Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

A Quantum Stochastic Walk Approach to Dynamic Multipath Routing under Correlated Cryptographic Degradation

Chinmay Patwardhan

chinmay.patwardhan@mitwpu.edu.in

MIT World Peace University

Shubhalaxmi Joshi

MIT World Peace University

Article

Keywords: Quantum StochasticWalk, Post-Quantum Cryptography, Software-Defined Networking, GKLS Master Equation, Multipath Routing, Correlated Failure, Kyber, SPHINCS+, McEliece, OpenFlow, P4, Non-Homogeneous Poisson Process, Control Plane Latency

Posted Date: April 10th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-9320492/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

A Quantum Stochastic Walk Approach to Dynamic Multipath Routing under Correlated Cryptographic Degradation

Chinmay Patwardhan, Shubhalaxmi Joshi
Department of Computer Science and Applications
Dr. Vishwanath Karad MIT World Peace University
Pune, India
chinmay.patwardhan@mitwpu.edu.in, shubhalaxmi.joshi@mitwpu.edu.in

***Abstract*—Classical multipath routing protocols, including Equal-Cost Multi-Path (ECMP) and Shortest Path First (SPF), optimize for physical link metrics and are mathematically incapable of representing correlated algorithmic failure across cryptographic primitives. As financial-grade network architectures migrate to Post-Quantum Cryptography (PQC) under NIST mandate, this representational gap introduces a systemic risk: a zero-day compromise of a lattice-based algorithm (e.g., Module Learning With Errors, Module-LWE) leaves classical load balancers routing traffic into cryptographically compromised paths while all physical links remain active. This paper presents a Quantum Stochastic Walk (QSW) control plane operating over a Software-Defined Network (SDN), designed to detect and respond to correlated cryptographic degradation events. The architecture encodes heterogeneous PQC overhead Kyber (lattice, low latency), SPHINCS+ (hash, medium latency), and McEliece (code, high latency) into a utility vector $\mathbf{u}$, and captures shared failure probability via a dual-covariance matrix $\boldsymbol{\Sigma}$ that decomposes network-layer and cryptographic-layer correlations with equal weight. A discretized approximation of the Gorini-Kossakowski-Lindblad-Sudarshan (GKLS) master equation evolves a density matrix ρ over $N = 30$ logical routing paths, blending coherent quantum channel dynamics with a classical Google-matrix baseline at mixing parameter $\omega = 0.2$. The coherent update $(V\rho_t V^\dagger)$ and stochastic update together carry an $\mathcal{O}(N^3)$ per-iteration complexity, tractable for physical port densities up to $N = 128$. Upon simulation of a lattice-compromise shock event, the QSW engine redistributed traffic away from the compromised cluster reducing lattice-path utilization from 43.7% to 25.3% and increasing hash-based and code-based path utilization to 40.4% and 34.2%, respectively while maintaining 100% packet reachability throughout the transition. GPU/CPU convergence was achieved in 8.32 ms under the shock state. A 50-stream Non-Homogeneous Poisson Process (NHPP) traffic load with micro-burst injection confirmed correct multipath saturation at 188 Mbit/s aggregate throughput. The Python-based OpenFlow update latency of $t_{\text{flow_mod}} \approx 42$ ms is identified as the binding deployment constraint, arising from OS-level user/kernel-space serialization; the required production path is migration to a P4 programmable data plane with native C++ or SmartNIC execution of the $\mathcal{O}(N^3)$ tensor operations.**

***Index Terms*—Quantum Stochastic Walk, Post-Quantum Cryptography, Software-Defined Networking, GKLS Master Equation, Multipath Routing, Correlated Failure, Kyber, SPHINCS+, McEliece, OpenFlow, P4, Non-Homogeneous Poisson Process, Control Plane Latency**

Corresponding author: Chinmay Patwardhan (chinmay.patwardhan@mitwpu.edu.in)

I. Introduction

HIGH-THROUGHPUT financial transaction networks, such as the Indian Unified Payments Interface (UPI), impose strict sub-10 ms end-to-end latency bounds across their critical processing paths. The ongoing standardization and mandatory migration to Post-Quantum Cryptography, driven by NIST's selection of lattice-based, hash-based, and code-based algorithms [1], introduces heterogeneous computational overheads that are non-uniformly distributed across network paths. Kyber (Module-LWE) carries low encapsulation latency; SPHINCS+ (hash-based) carries medium latency; McEliece (code-based) carries high latency [2]. When all three algorithm classes are co-deployed across a shared SDN fabric, the result is a multi-class overhead distribution that classical routing protocols cannot model. The fundamental limitation of ECMP and SPF in this context is not computational but representational. These protocols operate on scalar link metrics bandwidth, hop count, or delay and apply hash-based flow assignment across equal-cost paths [4]. No state encoding the cryptographic algorithm class of each path is maintained, and consequently the correlation structure arising when multiple paths share a common mathematical vulnerability is invisible to the control plane. If a zero-day exploit targets the Module-LWE hardness assumption underlying Kyber, all paths in the lattice-based cluster are simultaneously compromised. ECMP continues distributing traffic across those paths because the physical links remain active and their scalar metrics are unchanged: the network is cryptographically compromised while appearing operationally normal to the routing layer. To close this representational gap, the present work proposes a QSW control plane operating over an SDN fabric. The architecture exploits the GKLS master equation [11], [12] to mix coherent (quantum) correlation detection with incoherent (classical) stochastic routing. The coherent channel, derived from a Hamiltonian encoding the dual-covariance structure of the network, propagates utility shocks simultaneously across all correlated paths via quantum interference. The incoherent (Google-matrix) channel regularizes the resulting probability distribution against classical utility constraints. The system therefore operates as a hybrid quantum-classical oscillator: sensitive to cryptographic correlation at the control plane, and

grounded to physical link state at the data plane. This paper makes the following contributions:

