent a transformative approach to managing the complexity and demands of modern dynamic cloud infrastructures.

3.4 Hybrid SDN Networks

Hybrid Software-Defined Networking (SDN) [7] represents an approach that combines the flexibility of SDN with the proven reliability of traditional, legacy network devices within a unified operational environment. This integration supports organizations in gradually adopting SDN technologies, allowing them to leverage existing infrastructure investments while incrementally introducing programmable network elements. Hybrid SDN deployments are often structured using various models, such as the topology-based model—which involves deploying OpenFlow switches in specific parts of the network only—and the service-based model, where SDN controllers manage particular network services like firewalls, while other aspects remain under legacy control. Additionally, class-based and integrated hybrid approaches have been developed to offer even greater flexibility, catering to the diverse requirements of modern networks.

Hybrid Software-Defined Networking (SDN) models—namely topology-based, service-based, and class-based approaches—are not mutually exclusive and may coexist concurrently within a single network infrastructure. Scholarly research and technical analyses demonstrate that the integration of multiple hybrid SDN models is both feasible and often advantageous, allowing network operators to realize a broader spectrum of design trade-offs and implement tailored migration strategies.

For instance, an operator might employ a topology-based hybrid SDN model to ensure compatibility with legacy systems in certain network segments, while simultaneously adopting a service-based model in other segments to deliver specific premium services. Furthermore, purely SDN-enabled zones, such as data centers, may also be maintained within the same overarching network environment. This multi-layered or hybrid deployment paradigm facilitates incremental SDN adoption, enables precise architectural optimizations, and supports adaptation to evolving organizational requirements.

The concurrent application of multiple hybrid models allows operators to:

- Implement gradual SDN deployment across different regions or services (e.g., topology-based hybrid SDN for regional rollouts and service-based hybrid SDN for targeted services).
- Assign distinct traffic classes (via class-based hybrid SDN) or service categories to either SDN or traditional control planes depending on operational or business needs.
- Maintain flexibility in addressing coexistence challenges while minimizing disruptions to legacy systems.

One of the major advantages of hybrid SDN architectures is their cost-effectiveness during network transition phases. Organizations can implement SDN technologies in a phased manner, minimizing risks and upfront expenses associated with complete infrastructure overhauls. By maintaining the stability and operational familiarity of legacy systems, organizations can ensure ongoing service reliability while exploring the benefits of SDN, such as improved automation, traffic management, and policy enforcement. This balance makes hybrid SDN especially appealing for service providers and large enterprises that have significant investments in existing network hardware and require uninterrupted network operations during the migration process.

However, hybrid SDN environments introduce their own set of challenges. The coexistence of multiple control paradigms often leads to increased network complexity, as teams must manage both traditional and SDN-based network components. This complexity can result in policy conflicts, inconsistent traffic handling, and operational difficulties, particularly as the intricacies of configuration and troubleshooting multiply. Despite these limitations, hybrid SDN remains a pragmatic approach for organizations seeking to modernize their network infrastructure while preserving the stability and value of their legacy systems. Its use is particularly prevalent in large-scale environments that require both gradual evolution and high levels of service reliability.

4 Comparative Analysis

OpenFlow-Based SDN relies on a centralized controller that communicates directly with OpenFlow-enabled switches, separating the control and data planes. This open standard enables interoperability between vendors and provides network programmability but usually requires specialized hardware or firmware support. Deployment is moderate in complexity and is often found in data centers and experimental networks, where flexibility in traffic management and rapid fault recovery are critical.

API-Based SDN uses controllers interfacing with existing, often proprietary, equipment through legacy command-line or vendor-specific APIs. This method achieves high scalability across existing hardware without requiring device replacement, making it particularly attractive for enterprises aiming for incremental SDN adoption. However, the reliance on closed or semi-open vendor APIs can limit openness, potentially leading to vendor lock-in and reducing interoperability between different equipment suppliers.

Overlay SDN typically employs hypervisor-based or tunneling technologies (such as VXLAN) to establish virtual networks atop existing physical infrastructure, requiring no underlying hardware changes. This approach achieves high scalability and the lowest deployment complexity, suiting dynamic cloud environments where provisioning speed, multi-tenancy, and agility are top priorities. Its use in cloud data centers and service provider networks reflects the demand for rapid, software-driven network changes.

