Journal of Computing and Electronic Information Management
ISSN: 2413-1660 | Vol. 22, No. 1, 2026

raphRoute -Transfer: Topology-eneralizable Routing Convergence Optimization via raph Reinforcement Learning

Yuto Nakamura

University of Massachusetts Amherst, USA
y.nakamura2@umass.edu

Abstract: Fast and stable routing convergence is critical in large IP networks, and the interior-gateway-protocol (IGP) timers that govern failure detection (Hello/Dead intervals) expose a fundamental tension: aggressive timers detect failures quickly but inflate control overhead and trigger route flaps, whereas conservative timers are stable but slow. Recent work such as DRL-Adapt has shown that deep reinforcement learning can tune these timers better than static defaults, but it operates on a flat, globally-aggregated network state and emits a single network-wide timer, so it can neither exploit the spatial heterogeneity of real topologies nor transfer architecturally across networks of different size. We propose GraphRoute-Transfer, a graph-neural-network policy that assigns per-node timers from local structural features and is by construction permutation- and size-invariant. Because control-plane fragility and failure criticality are spatially heterogeneous, the cost-minimizing timer assignment varies across the graph; our policy learns this mapping and applies it zero-shot to unseen topologies of arbitrary size. Training is guided by a coordinate-descent search oracle on a convergence-cost objective, so the expensive per-topology optimization is amortized into a sub-millisecond inference. On 231 real topologies from the Internet Topology Zoo, GraphRoute-Transfer reduces mean convergence time by 37.3% relative to the OSPF default and to a flat DRL baseline, lowers the composite convergence-cost objective by 16.4% over the flat baseline, and attains 1.317 cost—within 0.3% of the search oracle—while running about 8,160× faster than the search. Crucially, a policy trained only on networks with ≤70 nodes maintains its gains on unseen networks up to 140 nodes, whereas the flat baseline degenerates to a global constant that cannot adapt.

Keywords: Routing convergence; Graph neural networks; Reinforcement learning; OSPF; Timer optimization; Topology generalization; Network reliability.

1. Introduction

Link-state interior gateway protocols (IGPs) such as OSPF [2] and IS-IS underpin routing in virtually every large enterprise and carrier network, while BGP [3] glues autonomous systems together. When a link or router fails, the network must re-converge: neighbours detect the failure, flood updated link-state advertisements, recompute shortest paths, and install new forwarding state. The speed and stability of this process directly determine packet loss, transient loops, and service availability [5], [6]. A dominant lever on convergence speed is the set of protocol timers—in particular the Hello interval and the derived Dead interval—that control how quickly an adjacency failure is detected [7].

These timers embody a sharp trade-off. Short Hello intervals detect failures within a fraction of a second but multiply control-plane traffic and, more importantly, make the protocol prone to spurious "down" events: when a Hello is lost to transient congestion or jitter, a still-healthy adjacency is torn down and later restored, producing a route flap and a long oscillation tail [7], [8]. Long intervals are stable but leave the network blind to failures for seconds. Operators therefore face a per-network, and in fact per-router, tuning problem that is tedious and error-prone.

Deep reinforcement learning (DRL) has recently been applied to exactly this problem. DRL-Adapt framework [10] formulates timer adaptation as a Markov decision process and uses a dueling double DQN to select global timer settings, reporting substantially faster convergence than static defaults across CAIDA-derived topologies while generalizing over a range of network sizes. DRL-Adapt is a compelling demonstration that learned control can outperform hand-set timers. However, it shares two limitations with most learned network-control agents. First, its state is a fixed-length vector of globally aggregated features and its action is a single network-wide timer; it cannot express that a well-connected core router and a fragile stub should be configured differently. Second, its policy network has a fixed input/output dimensionality, so transferring it to a network with a different number of nodes requires re-aggregation or retraining—generalization is statistical (the aggregated features happen to look similar) rather than architectural.

We argue that routing-timer optimization is intrinsically a graph problem. The cost-minimizing timer at a node depends on local structural properties—how central the node is (hence how often and how consequentially it is involved in failures) and how fragile its local control channel is (hence its flap risk). These properties are spatially heterogeneous and are naturally read off the topology. We therefore propose GraphRoute-Transfer, a graph neural network (GNN) policy that (i) consumes per-node local features, (ii) emits a per-node timer, and (iii) is permutation- and size-invariant, so a single trained model applies to any topology, of any size, in one forward pass.

