



# Enabling Emerging Edge Applications Through a 5G Control Plane Intervention

Mukhtiar Ahmad[†], Muhammad Ali Nawazish[†], Muhammad Taimoor Tariq[†], Muhammad Basit Iqbal Awan[†], Muhammad Taqi Raza[‡], Zafar Ayyub Qazi[†]
[†]School of Science and Engineering, LUMS, Pakistan
[‡]Department of Information Systems, University of Arizona, USA

## ABSTRACT

5G networks are considered potential enablers for many emerging edge applications, such as those related to autonomous vehicles, virtual and augmented reality, and online gaming. However, recent works have shown the cellular control plane is a potential bottleneck in enabling such applications — control plane operations are slow, frequent, and can directly impact the delay experienced by end-user applications. Moreover, failures in the cellular control plane can significantly degrade application performance. In this paper, we consider the problem of enabling latency-sensitive and safety-critical edge applications on 5G networks. We identify fundamental control plane design challenges and posit enabling these applications requires re-thinking the cellular control plane. We propose a new edge-based cellular control plane, *CellClone*, which provides fast and fault-tolerant control plane processing. *CellClone* employs multiple active control plane clones at the network edge to mask control plane faults and speedup control processing. Central to its design is a custom quorum-based consistency protocol that provides state consistency with low latency. Testbed evaluations using real cellular traces show a median improvement of more than 3.8× in speeding up control plane operations with outright node failures and stragglers. These improvements translate into better application performance; with *CellClone*, autonomous cars and VR applications reduce missed application deadlines by more than 90%.

## CCS CONCEPTS

• **Networks** → **Control path algorithms**; **Network reliability**; *Wireless access points, base stations and infrastructure.*

## KEYWORDS

Cellular Core, Control Plane, State Consistency, Fault Tolerance

**ACM Reference Format:**
Mukhtiar Ahmad[†], Muhammad Ali Nawazish[†], Muhammad Taimoor Tariq[†], Muhammad Basit Iqbal Awan[†], Muhammad Taqi Raza[‡], Zafar Ayyub Qazi[†]. 2022. Enabling Emerging Edge Applications Through a 5G Control Plane Intervention. In *The 18th International Conference on emerging Networking EXperiments and Technologies (CoNEXT '22), December 6–9, 2022, Roma, Italy.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3555050.3569130


*CoNEXT '22, December 6–9, 2022, Roma, Italy*

ACM ISBN 978-1-4503-9508-3/22/12…$15.00
https://doi.org/10.1145/3555050.3569130

## 1 INTRODUCTION

5G and next-generation cellular networks aim to support emerging latency-sensitive and safety-critical applications, such as safety applications for connected and autonomous vehicles [13, 47, 105], virtual and augmented reality [34, 94], remote surgery [20, 41], and multi-player online gaming [95, 104]. These applications place new demands of high reliability and ultra-low latency on the 5G cellular infrastructure, e.g., VR applications with head tracking systems require an end-end latency of less than 17ms to achieve perceptual stability [94] and edge-assisted safety applications for connected and autonomous cars have strict network delay budgets of about 20 ms or less [105].

To reduce end-end delays and enable such applications, 5G networks are building support for Edge Computing [39, 94]; a paradigm whereby applications are hosted closer to the users, on cell towers, cell aggregation sites, or other edge sites [37–39]. Many cellular providers are working with major cloud providers for deploying edge applications inside the cellular infrastructure [59, 72, 96].

However, there remain fundamental challenges in enabling such applications on cellular networks. A key architectural challenge is the design of the cellular control plane in the core; unlike the Internet control plane, it maintains a dynamic state for each user device to provide mobility support and session management. As a result, whenever a user moves, a user device enters into power-saving mode, or a device creates a new session, its control plane state needs to be updated, which in turn triggers data plane updates. The time taken by control plane operations can add delays on top of the delays provided by the underlying data plane of the radio access network (RAN) and cellular core network.

Prior work [15, 65] has shown that cellular control plane operations are slow, frequent, and cause delays in data services. A 19-month study [65] conducted across four major US carriers showed that control functions contribute 72–999 ms delays in session establishment. Similarly, mobility handovers can cause 24-178 ms delay in retaining data services. These delays can translate into poor performance for latency-sensitive applications, e.g., causing autonomous vehicles to miss 90% of application deadlines due to frequent handovers, which require migration of session state [15]. Importantly, control plane operations are frequent, e.g., *handover* can happen on average every 70 s in a walking scenario for a user [65]. Furthermore, the frequency of control messages is increasing with 5G deployments [81].

To reduce control plane delays for latency-sensitive applications, cellular networks are considering edge-based deployments of cellular core functions [29, 35]. In this paper, we posit that even with an edge deployment, providing fault-tolerant and fast cellular control processing remains an important challenge. Failures in the control

plane can hurt applications' performance [15, 23, 65, 84], causing up to 11 s delay in data access. These failures can cause state inconsistencies between the user device and the control plane in the core, contributing to increasing delays [15]. As cellular providers are rapidly shifting towards network function virtualization for their core network [12, 21, 74, 100], failures are expected to be commonplace, akin to the traditional data center and cloud deployments [31, 54, 55, 66]. Consequently, providing low control plane delays despite control plane faults will be vital for enabling emerging edge applications.

In this paper, we first identify the key design challenges in providing fast and consistent control processing, despite control plane faults. Before proceeding, we note that the 5G standard [10] leaves the choice of specific control plane replication and fault tolerance strategy to operators.

- **C1: High Delays with Synchronous Replication:** A natural design option to provide fault tolerance and satisfy state consistency requirements is to use synchronous replication. However, synchronous replication is known to add significant delays as the state updates need to be stored in at least a majority of replicas before it is safe to respond back to the client [60]. In the cellular context, one such example is *ECHO* [84], a recent fault-tolerant cellular core. Our testbed experiments show that *ECHO* adds significant delays in completing cellular control procedures, which in turn severely degrades the performance of latency-sensitive applications (§3.1). A key challenge is to reduce the delay cost of providing consistency in the cellular context.

In contrast, to reduce control plane delays, several state-of-the-art control plane proposals [15, 23, 80] use asynchronous replication. Among these proposals, *Neutrino* [15] provides state consistency by combining asynchronous replication with a replay-based mechanism. However, we show that asynchronous schemes can introduce inconsistencies and delays when dealing with node failures and stragglers.[1]

- **C2: Inconsistency Due to Non-determinism:** Our extensive analysis of the 5G standard [6] and several open-source 5G/4G implementations reveals that the cellular control plane executes a diverse set of non-deterministic operations (e.g., random ID generation, several timers, authentication vectors) (§3.2). Replay-based asynchronous designs can lead to incorrect failure recovery and disruptions in data services unless they explicitly handle non-determinism. One possible solution could be to ask developers to re-write code to avoid non-determinism. However, our analysis of different control plane implementations reveals this may not be viable as some of the reasons for non-determinism include security. In contrast, providing consistent processing with non-deterministic operations without introducing significant delays is challenging.
- **C3: Failure Detection can be a Potential Bottleneck:** State-of-the-art cellular core designs [15, 23, 58, 76, 80] are primary-based, where only a single stateful node actively processes a user's control traffic. These designs lead to a two-step failure recovery process; first detect the failure and then mitigate it. Our experiments show if the primary node fails, the time to detect failures can become a bottleneck in speeding up the overall failure recovery in an edge deployment (§3.3). A fundamental challenge is to significantly speed up failure detection without compromising on failure accuracy [27, 32] (§3.3).
- **C4: Adverse Impact of Stragglers:** We observe through experiments that if the primary control plane node becomes a straggler, the completion times of cellular control plane procedures are adversely impacted, which in turn degrades the performance of latency-sensitive applications (§3.4). A key reason is that cellular control plane procedures are short – in the order of a few hundred milliseconds [15, 65, 80]. This introduces a key challenge for mitigating stragglers; traditional speculation-based straggler detection techniques [17, 18, 31, 42, 89, 91, 102, 103] would not be effective because of the long waiting time before switching to a non-straggler node [16].

To address the above issues and enable emerging edge applications on 5G, we design and implement a new cellular control plane, *CellClone*. In designing *CellClone*, we synthesize several classical distributed systems ideas while leveraging multiple cellular domain-specific insights. Below, we describe the key ideas in our design.

- **Active Replication (to handle C3 and C4):** Instead of waiting and trying to predict control plane failures and stragglers, *CellClone* takes a *proactive* approach whereby multiple control plane nodes (clones) actively process a user's traffic, store and perform state updates, and generate output messages. This approach can mask the impact of stragglers and failures. However, a concern is the cost of extra processing. Firstly, we note that 5G networks are divided into multiple independent slices serving different classes of traffic [88] and the extra resources for *CellClone* will only be used for the 5G network slices serving latency-sensitive and safety-critical edge applications.[2] Secondly, we show processing cost in *CellClone* can be substantially reduced by using a recently proposed optimized serialization engine for cellular control messages [15]. Overall, we argue this is a reasonable tradeoff to enable new edge applications on 5G.
- **Fast Consistency Protocol (to handle C1):** We design a custom quorum-based consistency protocol that provides state consistency under failures while incurring a low delay overhead under normal (failure-free) scenarios. We leverage cellular-specific optimizations, based on two key observations: (i) cellular clients (users) are stateful and can be used to invoke stateful failure recovery to ensure correct processing, in case none of the replicas are up-to-date [15, 65]. We use this observation to minimize the need for synchronous replication. (ii) Our analysis of state operations in the control plane reveals that a majority of operations are deterministic and we *decouple* the processing of deterministic and non-deterministic operations. This allows us to efficiently deal with deterministic operations and devise separate mechanisms for non-deterministic operations.
- **Individualized Approach to Non-Determinism (to handle C2):** A general strategy to deal with non-determinism is to use one replica for synchronizing other replicas, which suffer from

[1] Stragglers are slow nodes, caused by hardware faults, software faults, or overload and known to be commonplace in datacenter and cloud deployments [31, 54, 55, 66].

[2] In the 5G jargon, this refers to the Ultra-Reliable and Low-Latency communication (URLLC) class.

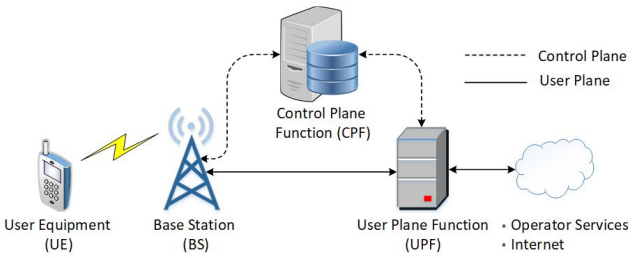


**Figure 1: An abstracted view of the 5G architecture.**

at least one additional RTT. Instead, our analysis of the non-deterministic operations in the cellular control plane reveals that there are opportunities for reducing these delays. We classify the different sources of non-determinism in the cellular control plane into three categories and design an approach to deal with each class individually with low failure-free overhead. Through this individualized approach, *CellClone* avoids two-phase synchronization for a majority of such updates (§4.4).

We implement the main cellular control plane function (CPF) and control traffic aggregator (CTA) in *CellClone*. Our CPF implementation is based on OpenAirInterface [53]. Our code is written in C/C++ programming language and is ≈5000 lines of code. We have made the source code of *CellClone* publicly available [78]. We compare *CellClone* with recent fault-tolerant control plane proposals such as *ECHO* [84] and *Neutrino* [15]. We also evaluate against an *Existing 5G* control plane implementation [53], which requires the user to re-attach to the network on the CPF failure (§6.2). *CellClone* improves control delays under failures by more than 2.8× and 4× as compared to *Neutrino* and *Existing 5G*, respectively. *CellClone*'s performance in failure-free scenarios is comparable to *Neutrino* and more than 58× better as compared to *ECHO*. These improvements can significantly boost the performance of edge applications; with *CellClone*, autonomous vehicles and VR applications can reduce missed application deadlines by more than 90%. *CellClone* achieves this while providing better scalability than *ECHO* and comparable to *Neutrino*. To reduce the resource footprint of *CellClone*, we integrate it with a fast serialization engine for cellular control traffic [15], which reduces the resource footprint by up to 40%.

