Quantifying the Performance of Cloud-Oriented Container Orchestrators on Emulated Tactical Networks

T. Kudla, M. Fogli, S. Webb, G. Pingen, N. Suri and H. Bastiaansen

IEEE Communications Magazine, 2022

Cite this as

T. Kudla, M. Fogli, S. Webb, G. Pingen, N. Suri and H. Bastiaansen, "Quantifying the Performance of Cloud-Oriented Container Orchestrators on Emulated Tactical Networks," in *IEEE Communications Magazine*, vol. 60, no. 5, pp. 74-80, May 2022, doi: 10.1109/MCOM.003.00975.

Quantifying the Performance of Cloud-Oriented Container Orchestrators on Emulated Tactical Networks

Thomas Kudla, Mattia Fogli, Sean Webb, Geert Pingen, Niranjan Suri, Harrie Bastiaansen

***Abstract*—Kubernetes is the de facto industry standard for orchestrating containerized applications in clouds. Performance analyses of Kubernetes are often conducted in resource-rich environments on the enterprise or data center level. At the tactical edge domain, network resources are scarce and therefore at the opposite end of the spectrum when compared to resources available to enterprises or data centers. Computing resources, e.g., mobile or edge devices, are typically interconnected via wireless communications links that are characterized by limited bandwidth, low reliability, high latency, and frequent disconnections. This experience article describes lessons learned in constructing a suitable emulation environment for experimentation and our findings and recommendations for running Kubernetes in tactical networks.**

I. Introduction

CONTAINERIZATION and microservice architectures are rapidly replacing traditional approaches to software development, such as big monolithic applications deployed in virtual machines [1]. However, as the number of deployed containerized applications increases, so does the complexity behind their lifecycle management. In this regard, several container orchestration solutions have emerged, e.g., Docker Swarm, Mesos, Nomad, and Kubernetes. Currently, Kubernetes is the de facto industry standard for orchestrating containerized applications across multiple nodes in cloud environments. The overall complexity increases because an orchestrator adds an additional layer to the current technology stack. However, such extra complexity could be compensated by potential benefits [2].

Cloud environments may provide similar benefits to the military domain as to the public domain (e.g., data processing, survivability [2]), and therefore improve the effectiveness of military missions [3]. The Information Systems Technology (IST)-168 Research Task Group (RTG) on *Adaptive Information Processing and Distribution to Support Command and Control* was formed in 2018 to investigate an adaptive and federated cloud architecture for the military domain. In military missions and even more in multi-national missions, a federation of cloud services becomes increasingly more relevant. In fact, using (or even combining) available resources from partner nations may improve the overall information availability and resiliency [4]. As a result, mission participants rely on up-to-date information from different sources and an efficient way to exchange and leverage the available data to support their mission goals. To fully utilize each participant's infrastructure, a federation at the service level is desirable. This means that participants may invoke services on partner infrastructures according to the policies in place. The IST-168 RTG supports that approach by developing scenarios for testing cloud environments in military mission contexts and by defining programming interfaces for federating such clouds on the service level. From a technical perspective, it became clear through our work that a container orchestration solution is necessary to enable an adaptive and federated cloud architecture. A container orchestration solution for military mission contexts should cope with disadvantaged network conditions. Therefore, such a solution must take as little bandwidth as possible, tolerate long timeouts due to high latency, and apply efficient re-connection mechanisms to cope with unreliable links.

In this context, a system of computing resources being available 'on-demand' is defined as a cloud. On the one hand, the physical space where computing or storage resources are located, either at an established facility, such as headquarters or inside a vehicle, is a factor for sizing a cloud. On the other hand, another sizing factor is the adequate 'intra-cloud' connectivity (i.e., network resources within a cloud for node-to-node communications) and the 'inter-cloud' connectivity (i.e., cloud-to-cloud or cloud-to-edge communications) [4]. Fig. 1 illustrates cloud-to-cloud and cloud-to-edge communications in a federated cloud infrastructure between two nations. In particular, partner clouds are deployed at headquarters and at the level of platforms (e.g., vehicles, ships, and aircraft), whereas dismounted soldiers connect as edge nodes.

Nations are starting to adopt cloud environments in military contexts. For example, in January 2021 the North Atlantic Treaty Organization (NATO) has selected the company Thales to provide a defense cloud solution. This cloud solution should be deployable from the headquarters level down to a forward base level [5]. The tactical edge level, primarily consisting of dismounted soldiers and vehicles on the move, is not integrated into this cloud environment. Similarily, in September 2021, the European Commission started a project for a "Military multi-domain operations cloud". Besides many features, this cloud should support federated services [6].

