A Low Latency and Consistent Cellular Control Plane

Mukhtiar Ahmad, Syed Usman Jafri, Azam Ikram, Wasiq Noor Ahmad Qasmi,
Muhammad Ali Nawazish, Zartash Afzal Uzmi, Zafar Ayyub Qazi
Department of Computer Science, SBASSE, LUMS, Pakistan

ABSTRACT

5G networks aim to provide ultra-low latency and higher reliability to support emerging and near real-time applications such as augmented and virtual reality, remote surgery, self-driving cars, and multi-player online gaming. This imposes new requirements on the design of cellular core networks. A key component of the cellular core is the control plane. Time to complete control plane operations (e.g. mobility handoff, service establishment) directly impacts the delay experienced by end-user applications. In this paper, we design Neutrino, a cellular control plane that provides users an *abstraction of reliable access to cellular services while ensuring lower latency*. Our testbed evaluations based on real cellular control traffic traces show that Neutrino provides an improvement in control procedure completion times by up to 3.1× without failures, and up to 5.6× under control plane failures, over existing cellular core proposals. We also show how these improvements translate into improving end-user application performance: for AR/VR applications and self-driving cars, Neutrino performs up to 2.5× and up to 2.8× better, respectively, as compared to existing EPC.

CCS CONCEPTS

• **Networks → Control path algorithms**; *Network protocol design*; *Middle boxes / network appliances*; *Wireless access points, base stations and infrastructure*; *Network reliability*;

KEYWORDS

Cellular Core, Control Plane, Consistency

ACM Reference Format:
Mukhtiar Ahmad, Syed Usman Jafri, Azam Ikram, Wasiq Noor Ahmad Qasmi, Muhammad Ali Nawazish, Zartash Afzal Uzmi, Zafar Ayyub Qazi. 2020. A Low Latency and Consistent Cellular Control Plane. In *Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication (SIGCOMM '20), August 10–14, 2020, Virtual Event, NY, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3387514.3406218

1 INTRODUCTION

The next-generation cellular networks are envisioned to support emerging and near real-time applications such as augmented and virtual reality, remote surgery, self-driving cars, cognitive assistance apps, and multi-player online gaming. Such applications require higher reliability and ultra-low latency, in the order of 10ms [53]. Consequently, 5G networks are aiming to support Edge Computing [53], with edge applications hosted closer to the users [21–23].

This, in turn, requires cellular providers to evolve their cellular core;[1] a key part of the cellular network which connects the IP backbone with the base stations and implements cellular-specific processing on user's control and data traffic. In existing cellular deployments, the cellular core functions are typically deployed in remote data centers which can cause path inflation [60, 62]. In 5G and beyond, cellular providers are expected to move these core functions to the edge (e.g., cell towers and central offices), resulting in a highly distributed core network architecture [24, 38, 48].

We postulate that the main challenge in providing low latency and reliable access with an edge-based core stems from the complexity of the control plane in the cellular core; unlike the internet control plane, the cellular control plane needs to keep dynamic state for each user device to support mobility. The cellular control plane regularly updates this user state to both establish and retain user's data access. The control plane establishes the user's data access to the internet or to other operator services by setting up the connectivity session for each device. This process requires installing states at the user device, base station, and core network elements. When moving to another base station, the control plane is responsible for retaining the data access by migrating the ongoing session states to the user's new location. Upon failures, the control plane needs to recover or recreate the session state.

Prior work [37] has shown that the time taken by the cellular control plane to process control traffic can have a direct impact on the delay experienced by user applications, e.g., control functions can contribute up to 1 s delay in session establishment. Moreover, mobile devices frequently generate control traffic; on average a mobile device generates a session establishment request every 106.9 s. In addition, failures of control plane functions can exacerbate these delays [14, 37], causing disruptions in data access [46].

To make matters worse, with 5G, the control traffic is expected to increase rapidly [44, 47] because of (i) a shift to smaller cell sizes, which will likely cause more mobility handoffs [13] and (ii) the proliferation of IoT devices with high control to data traffic ratio [35]. In addition, failures are expected to be commonplace in 5G core networks; similar to large service provider networks [25], core network deployments will be large and are increasingly based on software-based network functions (NFs) running on commodity hardware [24]. As a result, timely completion of control plane procedures in the presence of failures is going to be vital to provide reliable and low latency data access to user applications.