# 2 BACKGROUND AND MOTIVATION

## 2.1 Cellular Background

**Cellular Network Architecture:** A cellular network consists of (i) the Radio Access Network (RAN), consisting of base stations, and (ii) the Core Network, connecting the RAN with the Internet and operator services. Figure 1 shows a simplified 5G network architecture. Base stations (BS) provide first-hop wireless connectivity to the user device, known as User Equipment (UE). The User Plane Function (UPF) forwards the data traffic between the base stations and the Internet. The Control Plane Function (CPF) provides a variety of management functions that support user mobility, session management, radio resource allocation, and device authentication. By CPF, we refer to the Access and Mobility Functions (AMF) and Session Management Functions (SMF) in 5G. In a standalone 5G architecture, a 5G user is served by a 5G RAN and 5G core network. However, most of today's 5G deployments are based on a non-standalone architecture where 5G users are served by a 5G RAN and 4G core network [93]. In an edge-based deployment, the CPFs may be deployed at cell towers, cell aggregation sites, or other edge sites [15, 70]. In addition, front-end load balancers, also called control traffic aggregators (CTAs), are typically deployed between base stations and cellular core [15, 23, 80].

**Nature of Control Plane Operations:** The cellular control plane maintains a per-user state (size ≈5 KB for each user), which gets updated on short-time scales. When a user creates a new session, the device moves, or the user device enters power-saving mode, the control plane state needs to be updated. These updates are done through request-response messages, also referred to as control procedures [6], representing a set of related updates. For example, on user mobility, a *handover* procedure is triggered that exchanges several messages between the device, base station, and the CPF. These messages lead to several state variables updates in the CPF, e.g., a cell identifier, tracking area, and user tunnel identifiers. The CPF may also invoke external actions, e.g., ask the UPF to create a data session for the user, or fetch information from the subscriber database server (HSS). *Existing 5G* control plane standards [10] describe that the control plane state can be replicated in another peer control plane function or an external data store, USDF (Unstructured Data Storage Function) [10]. However, they leave the design and implementation to the operators.

## 2.2 Emerging Edge Applications

There are many emerging latency-sensitive and safety critical applications that will be deployed at the network edge. *CellClone* is targeted toward enabling such applications over 5G. Below we discuss some of these applications.

**Autonomous Vehicles:** Autonomous vehicles can leverage edge computing [105] and sensing capabilities [47], such as infrastructure-mounted LiDARs and stereo cameras, for safer and more efficient driving. In this class of applications: (i) vehicles augment their onboard perception with other vehicles or infrastructure sensing capabilities, and(/or) (ii) offload computations like planning operations to the edge for more efficient driving. These applications have strict latency budgets on the order of a few to tens of milliseconds and hence must be hosted at the network edge.

**Mobile VR:** Mobile devices running VR applications can offload computation to an edge host for efficiency [38]. There are several example of Edge-based VR services [48, 57, 97]. These applications require ultra-low latency in the order of a few ms [85] and require cellular service continuity as the user moves around [38].

**Multi-Player Online Gaming:** Massive multi-player online games are another example services that can be deployed at the network edge to achieve ultra-low-latency [22, 67, 104]. Such applications can have latency budgets in the order 20 ms.

**Edge Video Analytics:** Real-time video analytics is considered a killer application for Edge Computing [19, 73]. Video analytics when performed at the network edge can provide low latency and avoid the use of expensive network links to stream videos to the cloud [25]. Various edge analytics-based applications, such as traffic dashboards for urban mobility [19] require mobility support from 5G.

### 2.3 Requirements for 5G Control Plane

There are three important properties that a cellular control plane should satisfy for enabling the above edge applications:

(1) *Low delays* in completing control operations under normal (failure-free) scenarios to provide fast data access.
(2) *Fast failure recovery* from control plane faults to minimize disruptions.
(3) *Satisfy cellular state consistency requirements*, specifically provide 'Read your Writes' consistency, to avoid long delays due to state inconsistencies [15, 84].

## 3 CHALLENGES

In this section, we identify the key challenges in designing a fast and fault tolerant cellular control plane to enable edge applications over 5G. We drive this discourse through quantitative and qualitative discussions of existing designs.

### 3.1 Delays with Synchronous Replication

A natural design option to provide fault-tolerance and state consistency is to use synchronous replication. However, synchronous replication can have high delay overheads. To quantify the impact of synchronous replication on the cellular control plane delays and application performance, we implement *ECHO*.[3] *ECHO* uses an external data store, ZooKeeper [107], for storing the control plane state. ZooKeeper *writes* are performed by a leader synchronously through a two-phase commit protocol [46, 107]. Figure 2 shows the increase in *handover* procedure completion time (PCT), in comparison to two other baselines, *Existing 5G* and *Neutrino*. *ECHO* is more than 34× slower as compared to these designs even when the control plane is handling only one user per second. The performance of *ECHO* degrades more than three orders of magnitude as the load on the CPF increases. We observe that this can translate into severe degradation of latency-sensitive applications, e.g., VR applications regularly missing deadlines during a *handover* procedure.

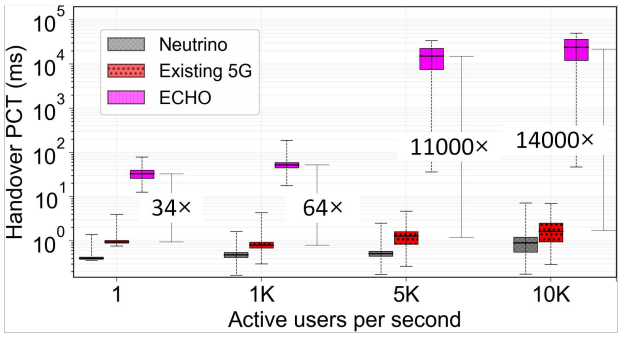


**Figure 2: Comparison of *handover* procedure completion time (PCT).**

**Challenge:** Our analysis reveals that one key challenge is that cellular control plane procedures (e.g., mobility *handover*, *initial attach*) consist of a 1:1 ratio of read and write operations on the control plane state. While data stores like ZooKeeper speed up reads and are effective for read-heavy workloads, speeding up write operations while providing consistency is hard.

### 3.2 Inconsistency Due to Non-determinism

Alternatively, designs that use asynchronous replication can significantly speed up control processing but are susceptible to inconsistencies with non-deterministic operations. Our extensive analysis of 3GPP specifications [5, 10, 11] and several open-source 4G/5G cellular core implementations (refer to Table 1) shows that a subset of operations executed by CPFs are *non-deterministic*. These include randomly generated identifiers like M-TMSI (MME Temporary Mobile Subscriber Identifier), several timers, such as T3512 [6, 28, 36], and generation of Authentication Vectors [7] (detail in §4.4), and others.

Consider the following example scenario (shown in Figure 3) through which we show how non-deterministic operations at CPF can lead to incorrect behavior, and consequently cause long delays in data access, during failure recovery. ① A UE sends an *initial attach* request message to the primary CPF. ② The primary CPF authenticates the device and assigns an M-TMSI, assuming its value is 123. ③ The primary CPF fails after the *initial attach* procedure completion but *before* the state synchronization happens with the backup. ④ As part of the failure recovery procedure, the logged messages (for the successfully executed *initial attach* procedure) are replayed by the backup CPF. Here the backup CPF can take either of the two implementation approaches: involve the device as part of the failure recovery procedure or perform the failure recovery locally. In the first case, the backup CPF tries to sync M-TMSI with the device. It requests the device for identification (through the Identity Request/Response procedure as defined in [8]), and then communicates a new M-TMSI value; whereas in the second approach, the backup CPF generates a new M-TMSI (say M-TMSI = 456) without contacting the device and tries to mask the failure recovery from the device. We find out that seamless failure recovery is not possible for both implementations approaches, and as a result, the device loses the network connection. ⑤a If the backup CPF decides to sync M-TMSI with the device during failure recovery, it needs to page the device with IMSI (known as paging with IMSI procedure [8]). In this case, the device will perform the *re-attach* procedure[4] with the network and face temporary service interruption. ⑤b In case, the backup CPF assigns M-TMSI locally (say M-TMSI=456) then the device cannot be paged by the network in the future – meaning the device will not be able to receive the downlink data packets due to M-TMSI mismatch.

**Challenge:** To address this problem and ensure correctness, we need to explicitly deal with non-deterministic operations. The challenge is to do so while incurring a minimal impact on control plane delays.

[3] For implementation and evaluation details, see §6.

[4] Refer to Section 5.6.2.2.2 in [8] that defines paging with IMSI as an abnormal procedure used for error recovery in the network.
[5] Authentication vector contains a random number, generated as a part of the initial *initial attach* procedure [7].
[6] IP address is allocated to the UE by the UPF on request from the CPF. DHCP based IP address allocation is a non-deterministic operation.
[7] The CPF install T3512 on the UE to periodically reports its location and track its reachability by a mobile reachable timer.

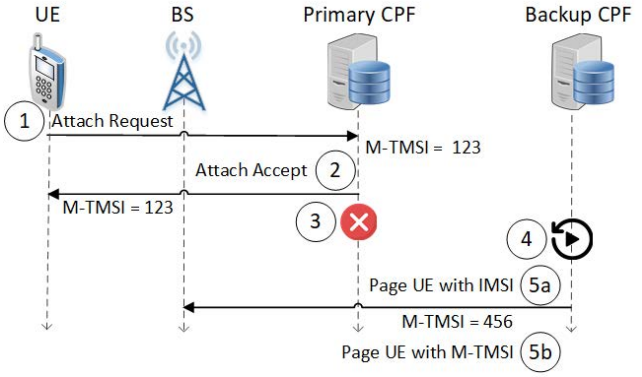


**Figure 3: An example of state inconsistency in message replay-based schemes [15] due to non-determinism.**

**Table 1: Sources of non-determinism in 4G/5G implementations (Impl) and 3GPP specifications [8].**

| Source Code | 4G/5G Impl | | 3GPP Specifications | | |
|---|---|---|---|---|---|
| | M-TMSI | S10 TEID | Auth Vec[5] | UE IP Add[6] | Timer T3512 [7] |
| OAI [53] | ✓ | ✓ | ✓ | ✓ | ✓ |
| Nucleus [51] | ✓ | NA | ✓ | ✓ | ✓ |
| Magma [40] | ✓ | NA | ✓ | ✓ | ✓ |
| CoLTE [92] | ✓ | NA | ✓ | ✓ | ✓ |
| srcEPC [98] | ✓ | NA | ✓ | ✓ | ✓ |
| Open5GS [82] | ✓ | NA | ✓ | ✓ | ✓ |

## 3.3 Slow Failure Detection

In existing 4G/5G deployments, CPF failure detection is performed through heartbeat messages by either the BS or another network function like UPF, with timeouts in the order of tens of seconds[30], with a default value of 30 seconds [50]. We conducted experiments to understand the impact of the *failure detection timeout* (and the corresponding heartbeat intervals) on latency-sensitive edge applications.

