TASK-AWARE COMMUNICATION FOR DISTRIBUTED LLM AGENT SYSTEMS

Anonymous authors
Paper under double-blind review

ABSTRACT

Multi-agent systems (MAS) powered by large language models increasingly coordinate reasoning, tool use, and data processing across distributed environments, where task completion time depends largely on the quality of network connectivity. Existing MAS frameworks focus on managing task workflow and resources on the application level, overlooking the opportunities for performance optimization in the underlying network substrate. Building on the emerging path-aware networking paradigm, we propose *FlowAhead*, a framework that leverages task information and network measurements to choose optimal paths for inter-agent data transfers. Our evaluation shows that *FlowAhead* reduces the median task completion time of a three-agent video-reasoning workflow by 34.5% relative to default BGP routing, and by 51.5% without site-access limits, demonstrating a promising path toward improving efficiency for distributed MAS.

1 INTRODUCTION

Large language model (LLM) services are evolving from standalone chatbots into multi-agent systems (MAS) that coordinate tool-using agents to complete complex tasks. These systems decompose tasks into subtasks, assign them to specialized agents, and integrate their intermediate results through communication and coordination (Wu et al., 2024; Hong et al., 2024; Kim et al., 2024). This shift from single-agent to multi-agent execution brings network efficiency into focus: a single user task can involve hundreds of agent-to-agent messages as well as handoffs of bulky intermediate state, such as images, generated artifacts, and KV caches, across organizational and geographic boundaries; transferring such data over suboptimal or inferior network paths will make communication cost a dominating factor in the overall task completion time.

A growing body of work exploits task information to manage resources across agentic workflows. Parrot (Lin et al., 2024) uses semantic variables to expose data dependencies among LLM requests, enabling joint scheduling for end-to-end application performance. Agentix (Luo et al., 2026) tracks program-level execution context and uses it to prioritize and preempt LLM calls, reducing the completion time of agentic programs. PASTE (Sui et al., 2026) predicts forthcoming tool invocations from recurring workflow patterns and executes them speculatively alongside LLM generation, while PBKV (Zheng et al., 2026) predicts future agent invocations and uses their expected KV-cache reuse to guide cache eviction and prefetching. HAFS (Lee et al., 2026) uses flow sizes and predicted decoding times to schedule concurrent KV transfers. None of existing systems explores using task information to select network paths for optimizing agentic communication. Since available paths can differ substantially in latency, bandwidth, reliability, and cost, task-aware path selection presents an important yet underexplored opportunity for improving end-to-end agentic task performance.

Taking a first step in this direction, we introduce *FlowAhead*, a middleware layer that assigns inter-agent transfers to network paths based on the current workflow state. It builds on SCION (Chuat et al. (2022)), a next-generation path-aware Internet architecture that exposes network paths to applications and allows them to choose among these paths. At each scheduling decision, *FlowAhead* observes the task dependencies revealed so far, the data objects that are ready or in transit, and the available paths. Starting from a baseline assignment based solely on network conditions, *FlowAhead* evaluates alternatives by estimating when each object will arrive and when each downstream agent will have all the inputs needed to proceed. It accounts for contention among transfers sharing a bottleneck and adopts a new assignment only if it reduces the affected agents' average remain-

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

ing input-waiting time. This allows *FlowAhead* to reduce communication-induced stalls as the task unfolds, without requiring the complete workflow to be known in advance

We deploy two MAS workflows, long-video question answering with A4VL (Xu et al., 2026) and multi-agent debate (Du et al., 2024), and compare *FlowAhead* with BGP routing and random path selection over the same candidate paths, replaying identical model calls under every policy. We measure end-to-end task completion time and network time while varying background load, site-access capacity, workload type, and team size. At 25% network load with 100 Mbit/s site access, *FlowAhead* reduces the task completion time of three-agent A4VL by a median of 34.5% and its network time by a median of 56.4%; Debate's network time falls by the same fraction, but its completion time by only 5.9%, because transfers make up a much smaller share of it. At 35% load, the 90th-percentile completion time of three-agent A4VL falls by 35.9%. Without the access limit, its median reduction at 25% load grows to 51.5%. The results demonstrate the effectiveness of *FlowAhead* over different task types and network settings.

Contributions: (1) **Task-aware formulation of cross-domain transfers.** We model MAS execution as a task graph and relate the arrival of each data transfer to the readiness of its receiving tasks and to the overall task completion time. (2) **Middleware layer for task-aware path selection.** Introducing *FlowAhead* between the agent orchestrator and the network to select paths by considering network conditions and MAS information. (3) **Evaluation of distributed MAS scenarios,** demonstrating robust performance in diverse agent tasks, network topologies, and operating conditions.

2 Problem Formulation

We consider a distributed multi-agent system (MAS) whose agents run at fixed sites and pass intermediate objects to one another. An orchestrator decides which agents run, how they depend on earlier results, and when a branch is cancelled or retried. It does not carry the object payloads. Those bytes move directly from the producing agent to the consuming agent. Agent placement, payload content, and dependency semantics remain fixed by the application. The network can only change when a required object reaches its consumer.

2.1 Online Task Graph and Handoffs

At time t, the execution of MAS is represented by the directed graph G_t:

$$G_t = (V_t, E_t), \qquad s : V_t \to \mathcal{S}, \qquad e = (u, v, o_e) \in E_t. \tag{1}$$

Each node $v \in V_t$ denotes a single agent or tool invocation executed at site $s(v)$. An edge $e = (u, v, o_e) \in E_t$ represents a potential data dependency: invocation v may consume a specific version o_e of an object produced by invocation u. Because the execution graph is revealed incrementally, E_t contains only the dependencies known by time t, rather than all dependencies that may arise during the remainder of the task. We define r_e as its release time and B_e as its useful-byte size. We define $A_e \geq r_e$ as the earliest time at which v can use o_e. The receiver records A_e only after receiving and validating the complete committed version. Hence, $A_e - r_e$ is the delay from object release to availability at the consumer. This delay may be negligible when u and v are colocated.

We write $\mathcal{F}_t$ for the information available to an online policy. It contains the graph and branch state revealed so far, metadata for committed objects, and network observations collected by time t. It does not contain unrevealed branches, release times or sizes that have not yet been observed, or future path conditions. Values inferred for those quantities are predictions.

2.2 Task Completion Objective

Let $\mathrm{Req}(v)$ denote the set of input objects required by invocation v, and let q_v be the earliest time at which the MAS can launch invocation v. When invocation v requires all of its input objects, it becomes ready only after the MAS has admitted it and every required input has arrived. Thus, the earliest readiness time when all required inputs are available is

$$R_v = \max\left\{q_v, \max_{e \in \mathrm{Req}(v)} A_e\right\}. \tag{2}$$

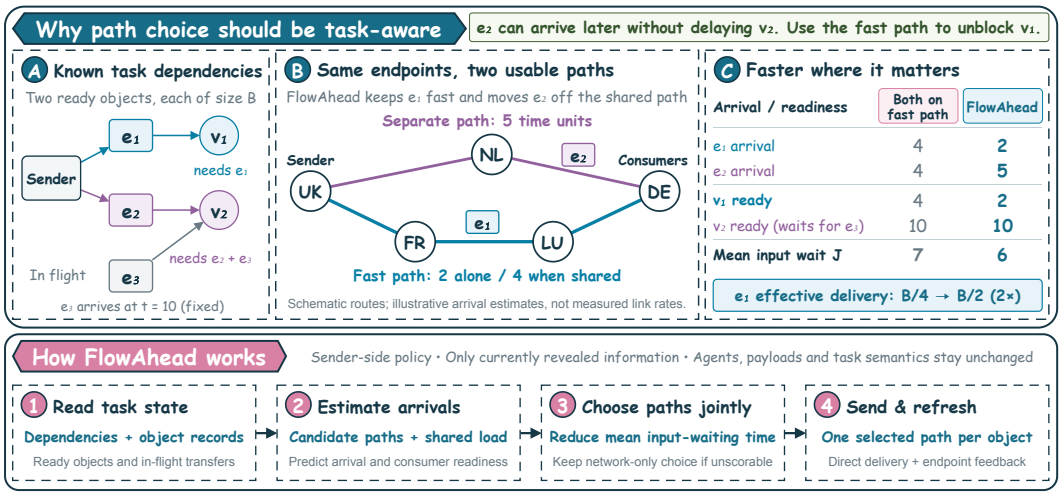

Figure 1: Overview of *FlowAhead* with an illustration of task-aware path selection.

If invocation v has no input objects, the inner maximum is omitted, making $R_v = q_v$. When the application selects one input from several alternatives, v must wait for the selected input. The network may not replace it with whichever object arrives first. Once invocation v is ready, it may still wait for resources and spend time computing before completion. Let F_v denote the completion time of invocation v. Let a_w denote the admission time of workflow w, and let $\mathrm{sinks}(w)$ denote the set of invocations that produce the final outputs of w. The task completion time of workflow w is

$$T_w = \max_{v \in \mathrm{sinks}(w)} F_v - a_w. \tag{3}$$

We choose paths authorized by $\mathcal{F}_t$ to reduce task completion time. The optimization accounts for downstream dependencies and shared-path contention while preserving endpoints, object versions, branch choices, and required inputs.

3 Task-Aware Path Selection over SCION

FlowAhead introduces an endpoint policy for inter-agent object transfers. As shown in Figure 1, it uses task dependencies from the MAS orchestrator to evaluate how candidate path assignments affect downstream input readiness and selects paths accordingly.

3.1 Task and Network Interface

The path-selection policy operates between the MAS orchestrator and the SCION network layer. The orchestrator exposes the dependencies that are currently eligible for transmission, while the network layer provides the paths that the sender is authorized to use together with their latest endpoint measurements. Based on this information, the policy assigns one path to each ready object at the local sender without changing the application's execution semantics.

On the task side, the policy receives the current graph snapshot G_t and the sender's object records. The orchestrator reports invocation, dependency, and branch events. At time t, $\mathcal{U}_t$ contains the local sender's committed objects whose payloads are ready but have not yet been assigned to a path. For each dependency e represented by an object in $\mathcal{U}_t$, the sender knows the versioned object identity, producer, consumer, release time $r_e \le t$, and size B_e. It also records objects already in flight, their assigned paths, and their acknowledged remaining bytes. Arrival times of completed transfers are observed rather than estimated. These records follow the execution semantics in Section 2.1.

On the network side, the policy receives the current SCION path handles and endpoint observations for each destination. SCION (Zhang et al., 2011) separates path construction and lookup from packet forwarding, allowing a source to choose a policy-compliant path and carry the corresponding forwarding information in its packets. The adapter removes expired or locally prohibited paths

and checks transport constraints such as the path MTU. Each remaining candidate includes its interface sequence, validity information, and measurement timestamp. When scoring a candidate joint assignment, the policy assumes that a SCION-facing adapter can estimate the remaining reception time for each ready or in-flight object, including effects on known in-flight transfers. It adds a receiver-provided validation-time estimate when needed and uses the resulting availability times to evaluate task outcomes.

Within the local sender's architecture, the policy module maps the current task state and candidate SCION paths to a path configuration $\mathbf{p} = (p_e)_{e \in \mathcal{U}_t}$, where $p_e \in \mathcal{P}_e(t)$ is the SCION path handle selected for the object associated with ready dependency e. The selected handle is passed to the transport adapter for packet forwarding. Different objects may use different paths concurrently, but the base policy neither stripes a single object across multiple paths nor sends redundant copies. The policy changes only the path used for transmission. It does not modify the agents, object contents, or application-selected dependencies.

3.2 Using MAS and Network Dependencies for Path Selection

Task dependencies identify which objects a consumer requires, but can not reveal when an unfinished agent will produce another input. *FlowAhead*'s path-selection policy uses these dependencies to compare the input-waiting consequences of candidate path assignments. The base rule applies when the required objects have already been committed and their availability can be observed or estimated by the network-side interface. For a ready or in-flight object e, the network-side interface supplies $\widehat{L}_e(\mathbf{p} \mid \mathcal{F}_t)$, the estimated remaining delay to consumer availability under the joint assignment $\mathbf{p}$. At decision time t, its predicted availability time is

$$\widehat{A}_e(\mathbf{p}) = t + \widehat{L}_e(\mathbf{p} \mid \mathcal{F}_t). \tag{4}$$

Completed transfers use observed availability times. The adapter evaluates concurrent transfers together, so a new path assignment may change the estimate for an in-flight object even though that object's path remains fixed.