Cloud environments usually rely on networks characterized by high bandwidth, low latency, and high reliability. At the other end of the spectrum, military mission networks, called Tactical Networks (TNs), are characterized by limited bandwidth, low reliability, high latency, and frequent disconnec-

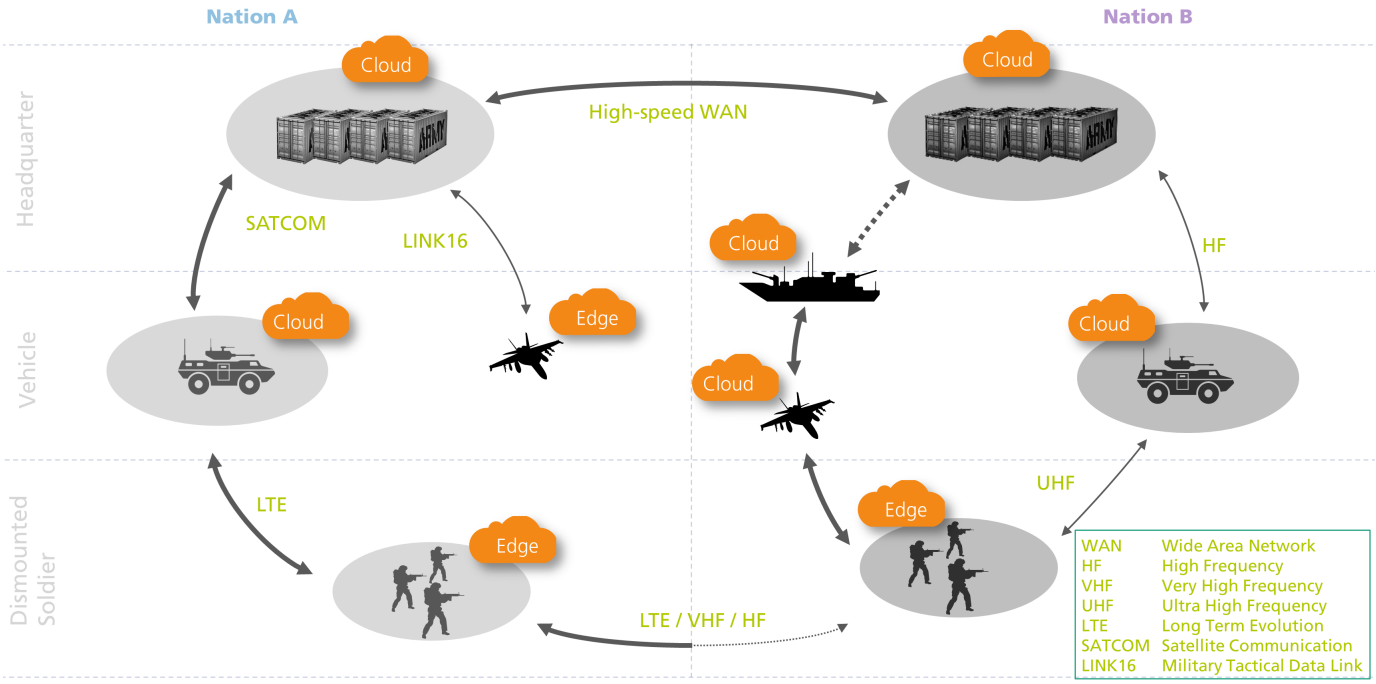

Fig. 1. Schematic overview of a federated cloud infrastructure between two nations.

tions. This is due to environmental constraints, wireless dynamics and mobility, and both intentional and non-intentional interference [7]. Therefore, it is a challenging, tedious, and expensive process to setup environments emulating (part of) those factors reliably. In [8], the authors describe a testbed for functional testing of complex information technology systems within TNs. They use a combination of existing physical military hardware (e.g., radios) and emulated components that interact with each other to run experiments on that setup. Following a similar approach, a dynamically configurable emulated environment has been set up for our experiments. The underlying network uses the Extendable Mobile Ad-hoc Network Emulator (EMANE) [9] to emulate a TN interconnecting cloud nodes.

In our previous work, we described the potential benefits of cloud environments for the military domain and depicted a federated cloud architecture to support adaptive command and control [4]. Then, we evaluated various Kubernetes distributions to enable the intra-cloud capabilities of the individual clouds jointly constituting the federated cloud architecture [10], [11]. This experience article lays out i) lessons learned in constructing a suitable emulation environment, ii) how to design experiments for quantifying the performance of Kubernetes in TNs, and iii) findings and recommendations on how to deploy Kubernetes-based clouds at the tactical edge.

