

An Analysis of Stateful Service Migration for Tactical Edge Environments

A. Amato, G. Bartolomeo, H. Bastiaansen, W. Datema, M. Fogli, T. Kudla, I. Raimondi, P. Sanchez and N. Suri

MILCOM 2026 - 2026 IEEE Military Communications Conference (MILCOM), 2026

Cite this as

A. Amato, G. Bartolomeo, H. Bastiaansen, W. Datema, M. Fogli, T. Kudla, I. Raimondi, P. Sanchez and N. Suri, “An Analysis of Stateful Service Migration for Tactical Edge Environments,” accepted for publication in *MILCOM 2026 - 2026 IEEE Military Communications Conference (MILCOM)*.

Notice

© 2026 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

An Analysis of Stateful Service Migration for Tactical Edge Environments

Alessandro Amato*, Giovanni Bartolomeo[†], Harrie Bastiaansen[‡], Willem Datema[‡],
Mattia Fogli[§], Thomas Kudla[¶], Ivo Raimondi[†], Pablo Sanchez^{||}, Niranjan Suri^{**}

* *Florida Institute for Human and Machine Cognition (IHMC)*, United States,

[†] *Technical University of Munich*, Germany,

[‡] *The Netherlands Organisation for Applied Scientific Research (TNO)*, The Netherlands,

[§] *University of Ferrara*, Italy,

[¶] *Fraunhofer Institute for Communication, Information Processing and Ergonomics*, Germany,

^{||} *University of Cantabria*, Spain,

^{**} *US Army DEVCOM Army Research Laboratory (ARL)*, United States

{aamato, nsuri}@ihmc.org, {giovanni.bartolomeo, ivo.raimondi}@tum.de, {harrie.bastiaansen, willem.datema}@tno.nl,
mattia.fogli@unife.it, thomas.kudla@fkie.fraunhofer.de, sanchez@teisa.unican.es, niranjan.suri.civ@army.mil

Abstract—Military operations produce ever-growing volumes of data. Given the nature of the tactical edge, processing these data close to their source is often the only viable option. No single coalition partner, however, owns enough computing resources to host all mission-critical services on its own, and partners therefore federate the resources they contribute. Service orchestration across the federation relocates services as conditions change, but it often treats them as stateless: the instance running on the source is terminated and a new one is started from scratch on the destination. Stateful service migration matters because many tactical services accumulate execution state as they run, and discarding it means repeating work that mission timelines may not afford, or that cannot be repeated at all. In this work, we analyze how checkpoint and restore can be applied to stateful service migration, and we evaluate the resulting strategies on a memory-heavy workload across representative network environments.

Index Terms—cloud computing, edge computing, service orchestration, stateful service migration, tactical networks

I. INTRODUCTION

Modern military operations generate unprecedented volumes of data from ubiquitous sensing assets on the battlefield. Processing these data close to their source is often the only viable option [1], [2], as transferring them to headquarters over denied, disrupted, intermittent, and limited (D-DIL) network links is prohibitively slow—or altogether infeasible. The tactical edge, however, is a hostile environment for computing: resources are scarce and transient, network links are unreliable, and no single coalition partner owns enough of either to run all mission-critical services on its own. Tactical units are therefore organized into Tactical Edge Clouds (TECs) and federated across partners [3], [4], with an orchestration layer that continuously monitors conditions and re-deploys services as circumstances change [5], [6]. Relocating a running service is thus a routine event rather than an exception.

Orchestration at the tactical edge, however, often treats services as stateless. Stateless migration implies that the instance running on the source is terminated and a new one is started from scratch on the destination, as no execution state needs to be preserved. Yet many services are stateful by nature. A federated learning aggregation service accumulates model updates round after round,

and a multi-target tracker maintains the tracks it has built so far; in both cases, restarting means starting over. In the best case, the lost work is simply repeated, and completion time grows accordingly. In the worst case, it cannot be repeated at all, as the data the service consumed may no longer be available. Either outcome is a problem because mission objectives carry timing constraints: a result that arrives “too late” is no longer actionable.

State migration is challenging. The state must be captured, transferred to the destination, and restored there, and at the tactical edge each of these steps is constrained. The transfer must cross D-DIL links, competing with mission traffic; the nodes involved may themselves be lost while the migration is in progress; and each partner enforces policies over its own resources that restrict what state may be moved, where, and along which path. Migration can be performed in different ways. The state can be captured at different levels, from the application, which serializes what it needs in order to resume, down to the container, which is captured as a whole. Within each level, different strategies determine when the state is captured and how much of it is transferred. The transfer itself can be direct, from source to destination, or indirect, for example relayed through the orchestrator. Each option has benefits and drawbacks, depending on the service at hand and on the conditions under which stateful service migration occurs.

