**ISSN:** 2583-9888
**DOI:** doi.org/10.51219/JAIMLD/utpal-barman/684

URF PUBLISHERS
connect with research world

Journal of Artificial Intelligence, Machine Learning and Data Science

https://urfpublishers.com/journal/artificial-intelligence

Vol: 9 & Iss: 3

*Research Article*

# FAIDN: A Congestion-Aware Framework for Assured Intent-Driven Networking

**Utpal Barman[1]*, Dr. Kailash Joshi[2] and Dr. Charanjit Singh[3]**

[1]Manager, IT, Charter Communications, Missouri, USA

[2]Professor, University of St. Louis Missouri (UMSL), St. Louis, Missouri, USA

[3]Associate Professor, Chandigarh University, Chandigarh, India

**Citation:** Barman U, Joshi K, Singh C, FAIDN: A Congestion-Aware Framework for Assured Intent-Driven Networking. *J Artif Intell Mach Learn & Data Sci 2026* 9(3), 3447-3460. DOI: doi.org/10.51219/JAIMLD/utpal-barman/684

**Received:** 13 June, 2026; **Accepted:** 29 June, 2026; **Published:** 01 July, 2026

***Corresponding author:** Utpal Barman, Manager, IT, Charter Communications, Missouri, USA



## ABSTRACT

Intent-Driven Networking (IDN) aims to simplify network management by enabling operators to specify what the network should achieve rather than how it should be configured. However, practical deployments of IDN often fail to satisfy user intent under dynamic network conditions such as congestion, failures and heterogeneous traffic. In this paper, we redefine Assured Intent-Driven Networking in terms of four operational properties: correctness guarantees, continuous verification, convergence and graceful degradation - rather than as an idealized error-free system.

FAIDN (Framework for Assured Intent-Driven Networking) is a congestion-aware intent execution framework built on Software-Defined Networking (SDN) which integrates intent translation, real-time network state awareness, verification and automated remediation into a closed control loop. Using decision-tree–based diagnostics, FAIDN detects congestion, identifies root causes across control, data, link and host planes and applies corrective actions such as rerouting, queue resizing, traffic prioritization and rate limiting.

FAIDN is implemented using the Ryu SDN controller and the Mininet emulator and evaluated under a heterogeneous background traffic environment comprising video streaming, VoIP and bursty variable bit rate flows. Performance is assessed for a video conferencing intent as the primary evaluation target. Experimental results demonstrate that FAIDN achieves a 56× throughput improvement over baseline SDN-based execution at peak congestion, maintains video conferencing intent satisfaction above 80% at 90% load and converges within 2-4 remediation steps. Evaluation across additional intent types remains as future work.

**Keywords:** Intent-Driven Networking, Software-Defined Networking, Ryu SDN Controller, Mininet Emulator, Congestion Control, Intent Satisfaction, Network Remediation, Closed-loop Control

## 1. Introduction

Reliably satisfying operator-defined intents with provable correctness, continuous verification and predictable convergence under dynamic network conditions remains an open challenge in Intent-Driven Networking (IDN). We term this goal assured intent driven networking, defined operationally rather than as an idealized error-free system. While IDN promises to decouple what the network should achieve - such as latency bounds, bandwidth guarantees or traffic isolation - from how it is configured, practical deployments frequently fail under congestion, traffic heterogeneity and resource oversubscription[1-4]. Managing heterogeneous environments spanning Realtime multimedia, healthcare IoT, industrial automation and cloud applications using device-centric approaches has become prohibitively complex and error-prone, motivating the shift toward intent-driven paradigms[5,6].

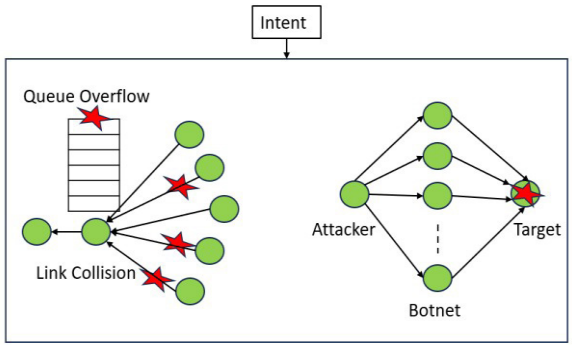


**Figure 1:** Assured intent implementation requires a Assured network: queue overflow, link collisions and Botnet attacks are bottlenecks for intent-driven networking.

Despite this promise, existing IDN systems consistently fail to satisfy user intent in dynamic environments[7]. As illustrated in **(Figure 1)**, network congestion is a primary cause, leading to increased latency, packet loss, jitter and application timeouts that directly constitute intent violations[8,9]. Congestion arises from dynamic traffic patterns, oversubscription, misconfigurations and failures[10] - effects that are especially damaging to delay sensitive applications such as video conferencing and VoIP, exposing a critical gap between intent specification and intent realization[11].

Existing congestion management approaches in SDN and self-driving networks focus on performance optimization, flow scheduling or traffic engineering[12-14], but lack explicit mechanisms for runtime verification, intent correctness and predictable behaviour under overload. ML-based self-driving networks introduce further challenges around stability, explainability and convergence, limiting their use in QoS-critical environments[15].

We argue that assured intent-driven networking does not require eliminating all failures - an unrealistic goal in distributed systems - but rather ensuring correct and predictable behavior under both normal and adverse conditions. Specifically, we define a assured IDN as one that provides: (1) correctness guarantees, (2) continuous verification, (3) convergence and (4) graceful degradation during intent execution. Based on these principles, this paper presents FAIDN, a congestion-aware IDN framework that treats congestion as a first-class intent violation, integrating detection, root-cause diagnosis, verification and automated remediation into a closed control loop.

The main contributions of this paper are as follows:

- We redefine assurance in IDN operationally, in terms of correctness, continuous verification, convergence and graceful degradation, aligning intent execution with assurance-based networking principles.
- We design FAIDN, a congestion-aware IDN architecture integrating intent translation, real-time state monitoring, runtime verification and automated remediation within a closed-loop control framework.
- We propose a decision-tree–based framework for congestion detection and root-cause analysis across the controller, data, link and host planes, enabling explainable and deterministic remediation.
- We implement and evaluate FAIDN on the Ryu SDN controller and Mininet under heterogeneous highload traffic, demonstrating improved intent satisfaction, predictable convergence and graceful degradation over baseline SDN-based execution.

## 2. Motivation

Efficient utilization of network resources while satisfying Quality of Service (QoS) requirements critically depends on the timely detection and resolution of network congestion. Congestion is a well-documented cause of performance degradation - leading to increased delay, packet loss, jitter and application timeouts - particularly for real-time and interactive services[8,9]. As networks increasingly support diverse and dynamic workloads, congestion remains a fundamental obstacle to reliable service delivery.

The challenge is compounded by the heterogeneity of modern traffic. Networks simultaneously carry elephant flows - large, long-duration transfers that consume sustained bandwidth - and mice flows - small, latency-sensitive, bursty transmissions typical of gaming, VoIP and web traffic. These competing flow types impose conflicting resource demands, making uniform congestion management strategies inherently insufficient.

Intent-Driven Networking (IDN) has emerged as a promising paradigm to address operational complexity by decoupling network control logic from low-level device configurations and enabling closed-loop orchestration for automated network management[1,2]. Such architectures leverage programmability and virtualization through Software-Defined Networking (SDN), Network Function Virtualization (NFV) and programmable data planes to dynamically configure and manage network resources[5,27]. Large-scale virtualization of heterogeneous network components is a key enabler of intent-driven operation.

Despite these advances, existing IDN and SDN-based solutions remain limited in their ability to resolve congestion in an automated and intent-aware manner. Many approaches provide only static or reactive configurations, which are insufficient to handle congestion across varying traffic conditions and network states[39-41]. Moreover, congestion mitigation is often treated as an isolated optimization problem rather than as an integral component of intent execution, resulting in limited automation and weak guarantees of sustained intent satisfaction.

Current IDN and SDN-based solutions are hampered by several interconnected limitations. Manual and error-prone configurations remain prevalent, particularly under

exceptional conditions such as network congestion or failures, which heightens the risk of misconfiguration and service disruption[27,55]. Compounding this, the static control logic embedded in many SDN controllers constrains their ability to adapt to rapidly changing traffic patterns and the heterogeneous demands of modern applications[55,56]. A further challenge is that intent execution is typically designed with the assumption of stable operating conditions, whereas real-world networks routinely experience overloads, dynamic traffic shifts, failures and security attacks[16,18]. Runtime verification mechanisms capable of ensuring that deployed configurations continue to satisfy user intents under such adverse conditions remain limited[15,25]. Finally, resource constrained IoT controllers often lack the programmability and abstraction necessary to support intent-driven operation across large-scale, heterogeneous IoT environments[43].

As network scale and heterogeneity continue to grow - driven by the proliferation of IoT devices, edge computing and latency-sensitive applications - the inability to reason about congestion dynamically leads to persistent intent violations[32,47]. These challenges collectively highlight the need for a congestion-aware, verifiable and adaptive IDN architecture that explicitly integrates congestion detection, root-cause analysis and automated remediation into the intent execution process.

Motivated by these observations, this work proposes a solution that systematically detects congestion, identifies its underlying causes across multiple network planes and applies targeted configurations that jointly address QoS requirements, interoperability and scalability constraints. By embedding these capabilities into a closed-loop intent execution framework, the proposed approach aims to improve intent satisfaction while ensuring predictable and reliable network behaviour.