II. Experimentation Environment

We used Kubernetes as the container orchestration technology for the experiments. There are several Kubernetes distributions available [12], each designed with a slightly different use case in mind. Specifically, we chose the vanilla Kubernetes distribution, i.e., K8s [13], which takes for granted robust and high-capacity links. Additional Kubernetes distributions used for the experiments are Lightweight Kubernetes (K3s) [14] and Kubernetes Native Edge Computing Framework (KubeEdge) [15]. Both are specifically designed for resource-constrained environments.

A. Experiment goals

In the military context, where resources cannot be taken for granted, a cloud environment depends on the resources available on site. The availability of such resources may change over time or depend on a specific military mission setup. Therefore, it is necessary to be able to re-configure these resources on demand or even set up a new cloud environment from scratch with new resources. This leads to our two main research questions:

- What are the minimum requirements for a Kubernetes distribution to successfully initialize a cloud in TNs (cloud initialization)?
- What are the minimum requirements for a Kubernetes distribution to successfully deploy containerized applications in a cloud in TNs (application deployment)?

We answered those questions experimentally by evaluating a broad range of experiment settings that may affect a cloud environment. The purpose of the experiments is to capture the overhead introduced by a given Kubernetes distribution while accomplishing orchestration-related tasks, such as cloud initialization and application deployment. As a result, we did not consider application-specific traffic, which is beyond the scope of this article. In fact, application developers should tailor applications for a given scenario by also considering the infrastructure overhead. In the case of the adaptive and federated cloud architecture mentioned above, such an infrastructure overhead includes the bandwidth penalty for running a container orchestration solution.

As Fig. 2 shows, cloud initialization and application deployment have two phases. The bootstrapping phase occurs first and concludes when the cloud's desired state matches the cloud's current state. In cloud initialization experiments, the desired state is to initialize a cloud with a given number of nodes. In application deployment experiments, the desired state is to deploy a given number of application replicas within the cloud. The steady state phase follows the bootstrapping one. Since the cloud's desired state matches the cloud's current state, no actions are expected to be performed during the steady state phase. Note that both those phases are crucial to assessing a given task's impact. Precisely, the bootstrapping phase quantifies the peak of resources demanded to reach a given cloud state, whereas the steady state phase quantifies the resources taken to maintain such a state.

This leads to more specific questions about how a given Kubernetes distribution performs in TNs. In particular:

- How do different experiment settings impact cloud initialization and application deployment duration?
- How do different experiment settings impact the average data rate during cloud initialization and application deployment?
  - What is the average data rate of cloud initialization, first for the phase when the cloud is being initialized (bootstrapping) and second when the cloud is initialized (steady state), as in Fig. 2?
  - What is the average data rate of application deployment, first for the phase when the applications are being deployed (bootstrapping) and second when the applications are deployed (steady state), as in Fig. 2?
- What experiment setting or combination of settings has the most impact on the performance of a given Kubernetes distribution in TNs?

B. Experiment settings

The experiment settings vary along the following dimensions:

- Distribution, i.e., the underlying Kubernetes distribution used as the container orchestrator;
- Cloud size, i.e., the number of nodes participating in a cloud, where one node acts as the master and the others as workers;
- Communications effects, i.e., limiting bandwidth, increasing latency and packet loss;
- Cloud workload, i.e., the number of applications being deployed in the application deployment experiments.

These six dimensions, counting all three communications effects, would lead to a very high number of experiments. For example, if every dimension has three values, then $3^6 = 729$ experiments each with a unique setting needed to be done. An experiment run is successful if the bootstrapping phase succeeds within a specific time frame. Due to the non-deterministic behavior of EMANE (e.g., a packet drop that needs to be re-transmitted), consecutive runs based on the same experiment settings might not fail consistently. In this regard, we decided to perform three runs for each experiment settings. By doing so, we could identify edge cases, i.e., those experiments whose settings result in alternated successes and failures. An experiment (three runs) took an hour roughly to be performed in our environment. As a result, 729 experiments would take about a month to run sequentially. To provide meaningful results in a reasonable time we needed to make a trade-off between time spent on running experiments and possible experiment settings.