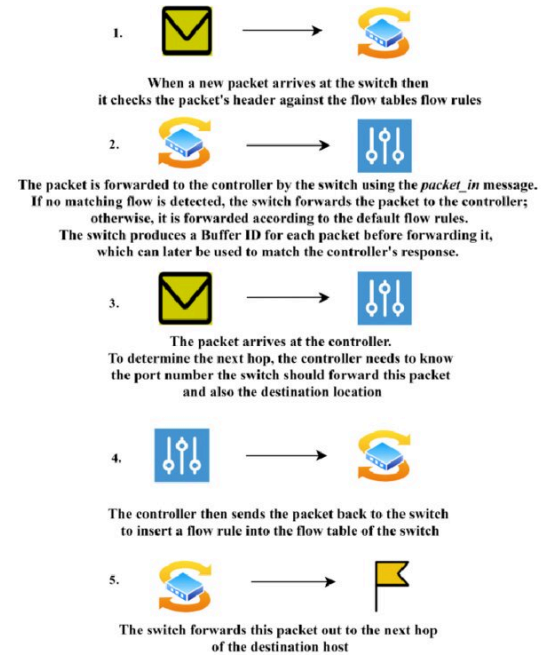


Figure 2: Workflow procedure of OpenFlow-based switches [8].

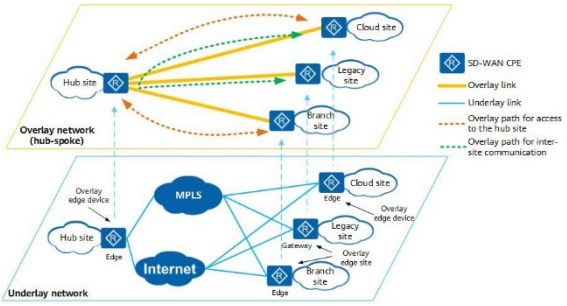


Figure 3: Overlay-based SDN architecture

Hybrid SDN combines traditional and SDN approaches through mixed control planes, supporting both legacy and SDN-capable devices. This allows for flexible, phased migration strategies in large, multi-vendor environments such as ISPs. While providing broad interoperability and customized evolution paths, the complexity of deployment and ongoing management can be significantly higher due to the need to integrate diverse systems and protocols and address the challenges of coordinating control logic across domains.

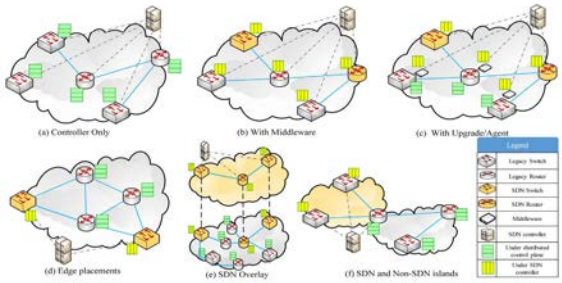


Figure 4: Various hybrid SDN Models classified based on architecture and components [9]

Each SDN architecture presents distinct trade-offs among openness, scalability, investment in hardware, and operational complexity. The optimal choice depends on existing infrastructure, desired level of automation, and long-term strategic goals for network evolution. The comparison is summarized in Table 1.

5 Numerical Comparison of SDN Implementation Methodologies

This section provides **quantitative simulation results** to compare different SDN implementation methodologies. Metrics considered include **latency, throughput, scalability, and resource utilization** as reported in research and simulation studies and is summarized in Table 2.