Because the per-node objective couples all nodes through global convergence events, naïve multi-head policy-gradient training is high-variance and collapses to the safe uniform policy. We instead guide training with a coordinate-descent search oracle that minimizes a convergence-cost objective on each training topology; the GNN is trained to reproduce the oracle from local features. This turns an expensive per-

topology optimization into an amortized, transferable, sub-millisecond inference—a learning-to-optimize view [22] that also yields a clean controlled comparison against a flat agent trained on the identical signal.

Contributions. (1) We formulate routing-convergence timer optimization as a per-node decision problem on the network graph and identify spatial heterogeneity of fragility and criticality as the property that makes per-node control valuable. (2) We design GraphRoute-Transfer, a size-invariant GNN policy that assigns per-node timers and transfers zero-shot across topology sizes. (3) We introduce an oracle-guided training scheme that amortizes per-topology search. (4) On 231 real Internet Topology Zoo networks we show a 37.3% convergence-time reduction over the OSPF default and a flat DRL baseline, near-oracle cost at ~8,000× lower configuration time, and—our headline result—robust zero-shot transfer to networks up to twice the largest training size, where the flat baseline cannot adapt at all.

2. Related work

2.1. Routing convergence and timer tuning

The decomposition of IGP convergence into detection, flooding, SPF computation, and settling is well established; Francois et al. [6] measured each component on production routers and showed sub-second convergence is achievable by careful timer and process tuning. Stability analyses of OSPF [7] and studies of delayed BGP convergence [5], [8] document how overly aggressive configuration causes oscillations and flaps, and how protocol timers dominate the reaction time to failures. Vendor guidance and operator practice encode these trade-offs as static rules of thumb (e.g. a default Hello of 10 s with a 4× Dead interval) applied uniformly across a network. These works establish the physical model and the convergence-cost objective we optimize, but treat configuration as a manual, network-wide, static decision rather than a data-driven, per-node one.

2.2. Learning for network control

Reinforcement learning for networking dates back to Q-routing [23] and has expanded to traffic engineering, congestion control [29], resource management [27], and SDN routing [28], [26], typically building on modern deep-RL machinery—value-based methods [9], [11], [12] with prioritized replay [14], and policy-based methods such as A3C [14], PPO [15] and the policy-gradient theorem [16]. Valadarsky et al. [24] framed routing as a learning problem and identified generalization to unseen demands and topologies as the central obstacle. Closest to us is DRL-Adapt [10], which learns global IGP timer policies with a dueling double DQN [12], [13] and prioritized replay [14] and demonstrates clear gains over static defaults; we adopt its MDP framing as our starting point and use a flat DRL agent in its spirit as our principal learned baseline. Our departure is architectural rather than algorithmic: we replace the global, fixed-dimensional flat policy with a per-node graph policy, which changes both what the agent can express (spatially varying timers) and how it generalizes (architecturally, across sizes).

2.3. Graph neural networks for networks

GNNs [21] such as GCN [18], GraphSAGE [19], and GAT [20], unified under the message-passing view [17] (MPNN), learn permutation-invariant functions on graphs and generalize across graph sizes. In networking, RouteNet [25] used message passing to predict per-path delay and jitter and generalize across topologies, and Almasan et al. [30] combined DRL with GNNs for routing optimization, arguing that graph structure is key to generalization. We bring this insight to convergence-timer control and, unlike modeling-oriented GNN work, output an actionable per-node configuration and quantify zero-shot size transfer against a matched non-graph baseline.

3. Problem formulation

We consider a network as an undirected graph $G=(V,E)$ with $|V|=n$ routers. Each node v carries a Hello timer h_v drawn from a discrete set of set-points; the Dead interval is $D_v=\rho h_v$ with $\rho=4$ following common practice. Over an operating window the network experiences a sequence of failure events (single links, router failures, and correlated multi-link/cascading failures). For an event we model the reconvergence time following the standard breakdown [6]:

$$T_{conv} = T_{detect} + T_{flood} + T_{SPF} + T_{settle},$$

where $T_{detect} \approx D_e - \tfrac{1}{2}h_e$ is governed by the timers at the failing element e, T_{flood} is the delay-weighted multi-source flooding time over G, T_{SPF} is the shortest-path recomputation time, and T_{settle} is a settling tail proportional to the number of route flaps triggered by the event.

Flap model. Each node has a latent fragility threshold θ_v: when its Hello interval is short relative to θ_v, a transient loss can be misread as a failure. The probability that node v flaps during an event is $\sigma(\beta(\theta_v - h_v))$, amplified near the failure and by traffic load. Fragility is spatially heterogeneous—central, heavily-multiplexed routers have more congested control channels—so θ_v grows with node centrality. Because failures are also more frequent and consequential at central nodes, the cost-minimizing timer is a non-trivial per-node function of local structure: it can be aggressive where a node is stable and its fast detection matters, and must be conservative where a node is fragile.

