A PROPOSAL FOR A SCALABLE EMERGENCY INFRASTRUCTURE: ENSURING COMMUNICATION CONTINUITY IN CRITICAL SITUATIONS

Daniel Nicolae BOBOC[1], Ioan TATARU[2], Ioana Corina BOGDAN[3], Emil SIMION[4*]

This paper proposes a dedicated telecommunications infrastructure for emergency situations, designed to sustain communication for companies and public institutions when the primary network experiences a major system failure. The proposed solution functions as an "emergency line", activating when the monitoring systems detect that the primary network has a critical service failure. The emergency system is rapidly deployed to maintain active communications until the primary network is fully restored. Once the main infrastructure is stable again, traffic is shifted back and the emergency system becomes inactive, effectively entering a low-cost "sleep mode". The solution addresses the vital need for communication continuity, reducing the risk of extended downtime through a temporary high-performance infrastructure that can be scaled dynamically to match the demand.

Keywords: Network Resilience Strategies, Disaster Recovery Architectures, Cloud Infrastructure, Self-Healing, Network Systems, Incident Detection, Emergency Communication Protocols, Cloud-based Network Solution

1. Introduction

Effective, uninterrupted communication is crucial in modern society. Major network outages can have significant consequences for businesses and critical services (such as health and emergency services, public infrastructure, information and communication technology, education or public administration) [1].

Environmental disasters, cyberattacks, as well as technical failures can seriously compromise the stability of vital systems including primary networks,

[1] PhD student, Faculty of Electronics, Telecommunications and Information Technology, POLITEHNICA University of Bucharest, Romania, e-mail: nicolae.boboc@stud.etti.upb.ro
[2] SMS & VoIP Engineer, e-mail ioanmario99@gmail.com
[3] Assistant professor, Faculty of Electrical Engineering and Computer Science, Department of Electronics and Computers, "Transilvania" University of Brasov & CiTi, Brasov, Romania, e-mail: corina.bogdan@unitbv.ro
[4] *Associate Professor, , Department of Mathematical Methods and Models, National University of Science and Technology POLITEHNICA Bucharest & CiTi, Bucharest, Romania, e-mail: emil.simion@upb.ro

leaving users without access to telephony or any data services. Given the ongoing reliance on cloud infrastructures and the public internet, ensuring communication resilience is a primary concern [2], [3].

Contemporary attacks carried out by cybercrime groups or state actors are often of DDoS or APT type and, once identified, require immediate action to isolate the threat and restore communications. The solution proposed in this paper provides an alternative, secure communication channel that ensures operational continuity during security-incident management. Ideally, the solution complies with an approved Protection Profile [4]. In the absence of such a tool, the system components to be implemented will refer to the Protection Profiles (PPs) approved for the categories: network devices, virtualization and container platforms, operating systems, software applications and machine learning (ML) components.

Maintaining active and operational activities during emergencies requires a resilient backup mechanism that can automatically take over communication traffic as soon as the primary infrastructure becomes unavailable. Network downtime can cost organizations an average of $9000 per minute and, when extended, may also damage trust and breach service-level agreements [5][6]. Therefore, continuous communication during crises is essential, not merely technical but central to operational and economic resilience. Accordingly, this work introduces an architecture for scalable emergency infrastructure designed to sustain critical communications during primary-network failures. The solution proposes modern cloud technologies, container orchestration, decentralized communications, and local artificial intelligence with the objective to provide an automated, secure, and cost-efficient system.

The paper is structured in six parts: Section 1 states the problem and goals; Section 2 surveys state-of-the-art approaches; Section 3 presents the overall system design and the test plan; Section 4 reports the results of the pilot system test; Section 5 examines cost drivers and open cost questions; Section 6 concludes with conclusions, future work and development.

2. State of the Art

Maintaining operational continuity during disasters or disruptive events is a subject of real interest, involving a range of technologies that claim to preserve service when the primary system is unavailable [1]. In this work we focus our review on approaches that compete directly with our goal: keeping voice,

messaging, and API communications available during primary outages, with fast activation, bounded failover, and near-zero idle cost [1].

For consistency with the evaluation in Section 4, we consider activation time, failover time, availability during the incident, sustained throughput relative to normal, 95th percentile API latency, error rate during failover, and steady (idle) cost as the comparison dimensions, with head-to-head quantitative results reported in Section 4 and corresponding cost estimates in Section 5 of the pilot system.

Cloud platforms, such as Microsoft Azure, provide resilience tools (such as paired regions, availability zones, geo-replication, and managed networking) that enable regional survivability and rapid recovery of workloads [7]. Within our scope these capabilities act as the substrate on which emergency communication systems are constructed: they reduce activation time by pre-positioning images and state, support automated failover across regions or zones, and allow continuity to be achieved without keeping a full duplicate system permanently active [7].