Table 1. Comparison of SDN Implementation Approaches

Implementation Method	Architecture	Scalability	Hardware Requirements	Openness/Standardization	Deployment Complexity	Typical Applications
OpenFlow-Based SDN	Controller + OpenFlow Switches	Moderate	OpenFlow-enabled switches	High (Open Standard)	Moderate	Data centers, experimental networks
API-Based SDN	Controller + Legacy APIs	High (device bound)	Existing HW, no replacement	Moderate-Low (Vendor APIs)	Low-Moderate	Enterprises, hybrid environments
Overlay SDN (Hypervisor)	Virtual networks (VXLAN, etc.)	High	No physical network changes	Moderate	Low	Cloud infrastructures
Hybrid SDN	Mixed control planes	Variable	Mixed legacy and SDN devices	Variable	High	ISP networks, gradual migration

Table 2. Numerical Comparison of SDN Implementation Methodologies

SDN Method	Average Latency (ms)	Throughput (Mbps)	Resource Savings	Scalability/Notes
OpenFlow-based SDN	16.8 – 20.5	680 – 900	Up to 68% lower delay than legacy net [10]	Scales well, but controller bottlenecks observed as topology grows [11]
API-based SDN	18 – 24 (REST API)	600 – 820	Moderate, depends on API load	Flexible integration, performance varies by controller and implementation [12]
Overlay SDN	22 – 31	500 – 780	Up to 85% increased simulation speed (with abstraction) [13]	High flexibility, but higher overhead due to encapsulation [13]
Hybrid SDN	15 – 19	750 – 920	89% fewer rules (vs. pure SDN with big-switch abstraction) [13]	Suitable for large, complex networks [13]

OpenFlow-based SDN significantly reduces transmission delays and improves throughput compared to traditional networks. Tests with four pods showed delays of 16.8–20.5 ms versus 49.7 ms in traditional setups, a reduction of about 68%, while throughput ranged from 680 to 900 Mbps depending on topology and flow table size. However, centralizing the control plane can increase CPU load at larger scales, potentially impacting performance. Different controllers vary in efficiency, with FloodLight offering higher throughput, OpenDayLight [14] providing lower latency and better scalability, and Ryu performing the least effectively.

Floodlight achieves higher throughput, especially in small-to-medium sized networks, due to its modular, streamlined Java-based architecture and relatively lightweight processing path. It offers efficient handling of OpenFlow messages and minimizes overhead through simple internal data structures and direct event processing. This simplicity accelerates packet processing when the number of switches is limited, allowing it to respond to flow requests with high speed and low management complexity. Notably, performance drops as network size or the number of switches increases [15].

OpenDaylight provides lower latency, particularly in medium and larger topologies or under high load, because of its highly modular, service-oriented microservices architecture and advanced event-handling strategy. OpenDaylight uses the Model-Driven Service Abstraction Layer (MD-SAL), allowing for asynchronous packet processing and efficient flow rule installation. Its support for parallelism and advanced cache management contributes to consistently low-latency performance—even as the network grows dense or complex. This makes it a preferred choice for environments where minimizing control plane delay is critical [16] [17].

Ryu, while praised for simplicity and prototyping flexibility, demonstrates the least effectiveness (in terms of throughput, scalability, and sometimes latency) for several reasons:

It is single-threaded and application-centric, designed for ease of customization over raw performance.
Its Python foundation—while developer-friendly—introduces interpretation, memory management, and threading limitations relative to Java (Floodlight) or enterprise-level platforms (OpenDaylight).
Ryu struggles to handle large numbers of concurrent packets or switches, resulting in a rapid drop in responses per second as the network scales.
As a result, for production-scale or high-load networks, Ryu is outperformed by both Floodlight and OpenDaylight in most key performance metrics [15] [18].
Despite some overhead, OpenFlow's programmability and dynamic traffic management offer greater flexibility and efficiency than traditional networks.
API-based SDN controllers driven by REST APIs typically exhibit latencies of 18–24 ms per request under normal loads, with delays increasing during traffic bursts. Their throughput ranges from 600 to 820 Mbps, heavily depending on the efficiency of the controller's API implementation. Additionally, each API call introduces a marginal processing overhead, especially when integrated with cloud or external orchestration platforms, impacting overall extensibility without a significant performance penalty.
Overlay SDN configurations, such as those using VXLAN tunneling, typically add 4–9 ms of latency per flow direction, leading to total delays of 22–31 ms in comparative studies. Throughput may decrease by 5–20% compared to direct-path SDN, resulting in sustained rates of 500–780 Mbps under load. Additionally, using the "big switch" abstraction method in simulations can reduce experiment execution time by 75%–85% compared to more detailed models.
Hybrid Software Defined Networking (SDN) integrates traditional distributed protocols with centralized SDN control, offering improved scalability, reduced latency (15–19 ms), and high throughput (750–920 Mbps) especially in large-scale networks by leveraging distributed control. This approach significantly cuts rules in forwarding devices—by up to 89% in simulations—thereby reducing controller load and accelerating experiment times by 75–85%. Hybrid SDN eases the costly full SDN deployment by mixing SDN and legacy devices, balancing benefits like flexible traffic engineering, quick failover, and optimized bandwidth while maintaining reliability through coexistence with protocols like OSPF. Various hybrid architectures allow tuning based on deployment scale and topology, enabling gradual migration to SDN with enhanced performance and manageability compared to pure SDN or traditional networks.
These values represent simulation results from published studies using platforms such as Mininet, NS-3, and large-scale abstracted networks, with outcomes varying depending on topology, hardware, and implementation. Hybrid SDN and model abstraction provide significant reductions in simulation time and required state, making them ideal for experimenting with very large networks. In contrast, API-based SDN performance tends to be more variable, so when incorporating orchestration or external integration, it is important to carefully profile the overheads associated with each request.