This paper presents, to the best of our knowledge, the first analysis of stateful service migration for Tactical Edge Environments (TEEs). This work is conducted within the North Atlantic Treaty Organization (NATO) Information Systems Technology (IST)-240 research task group (RTG) *Tactical Adaptivity Concept Convergence for Meshed Edge Service and Data Handling*. We focus on how checkpoint and restore can be applied to migrate stateful services across a federation of TECs. We conducted experiments on a memory-heavy workload across representative network environments, measuring checkpoint and restore at the process level, which lower-bounds the cost of a full container migration. Results show the benefits and drawbacks of the different migration strategies, and how these depend on the conditions of the network links.

The remainder of this paper is organized as follows. Section II

reviews related work. Section III describes the scenario. Section IV presents the migration techniques. Section V reports the experimental evaluation. Finally, Section VI concludes the paper and outlines directions for future work.

II. RELATED WORK

Bastiaansen et al. [3] proposed a federated architecture that organizes tactical units into TECs, i.e., clouds of one or more computational nodes deployed on vehicles, small aircraft, or carried by dismounted soldiers, and federates them across coalition partners, so that each partner pools resources while retaining sovereignty over its own assets. Kudla et al. [4] showed that existing open-source container orchestration platforms can operate under D-DIL conditions, but with scalability constraints severe enough to make federation essential rather than merely beneficial. Amato et al. [5] showed that adaptive orchestration substantially reduces service response times compared to centralized processing and rapidly restores service continuity when a TEC becomes unavailable, while the overhead introduced by the system services remains negligible; they later showed that heuristic scheduling algorithms achieve near-optimal placement quality at negligible computational cost, making them practical for the continuous re-orchestration that tactical networks (TNs) demand [6]. Across this body of work, however, relocating a service is a placement update: the affected instance is terminated and a fresh one is started at the new location, and whatever state it had accumulated is lost.

Stateful service migration may occur at the application level or at the container level. At the application level, the service itself decides what to serialize in order to resume, either externalizing its state by construction, as proposed for network functions [7] and serverless functions at the edge [8], or persisting it to storage that outlives the instance, as Kubernetes does through persistent volumes. At the container level, the instance is captured as a whole and no application knowledge is required. The two levels are better seen as a spectrum than a dichotomy: Bellavista et al. [9] distinguished application-agnostic migration, which copies all the service data, from application-aware migration, which transfers only the subset the service actually needs. Container-level migration originates in live Virtual Machine (VM) migration, where memory pages are transferred iteratively while the instance keeps running, so that it must be suspended only for a final round [10], [11]. Checkpoint/Restore in Userspace (CRIU) made the same capability available for containers. Oh and Kim [12] built on it a migration approach in which a checkpoint taken on the source host is transferred to the target host, where a new container is started with the in-memory state restored from it. Voyager [13] goes further, pairing CRIU-based in-memory migration with lazy replication of the filesystem in the background. Puliafito et al. [14] evaluated cold and live (pre-copy, post-copy, and hybrid) container migration in fog environments. Kaur et al. [15] surveyed container placement and migration strategies for cloud, fog, and edge data centers. However, all of the above work targets environments spared the combination of challenges that characterizes the tactical edge: severely constrained links, which the state transfer must share with mission traffic; resource transiency, which may remove either endpoint while a migration is in progress; and resource sharing policies, which each partner enforces over its

own assets [3]. Container-level migration makes particular sense in this setting, as the underlying infrastructure already treats containers as the smallest deployment unit across the federation [4], [5]. To the best of our knowledge, no prior work has analyzed how CRIU-based container migration behaves under these conditions.

III. SCENARIO

We consider a tactical coalition operation in the aftermath of a disaster that has damaged a coastal city. Roads are partially blocked, public network infrastructure is severely damaged, and emergency vehicles must be routed through the corridors that are still viable. Several partners contribute to the operation, each deploying its own assets, such as cameras, unmanned aerial systems (UASs), and ground vehicles. The assets that carry computational resources, or that are computational nodes themselves, are clustered into TECs and federated across the nations participating in the mission. The coalition needs a vehicle-recognition model that identifies ambulances, fire trucks, engineering vehicles, and blocked or abandoned vehicles. No partner can build such a model on its own, as each observes only a portion of the area and under different conditions: one sees narrow inner streets, another checkpoints, another coastal routes. Sending the raw imagery to a single location is not an option over the available links, so each partner trains locally on what it observes and shares only model updates, which an aggregation service combines round after round.