We identify the following key challenges in an edge-based cellular core for providing reliable and low latency data access.

SIGCOMM '20, August 10–14, 2020, Virtual Event, NY, USA

ACM ISBN 978-1-4503-7955-7/20/08…$15.00
https://doi.org/10.1145/3387514.3406218

[1] Cellular packet core for 4G/LTE is called Evolved Packet Core (EPC) and for 5G networks is referred to as 5G Core.

- **UE-Core State inconsistency:** Prior work [37, 46, 56] has shown that inconsistent user state, between the user equipment (UE) and the core network, can cause significant disruptions in data access for the users in 4G/LTE networks. We find that UE-Core inconsistencies, and thus prolonged disruptions in data access, can also occur in current 5G standardization proposals (§3). We also find recent proposals for cellular control plane do not provide consistency guarantees [14, 43] and, thus, offer no protection against such disruptions in data access. We argue cellular control plane needs to provide *state consistency guarantees* to provide reliable data access. A key challenge in this regard is to design a consistency protocol which provides fast failure recovery with minimal failure-free overhead.
- **Slow state updates:** We show that one key reason for delays in control traffic is slow processing of state updates between a user device, base station, and cellular control plane. Existing 4G/LTE networks and 5G proposals use ASN.1 [11, 20] for serializing control updates. Our experiments on real control traffic traces show that ASN.1 serialization can become a potential bottleneck for latency-sensitive applications like real-time augmented and virtual reality (§3).
- **Frequent control handovers:** Control handovers happen when a user moves from one location domain to another, requiring migration to a different control plane function (§3). A control handover can cause significant delays in data access, up to 1.9 s [37]. In an edge-based cellular core, control handovers will happen more frequently, making it important that these handovers be completed quickly. A key challenge is to migrate user state quickly to the appropriate control plane function.

To address the above challenges, we design, Neutrino, a cellular control plane that provides users an *abstraction of reliable access to cellular services while ensuring lower latency*. In designing Neutrino, we synthesize several existing techniques in distributed systems. Below, we describe the keys ideas in Neutrino:

- **Consistent UE processing:** We design a consistency protocol that minimizes service disruption under failures by ensuring that user devices are always able to receive *Read your Writes* consistency[58]. The consistency protocol provides fast failure recovery and small failure-free overhead. It uses a primary-backup state replication scheme, and consists of (i) per-procedure checkpointing and non-blocking synchronization of state, (ii) fast in-memory message logging, and (iii) a two-level failure recovery protocol (§4.2).
- **Fast serialization engine:** We design a serialization scheme for cellular state updates by optimizing FlatBuffers [28](§4.4). Our scheme significantly speeds up the processing of state updates, by up to 19.1× in comparison to ASN.1, while providing a relatively smaller increase in bandwidth usage.
- **Proactive geo-replication:** We reduce the delay incurred in control plane handovers by performing proactive geo-replication of user state (§4.3). Our results show this can lead to 7× improvement in completing control plane handovers.

We implement Neutrino, the redesigned cellular control functions and control traffic aggregator nodes.[2] It requires minimal changes to base stations which, instead of ASN.1, use Neutrino's FlatBuffers-based serialization engine. In rolling out 5G deployments, carriers are expected to make major upgrades on base stations [13], hence, we expect upgrading the serialization engine would not be a hindrance in adopting Neutrino.

[2]These nodes act as front-end load balancers from base stations to cellular core [14].

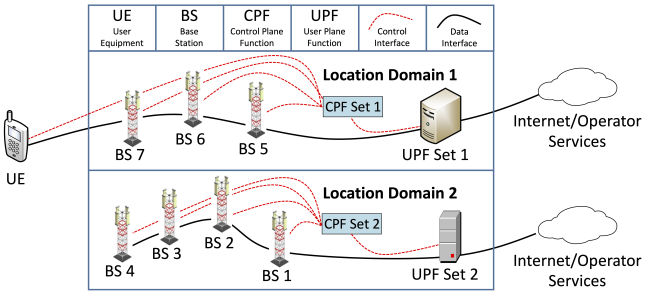

Figure 1: 5G network architecture.