1) A formal definition of the utility vector $\mathbf{u}$ and dual-covariance matrix $\mathbf{\Sigma}$ that jointly encode PQC overhead and cross-path failure correlation in a form consumable by a QSW Hamiltonian.
2) A discretized GKLS evolution rule for the routing density matrix ρ that blends coherent channel dynamics with a Google-matrix classical baseline, parameterized by a single mixing scalar $\omega = 0.2$, with a formal complexity analysis establishing $\mathcal{O}(N^3)$ per-iteration cost.
3) A synthetic traffic generation model based on a Non-Homogeneous Poisson Process (NHPP) with micro-burst injection, structured to force full utilization of the probability weight vector $\mathbf{w}$ via 5-tuple hashing in OVS select group tables.
4) Empirical validation on a 30-path emulation topology using Open vSwitch (OVS) and OpenFlow 1.3 select group tables, demonstrating correct traffic evacuation from a simulated lattice-compromise event with zero packet loss and 188 Mbit/s aggregate multipath throughput.
5) A quantitative analysis of control plane latency boundaries, identifying the Python user/kernel-space serialization overhead as the binding deployment constraint and specifying the P4/C++ implementation path required for production deployment.

The remainder of this paper is organized as follows. Section II surveys related work across three bodies of literature. Section III formalizes the mathematical architecture of the QSW control plane. Section IV describes the emulation environment and methodology. Section V presents empirical results and latency boundary analysis. Section VI concludes and identifies directions for future work.

II. Related Work

A. Post-Quantum Cryptography Performance in Networked Environments

The performance characteristics of NIST-selected PQC algorithms in networked and embedded contexts have been established through a series of systematic benchmarks. Bürstinghaus-Steinbach et al. [2] integrated Kyber and SPHINCS+ into the mbed TLS stack on embedded hardware and measured end-to-end TLS handshake latency under both algorithms. Their results established that Kyber-based key establishment performs comparably to elliptic-curve Diffie-Hellman on constrained devices, while SPHINCS+ introduces substantial signature-size and signing-time overheads that affect server-side throughput. These measurements constitute the empirical baseline for the utility hierarchy encoded in $\mathbf{u}$ in the present work.

Buruaga et al. [3] extended this analysis to SDN environments, demonstrating a deployable hybrid TLS architecture for SDN control planes and quantifying the added latency and processing overhead of PQC integration in SDN TLS flows. Both studies treat PQC overhead as a static, per-algorithm scalar. Neither work models the dynamic case in which a cryptographic algorithm is compromised mid-operation, nor does either work address the routing consequences of correlated algorithmic failure across multiple paths sharing the same PQC primitive.

B. Multipath Routing and Correlated Failure Mitigation

Classical multipath routing research has addressed physical-layer failure correlation through two principal mechanisms: adaptive probing and per-packet spraying. Liu et al. [6] proposed Partial Congestion-aware Load Balancing (PLB), which reduces ECMP-induced imbalance by probing a dynamically selected partial set of paths and rerouting flows away from congested segments. Bonato et al. [5] proposed REPS (Recycling Entropies for Packet Spraying), an entropy-recycling per-packet spraying approach that adaptively reroutes around congested or failing links with minimal control-plane state. Both PLB and REPS improve resilience to physical-layer failures and congestion.

Neither protocol, however, encodes cryptographic or application-layer correlation. PLB's probing mechanism detects congestion on active links; it has no signal path for algorithmic compromise of a cryptographic primitive. REPS's entropy recycling operates on 5-tuple hash diversity; it cannot distinguish between a physically degraded path and a cryptographically compromised path that remains physically active. The representational gap identified in Section I is therefore not addressable by extension of either framework: it requires a fundamentally different state representation at the control plane, which the dual-covariance matrix $\mathbf{\Sigma}$ and the QSW density matrix ρ provide.

C. Quantum Walks in Network Optimization

Quantum walks have been applied to network optimization problems through two lines of research. Bapat et al. [7] demonstrated that quantum routing protocols can implement arbitrary permutations of qubits under nearest-neighbor constraints with provably sub-swap-based routing times, establishing the speedup properties of quantum routing on path and grid topologies. Wang et al. [9] formalized routing algebra for quantum networks and proposed optimal routing with fairness-aware resource allocation heuristics tailored to probabilistic link behaviors. Most directly relevant to the present work, Chang et al. [10] applied QSWs to financial networks via the GKLS master equation, encoding asset covariance into a Hamiltonian to drive probability redistribution across correlated financial instruments. The present paper adapts the Chang et al. architecture to the network routing domain, translating asset covariance into the dual-covariance routing matrix $\mathbf{\Sigma}$ and replacing financial utility with the PQC-overhead-derived utility vector $\mathbf{u}$. The key architectural translation is the substitution of $\Sigma_{ij}^{\text{crypto}}$ for the financial correlation block, which maps shared cryptographic vulnerability rather than shared market exposure.