The aggregation service is initially placed on the TEC of one of the partners, which we refer to as partner A. In every round, the service collects the model updates produced by the other partners and returns the model it has aggregated from them, so its placement must satisfy the requirements that come with the service, from the compute the aggregation demands to the links to the other partners and the trust that whoever requests the service wants guaranteed in the partner that hosts it. As the operation unfolds, any of these requirements may cease to hold. The link of partner A may degrade to the point that the updates of the round can no longer be collected within the time the mission allows, or its TEC, carried by vehicles that advance beyond the area under coalition control, may fall at risk of capture, so that the headquarters that requested the service no longer trusts partner A to hold the aggregated model. Whatever the reason, the placement is no longer feasible, and the orchestrator relocates the aggregation service to the TEC of partner B while training round 12 is in progress. Migrating the service without its state means restarting the training from scratch and discarding what the coalition has produced so far. What must survive the migration is the current training round, the partial model checkpoint, the optimizer state, the recent aggregation cache, and the metadata about the participating partners. With this state preserved, training continues on partner B as round 13, rather than restarting from round 0.

Stateful service migration can be realized in different ways, and in this work we focus on checkpoint and restore at the container level: the instance running on partner A is captured as a whole, and the resulting checkpoint is restored on partner B. Several strategies fall within this family, and we present them in Section IV. Orthogonal to the strategy is the scheme along which the checkpoint travels: directly from partner A to partner B, or indirectly, with the orchestrator retrieving it from partner A and delivering it to partner

B. The indirect scheme is not merely a fallback for when the two TECs cannot reach each other, as each partner retains authority over its own resources and may not permit a direct exchange.

IV. MIGRATION TECHNIQUES

In this work, we consider the migration of *stateful* services. Unlike a stateless redeployment, in which a fresh instance is started on the destination and all intermediate results are lost, a stateful migration preserves the execution state of the running service, such as memory contents, open files, runtime data structures, and, where required, established network connections. Every mechanism evaluated in this paper follows the same logical sequence: (i) *checkpoint* the state on the source, (ii) *transfer* it to the destination, (iii) *restore* the service there, and (iv) *validate* that execution continues correctly. The main difference between the mechanisms evaluated in this work is the strategy used to checkpoint and restore the workload and how it affects the *downtime* window given the networking conditions under which the migration occurs. It is important to note that the service downtime is distinct from total *migration time*. For example, a certain migration strategy can take longer to complete the full migration, but the application can stay active longer before being paused and moved, therefore reducing the amount of time the application is not responsive. In a TEE, both metrics matter, since the migration time affects the network link performance and the proactive planning needed to avoid unavailability, while downtime affects the service availability and the ability to react to a change of trust.

All the strategies we evaluate operate at the process level using CRIU [16], which checkpoints a running Linux process tree, including its memory pages, process hierarchy, file descriptors, and other kernel-visible execution state, into a set of image files, and later recreates the process tree from those images on another node. CRIU is also the mechanism underlying the container checkpoint/restore paths of modern orchestrators. The three strategies below differ only in *when* the state crosses the network relative to the freeze; the checkpoint/restore primitive is the same.

A. Cold Migration

Cold migration is the baseline. The service is stopped first, the source then creates a complete checkpoint, the resulting archive is transferred, and the service is restored on the destination [16]. Concretely, when a process is stopped, the process tree is dumped into an image directory and the directory is compressed into a single archive. The archive is copied to the destination, unpacked, and the process is restored from the images. The frozen source copy is destroyed only after the restored service has been validated, providing a clean fallback if any step fails. All phases lie inside the downtime window, so the unavailable period grows directly with state size and link degradation. In exchange, the control flow is simple and robust because no state remains in flight after the archive has been delivered, and the destination no longer depends on the source once the restore completes.

B. Pre-Copy Migration

Pre-copy moves memory state *before* the application is paused. The source performs one or more memory *pre-dumps* while the service keeps running. CRIU's memory tracking lets each iteration capture only pages dirtied since the previous one [17]. Each pre-dump

TABLE I: Timing boundaries of the evaluated migration techniques and their tactical implications.