We evaluate Neutrino's performance by replaying real control traffic traces through a DPDK-based traffic generator in our testbed. Our results show an improvement in control procedure completion times by up to 3.1× without failures, and up to 5.6× under control plane failures, over existing EPC. Neutrino handles bursty IoT control traffic better than existing EPC, showing up to 2× improvement in median control procedure times (§6.3 and §6.4). We also quantify the impact of Neutrino on application performance: for AR/VR applications and self-driving cars, Neutrino performs 2.5× and 2.8× better, respectively, as compared to existing EPC. Neutrino also improves video startup latency by up to 37× and reduces page load times by 3.2× (§6.6). In comparison to recent proposals, Neutrino performs 3.4× and 1.3× better than SkyCore [40] and DPCM [37], respectively (§6.3). To motivate Neutrino's design choices, we also perform a factor analysis (§6.7).

2 BACKGROUND AND MOTIVATION

2.1 Cellular Network Architecture

Figure 1 shows a simplified 5G network architecture, consisting of three main types of components: Base Stations (BS), Control Plane Functions (CPF), and User Plane Functions (UPFs).[3] This high-level 5G network architecture is similar to that in 4G/LTE wherein each User Equipment (UE) is provided radio access via a nearby base station. The user plane functions are responsible for delivering data and voice traffic, over the core network infrastructure, to/from the internet and operator's multimedia services, respectively. The control plane functions facilitate data delivery by providing support for user mobility, session management, radio resource allocation, and authentication.

The control plane establishes the user's data access to the internet or to other operator services by setting up the connectivity session for each device. This process requires installing states at the user device, the base station, and core network elements. When a user moves closer to a new base station, a *handover* takes place during which the control plane is responsible for retaining the data access by migrating the ongoing session states to the user's new location.

[3]In 5G nomenclature, the base station is referred to as the gNodeB or simply the gNB, and the key control plane function which interfaces with the base stations, is called Access and Mobility Management Function (AMF).

In the event of a failure, the control plane needs to recover or recreate the session state.

To support user mobility over a wider area, the cellular deployments are divided into *location domains*.[4] Each domain has multiple BSs and UPFs, and is managed by a set of CPFs. When a user device moves from one location domain to another, it triggers a *control handover* to a different set of CPFs and UPFs.

2.2 Impact of Cellular Control Processing

The cellular control plane is critical in providing low latency data access to mobile users. Prior studies have shown:

- **Control plane latency directly impacts data latency:** A 19-month study [37] conducted across four major US carriers, showed that control functions contribute 72.5–999.6 ms latency in session establishment. The control handover can cause up to 1.9 s for the delay in data access.
- **Mobile devices frequently generate control traffic:** The same study showed that on-average a mobile device generates session establishment request every 106.9 s. With high-mobility applications (like autonomous vehicles) and smaller cell sizes in 5G,[5] we would expect a rapid increase in the frequency of control traffic.
- **Failures exacerbate control plane delays:** Failures in the control plane can exacerbate these control plane delays [14, 37, 46], causing up to 11 s delay in data access. These failures happen rather frequently; 4-5% of the control connection requests experience some sort of failure [37].

2.3 Cellular Edge Applications

One of the key goals in 5G is to build support for Edge Computing to enable latency-sensitive applications [23].[6] Cellular providers have already started testing/deploying edge applications:

- **Edge-based video analytics**: Verizon, has recently conducted edge computing tests over its 5G network. It hosted an AI-powered face recognition application inside its edge network. The test consisted of transmitting video of a crowd over Verizon's 5G network to an edge application, which would scan the faces in that crowd for potential matches against a database. The goal was to show police could use edge computing functions to quickly find someone in a crowd, rather than having to wait for that video to be sent to a distant data center to be analyzed [38].
- **Accelerating online mobile gaming:** China mobile in partnership with Tencent Cloud is providing accelerated online mobile gaming experience through its 5G-based edge deployment [57]. Many multi-player online games (e.g., Fortnite) need sub-100 ms latency for smooth player control which can be hard to provide with remote clouds.
- **Real-time augmented/virtual reality (AR/VR)**: Typical displays on AR/VR devices have a refresh rate of between 60 Hz and 90 Hz which translates into delay budgets of 11.1 ms and 16.7 ms, respectively. To enable real-time AR/VR, Verizon now provides support for offloading mobile headsets' graphical computations to Verizon's 5G wireless edge [34].