We design an adapter to estimate how shared bottlenecks affect transfer rates and completion times. It uses path information and endpoint measurements to group transfers by their dominant shared bottleneck, then approximates contention by assuming equal throughput sharing within each group. Let $\mathcal{I}_t$ denote the known in-flight transfers, whose paths remain fixed, and let $g_e(\mathbf{p})$ denote the bottleneck group of object e under assignment $\mathbf{p}$. For each object, b_e is the remaining useful bytes and $\widehat{d}_e(p_e)$ is the remaining connection, propagation, and validation overhead on its selected or fixed path p_e, excluding the byte-transfer time accounted for by $b_e/\widehat{r}_e(\mathbf{p})$. The group's single-flow effective throughput, $\widehat{c}_g > 0$, is estimated from endpoint measurements in bytes per second. For each candidate assignment, the adapter holds this throughput and the corresponding group counts fixed over the prediction interval and estimates the completion times as

$$n_g(\mathbf{p}) = \sum_{j \in \mathcal{U}_t \cup \mathcal{I}_t} \mathbf{1}\{g_j(\mathbf{p}) = g\}, \widehat{r}_e(\mathbf{p}) \quad = \frac{\widehat{c}_{g_e(\mathbf{p})}}{n_{g_e(\mathbf{p})}(\mathbf{p})}, \widehat{L}_e(\mathbf{p} \mid \mathcal{F}_t) = \widehat{d}_e(p_e) + \frac{b_e}{\widehat{r}_e(\mathbf{p})}. \tag{5}$$

Here $b_e = B_e$ for a ready object, while for an in-flight object b_e is the remaining useful-byte count determined from acknowledgments. A new assignment changes the number of transfers sharing each bottleneck and thus the predicted rates of both ready and in-flight objects.

Before comparing candidates, the policy fixes a set $\mathcal{C}_t$ of known, not-yet-started consumers whose input arrivals may be affected by any candidate, including through contention with in-flight transfers. Their application admission conditions must already be satisfied, their complete required-input sets must be known, and all required objects must have been committed. If any potentially affected consumer lacks this information or still awaits an uncommitted output, the policy retains the network-only assignment for this decision. It does not remove that consumer from the comparison and treat the unknown cost as zero. For each $v \in \mathcal{C}_t$, Equation 2 then gives the predicted time at which its inputs will be ready from the current decision onward:

$$\widehat{R}_v(\mathbf{p}) = \max\left\{t, \max_{e \in \mathrm{Req}(v)} \widehat{A}_e(\mathbf{p})\right\}. \tag{6}$$

The maximum captures the effect of dependencies: advancing one object does not unblock v while another required input arrives later. This quantity measures input readiness; resource queues and

computation may delay actual execution further. A candidate is left unscored if any required arrival estimate is unavailable, while $\mathcal{C}_t$ remains fixed. For a nonempty $\mathcal{C}_t$, the policy compares the mean remaining input-waiting time:

$$J_t(\mathbf{p}) = \frac{1}{|\mathcal{C}_t|} \sum_{v \in \mathcal{C}_t} \big(\widehat{R}_v(\mathbf{p}) - t\big). \tag{7}$$

A lower score indicates earlier input readiness, but not necessarily faster task completion. We therefore also measure how often the comparison is feasible and whether it reduces completion time.

Starting from the assignment returned by a network-only policy, the task-aware selector retains at most K authorized candidates per ready object. It repeatedly evaluates single- and two-object path changes from the current assignment and accepts the best fully scored neighbor only if it reduces J_t by more than the configured tolerance. The search stops when no fully scored neighbor meets this condition or when its score or time budget is exhausted. Algorithm 1 in Appendix B.1 specifies the full selection logic.

4 Evaluation

We organize our empirical evaluation around three research questions:
RQ1 (Background load). Can *FlowAhead* reduce end-to-end MAS task completion time relative to default routing, and how does the benefit change with background load?
RQ2 (Workload sensitivity). How does the performance benefit of path selection vary across MAS workflows with different communication patterns?
RQ3 (Collaboration scale and access capacity). As the number of collaborating agents grows, how do team size and site-access capacity affect the benefit of *FlowAhead*?

4.1 Experimental Settings

Benchmarks. We use two published multi-agent workflow benchmarks to cover both large video transfers and small text exchanges. For the video workload, we adapt A4VL (Xu et al., 2026) to a fixed subset of 20 long-video question-answering tasks from Video-MME (Fu et al., 2025), with video payloads ranging from 85 to 914 MB. For the text workload, we adapt multiagent debate (Du et al., 2024) to a fixed subset of 100 GSM8K math questions (Cobbe et al., 2021). We run each task with different number of agents on the same task inputs. The six-agent versions extend the published workflows rather than reproduce their original experimental configurations. In particular, our A4VL variant builds on the authors' three-agent implementation using their configurations.

Network topology. We vary background demand and site-access capacity to examine how these constraints affect the evaluated network policies. The backbone is an emulated 22-PoP, 33-link subgraph derived from the GEANT 2012 topology in the Internet Topology Zoo (Knight et al., 2011). We assign 10 Gbit/s to 31 links and 2.5 Gbit/s to the remaining two. One-way propagation delay is calculated from great-circle distance using a path-stretch factor of 1.5 and a propagation rate of 5 μs/km. In the main setting, each agent site connects to its city PoP through a 100 Mbit/s access link. An idealized sensitivity setting removes this access cap but keeps the backbone and task inputs unchanged. To distribute background demand across the network, we use the SNDlib GEANT demand matrix (Orlowski et al., 2010), which was derived from historical NetFlow and routing measurements (Uhlig et al., 2006). We treat it as a spatial traffic template, not as a contemporaneous measurement of the 2012 topology. Demands follow the modeled default shortest-hop routes and are scaled to four nominal background-load levels of 0%, 10%, 25%, and 35%. On each oversubscribed link, we scale routed demand down by the ratio of capacity to arrival rate and repeat this adjustment until the background load stabilizes. These levels represent constructed pressure scenarios rather than measured GEANT operating states. We do not transmit background packets in the emulation.

Effective link model and transport. We represent background demand through effective transfer rates and path delays. For a directed link ℓ with capacity C_ℓ and modeled background arrival rate D_ℓ, the per-flow effective-rate parameter is $R_\ell = \max(C_\ell - D_\ell, 0) + 20\,\mathrm{Mbit/s}$. The additive term applies even when demand is below capacity. It is a tunable abstraction motivated by the established dependence of TCP throughput on loss and RTT (Mathis et al., 1997; Floyd, 2008; Energy Sciences Network (ESnet), 2024), rather than a value fitted to GEANT. A path's rate is the minimum of its

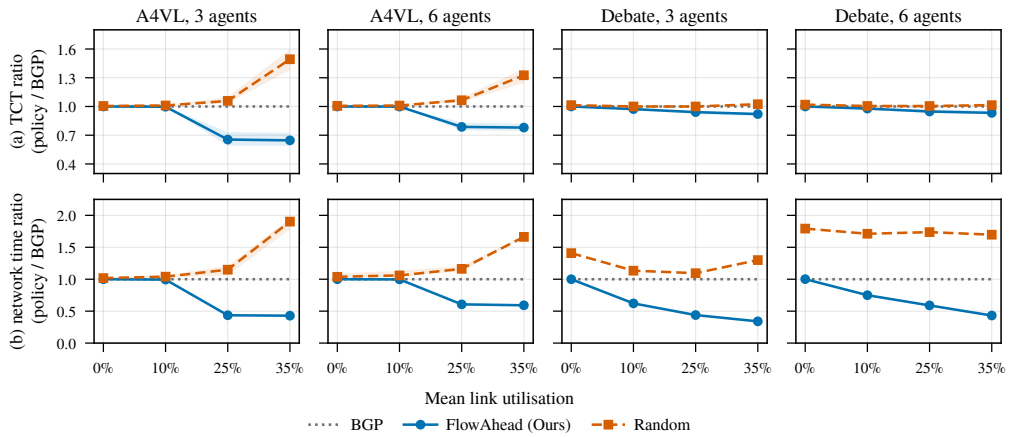

Figure 2: Paired per-task ratios to BGP versus background load under 100 Mbit/s: medians (markers) and interquartile ranges (bands) for task completion time (top) and network time (bottom).

link rates and, in the main setting, the 100 Mbit/s site-access rate. The no-cap setting removes only the access limit. One-way path delay is the sum of the modeled propagation delays, with an additional 50 ms for each saturated link. The delay penalty is likewise a tunable queueing abstraction, motivated by the bandwidth–delay-product basis of conventional router-buffer sizing (Appenzeller et al., 2004). Both the additive 20 Mbit/s term and the fixed 50 ms penalty are explicit model parameters, not measured GEANT quantities. Mininet (Lantz et al., 2010) applies these rates and delays through Linux `tc` HTB and netem, and application payloads are carried by TCP CUBIC (Ha et al., 2008). We configure no packet-loss process in netem; the emulator therefore omits the loss and retransmission effects observed on congested Internet and interconnection paths (Dhamdhere et al., 2018; Measurement Lab Consortium, 2014).

Policies and paired-replay protocol. We compare 3 policies over the same candidate paths for each site pair. BGP is the modeled default route, which selects the fewest-hop path. At the network layer, *FlowAhead* predicts each path's transfer time as the RTT plus the object size divided by the available bandwidth and selects the path with the smallest predicted transfer time, following the endpoint path-choice model of SCION (Appenzeller et al., 2004). Operational SCIERA measurements independently demonstrate that selecting among SCION paths can reduce path RTT (Wirz et al., 2025). The random-path control draws uniformly from the same candidates using one fixed seed, and every candidate set includes the BGP route. We record model calls once and replay them under all three policies, running the policies for each task consecutively on the same machine. Except for the absolute-median scaling comparison in RQ3, ratios are computed per task relative to the paired BGP replay and then summarized across tasks. For task-level outcome counts, a completion-time difference within the larger of 0.5% and 0.1 s is treated as equal to absorb replay jitter.

4.2 RQ1: When Does Path Selection Reduce Task Completion Time?

We first evaluate if network path choices can significantly affect the completion of distributed MAS tasks as modeled background load rises. Figure 2 summarizes the median per-task task-completion-time and summed network-time ratios to BGP across all four workloads and load levels. Figures 8 and 9 report the full absolute distributions, while Figures 10 and 11 report paired per-task ratios. Site access is limited to 100 Mbit/s in these experiments.

The experiments reveal four findings. First, *FlowAhead* substantially reduces network time compared with BGP and random selection under heavy background load. At 25% load, both A4VL and Debate reduce network time by 56.4% versus BGP with three agents. In contrast, random selection has median ratios above one for both workloads at the same load. Second, path selection provides little improvement in task completion when there is no background traffic. At zero background load, the median ratio of task completion time to that under default routing is about one for all four workloads. Third, the reduction in task completion time becomes more pronounced as background load increases. At 25% load, median reductions relative to BGP reach 34.5% for A4VL with three agents

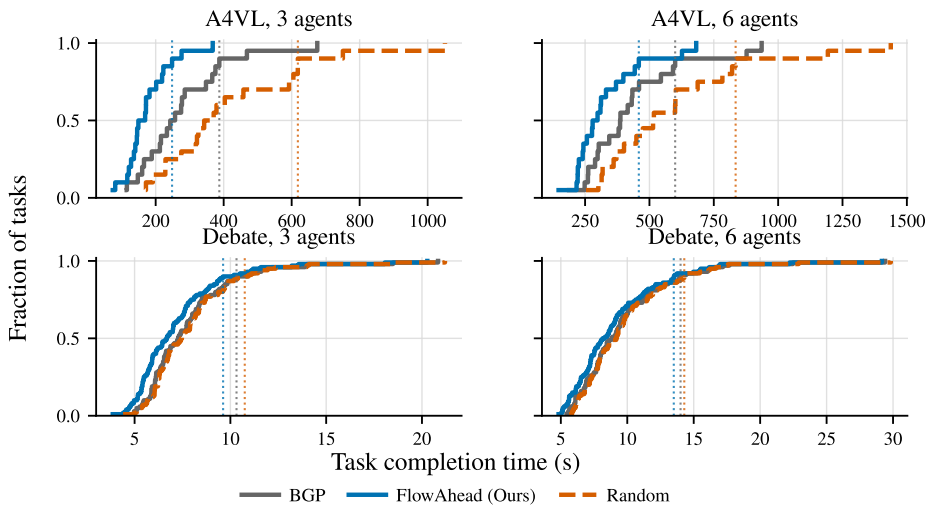

Figure 3: Empirical CDFs of task completion time at 35% background load under 100 Mbit/s site access; dotted vertical lines mark each policy's 90th percentile.