Technique	Work inside downtime	Tactical implication
Cold	Full checkpoint, archive transfer, restore.	Longest interruption, but no dependency on the source after delivery. Clean fallback on the source.
Pre-copy	Final (delta) dump, its transfer, restore. Pre-dumps run while the service is live.	Short freeze at the cost of longer link occupation. Suits migrations planned ahead of predicted unavailability.
Post-copy	Minimal initial checkpoint, its transfer, restore.	Shortest freeze, but progress depends on the source page server after restore. Fragile under jamming or loss of the source node.

writes its own image directory, and later directories reference earlier ones through CRIU's `previous-image` mechanism, forming an image chain that is shipped to the destination while the service continues to run. The service is frozen only for the final dump, which happens when the number of dirty pages compared to the previous iteration is lower than a predefined delta. The final dump contains only the remaining memory pages relative to the last pre-dump. Downtime can shrink substantially because most of the memory has already crossed the link when the application freeze begins. The cost is a longer total migration time and more transferred data, since pages dirtied between iterations are sent again. Additionally, if the workload we are migrating dirties memory pages quickly, we must perform more pre-dump loops before stopping the application, with diminishing returns on the size of the final delta. In such a case, the migration takes even longer and consumes more network resources.

C. Post-Copy Migration

Post-copy inverts the boundary. A minimal initial checkpoint is created and transferred. The initial checkpoint holds only the process metadata, file descriptors, and register state needed to recreate the process at the destination, but deliberately leaves the bulk of the memory pages on the source [18]. The service is restored on the destination as early as possible without state, and a page server is instantiated on the source. When the application resumes, the destination runs a lazy-pages daemon that fetches missing memory pages on demand whenever the restored process first accesses them. The initial transfer, and thus the measured downtime, can be very small. The restored service, however, remains dependent on source-to-destination connectivity until every needed page has arrived. If the link stalls, the service is nominally running but makes no progress, because the memory page-faults cannot be resolved. Additionally, the performance of the restored service can degrade if the page server is slow or the link is lossy, and the source copy cannot be destroyed until the full transfer succeeds.

These boundaries map directly onto the scenario described in Section III. Under *predicted unavailability*, e.g., a link degrading toward the point where the state can no longer cross it, or an expected jamming window, pre-copy is attractive because the bulk of the transfer happens while the service still runs, and only a short freeze must fit before the blackout. Under ideal networking conditions, post-copy guarantees minimal downtime, but in the event of a *change of trust*, e.g., the nodes of partner A at risk of capture, the restored service might still depend on the source that is no longer trusted, and residual

state remains there until lazy paging completes. In such cases, cold and pre-copy migration allow the source images to be destroyed immediately after a validated restore. Table I summarizes the tactical implications for each of these strategies. Then, in Section V we evaluate them under a range of emulated network conditions, from benign access networks to disadvantaged tactical links, and we quantify the trade-offs between downtime, total migration time, and success rate.

V. EVALUATION

A. Setup

Testbed. The experiments run on three Ubuntu 24.04 VMs (2 vCPUs, 2 GB RAM, 10 GB disk each) provisioned with Multi-pass [19] and Terraform [20]. Two VMs are the migration endpoints: the *source* node runs the workload and produces its checkpoints, while the *destination* node receives the state and restores the workload. The third VM is an *orchestration* node, which embodies the logically centralized coordinator of Section III: it initiates each migration, drives the checkpoint, transfer, and restore steps on the endpoints, validates the outcome, and collects the measurements. The orchestration node sits on the control path only and never carries service state, ensuring that the state transfer is peer-to-peer, flowing directly from source to destination over Secure Shell (SSH)/Secure Copy Protocol (SCP). Python orchestration scripts apply the network conditions with Linux traffic control (`tc/netem`) on all inter-node traffic and record per-phase timings, archive sizes, success flags, and `node-exporter`'s resource snapshots on both endpoints.

Workload. The migrated service is *Game of Life (GoL)*, a 2048×2048 Conway's Game of Life simulation [21]. It carries roughly 32 MiB of continuously changing heap, making it representative of long-running, memory-heavy stateful services, and its state is updated on every generation, which exercises CRIU's dirty-page tracking in the pre-copy strategy and the lazy-page path in post-copy. Note that CRIU operates directly on the Linux process, below the container-orchestration layer. The measured costs are therefore a lower bound for a full container migration, which would additionally move runtime metadata, filesystem state, and orchestration decisions.