[4] Termed as *tracking/registration area* in 4G/5G.
[5] 5G base stations are expected to also support millimeter-wave and underutilized UHF frequencies between 300 MHz and 3 GHz, leading to smaller cell sizes.
[6] This is referred in the 5G world as Multi-access Edge Computing (MEC).

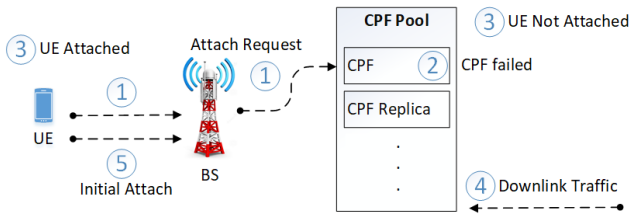

Figure 2: An example scenario resulting in inconsistent user state between the UE and core network.

To support edge applications, cellular core functions also need to be provisioned at the edge. We consider a deployment model (details provided in §4.3) where cellular core functions are hosted on the edge: cell towers or central offices.[7]

3 CHALLENGES

In this section, we discuss the key challenges in providing low latency and reliable access in an edge-based cellular control plane.

3.1 User State Inconsistency

A common strategy to provide fault tolerance in the cellular control plane is to replicate the UE state across multiple CPFs [14]. With partition tolerance, this replication strategy makes it impossible to simultaneously provide availability and state consistency across replicas (CAP Theorem). Given this fundamental constraint, recent proposals [14, 43] on cellular control plane simply do not provide any consistency guarantees. For example, SCALE [14] updates the UE state on the replicas asynchronously, only when a UE transitions from the connected state to an idle state. Between these transitions, there can be many UE state changes, and, therefore, SCALE cannot guarantee that a replica will always have the updated UE state. 5G core system architecture as proposed in 3GPP also does not provide any protocol to keep CPFs consistent [8].

Sacrificing consistency in UE state across replicas leads to an inconsistent user state between the UE and the core network, causing major delays [46, 56]. We use Figure 2 and illustrate this through the following example:

(1) UE attaches to the network by executing an *Initial Attach* procedure[8] on the CPF through the base station.
(2) After the completion of the *Initial Attach* procedure, and before updating the UE state at the replica, the CPF fails.
(3) At this stage, the user state at the UE is *Attached* whereas it is *Not Attached* at the core network.
(4) At the stage, if the user receives a voice call or downlink data, the core network will not be able to send it to the UE.
(5) The UE will get the connectivity back when it either re-executes *Attach* or sends a location update message and it fails.

A similar case in the context of 4G/LTE networks was observed in [46] leading to the UE not having data access for several minutes. An important point to note is that in case of a CPF failure, a UE

[7] Telecom providers have many small central offices that were traditionally used to host switching equipment and serve as aggregation points. These are now being considered to host edge applications and core functions [38, 39].
[8] Refers to a sequence of request/response control messages between the UE/BS and CPF to create necessary UE control state.

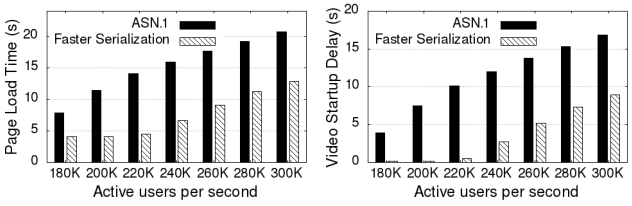

Figure 3: Page load time and video startup delay.

can be used to recreate a consistent UE state at the core network by issuing a *Re-Attach* request [37].

3.2 Slow State Updates

We show that one key reason for the slow processing of state updates between a user device, base station, and cellular control plane in 5G and existing 4G/LTE is the slow serialization of control messages. Control messages between the UE/BS and CPF are serialized using ASN.1 [11]. Our experiments on real control traffic traces show that ASN.1 serializing can become a potential source of a bottleneck in edge-based cellular core deployments.[9] Figure 3 shows the impact of using a faster serialization on video startup delay and page load times. We observe that by changing the serialization, we could significantly improve page video startup delay (up to 37×) and page load time (up to 3.2×).[10]

ASN.1 is used in cellular networks because it provides (i) backward compatibility with existing cellular networks and (ii) small encoded message size, however, it is very slow at encoding/decoding cellular control messages. Our analysis of cellular control messages based on 3GPP standard [11] and real control traces [45] shows that a single control message can consist of multiple elements and the data in these messages is organized hierarchically, with potentially multiple nested elements. When decoding these messages, ASN.1 has to (i) traverse all the previous bytes before accessing a field and (ii) perform additional memory allocations during decoding, increasing the overall processing time.