## 3. Background and Related Work

This section reviews foundational technologies and prior research related to ML-based network analytics, programmable networking, self-driving networks, SDN and NFV, congestion control and intent-based networking. Each subsection identifies the limitations of existing approaches and positions FAIDN with respect to the current state of the art.

### 3.1. Background

Recent advances in ML-based network analytics have demonstrated strong potential for enabling autonomous and intelligent network operations, including monitoring, control and optimization of complex network systems[3,4]. In parallel, the evolution of network programmability has significantly enhanced the ability to remotely configure, monitor and manage network devices in a flexible and vendor-agnostic manner. In particular, the emergence of the protocol independent packet processing language P4 enables fine-grained programmability of forwarding devices, facilitating the design of highly programmable networks[5].

To support the design, development and prototyping of programmable and intent-driven architectures, simulation and emulation platforms play a crucial role. Network Simulator-3 (ns-3)[6] and the Mininet emulator[7] are widely adopted tools for evaluating networking schemes prior to physical testbed deployment, enabling controlled experimentation with new control logic, traffic patterns and configuration strategies under realistic conditions. The following subsections survey key solution approaches explored in the literature.

**3.1.1. Machine learning and analytics:** ML and data analytics techniques have been extensively studied for networking operations, including delay and bandwidth optimization[8], detection and mitigation of DoS attacks[9] and dynamic resource allocation[10]. In particular, reinforcement learning (RL) has proven effective in dynamic and partially observable network conditions, with applications in TCP congestion control[11,12], Quality of Experience (QoE) optimization[13], resource management[14] and runtime verification of P4-programmable devices[15].

Despite these advances, ML-driven solutions face persistent challenges related to learning instability, convergence guarantees, reproducibility and operational reliability - particularly in large-scale, endto-end deployments. These limitations motivate complementary approaches that emphasize deterministic behavior and verifiable correctness, as adopted in FAIDN.

Beyond software-based ML, the emergence of programmable data plane devices - including SmartNICs and P4-programmable switches - extends the reach of analytics to the forwarding plane, enabling per-packet telemetry collection at line rate without controller involvement[5]. FAIDN leverages this programmability indirectly through the Ryu controller's southbound OpenFlow interface, which provides the telemetry granularity required by the congestion detection module.

**3.1.2. Self-driving and self-organizing networks:** The concept of self-driving and self-organizing networks has gained significant attention[16-18]. These autonomous networks aim to monitor, manage, optimize, defend and analyze themselves with minimal human intervention. By combining analytics, control automation and closed-loop decision-making, selfdriving networks reduce manual operational overhead and improve efficiency.

However, such approaches often prioritize performance optimization and adaptability over formal correctness, verification and predictable convergence - properties that are critical for intent-driven and QoS-sensitive applications. This gap directly motivates the design principles underlying FAIDN.

**3.1.3. Intent Translation in SDN Controllers:** At the control-plane level, SDN controllers including NOX[19], DISCO[20], Floodlight[21], OpenDaylight[22], ONOS[23] and ONIX[24] have incorporated or are exploring intent specification and translation mechanisms. While these platforms provide essential building blocks for intent-driven networking, their support for runtime intent verification, congestion aware remediation and closed-loop assurance remains limited - motivating the need for enhanced architectures that integrate intent translation with continuous monitoring and adaptive control.

### 3.2. Related work

This section surveys existing work most relevant to FAIDN across four research areas: self-driving and autonomous networks, SDN and NFV infrastructures, congestion control mechanisms in SDN environments and intent-based networking systems. Each subsection summarizes representative contributions, identifies key limitations - particularly the absence of runtime verification, congestion-aware remediation and predictable convergence - and positions FAIDN with respect to the current state of the art.

**3.2.1. Self-driving networks (Self-DN):** Networks have increasingly evolved toward self-driving architectures that autonomously measure, manage and control operations with minimal human intervention[18]. These systems rely heavily on ML and data analytics for adaptive decision-making[16,25] and industry players such as Juniper Networks have actively promoted such solutions[26].

While self-driving networks demonstrate strong automation potential, ML-based approaches introduce challenges related to stability, explainability, repeatability and verification - particularly in dynamic and large-scale deployments[16,18]. Without sufficient programmability and structured control, enforcing business intent reliably remains difficult. In contrast, FAIDN adopts a rule-based, decision-tree– driven approach to enable deterministic behavior, explainable decision-making and predictable convergence under congestion.

**3.2.2. SDN and NFV:** Software-Defined Networking (SDN) and Network Function Virtualization (NFV) decouples the control and data planes, enabling centralized programmability and control. Extensive research has proposed SDN-based schemes to improve performance, scalability and flexibility[27-29], with applications spanning transportation[30,31], healthcare[32] and smart grid systems[33]. To address centralized scalability limitations, distributed SDN control planes have been proposed[34-36], though static switch-controller assignments often result in load imbalance and controlplane congestion[37,38].

Despite these advances, existing SDN and NFV solutions generally lack high-level intent abstraction and closedloop assurance - capabilities essential for sustainable and flexible network deployment and directly addressed by FAIDN.

**3.2.3. Congestion control mechanisms:** Extensive research has addressed congestion control in SDN environments. Representative solutions - including SD-TCP[39], SD-GCC[40] and Hedera[41] - dynamically schedule flows, adjust rates or reallocate bandwidth to mitigate congestion **(Table 1)**. Other approaches address flow identification, rate limiting and attack aware mitigation[42-44].

**Table 1:** Existing congestion-aware SDN approaches.

| Schemes | Reason for Congestion | Location of Congestion | React or Location | Detection | Configuration | Performance Evaluation |
|---|---|---|---|---|---|---|
| SDTCP[39] | Oversubscribing | Switch | Switch and Host | Queue Size | Adjusting the advertised TCP window | Mininet |
| SDGCC[40] | Oversubscribing | Switch | End Host | Queue Size | Divide network bandwidth across active VMs | Testbed |
| Hedera[41] | Elephant flows | Switch | Switch | Identify flows | Dynamic flow scheduling | Testbed |
| SCCP[42] | Large number of flows | Switch | Switch and Host | Flow counting at port | Limits the data rate of the TCP senders | Mininet, NS-3 |
| iAcceSD[43] | Unknown flows | Edge Switch | Switch and Host | ML- based flow identification | Dynamic flow and classifier | Mininet, NS-3 |
| MinRule[44] | Link flooding attacks (LFA) | Switch | Host, Switch Link | Detect flooding attack | Rerouting | NS-3 |

**(Table 1)** summarizes these existing congestion-aware SDN approaches.

While effective within specific scenarios, these mechanisms operate independently of intent semantics - treating congestion as a performance optimization problem rather than as a violation of high-level network intent. FAIDN explicitly bridges this gap by integrating congestion detection, diagnosis and remediation directly into the intent execution loop.

**3.2.4. Intent-Based Networking (IBN):** Recent research has explored IBN across wired and wireless networks, 5G infrastructures, IoT environments and cloud systems. Notable contributions include intent-based ONOS northbound interfaces for 5G service management[45], intent-driven mobile backhaul architectures[46], network slicing mechanisms[47] and automation frameworks for IoT requirements[48]. Industry initiatives from Cisco [49], Huawei[50], Gartner[51] and Juniper Apstra[52] further reflect the growing commercial adoption of intent-driven automation.

## 4. System Model and Assurance Properties

We define a assured intent-driven network operationally - not as an error-free ideal, but as one that maintains four verifiable properties during intent execution under dynamic and adverse conditions[1,2]. These properties collectively provide the behavioral guarantees necessary for practical IDN deployment and form the design basis for FAIDN **(Table 2)**.

**Table 2:** Properties for intent execution.

| Property | Definition | FAIDN Mechanism |
|---|---|---|
| Correctness | Intent semantics preserved during translation | Feasibility and consistency validation before deployment |
| Continuous Verification | Ongoing compliance monitoring at runtime | Telemetry-driven KPI evaluation against active intents |
| Convergence | Stable configuration reached after reconfiguration | Prioritized, sequential remediation strategies |
| Graceful Degradation | Controlled performance reduction under overload | Priority-aware intent scheduling |

**(Table 2)** summarizes each property and its corresponding mechanism in FAIDN.

In FAIDN, assurance is defined operationally rather than idealistically. Specifically, a assured intent-driven network is one that maintains correctness, continuous verification, convergence and graceful degradation during intent execution, particularly under dynamic and adverse network conditions. These properties collectively ensure predictable, verifiable and robust network behaviour,

which is essential for practical deployment of intent-driven and autonomous networking systems[1,2].

### 4.1. Correctness guarantees

Correctness guarantees ensure that the translation of high-level user intents into low-level network configurations preserves the intended semantics of user requirements. Prior studies have shown that ambiguities or inconsistencies during intent translation can lead to misconfigurations and unintended network behavior[25,45]. To address this, each intent in FAIDN is validated for feasibility, consistency and compatibility with existing intents before deployment. This validation step prevents conflicting or unsatisfiable configurations and establishes a sound baseline for reliable intent execution[18].

### 4.2. Continuous verification

Continuous verification is a fundamental requirement for maintaining intent satisfaction in dynamic network environments. FAIDN continuously monitors the operational network state using telemetry collected from controllers, switches, links and hosts. Key performance indicators-including queue occupancy, throughput, delay and packet loss-are evaluated against the constraints specified by active intents[15,53]. This runtime verification approach aligns with prior work on assurance-based and self-driving networks, which emphasizes the importance of ongoing validation rather than one-time configuration checks[16,18].