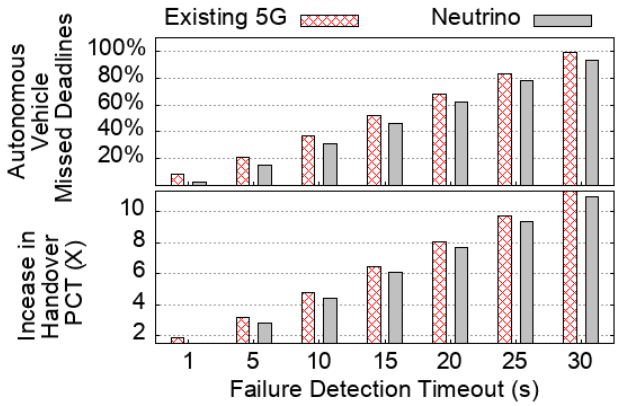


**Figure 4: Impact of CPF failure detection timeout on an autonomous vehicle application.**

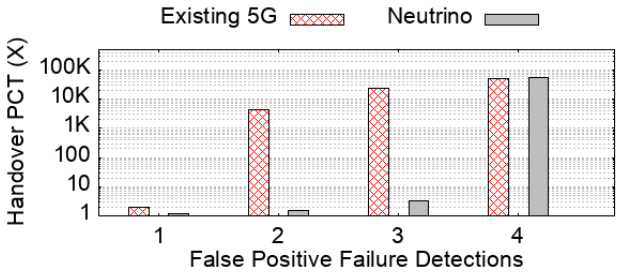


**Figure 5: Increase in handover PCT with false positive failure detect ions.**

Our experiments show that during CPF failures, the time to detect failures has a significant impact on the time to complete control procedures, which can translate into severe performance degradation for latency-sensitive edge applications like autonomous vehicles and AR/VR applications. Figure 4 (top) shows that for an autonomous vehicle,[8] more than 90% of the application deadlines are missed when CPF fails while the user is performing a handover with a failure detection timeout of 30 s. As we decrease the failure detection timeout, the percentage of missed deadlines decreases considerably; with a failure detection time of 1s, the percentage of missed deadlines drops to around 8% with *Existing 5G* and less than 3% with *Neutrino*. The drop in application performance is explained by the corresponding increase in *handover* procedure completion time (PCT) during a CPF failure, as shown in figure 4 (bottom).

**Challenge:** A challenge with simply using more aggressive failure detection timeouts is that it increases the likelihood of falsely flagging a CPF as failed, which may in turn increase the load on other CPFs [45, 69]. In the context of emerging latency-sensitive applications, the delay budgets are so small that it is hard to find a good tradeoff. Figure 5 shows that even in the case of one false failure detection of CPF (not assuming idle backups), there is a 2× increase in the time to complete the *handover* control procedure.

## 3.4 Adverse Impact of Stragglers

We observe that if the primary CPF is a straggler, the completion times of control plane jobs are adversely impacted, which in turn degrades the performance of latency-sensitive applications. Figure 6 shows the impact of a straggler CPF on the missed deadlines for a Virtual Reality application. We observe that during the period a CPF becomes a straggler there are three orders of magnitude increase in *handover* PCT, which in turn translates into the application missing deadlines for a window of 6-7 seconds. A key reason for this adverse impact is that control plane jobs are very short. On the other hand, *ECHO* is not impacted by a single straggler but its failure-free *handover* PCT is more than two orders of magnitude higher than the primary-based designs and as a result, the application performance degrades for a window of 34 seconds.

**Challenge:** As the procedure completion times are in the order of a few hundred milliseconds [15, 65, 80] which represents really short jobs, the existing speculation-based straggler detection techniques would not be effective for short jobs due to the wait time required before switching to a non-straggler node [16].

[8]For information on the experimental setup, see §6.4.

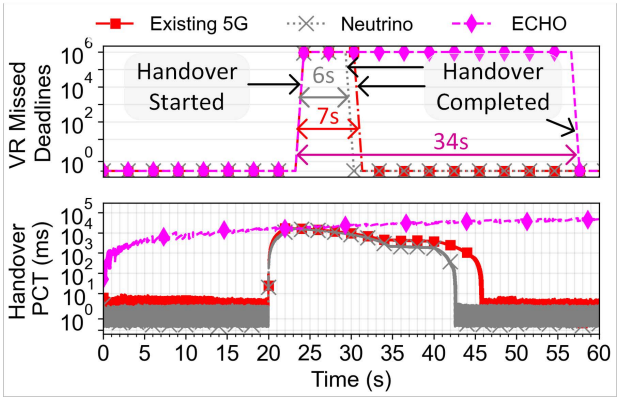


**Figure 6: The impact of straggler CPF on a VR application during *handover*.**

## 4 DESIGN

To address the challenges discussed in §3, we design a new cellular control plane, *CellClone*. In this section, we first provide an overview of our design, explaining the motivation and intuition for the key ideas. We then describe *CellClone*'s replication technique, consistency protocol, and fault-handling process. We also discuss the correctness of our design under failures.

### 4.1 Design Overview

Below we provide an overview of the key ideas in *CellClone*.

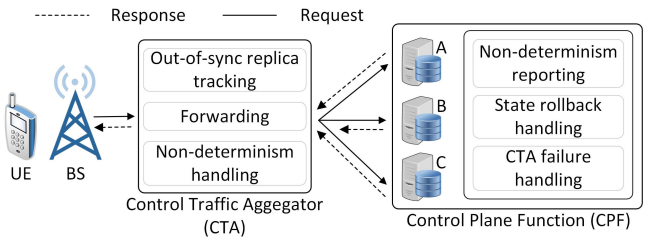


**Figure 7: *CellClone*'s system architecture.**

**Active Replication (to handle C3 and C4):** *CellClone* leverages a form of active replication whereby multiple CPF replicas process a user's traffic, store and perform state updates, and generate output messages. These CPFs independently process a message and generate a response. This form of replication moves faults off the critical path. In contrast to *ECHO*'s [84] use of a non-optimized external data store, *CellClone* uses a custom local state store.

**Custom Quorum for Fast Response Time (to handle C1):** *CellClone* implements a fast quorum-based protocol to reduce control plane delays. In contrast, traditional quorum-based protocols [63, 86] can be slow as they have to wait for at least a majority of nodes to respond before they can send back the response to the UE to ensure consistency. In other words, the read ($R$) and write ($W$) quorums should always intersect ($R + W > N$ where $N$ is the quorum size) to ensure consistency and $W > N/2$ (for majority quorum) to provide fault tolerance. Our approach is to speed up the processing by avoiding the $R + W > N$ & $W > N/2$ constraints. To ensure state consistency, we avoid serving the user with out-of-sync CPF [61] by keeping additional per-user state at the CTA (§4.3) during a procedure execution. To enable this, *CellClone* uses three key ideas: (i) it implements out-of-sync replica tracking at the CTA and avoids using responses from out-of-sync CPFs, (ii) in the event when all CPF replicas become outdated, it invokes state recovery from the UE to ensure correctness, and (iii) decouples the processing of deterministic and non-deterministic operations to avoid two-phase synchronization for deterministic operations, as shown in table 2.

**Individualized Approach to Handle Non-Determinism (to handle C2):** As multiple CPFs independently process a user's control messages and perform local state updates, the subsequent output messages from different CPFs may be different because of a non-deterministic operation. If not handled correctly, this can lead to users experiencing disruptions in data services. A strawman approach could be to roll back the state for the UE on the out-of-sync CPFs from an up-to-date CPF on every non-deterministic operation. This approach will work correctly but at the cost of one RTT delay on every non-deterministic operation. In contrast, we devise an individualized approach to provide negligible delay for most of the non-deterministic operations as depicted in table 2. We first classify the different sources of non-determinism in the cellular control plane into three broad categories; (a) *UE triggered local non-determinism*, (b) *non-determinism due to external actions* from the CPF, and (c) *non-determinism due to various timers* (§4.4). We then identify how each type can be handled with low overhead. We describe in §4.4 our individualized approach to dealing with different classes of non-determinism.

**Resultant Architecture**: *CellClone* re-architects the key cellular control plane entity; referred to as Mobility Management Entity in 4G/LTE packet core [3] and Access and Mobility Management Function (AMF) and Session Management Function (SMF) in 5G [10]. Figure 7 shows *CellClone*'s overall system architecture. BSs directly connect to the CTA. The CTA forwards every control message to members of the quorum. All communication to and from a CPF happens through a CTA while the rest of the network functions can directly communicate with each other. All the CPFs process the request and send a response to the CTA. The CTA filters out duplicate responses and relays the fastest response (assuming $W = 1$) to the BS.

### 4.2 Active replication in *CellClone*

**Quorum Selection:** In *CellClone*, control plane messages are sent to a group of replicas that form a quorum. A quorum in *CellClone* consists of total $N$ nodes. The write quorum size, $W$, can be set to 1 or more replicas ($W \leq N$). $W$ determines the durability of the state. For instance, $W = 2$ implies the CTA will send a response to the BS only when replies from at least two CPFs are received. For $W = 1$, the fastest response from a CPF is relayed to the BS. In this case, there is a good chance that an up-to-date user state exists on more than one CPF and a CPF failure can be completely masked from the user, however, interruption-free failure recovery cannot be guaranteed. Setting a higher value of $W$ provides durability at the cost of extra delays (§4.5). However, irrespective of the value of W, *CellClone* can provide 'Read your Writes' consistency. We allow $N$ and $W$ to be configurable by the cellular network operator.

| Mechanism | Scope | Benefit | Overhead |
|---|---|---|---|
| *Active replication* | All operations | Masks faults | Negligible |
| *Duplicate filtration* | All operations | Correctness | Negligible |
| *Out-of-sync CPF tracking* | All operations | Correctness | Negligible |
| *State rollback recovery* | Small % of non-deterministic operations | Correctness | One extra RTT |
| *State recovery from the UE* | Rare: Failure of CTA and $\geq W$ CPFs failures | Correctness | Reattach cost |

**Table 2: A summary of key mechanisms in *CellClone*.**

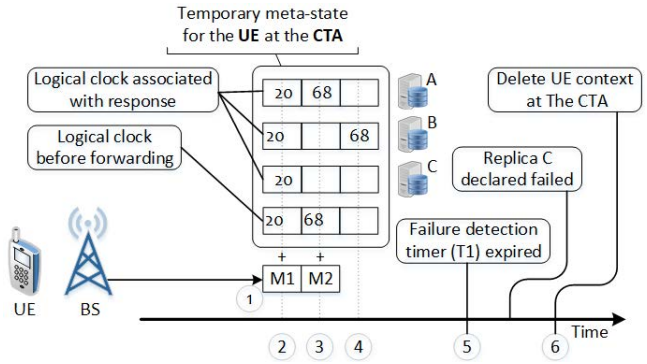


**Figure 8: Temporary meta-state for the user at the CTA during control plane operations execution.**

**Duplicate Filtration at the CTA:** The CTA forwards every control message, along with a logical clock timestamp [62], to all the quorum members. Every quorum member performs state updates, generates a response message, and sends it to the CTA with the same logical clock timestamp. The CTA uses a pair of user unique ID (IMSI or M-TMSI) [9] and logical clock timestamp to filter out duplicate replies (i.e., replies with the same logical timestamp). The CTA routes the fastest response to the BS/UE and drops the rest of the replies.

## 4.3 Quorum-based consistency protocol

**Out-of-Sync CPF Tracking:** Always using the fastest response(s) from a quorum of CPF replicas can lead to inconsistencies. One key goal in *CellClone* is to avoid the use of responses from out-of-sync CPF replicas. For this purpose, the CTA tracks out-of-sync replicas that lag behind the other CPFs during the state update operations. Figure 8 shows a sequence of operations performed at the CTA during the execution of a control plane procedure. The sample procedure has a total of 2 messages, M1 and M2. The quorum in this sample scenario consists of 3 CPFs and a write quorum of 1 CPF. During a procedure, CTA maintains a per-user temporary meta-state to track the out-of-sync replicas. The temporary meta-state includes, (i) the logical clock associated with the control message before forwarding, and (ii) the logical clock associated with the response from each CPF in the quorum. The CTA relays the fastest response from an up-to-date replica (a replica that acknowledged the reception of all control messages in sequence) to the UE. In figure 8 the horizontal axis represents time. The sequence of steps are as follows:

[9]International Mobile Subscriber Identity (IMSI) is a user's permanent ID and only included in first message to the CPF. Cellular networks avoid frequently exposing IMSI due to security reasons, therefore, M-Temporary Mobile Subscriber Identity (M-TMSI) is used for subsequent messages.