III. Mathematical Architecture

A. Topology and Path Classification

The routing fabric is modeled as a set of $N = 30$ logical paths, indexed $i \in \{0, 1, \ldots, 29\}$. Each path is assigned to one of three PQC algorithm classes:

- **Lattice class** ($i \in [0, 9]$): Paths secured by Kyber (Module-LWE). These paths exhibit the lowest encapsulation overhead and highest base utility.
- **Hash class** ($i \in [10, 19]$): Paths secured by SPHINCS+ (hash-based signatures). These paths exhibit medium encapsulation overhead and intermediate utility.
- **Code class** ($i \in [20, 29]$): Paths secured by McEliece (code-based KEM). These paths exhibit the highest encapsulation overhead and lowest base utility.

This partitioning reflects the well-characterized performance hierarchy of PQC algorithm families in networked environments [2], [3]. The logical path index i serves as the quantum system's basis state; the density matrix element ρ_{ii} represents the probability that a given packet flow is assigned to path i.

B. Utility Vector

The utility u_i of path i is defined as the composite inverse of its PQC encapsulation overhead τ_i^{pqc} and its base network latency τ_i^{net}:

$$u_i = \frac{1}{\tau_i^{\text{pqc}} + \tau_i^{\text{net}}} \tag{1}$$

The utility vector $\mathbf{u} \in \mathbb{R}^N$ aggregates all path utilities. Under nominal operating conditions, lattice-class paths ($i \in [0, 9]$) attain $u_i \approx 0.8$–1.0 due to the low computational cost of Kyber key encapsulation. Under a simulated algorithmic compromise, the utility of all lattice-class paths is set to $u_i = 0.01$, reflecting the effective disqualification of those paths from trusted routing. The utility vector is the sole input channel through which external threat intelligence feeds into the QSW engine; a real-time PQC monitoring subsystem would update $\mathbf{u}$ continuously as overhead measurements are collected.

C. Dual-Covariance Matrix

Classical network covariance matrices encode only physical-layer correlations (e.g., shared Autonomous System Numbers, shared physical fiber segments). The dual-covariance matrix $\mathbf{\Sigma} \in \mathbb{R}^{N \times N}$ introduced here decomposes the total path correlation into two orthogonal components:

$$\Sigma_{ij} = \frac{1}{2}\,\Sigma_{ij}^{\text{net}} + \frac{1}{2}\,\Sigma_{ij}^{\text{crypto}} \tag{2}$$

where Σ_{ij}^{net} captures shared network infrastructure (paths i and j traversing the same Autonomous System are assigned $\Sigma_{ij}^{\text{net}} = 1$, otherwise 0), and $\Sigma_{ij}^{\text{crypto}}$ captures shared cryptographic vulnerability: paths belonging to the same PQC algorithm class are assigned $\Sigma_{ij}^{\text{crypto}} = 1.0$; paths belonging to different classes are assigned $\Sigma_{ij}^{\text{crypto}} = 0$. This formulation ensures that a compromise event targeting one PQC class propagates its correlation signal uniformly across all paths in that class, a property structurally absent from all scalar link-metric routing frameworks [5]. The equal-weight decomposition (0.5/0.5) reflects the assumption that network-layer and cryptographic-layer failure risks contribute equally to total path correlation; this weight may be tuned by operators with empirical failure-rate data. The complete architecture of the QSW control plane mapping utility and covariance into the GKLS evolution block is illustrated in Figure 1.

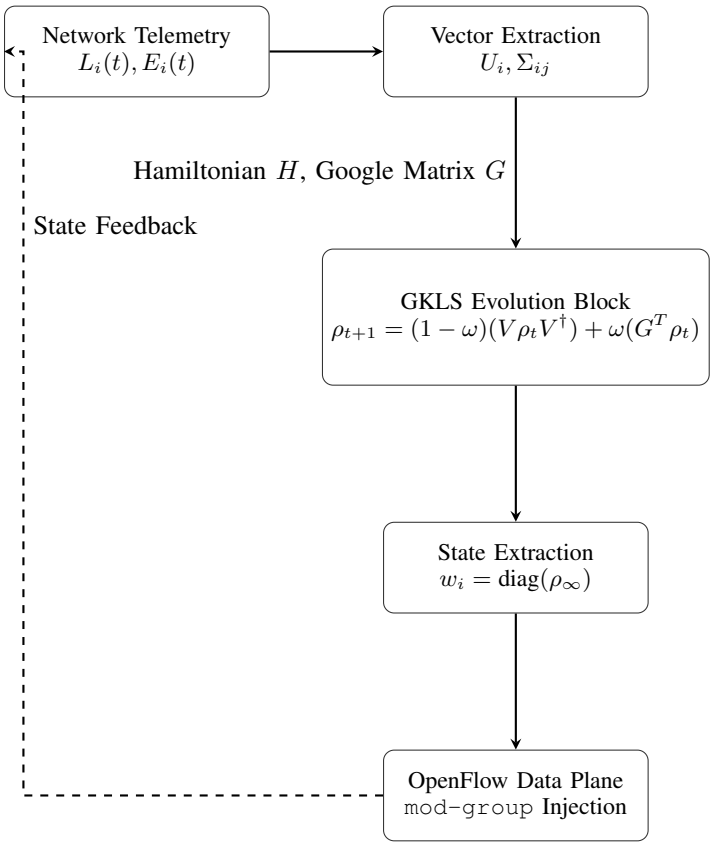


Fig. 1. Architecture of the QSW Control Plane mapping utility and covariance into GKLS evolution, terminating at the SDN data plane.