Control overhead. Each node emits Hello packets at rate $1/h_v$, so its steady-state control traffic grows as short timers are chosen; a failure additionally triggers link-state flooding whose volume scales with the affected subgraph and any induced flaps. We measure overhead in kB per node, combining the per-node Hello traffic with an equal share of the event-triggered flooding. Overhead therefore pushes timers longer, opposing the detection incentive and completing a three-way tension between speed, overhead, and stability.

Why per-node control matters. If the cost-minimizing configuration were uniform, a global agent would suffice. It is not: because fragility and criticality are heterogeneous and only weakly correlated across nodes, the best per-node assignment strictly dominates the best global one. Empirically (Section VI) the per-node search oracle attains $J=1.313$, about 16.7% below the best uniform timer ($J=1.575$); this gap is the headroom a per-node policy can capture and a global one cannot. The learned configuration in Fig. 1 illustrates the resulting non-uniform timer field.

Objective. Let T_{base} and O_{base} be the mean convergence time and control overhead under the OSPF default configuration on the same topology. We minimize the per-topology convergence-cost

$$J = T_{conv}/T_{base} + \tfrac{1}{2}\cdot O/O_{base} + 0.15\cdot F,$$

averaged over events and seeds, where O is control overhead (kB/node) and F is the flap count. Normalization by

the default makes J comparable across scales. A configuration policy π maps G to a per-node timer vector; we seek π that minimizes E[J] and, critically, that transfers to topologies unseen during training.

4. Graphroute-transfer

4.1. Per-node features and graph policy

For each node we compute an eight-dimensional local feature vector: normalized degree and betweenness centrality, mean incident link delay, the fragility proxy θ_v, the current Hello set-point, a decayed recent-event indicator, hop-distance to the most recent failure, and episode progress. The policy is a GraphSAGE-style [19] message-passing network: three layers of self-and-neighbour aggregation over the symmetric-normalized adjacency, followed by a per-node action head that outputs a distribution over Hello set-points and a per-node value head. Because every operation is shared across nodes and aggregation is permutation-invariant, the same parameters apply to a graph of any size and any node ordering—the architectural basis for zero-shot transfer.

At deployment the policy reads the topology once and emits all per-node timers in a single forward pass (a one-shot configuration), so inference cost is a few hundred microseconds and independent of any search.

4.2. The flat (DRL-Adapt-style) baseline

To isolate the value of the graph structure we implement a flat baseline in the spirit of DRL-Adapt [10]: a multilayer perceptron on globally-aggregated features (mean, max, and min of the node features, 24 dimensions) that emits a single network-wide Hello set-point. It has the same action set and is trained on the identical target signal; it differs only in that it cannot observe or act on per-node structure. This makes the comparison a controlled test of architecture rather than of training procedure.

4.3. Oracle-guided learning

The per-node objective couples all nodes through global events, so simultaneously exploring n independent action heads with policy gradients is high-variance and, in our experiments, collapses to the conservative uniform policy. We instead obtain high-quality targets with a cheap coordinate-descent oracle that, per training topology, starts from the best uniform set-point and greedily improves one node at a time to minimize J. The GNN is trained by cross-entropy to reproduce the oracle's per-node decisions from local features; the flat baseline is trained to predict the best uniform action. This is a learning-to-optimize scheme [22]: the oracle is an expensive per-topology optimizer, and the GNN amortizes it into a transferable constant-time policy. The oracle is used only during training; at test time only the learned policy runs.

4.4. Why not end-to-end policy gradient?

One might train the graph policy directly by policy gradient on E[J]. The difficulty is credit assignment: a single global convergence event produces one scalar cost shared by all n simultaneously-sampled per-node actions, so the advantage seen by any one node is dominated by the noise of the other n−1. In our experiments this drives the entropy-regularized policy to the low-variance safe solution—the conservative uniform timer—discarding the very per-node structure we want. The oracle-guided scheme sidesteps this: the coordinate-descent search resolves per-node credit offline by construction, and supervised imitation of its decisions is low-variance and stable. It also yields a strictly controlled comparison, since the graph and flat policies are then trained on an identical target.