### 4.3. Convergence

Convergence is defined as the network reaching a stable, intent-compliant operating state within bounded time following a corrective reconfiguration event. Upon detecting deviations between the observed network state and the specified intents, FAIDN applies corrective actions aimed at restoring compliance while ensuring convergence to a stable network configuration. Uncoordinated or overly aggressive reconfigurations can lead to oscillations and instability in SDN-based systems[55,56]. To mitigate this risk, FAIDN employs prioritized and sequential remediation strategies, ensuring that corrective actions are applied incrementally and deterministically. This design choice promotes predictable convergence and avoids conflicting control decisions[34,36].

### 4.4. Graceful degradation

Graceful degradation is characterized by a monotonic, policy-governed reduction in performance as offered load exceeds available capacity, without abrupt collapse or oscillatory behaviour. Graceful degradation addresses scenarios in which network resources are insufficient to satisfy all active intents simultaneously. Instead of abrupt failures or indiscriminate performance collapse, FAIDN degrades network performance in a controlled and policy-aware manner. Lower-priority intents may experience reduced service levels, while higher-priority or mission-critical intents continue to receive preferential treatment[47,49]. Such controlled degradation is particularly important in multi-tenant, IoT and real-time application environments, where partial service continuity is preferable to complete disruption[32,43].

## 5. Problem Formulation

The existing literature has explored the evolution of multiple network architectures, including self-driving networks (Self-DN), Software-Defined Networking (SDN), distributed SDN and Intent-Based Networking (IBN)[16,18,27,34,45]. While these approaches address specific challenges related to automation, scalability and programmability, many limitations remain unresolved. Despite advances in SDN, NFV and intent-based networking - and broader efforts toward self-managed and self-optimized networks - a significant degree of manual configuration remains necessary during network design, deployment and operation[27,55]. Modern network devices such as gateways, routers and switches often run complex and distributed control software that is closed and proprietary[54]. Network administrators configure individual devices using vendor-specific interfaces that vary across products, protocols and standards, even within the same vendor ecosystem[54]. Such heterogeneity complicates network programmability and hinders consistent intent enforcement. Moreover, many state-of-the-art solutions focus on implementing intent over already designed and deployed networks, which limits their ability to adapt dynamically to evolving requirements and operating conditions[45,49]. As a result, these networks remain insufficiently intelligent and poorly suited to rapidly changing environments.

With the large-scale deployment of heterogeneous IoT and edge devices, local controllers play a critical role in network management and interfacing[32,43]. However, traditional IoT controllers often lack the programmability and abstraction required to support intent-driven operation. Since most intents are closely tied to application QoS and resource requirements - such as guaranteeing bandwidth for a scheduled video conferencing session or provisioning network slices for new IoT applications - a predictable and reliable network infrastructure is essential[47,48]. In this context, the ability to detect and resolve congestion dynamically becomes a key requirement for achieving reliable intent execution.

### 5.1. Congestion in intent-driven networks

Network congestion occurs when the offered traffic load exceeds the network's capacity to process and forward packets efficiently. Although congestion is often a transient condition, persistent congestion may indicate fundamental design, configuration or control issues[53]. Network congestion can be characterized by monitoring the normalized rate of queue accumulation across network nodes. If A(t) denotes the aggregate number of queued packets at time t, "I" is the measurement interval and "p" is the packet injection rate, the congestion index Γ is defined as[59]:

$$\Gamma = \lim_{t \to \infty} \frac{A(t+I) - A(t)}{Ip} \quad (1)$$

When Γ > 0, packets accumulate faster than they are drained, indicating congestion. When Γ ≤ 0, the network is draining the queue at a rate equal to or exceeding the injection rate. FAIDN triggers congestion remediation when Γ exceeds a configurable threshold Γ_th > 0 for a sustained interval T_detect, preventing transient spikes from causing spurious reconfigurations.

In the context of intent-driven networking, congestion is not merely a performance concern - it is a direct cause of intent violation. When congestion degrades throughput, increases latency or causes packet loss, active intents specifying QoS

bounds are violated. This work focuses on four dominant congestion causes that lead to intent violations: over-subscription, unknown or encrypted traffic flows, security attacks and poor network design or misconfiguration **(Table 3)**.

**Table 3:** Congestion Causes and FAIDN Responses.

| Congestion Cause | Origin | Affected Plane | FAIDN Response |
|---|---|---|---|
| Oversubscription | Traffic volume exceeds capacity | Controller, Switch, Link, Host | Queue resizing, rerouting, rate limiting |
| Unknown / Encrypted Flows | Dynamic ports, encryption | Data plane, Edge switch | ML-based flow identification, classifier update |
| Security Attacks | DoS, flooding, worms | Switch, Host, Link | Traffic isolation, rerouting, rate limiting |
| Poor Design / Misconfiguration | Vendor-specific config errors | Control plane, Data plane | Intent validation, configuration verification |

These factors, summarized in Table 3, motivate the need for an intent-driven framework that integrates intent translation, automated implementation and congestionaware remediation across multiple network planes.

**5.1.1. Over-subscription:** Over-subscription occurs when the network is required to handle traffic volumes beyond its designed capacity. In SDN environments, congestion due to oversubscription may arise at multiple entities, including controllers, switches, links and end hosts. SDN switches-particularly hardware switches relying on Ternary Content-Addressable Memory (TCAM)-are inherently resource-constrained, while software switches such as Open vSwitch rely on general-purpose memory and may not scale efficiently under high load[54].

The SDN control plane is also vulnerable to congestion and scalability limitations. A high rate of flow setup requests can overwhelm a centralized controller, creating a control-plane bottleneck[55]. Furthermore, excessive control traffic may lead to choke points near the controller, degrading overall network responsiveness[56]. Effective identification of traffic characteristics, flow durations and malicious activity is therefore essential to prevent over-utilization of network resources.

FAIDN addresses over-subscription through dynamic queue resizing, traffic rerouting and rate limiting, applied incrementally via its closed-loop remediation module.

**5.1.2. Unknown and encrypted traffic flows:** Modern networks carry highly heterogeneous traffic generated by diverse devices and applications that often use dynamically assigned ports or encryption. This significantly increases the complexity of traffic identification and classification[43]. Traditional SDN approaches rely on port-based classification or Deep Packet Inspection (DPI), both of which become increasingly ineffective and expensive in terms of latency and processing overhead as encrypted and variable traffic proliferates. These limitations further exacerbate congestion and hinder timely remediation.

FAIDN mitigates this challenge through ML-based flow identification and dynamic classifier updates, enabling timely remediation without relying on port-based inspection.

**5.1.3. Security attacks:** Security threats such as viruses, worms and Denial-of-Service (DoS) attacks can induce severe congestion in SDN-based networks. Compromised hosts or switches may be exploited to generate excessive traffic, host illicit services or content or launch flooding attacks, resulting in abnormal traffic surges and widespread service degradation[9,44]. Such attack-induced congestion directly violates application intents and necessitates integrated detection and mitigation mechanisms.

FAIDN responds to attack-induced congestion through traffic isolation and rerouting, integrated directly into the intent execution loop.

**5.1.4. Poor network design and misconfiguration:** Poor network design and misconfiguration remain persistent challenges in SDN deployments. As network devices are configured using heterogeneous and often vendor-specific interfaces, inconsistencies and errors are common[27,54]. Inadequate design may lead to broadcast or multicast storms, causing severe performance degradation. Additionally, faulty or misconfigured devices may fail to deliver agreed-upon data rates, further impacting QoS and intent satisfaction[55].

FAIDN addresses misconfiguration proactively through intent validation before deployment, preventing conflicting or unsatisfiable configurations from reaching the data plane.

Together, these four congestion causes represent the primary failure modes that FAIDN is designed to detect, diagnose and remediate. The following section describes the FAIDN architecture and the mechanisms through which these responses are implemented within a closed-loop intent execution framework.

## 6. Faidn Architecture

The proposed FAIDN architecture explicitly addresses network congestion as a primary factor affecting Quality of Service (QoS) and resource allocation during intent execution. Rather than relying on static or reactive configurations, FAIDN enables automated generation and deployment of congestion-aware configurations to ensure that user intents are satisfied under dynamic network conditions. **(Figure 2)** presents an overview of the proposed intent-driven SDN architecture.

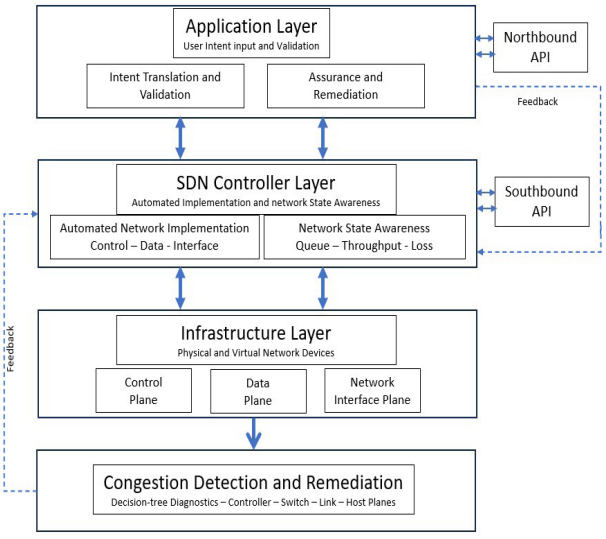


**Figure 2:** Intent-based SDN.

### 6.1. System model

The FAIDN architecture is designed to instantiate the four assurance properties defined in Section 4 - correctness, continuous verification, convergence and graceful degradation - within a practical SDN deployment across multiple domains including real-time multimedia, healthcare IoT, industrial automation and cloud applications among others. To achieve this, FAIDN follows a closed-loop control model composed of four functional modules, each responsible for a distinct phase of intent execution[16,18]. The architecture supports dynamic addition of new flows, provisioning of virtual networks and adaptive resource allocation within bounded time constraints[27,34].