D. QSW Hamiltonian Construction

The QSW Hamiltonian $H \in \mathbb{C}^{N \times N}$ is constructed from the dual-covariance matrix and the utility vector. Off-diagonal elements encode the coupling strength between paths:

$$H_{ij} = \begin{cases} -\Sigma_{ij} & i \neq j \\ u_i & i = j \end{cases} \tag{3}$$

The diagonal encodes the self-energy of each path (its utility), while the off-diagonal encodes the coherent tunneling amplitude between correlated paths. The time-evolution operator for a timestep Δt is the unitary matrix:

$$V = e^{-i\Delta t H} \tag{4}$$

computed via matrix exponentiation (`scipy.linalg.expm`). The coherent channel V maps the correlation structure of the network into the quantum amplitude domain, enabling the density matrix to simultaneously represent superpositions of routing states across correlated path clusters.

E. Google-Matrix Classical Baseline

To prevent the density matrix from evolving into an unphysical superposition state disconnected from classical routing constraints, a classical stochastic baseline is enforced via the Google transition matrix $G \in \mathbb{R}^{N\times N}$. The matrix G is constructed as the column-stochastic normalization of the utility vector:

$$G_{ij} = \frac{u_j}{\sum_{k=1}^{N} u_k} \tag{5}$$

This construction ensures that the classical baseline allocates probability mass proportional to path utility, independent of quantum coherence effects. The transposed matrix G^T acts on the diagonal of the density matrix in the evolution rule, consistent with the classical Markov chain interpretation [15].

F. GKLS Density Matrix Evolution

The Gorini-Kossakowski-Lindblad-Sudarshan (GKLS) equation [11], [12] describes the Markovian evolution of the density matrix of an open quantum system. In its continuous form, the GKLS generator decomposes the evolution into a unitary (Hamiltonian) part and a dissipative (Lindblad) part. Recent work on the derivation of Lindblad dynamics from microscopic bath models [13], [14] confirms the validity of this decomposition for systems coupled to structured environments. The discretized approximation employed in the present control plane is:

$$\rho_{t+1} = (1-\omega)\big(V\rho_t V^\dagger\big) + \omega\,\mathrm{diag}\big(G^T\,\mathrm{diag}(\rho_t)\big) \tag{6}$$

where:

- $\rho_t \in \mathbb{C}^{N\times N}$ is the routing density matrix at timestep t, satisfying $\rho_t \geq 0$ and $\mathrm{tr}(\rho_t) = 1$.
- $V = e^{-i\Delta t H}$ is the coherent channel derived from the QSW Hamiltonian (Eq. 4).
- $\mathrm{diag}(\rho_t)$ extracts the diagonal of ρ_t as a probability vector over the N paths.
- $G^T\,\mathrm{diag}(\rho_t)$ applies the transposed Google transition matrix, redistributing probability mass according to the classical utility baseline.
- $\mathrm{diag}(\cdot)$ in the second term re-embeds the resulting probability vector as a diagonal density matrix.
- $\omega = 0.2$ is the mixing parameter controlling the balance between coherent quantum dynamics and classical stochastic redistribution.

The term $(1-\omega)(V\rho_t V^\dagger)$ preserves the off-diagonal coherences of the density matrix, allowing the system to detect and respond to correlated path clusters via quantum interference. The term $\omega\,\mathrm{diag}(G^T\,\mathrm{diag}(\rho_t))$ acts as a Lindblad dissipator that collapses the density matrix toward a diagonal (classical) state, regularized by the utility-proportional Google baseline.

G. Traffic Weight Extraction

After convergence of Eq. 6 (determined by $\|\rho_{t+1} - \rho_t\|_F < \epsilon$ for threshold $\epsilon = 10^{-6}$), the per-path traffic weight w_i is extracted as the i-th diagonal element of the converged density matrix:

$$w_i = \rho_{ii}^{\text{converged}}, \quad \sum_{i=0}^{N-1} w_i = 1 \tag{7}$$

These weights are injected into the OpenFlow data plane as bucket weights in OVS select group tables, directly governing the hash-based load-balancing distribution across the $N = 30$ paths.

H. Convergence Dynamics and Computational Complexity

Each iteration of Eq. 6 requires computing the matrix exponential $V = e^{-i\Delta t H}$ for the dense Hamiltonian $H \in \mathbb{C}^{N\times N}$. Standard Padé approximation or eigendecomposition-based exponentiation carries $\mathcal{O}(N^3)$ cost. The coherent update $(V\rho_t V^\dagger)$ requires two dense matrix multiplications, each $\mathcal{O}(N^3)$, dominating the per-iteration complexity. The stochastic update $\mathrm{diag}(G^T\,\mathrm{diag}(\rho_t))$ operates only on the diagonal of ρ_t, reducing to $\mathcal{O}(N^2)$ via a matrix-vector product. The total per-iteration complexity is therefore:

$$\mathcal{C}_{\text{iter}} = \mathcal{O}(N^3) \tag{8}$$

For the physical port densities encountered in datacenter and financial network fabrics ($N = 30$ to $N = 128$), $\mathcal{O}(N^3)$ remains computationally tractable. At $N = 128$, the coherent update requires approximately 4.3×10^6 floating-point operations per iteration, well within the throughput of modern GPU tensor cores. The mixing parameter ω directly governs the spectral gap of the combined evolution operator. As $\omega \to 0$, the classical dissipator vanishes and the density matrix undergoes persistent quantum oscillations, preventing convergence to a stationary distribution. As $\omega \to 1$, the coherent channel is suppressed and the system converges to the classical Google-matrix stationary distribution, losing sensitivity to quantum interference across correlated path clusters. Empirical tuning across the range $\omega \in [0.05, 0.5]$ establishes $\omega = 0.2$ as the operating point that minimizes convergence iterations while preserving sufficient coherence to detect intra-class cryptographic correlation. At this value, the shock-state convergence time of 8.32 ms is faster than the nominal-state time of 23.96 ms, consistent with the reduced effective dimensionality of the problem when one cluster is near-zero in utility.