4.5. Training algorithm

Training proceeds in two stages. Offline, for each training topology we run the coordinate-descent oracle over the discrete set-points to obtain a per-node target assignment. Online, we minimize the mean cross-entropy between the policy's per-node action distribution and the oracle target over mini-batches of topologies with Adam, clipping gradients for stability. The flat baseline instead minimizes the cross-entropy between its single global logit and the modal oracle action per topology—the best a uniform policy can do. Both use the same set-points, optimizer, and budget; only the architecture and the granularity of the target differ. At inference the oracle is discarded and a single forward pass of the learned policy configures the whole network.

4.6. Complexity

A forward pass is $O(L(|E|d + nd^2))$ for L layers and hidden width d, i.e. linear in the number of edges—practical for large networks. The oracle is far costlier (many simulated episodes per node per sweep), which is precisely why amortization matters.

5. Experimental setup

5.1. Dataset

We use the Internet Topology Zoo [4], a curated collection of real wide-area network topologies annotated with geographic coordinates. We parse the GraphML files, take the largest connected component, and derive per-link propagation delays from node coordinates via the haversine distance at fiber speed. After filtering we obtain 231 real topologies spanning 8–140 nodes, partitioned into size buckets Small (8–20), Medium (20–40), Large (40–70), and XLarge (70–140). We use a stratified 60/40 train/test split within each bucket; models are trained only on topologies with ≤70 nodes (Small/Medium/Large), so the XLarge bucket is entirely unseen and larger than anything in training, giving a strict zero-shot size-transfer test.

5.2. Convergence simulator

We built a physically-grounded discrete-event convergence model implementing the detection/flooding/SPF/settling decomposition and the heterogeneous flap process of Section III, with correlated failure modes (single link, router, multi-link, cascade). Betweenness-biased-plus-uniform sampling selects failing elements so that both core and edge nodes fail. All reported metrics are averaged over 10 random seeds per topology.

5.3. Baselines and metrics

We compare GraphRoute-Transfer against: the flat DRL-Adapt-style agent; an adaptive rule-based heuristic that shortens timers where recent events are high; three static configurations (Default Hello=10 s, Aggressive=1 s, Conservative=20 s); an untrained-GNN Random-Agent control; and the coordinate-descent Oracle as an upper-bound reference. Metrics are mean convergence time T_conv (s), control overhead (kB/node), route flaps per event, and the composite cost J (lower is better). We also report per-topology

configuration time to quantify the amortization benefit.

5.4. Implementation

The policy has three message-passing layers of width 64. Oracle-guided training runs 600 updates with Adam; inference latency is measured on CPU. Key settings are listed in Table 1. The full pipeline (oracle construction, training, and all five experiments over 231 topologies) completes in about 16 minutes on a single core, underscoring the low cost of the amortized approach.

Table 1. Model and training configuration.

Component	Setting
Message-passing layers	3 (GraphSAGE-style)
Hidden width	64
Hello set-points (K)	{1, 2, 4, 6, 10} s
Dead interval	4 × Hello
Node features	8 (topology / protocol / event)
Optimizer	Adam, lr 3×10^{-3}
Training updates	600 (batch 8 topologies)
Oracle	coordinate descent, 2 sweeps
Eval seeds / topology	10
Objective weights (α, β, η)	1.0, 0.5, 0.15

6. Results and analysis

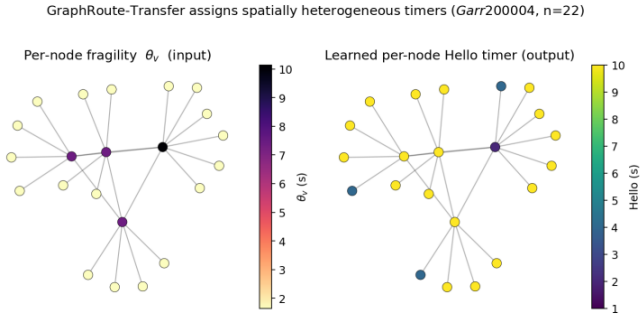

Fig. 1 Learned per-node configuration on a real topology. Left: input per-node fragility θ_v. Right: the Hello timer assigned by GraphRoute-Transfer. The policy produces a spatially heterogeneous configuration—balancing detection urgency at central nodes against flap risk—that no single global timer can express.

6.1. Overall performance and the convergence–stability trade-off (E1)

Table 2 and Fig. 2 summarize performance over the 94 test topologies. GraphRoute-Transfer attains the lowest composite cost of any method, J=1.317, versus 1.575 for both the flat DRL agent and the OSPF default—a 16.4% reduction—while cutting mean convergence time by 37.3% (from 35.3 s to 22.1 s). Remarkably it matches the coordinate-descent oracle (J=1.313) to within 0.3%.