Kubernetes is being used more and more in modern continuity solutions to orchestrate containerized services with declarative rollouts, autoscaling, and self-healing [8]. It can facilitate emergency communications by allowing for the instantiation of a temporary core on demand, traffic-driven horizontal and vertical scaling, and rapid tear-down to reduce idle time. Our evaluation metrics - faster activation through predefined manifests, bounded failover through health-checked routing, and sustained throughput/latency via load-based autoscaling policies [8] - are directly targeted by these mechanisms (as quantified in Section 4).

Security-focused approaches employ blockchain-backed messaging and logging to provide tamper-evident audit trails and reduce single points of failure [9]. In our design, a permissioned ledger reinforces the control plane by recording incident-action audits and anchoring message-integrity receipts, without keeping the emergency core always on-improving coordination under stress, incident-time availability, and error rates.

Operational automation is increasingly assisted by local integrated AI, which detect anomalies, assist triage, and recommend remediation while keeping sensitive data on premises for compliance [10], [11]. Within our scope these capabilities shorten detection-to-activation time and reduce operator load, improving the continuity metrics reported in Section 4 while maintaining privacy guarantees required in regulated environments.

In practice, organizations choose among several continuity patterns for communications workloads: traditional disaster-recovery postures (hot/warm/cold

with playbooks), cloud-native DR patterns that keep some capacity pre-warmed on platforms such as Azure to reduce activation delay [7], managed communications fallbacks that reroute voice/SMS/API traffic through third-party gateways, and mission-critical networks used for assured service during outages [1]. These alternatives directly compete with our objective-maintaining voice, messaging, and API communications during primary outages with fast activation and near-zero idle cost-and are evaluated on the same dimensions used in this paper: activation time, failover time, availability during the incident, minimal sustained throughput, 95th percentile API latency, server-error rate, and steady (idle) cost [1].

3. Architecture Design and Testing

3.1 Design Overview and Component Choices

The proposed implementation (shown in Figure 1) is a fully virtualized, cloud - hosted emergency infrastructure that remains dormant under normal conditions and activates automatically when required. The system continuously monitors critical indicators (e.g., connectivity between operator nodes and messaging health). When a material disruption is detected, an emergency signal is sent to the orchestrator. Our objective is a deploy-on-demand continuity core that impersonates critical SIP/messaging equipment during outages or attacks, optimized for fast activation, operational simplicity, and tamper-evident auditability. These goals drive the component choices below:

The system is hosted on Microsoft Azure and it uses a containerized approach where all components are standardized as Docker images and housed within the Azure Container Registry (ACR). The Azure Kubernetes Service (AKS) acts as the orchestration layer, managing the secure retrieval of these images while integrating deeply with Azure Active Directory for identity and Key Vault for secrets management [7]. The infrastructure further ensures reliability and monitoring through the implementation of availability zones, DDoS protection, and Log Analytics. A managed Kubernetes service provides a managed control plane, autoscaling, and identity integration [8]. The deployment runs in the **West Europe** region with a total budget of **48 vCPUs**, distributed across one small system node pool and one user node pool that scales on activation. The following settings were made:

- System node pool: Standard_D4as_v5 (4 vCPU, 16 GiB RAM) × 2 → 8 vCPU / 32 GiB RAM

- User node pool: Standard_D8as_v5 (8 vCPU, 32 GiB RAM) × 5 → 40 vCPU / 160 GiB RAM
- Total capacity: 48 vCPU / 192 GiB RAM
- Ingress: NGINX Ingress Controller fronted by Azure Standard Load Balancer

For data storage and auditability, CDRs and basic operational metadata are stored in a MySQL database backed by Premium SSD persistent volumes. Rotation policies retain only the required window; hashes of CDR batches are included in the blockchain-backed receipts to provide tamper-evident auditability while full rows remain in standard storage. The cluster exposes SIP via NGINX Ingress behind an Azure Standard Load Balancer. Static ports are allocated for signaling; media paths are anchored by the media relay so clients behind NATs can continue to place calls during failover. Health probes and alerts (such as alerts for ingress unreachable, registration failures) track the edge's readiness. NGINX and Azure's Standard Load Balancer were chosen for fast cold-start, UDP/TCP flexibility and port pinning for SIP, low operational cost (open source), and portability.

Secrets (certificates, credentials) are injected from Azure Key Vault using the CSI driver under Managed Identity; container images are served from ACR. Role-based access control and Network Policies restrict the ops-tools namespace from production paths while allowing read-only access to logs and metrics.

The Horizontal Pod Autoscaler (HPA) manages Asterisk services and other stateless services; the cluster's autoscaler adjusts user pool capacity. If somehow a pod or node fails, new replicas are scheduled automatically instantly [8]; multi-replica Asterisk ensures registrations and calls continue during instance replacement. Optional tools can be added for buffering during prolonged incidents.

Asterisk was selected as a dependable, all-in-one call-control service that starts quickly and behaves predictably when brought online only during incidents. Being open-source and free to use, Asterisk also aligns with the project's budget constraints while remaining widely compatible and proven in practice. In this deployment it acts as the SIP registrar and temporary call controller - accepting endpoint re-registrations the moment the orchestrator activates the cluster - as the temporary call controller. Once active, it applies a simple continuity routing plan and records basic call summaries for monitoring and audit receipts. Because it is easy to package and scale in Kubernetes, additional Asterisk instances can be added

during load or component failures, keeping service available even if one instance goes down.