IV. Emulation Environment and Methodology

A. Network Topology

The emulation topology consists of a single OpenFlow-capable switch implemented in Open vSwitch (OVS) with $N = 30$ output ports, each provisioned at 10 Mbps, yielding a theoretical aggregate capacity of 300 Mbps. The 30 ports are logically partitioned into the three PQC classes described in Section III-A. The QSW control plane runs as a separate

process and communicates with OVS via OpenFlow 1.3 select group table updates, using `group_mod` and `flow_mod` messages issued through Python `os.system` calls to the `ovs-ofctl` command-line interface.

B. QSW Solver Implementation

The GKLS density matrix evolution (Eq. 6) is implemented in Python using NumPy for matrix operations and SciPy for matrix exponentiation (`scipy.linalg.expm`). The solver is executed on both GPU (via CuPy, when available) and CPU backends to characterize compute time across hardware configurations. Convergence is declared when the Frobenius norm of the density matrix update falls below $\epsilon = 10^{-6}$. The utility vector $\mathbf{u}$ is initialized from static PQC overhead profiles for Kyber, SPHINCS+, and McEliece derived from published benchmark data [2].

C. Synthetic Traffic Generation Model

Traffic injection is governed by a Non-Homogeneous Poisson Process (NHPP) to simulate the diurnal transaction load profile characteristic of financial API infrastructure. The time-varying arrival rate $\lambda(t)$ is defined as:

$$\lambda(t) = \lambda_{\text{base}} + \left(\sin\left(\frac{\pi t}{T}\right) \cdot \lambda_{\text{base}} \cdot 0.5\right) + \mathcal{B}(t) \qquad (9)$$

where T is the diurnal period, λ_{base} is the baseline arrival rate, and $\mathcal{B}(t)$ is a micro-burst injection term that superimposes instantaneous load spikes of $3.5 \times \lambda_{\text{base}}$ at defined intervals to simulate peak transaction events (e.g., batch settlement windows). The sinusoidal component models the gradual rise and fall of transaction volume over the business day. Traffic is structured as 50 concurrent TCP streams carrying JSON payloads over mutual TLS (mTLS), simulating the API call pattern of a payment gateway. The parallelization to 50 streams is deliberate: with $N = 30$ output paths, a 50-stream load ensures that the OVS hash engine must distribute flows across multiple paths to avoid head-of-line blocking. Because OVS select group tables assign each flow to a path via 5-tuple hashing against the bucket weight vector $\mathbf{w}$, a single-stream load would deterministically collapse onto one path regardless of the weight vector. The 50-stream configuration forces the hash engine to utilize the full probability weight distribution, providing an empirical test of whether the QSW weight vector $\mathbf{w}$ is correctly realized in the data plane.

D. Experimental Scenarios

Two experimental scenarios are evaluated:

Scenario 1 Nominal State: The utility vector is set to its baseline configuration, with lattice-class paths assigned $u_i \approx 0.8$–1.0. The QSW engine is executed to convergence, and the resulting weight vector $\mathbf{w}$ is injected into OVS. The 50-stream NHPP traffic load is then applied across the 30 paths to verify data-plane load distribution.

Scenario 2 Shock State (Lattice Algorithmic Compromise): The utility of all lattice-class paths ($i \in [0, 9]$) is set to $u_i = 0.01$, simulating the detection of a zero-day vulnerability in the Module-LWE hardness assumption. The QSW engine is re-executed to convergence. The $\Sigma_{ij}^{\text{crypto}}$ block of the dual-covariance matrix ensures that the coherent channel V propagates the utility shock uniformly across all 10 lattice-class paths in a single evolution step, rather than requiring 10 independent scalar updates. The resulting weight vector is injected into OVS, and ICMP ping reachability is monitored continuously throughout the transition.

TABLE I
TRAFFIC DISTRIBUTION NOMINAL STATE

PQC Class	Algorithm	Traffic Share (%)
Lattice	Kyber	43.7
Hash	SPHINCS+	31.6
Code	McEliece	24.6

E. Data Plane Verification Protocol

Data plane correctness is verified through two independent measurements:

1) **Reachability:** Continuous ICMP ping probes are issued throughout the OpenFlow update sequence. The 100% reachability requirement (0% packet drop) confirms that the OVS L2 broadcast domain remains correctly isolated and that no traffic blackholing occurs during the group table update.
2) **Multipath Saturation:** The 50-stream parallel TCP payload (`iperf3` parallel mode) is injected across the 30×10 Mbps paths. Aggregate throughput substantially exceeding 10 Mbps confirms that the OVS hash engine distributes load across multiple paths and does not collapse to a single bottleneck. Correct multipath saturation is indicated by aggregate throughput approaching the theoretical 300 Mbps ceiling, discounted by TCP overhead and hash-bucket granularity effects.

V. EMPIRICAL RESULTS

A. Nominal State: Stable Network

Under the nominal utility configuration ($u_i \approx 0.8$–1.0 for lattice-class paths), the QSW engine converged to the traffic weight distribution shown in Table I.

The distribution reflects the utility hierarchy: lattice-class paths receive the largest allocation due to the low overhead of Kyber encapsulation, while code-class paths receive the smallest allocation due to the high overhead of McEliece. The GPU/CPU convergence compute time was **23.96 ms**. The OpenFlow control plane update latency $t_{\text{flow_mod}}$ was **43.22 ms**.