The single-objective baselines expose why a learned trade-off is needed. Aggressive-Static achieves the fastest detection (12.1 s) but at a catastrophic 21.3 flaps/event and 10× the overhead, giving the worst cost (J=8.52). Conservative-Static is stable but slow (70.1 s). The heuristic and the untrained Random-Agent obtain low T_conv only by being indiscriminately aggressive, incurring 5–6 flaps/event. GraphRoute-Transfer is the only method in the desirable low-T_conv, low-flap corner of Fig. 2: it is aggressive precisely where doing so is safe and beneficial, and conservative elsewhere.

The flat DRL agent is instructive: trained on the same oracle signal, it collapses to the best uniform timer (Hello=10 s) because its aggregate input cannot localize where aggression is safe. It therefore ties the default and never realizes the per-node gains the graph policy captures.

Table 2. Overall performance on 94 held-out real topologies (lower is better; best learned value in bold context).

Method	T_conv (s)	OH	Flaps	J
GraphRoute-Transfer (ours)	22.1	0.87	0.66	1.317
Flat-DRL-style	35.3	0.74	0.47	1.575
Adaptive-Heuristic	16.3	1.80	5.27	2.474
Default-Static (10 s)	35.3	0.74	0.47	1.575
Aggressive-Static (1 s)	12.1	7.37	21.29	8.520
Conservative-Static (20 s)	70.1	0.37	0.00	2.250
Random-Agent	16.5	1.86	6.00	2.627
Oracle (search, ref.)	20.2	0.93	0.73	1.313

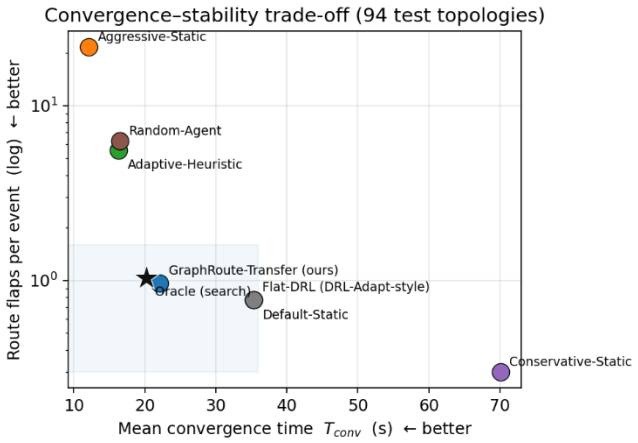

Fig. 2 Convergence–stability trade-off. GraphRoute-Transfer and the search oracle occupy the low-latency, low-flap corner; static and heuristic baselines trade one objective for the other.

6.2. Scalability and zero-shot size transfer (E2, E4)

Fig. 3 breaks performance down by size bucket. GraphRoute-Transfer maintains T_conv in the 20–26 s range across all buckets, including the XLarge networks (70–140 nodes) that are entirely unseen and larger than any training graph—a strict zero-shot generalization result enabled by the size-invariant architecture. The flat baseline is flat in a different sense: a constant ~35 s everywhere, because a single global timer cannot exploit any structure regardless of scale.

Table 3 makes the transfer explicit: we train each architecture on a single size bucket and evaluate on all buckets. GraphRoute-Transfer trained on any bucket transfers to all sizes (e.g., trained on Medium it achieves 24.4/19.3/20.9/25.2 s on Small/Medium/Large/XLarge), with Medium/Mixed training generalizing best. The flat agent yields ~35.3 s regardless of training or test size—it collapses to the same global constant and cannot transfer because there is no per-node mapping to transfer. This is the core evidence for our thesis: generalization here is architectural, not statistical.

Two further observations reinforce the point. First, the flat agent's near-zero variance across sizes (constant 35.3 s) is not stability but degeneracy: it has no size-dependent behaviour

to vary. GraphRoute-Transfer, by contrast, adapts its output to each topology, which shows up as a larger spread of per-topology convergence times (std 5.60 s in Table 2) precisely because it is doing useful work. Second, the small difference between the in-distribution buckets (Small/Medium/Large) and the zero-shot XLarge bucket indicates a modest generalization gap: the learned local-feature-to-timer mapping remains valid at scales never seen in training—the behaviour a size-invariant architecture should exhibit and a fixed-dimensional one cannot.