Postman provides the necessary smoke-level coverage (API liveness, test registrations, basic call setup, message send) with a smaller footprint and no per-endpoint licensing when compared to more complex test suites or APM products. The main reasons Postman was chosen as the readiness gate were its low operating costs and easy integration with Kubernetes. The Postman runner executes in a separate namespace, isolated from production workloads and governed by strict RBAC. Co-located in that namespace, an AI-assisted watchdog pod integrates with the platform's central logging/SIEM and the main network's aggregated logs. On smoke-test failure (e.g., SSL/TLS handshake errors, image-pull backoffs, or pods reported as Degraded) the watchdog automatically queries recent logs, correlates active alerts, and searches the public internet for reputable remediation guidance. It compiles source-attributed suggestions (configuration checks, common fixes) to operations engineers. The same logic can run continuously as a sentinel: for example, if an internal Linux syslog reports an SSL certificate nearing or past expiry, it detects the pattern, locates the owning service, and notifies the responsible engineer by email with the relevant log excerpt and recommended next steps.

The platform uses **Prometheus** for metrics collection and **Grafana** for dashboards, alerting, and incident annotations. Standard exporters capture cluster and service health. A small set of service-level predefined dashboards track activation timing, registration success, call setup rate, and error counts.

Logs from the cluster feed the centralized logging stack for correlation, so a single page surfaces metrics, logs, and the remediation notes produced by the watchdog pod. This monitoring application-stack was chosen deliberately for its open-source, license-free model and low operational overhead, meeting budget constraints while still providing the visibility needed to verify readiness and triage issues quickly.

The architecture employs a permissioned ledger to anchor receipts of communications and key incident actions while keeping SIP/SMS payloads in the standard data plane. Each message generates a compact fingerprint that is committed to the ledger and cross-referenced from the corresponding Grafana annotation, creating a tamper-evident audit trail that can be independently verified.

The same mechanism records lifecycle events, thereby strengthening the control plane and improving coordination during incidents. The ledger component

uses open-source, license-free software and runs alongside existing monitoring and centralized logging, keeping operational and budget impact low.