In our first evaluation of Kubernetes in TNs [10], we narrowed the number of experiments down by limiting the possible values for each dimension. For the distribution, there were only three values, i.e., K8s, K3s, and KubeEdge. The number of nodes was six, twelve, eighteen, and twenty-four. The cloud workload was either ten or twenty replicas per worker node. The communications effects were split into four network scenarios, starting from a baseline, which is an unconstrained 1 GBits Local Area Network (LAN), and continuing to a best (BEST), to an average (AVG), and to a worst (WORST) scenario. From LAN to WORST, the link quality was degraded by fixed values in bandwidth, latency, and packet loss simultaneously. The parameter values defining such network scenarios were chosen based on modern military radios that are either currently in use or proposed for the near future in TNs.

In our second evaluation [11], we focused on K8s because it performed slightly better then K3s and KubeEdge builds upon K8s. Also, we fixed the number of nodes to six because results from our previous work showed that K8s worked for cloud initialization and application deployment at least in the AVG scenario with at most six nodes. Additionally, instead of having all the network parameters (i.e., bandwidth, latency, and packet loss) fixed for a network scenario, two were fixed and the remaining one was incrementally degraded in distinct steps.

In both evaluations, the available network bandwidth was equally distributed among all nodes. For instance, in an experiment involving six nodes under the AVG scenario, each node could take up to 333.33 Kbps, whereas for twelve nodes, the maximum bandwidth per node would have been 166.66 Kbps. Table I shows the experiment settings used in both evaluations.

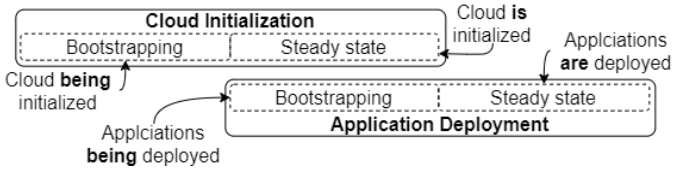

Fig. 2. Experiment phases.

C. Physical components

The testbed is composed of two physical servers, as illustrated in Fig. 3. The Cloud Node (CN) server is responsible for hosting up to 24 virtual machines with Ubuntu as operating system. Each virtual machine is connected to two networks, one for data traffic used by Kubernetes for accomplishing orchestration-related tasks, and one for control traffic used

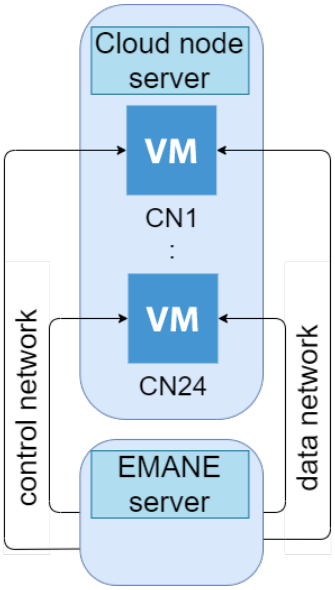

Fig. 3. Testbed setup.

to start and stop experiments and collect results. The other server runs EMANE, which is an open-source framework for emulating real-world wireless network conditions. The CNs are connected to each other through EMANE. For example, a connection from CN 1 to CN 24 has to pass through EMANE, which applies necessary communications effects, such as bandwidth limits, packet delay, or packet loss. Additionally, that server is also used to control the experiments and collect data from the nodes by using the control network. A detailed technical description of the setup can be found in [10], [11].

III. Running and Analyzing the Experiments

Several tools have been used to run the experiments and gather data. The experiment settings were configured and managed by Ansible. Ansible is an information technology automation tool. It can configure systems and deploy software by means of configuration files called playbooks. Such playbooks are written in a human-readable markup language, allow for parameter substitution at run time, and specify tasks to run on the CNs. One task is to log network traffic with TCPDUMP, a network packet analyzer capable of intercepting and storing raw IP packets that are being sent or received by the system it is running on. The complete set of tasks that run on CNs included preparatory downloading of Kubernetes and container images, enabling communications effects in EMANE, logging network information with TCPDUMP, initializing the cloud, deploying applications to the cloud, and cleanup of the environment to prepare for the next run of a given experiment. Parameter substitution enabled the reuse of the playbooks and simplified the iterative changes to settings such as the available network bandwidth over each experiment run. The use of Ansible allowed multiple experiment runs to be automated in a controlled manner. This automation helped us to reduce the number of experiments that have to be run, which is preferable as stated in II-B. We were able to skip runs during the experimentation by monitoring the number of failed runs per experiment setting. If all three runs failed for the current experiment setting, then successive runs would also fail. This is because successive runs would have worse settings since we run our experiments from unconstrained to constrained settings. Therefore, such runs could be skipped.