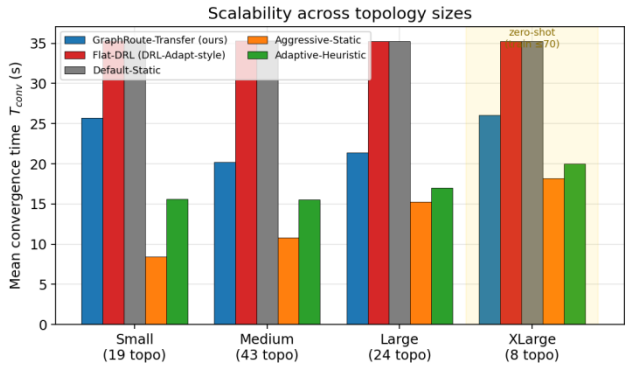

Fig. 3 Convergence time by topology size. Models are trained only on ≤70-node networks; the XLarge bucket (shaded) is zero-shot. GraphRoute-Transfer sustains its advantage at unseen scales.

Table 3. Zero-shot size transfer: T_conv (s) when training on one size and testing on all. GNN transfers; Flat collapses to a constant.

Train \\ Test	Small	Medium	Large	XLarge
GNN / Small	24.4	20.1	20.8	25.5
GNN / Medium	24.4	19.3	20.9	25.2
GNN / Large	28.3	23.6	23.1	29.1
GNN / Mixed≤70	26.6	21.0	21.7	27.4
Flat / Mixed≤70	35.3	35.3	35.2	35.3

6.3. Robustness to failure scenarios (E3)

Fig. 4 evaluates the four failure modes on Large test topologies. GraphRoute-Transfer keeps convergence at 21–28 s with under ~1 flap/event across all modes, degrading gracefully for correlated multi-link and cascade failures. Aggressive-Static achieves comparable or lower T_conv only by incurring 30–63 flaps/event—operationally unacceptable. The learned policy thus retains its favourable trade-off under stress, not just on average.

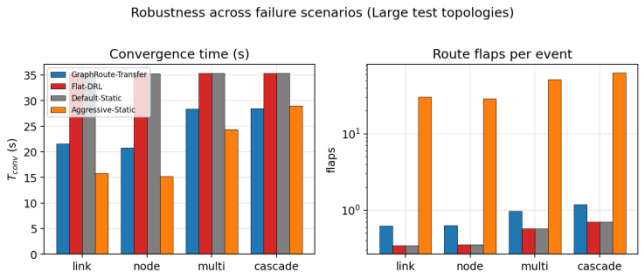

Fig. 4. Robustness across failure scenarios (Large test topologies). GraphRoute-Transfer preserves low convergence time with negligible flaps; the aggressive baseline is fast only at the cost of severe instability (note log scale on flaps).

6.4. Amortization: near-oracle quality at a fraction of the cost

Fig. 5 quantifies the learning-to-optimize benefit. GraphRoute-Transfer reaches essentially oracle-level cost (J=1.317 vs 1.313) but configures a topology in 0.44 ms versus 3.59 s for the search—about 8,160× faster—and, unlike the search, the learned policy transfers to new topologies without any per-topology optimization. This is what makes the approach deployable: the expensive optimization is paid once, offline, and reused everywhere.

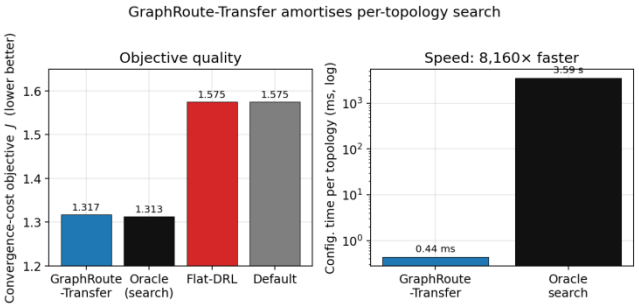

Fig. 5. GraphRoute-Transfer amortizes per-topology search: near-identical objective quality (left) at roughly four orders of magnitude lower configuration time (right, log scale).

6.5. Ablations (E5)

Table 4 ablates the two design axes. Replacing the graph policy with the flat MLP (architecture ablation) is the most damaging change—convergence rises from 22.1 s to 35.3 s—confirming that the graph structure, not the training signal, delivers the gains. Among input features, removing the topology group (degree/betweenness/delay/fragility) is by far the most harmful (T_conv 25.1 s), verifying that the policy grounds its per-node decisions in structural signals. The recent-event feature contributes little here because timers are configured once from the initial state, when no event has yet occurred.

Table 4. Ablation of architecture and input-feature groups.