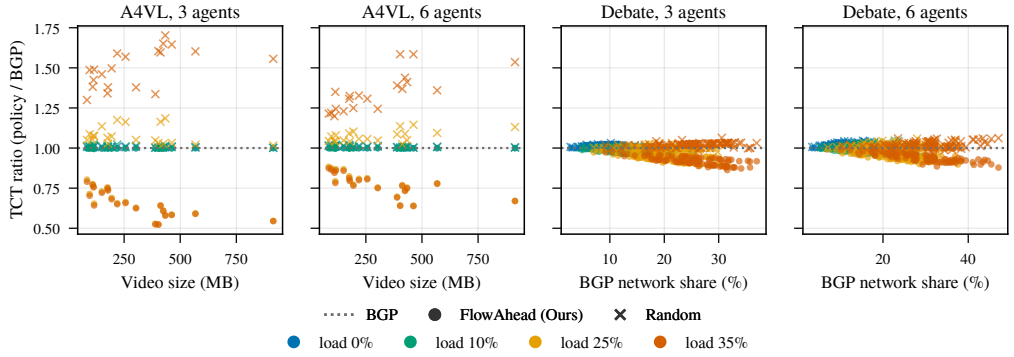

Figure 4: Per-task TCT ratios to paired BGP replays versus video size (A4VL) or BGP network-time share (Debate), under 100 Mbit/s site access.

and 21.3% with six, increasing to 35.4% and 22.0% at 35% load. Over the same load increase, Debate's median reductions grow from 5.9% to 7.9% with three agents and from 5.2% to 6.7% with six. Fourth, the improvement extends beyond the median to tail task latency. At 35% load, Figure 3 shows that the 90th-percentile TCT falls from 387.2 to 248.2 s for three-agent A4VL and from 599.9 to 458.7 s for six-agent A4VL, reductions of 35.9% and 23.5%. The corresponding reductions for Debate are smaller, at 6.8% and 3.6%, foreshadowing the workload sensitivity examined next.

4.3 RQ2: Which MAS Workloads Are Sensitive to Path Choice?

We next evaluate how the benefit of path selection differs between MAS tasks and workloads. This comparison is needed because a reduction in network time may have different effects on task completion depending on how much the task relies on communication. We compare A4VL, which transfers large videos, with Debate, which exchanges short text messages, using the same network settings as in RQ1. Figure 4 shows each task's completion-time ratio to its paired BGP replay under all three policies. To examine why the benefit varies across tasks, we plot this ratio against video size for A4VL and communication share for Debate.

The results illustrate that tasks with larger communication demands tend to achieve larger reductions in TCT. At 25% and 35% background load, A4VL tasks with larger videos generally have lower TCT ratios in Figure 4. A similar pattern appears in Debate, where tasks with a larger communication

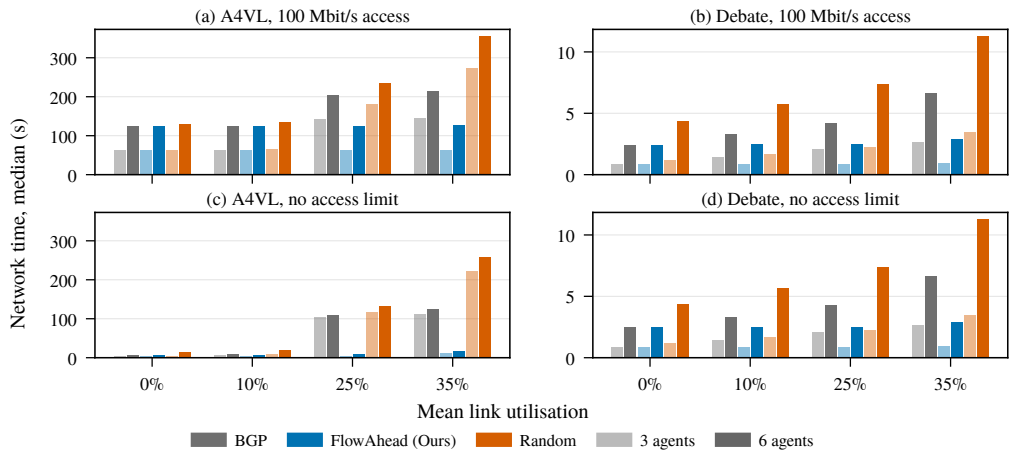

Figure 5: Median per-task summed transfer time versus background load for three- and six-agent teams: 100 Mbit/s site access (top) and no access limit (bottom).

share tend to benefit more despite their small message sizes. The completion-time decomposition in Figure 7 shows that A4VL spends substantially more absolute time transferring data than Debate, explaining why comparable network-time reductions translate into different task-completion-time gains. At 0% and 10% load, however, A4VL points remain close to one across the range of video sizes. The benefit therefore depends on background load as well as communication intensity.

4.4 RQ3: Does Path Selection Remain Effective as Collaboration Grows?

Finally, we evaluate whether path selection continues to reduce network time as more agents collaborate, and whether site-access capacity limits this benefit. Larger teams introduce additional data exchanges, so the benefit observed with a small team may not carry over. We compare BGP, *FlowAhead*, and random selection using three and six agents. Figure 5 reports median network time with 100 Mbit/s site access and without access limit. The two settings use the same tasks and network setup. Network time is the sum of transfer durations within each task, counting overlapping transfers separately rather than measuring TCT. The percentage reductions reported for this figure are computed from the plotted policy medians, rather than as medians of per-task ratios.

We find that path selection continues to reduce network time as teams grow from three to six agents, though the relative gains are smaller when site access is limited. At 25% background load, the medians in Figure 5 show that A4VL's network time is 56.3% lower than under BGP with three agents and 39.0% lower with six. The corresponding reductions for Debate are 56.4% and 41.0%. Both kinds of agent tasks therefore benefit from *FlowAhead* at both team sizes.

Removing the access limit brings A4VL's gains with three and six agents much closer together. At the same 25% background load, Figure 5(c) shows network-time reductions relative to BGP of 96.2% with three agents and 92.4% with six. The gap falls from 17.3 to 3.8 percentage points. At the task level, the complete trends in Figure 13 show that removing the access limit increases A4VL's median TCT reduction at 25% load from 34.5% to 51.5% with three agents and from 21.3% to 35.7% with six agents. The full absolute TCT and network-time distributions are reported in Figures 15 and 16. Debate remains nearly unchanged, at 5.9% and 5.2% for three and six agents, respectively. Across both access settings and all 1,920 paired task–load comparisons, the selector is faster in 1,413 cases, equal within the stated tolerance in 507, and never slower. This shows that site access capacity limits how much video transfers can benefit from path selection. The smaller relative gain with six agents does not, by itself, mean that path selection becomes ineffective as teams grow. This difference suggests that the two workloads face different communication bottlenecks. A4VL's large video transfers are more sensitive to path bandwidth, while Debate's small messages are more sensitive to latency over repeated exchanges.

5 Related Work

Workflow-aware agent and network control. Workflow-aware optimization is studied in both LLM serving and communication networks. Parrot (Lin et al., 2024) and Agentix (Luo et al., 2026) optimize execution at the application or program level. Aalo (Chowdhury & Stoica, 2015) addresses coflows with unknown sizes, while Siphon (Liu et al., 2018) combines dependency-aware WAN scheduling with routing. HAFS (Lee et al., 2026) links the ordering of inter-agent KV transfers to receiver decoding time. ASGE-RR (Vatten & Jiang, 2026) selects service replicas and request/response paths for agent calls revealed at runtime. It uses predicted workflow continuations and revisable soft reservations to preserve capacity for future calls. We consider necessary object handoffs between fixed agent endpoints, with path selection limited to each sender's committed, ready objects. Unlike ASGE-RR, our policy neither selects replicas nor maintains a capacity calendar. It instead estimates and improve network quality from endpoint observations.

Path-aware networking. Path-aware networking spans path exposure and transport-level control. SCION (Zhang et al., 2011) exposes authorized inter-domain paths to endpoints. Polaris (Ehsani Moghadam et al., 2025) uses network feedback for path selection, while DMTP (John et al., 2023) and UMCC (Gartner & Hausheer, 2024) develop deadline-aware multipath transport and shared-bottleneck-aware congestion control, respectively. These systems provide path or transport mechanisms, but do not choose joint object-to-path assignments based on an MAS dependency graph. Our policy uses the existing path interface and delegates congestion control and reliable delivery to the transport. It propagates network-side arrival estimates along revealed task chains and across consumers waiting for multiple inputs. It then scores assignments by predicted workflow outcomes rather than by independent path or flow scores.

6 Discussion

When path selection matters. Our results show that adding more agents does not, by itself, make path selection more beneficial. The available network paths must differ enough to offer a useful choice. With no background load, the median TCT ratio stays close to one across all four workloads, leaving little room for improvement. Faster delivery can speed up the task considerably when communication accounts for a non-trivial share of execution time. Where that communication occurs in the dependency graph matters just as much. An earlier object arrival reduces TCT only if it advances a downstream readiness event that constrains final completion. Even a workload with substantial total communication may gain little if agent computation or another required input at a join hides the improved transfers. Therefore, path diversity, communication intensity, and downstream dependencies jointly determine the effectiveness of path selection.

Why network-time gains do not translate proportionally into TCT gains. Our results on A4VL and Debate show that the same reduction in network time can have very different effects on task completion. At 25% background load, *FlowAhead* reduces median network time by 56.4% relative to BGP in both three-agent workloads. The corresponding median TCT reductions, however, are 34.5% for A4VL and only 5.9% for Debate. A4VL transfers large video objects, and waiting for their delivery accounts for a substantial delay before downstream processing can continue. Faster transfers therefore remove a meaningful part of its completion time. Debate exchanges much smaller messages, and computation and other waits account for a larger share of task execution. Reducing network time has less effect on its TCT in this case.

7 Conclusion

Distributed MAS can incur substantial task-level delay when required intermediate objects traverse heterogeneous and congested network paths. We formulate *FlowAhead*, a task-aware endpoint policy that uses currently revealed dependencies and network estimates to choose joint network path assignments for committed, ready objects without changing agent placement or application semantics. Our experiments show that path choice can materially affect distributed MAS completion time. Future work should isolate the incremental value of task information against a network-aware baseline with matched path and observation capabilities, and examine whether this benefit generalizes across broader workflow structures and real-world deployment settings.

AI USE STATEMENT

We used generative AI tools to assist with research planning, literature retrieval, code and data analysis, and manuscript drafting and editing. We did not use generative AI to generate the benchmark datasets. The authors reviewed all AI-assisted content and take full responsibility for the final manuscript, artifacts, and results.

ETHICS STATEMENT

This work uses published benchmarks and controlled network emulation; it does not involve a production deployment or newly recruited human participants. Because task metadata may reveal organizational relationships or workflow states, deployments and released traces should minimize sensitive information and enforce appropriate access controls. The method may also benefit harmful distributed-agent applications, so it should be used only for authorized workloads and networks.

REFERENCES

Guido Appenzeller, Isaac Keslassy, and Nick McKeown. Sizing router buffers. In *Proc. of ACM SIGCOMM*, pp. 281–292, 2004. doi: 10.1145/1015467.1015499.

Mosharaf Chowdhury and Ion Stoica. Efficient coflow scheduling without prior knowledge. In *Proc. of ACM SIGCOMM*, 2015.

Laurent Chuat, Markus Legner, David Basin, David Hausheer, Samuel Hitz, Peter Müller, and Adrian Perrig. *The Complete Guide to SCION. From Design Principles to Formal Verification*. Springer International Publishing AG, 2022. URL https://link.springer.com/book/10.1007/978-3-031-05288-0.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. arXiv preprint arXiv:2110.14168, 2021.