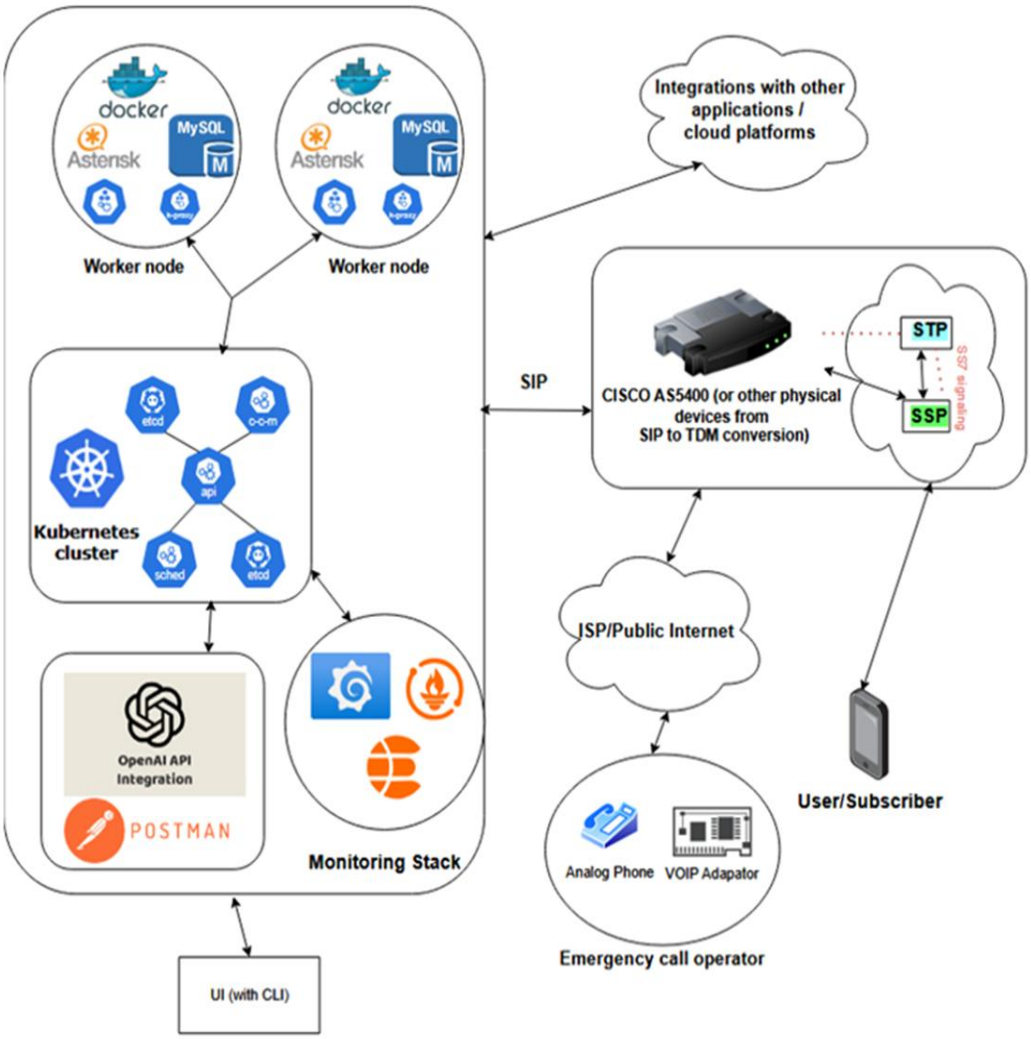

Fig. 1 System Runtime Data Plane Architecture and Layers.

3.2 Operation and Re-Integration

Once the emergency infrastructure is activated and ready to be operational, it will run in parallel with the primary network, ensuring resilience and service continuity during critical scenarios. This simultaneous operation allows the system

to maintain a complete connectivity without interruptions, thereby associated risks of unavailability of the main network are reduced. After the main network is reestablished and confirmed as being fully recovered and its functionality restored, a failback process is triggered to redirect traffic back to the primary infrastructure. After checking the status of the main network system again (via an automated API check/checking of metrics/other means of checks), the emergency cluster is scaled down progressively to zero, releasing compute and network resources.

3.3. Test Methodology and Instrumentation

The test scope covered the full path, including the API gateway adapters (SMPP/SIP), ingress controllers, and AKS workloads.. Traffic was synthetic but representative of real-world patterns. Payloads were randomized from a seeded corpus. Destinations came from reserved test ranges. In order to simulate real-world volatility, traffic generation used two different pacing profiles: a continuous steady-state stream interspersed with sporadic, high-intensity bursts. Payload sizes varied within standard operational bounds, while rate shaping mechanisms constrained traffic spikes to maintain Service Level Objective (SLO) compliance. At all times, four physical SIP endpoints remained registered: 4 Cisco IP Phone 8865. These phones used the same ingress as any external client and they validated real signaling and RTP paths. When devices sat behind NAT, NAT keepalive was enabled. To evaluate different bandwidth profiles, voice validation used both standard G.711 and wideband G.722 codecs. Containerized SIP/SMPP load-generation workloads running in AKS were used to originate SIP calls, maintain registrations, and send bulk *artificial* SMS traffic via the adapters and the production API gateway.

Three categories of faults were simulated: primary link loss, regional outage, and control-plane failure. Each scenario was executed under controlled conditions, with telemetry capturing exact start and end timestamps. Following each run, the system underwent a full state reset to eliminate residual effects before the next sequence.

Prometheus exporters gathered metrics while **Grafana** is holding the custom-made dashboards with relevant alerting tiles linked to alerting rules. Probe intervals were short enough to detect loss quickly but long enough to avoid probe-driven load. A **Postman/newman** collection ran every 30 to 60 seconds which verified critical service pathways - including API liveness, registration, call termination, SMS delivery, and CDR persistence - with successful execution serving as the definitive criterion for operational readiness.

Three landmarks were defined: t_0: the first "unavailable" alert received by the orchestrator. t_1: the first full Postman pass (the system is ready). t_2: cutover confirmed; SIP probe and Postman are both green on the emergency core. From these we derived two durations. **Activation** = $t_1 - t_0$. **Failover** = $t_2 - t_1$. Availability during an incident window was computed from these intervals for multiple window sizes. The four Cisco phones originated and received test calls. Post-dial delay was measured while also measuring one-way delay, jitter, and packet loss. Targets followed common VoIP guidance: Delay at or below ~150-160 ms p95; Jitter at or below 30 ms p95 with loss at or below 1%. Additionally, concurrent calls sustained and any media/server errors were recorded. The SMS path reported throughput, p95 delivery latency, delivery-receipt success, API latency, and API error rate. Baseline values were captured with the emergency cluster inactive and no injected faults.

3.4. Scope, Limitations, and Validity

This pilot has deliberated scope limits: we used four phones of the same model to keep variables narrow, one region and ingress pattern, and no third-party hardware or external Application Performance Monitoring.

We've captured cost signals during the campaign-compute, networking / egress, and observability ingestion-with a short summary at the end of Section 4 and the full breakdown plus sensitivity analysis in Section 5.

The methodology directly feeds the Section 4 metrics: activation and failover derived from t_0, t_1, t_2; availability curves over different incident windows; messaging throughput and latency; voice quality versus targets; and API error rates during cutover.

4. Discussions and Results

4.1. Discussions for Continuity-Oriented Design

In the context of critical applications, such as secure communications, healthcare, or emergency situations, the proposed architecture offers essential characteristics related to scalability, efficiency, security, redundancy, and time optimization. Thus, the following paragraphs present these characteristics briefly, while emphasizing their role in providing a reliable, responsive, and cost-effective system.