Network profiles. Each experiment sweep is repeated under the seven emulated profiles of Table II, ranging from commercial access networks to disadvantaged tactical links. The WiFi 6, 5G, and Long-Term Evolution (LTE) values follow representative edge measurements [22], and the Starlink profile follows published service values [23]. The three TEE profiles are derived from the emulated tactical-network conditions reported by Kudla et al. [4] and bracket the range of connectivity a deployed unit can realistically expect at the tactical edge. TEE Best represents a favorable tactical link, such as a well-provisioned satellite communications (SATCOM) or line-of-sight radio connection operating without interference, and marks the upper bound of what a TEE typically offers. TEE Avg represents routine operating conditions, in which node mobility, terrain masking, and channel sharing halve the usable bandwidth and double both latency and loss. TEE Worst represents a disadvantaged link at the edge of usability, such as a narrowband very high frequency (VHF) combat-net radio or a contested channel under electronic attack, combining sub-Mbit/s bandwidth with multi-second latency and heavy loss. Note that, unlike the commercial profiles, which differ

TABLE II: Emulated network profiles.

Profile	Bandwidth (Mbit/s)	Latency (ms)	Loss (%)
WiFi 6	800	5	0.00001
5G	400	10	0.01
LTE	50	40	0.08
Starlink	100	45	1.0
TEE Best	4	200	5.0
TEE Avg	2	400	10.0
TEE Worst	0.65	3000	15.0

TABLE III: Median downtime in seconds (successful/attempted runs) for each migration strategy.

Profile	Cold	Pre-copy	Post-copy
WiFi 6	4.0 (40/40)	3.4 (40/40)	3.0 (40/40)
5G	4.4 (40/40)	3.7 (40/40)	3.2 (40/40)
LTE	7.7 (40/40)	6.0 (39/40)	– (0/40)
Starlink	51.4 (40/40)	18.0 (40/40)	– (0/40)
TEE Best	472.1 (40/40)	107.1 (40/40)	– (0/40)
TEE Avg	1198.6 (40/40)	218.5 (40/40)	– (0/40)
TEE Worst	– (0/30)	– (0/30)	– (0/30)

TABLE IV: Median downtime vs. median total migration time under the TEE Avg profile, in seconds.

Strategy	Downtime (s)	Migration time (s)
Cold	1198.6	1219.2
Pre-copy	218.5	1988.7

TABLE V: Median archive sizes transferred inside the downtime window. Ranges span the per-profile medians over successful runs.

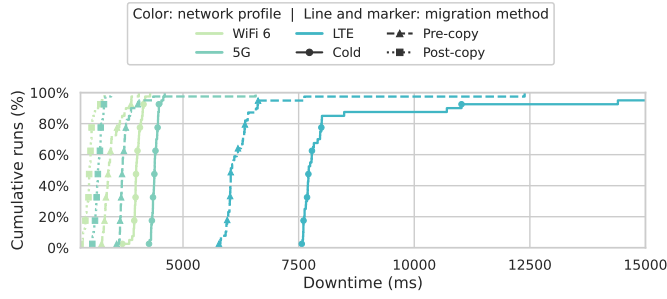
Strategy	Archive size
Cold (full checkpoint)	12.09 MiB
Pre-copy (final delta)	1.95–7.71 MiB
Post-copy (initial checkpoint)	25.3 KiB

mainly in bandwidth, the TEE profiles degrade bandwidth, latency, and loss jointly, reflecting the D-DIL conditions of Section III under which the migration strategies diverge most sharply.

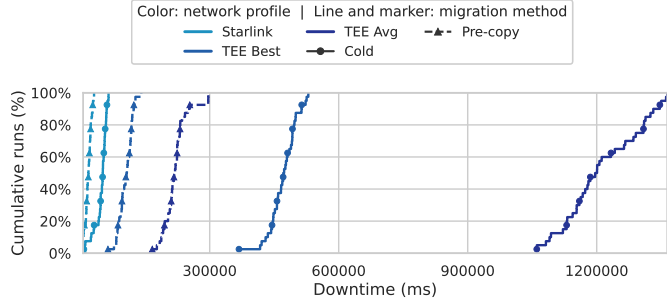
Migration procedure and metrics. The orchestration node configures each experiment run from a configuration file. Each configuration contains the migration strategy and the network profile. Each configuration is attempted 40 times (30 under TEE Worst, where single runs become very long). Downtime is computed as checkpoint plus transfer plus restore time. Success is validated at the application level. A run counts as successful only if the restored workload demonstrably makes progress, i.e., the GoL output keeps updating on the destination. This criterion matters for post-copy in particular, where a process can be restored and still stall on remote page faults. Timing distributions use successful runs only, and failures are reported separately.