6 Emerging Trends and Research Challenges

Centralized SDN controllers face limitations with scale and failure resilience. To address these challenges, research is focused on approaches such as distributed, hierarchical, and multi-domain control planes. These methods aim to improve fault tolerance and reduce latency in SDN environments.
Northbound APIs remain fragmented across different SDN implementations. Establishing standard, portable APIs would enhance the portability of SDN applications and foster the development of a more robust ecosystem [19].
SDN presents several critical security challenges due to its architectural design, primarily the centralization of network control in the SDN controller, which becomes a high-value target [20]. The communication channels between the controller and network devices are vulnerable to man-in-the-middle attacks if not properly secured, while denial-of-service attacks can overwhelm these components, leading to service disruption. Insider threats, unauthorized access, and exploits via APIs further increase the attack surface. To address these concerns, encryption is fundamental: Transport Layer Security (TLS) should be employed for communications between the controller and both the switches (southbound) and applications (northbound) to prevent eavesdropping and tampering. End-to-end encryption at the application level can provide additional confidentiality for sensitive data flows, even if intermediate network segments are compromised. Symmetric encryption algorithms like AES are commonly used, with lightweight protocols gaining traction for resource-constrained scenarios such as IoT devices integrated into SDN [21].
In tandem with encryption, robust authentication and access control frameworks are essential for securing the SDN environment. Multi-factor authentication ensures that only authorized users and applications can access the controller and its APIs, while attribute-based encryption enables fine-grained, policy-driven access control that adapts to dynamic network conditions. Role-based access control limits privileges to mitigate insider threats, and federated identity management supports secure cross-domain collaboration. Securing the controller itself through OS hardening, minimizing attack surfaces, and protecting APIs with authentication tokens and input validation reduces vulnerabilities. Intrusion detection and prevention systems integrated with the controller can identify anomalies and mitigate threats like DoS attacks through behavioral analysis and rate limiting [22] [23]. Proper network design, including distributed controller deployments and network segmentation, enhances resilience against attacks. Finally, secure coding practices, regular software audits, and change management ensure that SDN applications do not introduce new vulnerabilities, making a comprehensive, multi-layered

security approach vital for safeguarding SDN infrastructures [24].

Recent studies highlight significant differences in security vulnerabilities between hybrid Software-Defined Networking (SDN) and pure OpenFlow setups, particularly regarding controller hijacking risks. The SDN controller, as a centralized management entity, remains a prime target for attacks in both architectures. Pure OpenFlow networks face vulnerabilities due to loosely connected OpenFlow switches and potentially compromised southbound channels; attackers can exploit these to impersonate switches or intercept control communications, especially when encryption like TLS is disabled for performance reasons [25]. Hybrid SDN, integrating traditional and SDN elements, introduces additional complexity and attack surfaces, including legacy links and devices, which may lack the same security mechanisms and can serve as entry points for attackers to manipulate flow control or launch denial-of-service attacks on the controller [26] [27]. Comparative analyses point out that hybrid SDNs, while beneficial for migration and flexibility, suffer from diverse inter-layer attack vectors such as man-in-the-middle attacks on control-to-switch communication and threats from unprotected applications in the control layer that can be manipulated to disrupt flows. Mitigation strategies emphasize robust authentication, secured control channels, and monitoring frameworks that detect anomalies in flow requests and topology changes—issues critical in both setups but more complex in hybrids due to their heterogeneous nature. Overall, while pure OpenFlow systems have more straightforward vulnerabilities primarily linked to controller-switch interactions, hybrid SDNs must address a broader, more complex spectrum of security risks owing to their mixed architecture [28].