Ablation	T_conv (s)
Full model	22.1
– event feature	21.3
– protocol features	22.3
– topology features	25.1
Flat MLP (no graph)	35.3

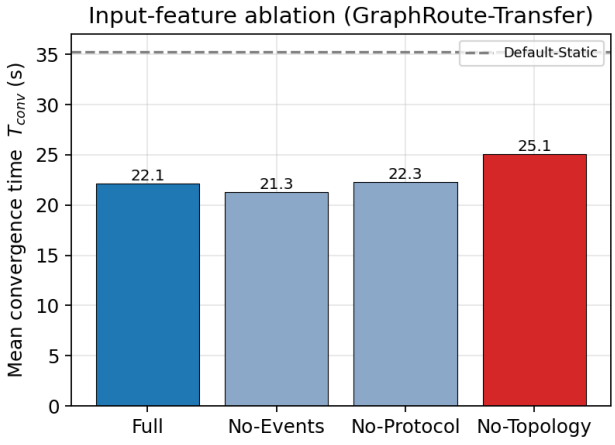

Fig. 6 Input-feature ablation. Topology features are the most important input to the per-node policy.

7. Discussion and limitations

Our results support a simple thesis: routing-timer optimization is a graph problem, and a size-invariant graph policy captures per-node structure that a global agent cannot, yielding both better configurations and—uniquely—zero-shot transfer across network sizes. The oracle-guided scheme makes training stable and casts the method as amortized optimization.

Limitations. First, like DRL-Adapt and much of the learned-networking literature, our evaluation is simulation-based; although the simulator implements the standard convergence decomposition and a physically-motivated flap process calibrated on real topologies, validation on testbeds or production traces is future work. Second, the flap and fragility model, while grounded in operational experience [7], is an abstraction; incorporating measured jitter and control-plane load would sharpen realism. Third, we optimize a one-shot static per-node configuration; extending the graph policy to reconfigure online in response to observed events is a natural next step, for which the per-node value head we already train provides a foundation. Finally, we tune Hello/Dead timers; the same per-node graph formulation could target SPF-delay and LSA-throttle timers or BGP parameters [3].

8. Conclusion

We presented GraphRoute-Transfer, a graph-neural-network policy that assigns per-node routing-convergence timers and, by construction, transfers zero-shot across topologies of arbitrary size. Building on the DRL timer-optimization framing of DRL-Adapt [1] but replacing its flat global agent with a size-invariant graph policy, and training it to amortize a convergence-cost search oracle, we obtain a 37.3% convergence-time reduction over the OSPF default and a matched flat DRL baseline on 231 real Internet Topology Zoo networks, near-oracle cost at roughly 8,000× lower configuration time, and robust generalization to networks twice the largest training size—where the flat baseline cannot adapt. The results argue for treating network-control policies as graph-native and size-invariant by design.

References

[1] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., & Hassabis, D. (2016). Mastering the game of Go with deep neural networks and tree search. Nature, 529(7587), 484-489. https://doi.org/10.1038/nature16961

[2] Knight, S., Nguyen, H. X., Falkner, N., Bowden, R., & Roughan, M. (2011). The Internet topology zoo. IEEE Journal on Selected Areas in Communications, 29(9), 1765-1775. https://doi.org/10.1109/JSAC.2011.111002

[3] Moy, J. (1998). OSPF version 2 (RFC 2328). IETF.

[4] Rekhter, Y., Li, T., & Hares, S. (2006). A Border Gateway Protocol 4 (BGP-4) (RFC 4271). IETF.

[5] Labovitz, C., Ahuja, A., Bose, A., & Jahanian, F. (2001). Delayed Internet routing convergence. IEEE/ACM Transactions on Networking, 9(3), 293-306. https://doi.org/10.1109/90.929852

[6] Francois, P., Filsfils, C., Evans, J., & Bonaventure, O. (2005). Achieving sub-second IGP convergence in large IP networks. ACM SIGCOMM Computer Communication Review, 35(3), 35-44. https://doi.org/10.1145/1070873.1070880

[7] Basu, A., & Riecke, J. G. (2001). Stability issues in OSPF routing. In Proceedings of the ACM SIGCOMM Conference (pp. 225-236). ACM. https://doi.org/10.1145/383059.383075

[8] Griffin, T. G., & Premore, B. J. (2001). An experimental analysis of BGP convergence time. In Proceedings of the IEEE International Conference on Network Protocols (ICNP) (pp. 53-61). IEEE. https://doi.org/10.1109/ICNP.2001.992887

[9] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., & Hassabis, D. (2015). Human-level control through deep reinforcement learning. Nature, 518(7540), 529-533. https://doi.org/10.1038/nature14236

[10] Wang, B., Wang, Z., Zhao, W., Zhang, F., & Shang, W. (2026). DRL-Adapt: Deep reinforcement learning for adaptive routing convergence optimization in large-scale networks. IEEE Open Journal of the Computer Society, 7, 1-14. https://doi.org/10.1109/OJCS.2026.3687441

[11] van Hasselt, H., Guez, A., & Silver, D. (2016). Deep reinforcement learning with double Q-learning. In Proceedings of the AAAI Conference on Artificial Intelligence (Vol. 30, No. 1, pp. 2094-2100). AAAI Press.