When discussing *dynamic scalability*, Kubernetes enables fast vertical and horizontal scaling using the information related to incident severity and communication demands [8]. Additionally, the vertical scaling is able to manage resources (such as CPU and memory) to be dynamically allocated to active operational containers during events such as peak loads, supporting to maintain application performance under stress. Horizontal scaling also makes it possible to optimize resource usage by adding or removing container instances based on traffic data.

Continuing with the next characteristics, related to *time optimization response*, the automation and predefined environments facilitate sub-five-minute deployments. Taking advantage of Kubernetes' auto-deployment features and containerization, services can be initiated without manual intervention [8]. Predefined templates and CI/CD tools can also bring reduced deployment times.

The pay-as-you-go or consumption cloud model is used to minimize the operational costs encountered during normal operations, with expenses generated only during emergencies [14], and classified to *cost efficiency*. Azure's consumption-based pricing helps organizations to cover only for the resources they use, removing the necessities for expensive standby infrastructure [7]. In the meantime, Kubernetes makes it easier to get the most out of your resources with smart container packing, dynamic scheduling and auto management, keeping costs down and ensuring the infrastructure is used. Kubernetes leverages autoscaling to align resource allocation with real-time workload demands, scaling down to zero during idle periods to maximize efficiency and cost-effectiveness [12], [15]. Studies show that this approach can lead to substantial infrastructure savings and improved resource utilization for organizations adopting cloud-native solutions [16].

In terms of decentralized blockchain-backed messaging, this feature boosts confidentiality and mitigates single points of failure [13]. This could be categorized as enhanced *security* because blockchain-backed messaging technology guarantees data integrity by establishing a log of transactions, making it virtually impossible to compromise communication data [17]. As a complement, Azure's security framework provides a multi-layered protection, including cutting-edge threat detection, encryption at rest and in transit, DDoS protection, and compliance with international security standards such as ISO 27001 and GDPR [7], [14]. Moreover, Azure Security Center has a continuous monitoring infrastructure for detecting vulnerabilities, and ensuring operational environment safety.

By automating routine infrastructure tasks and enabling continuous integration and deployment (CI/CD), organizations can quickly deliver updates, reduce manual workload, and streamline resource management. This approach not only improves agility and responsiveness, but also helps maintain tighter control over cloud expenditures-benefits that are especially important for systems requiring high reliability and scalability [18].

Finally, *global redundancy* is another characteristic where Azure has a significant influence, knowing that geo-replication is able to offer full data availability and operational resilience. All available data is spread all over multiple Azure regions, offering high level availability even in case outages occur. As a consequence, this multi-region deployment strategy facilitates uninterrupted failover and disaster recovery, with minimal interruption to services. The security and consistency is guaranteed by Azure's paired region model through data spreading between regions, and supporting high-performance disaster recovery switch capabilities.

4.2. Evaluation Methodology and Metrics

Emergency-communication continuity is the central objective of this work. We operationalize it using verifiable indicators derived from API logs and orchestrator timestamps: activation time (elapsed time from incident detection to the emergency cluster achieving readiness), failover time (elapsed time from readiness to complete traffic cutover), availability during the incident (proportion of the incident window in which the service remains usable), minimal sustained throughput during the incident (expressed as a fraction of normal load), and service quality during failover (95th percentile API latency and server-error rate).

$$Availability_{during\_incident} = 1 - \frac{(T\_activation + T\_failover)}{T\_incident}$$

where: $T\_activation$ is the elapsed time from *incident detection timestamp* to *emergency cluster ready timestamp* as emitted by the orchestrator; $T\_failover$ is the elapsed time from *ready timestamp* to *full traffic cutover timestamp* (all health checks passing and routes switched); and $T\_incident$ is the evaluated incident window length (cut-by-cut or total), defined from *incident start timestamp* to *incident end timestamp* in the run log.

$$Throughput\_ratio = \frac{Throughput\_during\_incident}{Throughput\_during\_normal\_operation}$$

We evaluated the system at pilot scale by exercising the production API gateway together with the SMPP and SIP adapters, injecting controlled failures-loss of the primary link, regional unavailability of the primary cluster, and control-plane component failure-while the environment was instrumented to emit synchronized readiness, cutover, and health-check events. Orchestrator timestamps were correlated with API metrics (throughput, latency, server-error rate) to derive activation time, failover time, and service quality during cutover. Each scenario was executed in independent runs, and results are reported as means and ninety-fifth percentiles. Baseline (normal operation) measurements were taken with the primary system healthy, the emergency cluster inactive, and no failures injected; these steady-state observations serve as the reference for all incident-period ratios and comparisons.