TABLE I
Experiment Settings

Experiment Settings	Values [10]	Value(s) [Step] [11]
Distribution	K8s, K3s, KubeEdge	K8s
Container runtime	containerd	CRI-O
No. CNs	6, 12, 18, 24	6
No. replicas/worker	10, 20	10, 20
Network scenario	LAN/BEST/AVG/WORST	LAN, AVG
Bandwidth (Mbps)	- 4 2 0.65	0.5 - 4 [0.5]
Latency (s)	- 0.2 0.4 3	0 - 8 [0.4]
Packet loss (%)	- 5 10 15	0 - 40 [2]

A. Running the experiments with Ansible and data collection

We fully automated the entire experiment lifecycle with Ansible playbooks. This allows us to perform the experiments in a consistent and reproducible manner. As shown in Fig. 4, the experiment lifecycle consists of three stages: setup, experiment run, and cleanup. The setup stage is where the testbed is prepared for a given experiment run (i.e., cloud initialization or application deployment) according to specific experiment settings. The setup stage begins with deploying the software packages a given Kubernetes distribution relies on. Then, communications effects were enabled in EMANE according to a given network scenario. The setup stage includes an additional task while performing application deployment experiments, i.e., cloud initialization. The rationale behind the setup stage is to provide CNs with everything they need to run a given orchestration-related task. This allows us to quantify the overhead due to such a task in the following stage. Note that the setup stage would be performed beforehand in a real-world mission context, e.g., at headquarters.

After the setup stage, the experiment run can be executed (see blue dashed lines in Fig. 4). First, TCPDUMP was started on each CN to capture the network traffic of an experiment run. For cloud initialization experiments, runs focused on the overhead introduced by a given Kubernetes distribution to initialize a CN as the master (this takes only time) and, after that, by workers to join the master (this takes both time and network resources). In application deployment experiments, runs focused on deploying a given number of application replicas per worker. It is worth remarking that the setup stage of such experiments already provides a fully initialized cloud and pre-pulls the container image of the application to be deployed on the workers.

Lastly, there is the cleanup stage. The cleanup stage performs the rollback of the steps that occurred during the previous stages. By doing so, consecutive experiments with different settings do not interfere with each other. The cleanup stage also includes uploading the files with captured network traffic generated during an experiment run on a central server for later analysis.

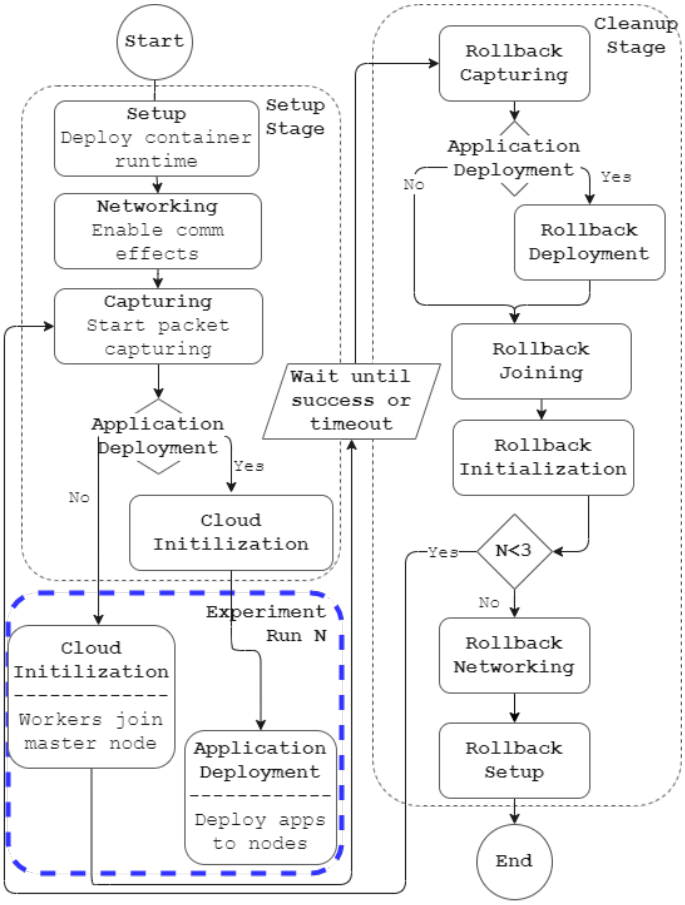

Fig. 4. The lifecycle of an experiment.

B. Analysis framework and graph generation using Docker

The analysis framework was run inside a Docker container. Similar to the benefits of using Ansible for running the experiments, the use of Docker for analysis simplified the configuration, automation, and reproducibility of data analysis. Data preprocessing, statistical analysis, and graph generation were implemented using R (a language and environment for statistical computing and graphics), Seaborn (a Python data visualization library), and Statsmodels (a Python module that provides different statistical models, support for statistical tests, and statistical data exploration). The path to experiment data was passed to the Docker container when it was started. This enabled analysis to be rerun as required on new data sets with no change to the internal settings of the Docker container.