[12] Wang, Z., Schaul, T., Hessel, M., van Hasselt, H., Lanctot, M., & de Freitas, N. (2016). Dueling network architectures for deep reinforcement learning. In Proceedings of the 33rd International Conference on Machine Learning (ICML) (pp. 1995-2003). PMLR.

[13] Schaul, T., Quan, J., Antonoglou, I., & Silver, D. (2016). Prioritized experience replay. In Proceedings of the International Conference on Learning Representations (ICLR).

[14] Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., & Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In Proceedings of the 33rd International Conference on Machine Learning (ICML) (pp. 1928-1937). PMLR.

[15] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. arXiv, arXiv:1707.06347. https://doi.org/10.48550/arXiv.1707.06347

[16] Sutton, R. S., McAllester, D., Singh, S., & Mansour, Y. (2000). Policy gradient methods for reinforcement learning with function approximation. In Advances in Neural Information Processing Systems 13 (NIPS) (pp. 1057-1063).

[17] Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., & Dahl, G. E. (2017). Neural message passing for quantum chemistry. In Proceedings of the 34th International Conference on Machine Learning (ICML) (pp. 1263-1272). PMLR.

[18] Kipf, T. N., & Welling, M. (2017). Semi-supervised classification with graph convolutional networks. In Proceedings of the International Conference on Learning Representations (ICLR).

[19] Hamilton, W. L., Ying, R., & Leskovec, J. (2017). Inductive representation learning on large graphs. In Advances in Neural Information Processing Systems 30 (NeurIPS) (pp. 1024-1034).

[20] Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., & Bengio, Y. (2018). Graph attention networks. In Proceedings of the International Conference on Learning Representations (ICLR).

[21] Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., & Monfardini, G. (2009). The graph neural network model. IEEE

Transactions on Neural Networks, 20(1), 61-80. https://doi.org/10.1109/TNN.2008.2005605

[22] Bengio, Y., Lodi, A., & Prouvost, A. (2021). Machine learning for combinatorial optimization: A methodological tour d'horizon. European Journal of Operational Research, 290(2), 405-421. https://doi.org/10.1016/j.ejor.2020.07.063

[23] Teng, D., Rhee, M., Qin, Y., Zi, B., & Liu, W. (2026). SW-SpeedDLM: Sliding-window speculative decoding for diffusion language models under long-context constraints. Mathematics, 14(12), Article 2137. https://doi.org/10.3390/math14122137

[24] Wang, Z., Yang, J. S., Shang, W., & Ding, J. (2026). FairPromote: Explainable and fairness-aware talent promotion prediction via adversarial debiasing and SHAP-based interpretation. IEEE Access, 14, 1-16.

[25] Teng, D. (2025). TEAS: Token- and energy-aware autoscaling for cost-efficient LLM serving. AI and Data Science Journal, 6(3), 1-10.

[26] Zhang, F., Guo, Z., Ding, J., Yang, J., & Liu, W. (2026). Adaptive sensor fusion for robust perception in dense fog: A gated vision and LiDAR integration framework. Sensors, 26(12), Article 3728. https://doi.org/10.3390/s26123728

[27] Zi, B. (2024). Large language models for enterprise workflow automation in financial operations. Innovation and Technology Studies, 1(1), 24-29.

[28] Ding, J., Shen, Z., & Liu, W. (2026). Game-theoretic cost-sensitive adversarial training for robust cloud intrusion detection against GAN-based evasion attacks. Applied Sciences, 16(8), Article 3944. https://doi.org/10.3390/app16083944

[29] Zi, B. (2024). Cloud-native distributed systems for real-time payment intelligence. AI and Data Science Journal, 1(1), 51-56.

[30] Liang, Y., Jiao, Y., Ping, W., Fan, H., & Han, X. (2026). Adaptive event-driven labeling: A neuro-symbolic multiagent framework for causal inference in non-stationary time series. IEEE Access, 14, 1-18.