(1) On reception of the first message (M1) from the UE, CTA creates a temporary meta-state for the UE. Message M1 is associated with a logical clock (20) and the CTA forwards M1 (with its logical clock) to the quorum. The CTA also caches the logical clock value in its volatile memory.
(2) Every member of the quorum replies to the CTA with the logical clock of M1 (20). The CTA forwards the fastest response (from replica **B**) to the UE. Figure 8 shows that the reply (with logical clock 20) for M1 is received from all the replicas.
(3) For M2, CTA receives the fastest response from replica **A** (with logical clock 68) and no response is received for it from replicas **B** and **C**. The CTA starts failure detection timer **T1** (configurable timeout) when the procedure completes on the fastest CPF, i.e. **A** in this case.
(4) Before the expiry of timer **T1**, replica **B** responds for M2 (with logical clock 68) to the CTA. Replica **C** remains out-of-sync at this stage.
(5) Failure detection timer **T1** expires. The CTA marks replica **C** as failed. Replica **C** can no longer serve the UE.
(6) The CTA deletes the temporary meta-state for the UE.

Next, we discuss different scenarios in the context of figure 8.

*Normal scenario*: In normal scenarios, all the quorum members behave just like replica **A**. In step **3**, a response is received from all the quorum members and temporary UE context is deleted on the reception of a response from all the members.

*Straggler scenario*: In this scenario, at least one of the quorum members is outdated at step **3** (replica **B** in figure 8). However, the straggler nodes are able to catch up before the expiration of timer **T1**.

*Failure scenario*: In this scenario, at least one of the quorum members (replica **C** in figure 8) has not responded even on the expiry of timer **T1**. In this case, the CTA marks the outdated replica as failed.

## 4.4 Handling Non-Determinism at the CPF

Given some initial same state on the CPFs, a non-deterministic state update can produce different state on different CPFs. We classify non-deterministic operations at CPF into three categories and describe specific mechanisms for handling them with minimal overhead while providing consistency.

**UE Triggered Local Non-Determinism at CPF:** In this case, the non-deterministic operation is executed at the CPF as a result of a request received from the UE. For instance, during the *initial attach* procedure, the CPF assigns M-TMSI (a temporary user ID used) to each user. The M-TMSI is a local user ID, unique only within the CPF, and allocated through a random number generator. This implies that all the quorum members may assign a different M-TMSI to the same UE that violates the UE state consistency requirement.

To ensure correct control processing, a key requirement here is that every CPF must have the same M-TMSI value.

To handle this issue, *CellClone* (i) modifies CPFs to add a non-determinism flag along with the corresponding modified state when they send a message response to the CTA, and (ii) enables CTA to choose the response that came from the fastest CPF. The CTA informs the fastest replica that its response is used to roll back the state on the rest of the replicas before replying to the UE. This information is useful for consistent recovery of the CTA, in the event it fails after a non-deterministic execution on the CPF but before being able to roll back state on the rest of the replicas (§4.5). This type of non-determinism incurs one RTT overhead, however, it is observed only in the *initial attach* procedure.

**Non-Determinism due to External Actions at CPF:** A non-deterministic operation may be executed at an external node as a result of a request received from the UE. For instance, during the *initial attach* procedure, the CPF requests HSS (Home Subscriber Server) for security authentication of the user and UPF for IP address allocation. For security authentication, the HSS provides multiple Authentication Vectors (AVs) to the CPF. Each AV contains a random number (required for user's security) generated at the HSS. Similarly, UPF may assign an IP address to the UE through a DHCP server. In both cases, every member of the quorum may end up getting different AVs and IP addresses.

*CellClone* handles this issue with duplicate message filtration at the CTA.[10] Every quorum member generates a request for AVs/IP toward the HSS/UPF through the CTA. The CTA forwards only the fastest request to the HSS/UPF. On reply to the message, the CTA forwards it to the quorum, and as a result, the state consistency requirement is satisfied.

**Timer-Triggered Local Non-Determinism:** The third case of non-determinism arises because of asynchronous state transition at the CPF due to a local event (e.g., timeout). Each CPF maintains various timers per UE for different purposes, e.g., T3512 [36] for tracking the UE location for network-originated services [9]. A potential issue is that the same timer for the same UE on different CPFs may expire on different time instances. As a result, a UE may have a different state on different members of the quorum. For instance, triggering of the mobile reachability timer can cause a UE to have different reachability status on different CPFs, which violates user state consistency requirement. As a consequence, for the same UE, some of the CPFs may reject the network-originated data requests (due to the user not being available) and some of them may send a paging message to the BS to wake up the UE. A key requirement here is for the UE to have a consistent UE state (available or not available) across every CPF. The fastest response-based duplicate message filtration at the CTA also solves this issue.

## 4.5 Handling CTA Failure

In §4.3, we discussed how *CellClone* is transparent to CPF failures. In this section, we discuss how a CTA failure can be recovered by caching the temporary UE context (required at the CTA) at the CPFs during a procedure execution. In *CellClone*, interruption-free CTA failure recovery is possible only if $W - F >= 1$ is satisfied when CTA failure happens, where $W$ is the write quorum size and $F$ is the count of failed CPFs. The $W - F >= 1$ condition ensures that at least one up-to-date CPF from the write quorum size survived overlapping failure with the CTA. The information available at the up-to-date CPF is enough to recover the temporary meta-state context for the UE. We also illustrate this point through a specific example in Appendix A.

[10] Alternatively, another node (switch) could be used to perform duplicate filtration for request directed at HSS/UPF.

## 4.6 Correctness Under Failures

In this section we describe how *CellClone* performs consistent recovery from various types of failures.

**At Least One Up-to-Date CPF is Alive:** In this case, at least one up-to-date CPF survives failure. As the CTA avoids fallback on an out-of-sync replica, the up-to-date CPF will be used to serve the user ensuring state consistency.

**All Up-to-Date CPFs Fail:** In this case, all the CPFs having up-to-date user state fail. The CTA avoids forwarding the user requests to the out-of-sync replicas and asks the user to re-attach to re-create a fresh consistent user state on all the $N$ CPFs.

**CTA Failure:** If the CTA fails, its lost state can be recovered with the help of the cached state at the CPFs, provided at least one up-to-date CPF is alive. In case there is no up-to-date CPF, the user device is asked to re-create a fresh consistent user state on all the $N$ CPFs.

# 5 IMPLEMENTATION

We have implemented *CellClone* and all supporting components and functions in the C/C++ programming language. A summary of our implementation is given below:

**Traffic Generator:** Our traffic generator emulates both UE and BS; it is based on DPDK (v 17.11 [2]) for fast I/O operations and is similar to the traffic generator used in [90]. Our traffic generator does not emulate wireless delays of the UE and BS. It replays real cellular control traffic traces [83]. In our implementation, the base station communicates with the CTA using S1AP/NGAP—the protocol currently used in 4G/5G networks between the base stations and CPFs.

**Control Traffic Aggregator:** Our CTA module receives control traffic from the BS through a custom DPDK application. A producer thread reads packets from the NIC to ring buffers shared with multiple consumer threads. Consumer threads read packets from the shared ring buffers and transmit those to the quorum. We have a similar multi-threaded design for communication from the CTA to the BS. We use the DPDK hash table to track out-of-sync replicas. Our CTA implementation consists of a total of 2019 lines of code.

**Control Plane Function:** Our CPF implementation supports the following three control procedures: (*i*) *initial attach*, (*ii*) *handover* with CPF change, and (*iii*) *service request*. To coordinate various control procedures, we have implemented state machines at both the CPF and the traffic generator. Our CPF implementation is based on OAI [53] and in addition, handles the non-deterministic operations and also performs CTA failure detection and recovery. We implemented two versions of *CellClone*:

(1) *CellClone-ASN.1*: uses ASN.1 [1] based serialization engine, also used in existing 5G designs.

(2) *CellClone-FBs*: uses the modified FlatBuffers (FBs) [43] based serialization engine of *Neutrino* [15].

Our CPF implementation uses DPDK for fast IO operations. Specifically, we are using DPDK's network interface APIs for communication with network interface cards and huge pages for fast memory operations. We implement an in-memory state store for the users using the STL map container. Our CPF modifications consist of a total of 2886 lines of code.

# 6 EVALUATION

We evaluate the efficacy of *CellClone* in terms of 5G control-plane procedures and applications, handling of failure scenarios, and scalability. The summary of our evaluation results is as follow:

- **Improvement in Procedure Completion Time (PCT) under Failures:** *CellClone-ASN.1* improves PCTs by more than 2.8× and 4× as compared to *Neutrino* and *Existing 5G*, respectively under failure scenarios.
- **PCT Improvement with Straggler CPFs:** *CellClone-ASN.1* improves PCTs by more than 3.8× and 6.2× as compared to *Neutrino* and *Existing 5G*, with straggler CPFs.
- **Performance under Failure-Free Scenarios:** *CellClone-ASN.1* performs more than three orders of magnitude better as compared to *ECHO*. *CellClone-FBs* perform up to 1.2× and 1.8× better as compared to *Neutrino* and *Existing 5G* under normal scenarios.
- **Applications' Performance:** *CellClone* significantly improves the performance of autonomous vehicles and AR/VR applications in both straggler and failure scenarios.
- **Scalability and Sesource Utilization:** *CellClone* is highly scalable with a modest increase in systems resources.

## 6.1 Systems Settings and Methodology

Our test setup consists of 11 co-located servers running Ubuntu 18.04.3 with kernel 4.15.0-74-generic. Each server is a dual-socket with 18 cores per socket, Intel Xeon(R) Gold 5220 CPU @ 2.20 GHz, and with a total memory of 128 GB. All servers are also equipped with Intel X710 40 Gb (4 x 10) NIC. Five servers are used for the control plane, one for the control traffic generator, one for CTA, and three for multiple CPFs running as independent processes. Three servers are used for the user plane, each one hosts Intel's UPF [52], edge application server, and client application (such as an autonomous vehicle or AR/VR), respectively. Three servers host a ZooKeeper [49] cluster for *ECHO* [84] evaluation. Our experiments are run with real control traffic traces collected from ng4T [83]. Our open-source [78] traffic generator emulates two different types of control traffic loads: (i) procedure-specific uniform control traffic load to emulate a fixed number of control procedure requests per second and (i) 10 Gbps bursty traffic to emulate a large number of IoT devices sending requests in a synchronized pattern. Our evaluation results represent both uniform (with 10K procedures per second) and bursty control traffic.

**Failures:** Failure means a CPF is unable to respond to the BS due to a hardware/software fault or network partitioning. In our evaluation, CPF failure is induced through a script by killing the corresponding process. By default, one CPF is killed, otherwise specified. A heartbeat (HB) and a timeout-based mechanism are used to detect the failure.

**Stragglers:** Our straggler implementation is similar to [24]. We consider two types of stragglers:

*T-Straggler*: For temporary straggler (T-Straggler), we randomly delay responses for 10% of the control messages — by delays experienced by messages in the $90^{th}$ percentile.

*P-Straggler*: For permanent straggler (P-Straggler), we delay all the control messages by a factor equal to the median control delay when a CPF is subjected to 50% additional control traffic load [80].

**Experimental Configurations:** By default, the total number of CPFs in the experiments is 6. *CellClone*'s quorum size for all the experiments is $N = 3$, with default write quorum size $W = 1$, however, we also show results for $W = 2$. All the experiments are performed for 60 seconds, a default rate of 10K users per second, unless specified otherwise.