Adapting SDN to support emerging technologies such as 5G networks, edge and fog computing, and Internet of Things (IoT) scenarios requires innovative solutions. These solutions need to address orchestration challenges, scalability issues, and strict latency constraints to fully leverage the potential of these new paradigms.

Machine learning-driven SDN controllers have emerged as a pivotal trend, especially with techniques like reinforcement learning being applied for dynamic flow routing. These AI-enhanced controllers enable networks to adapt traffic paths in real-time, optimizing performance and resource utilization. Additionally, AI-based anomaly detection using advanced models such as transformers is gaining prominence, allowing SDN systems to analyze traffic patterns in depth, detect irregularities, and proactively respond to potential threats or performance degradations. This AI and ML infusion facilitates predictive network behavior, self-optimization, and improved security, transforming SDN into a more intelligent and autonomous system [29] [30] [31].

In the context of 6G, SDN plays a transformative role in enabling network slicing, ultra-reliable low-latency communication (URLLC), and distributed edge control planes. As 6G networks aim for unprecedented speed, reliability, and flexibility, SDN's centralized and programmable control architecture becomes essential for dynamically managing diverse slices of the network tailored to specific application requirements, such as IoT, immersive multimedia, and autonomous systems. Moreover, SDN facilitates distributed control at the network edge, crucial for URLLC services that demand minimal latency and high availability. The integration of SDN with AI technologies in 6G also enables smarter orchestration, resource allocation, and adaptive network management, addressing the complex demands of next-generation wireless infrastructures [32] [33] [34] [35].

Green SDN is another emerging trend focused on sustainability and energy efficiency in network operations. Innovative SDN architectures are being designed to support dynamic switch power management, allowing SDN controllers to regulate and reduce the power consumption of switches and other network components based on real-time traffic conditions and demand. This not only lowers operational costs but also aligns with global efforts to reduce the carbon footprint of ICT infrastructures. Energy-aware SDN solutions contribute to ecological sustainability by enabling networks that intelligently balance performance with power efficiency, a growing research and industry focus in the face of increasing environmental concerns [36] [37].

7 Conclusion

This survey reviewed primary methodologies for implementing SDN, outlining OpenFlow-based, API-driven, overlay-based, and hybrid approaches. Each methodology presents unique scalability trade-offs, hardware dependence, and deployment complexity. Effective SDN adoption requires careful selection aligned with organizational requirements and infrastructure. Future challenges remain in controller scalability, API standardization, security, and integration with emerging technologies.

Disclosure of Potential Conflicts of Interest

The Authors declare that there is no conflict of interest

Reference

[1] H. F. Xavier and S. Seol, “A comparative study on control models of software-defined networking (SDN),” ces, vol. 7, pp. 1747–1753, 2014, doi: 10.12988/ces.2014.411234.

[2] N. McKeown et al., “OpenFlow,” SIGCOMM Comput. Commun. Rev., vol. 38, no. 2, pp. 69–74, Mar. 2008, doi: 10.1145/1355734.1355746.

[3] ONF. Open Networking Foundation. https://opennetworking.org/.

[4] Open Networking Foundation. (2014). SDN Architecture Overview. Whitepaper.

[5] D. Tsolkas et al., “Network & Service Management Advancements - Key frameworks and Interfaces towards open,

Intelligent and reliable 6G networks," Zenodo, Mar. 2025, doi: 10.5281/ZENODO.15011613.

[6] H. Elabd, J. Dingel, T. F. Lau, and A. Tizghadam, "Enhancing automated network function onboarding through language extension and code refactoring," Softw Syst Model, Aug. 2025, doi: 10.1007/s10270-025-01311-3.