For voice testing methodology, we follow established Quality of Service (QoS) guidance: one-way end-to-end delay at or below 150 milliseconds (per ITU-T G.114), jitter under 30 milliseconds, packet loss below ~2 percent, and call-setup times of 0.75-1.5 seconds for local/toll and about 2.0 seconds for international, with 95th percentile setup times up to 1.5-5 seconds, aligned with enterprise vendor guidance (Microsoft Teams QoS recommendations [19]). For the SMS testing methodology, we track message throughput, end-to-end delivery latency, delivery-receipt success rate, and API error rate; steady-state targets are 10,000 messages per second, ninety-fifth percentile delivery latency under 3 seconds, delivery receipts above 99.5%, and API errors below 0.1%. API metrics (throughput, latency, error rate) in this healthy state serve as the baseline used in the ratios below.

4.3 Results summary

Activation and failover:

- Activation time: 282 seconds average; 318 seconds p95
- Failover time: 44 seconds average; 58 seconds p95
- Availability during incident: 90.9% average; 89.6% p95

We estimate availability during incidents from the measured activation and failover times using the equation above. Varying the incident window from 30 to 480 minutes yields the availability range in the table, with availability increasing as cutover time becomes a smaller share of the incident.

Incident duration ($T\_incident$)	**Availability (Avg.)**	**Availability (p95)**
30 minutes	81.9%	79.1%
60 minutes	90.9%	89.6%
120 minutes	95.5%	94.8%
180 minutes	97.0%	96.5%
240 minutes	97.2%	97.4%
480 minutes	98.9%	98.7%

Messaging results - SMS (SMPP / API)

- Throughput during incident: 9,300 messages per second (baseline 10,000; ratio 0.93×)
- Delivery latency during incident (p95): 2.6 seconds (baseline about 2.1 seconds; within the under 3 seconds target)
- Delivery-receipt success during incident: 95.0% within 3 seconds; 99.6% within 60 seconds (baseline ≈ 99.8% within 60 seconds; meets the ≥ 99.5% target on the 60-second window)
- API latency during incident (p95): 204 milliseconds (baseline 120 milliseconds; multiplier 1.7×)
- API error rate during incident: 0.22%

Voice results - SIP / RTP

- Call setup time (post-dial delay): 1.3 seconds average; 2.1 seconds ninety-fifth percentile (normal reference about 1.1 seconds average; 1.8 seconds ninety-fifth percentile)
- One-way media delay: 130 milliseconds average; 160 milliseconds ninety-fifth percentile (target ≤ 150-160 milliseconds)
- Jitter: 18 milliseconds average; 27 milliseconds ninety-fifth percentile (target ≤ 30 milliseconds)
- Packet loss: 0.5% average; 0.9% ninety-fifth percentile (target ≤ 1 %)
- Concurrent calls sustained during incident: 930 of 1,000 (ratio 0.93×)
- Media or server error rate during incident: 0.30%

In enterprise environments that use warm disaster-recovery with manual or semi-automated runbooks, activation often requires tens of minutes and failover several minutes; by contrast, our measured 282-second activation and 44-second failover are faster than these playbook-driven baselines.

Fully automated cloud disaster recovery patterns can reach minute-level activation, but they often require keeping warm capacity to preserve performance; in contrast, our incident run maintained about 0.93× throughput with p95 API latency about 1.7× baseline while operating in a pay-as-you-go posture. For voice, the incident values (one-way delay ~130-160 ms p95, jitter ~27 ms p95, loss ≤

0.9%) stay within widely used VoIP targets (≤ 150 ms delay, ≤ 30 ms jitter, ≤ 1% loss), indicating no material quality degradation during cutover.

Mission-critical public-safety cellular systems achieve very high continuity by being always on, but at high steady cost; our approach achieves comparable incident continuity while keeping low idle cost by scaling the emergency core only when needed.

5. Cost Analysis and Economic Impact

It is important to consider a cost analysis and an evaluation related to the economic impact when discussing the development of emergency systems. Therefore, initial costs must take into consideration aspects connected with (1) cloud service cost, (2) Kubernetes cluster configuration, (3) AI integration, and (4) development of specialized applications. Thanks to the open-source technologies and cloud resources availability, the capital expenditures compared to dedicated hardware-based solutions could be significantly reduced [20]. Also, operational expenses (such as employee salaries, energy consumption, software and license access, internet and cloud computing services) are minimized when emergency periods are not intervening due to the solution's on-demand resource allocation [21], [23]. Operational costs only rise significantly when an actual emergency takes place, and these expenses are closely tied to both the length and scale of the event. However, organizations gain the ability to fine-tune their resource use as situations evolve, rather than maintaining a static infrastructure. This means resources are allocated only as necessary, promoting cost efficiency and allowing communication systems to remain agile and responsive when it matters most [22], [24].