## 6.2 Baselines

***Existing 5G*:** It is a representation of the existing 4G/LTE MME and 5G AMF+SMF [10] deployments, a modified version of the OpenAirInterface [87] codebase. A single CPF actively serves the UE and requires UEs to *Re-Attach* on a CPF failure due to the absence of any protocol for consistent recovery from the backup by the 3GPP [10]. Instead of kernel sockets, *Existing 5G* uses DPDK [2] for fast I/O operations.

**Neutrino:** Our evaluation is based on the open-source codebase [79] of *Neutrino* [14].

***ECHO*:** It is a modified version of the *Existing 5G* and instead of the internal state store, it uses ZooKeeper [49] for state storage. We implement the best performing version "Disk-nFC" of the *ECHO* [84], which avoids synchronous disk writes.

## 6.3 Latency improvement in Procedure Completion Time (PCT) with *CellClone*

**PCTs under Failure:** Figure 9 shows median *service request* PCT under a CPF failure, with different schemes. The figure shows that in an ideal case when failure detection takes a negligible time (configuration used in [15]), *CellClone-FBs* performance is 2.8× better than *Neutrino* and 4× better than the *Existing 5G*. Here *Neutrino* is 3.3× better than *Existing 5G*. The figure also shows that *CellClone* reduces PCTs by more than $10^3$× as compared to all the primary-based designs (*Neutrino* and *Existing 5G*), with the failure detection time of 1 s. This is because the time to detect failures dominates the total failure recovery time (before a user can be served by a new CPF) in all the primary-based designs. The figure shows that failure detection timeout does not impact both *CellClone* and *ECHO*. This is because multiple active CPF clones process every control message in *CellClone*, hence its performance under failure remains similar. *CellClone* performs about 58× better than *ECHO*, because of the improvement it provides over *ECHO* in normal cases. *CellClone* performance under multiple failures remains similar as long as the failed clones count is less than the write quorum size.

**PCTs with Stragglers:** Figure 10 shows *service request* PCTs when a CPF behaves as a permanent straggler (P-Straggler) from 58-73 seconds. The figure shows that *CellClone-FBs* provide the lowest PCTs at all times. *Neutrino*'s performance is closer to *CellClone-FBs* in normal scenarios but more than three orders of magnitude slower when the CPF is a straggler. Both *ECHO* and *CellClone* are

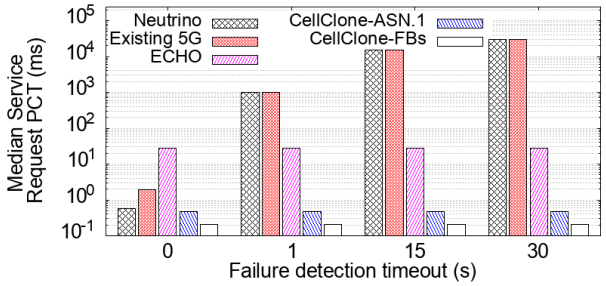


**Figure 9: Service request PCT comparison under failure, with varying CPF failure detection timeouts.**

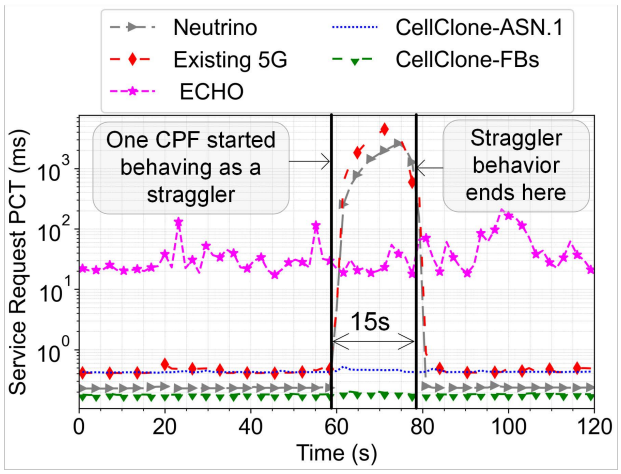


**Figure 10: A comparison of *CellClone*'s *service request* PCT with other schemes in a P-Straggler scenario.**

not impacted by the straggler but *ECHO* is up to 58× slower than *CellClone* in median PCT, which is again due to the failure-free improvement of *CellClone* over *ECHO*.

**PCTs with Non-Deterministic Operations:** Figure 11 shows the comparison of the PCT of *initial attach* with different schemes. In normal scenarios, *CellClone-FBs* provide the lowest median PCT. The median PCT for *CellClone-FBs* and *CellClone-ASN.1* is 0.7× and 0.8× of the median PCT for *Neutrino* and *Existing 5G*, respectively. The figure shows that the fastest response-based *CellClone* design performs better as compared to other schemes even when it incurs one additional RTT (between the CTA and CPF) to keep a backup of the non-deterministic operation (§4.4) at the CPF. In T-Straggler case, *CellClone-ASN.1* performs 4.2× better than *Existing 5G*, for median PCTs. Similarly, in the case of P-Straggler, *CellClone-ASN.1* performs more than three orders of magnitude better than *Existing 5G* and *Neutrino* in terms of median PCT. Performance of both *CellClone* and *ECHO* are not impacted by the straggler but *ECHO* provides more than three orders of magnitude higher completion time as compared to *CellClone*.

**PCTs under Failure-Free Scenarios:** In figure 12, we compare the performance of *CellClone-ASN.1* with baseline *ECHO* & *Existing 5G*, and *CellClone-FBs* with baseline *Neutrino*, in failure-free scenarios, for three different procedures. The figure shows that for the fastest replica-based reply configuration ($W = 1$), both *CellClone-ASN.1*

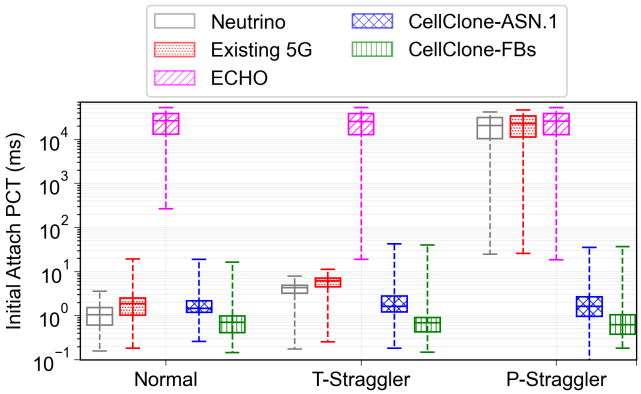


**Figure 11: Initial Attach PCT comparison. Initial Attach involves multiple non-deterministic operations.**

and *CellClone-FBs* perform better than the respective baselines. When *CellClone* waits for at-least two replicas before responding to the user ($W = 2$), *CellClone-FBs* still perform better than *Neutrino* due to message logging and state checkpointing overhead in *Neutrino*. *ECHO* is at least three orders of magnitude slower than *CellClone* due to the use of an external store ZooKeeper not suitable for the cellular workload. Here *CellClone-ASN.1*'s performance is mostly close to the *Existing 5G* and at most 0.84× slower. When *CellClone* waits for all the replicas before responding to the user ($W = 3$), its performance is still three orders of magnitude better than *ECHO*. Here *CellClone-FBs*'s performance is close to *Neutrino* while *CellClone-ASN.1* is slightly slower than *Existing 5G*. Our prototype of *Existing 5G* does not implement any mechanisms for fault-tolerance, therefore it incurs no failure-free overhead. However, in practice when the *Existing 5G* will implement any particular fault-tolerance protocol, the performance of *CellClone* in comparison to *Existing 5G* will further improve.

## 6.4 Impact on Application Performance

To measure the impact of *CellClone* on application performance, we interface Intel's 5G UPF [52] with *CellClone*. A UE connected to *CellClone* can create a new session, delete an existing session, and modify the existing bearer on the UPF through the S11/N4 interface [10, 11]. Our application data traffic passes through the UPF before reaching the edge server.

**Simulation Setup:** To measure the impact of mobility on the application performance, we set up two different client applications on UE: (i) a CARLA emulator [26, 33] for self-driving cars and (ii) a VR emulator [71]. Both of these applications offload processing to an edge server [14]. The self-driving car periodically transmits sensors data to the network edge to take driving decisions [68, 99, 106]. The VR application performs remote rendering on edge [101]. As we aim to see the impact of network delay on applications during *handover*, we do not perform any additional processing of the data at the edge; the edge server simply sends a hard-coded response to the application as soon as input data is received. On the client side, we count the number of responses that miss the application-specific deadline. We perform experiments for 5 minutes while the client is subjected to a *handover* every 25 s on average.

| | *ECHO* / *CellClone-ASN.1* | | | *Existing 5G* / *CellClone-ASN.1* | | | *Neutrino* / *CellClone-FBs* | | |
|---|---|---|---|---|---|---|---|---|---|
| **Procedure** | $W = 1$ | $W = 2$ | $W = 3$ | $W = 1$ | $W = 2$ | $W = 3$ | $W = 1$ | $W = 2$ | $W = 3$ |
| Initial Attach | 15K | 13K | 11K | 1.18 | 1.04 | 0.89 | 1.77 | 1.4 | 1.12 |
| Handover | 9.3K | 8.3K | 7.4K | 1.05 | 0.94 | 0.85 | 1.52 | 1.37 | 1.02 |
| Service Request | 9.5K | 5.9K | 5.3K | 1.35 | 0.84 | 0.76 | 1.43 | 1.28 | 0.93 |

**Figure 12: *CellClone*'s fault-free $99^{th}$ percentile PCT comparison with existing systems for write quorum size $W = 1$–$3$ and $N = 3$. For fairness purposes, we compare *CellClone-ASN.1* with *ECHO* & *Existing 5G* and *CellClone-FBs* with *Neutrino*. Higher than 1 values represent improvement.**

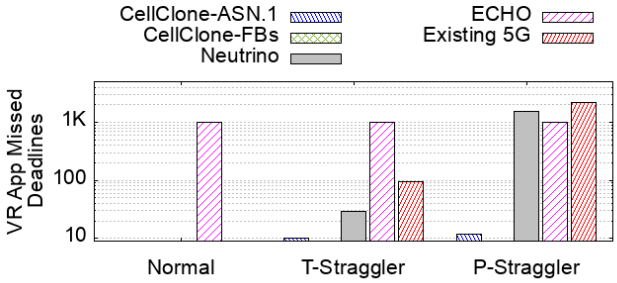


**Figure 13: Impact of a straggler CPF on VR App.**

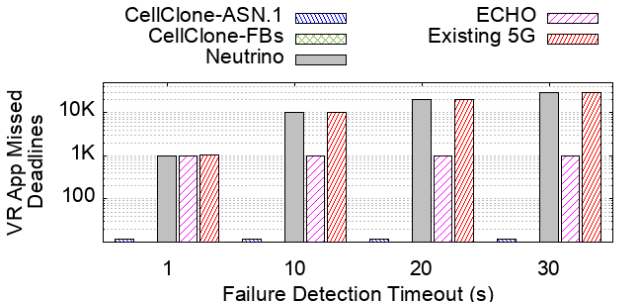


**Figure 14: Impact of failure detection on VR App.**

Virtual reality (VR) applications require a latency of less than 16 ms [94] to achieve perceptual stability. Figure 13 shows the number of application packets that miss the VR application-specific deadline. The figure shows the performance of both *CellClone* and *ECHO* are not impacted by straggler, but *ECHO* misses approximately $10^3$ more deadlines as compared to the *CellClone* due to its failure-free slow response. When users are served by a T-Straggler CPF, all the primary-based systems missed at least 29 deadlines. For P-Straggler, all the primary based designs missed more than 1500 application deadlines compared to *CellClone-ASN.1* that missed 12 deadlines.

Figure 14 shows the count of missed deadlines for a VR application when different failure detection timeouts are used to detect the CPF failure. It shows that even for the lowest failure detection timeout, all the primary-based designs miss at least $10^3$ deadlines, whereas *CellClone-ASN.1* misses 12 deadlines, irrespective of the failure detection value. Both *CellClone* and *ECHO* are not impacted by the failure detection timeout but still, *ECHO* misses approximately 1000 more deadlines as compared to *CellClone* because of its failure-free slow response.