Amogh Dhamdhere, David D. Clark, Alexander Gamero-Garrido, Matthew Luckie, Ricky K. P. Mok, Gautam Akiwate, Kabir Gogia, Vaibhav Bajpai, Alex C. Snoeren, and kc claffy. Inferring persistent interdomain congestion. In *Proc. of ACM SIGCOMM*, 2018. doi: 10.1145/3230543.3230549.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B. Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate. In *Proc. of ICML*, 2024.

Elham Ehsani Moghadam, Patrick Wicki, François Wirz, Jordi Subirà Nieto, Yih-Chun Hu, and Adrian Perrig. Polaris: End-to-end path optimization by end hosts. In *Proc. of IEEE/ACM IWQoS*, 2025.

Energy Sciences Network (ESnet). Packet loss (Fasterdata knowledge base). Online, 2024. URL https://fasterdata.es.net/network-tuning/tcp-issues-explained/packet-loss/. Accessed September 25, 2026.

Sally Floyd. Measurement studies of end-to-end congestion control in the Internet. Online, 2008. URL https://www.icir.org/floyd/ccmeasure.html. Accessed September 25, 2026.

Chaoyou Fu, Yuhan Dai, Yongdong Luo, Lei Li, Shuhuai Ren, Renrui Zhang, Zihan Wang, Chenyu Zhou, Yunhang Shen, Mengdan Zhang, Peixian Chen, Yanwei Li, Shaohui Lin, Sirui Zhao, Ke Li, Tong Xu, Xiawu Zheng, Enhong Chen, Caifeng Shan, Ran He, and Xing Sun. Video-MME: The first-ever comprehensive evaluation benchmark of multi-modal LLMs in video analysis. In *Proc. of IEEE/CVF CVPR*, 2025.

Marten Gartner and David Hausheer. UMCC: Uncoupling multipath congestion control through shared bottleneck detection in path-aware networks. In *Proc. of IEEE LCN*, 2024.

Sangtae Ha, Injong Rhee, and Lisong Xu. CUBIC: A new TCP-friendly high-speed TCP variant. *ACM SIGOPS Oper. Syst. Rev.*, 42(5):64–74, 2008.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. MetaGPT: Meta programming for a multi-agent collaborative framework. In *Proc. of ICLR*, 2024.

Tony John, Adrian Perrig, and David Hausheer. DMTP: Deadline-aware multipath transport protocol. In *Proc. of IFIP Networking*, 2023.

Sehoon Kim, Suhong Moon, Ryan Tabrizi, Nicholas Lee, Michael W. Mahoney, Kurt Keutzer, and Amir Gholami. An LLM compiler for parallel function calling. In *Proc. of ICML*, 2024.

Simon Knight, Hung X. Nguyen, Nickolas Falkner, Rhys Bowden, and Matthew Roughan. The internet topology zoo. *IEEE J. Sel. Areas Commun.*, 29(9):1765–1775, 2011.

Bob Lantz, Brandon Heller, and Nick McKeown. A network in a laptop: Rapid prototyping for software-defined networks. In *Proc. of ACM HotNets*, 2010.

Goodsol Lee, Juheon Yi, Jinglu Wang, Haowen Xu, Saewoong Bahk, and Yan Lu. Coordinated networking for on-device agent-augmented real-time communication. arXiv preprint arXiv:2607.22854, 2026.

Chaofan Lin, Zhenhua Han, Chengruidong Zhang, Yuqing Yang, Fan Yang, Chen Chen, and Lili Qiu. Parrot: Efficient serving of LLM-based applications with semantic variable. In *Proc. of USENIX OSDI*, 2024.

Shuhao Liu, Li Chen, and Baochun Li. Siphon: Expediting inter-datacenter coflows in wide-area data analytics. In *Proc. of USENIX ATC*, 2018.

Michael Luo, Xiaoxiang Shi, Colin Cai, Tianjun Zhang, Justin Wong, Yichuan Wang, Chi Wang, Yanping Huang, Zhifeng Chen, Joseph E. Gonzalez, and Ion Stoica. Agentix: An efficient serving engine for LLM agents as general programs. In *Proc. of USENIX NSDI*, 2026.

Matthew Mathis, Jeffrey Semke, Jamshid Mahdavi, and Teunis Ott. The macroscopic behavior of the TCP congestion avoidance algorithm. *ACM SIGCOMM Comput. Commun. Rev.*, 27(3):67–82, 1997. doi: 10.1145/263932.264023.

Measurement Lab Consortium. ISP interconnection and its impact on consumer Internet performance. Technical report, Measurement Lab, 2014. URL `https://www.measurementlab.net/publications/M-Lab_Interconnection_Study_US.pdf`.

Sebastian Orlowski, Roland Wessäly, Michał Pióro, and Artur Tomaszewski. SNDlib 1.0—survivable network design library. *Networks*, 55(3):276–286, 2010.

Yifan Sui, Han Zhao, Rui Ma, Zhiyuan He, Hao Wang, Jianxun Li, Kaiqiang Xu, Kai Chen, and Yuqing Yang. Parallelizing tool execution and LLM generation for low-latency agent serving. arXiv preprint arXiv:2603.18897, 2026.

Steve Uhlig, Bruno Quoitin, Jean Lepropre, and Simon Balon. Providing public intradomain traffic matrices to the research community. *Comput. Commun. Rev.*, 36(1):83–86, 2006.

Trond Vatten and Yuming Jiang. ASGE-RR: Agentic service graph embedding with revisable reservations for dynamic AI-agent calls. arXiv preprint arXiv:2608.06033, 2026.

François Wirz, Marten Gartner, Jelte van Bommel, Elham Ehsani Moghadam, Grace H. Cimaszewski, Anxiao He, Yizhe Zhang, Henry Birge-Lee, Felix Kottmann, Cyrill Krähenbühl, Jonghoon Kwon, Kyveli Mavromati, Liang Wang, Daniel Bertolo, Marco Canini, Buseung Cho, Ronaldo A. Ferreira, Simon Peter Green, David Hausheer, Junbeom Hur, Xiaohua Jia, Heejo Lee, Prateek Mittal, Omo Oaiya, Chanjin Park, Adrian Perrig, Jerry Sobieski, Yixin Sun, Cong Wang, and Klaas Wierenga. Scaling SCIERA: A journey through the deployment of a next-generation network. In *Proc. of ACM SIGCOMM*, 2025. doi: 10.1145/3718958.3750532.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W. White, Doug Burger, and Chi Wang. Autogen: Enabling next-gen LLM applications via multi-agent conversations. In *Proc. of COLM*, 2024.

Yichang Xu, Gaowen Liu, Ramana Rao Kompella, Tiansheng Huang, Sihao Hu, Fatih Ilhan, Selim Furkan Tekin, Zachary Yahn, and Ling Liu. A multi-agent perception-action alliance for efficient long video reasoning. In *Proc. of IEEE/CVF CVPR*, 2026.

Xin Zhang, Hsu-Chun Hsiao, Geoffrey Hasker, Haowen Chan, Adrian Perrig, and David G. Andersen. SCION: Scalability, control, and isolation on next-generation networks. In *Proc. of IEEE S&P*, 2011.

Haoyu Zheng, Fangcheng Fu, Jia Wu, Binhang Yuan, Yongqiang Zhang, Hao Wang, Yuanyuan Zhu, Xiao Yan, and Jiawei Jiang. Efficient serving for dynamic agent workflows with prediction-based KV-cache management. arXiv preprint arXiv:2605.06472, 2026.

APPENDIX ROADMAP

The appendices collect the notation used in the formulation and method (Appendix A), the task-aware policy-selection algorithm and a worked example(Appendix B), and supplementary results from the paired replays in Section 4 (Appendix C).

A KEY NOTATION AND FORMAL DEFINITIONS

The following table collects the notation needed to follow the formulation and method. Local indices and derived expressions are defined where they appear.

Table 1: Core notation for the task model and path policy. A hat denotes an estimate based on information available at decision time.

Symbol	Meaning
Task graph and completion	
$G_t = (V_t, E_t)$	Execution graph revealed by time t, with invocation nodes V_t and known dependency edges E_t.
$e = (u, v, o_e)$	Dependency from producer invocation u to consumer invocation v for object version o_e.
$\mathcal{F}_t$	Task and network information available to an online policy at time t.
r_e, B_e	Release time and useful-byte size of the committed object associated with edge e.
A_e	Time when the complete object is available to its consumer after reception and validation.
$\mathrm{Req}(v)$	Application-selected objects required by invocation v.
q_v, R_v	Earliest application launch time and resulting input-readiness time of invocation v.
F_v	Completion time of invocation v.
a_w, T_w	Admission time and final-output completion time measured from admission for workflow w.
$\mathrm{sinks}(w)$	Invocations producing the final outputs of workflow w.
Endpoint path choice	
$\mathcal{U}_t$	Sender's committed, ready objects not yet bound to paths at time t.
$\mathcal{P}_e(t), p_e$	Authorized candidate paths for object e and its assigned path $p_e \in \mathcal{P}_e(t)$.
$\mathbf{p} = (p_e)_{e \in \mathcal{U}_t}$	Joint path assignment for the sender's ready objects.
Input-readiness estimate and score	
$\widehat{L}_e(\mathbf{p} \mid \mathcal{F}_t)$	Estimated remaining time to complete reception and validation of a ready or in-flight object under assignment $\mathbf{p}$.
$\widehat{A}_e(\mathbf{p})$	Predicted consumer-availability time of object e under assignment $\mathbf{p}$.
$\mathcal{C}_t$	Fixed set of known, not-yet-started consumers potentially affected by the compared assignments, including effects on in-flight objects.
$\widehat{R}_v(\mathbf{p})$	Predicted input-readiness time from the current decision onward, defined only when admission is satisfied and every required input has observed or estimated availability.
$J_t(\mathbf{p})$	Mean remaining input-waiting time across all consumers in $\mathcal{C}_t$, when the comparison is supported.

A.1 DYNAMIC TASK GRAPH AND HANDOFF RECORDS

The execution graph G_t records invocations and versioned object dependencies exposed by the MAS at time t. A materialized handoff records its release time, committed size, and eventual consumer availability; future values remain explicitly unknown. Predictions, if used later, carry separate provenance and observation timestamps.

A.2 Network State and Supported Configuration

The policy can only act on what the endpoints can actually observe, and the network record is its single source of path information. The record contains candidate identifiers, the AS/interface sequences exposed for each candidate, validity intervals, measurement timestamps, and every path a transfer actually uses. It also records path overlap directly, so no assumption of independent bottlenecks enters the comparison. The application's own role stays at the endpoint boundary. It configures the endpoint path choices or policies the deployment supports, but it never gains arbitrary control over transit capacities or intra-AS routes. *FlowAhead*'s path-selection policy further assumes that a SCION-facing adapter can estimate complete-reception times for a candidate joint assignment before the path is bound, and it treats receiver-provided validation estimates as a separate term. Required objects that are not yet committed keep unknown availability times, so a comparison that would affect a consumer waiting for such an object uses the network-only assignment.

B Policy Selection

B.1 Task-Aware Policy Selection Algorithm

Algorithm 1 implements the local comparison from Section 3.2. It starts from the network-only assignment, obtains supported arrival estimates, and compares candidates by the mean remaining input-waiting time in Equation 7. The comparison uses one fixed consumer set $\mathcal{C}_t$ throughout, namely every known, not-yet-started consumer that a candidate could affect, including consumers of in-flight objects. The decision retains the network-only assignment whenever any member of this set still has unresolved required inputs, is awaiting an uncommitted object, or has not satisfied its application admission conditions. A candidate also goes unscored if it lacks supported arrival estimates for this same set, since the set is fixed and consumers cannot be dropped to improve its score. We infer no future agent execution time or output-release time anywhere in the comparison.

Only objects in $\mathcal{U}_t$ receive new path assignments. In-flight objects keep their existing paths, but their arrival estimates and affected consumers remain part of the comparison. This matters because one ready object can still warrant a task-aware comparison when its path choice affects an in-flight transfer, and the algorithm does not require two agents to release objects simultaneously. If no improvement is accepted, it returns the network-only assignment.