Data analysis was performed in three steps:

1) *Divide a raw data set into succeeded and failed experiments:* A cloud initialization experiment was marked as 'succeeded' if all nodes joined the cloud within a 5-minute timeout. An application deployment experiment was marked as 'succeeded' if all application replicas were successfully deployed within a 5-minute timeout. All other cloud initialization and application deployment runs not meeting these timeouts were considered failed. Note that multiple runs were done per experiment setting because there are cases of partial failure, where some runs succeeded and some failed (i.e., edge cases).
2) *Split traffic based on bootstrapping and steady state phases:* As illustrated in Fig. 2, both the cloud initialization and application deployment experiments have two phases: bootstrapping and steady state. A distinct packet was sent using a special ping tool, called Nping, at the start and end of the bootstrapping to distinguish these phases in the network logs. Nping is a command-line tool capable of sending IP ping packets with user-specific content.
3) *Statistical analysis and creation of graphs:* The captured network traffic was automatically plotted in 3d graphs using Python to better visualize the results per experiment setting. In addition, a multinomial logistic regression test is run on the data to verify the significance of the effect of the settings under investigation on experiment success.

IV. Experimental findings and recommendations

This section lays out the performance results from our evaluations of various Kubernetes distributions and provides recommendations for deploying such distributions in TNs. The discussion about the performance results revolves around the research questions posed in Section II. In this regard, Table II details the maximum number of workers and replicas per worker that a given Kubernetes distribution can manage while performing under the defined network scenarios (see Table I). Note that Table II does not include the WORST scenario because K8s, K3s, and KubeEdge failed consistently under such a scenario (both cloud initialization and application deployment). The successes vs. failures analysis helps reveal patterns about "who works where" (hence, the minimum requirements a given distribution needs to run in TNs) but not about "how it works." In this regard, we explored bootstrapping and steady state phases of succeeded experiments for cloud initialization (Section IV-A) and application deployment (Section IV-B).

A. Cloud Initialization

With no communications effects in place, K8s completed the bootstrapping phase faster than K3s and KubeEdge. However, as the networking conditions worsen, K8s and K3s suffered more than KubeEdge. For example, KubeEdge took a constant time from the LAN (196 seconds (s)) to the BEST (196 s) to the AVG scenario (190 s) while managing 5 workers. Instead, K8s and K3s needed increasingly more time. K8s was the fastest (117 s) under the LAN scenario, where K3s was still faster (155 s) than KubeEdge. We observed the same trend

TABLE II
Experiment Boundaries

Distribution	BEST	AVG
K8s	$\leq$ 11 workers 20 replicas per worker	$\leq$ 5 workers 20 replicas per worker
K3s	$\leq$ 11 workers 20 replicas per worker	$\leq$ 5 workers 20 replicas per worker
KubeEdge	$\leq$ 17 workers 20 replicas per worker	$\leq$ 11 workers 20 replicas per worker

under the BEST scenario, where K8s and K3s needed 143 s and 181 s, respectively. However, K8s performed similarly to KubeEdge in the AVG scenario (174 s), whereas K3s became the slowest (220 s).

Based on our results, KubeEdge is the best option in terms of bandwidth consumption. Not only does KubeEdge require less bandwidth to run, it also does not increase the bandwidth consumption as much as K8s and K3s do when networking conditions worsen and cloud size grows. For example, while initializing 5 workers under the LAN scenario, the average bandwidth consumption during the bootstrapping phase for K8s, K3s, and KubeEdge was 59 Kbps, 63 Kbps, and 8 Kbps, respectively (then, the average bandwidth consumption for the steady state phase was 36 Kbps, 46 Kbps, and 15 Kbps, respectively). By increasing the cloud size to 12 CNs, the average bandwidth consumption raised by 2.2 times for K8s, 2.3 times for K3s, and 2.4 times for KubeEdge. Under the AVG scenario, K8s and K3s consumed 4.4 and 6 times more bandwidth, respectively, whereas KubeEdge took the same average bandwidth as under the LAN scenario. The average bandwidth consumption for the steady state phase followed a similar pattern.