## 6.5 *CellClone*'s Scalability, Resource Requirements, and Processing Delay

**Scalability:** Figure 15 shows the maximum service rate *CellClone* can support to match the $99^{th}$ percentile procedure completion

| **Procedure** | **Neutrino** [15] | **ECHO** [84] | **CellClone ASN.1** | **CellClone FBs** |
|---|---|---|---|---|
| Init. Attach | 40K | **150** | 12K | 71K |
| Handover | 15K | **80** | 6.8K | 42K |
| Service Req. | 150K | **1** | 11.5K | 125K |

**Figure 15: *CellClone*'s service rate comparison with other schemes to match $99^{th}$ percentile PCT of the *Existing 5G* when serving a burst of 10K users. Higher values are better.**

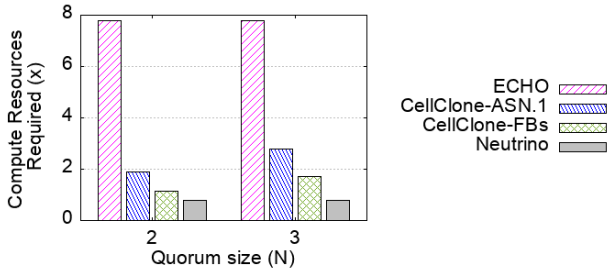


**Figure 16: *CellClone*'s normalized resources usage comparison with other proposals.**

times of *Existing 5G* (with a burst of 10K active users). The figure shows that overall both *CellClone-FBs* and *Neutrino* are the most scalable designs when subjected to bursty control traffic. *CellClone-ASN.1*'s performance is closer to the *Existing 5G*. The figure shows that *ECHO* serves the least burst size because its performance depends on ZooKeeper, which is not suited for cellular workloads.

**Resource Utilization:** Figure 16 shows the compute resources required for different schemes with *Existing 5G* as a baseline. To provide a better performance, *CellClone* (CTA and CPFs combined) consumes some extra resources. For $N = 2$ and $W = 1$, *CellClone-ASN.1* incurs up to 1.9× overhead compared to *Existing 5G*. *CellClone-FBs* require a relatively modest increase of up to 1.2× in compute resources as compared to *Existing 5G*. *Neutrino* requires the least and *ECHO* the most compute resources, respectively among all the schemes. For $N = 3$ and $W = 2$, *CellClone-ASN.1* and *CellClone-FBs* require 2.8× and 1.7× compute resources respectively, as compared to *Existing 5G*.

**Processing Delay Comparison:** Figure 17 compares the message processing time at the CPF for *CellClone* with other schemes. The figure shows that *Neutrino* and *CellClone-FBs* provide the lowest message processing time, with a median value of 3 $\mu$s. The median message processing time for *CellClone-ASN.1* and *Existing 5G* is around 9 $\mu$s, which is 3× higher as compared to the *CellClone-FBs* and *Neutrino*. *ECHO* provides the highest median message processing time of 1100 $\mu$s, more than 120× higher than the rest

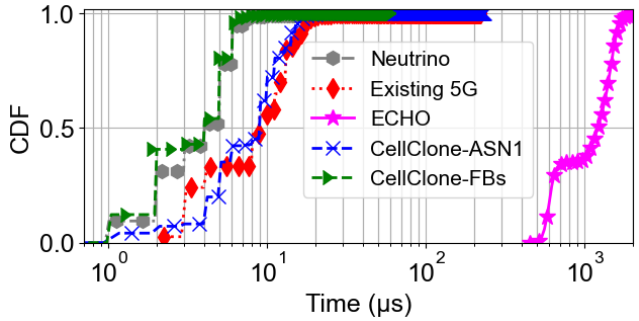


**Figure 17: *CellClone*'s processing delay comparison with other systems.**

of the designs. The variations across message processing times are due to different messages in the *initial attach* procedure.

## 7 RELATED WORK

**Primary-based CPF Designs:** Several recent proposals [15, 23, 58, 65, 76, 80] assume primary-based designs. *SCALE* [23] proposes the use of consistent-hashing for scaling a software 4G/LTE MME to handle increasing signaling load. *MMLite* [80] proposes the use of skewed consistent hashing for distributing control traffic more efficiently. *SkyCore* [76] broadcasts state updates to the neighboring nodes to minimize control plane delays for handover. *DPCM* [65] leverages device states to speed up the control plane operations and handle transient radio failures. *Neutrino* [15] leverages modified FlatBuffers [44] to speed-up control operations and proposes a failure recovery protocol, which is based on logging of control messages and structured geo-aware state replication. All these schemes depend on failure detectors for failure recovery which can delay switching to a new replica. Similarly, none of these designs handle straggler nodes.

**Quorum-based External State Store:** *ECHO* proposes a 4G/LTE control plane design with control processing replicated across multiple stateless processing threads and user state stored in an external data store, ZooKeeper [49]. *ECHO* provides fault tolerance but our evaluation shows the data store it uses is not optimized for 5G cellular workloads.

**Centralized Cellular Control Plane Architectures:** There are several proposals for architecting SDN-style cellular core [56, 64, 77]. A common theme in all these works is to have a logically centralized cellular control plane (i.e., MME) with a programmable data plane. However, unlike *CellClone*, centralized control plane architectures may not be suitable for achieving low control plane delays in edge deployments.

**Consolidated Cellular Core Architectures:** Other works consolidate cellular core data and control plane functions [75, 76, 90]. PEPC [90] slices the cellular core by users. It consolidates device states and refactors core functions. *PEPC* improves the overall cellular core performance. SoftBox[75] also proposes a consolidated cellular core architecture. However, these designs do not provide fault tolerance.

## 8 DISCUSSION

**Failure Handling:** Ensuring low latency data access also requires failure handling of other network functions (such as unified data management function and authentication server function) besides the CPF. In this work, we focus on CPF failure handling as (i) control plane operations performed by the CPF are more frequent [65] as compared to the rest of the network functions (such as the authentication server function) and (ii) it is difficult to handle due to the highly dynamic per-user state it holds. The rest of the network functions can be relatively easily handled.

**Architectural Compliance with 5G:** Any system that provides fault-tolerance introduces a certain level of complexity in design in comparison to the systems providing no fault-tolerance. Recent proposals [15, 80] are examples of the systems providing fault-tolerance. Like the front-end load balancer used in these systems, our design requires a software module (CTA) in front of CPFs only. CTA encapsulates the CPF(s) to ensure state consistency and avoid fallback on out-of-sync replicas when no up-to-date CPF exits. For the external world, CTA acts as a CPF. As CTA in our design is only required in front of the CPFs, we see no fundamental limitations in using our control plane design in service-based communication architecture of 5G.

**Deployability and Performance Trade-Offs:** *CellClone* is a drop-in replacement of the existing CPF. It requires no hardware or software changes on the UE and BS. Software changes on the BS and UE are only required when FlatBuffers based serialization of *Neutrino* [15] is used. As a trade-off between performance and compute resources, *CellClone* favors performance by using up to 1.7× more resources as compared to *Existing 5G*, when the replication factor is 3. However, these extra resources are only required for the 5G network slice serving ultra-reliable and low-latency communication (URLLC) [88] based applications while the rest of the network slices in 5G can be configured to use fewer active clones hence fewer resources.

**Our Approach to Non-Determinism:** The non-determinism examples in Table 1 are not comprehensive. However, all kinds of non-determinism can be handled with our design (§4.4). A different design approach would be to avoid non-deterministic operations but it may (i) require changes on multiple network functions, such as UPF and/or HSS, and (ii) may compromise security, e.g., a random number generated at the HSS is required for security [4].

**Quorum Selection:** Our evaluation is based on all the active CPF clones co-located at the same edge site. However, for fault-tolerance and/or load balancing purposes, a quorum can be created with nodes from different edge sites with an obvious trade-off of delay.

## 9 CONCLUSION

In this work, we demonstrate that redesigning the cellular control plane is an important step toward enabling emerging edge applications. Our contributions are two-fold. First, we identify the key challenges in designing a fast and fault tolerant cellular control plane. Second, we designed a new control plane, *CellClone*, that masks control plane faults while providing consistent control plane processing. Testbed experiments show *CellClone* plane can significantly improve edge application performance over 5G.

## ACKNOWLEDGMENTS

We thank our shepherd Joerg Widmer and the anonymous CoNEXT reviewers for their valuable feedback.

# REFERENCES


[1] ASN.1. https://asn1.io/. [Online; accessed 03-02-2022].

[2] Data Plane Development Kit. https://www.dpdk.org/. [Online; accessed 03-Feb-2022].

[3] 3GPP Ref #: 29.272. Mobility Management Entity (MME) and Serving GPRS Support Node (SGSN) related interfaces based on Diameter protocol. www.3gpp.org/DynaReport/29272.htm. [28/09/2022].

[4] 3GPP TS 33.501. Security architecture and procedures for 5g system. https://www.etsi.org/deliver/etsi_ts/133500_133599/133501/15.04.00_60/ts_133501v150400p.pdf. [Online; accessed 29-09-2022].

[5] 3GPP Ref #: 36.413. S1 Application Protocol (S1AP). www.3gpp.org/dynareport/36413.htm. [Online; accessed 28-09-2022].

[6] 3GPP. 3GPP Specifications. http://www.3gpp.org. [Online; accessed 03-Feb-2022].

[7] 3GPP. Security architecture and procedures for 5G System. https://www.etsi.org/deliver/etsi_ts/133500_133599/133501/15.02.00_60/ts_133501v150200p.pdf. [Online; accessed 29-09-2022].

[8] 3GPP Ref #: 124.301. Non-Access-Stratum (NAS) protocol Stage 3. https://www.etsi.org/deliver/etsi_ts/124300_124399/124301/15.04.00_60/ts_124301v150400p.pdf. [Online; accessed 28-09-2022].

[9] 3GPP Ref #: 23.401. General Packet Radio Service (GPRS) enhancements for Evolved Universal Terrestrial Radio Access Network (E-UTRAN) access. http://www.3gpp.org/DynaReport/23401.htm. [Online; accessed 28-09-2022].

[10] 3GPP Ref #: 23.501. System architecture for the 5G System (5GS). https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3144. [Online; accessed 29-09-2022].

[11] 3GPP Ref #: 24.301. Non-Access-Stratum (NAS) protocol. www.3gpp.org/dynareport/24301.htm. [Online; accessed 28-09-2022].

[12] Affirmed. Virtual Evolved Packet Core (vEPC). https://www.affirmednetworks.com/products-solutions/virtual-evolved-packet-core/. [Online; accessed 29-09-2022].

[13] Fawad Ahmad, Hang Qiu, Ray Eells, Fan Bai, and Ramesh Govindan. {CarMap}: Fast 3d feature map updates for automobiles. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, pages 1063–1081, 2020.

[14] Mukhtiar Ahmad, Syed Muhammad Nawazish Ali, Muhammad Taimoor Tariq, Syed Usman Jafri, Adnan Abbas, Syeda Mashal Abbas Zaidi, Muhammad Basit Iqbal Awan, Zartash Afzal Uzmi, and Zafar Ayyub Qazi. Neutrino: A fast and consistent edge-based cellular control plane. *IEEE/ACM Transactions on Networking*, pages 1–16, 2022.

[15] Mukhtiar Ahmad, Syed Usman Jafri, Azam Ikram, Wasiq Noor Ahmad Qasmi, Muhammad Ali Nawazish, Zartash Afzal Uzmi, and Zafar Ayyub Qazi. A Low Latency and Consistent Cellular Control Plane. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, pages 648–661, 2020.