B. Results

Downtime. Table III summarizes the results and Figure 1 shows the corresponding downtime CDFs. On access networks, the three strategies remain within a factor of 1.5 of each other, and the choice of strategy is secondary. As soon as the link degrades, the transfer phase dominates and the strategies diverge sharply. Pre-copy, which moves memory while the service still runs, reduces median downtime relative to cold migration by 22% under LTE,

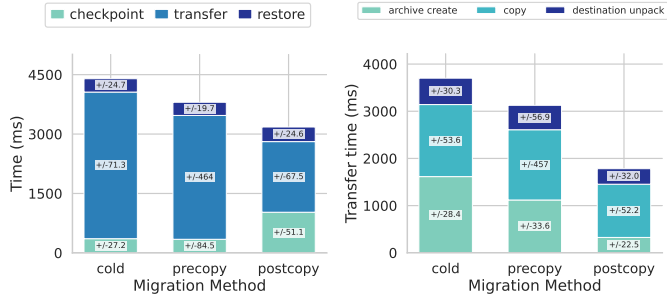


(a) Access networks (WiFi 6, 5G, LTE).



(b) Starlink and tactical edge profiles.

Fig. 1: GoL downtime Cumulative Distribution Functions (CDFs). TEE Worst produced no successful GoL runs.



(a) Downtime phases: checkpoint, transfer, and restore. (b) Transfer subphases: archive create, copy, and unpacking.

Fig. 2: Migration phase breakdown under the 5G profile.

65% under Starlink, 77% under TEE Best, and 82% under TEE Avg (from about 20 to 3.6 minutes, Table III). Table IV summarizes the differences between total migration time and actual downtime for the TEE Avg profile. Pre-copy occupies the system 63% longer than cold migration, keeping the constrained link busy for most of that period, but it reduces the application downtime by 82% compared to cold migration. Post-copy, which moves memory only after the restore, is the only strategy that fails in all LTE, Starlink, and TEE experiments, due to the low networking latency required to avoid timeouts in the memory page transfer after the restore.

Takeaway. On access networks any strategy yields seconds of downtime and the choice is secondary. On the disadvantaged links of a TEE the transfer dominates: pre-copy cuts median downtime by up to 82% relative to cold migration, at the cost of occupying the constrained link 63% longer.

Phase breakdown. Figure 2 decomposes the migration into

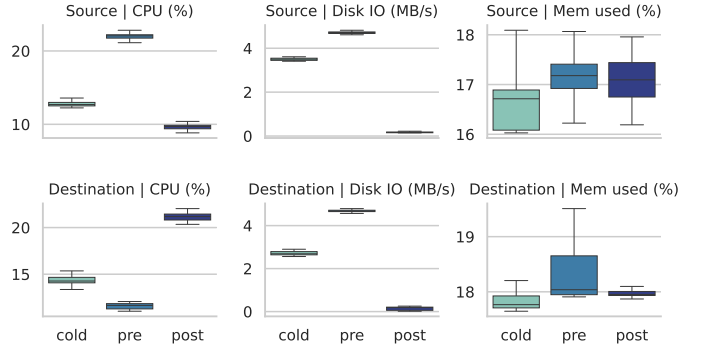


Fig. 3: GoL per-node resource usage under the 5G profile.

its components for the representative 5G profile. The downtime of an application consists of the checkpoint (or final dump), the transfer, and the restore phases shown in the figure. The transfer itself further decomposes into archive creation, the network copy and destination-side unpacking, shown in Figure 2b. On a fast link, fixed costs dominate. For cold migration, creating the archive (1.6 s median) and unpacking it (0.6 s) together exceed the copy itself (1.5 s), while checkpoint and restore stay near 0.3 s each. These fixed components are essentially independent of the network, whereas the copy grows with archive size and link degradation. To point out the effect of archive size, Table V shows the median size of the archive that must be transferred inside the downtime window. The archive sizes help explain the divergence in downtime variations across migration techniques (Table IV): as the profiles worsen, the 12.09 MiB cold-migration archive turns the copy into the dominant term, while pre-copy keeps the in-downtime copy small (its bulk transfer happens during the pre-dump phase, 21.8 s on 5G, while the service is still running). Post-copy ships only its small initial checkpoint before restoring, minimizing the downtime even further. Unfortunately, post-copy fails to make progress on the memory-page transfer at the destination when the network link is constrained.

Takeaway. On fast links downtime is dominated by fixed, network-independent costs such as archive creation, unpacking, and restore. On constrained network links, the file transfer becomes the dominant term. Cold migration increases reliability, but pre-copy reduces downtime by moving most of the application state before the application is interrupted.