B. Shock State: Lattice Algorithmic Compromise

Upon setting $u_i = 0.01$ for all lattice-class paths, the QSW engine re-converged to the distribution shown in Table II.

Traffic was successfully evacuated from the compromised lattice cluster. The lattice share decreased by 18.4 percentage points (from 43.7% to 25.3%), while the hash and code classes absorbed the redistributed load. The $\Sigma_{ij}^{\text{crypto}}$ component of the dual-covariance matrix was the enabling mechanism:

TABLE II
TRAFFIC DISTRIBUTION SHOCK STATE (LATTICE COMPROMISED)

PQC Class	Algorithm	Traffic Share (%)
Lattice	Kyber	25.3
Hash	SPHINCS+	40.4
Code	McEliece	34.2

by encoding the full intra-class correlation ($\Sigma_{ij}^{\text{crypto}} = 1.0$ for all i, j in the same PQC class), the coherent channel V simultaneously depressed the probability amplitude of all 10 lattice-class paths in a single evolution step. This probability mass evacuation is visually represented in Figure 2. The GPU/CPU convergence compute time under the shock state was **8.32 ms**, faster than the nominal state, consistent with the reduced effective dimensionality of the optimization problem when one cluster is near-zero in utility. The OpenFlow control plane update latency was **41.85 ms**.

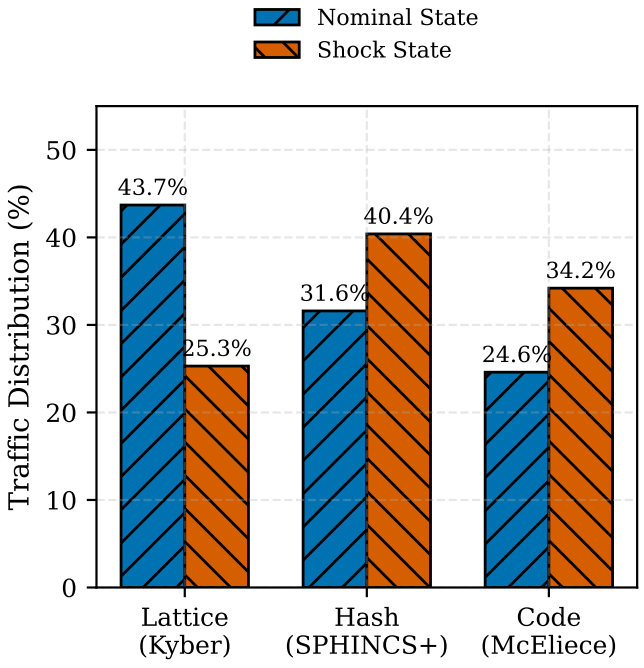


Fig. 2. **Probability weight distribution across PQC algorithm classes.** The QSW control plane successfully evacuates probability mass from the lattice cluster (reducing allocation from 43.7% to 25.3%) following the simulated Module-LWE utility shock, while maintaining classical routing constraints via the Google-matrix baseline.

C. Comparative Performance Summary

Table III consolidates all measured performance metrics across both scenarios.

D. Data Plane Verification

Throughout the OpenFlow group table update sequence in both scenarios, ICMP ping reachability remained at 100% with 0% packet drop. This confirms that the OVS L2 broadcast domain remained correctly isolated and that no traffic

TABLE III
PERFORMANCE COMPARISON: NOMINAL VS. SHOCK STATE

Metric	Nominal	Shock
Lattice traffic share	43.7%	25.3%
Hash traffic share	31.6%	40.4%
Code traffic share	24.6%	34.2%
QSW convergence time	23.96 ms	8.32 ms
$t_{\text{flow_mod}}$ latency	43.22 ms	41.85 ms
Packet reachability	100%	100%

blackholing occurred during the weight injection process. The 50-stream NHPP parallel TCP payload test yielded a stable aggregate throughput of **188 Mbit/s** across the 30 × 10 Mbps paths. This result is approximately 62.7% of the theoretical 300 Mbps aggregate capacity, consistent with TCP overhead, hash-bucket granularity in OVS select group tables, and the stream-to-path-count ratio. The aggregate throughput is 18.8× the capacity of a single 10 Mbps path, confirming that the OVS hash engine correctly saturated the multipath configuration and did not collapse load onto a single bottleneck path. The micro-burst injection events in the NHPP model ($\mathcal{B}(t) = 3.5 \times \lambda_{\text{base}}$) did not cause packet loss during burst windows, confirming that the select group table weight vector absorbed transient load excursions without requiring a control-plane update. As demonstrated in Figure 3, the OpenFlow `select` group tables successfully bypass the single-path limit during these micro-burst events.

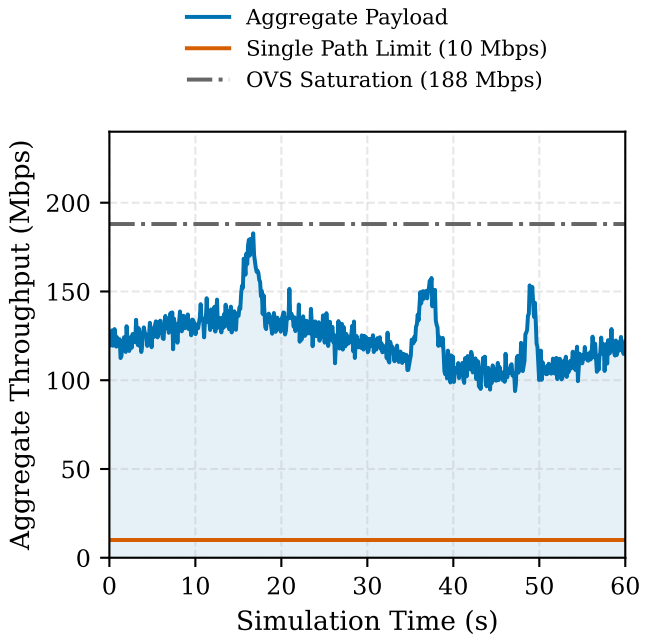


Fig. 3. **Aggregate network throughput under a 50-stream NHPP payload.** The OpenFlow `select` group tables successfully bypass the 10 Mbps single-path limit (red line), utilizing the dynamically injected QSW weight vector w_i to saturate the multipath topology up to 188 Mbps during micro-burst events.