The remaining rules ensure that a selected assignment is still valid when it takes effect. Neighbor ties are resolved with stable object-version and path identifiers, and only fully evaluated candidates may replace the incumbent. Immediately before dispatch, the caller rechecks the graph, object versions, branch state, and path validity for every returned assignment, including a network-only fallback. A stale assignment triggers a fresh selection, while a NoPath result follows the application's wait or retry rules. The transport binds one selected path per object, and the receiver records A_e only after the complete committed version passes validation. Later events trigger a new decision only for objects that are ready and unbound at that point.

B.2 A Task-Aware Path Selection Example

This example shows what task dependencies add to path selection: two assignments with the same transfer-time multiset can differ in how soon they make consumers ready, because an earlier object arrival is useful only if another required input does not remain the bottleneck. Consider two equal-sized objects e_1 and e_2 ready at the same sender at time zero. Consumer v_1 waits only for e_1; consumer v_2 requires e_2 and a third object e_3. Object e_3 has already been produced and is in flight on an independent path, with network-estimated availability at time ten under every candidate. Both consumers have satisfied their application admission conditions. The network-side interface estimates that a fast route delivers either local object in two time units when used alone, or both at time four when they share it; a separate route delivers either in five. These illustrative availability estimates include delivery and validation and assume no other consumers are affected.

Sending both objects on the fast route gives input-readiness times $(4, \max(4, 10)) = (4, 10)$ and score $J_t = 7$. Sending e_1 on the fast route and e_2 on the separate route gives $(2, \max(5, 10)) = (2, 10)$ and score $J_t = 6$. Reversing that assignment gives $(5, \max(2, 10)) = (5, 10)$ and score $J_t = 7.5$. The split assignments have the same transfer-time multiset $\{2, 5\}$, but the required-input

Algorithm 1: Task-aware policy selection at one sender

Input: Online state $\mathcal{F}_t$; network-only policy π_{net}; path cap K, score cap H (including the initial score), time budget τ, tolerance $\varepsilon \geq 0$
Output: Path assignment for the local ready objects, or NOPATH

Start timer; take a snapshot of $\mathcal{F}_t$ and form the committed, ready, unbound set $\mathcal{U}_t$
Record in-flight objects, their fixed path bindings, and acknowledged remaining bytes
Filter authorized, valid, transport-compatible paths $\mathcal{P}_e(t)$ for each $e \in \mathcal{U}_t$
if $\mathcal{U}_t = \emptyset$ **then**
  return $\emptyset$
if $\exists e \in \mathcal{U}_t : \mathcal{P}_e(t) = \emptyset$ **then**
  return NOPATH
$\mathbf{p}^0 \leftarrow \pi_{\text{net}}(\mathcal{U}_t, \{\mathcal{P}_e(t)\}, \mathcal{F}_t)$; retain its paths when screening to K per object
Fix the revealed graph and all known, not-yet-started consumers $\mathcal{C}_t$ potentially affected by the compared assignments
if $\mathcal{C}_t = \emptyset$ *or any member lacks resolved required inputs, committed objects, or satisfied admission conditions* **then**
  return $\mathbf{p}^0$
if *the remaining decision budget is insufficient* **then**
  return $\mathbf{p}^0$
$(\mathbf{p}, j) \leftarrow (\mathbf{p}^0, J_t(\mathbf{p}^0))$
`// The initial score counts toward` H`.`
if j *is undefined* **then**
  return $\mathbf{p}^0$
while *fewer than* H *scores and elapsed time* $< \tau$ **do**
  Generate single-object and two-object path changes only for objects in $\mathcal{U}_t$, using retained authorized candidates
  `// In-flight paths and the consumer set` $\mathcal{C}_t$ `stay fixed.`
  Estimate affected arrivals; skip a neighbor if any required arrival estimate is unsupported
  Score supported neighbors within the remaining budget; let $\mathbf{q}$ be the best fully scored neighbor
  if *no* $\mathbf{q}$ *exists or* $J_t(\mathbf{q}) \geq j - \varepsilon$ **then**
    break
  $(\mathbf{p}, j) \leftarrow (\mathbf{q}, J_t(\mathbf{q}))$
return $\mathbf{p}$

sets distinguish their value: speeding up e_2 is hidden by the wait for e_3. No agent execution time enters this calculation.

If e_3 were still being produced by a reasoning agent, its availability would be unknown and this numerical comparison would not be made. Even in the fully known case, earlier input readiness does not establish an improvement in task completion time or its tail. The opportunity requires a trade-off between object arrivals: when paths do not interact and every object can independently use its fastest route, the policy has no dependency-based improvement to make over those choices.

C SUPPLEMENTARY EVALUATION RESULTS

This appendix reports the remaining figures from the paired replays in Section 4; they use the same tasks and network settings as the main-text figures. Every result plot compares BGP, *FlowAhead*, and random selection. Each candidate set contains at most eight paths of at most six hops. *FlowAhead* receives each candidate's current available bandwidth and delay from the emulation; a deployed endpoint would have to estimate available bandwidth, so this assumption favors *FlowAhead*.

Replayed completion times differ from the live run by less than 1.5 s. As in Section 4.4, network time is the sum of transfer durations within a task; only the completion-time breakdowns count overlapping transfers once.

C.1 Network Topology and Route Choice

Figure 6 shows the route choice behind the largest gains. The data source is in London (D) and the A4VL orchestrator in Frankfurt (H), so every video first crosses the London–Frankfurt hop. With no background load, BGP and the selector both use UK–NL–DE. At 10% load, the RTT of this route rises from 10.8 ms to 60.8 ms and the selector switches to UK–FR–LU–DE with 12.3 ms, but with 100 Mbit/s site access both routes remain limited to 100 Mbit/s. At 25% and 35% load, the BGP route crosses a saturated link and offers 20 Mbit/s at 110.8 ms RTT, whereas the selected route keeps 100 Mbit/s at 12.3 ms; without the access limit, the selected route offers 8128 Mbit/s at 25% load. With 100 Mbit/s access, the Frankfurt–Madrid hop changes route only at 35% load, where the selector reduces the RTT from 72.7 ms to 23.0 ms, and the Frankfurt–Stockholm hop uses the same route under both policies at every load.

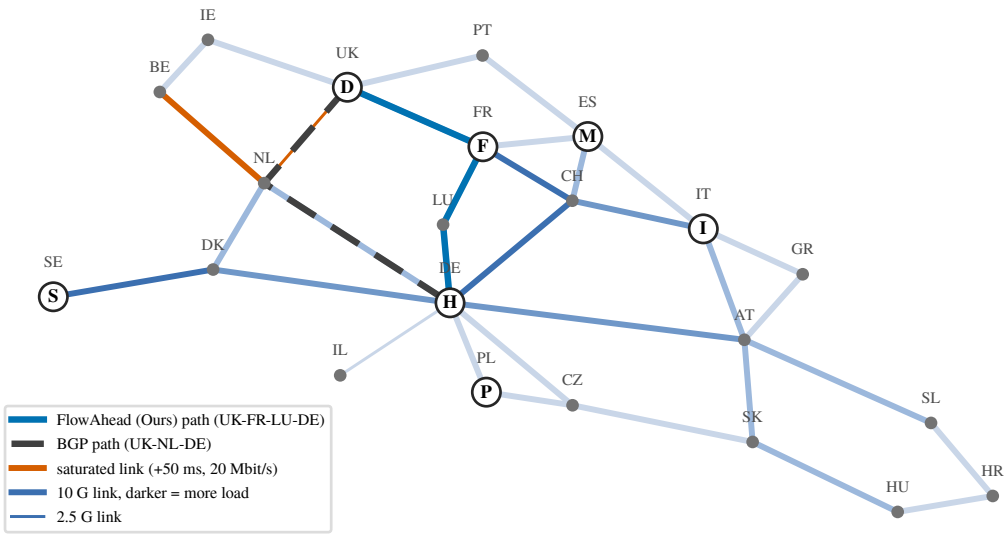

Figure 6: GEANT 2012 backbone used in the emulation (22 PoPs and 33 links; force layout, not geographic positions). Line width encodes capacity (10 or 2.5 Gbit/s) and color the mean background utilization at 25% load; saturated links (vermilion) add 50 ms of queueing and leave 20 Mbit/s to a new flow. Lettered nodes host agents: D London, H Frankfurt, M Madrid, and S Stockholm, plus F Paris, I Milan, and P Poznań for six agents. For the London–Frankfurt hop, BGP follows UK–NL–DE (20 Mbit/s, 110.8 ms RTT), whereas *FlowAhead* uses UK–FR–LU–DE (12.3 ms RTT; 100 Mbit/s with the access limit and 8128 Mbit/s without it). Random selection is not drawn because it samples a candidate independently for each transfer and has no single fixed route. The drawing is otherwise identical for both access settings.

C.2 Results with 100 Mbit/s Site Access

Figure 2 summarizes the per-task-ratio trends in the main setting. At 10% load, the median network-time ratio of three-agent Debate already falls to 0.622, consistent with the selector avoiding the queueing delay on the congested London–Netherlands link, whereas that of three-agent A4VL stays at 0.994 because both London–Frankfurt routes are limited by the access link. Random selection never improves the median; its median completion-time ratios lie between 1.000 and 1.493. Figures 8 and 9 provide absolute distributions in seconds, while Figures 10 and 11 normalize each paired replay to BGP.

Figure 7 shows why network-time savings become smaller completion-time savings. For three-agent A4VL at 25% load, the median network part falls from 141.5 s under BGP to 61.7 s under the selector, while agent inference and local processing remain at about 95 s. For three-agent Debate, the network part falls from 1.5 s to 0.6 s, but about 6 s of agent time remains. Model calls are identical across policies; small shifts in the agent part arise because transfer timing changes how parallel calls overlap.

The absolute CDFs make the workload and tail scales visible, whereas the ratio CDFs isolate relative gains despite those scale differences. Figure 12 traces one task per workload at 25% load. Under BGP, both A4VL tasks stall on the first video hop over the saturated London–Netherlands link, and every Debate message round incurs the 50 ms queueing delay on that link.

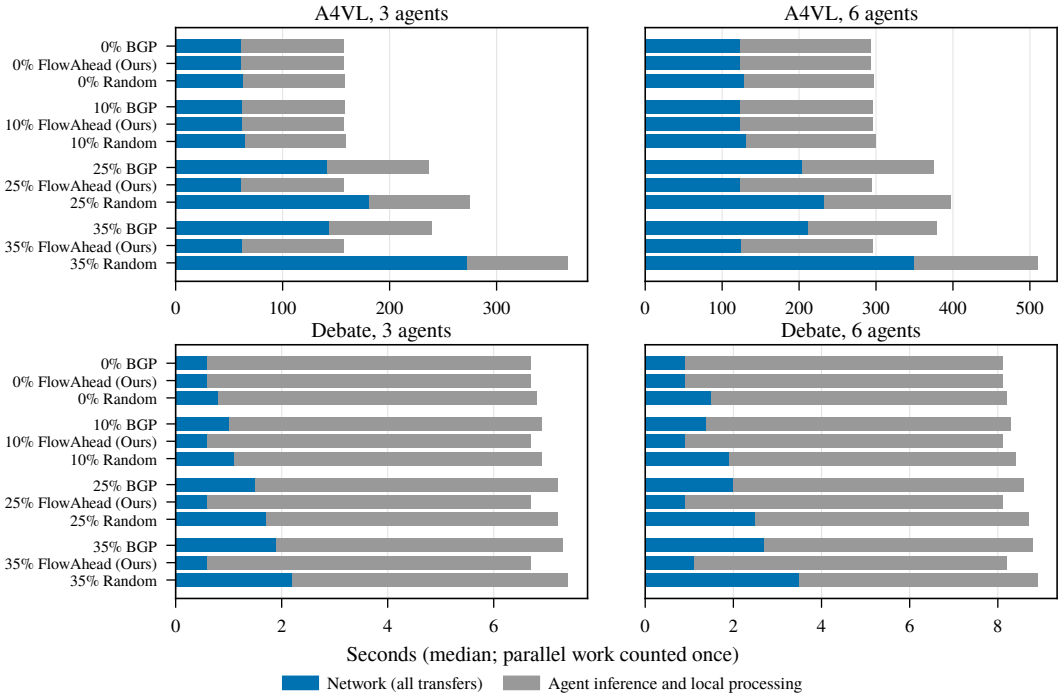

Figure 7: Composition of the median task completion time with 100 Mbit/s site access into network time, counted as wall-clock time with at least one transfer in flight, and agent inference and local processing. Bars compare BGP, *FlowAhead*, and random selection at each background load; the two parts sum to the completion time.