**6.1.1. Intent translation and validation:** This module accepts high-level business or application intents specified by end users and translates them into candidate network configurations. During this process, intents are validated for feasibility, consistency and semantic correctness to prevent conflicting or unsatisfiable configurations[1,25,45]. The intent controller is designed to support multiple intent types, including QoS guarantees, traffic isolation and resource provisioning.

**6.1.2. Automated network implementation:** This module deploys validated configurations across the control, data and interface planes of the network. It is responsible for orchestrating configuration changes in response to new intents or evolving network conditions. Flexibility at the control plane includes adapting controller placement and switch-to-controller assignments, while data-plane flexibility involves dynamic adjustment of flows, queues and topology level resources[27,34,36].

**6.1.3. Network state awareness:** FAIDN continuously collects real-time telemetry from network components under its administrative control, including controllers, switches, links and hosts. This monitoring is protocol- and transport-agnostic, enabling comprehensive visibility into network performance indicators such as queue occupancy, throughput and packet loss[15,53].

**6.1.4. Assurance and Remediation:** This module continuously verifies whether the deployed configurations satisfy the original user intents. When violations are detected, FAIDN applies corrective actions-such as traffic blocking, queue resizing, rerouting or capacity adjustment-to restore compliance. This closed-loop assurance mechanism is essential for maintaining intent satisfaction under congestion and dynamic workloads[16,18,49].

As illustrated in **(Figure 2)**, these four modules form a closed loop: translated intents flow into the implementation module, which deploys configurations monitored by the state awareness module, whose telemetry feeds the assurance module, which triggers re-translation when violations are detected. Together, these modules ensure that intent execution remains continuously aligned with the observed network state, enabling predictable and reliable behavior.

### 6.2. FAIDN algorithm

Algorithm 1 provides an overview of the configuration selection and deployment process for intent execution in FAIDN. Given a user intent and optional supporting intents, the algorithm identifies the required configurations and applies them across the appropriate SDN planes. FAIDN invokes four plane-specific procedures: cpFAIDN (control plane), dpFAIDN (data plane), npFAIDN (network interface plane) and dlFAIDN (link plane).

**Algorithm 1:** Automating SDN for FAIDN

```
Input: intent, supportIntents (optional), Ct(f), Dt(f), Nt(f),
network state S
Output: deployed configurations

Accept intent from user
while supportIntents required do
    Accept supportIntent from user
end while
conf ← Translate(intent)      // intent translation module
Automate(conf, now)           // network automation module

procedure Automate(conf, now)
    for all conf in S do
        if conf applies to Ct(f) then
            cpFAIDN(Ct(f), now)
        else if conf applies to Dt(f) then
            dpFAIDN(Dt(f), now)
        else if conf applies to Nt(f) then
            npFAIDN(Nt(f), now)
        else
            dlFAIDN(Lt(f), now)
        end if
    end for
    S ← CollectNetworkState()
    VerifyIntent(intent, S)
end procedure
```

Algorithm 2 provides the closed-loop verification and remediation loop

Algorithm 2: FAIDN Closed-Loop Verification and Remediation

Input: intent, S (network state), remediationTree

Output: restored intent compliance or graceful degradation

```
loop
    S ← CollectNetworkState()          // poll every 500 ms
    violation ← VerifyIntent(intent, S)
    if violation = FALSE then
        break                          // intent satisfied; exit loop
    end if
    cause ← DiagnoseRoot(violation, S, remediationTree)
    action ← SelectAction(cause, remediationTree,
leastDisruptiveFirst)
    if action = NULL then
        GracefulDegrade(intent, priority)   // no actions remain
        break
    end if
    ApplyAction(action)
    steps ← steps + 1
    if steps > MAX_STEPS then
        GracefulDegrade(intent, priority)
        break
    end if
end loop
NotifyOperator(intent, steps, finalState)
```

### 6.3. Congestion detection and remediation

Building on the congestion taxonomy established in Section 3, this section describes the specific detection and remediation mechanisms FAIDN employs across each network plane. FAIDN models congestion as a multi-plane phenomenon that may originate at controllers, switches, links or hosts. Decision trees are used to detect congestion symptoms such as queue buildup and packet loss, identify root causes including over-subscription, unknown flows, attacks and misconfiguration and select appropriate remediation actions including rerouting flows, resizing queues, adjusting priorities, rate limiting senders and isolating malicious traffic. These actions are applied incrementally to ensure convergence[34,36].

**6.3.1 Decision tree for controller congestion (Figure 3):** When the controller exhibits high CPU utilization or packet processing delays exceeding threshold, FAIDN identifies control-plane congestion. **(Figure 3)** presents the master congestion detection tree, encompassing all four network planes. The controller branch evaluates flow request arrival rate, OpenFlow message, queue depth and controller load distribution. Remediation actions include redistributing switches to alternative controllers or throttling flow setup requests.

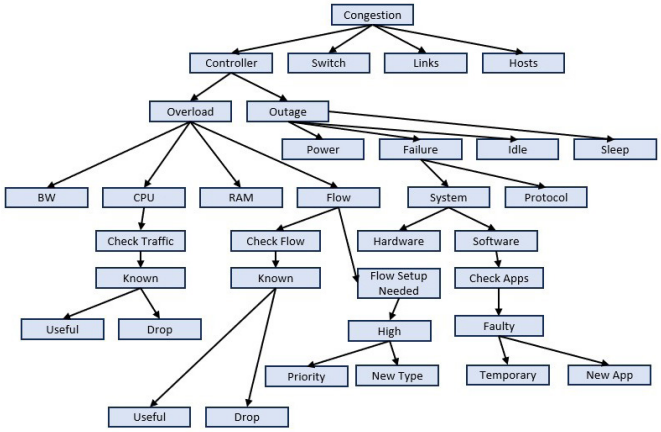


Figure 3: Decision tree for controller-plane congestion detection.

**6.3.2. Decision Tree for Switch Congestion (Figure 4 and Figure 5):** Switch-level congestion represents one of the most common bottlenecks in SDN deployments, arising from resource exhaustion at the data plane. FAIDN monitors three primary metrics at each switch: port utilization, queue depth and flow table occupancy. When any of these metrics exceeds a predefined threshold, the decision tree in **(Figure 4)** is invoked to confirm the presence of congestion and initiate root-cause analysis.

The detection tree evaluates port utilization first, as sustained high port utilization is the most direct indicator of data-plane overload. If port utilization is within acceptable bounds, queue depth is assessed - excessive queue buildup indicates that packets are being held longer than the intent's latency constraints permit. Flow table occupancy is evaluated last, as TCAM exhaustion in hardware switches can cause flow setup failures and silent packet drops, both of which constitute intent violations even when port-level metrics appear normal.

Once congestion is confirmed, the root-cause analysis tree in Fig. 5 classifies the underlying factor - whether the congestion is due to over-subscription, unknown or encrypted flows, security attacks or misconfiguration - and selects the corresponding remediation pathway. This two-stage design, separating detection from root cause classification, ensures that remediation actions are targeted rather than generic, reducing the risk of oscillatory reconfigurations.

**6.3.3. Decision tree for link congestion (Figure 6):** Link-level congestion differs from switch congestion in that it manifests between network nodes rather than within them, making it detectable only through interdevice telemetry rather than per-switch counters. FAIDN triggers link congestion detection under two primary conditions: when link utilization exceeds a configurable threshold $\tau\_l$ and when elephant flows are identified as occupying a disproportionate share of available link bandwidth.

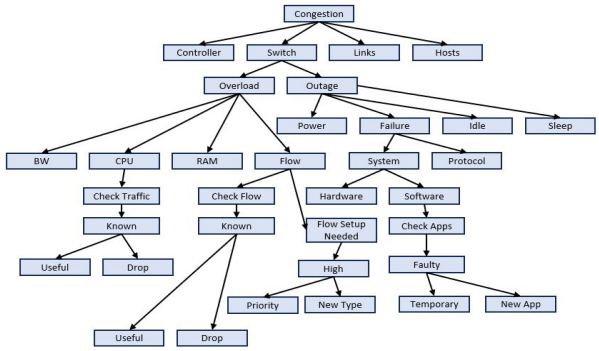


**Figure 4:** Decision tree for switch-plane congestion detection.

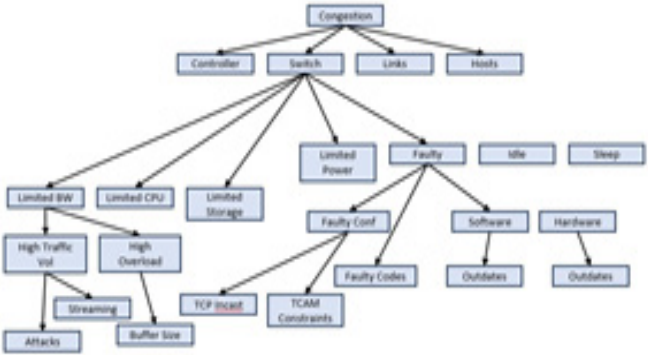


**Figure 5:** Decision tree for switch-plane root-cause analysis.

The decision tree in **(Figure 6)** first evaluates aggregate link utilization. If utilization exceeds $\tau\_l$, FAIDN determines whether the overload is caused by a small number of high-volume elephants flows or by a broader surge in traffic volume. This distinction is critical for remediation selection: elephant flow congestion is best addressed through dynamic flow scheduling or rerouting, while general overload may require rate limiting or queue prioritization across multiple flows.