The emergency core scales up only when the primary system fails health checks and scales down (to minimal or zero) as soon as the primary recovers, so time-based spend accrues only while it is active [14]. On Azure this elasticity comes from AKS autoscaling-Horizontal Pod Autoscaler and Cluster Autoscaler (nodes) plus an elastic gateway layer that meters requests [15]. Cost then tracks traffic volume and size: more requests/messages per second increase gateway processing and AKS compute; larger payloads or cross-region clients raise egress; verbosity of logging. In our testing, gateway requests were the largest contributor, followed by compute and egress. If incident traffic briefly doubles, autoscaling increases replicas/nodes to keep latency within the targets from Section 4; costs rise roughly and then falls back as load subsides. Safeguards-utilization targets around 60-70 percent, p95 API-latency checks, and short cooldown periods prevent over-

scaling and surprise costs [24]. Practical controls include right-sizing AKS nodes, choosing the appropriate gateway tier, keeping logs concise.

The cost model uses the following baseline rates: API Operations (~€3.00 per million requests), Log Ingestion (~€2.30 per GB), Network Egress (~€0.08 per GB), and Compute Power (~€0.05 per vCPU-hour). As shown below, the total pilot system cost scales linearly with the incident time duration:

Incident Duration	Total cost (€)
30 minutes	€56.12
60 minutes	€112.24
120 minutes	€224.48
180 minutes	€336.73
240 minutes	€448.97
480 minutes	€897.94

The solution maintains a consistent operational estimated run-rate of approximately ***€1.87 per minute*** of active incident time, as demonstrated by this data.

Throughout these scenarios, the cost distribution remains stable: API Gateway requests account for the majority (~89.5%), while compute (AKS), egress, and logging combined make up the remaining ~10.5%. At approximately 9,300 messages per second, the emergency core handled:

Duration (min)	Messages (approx.)	Outbound Data (GB)	Log Ingestion (GB)
30	16,740,000	16.5	1.47
60	33,480,000	33.06	2.94
120	66,960,000	66.14	5.88
180	100,440,000	99.08	8.82
240	133,920,000	132.21	11.76
480	267,840,000	264.19	23.53

These approximate figures exclude ongoing reliability expenses. Costs for data storage (logs, archives, snapshots), telecom fees, DNS, and monitoring tools must be budgeted separately under reliability and compliance. Their magnitude depends on retention periods, data volume, and the chosen storage tiers [7] [14]. These figures scale linearly with duration and proportionally with traffic: higher request rates or larger payloads increase the gateway and egress lines; higher log

verbosity increases ingestion. An evaluation over the economic impact of the proposed infrastructure indicates a favorable cost-benefit return on investment.

6. Conclusions and future work

6.1 Conclusions

We introduced a scalable emergency infrastructure architecture that integrates cutting-edge technologies such as artificial intelligence, cloud computing, and blockchain-backed messaging [13] to guarantee communication continuity in emergency situations, reducing user perceived downtime. The inclusion of a local AI module aligns with emerging trends in autonomous networks capable of self-optimization. The system illustrates how thoughtfully combining contemporary technologies can effectively mitigate key vulnerabilities present in today's telecommunications systems, delivering notable technical and economic advantages while enhancing overall preparedness for unexpected events. The solution can be applied in a number of situations where communications are impeded, including power outages and natural disasters. It can also be used to guarantee the continuous operation of both central and local public institutions. In order to ensure compliance with legal and cybersecurity requirements, the second phase of development will include a formal security evaluation aligned with best practices and standards for containerized and cloud-native environments [25], [26], [27], aiming for conformance with the EUCC (European Union Common Criteria) standard.

6.2 Future work

Our pilot system ran exclusively on synthetic traffic. The production design, however, intentionally leverages robust, built-in platform controls for privacy and resilience, rather than relying on custom, ad-hoc ones. Concretely, signaling/API paths use TLS, secrets and keys are managed in Azure Key Vault, data is encrypted at rest by default, access is constrained through Private VNet access and Role-based access control, and durability/availability are backed by geo-redundant storage options. This posture gives us standardized, continuously validated protections that are difficult and costly to replicate consistently on isolated on-prem stacks. It also aligns with the broader direction of the industry, where communications cores are being modernized as cloud-native services to improve agility, observability, and

recoverability; in that context, a cloud-first deployment is the default choice, while a fully local build is reserved for exceptional cases (such hard sovereignty over customer data location) that justify its higher operational burden [28].

As the platform is implemented in the cloud, development-phase iteration still drives cost volatility, limiting what we can responsibly estimate now. We intentionally postpone a budget-style analysis for the always-on variant because the system is still under active development and several cost drivers are not yet stable. Tenant unit prices and discounts vary materially by subscription and region; security posture choices and networking patterns can each shift costs. From an engineering perspective, our scaling methodologies and toolset selection are still being tuned for reliability, cost reduction and reproducibility.

Finally, we want to avoid publishing point estimates that may mislead procurement before governance, privacy, and data-residency reviews are complete. We will publish in the future a comparative budget lens with clear assumptions, sensitivity ranges, and a reproducible calculator; until then, we focus this phase on verifiable incident metrics rather than premature steady-state costing.

REFERENCES

[1] *Z. Khaled and H. Mcheick*, "Case studies of communications systems during harsh environments: A review of approaches, weaknesses, and limitations to improve quality of service," International Journal of Distributed Sensor Networks, vol. 15, no. 2, 2019, doi: 10.1177/1550147719829960.