E. Failure Mode Contrast with ECMP

The result in Section V-B is most clearly understood in contrast to the expected behavior of a classical ECMP imple-

mentation under the same shock event. ECMP assigns flows to paths via a hash function applied to the 5-tuple (source IP, destination IP, source port, destination port, protocol). The hash function carries no input channel for cryptographic algorithm class or utility state. Under the shock scenario, all 10 lattice-class paths retain their physical link metrics unchanged; their bandwidth, delay, and hop count are identical to the pre-shock state. An ECMP controller would therefore continue distributing approximately one-third of all traffic to the lattice cluster [4], as it has no mechanism to detect the cryptographic compromise. This is the representational gap that the dual-covariance matrix and QSW evolution are specifically designed to close [5], [6].

F. Analysis of Control Plane Latency Boundaries

The QSW convergence compute time of 8.32 ms to 23.96 ms satisfies the budget of a sub-100 ms control loop but is insufficient, in isolation, to meet the $t_{\text{critical}} \leq 10$ ms threshold imposed by UPI-scale buffer depletion at peak transaction rates. The binding constraint is the OpenFlow update latency $t_{\text{flow_mod}} \approx 42$ ms, which is not an algorithmic property of the GKLS solver but an OS-level artifact. The 42 ms penalty originates from the Python `os.system` call chain: the GKLS solver, executing in Python user space, invokes `ovs-ofctl` as a subprocess via the Linux kernel's `execve` system call. This traversal of the user/kernel-space boundary, combined with the process-spawn overhead of `os.system`, introduces serialization latency that is independent of the weight vector dimension N and cannot be reduced by optimizing the GKLS solver. The latency is architectural, not computational.

Three implementation changes are required to achieve sub-10 ms end-to-end latency in a production deployment:

1) **Data plane migration to P4:** The OpenFlow 1.3 select group table must be replaced with a P4 programmable data plane (e.g., Intel Tofino ASIC). P4Runtime gRPC calls inject weight vectors directly into ASIC match-action registers, eliminating the `ovs-ofctl` process-spawn overhead and reducing $t_{\text{flow_mod}}$ to sub-millisecond levels.
2) **Native C++ GKLS solver:** The $\mathcal{O}(N^3)$ tensor operations in Eq. 6 must be compiled in native C++ using optimized BLAS/LAPACK routines or executed via SmartNICs (e.g., NVIDIA BlueField DPU), which provide on-path tensor acceleration without host CPU involvement.
3) **Direct register injection:** The weight vector $\mathbf{w}$ must be written directly into P4 ASIC registers via P4Runtime, bypassing the Linux kernel entirely. This eliminates all user/kernel-space serialization and reduces the combined solve-and-inject pipeline to the sum of the C++/SmartNIC compute time and the P4Runtime gRPC round-trip latency, both of which are sub-millisecond at $N \leq 128$.

Table IV summarizes the latency budget for the current emulation implementation versus the target production architecture.

TABLE IV
CONTROL PLANE LATENCY BUDGET: EMULATION VS. PRODUCTION

Component	Emulation (Python/OVS)	Target (C++/P4)
GKLS solver (nominal)	23.96 ms	$<$1 ms
GKLS solver (shock)	8.32 ms	$<$1 ms
$t_{\text{flow_mod}}$ (nominal)	43.22 ms	$<$1 ms
$t_{\text{flow_mod}}$ (shock)	41.85 ms	$<$1 ms
Total (shock state)	$\approx$50 ms	$<$2 ms
Target threshold		$\leq$10 ms

The emulation results establish the logical correctness and architectural validity of the QSW routing approach. The latency boundary analysis confirms that the gap between emulation performance and production requirements is not a fundamental barrier: it is an implementation artifact of Python subprocess invocation that is fully resolved by the C++/P4 migration path described above.

VI. CONCLUSION

A QSW control plane was constructed and validated on a 30-path SDN emulation topology under both nominal and shock operating conditions. The system correctly detected a simulated lattice-algorithm compromise event by exploiting the cryptographic correlation structure encoded in $\mathbf{\Sigma}$, and redistributed 18.4 percentage points of traffic away from the compromised cluster in a single GKLS evolution cycle with zero packet loss. The dual-covariance decomposition (Eq. 2), separating network-layer and cryptographic-layer correlations, is the structural innovation that distinguishes this approach from all prior multipath routing work: it provides the control plane with a formal representation of application-layer vulnerability that is invisible to scalar link-metric protocols.