If link utilization is within bounds but intent violations - such as increased latency or packet loss - are still observed, FAIDN inspects for elephant flows that may be crowding out latency-sensitive mice flows without triggering absolute utilization thresholds. This condition, where throughput-sensitive flows consume link capacity without triggering absolute utilization alarms, thereby degrading latency-sensitive flows - is sometimes termed covert congestion or hidden congestion[41] discusses elephant-mice flow interaction. It is particularly harmful to VoIP and video conferencing intents and requires prioritization-based remediation rather than rerouting. By explicitly distinguishing these two congestion modes, FAIDN avoids misapplied remediation that could worsen QoS for high-priority intents.

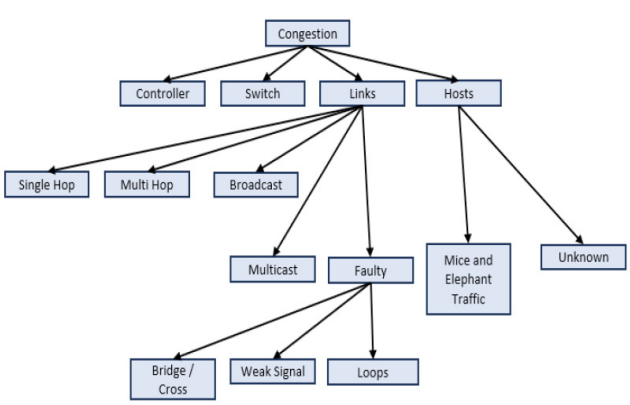


**Figure 6:** Decision tree for link-plane congestion detection.

**6.3.4. Decision tree for host congestion (Figure 3, Host Branch):** Host-plane congestion arises when end hosts-rather than network devices-are the limiting factor in intent satisfaction. FAIDN monitors two primary host-plane conditions. First, active host overload occurs when a host's CPU or memory utilization exceeds threshold τ_h, causing it to drop or delay packet processing even when network-side resources are available. This manifests as asymmetric throughput degradation localized to flows involving that host. Second, host idle or sleep states can result in silent flow drops that superficially resemble link failures.

The host branch of the decision tree in **(Figure 3)** first evaluates whether the host is active. If active, it assesses whether the constraint is bandwidth (BW), CPU or RAM. If the host is idle or in a sleep state, FAIDN issues a host wake-up or reachability alert rather than a network-side remediation action. When host CPU or RAM is the binding constraint, FAIDN applies rate limiting at the upstream switch to reduce the offered load to the host, preserving intent compliance for other flows while preventing the overloaded host from becoming a persistent bottleneck. Host-side remediation is always coordinated with the switch-plane module to ensure that flow table entries remain consistent with the host's reduced capacity.

**6.3.5. Remediation action selection (Figure 7)**: Once a congestion event is detected and its root cause identified, FAIDN applies remediation actions according to a least-disruptive-first principle. This ordering is central to FAIDN's convergence guarantee: by applying the most conservative corrective action first and escalating only if the violation persists, FAIDN avoids aggressive reconfigurations that could introduce oscillations or temporarily worsen performance for unaffected intents.

The remediation decision tree in **(Figure 7)** organizes available actions into an ordered hierarchy. Queue resizing is attempted first, as it adjusts buffer allocation without altering traffic paths or flow assignments - making it the lowest-impact intervention. If queue resizing is insufficient to restore intent compliance, traffic prioritization is applied next, reordering flow scheduling to favor high-priority intents while reducing service to lower-priority flows. If violations persist, flow rerouting is triggered, redirecting affected traffic over alternative paths with available capacity. In cases where congestion is attributed to security attacks or unknown flows, traffic isolation and rate limiting are applied to suppress the offending traffic before other actions are attempted.

After each remediation step, FAIDN re-evaluates the network state through the Network State Awareness module and re-checks intent compliance through the Assurance module. This iterative verification loop continues until either compliance is restored or all available remediation options have been exhausted, at which point graceful degradation is activated - reducing service levels for lower-priority intents while preserving compliance for mission-critical ones. This design ensures that remediation is both deterministic and auditable, supporting the explainability requirements of intent-driven operation[16,18].

Together, the decision trees in **(Figures 3-7)** form FAIDN's structured congestion management subsystem. By separating detection, root-cause analysis and remediation into distinct but interconnected trees, FAIDN ensures that corrective actions are targeted, deterministic and explainable - directly supporting the correctness and convergence properties defined in Section 4. Section 7 describes the implementation of this framework using the Ryu SDN controller and the Mininet emulator.

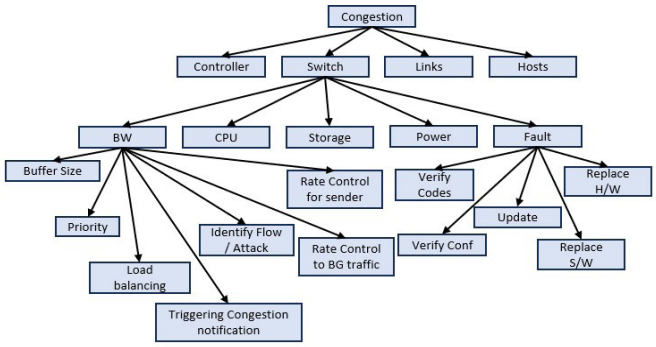


**Figure 7:** Decision tree for congestion remediation action selection.

## 7. Implementation and Performance Evaluation

This section describes the prototype implementation of FAIDN and presents a systematic performance evaluation under dynamic and congested network conditions. FAIDN is realized on the Ryu SDN controller, a widely adopted open-source platform for SDN research and evaluated using the Mininet network emulator over a configurable multi-switch topology with heterogeneous traffic workloads. The evaluation is designed to assess whether FAIDN instantiates the four assurance properties defined in Section 4 - correctness, continuous verification, convergence and graceful degradation - under conditions representative of real-world intent-driven deployments.

### 7.1. Evaluation objectives and metrics

The objective of the performance evaluation is to assess whether FAIDN satisfies the defining properties of assured intent-driven networking. Specifically, FAIDN is evaluated along four dimensions: intent satisfaction, measured by the network's ability to maintain QoS constraints as load increases[45,47]; throughput stability, measured as sustained application throughput as offered load approaches link capacity[39,41]; convergence behaviour, observed through the ability to reach a stable configuration following reconfiguration[34,55]; and graceful degradation, evaluated by controlled and predictable performance reduction under overload rather than abrupt collapse[16,18].

### 7.2. Experimental setup

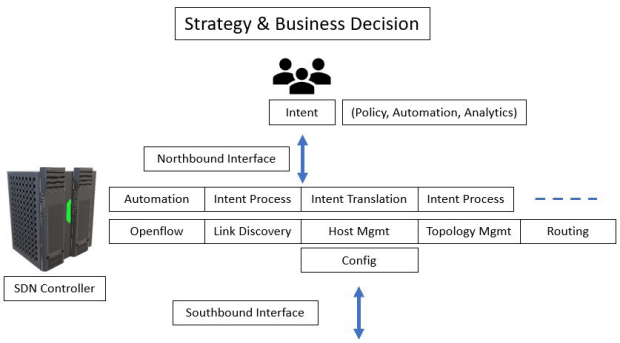


**Figure 8:** Ryu controller-based architecture for the proposed IDN.

The Ryu controller-based implementation architecture is illustrated in **(Figure 8)**, showing the northbound and southbound

interface components, intent processing modules and OpenFlow integration.

**7.2.1. Implementation platform:** FAIDN is implemented using the Ryu SDN controller and evaluated using the Mininet emulator, which is widely used for prototyping and validating SDN-based architectures[57,58]. The evaluation topology consists of multiple switches and hosts generating heterogeneous traffic patterns representative of real-world workloads, including video conferencing, video streaming, Voice over IP (VoIP), control traffic and bursty variable bit rate (VBR) flows.

These workloads are selected to emulate diverse application characteristics and QoS requirements, consistent with prior SDN and intent-based networking studies[39,43]. Each experiment is repeated five times under identical conditions. Results report the mean throughput across runs; observed variance was below 3% in all cases, confirming reproducibility.

As a baseline, we compare FAIDN against traditional SDN-based intent execution. In the baseline, the Ryu controller installs forwarding rules using reactive OpenFlow flow setup (packet-in / flow-mod) with default queue configuration (single FIFO queue, 75packet depth), no QoS queue differentiation, no DSCP or priority marking, no congestion monitoring and no automated remediation. Intents are translated to initial forwarding rules at deployment time but are not reverified or updated in response to congestion. This baseline represents the current practice in SDN deployments where intent translation is a one-time configuration step rather than a closed-loop process. Both FAIDN and the baseline operate on identical sixswitch Mininet topologies with identical traffic workloads, differing only in the presence or absence of FAIDN's monitoring, verification and remediation modules.