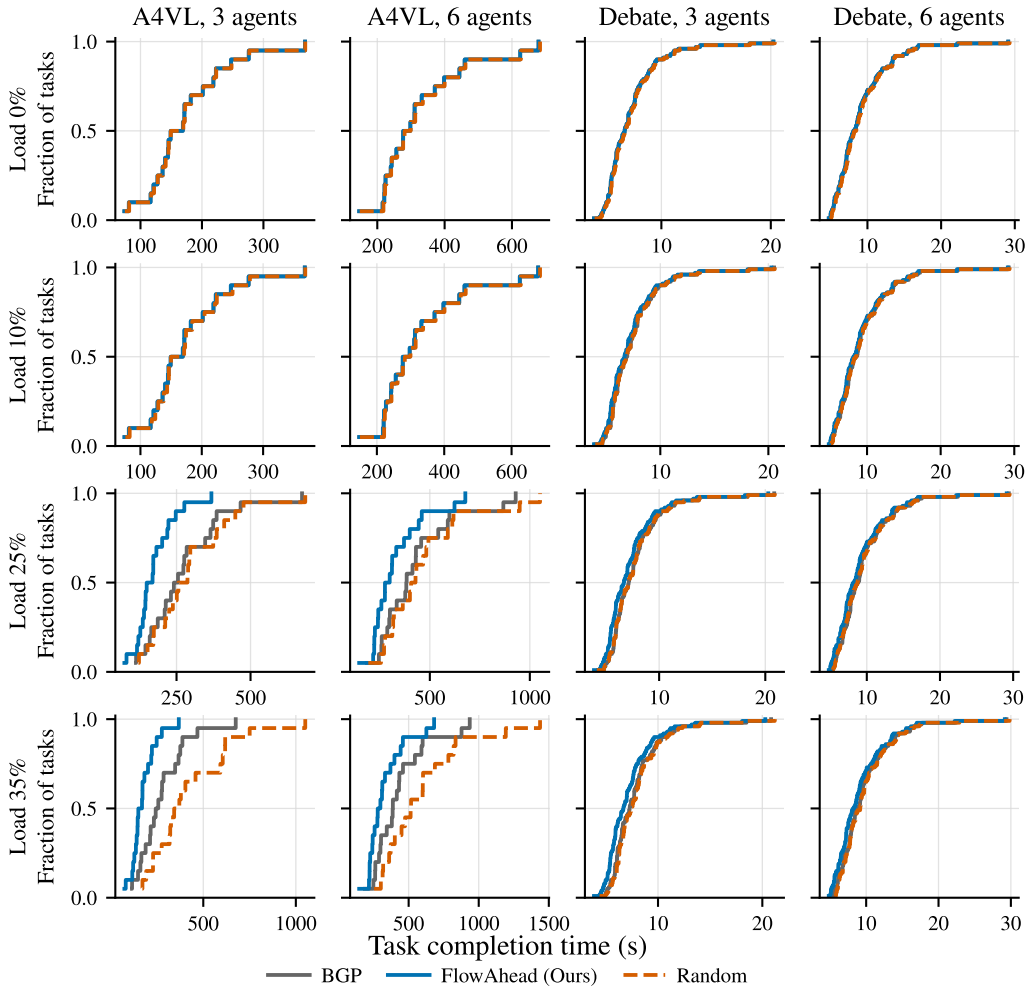

Figure 8: Absolute task-completion-time CDFs with 100 Mbit/s site access. Rows are background loads and columns are workloads; gray, blue, and dashed orange denote BGP, *FlowAhead*, and random selection, respectively. Overlapping curves indicate equal or nearly equal completion times.

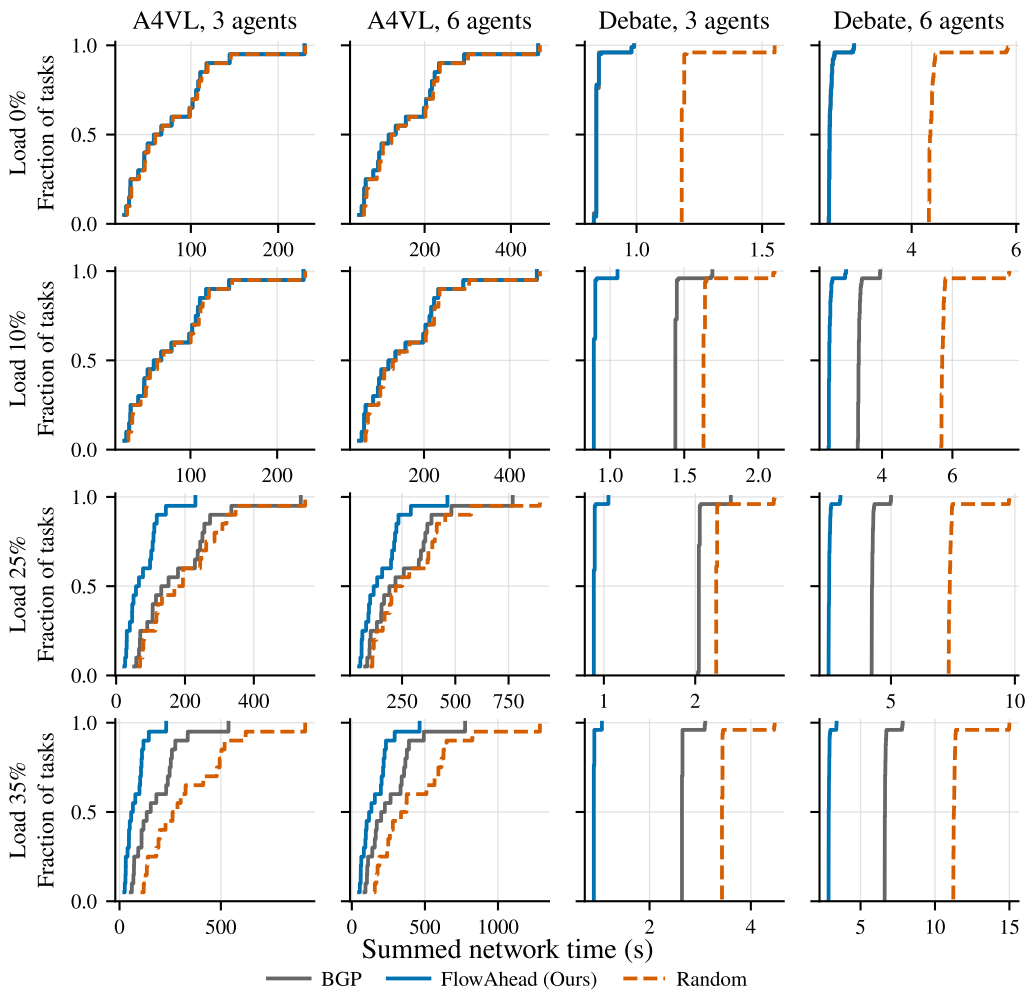

Figure 9: Absolute summed network-time CDFs with 100 Mbit/s site access. Network time excludes model inference; layout and line styles follow Figure 8.

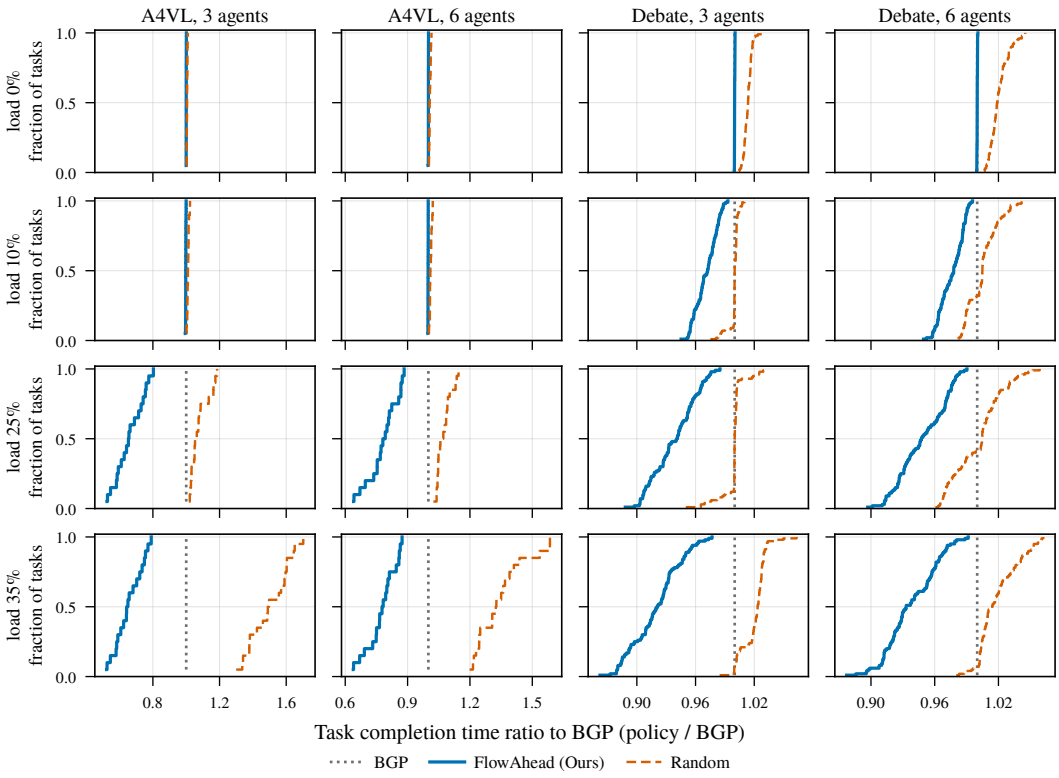

Figure 10: CDFs of the per-task completion-time ratio to BGP with 100 Mbit/s site access; rows are background loads and columns are workloads. Gray dotted, blue solid, and orange dashed lines denote BGP, *FlowAhead*, and random selection. BGP is the unit baseline; model calls are replayed identically so the ratios isolate path choice.

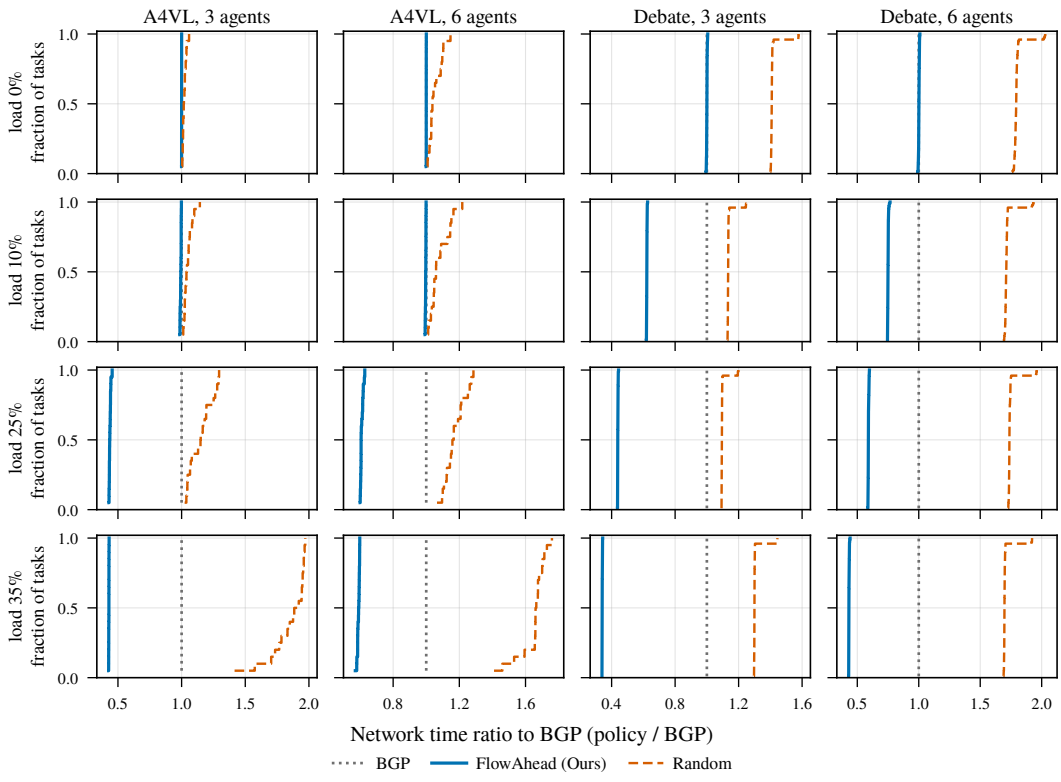

Figure 11: CDFs of the per-task summed network-time ratio to BGP with 100 Mbit/s site access. Network time excludes model inference; layout and line styles follow Figure 10.