Migration failures. Post-copy exhibits the sharpest cliff. Its initial archive is tiny (25.3 KiB versus 12.09 MiB for a cold checkpoint, Table V), which yields the best downtime on WiFi 6 and 5G. From LTE onward, however, every post-copy run restores and then fails validation (Table III). The resumed process stalls on remote page faults and never makes observable progress. In a TEE, where the source may be jammed, displaced, or captured mid-migration, this after-restore dependency is a serious liability.

Takeaway. Post-copy’s minimal downtime hides a hard dependency on the source after restore. LTE and TEE networks stall on remote page faults and fail validation.

Resource usage. Figure 3 breaks down CPU, memory, and disk Input/Output (I/O) by node for the representative 5G profile. Cold migration is balanced, with a median CPU utilization of 12.9% on the source and 13.5% on the destination, and disk I/O of 3.4

and 2.7 MB/s, respectively, as the archive is created, transferred, unpacked, and restored. Pre-copy shifts load to the source (22.4% vs. 11.1% CPU) and almost doubles disk activity on both nodes (≈ 4.5 MB/s), reflecting the repeated pre-dump and archive handling. Post-copy shifts load to the destination (8.2% source vs. 25.9% destination CPU) and nearly eliminates disk I/O (≈ 0.15 MB/s), as the cost moves into remote page serving. Memory utilization stays in a comparable 16–18% band across all strategies. For resource balancing in a TEE, this means the strategy choice also decides *which* node pays for the migration: pre-copy burdens a source that may be about to lose power or connectivity, while post-copy burdens the destination.

Takeaway. The strategy choice also decides *which* node pays for the migration: pre-copy loads CPU and disk on the source, post-copy shifts the cost to the destination, and cold migration splits it evenly. In a TEE this matters when one endpoint is about to lose power or connectivity.

Summary. No single strategy wins everywhere. Cold migration is adequate for small state on any link and leaves no post-restore dependency. Pre-copy is the only strategy that keeps memory-heavy services available across disadvantaged links, provided the longer link occupation is acceptable and can be scheduled ahead of a predicted blackout. Post-copy should be restricted to reliable links or workloads with small working sets. Under the worst tactical profile, no strategy completed a single migration, underlining a research gap for new migration strategies suitable for TEE conditions. State-of-the-art migration strategies in a TEE must be planned and executed before conditions degrade past the point where the state can still cross the link.

VI. CONCLUSION AND FUTURE WORK

This paper presented, to the best of our knowledge, the first analysis of stateful service migration for TEEs, comparing cold, pre-copy, and post-copy checkpoint and restore across seven emulated network profiles. Future work follows from the conditions under which no strategy succeeded. We plan to decouple the state transfer from the migration decision through periodic checkpoint and diff update, so that only the last delta has to cross a degraded link, and to investigate multi-node migration, in which the state is kept on more than one TEC and the destination is restored from whichever replica is reachable. We also plan to cover the indirect transfer scheme, which resource sharing policies may force upon a federation, and to quantify the additional cost of full container migration over the process-level costs reported here. Finally, we plan to broaden the evaluation beyond the memory-heavy workload considered here, to stateful services that are representative of the tactical edge and migrated as containers rather than as raw processes.

ACKNOWLEDGMENT

The work presented in this article was conducted as part of the NATO IST-240 RTG *Tactical Adaptivity Concept Convergence for Meshed Edge Service and Data Handling*, which built and extended upon its predecessor NATO IST-193 RTG on *Edge Computing at the Tactical Edge*. We thank the NATO IST-Panel for the opportunity to conduct this research and the participating partners for their valuable contributions in a stimulating and cooperative environment.