**7.2.2. Network topology**

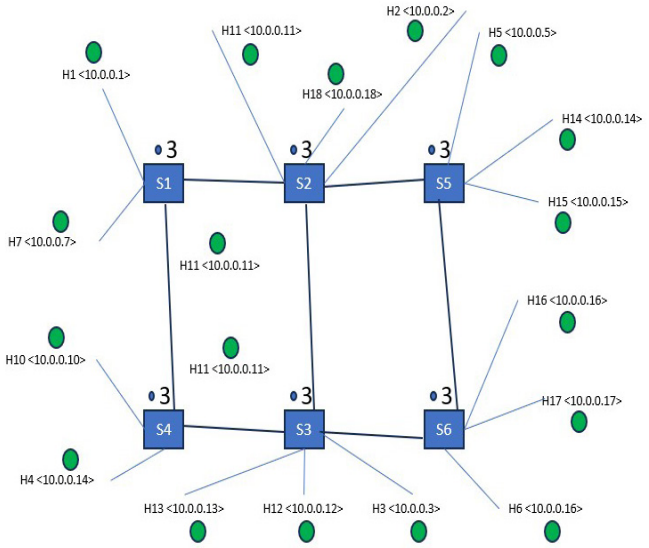


**Figure 9:** Mininet Topology.

The evaluation topology is designed to support dynamic reconfiguration and congestion remediation across heterogeneous traffic flows. Congestion scenarios are emulated using Mininet[57], which enables controlled experimentation under reproducible conditions. The overall implementation architecture and evaluation topology are illustrated in **(Figure 9)**. Each inter-switch link is configured with a bandwidth of 10 Mbps and a propagation delay of 5 ms. Host-to-switch links operate at 10 Mbps. OpenFlow switch buffers are configured with a default queue depth of 75 packets. This is derived as follows: at 10 Mbps with 1000-byte video conferencing packets, each packet incurs a serialization delay of (1000 × 8) / 10^7 = 0.8 ms. With a 5 ms propagation delay per link, the remaining queuing budget to satisfy the 60 ms end-to-end latency intent is 55 ms, permitting a maximum of floor (55 / 0.8) = 68 packets. A default of 75 packets provides headroom for smaller-packet flows (e.g., VoIP at 60 bytes, ICMP at 64 bytes) while remaining within the constraint. During active remediation, FAIDN adjusts queue depth to 40 packets for latency-sensitive flows (VoIP, video conferencing) and up to 250 packets for throughputpriority flows (video streaming, file transfer).

The evaluation topology consists of six switches (S1-S6) and 18 hosts interconnected as illustrated in Fig. 9. Heterogeneous traffic is generated as follows: all hosts generate background ICMP traffic to randomly selected peers; Hosts 1 and 6 conduct a video conferencing session using iperf (tool to measure network bandwidth and performance); Hosts 4 and 5 stream video using VLC media player; Hosts 2 and 11 and Hosts 3 and 18, establish two VoIP connections using iperf; and Hosts 7 and 17 and Hosts 8 and 16, generate bursty VBR flows using D-ITG (Distributed Internet Traffic Generator). Loop handling among switches is managed by the Ryu STP module (ryu.app.simple_switch_stp). FAIDN automates three classes of configuration in the evaluation: queue resizing, flow prioritization and load balancing across available paths.

**7.2.3. Traffic generation:** To generate heterogeneous traffic workloads, multiple hosts in the topology employ a range of traffic generation tools, including VLC, iperf, Ping and D-ITG. These tools are used to emulate application-specific traffic such as video streaming, video conferencing, VoIP, control packets and bursty flows **(Table 4)**.

The corresponding traffic characteristics, including data rates, packet sizes and QoS objectives, are summarized in **(Table 4)**.

**7.3. Results**

This section presents experimental results evaluating FAIDN along the four dimensions defined in Section 5. Results are compared against the baseline SDN configuration defined in Section 7.2.1.

**Table 4:** Traffic generation parameters for evaluation workloads.

| Tools | Application | Traffic Type | Data Rate | Packet Size | Packet Interval | QoS |
|---|---|---|---|---|---|---|
| VLC | Video Streaming | MPEG | 32-600 Kbps | 128 - 1518 Bytes | 60 ms(Max) | Throughput |
| iperf | Video Conferencing | CBR | 500 Kbps | 1000 Bytes | 60 ms (Max) | Throughput, Delay, Jitter |
| iperf | VoIP | CBR | 64 Kbps | 60 Bytes | 60 ms (Max) | Throughput, Delay, Jitter |
| Ping | - | CBR | 0.5 Kbps | 64 Bytes | - | Delay |

| | | | | | | |
|---|---|---|---|---|---|---|
| D-ITG† | VBR | VBR | Dynamic VBR | Dynamic VBR | Dynamic VBR | Dynamic VBR |
| D-ITG† | File Transfer | VBR | Dynamic VBR | Dynamic VBR | Dynamic VBR | Dynamic VBR |

†D-ITG parameters vary dynamically to simulate bursty traffic; no fixed rate, packet size or interval is preconfigured. QoS objectives: throughput and delay (VBR), throughput (File Transfer).

**7.3.1. Intent satisfaction under increasing load:** To evaluate intent satisfaction, we gradually increase the offered load for latency- and throughput-sensitive applications while monitoring achieved throughput and compliance with QoS constraints. As traffic load increases, baseline SDN-based intent execution exhibits significant throughput degradation and frequent intent violations due to unmanaged congestion.

**Table 4:** Traffic generation parameters for evaluation workloads.

| Tools | Application | Traffic Type | Data Rate | Packet Size | Packet Interval | QoS |
|---|---|---|---|---|---|---|
| VLC | Video Streaming | MPEG | 32-600 Kbps | 128 - 1518 Bytes | 60 ms(Max) | Throughput |
| iperf | Video Conferencing | CBR | 500 Kbps | 1000 Bytes | 60 ms (Max) | Throughput, Delay, Jitter |
| iperf | VoIP | CBR | 64 Kbps | 60 Bytes | 60 ms (Max) | Throughput, Delay, Jitter |
| Ping | - | CBR | 0.5 Kbps | 64 Bytes | - | Delay |
| D-ITG† | VBR | VBR | Dynamic VBR | Dynamic VBR | Dynamic VBR | Dynamic VBR |
| D-ITG† | File Transfer | VBR | Dynamic VBR | Dynamic VBR | Dynamic VBR | Dynamic VBR |

†D-ITG parameters vary dynamically to simulate bursty traffic; no fixed rate, packet size or interval is preconfigured. QoS objectives: throughput and delay (VBR), throughput (File Transfer).

**Table 5:** Nominal bandwidth allocation across concurrent traffic classes (10 Mbps link).

| Traffic Class | Allocation | Tool | Protocol |
|---|---|---|---|
| Video Conferencing | 5.0 Mbps | iperf | UDP / CBR |
| Video Streaming | 2.0 Mbps | VLC | MPEG |
| VoIP (x2 sessions) | 0.128 Mbps | iperf | UDP / CBR |
| Bursty VBR (x2) | 2.0 Mbps | D-ITG | VBR |
| ICMP Background | 0.001 Mbps | Ping | ICMP |

**Table 6:** Throughput of the video conferencing intent flow (iperf, UDP, 1000byte packets) under increasing aggregate offered load.

| Loads (%) | FAIDN (Mbps) | Intent in SDN (Mbps) |
|---|---|---|
| 10 | 7.83 | 7.76 |
| 30 | 6.56 | 3.89 |
| 50 | 5.87 | 1.06 |
| 70 | 5.43 | 0.91 |
| 90 | 5.05 | 0.09 |

† Link bandwidth: 10 Mbps shared across all traffic classes **(see Table 5)**. FAIDN configurations applied: queue resizing, flow prioritization and rerouting.

**Table 7:** Intent satisfaction rate comparison between FAIDN and baseline SDN for a video conferencing intent.

| Loads (%) | FAIDN Intent Satisfaction Rate (%) | Baseline Intent Satisfaction Rate (%) |
|---|---|---|
| 10 | 100 | 98 |
| 30 | 96 | 61 |
| 50 | 91 | 18 |
| 70 | 87 | 14 |
| 90 | 82 | 1 |

In contrast, FAIDN dynamically applies congestion remediation actions-including traffic rerouting, queue resizing and flow prioritization-in response to observed violations. As a result, intents remain satisfied over a wider range of operating conditions. **(Table 6)** presents quantitative throughput results and Table 7 presents the corresponding intent satisfaction rate - defined as the percentage of time the active intent's QoS constraints are met - across the same load levels. The results demonstrate that FAIDN sustains higher effective throughput for intent-related traffic, particularly under moderate to high load, indicating improved intent satisfaction compared to the baseline[39,41].

**Table 8:** Latency comparison between FAIDN and baseline SDN for a video conferencing intent.

| Loads (%) | FAIDN Latency (ms) | Baseline latency (ms) |
|---|---|---|
| 10 | 12 | 13 |
| 30 | 18 | 45 |
| 50 | 24 | 180 |
| 70 | 31 | 390 |
| 90 | 38 | >=820 † |

† Latency exceeded 820 ms at 90% load; exact measurement unavailable due to frequent packet drops.

**(Tables 5-7)** report throughput and latency for the video conferencing intent flow only (iperf CBR, 1000-byte packets, UDP), which operates alongside the other classes listed above. The video conferencing flow competes for bandwidth under increasing offered load; FAIDN's remediation actions (queue resizing, prioritization, rerouting) selectively protect this highpriority intent.