B. Application Deployment

K8s was the fastest (but only slightly faster than KubeEdge) in completing the bootstrapping phase with no communications effects in place. As the networking conditions worsen, KubeEdge performed best, with K8s faster than K3s. For example, K8s, K3s, and KubeEdge took 46 s, 71 s, and 46 s, respectively, to deploy 10 replicas per worker under the LAN scenario (5 workers overall). Under the BEST scenario, KubeEdge (45 s) became faster than K8s (65 s) and K3s (104 s). Then, the AVG scenario depicts an even clearer-cut disparity, where K8s and K3s took twice (129 s and 136 s, respectively) than KubeEdge (55 s).

Although cloud initialization and application deployment may occur at headquarters (where network resources are abundant), K8s, K3s, and KubeEdge still consume resources to accomplish orchestration-related tasks when deployed at the tactical edge (where network resources are limited). Therefore, an in-depth analysis of the steady state phase is critical. It is worth remarking that application deployment experiments assumed a Kubernetes-based cloud already initialized. As a result, the average bandwidth consumption characterizing application deployment's steady state phase included both workers (due to cloud initialization) and replicas (due to application deployment) monitoring. Under the BEST scenario (5 workers overall), K8s, K3s, and KubeEdge consumed 40 Kbps, 81 Kbps, and 25 Kbps, respectively, after deploying 10 replicas per worker. By increasing the cloud size to 12 CNs, the average bandwidth consumption of K3s became dramatically high (309 Kbps), whereas K8s and KubeEdge took 102 Kbps and 74 Kbps, respectively. We observed similar results while deploying 20 replicas per worker. Note that the available bandwidth per CN is 333 Kbps under the BEST scenario with 12 CNs, meaning that the master of a K3s-based cloud would almost use its whole bandwidth to accomplish orchestration-related tasks.

C. Recommendations

Achieved performance results showed that deploying a container orchestration solution under disadvantaged network conditions is feasible under specific circumstances. Such circumstances depend on a combination of factors, i.e., cloud size (how many participating nodes), cloud workload (how many running applications), network parameters (bandwidth, packet loss, and latency), and the container orchestration solution itself. In this regard, Table II provides insights on how to size a cloud and its workload based on a given network scenario and a Kubernetes distribution. A key takeaway is that a single giant cloud is not feasible, backing up the concept of a federation of clouds.

A further outcome is that narrowband networks do not fit the requirements demanded by the Kubernetes distributions we experimentally evaluated. In fact, such networks consist of long-range communications links primarily intended for voice and with limited data support, providing bandwidth an order of magnitude less than the worst scenario we defined.

Concerning the Kubernetes distributions, KubeEdge proved to be the most promising technology. It is worth mentioning that KubeEdge is not a solution per se but needs a K8s-based cloud (even a single master node) as a prerequisite. We believe the synergy between K8s and KubeEdge could lay a solid foundation for enabling the intra-cloud capabilities of the individual clouds jointly constituting an adaptive and federated cloud architecture. Specifically, K8s may build the kernel of the cloud. The primary objective of the kernel is to build an environment where services and underlying resources are available over time. By doing so, the kernel would benefit from typical features provided by traditional cloud environments, such as scaling out (in) of services, transparent migration of services across different hosts, and traffic load balancing, among others. In this regard, we recommend clustering co-located nodes sharing adequate connectivity, such as resources within a vehicle, an aircraft, or a ship. Then, dynamic tactical elements should join the cloud as edge nodes. The kernel should tolerate (even long) disconnections from edge nodes (e.g., unmanned aerial vehicles) and synchronize the cloud state when (and if) possible. This is where KubeEdge comes in with its cloud-edge architecture and UDP-based communications (i.e., QUIC), promoting lightweight, efficient, and autonomous edge nodes. Such edge nodes might physically bring cloud services close to mobile dismounted units that cannot directly connect to a partner cloud. This approach may enable modern cloud capabilities on the battlefield.

V. Conclusion

This experience article described our experimental approach in quantifying the performance of various Kubernetes distributions in an emulated TN. Based on performance results we achieved, we provide recommendations for running a Kubernetes-based cloud while living within the constraints of the tactical domain. This supports the ambition of the NATO IST-168 RTG in developing a federated cloud infrastructure for adaptive command and control.

ACKNOWLEDGMENT

The work presented in this paper is being done as part of the NATO IST-168 RTG on *Adaptive information processing and distribution to support Command and Control*. We would like to thank the NATO IST-Panel for providing us the opportunity to do this highly relevant and interesting research and the individual participating partners for providing valuable inputs within a stimulating and cooperative setting.

REFERENCES

[1] F. Ponce, G. Márquez, and H. Astudillo, "Migrating from monolithic architecture to microservices: A rapid review," in *2019 38th International Conference of the Chilean Computer Science Society (SCCC)*, 2019, pp. 1–7.