REFERENCES

- [1] G. Lewis, S. Echeverría, S. Simanta, B. Bradshaw, and J. Root, "Tactical cloudlets: Moving cloud computing to the edge," in *2014 IEEE Military Communications Conference*, 2014, pp. 1440–1446.
- [2] M. Satyanarayanan, R. Schuster, M. Ebling, G. Fettweis, H. Flinck, K. Joshi, and K. Sabnani, "An open ecosystem for mobile-cloud convergence," *IEEE Communications Magazine*, vol. 53, no. 3, pp. 63–70, 2015.
- [3] H. Bastiaansen, J. van der Geest, C. van den Broek, T. Kudla, A. Isenor, S. Webb, N. Suri, M. Fogli, B. Canessa, A. Masini *et al.*, "Federated control of distributed multi-partner cloud resources for adaptive c2 in disadvantaged networks," *IEEE Communications Magazine*, vol. 58, no. 8, pp. 21–27, 2020.
- [4] T. Kudla, M. Fogli, S. Webb, G. Pinggen, N. Suri, and H. Bastiaansen, "Quantifying the performance of cloud-oriented container orchestrators on emulated tactical networks," *IEEE Communications Magazine*, vol. 60, no. 5, pp. 74–80, 2022.
- [5] A. Amato, H. Bastiaansen, W. Datema, M. Fogli, R. Fronteddu, R. Galliera, J. van der Geest, T. Kudla, P. Sanchez, N. Suri, and S. Watson, "A performance cost/benefit analysis of adaptive computing in the tactical edge," in *2024 International Conference on Military Communication and Information Systems (ICMCIS)*, 2024, pp. 1–8.
- [6] A. Amato, H. Bastiaansen, J. Coleman, W. Datema, M. Fogli, J. Van Der Geest, B. Krishnamachari, T. Kudla, P. Sanchez, and N. Suri, "Evaluating scheduling algorithms for adaptive orchestration in federated tactical edge cloud environments," in *2025 International Conference on Military Communication and Information Systems (ICMCIS)*, 2025, pp. 1–10.
- [7] M. Kablan, B. Caldwell, R. Han, H. Jamjoom, and E. Keller, "Stateless network functions," in *HotMiddlebox 2015 - Proceedings of the 2015 ACM SIGCOMM Workshop on Hot Topics in Middleboxes and Network Function Virtualization, Part of SIGCOMM 2015*, 2015, p. 49–54.
- [8] T. Pfandzelter and D. Bermbach, "Enoki: Stateful distributed faas from edge to cloud," in *MiddleWEdge 2023 - Proceedings of the 2nd International Workshop on Middleware for the Edge, Part of: ACM/IFIP Middleware 2023*, 2023, p. 19–24.
- [9] P. Bellavista, A. Corradi, L. Foschini, and D. Scotece, "Differentiated service/data migration for edge services leveraging container characteristics," *IEEE Access*, vol. 7, pp. 139 746–139 758, 2019.
- [10] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live migration of virtual machines," in *2nd Symposium on Networked Systems Design and Implementation, NSDI 2005*, 2005, p. 273–286.
- [11] M. Nelson, B.-H. Lim, and G. Hutchins, "Fast transparent migration for virtual machines," in *ATEC '05*. USA: USENIX Association, 2005, p. 25.
- [12] S. Oh and J. Kim, "Stateful container migration employing checkpoint-based restoration for orchestrated container clusters," in *2018 International Conference on Information and Communication Technology Convergence (ICTC)*, 2018, pp. 25–30.
- [13] S. Nadgowda, S. Suneja, N. Bila, and C. Isci, "Voyager: Complete container state migration," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, 2017, pp. 2137–2142.
- [14] C. Puliafito, C. Vallati, E. Mingozzi, G. Merlino, F. Longo, and A. Puliafito, "Container migration in the fog: A performance evaluation," *Sensors*, vol. 19, no. 7, 2019.
- [15] K. Kaur, F. Guillemin, and F. Sailhan, "Container placement and migration strategies for cloud, fog, and edge data centers: A survey," *International Journal of Network Management*, vol. 32, no. 6, 2022.
- [16] CRIU Project, "Live migration," https://criu.org/Live_migration, accessed: 2026-05-06.
- [17] —, "Iterative migration," https://criu.org/Iterative_migration, accessed: 2026-05-06.
- [18] —, "Lazy migration," https://criu.org/Lazy_migration, accessed: 2026-05-06.
- [19] Canonical, "Multipass," <https://canonical.com/multipass>, accessed: 2026-06-07.
- [20] HashiCorp, "Terraform," <https://www.terraform.io/>, accessed: 2026-06-07.
- [21] M. Gardner, "Mathematical games: The fantastic combinations of john conway's new solitaire game" life," *Scientific American*, vol. 223, no. 4, pp. 120–123, 1970.
- [22] G. Bartolomeo, J. Cao, X. Su, and N. Mohan, "Characterizing distributed mobile augmented reality applications at the edge," in *Companion of the 19th International Conference on emerging Networking EXperiments and Technologies*, 2023, pp. 9–18.
- [23] Starlink, "Starlink specifications," <https://starlink.com/legal/documents/DOC-1470-99699-90>, accessed: 2026-05-27.