**(Table 8)** presents the corresponding end-to-end latency, further illustrating the contrast between FAIDN's controlled behavior and the baseline's collapse under increasing load.

As illustrated iu=n Table 9, FAIDN also preserves VoIP intent performance under increasing load. While baseline SDN allows VoIP jitter to rise above 38 ms at 50% load - approaching the 60 ms bound - FAIDN's priority-aware queue management maintains jitter below 10 ms across all evaluated load levels. This confirms that FAIDN's 'covert congestion' detection (Section 6.3.3) successfully identifies and remediates elephant-flow interference with latency-sensitive mice flows.

**(Table 6)** presents throughput results for a video conferencing intent under increasing load. At low load (10%), both FAIDN

and baseline SDN achieve comparable throughput (7.83 Mbps vs. 7.76 Mbps), confirming that FAIDN introduces negligible overhead under non-congested conditions. As load increases to 30%, the baseline degrades sharply to 3.89 Mbps - a 50% reduction - while FAIDN maintains 6.56 Mbps through queue resizing and flow prioritization. At 50% load and beyond, the baseline collapses to near-zero throughput (0.09 Mbps at 90% load), indicating persistent intent violations. FAIDN, by contrast, sustains 5.05 Mbps at 90% load — a 56× improvement over the baseline at peak congestion - demonstrating robust intent satisfaction through congestion-aware remediation. As shown in Table 7, FAIDN maintains an intent satisfaction rate above 80% even at 90% load, while the baseline drops to 1%, confirming robust intent correctness under congestion. FAIDN's monitoring module polls network state every 500 ms, detecting intent violations within one polling cycle (≤500 ms) in all evaluated scenarios, confirming the continuous verification property defined in Section 3.

**Table 9:** VoIP intent performance (jitter and packet loss) - FAIDN vs. baseline SDN.

| Loads (%) | FAIDN Jitter (ms) | Baseline Jitter (ms) | FAIDN Loss (%) | Baseline Loss (%) |
|---|---|---|---|---|
| 10 | 2.1 | 2.3 | 0.0 | 0.1 |
| 30 | 3.8 | 12.4 | 0.1 | 4.2 |
| 50 | 5.2 | 38.1 | 0.3 | 14.7 |
| 70 | 7.1 | 89.3 | 0.8 | 31.2 |
| 90 | 9.4 | > 150 (estimated) | 1.4 | 58.6 |

† Latency exceeded 820 ms at 90% load; exact measurement unavailable due to frequent packet drops.

### 7.3.2. Convergence and stability

**Table 10:** FAIDN convergence behavior across evaluated congestion scenarios.

| Congestion Event | Remediation Steps | Time to Stable State (s) | Baseline Oscillations |
|---|---|---|---|
| Over-subscription (30% load) | 2 | 1.4 | 7 † |
| Over-subscription (70% load) | 3 | 2.1 | 12 † |
| Unknown flow injection | 4 | 2.8 | 9 † |

† Baseline did not converge within the observation window; oscillation count reflects the number of conflicting reconfigurations observed.

Upon detecting congestion, FAIDN applies corrective actions sequentially using decision-tree–guided remediation strategies. This structured approach prevents conflicting reconfigurations and mitigates oscillatory behavior, which is commonly observed in reactive SDN control systems[55,56]. As shown in **(Table 10)**, FAIDN converged to a stable configuration within 2-4 reconfiguration steps and approximately 1.4-2.8 seconds across all evaluated congestion scenarios, compared to persistent oscillation observed in the baseline. These observations confirm that FAIDN achieves predictable convergence, an essential requirement for operationally reliable intent-driven networks[34,36].

**7.3.3. Graceful degradation under overload:** As shown in **(Table 6)**, as network load approaches and exceeds link capacity, FAIDN throughput declines monotonically from 7.83 Mbps at 10% load to 5.05 Mbps at 90% load - a controlled 35% reduction - while the baseline collapses from 7.76 Mbps to 0.09 Mbps, an uncontrolled 99% reduction. Rather than abrupt collapse, FAIDN degrades gracefully while preserving preferential treatment for higher-priority intents, ensuring partial service continuity under resource-constrained conditions[16,18,47]. As network load increases from 10% to 90%, FAIDN exhibits graceful degradation across all key performance dimensions. Throughput declines gradually from 7.83 to 5.05 Mbps **(Table 6)**, the intent satisfaction rate decreases moderately from 100% to 82% **(Table 7)** and latency grows in a controlled manner from 12 ms to 38 ms **(Table 8)**. Taken together, these trends collectively confirm that FAIDN upholds the graceful degradation property formally defined in Section 3.

The evaluation results collectively confirm that FAIDN successfully instantiates the four assurance properties under realistic heterogeneous traffic conditions. The current evaluation focuses on a single intent type (video conferencing) and a fixed six-switch topology; future work will extend the evaluation to multi-intent scenarios, larger topologies and additional congestion causes including security attacks and misconfiguration. Section 8 discusses broader implications and limitations of the proposed approach.

## 8. Discussion and Limitations

While FAIDN demonstrates significant improvement over baseline SDN-based intent execution, several aspects warrant further discussion. First, the decision tree–based diagnostic approach provides determinism and explainability but may require manual updates as new congestion patterns emerge - a limitation that ML based approaches partially address at the cost of stability and convergence guarantees. Second, the current evaluation is conducted on a six-switch Mininet topology with a single link bandwidth of 10 Mbps; scaling to larger, multi-domain topologies with diverse link capacities remains an open question. Third, the current implementation and evaluation address oversubscription, unknown flows and misconfiguration induced congestion; security attack-induced congestion is modelled in the decision trees (Section 6.3) but not yet exercised in the evaluation. Extending the implementation to cover all four congestion causes identified in Section 5.1 represents the most immediate direction for future work. Finally, formal verification of FAIDN's convergence guarantee - that the network always reaches a stable state within bounded steps - remains an open theoretical challenge.

Compared to the congestion control approaches surveyed in **(Table 1)**, FAIDN addresses a dimension none of them cover: intent semantics. SD-TCP[39] and SDGCC[40] manage congestion at the transport level without awareness of application intents; Hedera[41] optimizes flow scheduling but provides no runtime verification. FAIDN's 56× throughput advantage over unmanaged SDN at 90% load, combined with an intent satisfaction rate above 80%, demonstrates that embedding remediation directly into the intent execution loop provides substantially better QoS preservation than isolated congestion management.

**7.3.4. Multi-intent priority and graceful degradation:** As illustrated in **(Table 11)**, to validate FAIDN's priorityaware graceful degradation under resource contention, we configure two simultaneous primary intents: a video conferencing intent

(H1↔H6, priority level 2) and a VoIP intent (H2↔H11, priority level 1, higher priority). The network is loaded to 90% capacity with background traffic, creating a scenario where both intents cannot be simultaneously satisfied at full QoS.

**Table 11:** Multi-intent satisfaction under 90% load.

| Intent | FAIDN Satisfaction (%) | Baseline Satisfaction (%) |
|---|---|---|
| VoIP (Priotity 1) | 91 | 3 |
| Video Conference (Priority 2) | 74 | 1 |

FAIDN preserves the higher-priority VoIP intent at 91% satisfaction while gracefully reducing the video conferencing intent to 74% - a controlled trade-off governed by the priority scheduler. The baseline provides near-zero satisfaction for both intents, confirming that without explicit priority-aware remediation, all intents fail equally under overload.

## 9. Conclusion

This paper presented FAIDN, a congestion-aware framework for assured intent-driven networking, where assurance is defined operationally in terms of four properties: correctness guarantees, continuous verification, convergence and graceful degradation. Unlike existing SDN and intent-based networking solutions that treat congestion as an isolated performance problem, FAIDN integrates congestion detection, root-cause diagnosis and automated remediation directly into the intent execution loop through a structured decision-tree–based approach. Implemented on the Ryu SDN controller and evaluated using the Mininet emulator under heterogeneous highload traffic, FAIDN demonstrates a 56× throughput improvement over baseline SDN-based intent execution at peak congestion, converges to stable configurations within 2-4 remediation steps and degrades gracefully under overload conditions. These results demonstrate that intent-driven networks can be made predictable and resilient through careful system design rather than idealized assumptions. Future work will extend FAIDN to multi-intent scenarios, larger topologies and formal convergence verification.

## 10. Acknowledgment

The authors express sincere gratitude to colleagues and reviewers for their valuable feedback and guidance throughout this research.

### 10.1. Funding statement

This research did not receive any specific grant from funding agencies in the public, commercial or not-for-profit sectors.

### 10.2. Conflict of interest statement

The authors declare no conflict of interest related to the research, authorship or publication of this manuscript.