[7] X. Huang, S. Cheng, K. Cao, P. Cong, T. Wei, and S. Hu, "A Survey of Deployment Solutions and Optimization Strategies for Hybrid SDN Networks," IEEE Commun. Surv. Tutorials, vol. 21, no. 2, pp. 1483–1507, 2019, doi: 10.1109/comst.2018.2871061.

[8] P. Vaid, S. K. Bhadu, and R. M. Vaid, "Intrusion detection system in Software defined Network using machine learning approach - Survey," 2021 6th International Conference on Communication and Electronics Systems (ICCES). IEEE, pp. 803–807, July 08, 20.

[9] Sandhya, Y. Sinha, and K. Haribabu, "A survey: Hybrid SDN," Journal of Network and Computer Applications, vol. 100, pp. 35–55, Dec. 2017, doi: 10.1016/j.jnca.2017.10.003.

[10] T. E. Ali, M. A. Abdala, and A. H. Morad, "SDN Implementation in Data Center Network," jcm, pp. 223–228, 2019, doi: 10.12720/jcm.14.3.223-228.

[11] M. Alsaeedi, M. M. Mohamad, and A. A. Al-Roubaiey, "Toward Adaptive and Scalable OpenFlow-SDN Flow Control: A Survey," IEEE Access, vol. 7, pp. 107346–107379, 2019, doi: 10.1109/access.2019.2932422.

[12] O. Tiamiyu, S. Onidare, H. Akande, O. Ajayi, and A. Ogbotobo, "Implementation and Comparison of Software-Defined Network Controllers in various Simulated Network Environments." Springer Science and Business Media LLC, May 14, 2024. doi: 10.21203/rs.3.

[13] J. Yan, X. Liu, and D. Jin, "Simulation of a Software-Defined Network as One Big Switch," Proceedings of the 2017 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation. ACM, pp. 149–159, May 16, 2017. doi: 10.1145/3064911.3064918.

[14] OpenDaylight Project. https://www.opendaylight.org.

[15] L. Mamushiane, A. Lysko, and S. Dlamini, "A comparative evaluation of the performance of popular SDN controllers," 2018 Wireless Days (WD). IEEE, pp. 54–59, Apr. 2018. doi: 10.1109/wd.2018.8361694.

[16] D. Hui, S. H. Wei, L. O. Bi, X. Zhu, and S.-K. Yoon, "Performance Evaluation of Ryu, OpenDayLight and Floodlight Controllers in Diverse Software-Defined Networking Topologies," Jour. of Adv. Res. Design, vol. 132, no. 1, pp. 103–114, May 2025, doi: 10.

[17] A. Mardaus, E. Biernacka, R. Wójcik, and J. Domżał, "Open Source SDN Controllers – Operational and Security Issues." MDPI AG, Apr. 30, 2024. doi: 10.20944/preprints202404.1984.v1.

[18] Montazerolghaem and S. Imanpour, "Evaluation and Performance Analysis of the Ryu Controller in Various Network Scenarios," 2025, arXiv. doi: 10.48550/ARXIV.2505.19290.

[19] Rathore, Vishal, and Mahak Jatav. "Advancements in Computer Networking: A Comprehensive Overview of Emerging Technologies, Protocols, and Trends", IJCTET, vol. 12, no. 2, pp. 416–420, Apr. 2024, Accessed: Sep. 09, 2025.

[20] A. Rahdari et al., "Security and Privacy Challenges in SDN-Enabled IoT Systems: Causes, Proposed Solutions, and Future Directions," CMC, vol. 80, no. 2, pp. 2511–2533, 2024, doi: 10.32604/cmc.2024.052994.

[21] Gupta, Parth Mukul. "Software-Defined Networking (SDN): Revolutionizing Network Infrastructure for the Future." Software-Defined Network Frameworks. CRC Press, 2024. 89-108.

[22] S. E. Vadakkethil Somanathan Pillai and K. Polimetla, "Mitigating DDoS Attacks using SDN-based Network Security Measures," 2024 International Conference on Integrated Circuits and Communication Systems (ICICACS). IEEE, pp. 1–7, Feb. 23, 2024. doi: 10.