3.3 Frequent Control Plane Handovers

An edge-based cellular deployment will be highly geo-distributed. As UEs can be mobile, one challenge is to ensure that the UEs are mapped to close-by edge nodes to ensure low latency. However, if done naively,[11] this can also introduce frequent CPF handovers, causing extra delays in the processing of control traffic. Control handovers are known to cause significant delays in data access [37], often unacceptable for latency sensitive applications. In our experiments with a self driving car application, we observe that during control handovers, up to 90% of the application deadlines can be missed (§6.6).

4 DESIGN

To address the challenges in section §3, we design, Neutrino, a new cellular control plane. In this section, we first describe our goals in designing Neutrino and the resulting approach (§4.1). We then describe in detail (i) the design of Neutrino's consistency protocol that provides reliable state access by guaranteeing *Read your Writes* to every mobile user, while incurring little overhead (§4.2), (ii) the design of a proactive geo-replication scheme that helps in reducing the delay in control handovers (§4.3), and (iii) the design of a serialization scheme that speeds up user state updates (§4.4).

4.1 Design Goals and Approach

The primary goal driving the design of Neutrino is to provide **low control plane latency** to support latency-sensitive applications in an edge-based cellular core. Neutrino aims to provide low latency under (i) normal scenarios without any failures, (ii) with CPF failures, and (iii) in scenarios with frequent control handovers. Neutrino aims to provide the UEs with **consistent processing** under failures: any CPF replica processing a UE traffic, always operates on the latest UE state, guaranteeing "Read your Writes" consistency to the mobile client. Also, Neutrino's performance should **scale** well with the number of devices and the volume of the signaling traffic.

Below, we provide an overview of the key ideas behind Neutrino's design:

- **Replication with Two-levels of Failure Recovery:**
  - *Replicated UE State Store:* In Neutrino, UE control state is replicated to provide fault tolerance. We use a primary-backup scheme. The replicas are updated asynchronously, resulting in *non-blocking*, thus fast execution of control updates in the non-failure case. The motivation is to add minimal delay on the critical path of control traffic processing. Replicas are updated after the completion of every control procedure. However, as our synchronization protocol is *non-blocking* it does not ensure that replicas are always consistent.
  - *Two-level Failure Recovery:* To ensure *consistent processing* of UE traffic, we provide two levels of failure recovery. We keep an in-memory log of control messages at the control traffic aggregator node, and in the event of a CPF failure, if a replica CPF is not up to date, we replay these messages at the replica CPF to reconstruct the lost state. In the event of a CTA failure, the UE *Re-Attaches* to the network through a new CTA and recreates a consistent UE state at a new CPF.
- **Proactive Geo-Replication:** To minimize overheads of control handovers, we use proactive structured geo-replication, building on the idea of Geographical Hash Tables [52]. The key idea is to proactively replicate user state in a wider geographical area, to reduce the overhead of a UE moving from one geographical region to another. Neutrino implements this idea by using multiple consistent hash rings; these rings represent progressively larger geographical regions, and proactively replicates UE state in a CPF in a larger region to minimize control handover delays. We describe the detailed procedure in §4.3.
- **Fast Serialization Engine:** In §3, we showed that slow control updates can become a bottleneck in enabling latency-sensitive applications in an edge-based cellular core deployment. We show in §6.7.4, multiple existing serialization schemes provide faster control message processing than ASN.1; we evaluate FlexBuffer [26], Protocol Buffers [27], Fast-CDR [3], LCM [5], and FlatBuffers [28] with ASN.1 [1]. We observe that FlatBuffers provides the best trade-off in terms of performance and expressiveness. It significantly speeds-up message processing time and can be used to

[9] We used Packed Encoding Rules (PER) of ASN.1 in our experiments.
[10] For details regarding the experimental setup, please refer to §6.
[11] For example, always mapping the UE to the closest edge node.