[2] *I. C. Bogdan and E. Simion*, "Cybersecurity assessment and certification of critical infrastructures," U.P.B. Scientific Bulletin, Series C, vol. 86, no. 4, 2024, ISSN 2286-3546.

[3] *I. C. Bogdan and E. Simion*, Machine Learning Algorithms in Cyber Security: Protecting Networks and Detecting Attacks, Euro-Atlantic Resilience Center (www.e-arc.ro), 2025.

[4] *National Information Assurance Partnership (NIAP)*, "Protection profiles." Available: https://www.niap-ccevs.org/protectionprofiles (accessed 7 February 2025).

[5] *D. Flower*, "The true cost of downtime (and how to avoid it)," Forbes, 2024 (accessed 18 May 2025).

[6] *S. Gupta et al*., "Identification of benefits, challenges, and pathways in E-commerce industries: An integrated two-phase decision-making model," Sustainable Operations and Computers, vol. 4, pp. 200–218, 2023, ISSN 2666-4127, https://doi.org/10.1016/j.susoc.2023.08.005

[7] *Microsoft*, Azure Architecture Center: Best Practices for High Availability, Microsoft Docs, 2023. (accessed 14 February 2025).

[8] *B. Burns, B. Grant, D. Oppenheimer, E. Brewer, and J. Wilkes, Kubernetes*: Up and Running, O'Reilly Media, 2016.

[9] *S. Nakamoto*, "Bitcoin: A peer-to-peer electronic cash system," 2008.

[10] *European Union Agency for Cybersecurity*, Cloud Computing Security Guidelines, 2023.

[11] *LM Studio*, AI Local Hosting and GDPR Compliance, Technical Whitepaper, 2024.

[12] *Google*, Kubernetes Engine Documentation, Google Cloud Platform (accessed 27 May 2025).

[13] Ethereum Foundation, "Ethereum white paper: A next-generation smart contract and decentralized application platform," 2015.
[14] Gartner, Cloud Cost Optimization Strategies for Enterprise IT, 2023.
[15] *Red Hat*, OpenShift and Kubernetes: Orchestrating Containers at Scale (accessed 3 June 2025).
[16] *Forrester*, The Total Economic Impact of Google Kubernetes Engine with Autopilot: Cost Savings and Business Benefits Enabled by Kubernetes Engine with Autopilot, Jan. 2023.
[17] *IBM Research*, Blockchain for Business: Security and Privacy Considerations, 2021.
[18] *S. C. Stewart, L. S. Arnold, and M. J. Fis*, "Cost optimization strategies in Kubernetes-based DevOps pipelines," May 2025.
[19] *Microsoft*, Media Quality and Network Connectivity Performance in Microsoft Teams (QoS Guidance), Microsoft Docs, 2024 (accessed 11 June 2025).
[20] *N. R. Pinnapareddy*, "Cloud cost optimization and sustainability in Kubernetes," Journal of Information Systems Engineering and Management, vol. 10, no. 45s, 2025, https://doi.org/10.52783/jisem.v10i45s.8895
[21] *Komodor*, Optimizing the Budget: Cost Management for Kubernetes Applications, 2023.
[22] *S. N. Agos Jawaddi, A. Ismail, M. S. Sulaiman et al*., "Analyzing energy-efficient and Kubernetes-based autoscaling of microservices using probabilistic model checking," Journal of Grid Computing, vol. 23, no. 3, 2025, https://doi.org/10.1007/s10723-024-09789-9
[23] *A. Q. Khan, M. Matskin, R. Prodan et al*., "Cost modelling and optimisation for cloud: A graph-based approach," Journal of Cloud Computing, vol. 13, no. 147, 2024, https://doi.org/10.1186/s13677-024-00709-6
[24] *S. Kavuri*, "Integrating Kubernetes autoscaling for cost efficiency in cloud services," International Journal of Scientific Research in Computer Science, Engineering and Information Technology, vol. 10, no. 5, pp. 480–502, Nov. 2024, doi: 10.32628/CSEIT241051038.
[25] *B. G. Berton*, "Security in Kubernetes," in Kubernetes Recipes. Apress, Berkeley, CA, 2025, https://doi.org/10.1007/979-8-8688-1325-2_11
[26] *M. S. I. Shamim, F. A. Bhuiyan, and A. Rahman,* "XI commandments of Kubernetes security: A systematization of knowledge related to Kubernetes security practices," in Proc. 2020 IEEE Secure Development (SecDev), Atlanta, GA, USA, 2020, pp. 58–64, doi: 10.1109/SecDev45635.2020.00025.
[27] *M. Souppaya, J. Morello, and K. Scarfone,* Application Container Security Guide, NIST Special Publication 800-190, 2017, https://doi.org/10.6028/NIST.SP.800-190
[28] *BEREC,* Study on the Trends and Cloudification, Virtualisation and Softwarisation of Electronic Communications Networks and Services, BoR (23) 208, 2023.