[23] M. A. Setitra, M. Fan, I. Benkhaddra, and Z. E. A. Bensalem, "DoS/DDoS attacks in Software Defined Networks: Current situation, challenges and future directions," Computer Communications, vol. 222, pp. 77–96, June 2024, doi: 10.1016/j.comcom.2024.04.0.

[24] K. G. Yalda, D. J. Hamad, N. Tapus, and I. T. Okumus, "Security Issues in Software-Defined Networking (SDN) Environments," 2024 23rd RoEduNet Conference: Networking in Education and Research (RoEduNet). IEEE, pp. 1–8, Sept. 19, 2024. doi: 10.1109/roed.

[25] A. H. Janabi, T. Kanakis, and M. Johnson, "Survey: Intrusion Detection System in Software-Defined Networking," IEEE Access, vol. 12, pp. 164097–164120, 2024, doi: 10.1109/access.2024.3493384.

[26] J. Alotaibi, "A hybrid software-defined networking approach for enhancing IoT cybersecurity with deep learning and blockchain in smart cities," Peer-to-Peer Netw. Appl., vol. 18, no. 3, Mar. 2025, doi: 10.1007/s12083-025-01935-8.

[27] A. Rahdari et al., "Security and Privacy Challenges in SDN-Enabled IoT Systems: Causes, Proposed Solutions, and Future Directions," CMC, vol. 80, no. 2, pp. 2511–2533, 2024, doi: 10.32604/cmc.2024.052994.

[28] A. Rahdari et al., "Security and Privacy Challenges in SDN-Enabled IoT Systems: Causes, Proposed Solutions, and Future Directions," CMC, vol. 80, no. 2, pp. 2511–2533, 2024, doi: 10.32604/cmc.2024.052994.

[29] N. Mohamed, "Current trends in AI and ML for cybersecurity: A state-of-the-art survey," Cogent Engineering, vol. 10, no. 2, Oct. 2023, doi: 10.1080/23311916.2023.2272358.

[30] O. José Salcedo Parra, L. Correa Sánchez, and J. Gómez, "The Evolution of VANET: A Review of Emerging Trends in Artificial Intelligence and Software-Defined Networks," IEEE Access, vol. 13, pp. 49187–49213, 2025, doi: 10.1109/access.2025.3548640.

[31] D. Kalambe, D. Sharma, P. Kadam, and S. Surati, "A comprehensive plane-wise review of DDoS attacks in SDN: Leveraging detection and mitigation through machine learning and deep learning," Journal of Network and Computer Applications, vol. 235, p. 1040.

[32] Barsha Rani Das, Syed Rakib Hasan, Saifur Rahman Sabuj, Md Akbar Hossain, Sayan Kumar Ray, "A Comprehensive Survey on Emerging AI Technologies for 6G Communications: Research Direction, Trends, Challenges, and Opportunities," International Journal of Intelligent Networks, 2025.

[33] C. Yeh, Y.-S. Choi, Y.-J. Ko, and I.-G. Kim, "Standardization and technology trends of artificial intelligence for mobile systems," Computer Communications, vol. 213, pp. 169–178, Jan. 2024, doi: 10.1016/j.comcom.2023.11.004.

[34] O. M. S. Hassan and F. Keti, "A Review on the Challenges and Opportunities of Software Defined Networks Toward 5G and 6G," ejaset, vol. 3, no. 2, pp. 55–66, Mar. 2025, doi: 10.59324/ejaset.2025.3(2).05.

[35] Ali, Shabir, et al. "A roadmap to AI-assisted fog computing using SDN over 6G RAN." Intelligent Computing and Communication Techniques. CRC Press, 2025. 195-201.

[36] R. Singh, L. M. P. Larsen, E. Ollora Zaballa, M. S. Berger, C. Kloch, and L. Dittmann, “Enabling Green Cellular Networks: A Review and Proposal Leveraging Software-Defined Networking, Network Function Virtualization, and Cloud-Radio Access Network,” F.

[37] R. Singh, L. M. P. Larsen, E. Ollora Zaballa, M. S. Berger, C. Kloch, and L. Dittmann, “Enabling Green Cellular Networks: A Review and Proposal Leveraging Software-Defined Networking, Network Function Virtualization, and Cloud-Radio Access Network,” F.