## 11. References

1. Cohen R, Barabash K, Rochwerger B, et al. An intent-based Approach for Network Virtualization. IFIP/IEEE International Symposium on Integrated Network Management, 2013: 42-50.
2. Pang L, Yang C, Chen D, et al. A Survey on Intent-Driven Networks. IEEE Access, 2020;8: 22862-22873.
3. Wang M, Cui Y, Wang X, et al. Machine Learning for Networking: Workflow, Advances and Opportunities. Ieee Network, 2017;32: 92-99.
4. Bhattacharyya DK, Kalita JK. Network Anomaly Detection: A Machine learning Perspective. Chapman and Hall/CRC, 2013.
5. Bosshart P, Daly D, Gibb G, et al. P4: Programming Protocol-independent Packet Processors," ACM SIGCOMM Computer Communication Review, 2014;44: 87-95.
6. https://www.nsnam.org/overview/what-is-ns-3/
7. De Oliveira RLS, Schweitzer CM, Shinoda AA, et al. Using Mininet for Emulation and Prototyping software-defined networks. Colombian Conference on Communications and Computing (COLCOM), 2014: 1-6.
8. Madanapalli SC, Gharakheili HH, Sivaraman V. Assisting Delay and Bandwidth Sensitive Applications in a Self-Driving Network. ACM SIGCOMM Workshop on Self-Driving Networks (SelfDN), 2018: 22-27.
9. Demoulin HM, Pedisich I, Phan LTX, et al. Automated Detection and Mitigation of Application-Level Asymmetric DoS At tacks. Proceedings of the Afternoon Workshop on Self-Driving Networks, SelfDN, 2018: 36-42.
10. Singh SK, Jukan A. Machine-learning-based Prediction for Resource (re)allocation in optical data center networks. IEEE/OSA Journal of Optical Communications and Networking, 2018;10.
11. Winstein K, Balakrishnan H. TCP ex machina: Computer generated congestion control. ACM SIGCOMM Computer Communication Review, 2013;43: 123-134.
12. Dong M, Li Q, Zarchy D, et al. {PCC}: Re-architecting Congestion Control for Consistent High Performance," in 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI'15), 2015: 395-408.
13. Jiang J, Sun S, Sekar V, et al. Pytheas: Enabling Data-driven Quality of Experience Optimization using group-based Exploration Exploitation. 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI'17), 2017: 393-406.
14. Mao H, Alizadeh M, Menache I, et al. Resource Management with Deep Reinforcement learning. Proceedings of the 15th ACMWorkshop on Hot Topics in Networks, 2016: 50-56.
15. Shukla A, Hudemann KN, Hecker A, et al. Runtime Verification of P4 Switches with Reinforcement Learning. Proceedings of the 2019 Workshop on Network Meets AI & ML, 2019: 1-7.
16. Yaqoob T, Usama M, Qadir J, et al. On Analyzing Self Driving Networks: A Systems Thinking Approach. Proceedings of the Afternoon Workshop on Self-Driving Networks, 2018: 1-7.
17. https://www.juniper.net/us/en/dm/infographic/how-a-self-driving-network-works/
18. Kalmbach P, Zerwas J, Babarczi P, et al. Empowering Self-driving Networks. Proceedings of the Afternoon Workshop on Self-Driving Networks, 2018: 8-14.
19. Gude N, Koponen T, Pettit J, et al. NOX: Towards an Operating System for Networks. ACM SIGCOMM Computer Communication Review, 2008;38: 105-110.
20. Phemius K, Bouet M, Leguay J. DISCO: Distributed Multi domain SDN Controllers. IEEE Network Operations and Management Symposium (NOMS), 2014: 1-4.
21. http://www.projectfloodlight.org/floodlight
22. http://www.opendaylight.org
23. Berde P, Gerola M, Hart J, et al. ONOS: Towards an Open, Distributed SDN OS. Proceedings of the third workshop on Hot topics in software defined networking, 2014: 1-6.
24. Koponen T, Casado M, Gude N, et al. ONIX: A Distributed Control Platform for Large-scale Production Networks. OSDI, 2010;10: 1-6.

25. Jacobs AS, Pfitscher RJ, Ferreira RA, et al. Refining Network Intents for Self-driving Networks. Proceedings of the Afternoon Workshop on Self-Driving Networks, 2018: 15-21.
26. https://www.juniper.net/us/en/dm/the-self-driving-network/
27. Kim H, Feamster N. Improving Network Management with Software defined Networking. IEEE Communications Magazine, 2013;51: 114-119.
28. Lin S, Wang P, Akyildiz IF, et al. Towards Optimal Network Planning for Software-defined Networks. IEEE Transactions on Mobile Computing, 2018;17: 2953-2967.
29. Nam J, Jo H, Kim Y, et al. Operator defined Reconfigurable Network OS for Software-defined Networks," IEEE/ACM Transactions on Networking, 2019;27: 1206-1219.
30. Liu K, Feng L, Dai P, et al. Coding Assisted Broadcast Scheduling via Memetic Computing in SDN-based Vehicular Networks. IEEE Transactions on Intelligent Transportation Systems, 2018;19: 2420-2431.
31. Weng J, Zhang Y, Luo W, et al. Benbi: Scalable and Dynamic Access Control on the Northbound interface of SDNbased VANET. IEEE Transactions on Vehicular Technology, 2019;68: 822-831.
32. Wang B, Sun Y, Yuan C, et al. Lesla: A Smart Solution for SDN-enabled MMTC e-health Monitoring System. Proceedings of the 8th ACM MobiHoc 2018 Workshop on Pervasive Wireless Healthcare Workshop, 2018: 1-6.
33. Akkaya K, Uluagac AS, Aydeger A. Software defined Networking for wireless local networks in smart grid," in the IEEE 40th Local Computer Networks Conference Workshops (LCN Workshops), 2015: 826-831.
34. Dixit A, Hao F, Mukherjee S, et al. Towards an Elastic Distributed SDN controller. SIGCOMM Compute. Communication. Rev, 2013;43: 7-12.
35. Schiff L, Schmid S, Kuznetsov P. In-band Synchronization for Distributed SDN Control Planes. SIGCOMM Compute. Communication Rev, 2016;46: 37-43.
36. Bannour F, Souihi S, Mellouk A. Distributed SDN control: Survey, taxonomy and challenges. IEEE Communications Surveys Tutorials, 2018;20: 333-354.
37. Chahlaoui F, El-Fenni MR, Dahmouni H. Performance Analysis of Load Balancing Mechanisms in SDN Networks. Proceedings of the 2nd International Conference on Networking, Information Systems Security, NISS19, (New York, NY, USA), Association for Computing Machinery, 2019.
38. Cui J, Lu Q, Zhong H, et al. A Load-balancing Mechanism for Distributed SDN Control Plane using Response time. IEEE Transactions on Network and Service Management, 2018;15: 1197-1206.
39. Lu Y, Ling Z, Zhu S, et al. SDTCP: Towards Data center TCP Congestion Control with SDN for IoT applications. Sensors, 2017;17: 109.
40. Abdelmoniem AM, Bensaou B. SDN-based Generic Congestion Control Mechanism for Data centers: Implementation and Evaluation. Dept Computer Science Eng, 2016.
41. Al-Fares M, Radhakrishnan S, Raghavan B, et al. Hedera: Dynamic Flow Scheduling for Data Center Networks. Nsdi, 2010;10: 89-92.
42. Hwang J, Yoo J, Lee SH, et al. Scalable Congestion Control Protocol based on SDN in data centre networks. 2015 IEEE Global Communications Conference (GLOBECOM), 2015: 1-6.
43. Ahmed N, Misra S. Collaborative Flow-identification Mechanism for Software-defined Internet of Things. IEEE Internet of Things Journal, 2021: 1.
44. Biswas R, Wu J. Minimizing the Number of Rules to Mitigate Link Congestion in SDN-based Datacenters. Proc. of 15th International Conference on Networking, Architecture and Storage, 2021.
45. Addad RA, Dutra DLC, Bagaa M, et al. Benchmarking the ONOS Intent Interfaces to Ease 5G Ser vice Management. 2018 IEEE Global Communications Conference (GLOBECOM), 2018: 1-6.
46. Subramanya T, Riggio R, Rasheed T. Intent-based Mobile Back Hauling for 5G Networks. 2016 12th International Conference on Network and Service Management (CNSM), 2016: 348-352.
47. Aklamanu F, Randriamasy S, Renault E, et al. Intent-Based Real-Time 5G Cloud Service Provisioning," in 2018 IEEE Globecom Workshops (GC Wkshps), 2018: 1-6.
48. Nagendra V, Bhattacharya A, Yegneswaran V, et al. An Intent-based Automation Framework for Securing Dynamic consumer IoT infrastructures. Proceedings of The Web Conference, 2020: 1625-1636.
49. https://www.cisco.com/c/en_in/solutions/intent-based-networking.html
50. https://e.huawei.com/in/products/network-management-and-analysis-software
51. https://blogs.gartner.com/andrew-lerner/2017/02/07/intent-based-networking/
52. https://www.juniper.net/documentation/us/en/software/apstra/apstra4.0.0/
53. Mills K, Dabrowski C, et al. The need for realism when simulating network congestion. SpringSim (CNS), 2016: 1.
54. Nguyen X-N, Saucez D, Barakat C, et al. Rules Placement Problem in OpenFlow Networks: A survey," IEEE Communications Surveys & Tutorials, 2015;18: 1273-1286.
55. Voellmy A, Wang J. Scalable Software defined Network controllers. Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures and protocols for computer communication, 2012: 289-290.
56. Macapuna CAB, Rothenberg CE, Maurício MF. In-packet Bloom Filter-based Data center Networking with distributed OpenFlow Controllers. 2010 IEEE Globecom Workshops, 2010: 584-588.
57. Kaur K, Singh J, Ghumman NS. Mininet as Software defined Networking Testing Platform. International Conference on Communication, Computing & Systems (ICCCS), 2014: 139-142.
58. Asadollahi S, Goswami B, Sameer M. Ryu Controller's Scalability Experiment on Software defined Networks. 2018 IEEE international conference on current trends in advanced computing (ICCTAC), 2018: 1-5.
59. Kleinrock L. Queueing Systems, Volume 1: Theory. New York: Wiley-Interscience, 1975.