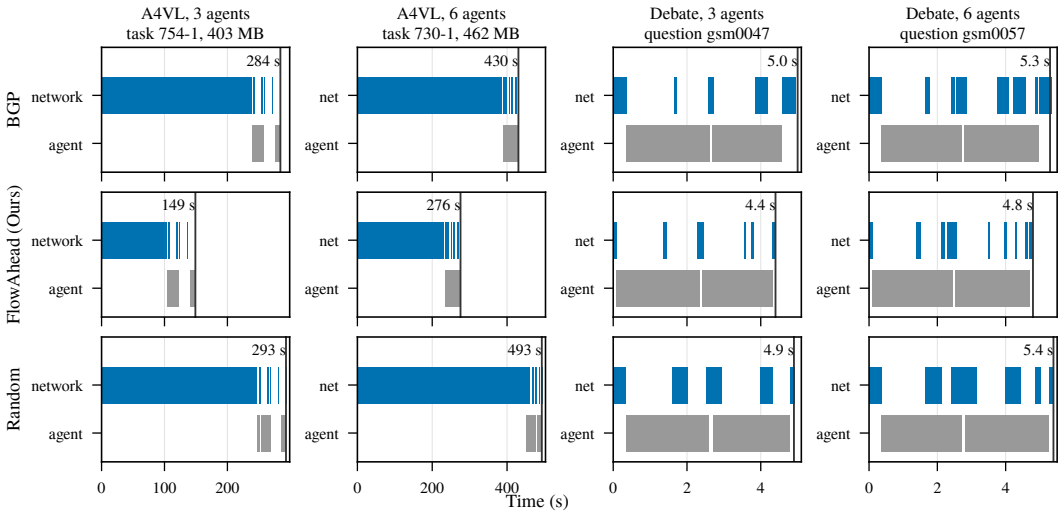

Figure 12: One task per workload at 25% load with 100 Mbit/s site access under BGP, *FlowAhead*, and random selection, with a shared time axis per column. Blue bars are transfers and gray bars agent/local work. Completion times under the three policies are 284.2/149.1/293.1 s for A4VL task 754-1 with three agents, 430.4/275.6/492.9 s for A4VL task 730-1 with six agents, 5.0/4.4/4.9 s for Debate question gsm0047 with three agents, and 5.3/4.8/5.4 s for gsm0057 with six agents.

C.3 Results without the Site-Access Limit

Figures 13–20 repeat the analysis without the 100 Mbit/s access limit, keeping the backbone, background loads, and task inputs unchanged. Figure 13 provides the complete per-task-ratio trends. At 25% load, A4VL's median network-time ratios are 0.039 and 0.072 for three and six agents. Even at 0% load, the three-agent network-time ratio is 0.888 because the BGP London–Frankfurt route includes one of the two 2.5 Gbit/s links, whereas the selected route uses only 10 Gbit/s links. The ratio rises to 0.091 at 35% load; over the same load change, the Frankfurt–Stockholm hop, which uses the same route under both policies, falls from 1868 to 274 Mbit/s.

Figure 14 shows that the selector reduces the median network part of three-agent A4VL at 25% load from 103.4 s to 3.9 s, so that about 95 s of agent time accounts for almost all of its completion time. Figures 15 and 16 show the complete absolute distributions, and Figures 17 and 19 show their paired ratios to BGP. In Figure 18, the A4VL completion-time ratio still decreases with video size (Spearman $\rho \approx -0.85$ at 25% and 35% load for both team sizes), with per-task ratios as low as 0.20. Figure 20 shows the corresponding task timelines.

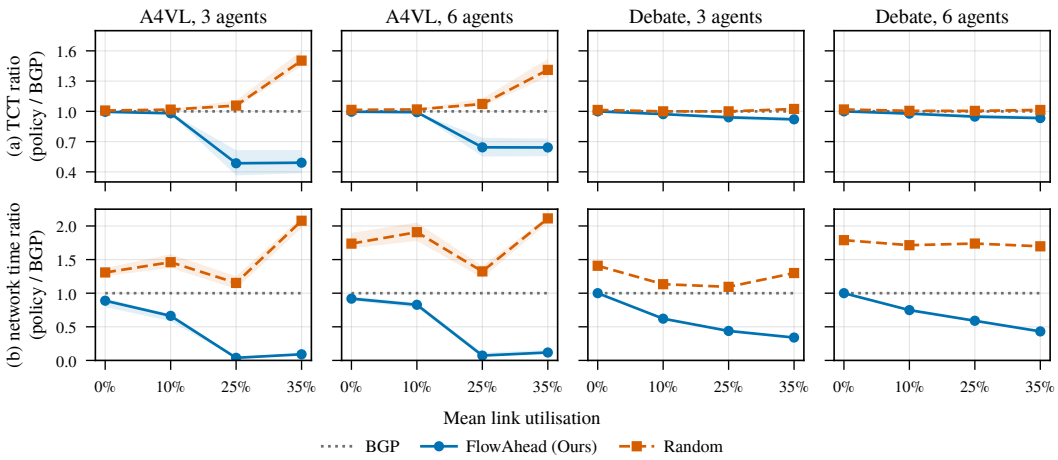

Figure 13: Median per-task ratio to BGP and interquartile range against background load without the site-access limit: (a) task completion time and (b) summed network time. Axes and line styles follow Figure 2, enabling direct comparison across access conditions.

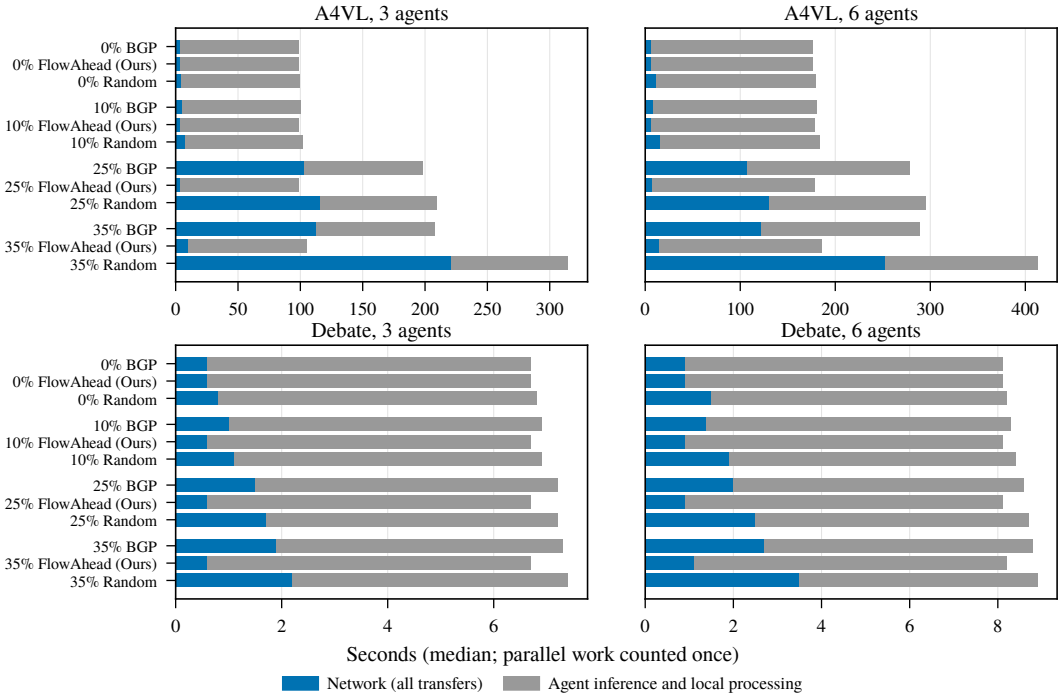

Figure 14: Composition of the median task completion time without the site-access limit. Definitions follow Figure 7.

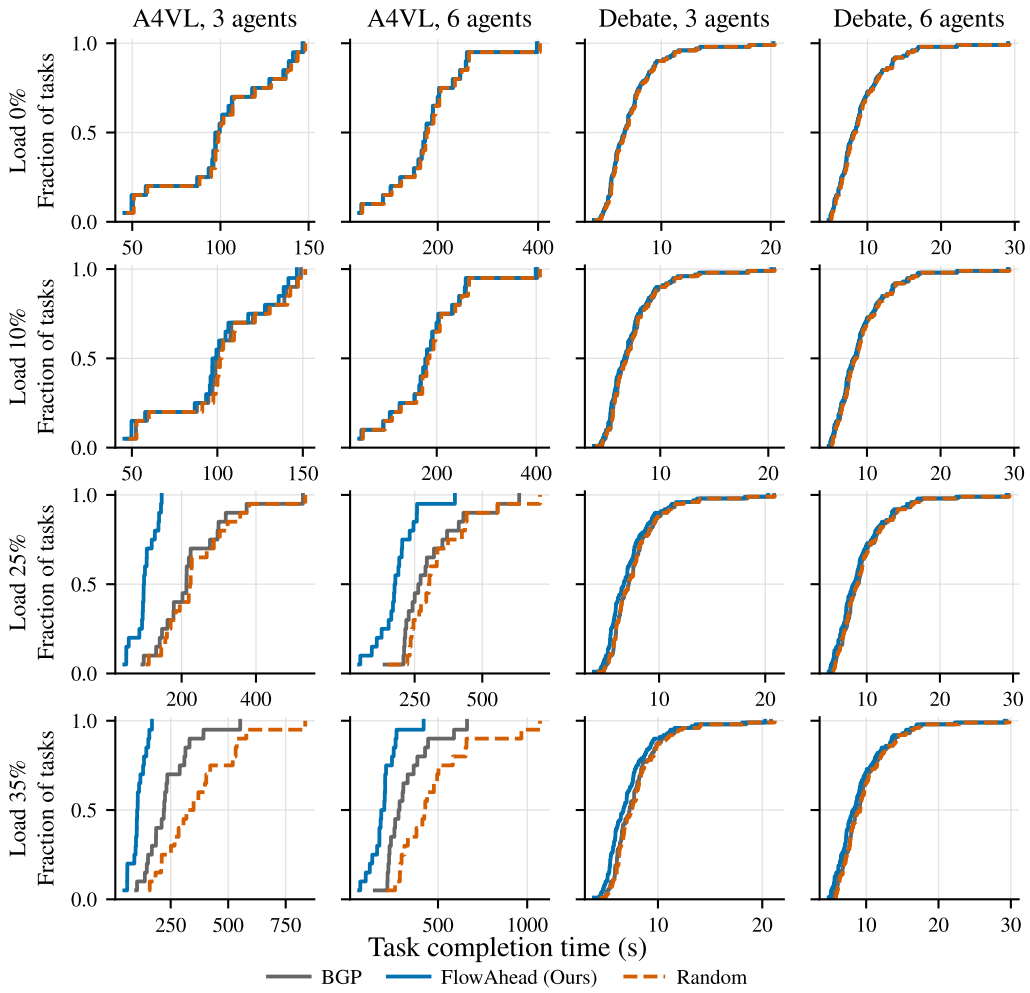

Figure 15: Absolute task-completion-time CDFs without the site-access limit. Rows are background loads and columns are workloads; line styles follow Figure 8.