[2] K. Zaerens, "Enabling the benefits of cloud computing in a military context," in *2011 IEEE Asia-Pacific Services Computing Conference*, 2011, pp. 166–173.

[3] W. Smith, G. Kuperman, M. Chan, E. Morgan, H. Nguyen, N. Schear, B. Vu, A. Weinert, M. Weyant, and D. Whisman, "Cloud computing in tactical environments," in *MILCOM 2017 - 2017 IEEE Military Communications Conference (MILCOM)*, 2017, pp. 882–887.

[4] H. Bastiaansen, J. v. d. Geest, C. v. d. Broek, T. Kudla, A. Isenor, S. Webb, N. Suri, M. Fogli, B. Canessa, A. Masini, R. Goniacz, and J. Sliwa, "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.

[5] "Nato selects thales to supply its first defence cloud for the armed forces," https://www.businesswire.com/news/home/20210125005006/en/, Jan 2021, [Online; accessed 2022-02-02].

[6] "Military multi-domain operations cloud," https://ec.europa.eu/info/funding-tenders/opportunities/portal/screen/opportunities/topic-details/edf-2021-digit-d-mdoc, Sep 2021, [Online; accessed 2022-02-02].

[7] K. Scott, T. Refaei, N. Trivedi, J. Trinh, and J. P. Macker, "Robust communications for disconnected, intermittent, low-bandwidth (dil) environments," in *2011 - MILCOM 2011 Military Communications Conference*, 2011, pp. 1009–1014.

[8] N. Jansen, D. Krämer, and M. Spielmann, "Testbeds for it systems in tactical environments," in *2014 IEEE Military Communications Conference*, 2014, pp. 1293–1298.

[9] "Emane, "extendable mobile ad-hoc network emulator"," https://github.com/adjacentlink/emane/wiki, [Online; accessed 2022-02-02].

[10] M. Fogli, T. Kudla, B. Musters, G. Pingen, C. Van den Broek, H. Bastiaansen, N. Suri, and S. Webb, "Performance evaluation of kubernetes distributions (k8s, k3s, kubeedge) in an adaptive and federated cloud infrastructure for disadvantaged tactical networks," in *2021 International Conference on Military Communication and Information Systems (ICMCIS)*, 2021, pp. 1–7.

[11] M. Fogli, G. Pingen, T. Kudla, S. Webb, N. Suri, and H. Bastiaansen, "Towards a cots-enabled federated cloud architecture for adaptive c2 in coalition tactical operations: A performance analysis of kubernetes," in *MILCOM 2021 - 2021 IEEE Military Communications Conference (MILCOM)*, 2021, pp. 225–230.

[12] D. Netis and J. Rupar, "List of leading kubernetes distributions," https://qllc.com/Q%20Sights%20-%20Leading%20Kubernetes%20Distributions, May 2020, [Online; accessed 2022-02-02].

[13] "Kubernetes (k8s): Production-grade container scheduling and management," https://github.com/kubernetes/kubernetes, [Online; accessed 2022-02-02].

[14] "Lightweight kubernetes (k3s)," https://github.com/k3s-io/k3s, [Online; accessed 2022-02-02].

[15] Kubernetes native edge computing framework (kubeedge). https://github.com/kubeedge/kubeedge. [Online; accessed 2022-02-02].

BIOGRAPHIES

Thomas Kudla (thomas.kudla@fkie.fraunhofer.de) is researcher and consultant at Fraunhofer FKIE. His background is computer science with focus on enterprise architectures and cloud computing.

Mattia Fogli (mattia.fogli@unife.it) is currently pursuing a Ph.D. degree with the Department of Engineering, University of Ferrara, Italy. His primary research activities focus on tactical networks and software-defined networking.

Sean Webb (sean.webb3@forces.gc.ca) is a computer scientist in the Maritime Information Support Group at the Atlantic Research Centre of DRDC Canada.

Geert Pingen (geert.pingen@tno.nl) is a researcher at TNO at the Monitoring and Control Services research group. His background is in artificial intelligence with a focus on machine learning and distributed systems.

Niranjan Suri (niranjan.suri.civ@army.mil) is the Associate for Research in the Information Sciences Division at the US Army Research Laboratory and a Senior Research Scientist at the Florida Institute for Human and Machine Cognition (IHMC).

Harrie Bastiaansen (harrie.bastiaansen@tno.nl) is business consultant at TNO. His background is in telecommunications and large scale ICT architectures. He is project lead of NATO IST-168.