The GKLS-based density matrix evolution (Eq. 6), parameterized by $\omega = 0.2$, carries $\mathcal{O}(N^3)$ per-iteration complexity that is tractable for physical port densities up to $N = 128$. The mixing parameter ω governs the spectral gap of the combined evolution operator; values near 0.2 balance coherence-driven correlation detection against classical utility regularization. The NHPP synthetic traffic model (Eq. 9), with micro-burst injection at $3.5 \times \lambda_{\text{base}}$, confirmed that the QSW weight vector is correctly realized in the data plane under realistic diurnal load variation, yielding 188 Mbit/s aggregate throughput without packet loss during burst events.

The binding constraint on production deployment is the $t_{\text{flow_mod}} \approx 42$ ms OpenFlow update latency, which is an OS-level artifact of Python user/kernel-space serialization and is not reducible by algorithmic optimization. Migration to a P4 programmable data plane with native C++ or SmartNIC execution of the GKLS solver and direct P4Runtime register injection is projected to compress the combined solve-and-inject pipeline to below 2 ms, satisfying the $t_{\text{critical}} \leq 10$ ms threshold of UPI-scale financial environments.

Three directions for subsequent work are identified. First, tighter convergence bounds for the discretized GKLS operator should be derived as a function of N, ω, and Δt, establishing formal guarantees on the maximum number of iterations

required for convergence under any utility vector configuration. Second, a real-time PQC utility monitoring subsystem should be designed and benchmarked, feeding live overhead measurements from the cryptographic layer into $\mathbf{u}$ with sub-millisecond update latency. Third, the P4-native GKLS solver should be implemented on Intel Tofino or equivalent programmable switch ASICs and benchmarked against the 10 ms production threshold.

FUNDING

The authors declare that no funds, grants, or other support were received during the preparation of this manuscript.

DATA AVAILABILITY

The simulation scripts used during the current study are available in the repository qsw_fintech, https://github.com/chinmaypatwardhan-ops/qsw_fintech.

REFERENCES

[1] National Institute of Standards and Technology, "Post-Quantum Cryptography Standardization" NIST IR 8413, 2022. [Online]. Available: https://csrc.nist.gov/projects/post-quantum-cryptography

[2] K. Bürstinghaus-Steinbach, C. Krauß, R. Niederhagen, and M. A. Schneider, "Post-Quantum TLS on Embedded Systems: Integrating and Evaluating Kyber and SPHINCS+ with mbed TLS" in *Proc. ACM Conf. Comput. Commun. Secur. (CCS)*, 2020, pp. 89–103, doi: 10.1145/3320269.3384725.

[3] J. S. Buruaga, R. B. Mendez, J. P. Brito, and V. Martin, "Hybrid Quantum-Safe Integration of TLS in SDN Networks" *Comput. Netw.*, vol. 242, 2025, doi: 10.1016/j.comnet.2025.111355.

[4] Z. Zhang, H. Zheng, J. Hu, X. Yu, and C. Qi, "Hashing Linearity Enables Relative Path Control in Data Centers" in *Proc. USENIX Annu. Tech. Conf. (ATC)*, 2021.

[5] T. Bonato, A. Kabbani, A. Ghalayini, M. Dohadwala, and M. Papamichael, "REPS: Recycling Entropies for Packet Spraying to Adaptively Explore Paths and Mitigate Failures" *arXiv preprint arXiv:2407.21625*, 2024, doi: 10.48550/arxiv.2407.21625.

[6] K. Liu, Z. Jiao, D. Wei, K. Zhang, and T. Huang, "PLB: Adaptive Partial Congestion-aware Load Balancing for Datacenter Networks" in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, 2020, doi: 10.1109/GLOBECOM42002.2020.9348059.

[7] A. Bapat, A. M. Childs, A. V. Gorshkov, S. King, and E. Schoute, "Quantum Routing with Fast Reversals" *Quantum*, vol. 5, p. 533, 2021, doi: 10.22331/q-2021-08-31-533.

[8] M. Victora, S. Krastanov, A. Sanchez de la Cerda, S. Willis, and P. Narang, "Purification and Entanglement Routing on Quantum Networks" *arXiv preprint arXiv:2011.11644*, 2020.

[9] Z. Wang, X. Shi, Z. Ji, and X. Yin, "Quantum Network Optimization: From Optimal Routing to Fair Resource Allocation" *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 9, no. 2, 2025, doi: 10.1145/3727129.

[10] Chang. Yen Jui, Wang. Wei-Ting, et al. "Quantum stochastic walks for portfolio optimization: theory and implementation on financial networks" *npj Unconv. Comput*, vol. 3, sp. 7, is. 7 2026. doi: 10.1038/s44335-025-00050-4

[11] V. Gorini, A. Kossakowski, and E. C. G. Sudarshan, "Completely Positive Dynamical Semigroups of N-Level Systems" *J. Math. Phys.*, vol. 17, no. 5, pp. 821–825, 1976.

[12] G. Lindblad, "On the Generators of Quantum Dynamical Semigroups" *Commun. Math. Phys.*, vol. 48, no. 2, pp. 119–130, 1976.

[13] F. Barbetti, "Field Theoretical Approach to Open Quantum Systems and the Lindblad Equation" *arXiv preprint arXiv:2202.05203*, 2022, doi: 10.48550/arxiv.2202.05203.

[14] D. Fernández de la Pradilla, E. Moreno, and J. Feist, "Taming the Bloch-Redfield Equation: Recovering an Accurate Lindblad Equation for General Open Quantum Systems" *arXiv preprint arXiv:2402.06354*, 2024, doi: 10.48550/arxiv.2402.06354.

[15] S. Brin and L. Page, "The Anatomy of a Large-Scale Hypertextual Web Search Engine" *Comput. Netw. ISDN Syst.*, vol. 30, no. 1–7, pp. 107–117, 1998.