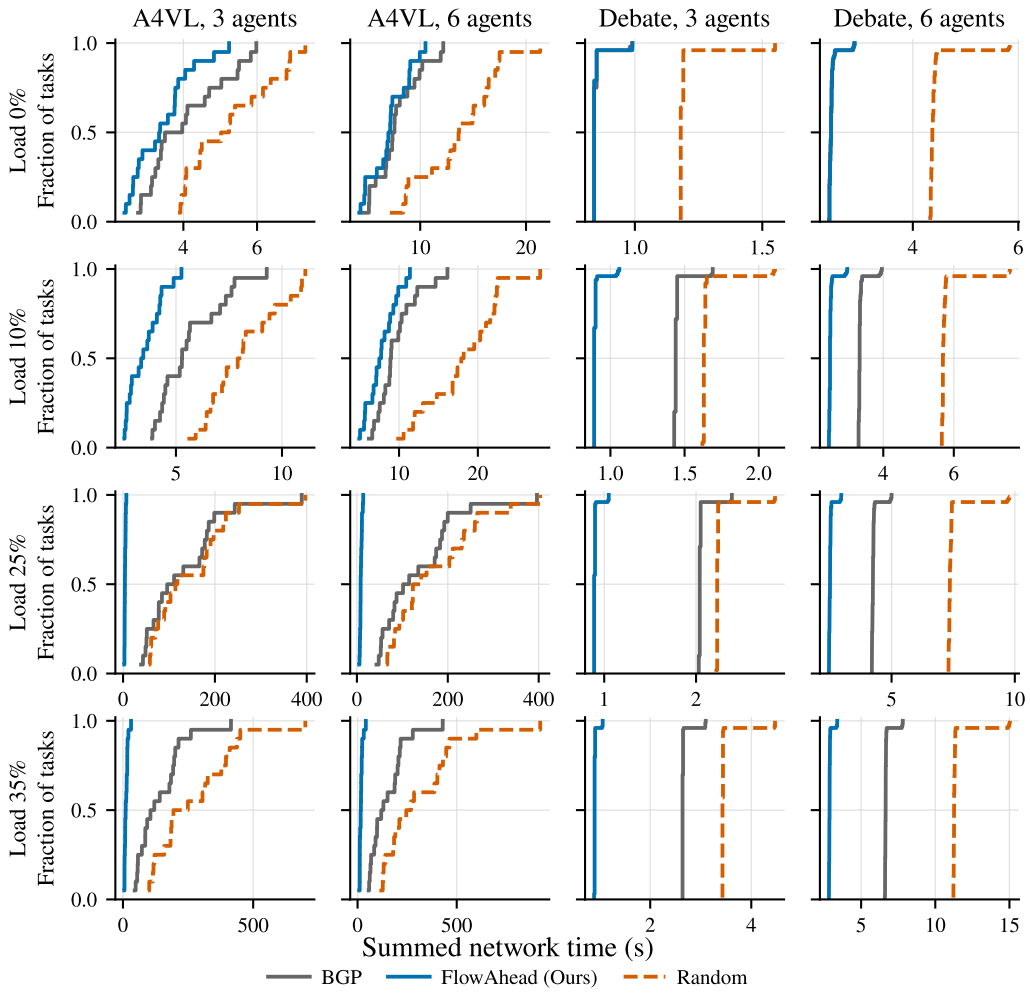

Figure 16: Absolute summed network-time CDFs without the site-access limit. Network time excludes model inference; layout and line styles follow Figure 8.

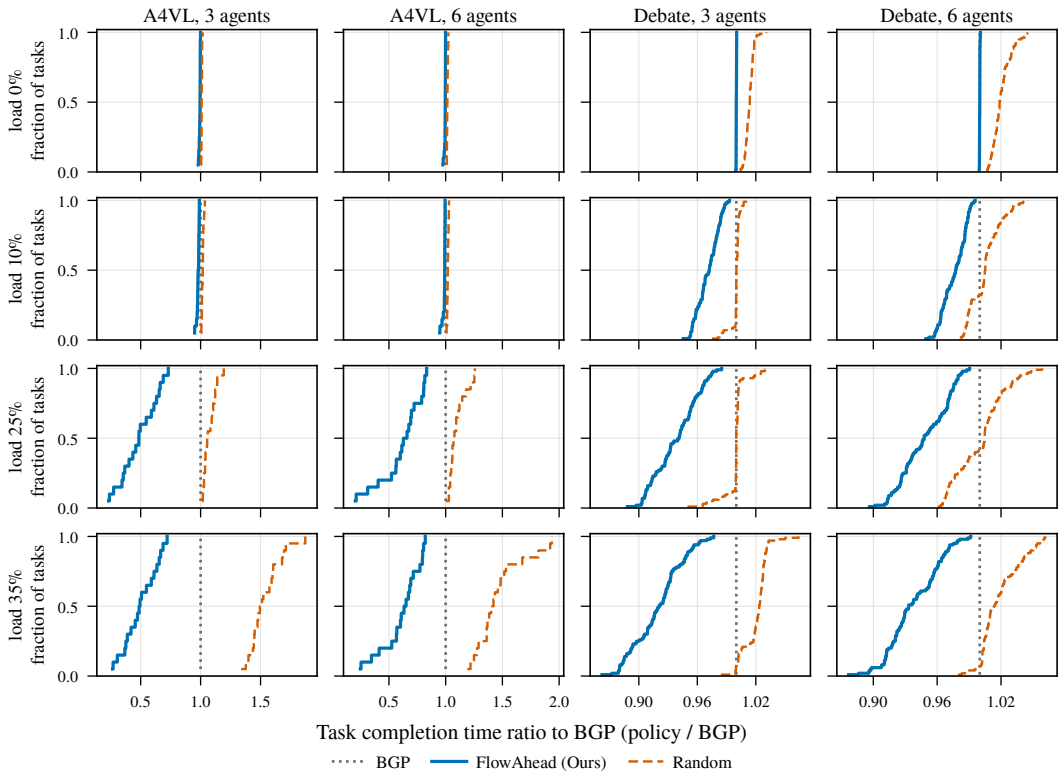

Figure 17: CDFs of the per-task completion-time ratio to BGP without the site-access limit. Layout and line styles follow Figure 10.

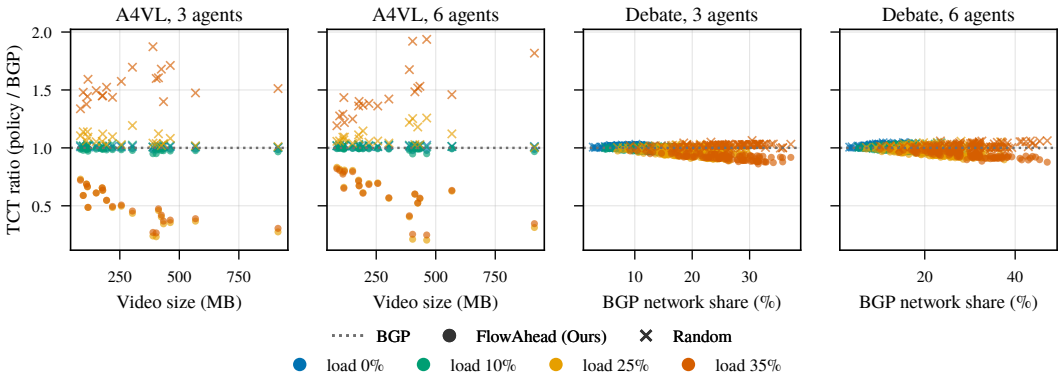

Figure 18: Per-task completion-time ratio to BGP under BGP, *FlowAhead*, and random selection without the site-access limit, plotted against video size for A4VL and communication share under BGP for Debate. Marker shape denotes policy and color denotes background load; the gray line is the BGP baseline.

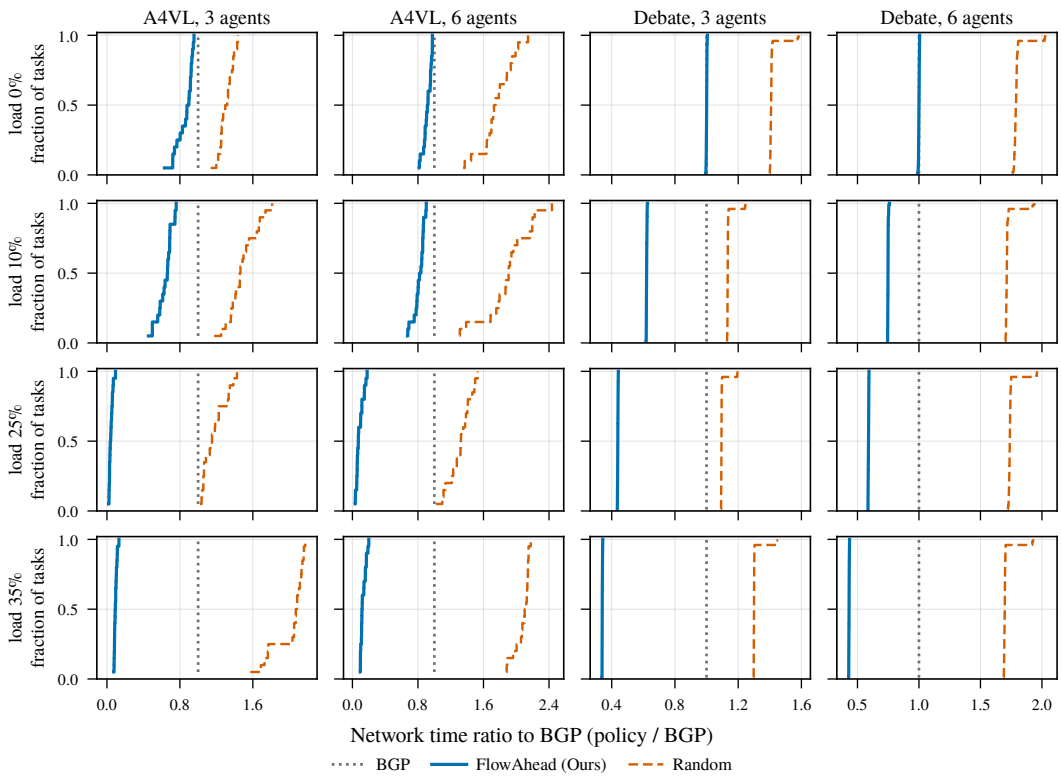

Figure 19: CDFs of the per-task summed network-time ratio to BGP without the site-access limit. Network time excludes model inference; layout and line styles follow Figure 11.

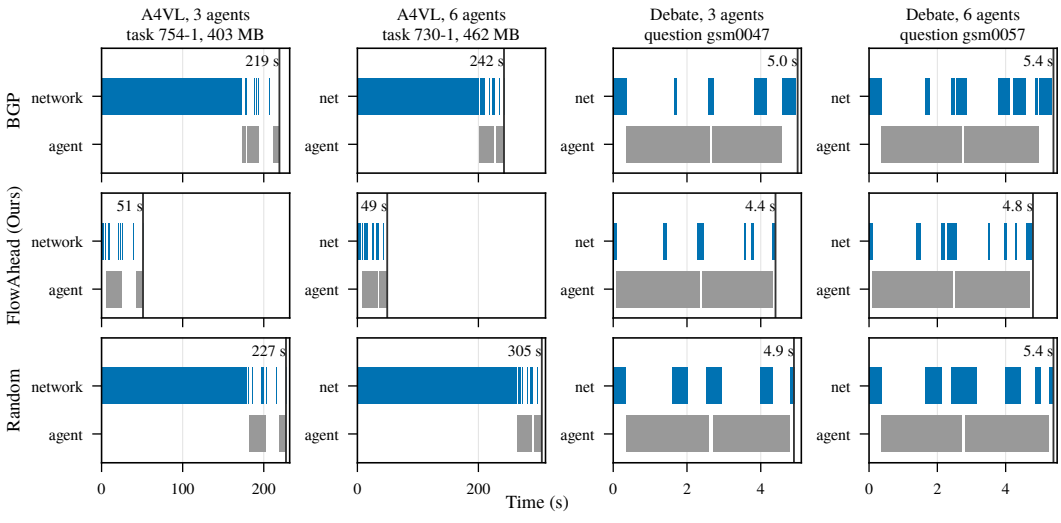

Figure 20: The tasks of Figure 12 at 25% load without the site-access limit under BGP, *FlowAhead*, and random selection. Completion times under the three policies are 219.3/51.1/227.4 s for A4VL task 754-1, 242.3/49.3/304.8 s for A4VL task 730-1, 5.0/4.4/4.9 s for Debate question gsm0047, and 5.4/4.8/5.4 s for gsm0057.