[16] Ganesh Ananthanarayanan, Ali Ghodsi, Scott Shenker, and Ion Stoica. Effective straggler mitigation: Attack of the clones. In *Presented as part of the 10th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 13)*, pages 185–198, 2013.

[17] Ganesh Ananthanarayanan, Michael Chien-Chun Hung, Xiaoqi Ren, Ion Stoica, Adam Wierman, and Minlan Yu. {GRASS}: Trimming Stragglers in Approximation Analytics. In *11th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 14)*, pages 289–302, 2014.

[18] Ganesh Ananthanarayanan, Srikanth Kandula, Albert G Greenberg, Ion Stoica, Yi Lu, Bikas Saha, and Edward Harris. Reining in the Outliers in Map-Reduce Clusters using Mantri.

[19] Ananthanarayanan, Ganesh and Bahl, Victor and Cox, Landon and Crown, Alex and Nogbahi, Shadi and Shu, Yuanchao. Video analytics-killer app for edge computing. In *Proceedings of the 17th annual international conference on mobile systems, applications, and services*, pages 695–696, 2019.

[20] Lab Animal. 5G-Powered Medical Robot Performs Remote Brain Surgery. https://www.automate.org/blogs/5g-powered-medical-robot-performs-remote-brain-surgery. [Online; accessed 28-09-2022].

[21] AT&T. AT&T on target for virtualizing 75% of its network by 2020. https://www.fiercetelecom.com/telecom/at-t-target-for-virtualizing-75-its-network-by-2020. [Online; accessed 28-09-2022].

[22] AWS. Cloud Gamingat the Edge. https://d1.awsstatic.com/product-marketing/Outposts/AWS%20Gaming%20Infographic.pdf. [Online; accessed 28-09-2022].

[23] Arijit Banerjee, Rajesh Mahindra, Karthik Sundaresan, Sneha Kasera, Kobus Van der Merwe, and Sampath Rangarajan. Scaling the lte control-plane for future mobile access. In *CoNEXT'15*, 2015.

[24] Hafiz Mohsin Bashir, Abdullah Bin Faisal, M Asim Jamshed, Peter Vondras, Ali Musa Iftikhar, Ihsan Ayyub Qazi, and Fahad R Dogar. Reducing tail latency using duplication: a multi-layered approach. In *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies*, pages 246–259, 2019.

[25] Bhardwaj, Romil and Xia, Zhengxu and Ananthanarayanan, Ganesh and Jiang, Junchen and Shu, Yuanchao and Karianakis, Nikolaos and Hsieh, Kevin and Bahl, Paramvir and Stoica, Ion. Ekya: Continuous learning of video analytics models on edge compute servers. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 119–135, 2022.

[26] carla.org. CARLA. https://github.com/carla-simulator/carla.git. [Online; accessed 28-09-2022].

[27] Tushar Deepak Chandra and Sam Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM (JACM)*, 43(2):225–267, 1996.

[28] CISCO. Access and Mobility Management Function. https://www.cisco.com/c/en/us/td/docs/wireless/ucc/amf/2021-04/config-and-admin/b_ucc-5g-amf-config-and-admin-guide_2021-04/m_timers.html. [Online; accessed 28-09-2022].

[29] Cisco. Cellular IoT in the 5G era. https://www.ericsson.com/en/reports-and-papers/white-papers/cellular-iot-in-the-5g-era. [Online; accessed 28-09-2022].

[30] CISCO. Mme administration guide, staros release 21.8. https://www.cisco.com/c/en/us/td/docs/wireless/asr_5000/21-8_6-2/MME-Admin/21-8-MME-Admin.pdf. [Online; accessed 28-09-2022].

[31] Jeffrey Dean and Sanjay Ghemawat. MapReduce: simplified data processing on large clusters. In *OSDI'2004*, 2004.

[32] Xavier Defago, Andre Schiper, and Nicole Sergent. Semi-passive replication. In *Proceedings Seventeenth IEEE Symposium on Reliable Distributed Systems (Cat. No. 98CB36281)*, pages 43–50. IEEE, 1998.

[33] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, pages 1–16, 2017.

[34] Elbamby, Mohammed S and Perfecto, Cristina and Bennis, Mehdi and Doppler, Klaus. Toward low-latency and ultra-reliable virtual reality. *IEEE Network*, 32(2):78–84, 2018.

[35] ERICSSON. The essential building blocks of E2E network slicing. https://foryou.ericsson.com/eso-network-slicing-e2e-paper.html. [Online; accessed 28-05-2022].

[36] ETSI. ETSI TS 124 501 V15.1.0 (2018-10). https://www.etsi.org/deliver/etsi_ts/124500_124599/124501/15.01.00_60/ts_124501v150100p.pdf. [Online; accessed 28-09-2022].

[37] ETSI. GS MEC 002: Multi-access Edge Computing (MEC); Framework and Reference Architecture. https://www.etsi.org/committee/1425-mec. [Online; accessed 28-09-2022].

[38] ETSI. GS MEC 002: Multi-access Edge Computing (MEC); Phase 2: Use Cases and Requirements. https://www.etsi.org/committee/1425-mec. [Online; accessed 28-09-2022].

[39] ETSI. MEC in 5G networks. https://www.etsi.org/images/files/ETSIWhitePapers/etsi_wp28_mec_in_5G_FINAL.pdf. [Online; accessed 28-09-2022].

[40] Facebook. Facebook Magma. https://github.com/magma/magma. [Online; accessed 28-09-2022].

[41] Caroline Frost. 5G is being used to perform remote surgery from thousands of miles away, and it could transform the healthcare industry. https://www.businessinsider.com/5g-surgery-could-transform-healthcare-industry-2019-8.

[42] Peter Garraghan, Xue Ouyang, Renyu Yang, David McKee, and Jie Xu. Straggler root-cause and impact analysis for massive-scale virtualized cloud datacenters. *IEEE Transactions on Services Computing*, 12(1):91–104, 2016.

[43] Google. FlatBuffers. https://opensource.google/projects/flatbuffers. [Online; accessed 28-09-2022].

[44] Google/GitHub. FlatBuffers. https://google.github.io/flatbuffers/. [Online; accessed 28-09-2019].

[45] Zhenyu Guo, Sean McDirmid, Mao Yang, Li Zhuang, Pu Zhang, Yingwei Luo, Tom Bergan, Madan Musuvathi, Zheng Zhang, and Lidong Zhou. Failure recovery: When the cure is worse than the disease. In *14th Workshop on Hot Topics in Operating Systems (HotOS {XIV})*, 2013.

[46] Guy Moshkowich. Architecture of ZAB – ZooKeeper Atomic Broadcast protocol. https://distributedalgorithm.wordpress.com/2015/06/20/architecture-of-zab-zookeeper-atomic-broadcast-protocol/. [Online; accessed 28-09-2022].

[47] He, Yuze and Ma, Li and Jiang, Zhehao and Tang, Yi and Xing, Guoliang. VIeye: semantic-based 3D point cloud registration for infrastructure-assisted autonomous driving. In *Proceedings of the 27th Annual International Conference on Mobile Computing and Networking*, pages 573–586, 2021.

[48] Huawei Lab. Cloud VR Network; Solution White Paper. https://www.huawei.com/minisite/pdf/ilab/cloud_vr_network_solution_white_paper_en.pdf. [Online; accessed 28-09-2022].

[49] Hunt, Patrick and Konar, Mahadev and Junqueira, Flavio Paiva and Reed, Benjamin. ZooKeeper: Wait-free Coordination for Internet-scale Systems. In *USENIX annual technical conference*, volume 8, 2010.

[50] IETF. Stream Control Transmission Protocol. https://tools.ietf.org/html/rfc2960. [Online; accessed 28-09-2022].

[51] Intel. Intel Nucleus. https://github.com/omec-project/Nucleus. [Online; accessed 28-09-2022].

[52] Intel. UPF-EPC. https://github.com/omec-project/upf-epc.git. [Online; accessed 28-09-2020].

[53] Open Air Interface. Open Air Interface. https://github.com/OPENAIRINTERFACE/openair-mme. [Online; accessed 28-09-2022].

[54] Jeff Dean. Designs, Lessons and Advice from Building Large Distributed Systems. https://www.cs.cornell.edu/projects/ladis2009/talks/dean-keynote-ladis2009.pdf. [Online; accessed 28-09-2022].

[55] Jeff Dean. Jeff Dean on Google Infrastructure. https://perspectives.mvdirona.com/2008/06/jeff-dean-on-google-infrastructure/. [Online; accessed 28-09-2022].

[56] Xin Jin, Li Erran Li, Laurent Vanbever, and Jennifer Rexford. SoftCell: Scalable and Flexible Core Network Architecture. In *CoNEXT'13*, 2013.

[57] Ju1ce. April-Tag-VR-FullBody-Tracker. https://github.com/ju1ce/April-Tag-VR-FullBody-Tracker. [Online; accessed 28-09-2022].

[58] Ryan Junguk Cho and Z. Jacobus Van. Mobilestream: A scalable, programmable and evolvable mobile core control plane platform. In *MobiCom'18*, 2018.

[59] Data Center Knowledge. Why Google Cloud and AT&T May Merge their Telco Edges. https://www.datacenterknowledge.com/edge-computing/why-google-cloud-and-att-may-merge-their-telco-edges. [Online; accessed 28-09-2022].

[60] Konstantinos Krikellas, Sameh Elnikety, Zografoula Vagena, and Orion Hodson. Strongly consistent replication for a bargain. In *2010 IEEE 26th International Conference on Data Engineering (ICDE 2010)*, pages 52–63. IEEE, 2010.

[61] Sathiya Prabhu Kumar, Raja Chiky, Sylvain Lefebvre, and Eric Gressier Soudan. Libre: A consistency protocol for modern storage systems. In *Proceedings of the 6th ACM India Computing Convention*, pages 1–9, 2013.

[62] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. In *Concurrency: the Works of Leslie Lamport*, pages 179–196. 2019.

[63] Leslie Lamport et al. Paxos made simple. *ACM Sigact News*, 32(4):18–25, 2001.

[64] Li Erran Li, Z. Morley Mao, and Jennifer Rexford. Toward Software-Defined Cellular Networks. In *IEEE, 2012 European Workshop on Software Defined Networking*, 2012.

[65] Yuanjie Li, Zengwen Yuan, and Chunyi Peng. A control-plane perspective on reducing data access latency in lte networks. In *Proceedings of the 23rd Annual International Conference on Mobile Computing and Networking*, pages 56–69, 2017.

[66] Li, Sihan and Zhou, Hucheng and Lin, Haoxiang and Xiao, Tian and Lin, Haibo and Lin, Wei and Xie, Tao. A characteristic study on failures of production distributed data-parallel programs. In *2013 35th International Conference on Software Engineering (ICSE)*, pages 963–972. IEEE, 2013.

[67] Limelight. EDGE COMPUTE FOR GAMING. https://www.limelight.com/resources/data-sheet/edge-compute-for-gaming/. [Online; accessed 28-09-2022].

[68] Shaoshan Liu, Liangkai Liu, Jie Tang, Bo Yu, Yifan Wang, and Weisong Shi. Edge computing for autonomous driving: Opportunities and challenges. *Proceedings of the IEEE*, 107(8):1697–1716, 2019.

[69] Vincent Liu, Daniel Halperin, Arvind Krishnamurthy, and Thomas Anderson. F10: A fault-tolerant engineered network. In *10th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 13)*, pages 399–412, 2013.

[70] M-CORD. Mobile CORD: Enabling 5G on CORD. http://opencord.org/wp-content/uploads/2016/03/M-CORD-March-2016.pdf. [Online; accessed 28-09-2022].

[71] Maxim. Open-source VR headset with SteamVR support. https://github.com/relativty/Relativty. [Online; accessed 28-09-2022].

[72] Microsoft. About Edge Zone Preview. https://docs.microsoft.com/en-us/azure/networking/edge-zones-overview. [Online; accessed 28-09-2022].

[73] Microsoft. Video analytics at the edge, an ideal technology for 5G cloud monetization. https://azure.microsoft.com/en-us/blog/video-analytics-at-the-edge-an-ideal-technology-for-5g-cloud-monetization/. [Online; accessed 28-09-2022].

[74] Microsoft. What is Azure Private 5G Core Preview? https://docs.microsoft.com/en-us/azure/private-5g-core/private-5g-core-overview. [Online; accessed 28-09-2022].

[75] Mehrdad Moradi, Yikai Lin, Z. Morley Mao, Subhabrata Sen, and Oliver Spatscheck. SoftBox: A Customizable, Low-Latency, and Scalable 5G Core Network Architecture. *IEEE Journal on Selected Areas in Communications*, 36:438–456, 2018.

[76] Mehrdad Moradi, Karthikeyan Sundaresan, Eugene Chai, Sampath Rangarajan, and Z Morley Mao. SkyCore: Moving core to the edge for untethered and reliable UAV-based LTE networks. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*, pages 35–49, 2018.

[77] Mehrdad Moradi, Wenfei Wu, Li Erran Li, and Zhuoqing Morley Mao. SoftMoW: Recursive and Reconfigurable Cellular WAN Architecture. In *CoNEXT'14*, 2014.

[78] Mukhtiar Ahmad, Syed Muhammad Ali Nawazish, Muhammad Taimoor Tariq, Muhammad Basit Iqbal Awan, Raza Taqi and Zafar Ayyub Qazi. CellClone's Github Repo. https://github.com/nsgLUMS/neutrino. [Online; accessed 25-09-2022].

[79] Mukhtiar Ahmad, Syed Muhammad Ali Nawazish, Muhammad Taimoor Tariq, Syed Usman Jafri, Adnan Abbas, Syeda Mashal Abbas Zaidi, Muhammad Basit Iqbal Awan, Zartash Afzal Uzmi and Zafar Ayyub Qazi. Neutrino's Github Repo. https://github.com/nsgLUMS/cellclone. [Online; accessed 25-09-2022].

[80] Vasudevan Nagendra, Arani Bhattacharya, Anshul Gandhi, and Samir R. Das. MMLite: A Scalable and Resource Efficient Control Plane for Next Generation Cellular Packet Core. In *Proceedings of the 2019 ACM Symposium on SDN Research*, SOSR '19, 2019.

[81] Arvind Narayanan, Eman Ramadan, Jason Carpenter, Qingxu Liu, Yu Liu, Feng Qian, and Zhi-Li Zhang. A first look at commercial 5G performance on smartphones. In *Proceedings of The Web Conference 2020*, pages 894–905, 2020.

[82] NextEPC. NextEPC. https://github.com/open5gs/open5gs. [Online; accessed 28-09-2022].

[83] Ng4T. Next generation Telecommunication Technology Testing Tools. https://www.ng4t.com/wireshark.html. [Online; accessed 28-09-2022].

[84] Binh Nguyen, Tian Zhang, Bozidar Radunovic, Ryan Stutsman, Thomas Karagiannis, Jakub Kocur, and Jacobus Van. ECHO: A reliable distributed cellular core network for hyper-scale public clouds. In *Mobicom '18*, page 163–178, New York, NY, USA, 2018. ACM.

[85] China Academy of Information and Communications Technology. *Virtual Reality/Augmented Reality White Paper.* [Online; accessed 28-09-2022].

[86] Diego Ongaro and John Ousterhout. In search of an understandable consensus algorithm. In *2014 {USENIX} Annual Technical Conference ({USENIX} {ATC} 14)*, pages 305–319, 2014.

[87] OpenAirInterface. OpenAirInterface: A 5G software alliance for democratising wireless innovation . http://www.openairinterface.org/, 2022. [Online; accessed 28-09-2022].

[88] Jose Ordonez-Lucena, Pablo Ameigeiras, Diego Lopez, Juan J Ramos-Munoz, Javier Lorca, and Jesus Folgueira. Network slicing for 5G with SDN/NFV: Concepts, architectures, and challenges. *IEEE Communications Magazine*, 55(5):80–87, 2017.

[89] Tien-Dat Phan, Guillaume Pallez, Shadi Ibrahim, and Padma Raghavan. A new framework for evaluating straggler detection mechanisms in mapreduce. *ACM Transactions on Modeling and Performance Evaluation of Computing Systems (TOMPECS)*, 4(3):1–23, 2019.

[90] Zafar Ayyub Qazi, Melvin Walls, Aurojit Panda, Vyas Sekar, Sylvia Ratnasamy, and Scott Shenker. A high performance packet core for next generation cellular networks. In *SIGCOMM'17*, 2017.

[91] Xiaoqi Ren, Ganesh Ananthanarayanan, Adam Wierman, and Minlan Yu. Hopper: Decentralized speculation-aware cluster scheduling at scale. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, pages 379–392, 2015.

[92] R. Saravanan and N. Ramaraj. Experiences: Design, Implementation, and Deployment of CoLTE, a Community LTE Solution. In *MobiCom '19*, 2019.

[93] Seshadri Sathyanarayan. Standalone (SA) and Non-Standalone (NSA) 5G Architectures: The various paths to 5G revenues and profitability. https://www.affirmednetworks.com/sa-and-nsa-5g-architectures-the-path-to-profitability/. [Online; accessed 28-09-2022].

[94] Satyanarayanan and Mahadev. The emergence of edge computing. *Computer*, 50(1):30–39, 2017.

[95] ScreenBeam. How to Reduce Latency or Lag in Gaming. https://www.screenbeam.com/wifihelp/wifibooster/how-to-reduce-latency-or-lag-in-gaming-2/. [Online; accessed 28-09-2022].

[96] SDxCentral. Amazon Wavelength. https://aws.amazon.com/wavelength/. [Online; accessed 28-09-2022].

[97] Shi, Shu and Gupta, Varun and Hwang, Michael and Jana, Rittwik. Mobile VR on edge cloud: a latency-driven design. In *Proceedings of the 10th ACM multimedia systems conference*, pages 222–231, 2019.

[98] srsEPC. srsEPC: an open-sourec 5G core network. https://github.com/srsran/srsRAN/tree/master/srsepc. [Online; accessed 28-09-2022].

[99] Tuyen X. Tran and Dario Pompili. Joint task offloading and resource allocation for multi-server mobile-edge computing networks. *IEEE Transactions on Vehicular Technology*, 68(1):856–868, 2019.

[100] Verizon. Virtual Network Services. https://www.verizon.com/business/products/networks/virtual-network-services/. [Online; accessed 28-09-2022].

[101] Virsabi. How 5G and edge computing will transform AR & VR use cases. https://stlpartners.com/articles/edge-computing/how-5g-and-edge-computing-will-transform-ar-vr-use-cases/. [Online; accessed 28-09-2022].

[102] Neeraja J Yadwadkar, Bharath Hariharan, Joseph E Gonzalez, and Randy Katz. Multi-task learning for straggler avoiding predictive job scheduling. *The Journal of Machine Learning Research*, 17(1):3692–3728, 2016.

[103] Matei Zaharia, Andy Konwinski, Anthony D Joseph, Randy H Katz, and Ion Stoica. Improving MapReduce performance in heterogeneous environments. In *Osdi*, volume 8, page 7, 2008.

[104] Zhang, Wuyang and Chen, Jiachen and Zhang, Yanyong and Raychaudhuri, Dipankar. Towards efficient edge cloud augmentation for virtual reality MMOGs.

In *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*, pages 1–14, 2017.

[105] Zhang, Xumiao and Zhang, Anlan and Sun, Jiachen and Zhu, Xiao and Guo, Y Ethan and Qian, Feng and Mao, Z Morley. EMP: Edge-assisted multi-vehicle perception. In *Proceedings of the 27th Annual International Conference on Mobile Computing and Networking*, pages 545–558, 2021.

[106] Junhui Zhao, Qiuping Li, Yi Gong, and Ke Zhang. Computation offloading and resource allocation for cloud assisted mobile edge computing in vehicular networks. *IEEE Transactions on Vehicular Technology*, 68(8):7944–7956, 2019.

[107] ZooKeeper. Two-phased Commit. https://zookeeper.apache.org/doc/r3.3.3/recipes.html#sc_recipes_twoPhasedCommit. [Online; accessed 28-09-2022].

# A CTA STATE CACHING AT THE CPF FOR FAILURE RECOVERY

Lets consider an example scenario to illustrate how CTA failure is handled in our design. Assume that the sample procedure depicted in figure 18 is an *initial attach* procedure, the processing of which contains some non-deterministic operations. Let's suppose the state update due to M1 (with logical clock 20) is a non-deterministic operation (i.e., M-TMSI allocation) and the CTA fails just after replying for M1 to the UE and before synchronizing state changes from the fastest replica (B in this case) on A. In this case, the new CTA can recover from the temporary state available on replicas A and B, however, an issue is that both the replicas are out-of-sync due to the last non-deterministic state update. However, the information, that M-TMSI was allocated by replica B is relayed to the user, is available on replica B (as discussed in §4.4). So the new CTA first synchronizes replica A with the state from replica B and after that marks replica A up-to-date

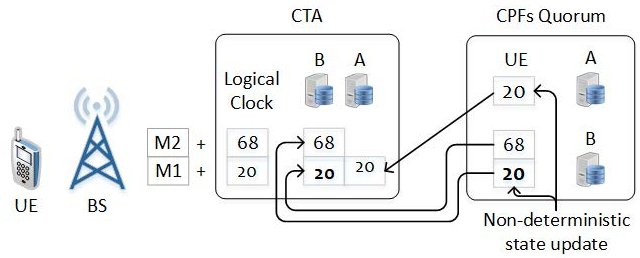


**Figure 18: CTA failure recovery with the distributed state available on the CPFs.**

# B ARTIFACT APPENDIX

## B.1 Abstract

This section provides an overview of the artifacts used for this paper. The artifacts include the source-code that implements our proposed design, instructions to install dependencies, deploy the source-code, configure the system, run experiments, and collect the results. Using the artifacts, you should be able to reproduce all the control plane evaluation results. The source-code of the proposed design is in C/C++ language and can be compiled with standard gnu compilation tool-chain.

Our experimental setup spans multiple servers. The artifacts includes a custom framework to automate all the control plane experiments.

## B.2 Description

*B.2.1 How to access.* All the artifacts, including the source-code and documentation, are available through the following sources:

- https://github.com/nsgLUMS/cellclone
- https://doi.org/10.6084/m9.figshare.21300813.v1

The archived version at https://figshare.com/ should provide a public access for more than 10 years. Providing both the GitHub repository and the FigShare archive in the paper allows us to maintain the archive and to have a citable version that reflects the state when the artifacts were granted.

*B.2.2 Hardware dependencies.* Our setup requires minimum 3 Linux machines equipped with DPDK compatible network interface cards with at least 3 ports (§6.1). Complete detail is provided in the README file in the main directly of the source-code.

*B.2.3 Software dependencies.* For a full list of dependencies, please see README file in the main directly of the source-code. The minimal requirement is Linux, Python and DPDK [2].

*B.2.4 Data sets.* This software requires no data set. All the required data is internally generated by the software.

## B.3 Installation

We have provided detailed instructions in the README file to setup network and install the software.

## B.4 Experiment workflow

- Use *config.json* to configure a particular experiment.
- Run *run_experiments.py* to execute an experiment. Experiments may take a few minutes depending on the configuration.
- Run *do_stats.py* to collect results.
- Run *plot.py* to plot the results.

A detailed step by step guide to run an experiment is provided in the README file in the main directory of the source-code.

## B.5 Evaluation and expected results

The README file in the main directory contains a sample figure and detailed instructions on how to configure the system to reproduce this figure. Once this figure is reproduced, further instructions are provided to generate other evaluation results in the paper.