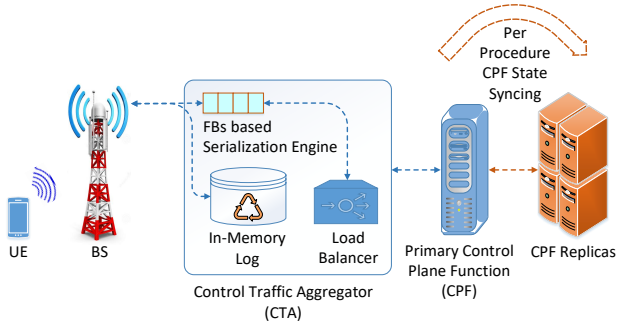

Figure 4: Neutrino's system architecture diagram.

express different types of control messages, e.g., cellular control message widely use unions and unsigned data types which are not supported by LCM. However, for cellular control messages, the encoded message sizes in FlatBuffers can be significantly larger than ASN.1 messages. To address this issue, we optimize FlatBuffers for cellular control traffic, resulting in lower encoded message sizes and slightly faster message processing times. We describe these optimizations in §4.4.

Resultant Architecture: Neutrino is an edge-based cellular core design, which re-architects the implementation of the key cellular control plane entity; referred to as Mobility Management Entity in 4G/LTE packet core [10], and Access and Mobility Management Function (AMF) and Session Management Function (SMF) in 5G [8]. We introduce Control Traffic Aggregator (CTA), a new addition in our design, which is similar to a front-end load balancer between the BS and control plane. Besides acting as a load balancer, the CTA, in addition, helps in maintaining the state consistency. In existing architectures, either front-end load balancers [14, 43] or the BS itself implements load balancing [7] for control traffic, however there is no support for state consistency.

BSs directly connect to the CTA. The CTA is responsible for (i) message logging, (ii) forwarding traffic to/from appropriate CPF, and (iii) CPF failure detection and recovery. CPF is the key control plane entity which (i) stores and updates user state based on requests from the UE/BS, (ii) creates, deletes and modifies data sessions on the key data plane entity UPF (User Plane Function), (iii) handles UE registration and mobility across the BSs and (iv) check-points user state on the replica CPF(s) upon procedure completion. UE's requests are served by one CPF at a time which is called primary CPF. An up-to-date CPF replica becomes primary in case the primary CPF fails. Figure 4 shows the overall flow of control traffic in Neutrino. BSs route control traffic to a nearby CTA. CTAs are responsible for mapping requests from UEs to a CPF, and routing responses back to the UE.

4.2 Consistency Protocol

User equipment (UE) is served by a single CPF at a time. This CPF is responsible for updating and storing the UE state (which includes the BS ID, data plane endpoint identifiers, and user tracking area). UE state updates are mainly originated from the UE (or a BS on behalf of the UE) and executed by the CPF and propagated to the rest of the network functions (NFs). These updates are generally in the form of request/response message patterns. Control procedures such as *service establishment* and *handover to another BS* are composed of several control messages.

4.2.1 UE State Consistency. The UE state between the UE and CPF needs to be consistent (see section 3.1). For example, UE and CPF should have the same copy of the tracking area list [11], otherwise, the core network (CN) may not be able to page or deliver downlink data to the UE. This state consistency requirement for a UE in a cellular system maps to *Read your Writes* consistency: *The effect of an update operation by a process (UE) on a data item x (UE state) will always be seen by a successive read operation on x by the same process (same UE)* [15, 16, 19, 58, 59].

If the state of a UE is maintained at just one CPF, the *Read your Writes* consistency is readily maintained. This is because, even if the CPF fails, the control messages never have to operate on stale data. In this case, the CPF failure event forces the UE to *Re-Attach* and recreate a consistent UE state [37]. It may be noted that even though *Re-Attach* is a valid option in the existing cellular systems to maintain consistency, it can lead to long delays in data access.

4.2.2 Consistency versus Availability. To increase the availability, our design allows replication of the UE state from the primary CPF onto N backup CPFs. A backup CPF takes over when the primary CPF fails and, therefore, the replicated UE state on the backup CPFs should remain consistent with that on the primary CPF.

If the state replication amongst the CPFs (from primary to backups) is done *synchronously*, it will lead to failure-free overheads (in terms of extra delays[12]) in the control plane, undermining the original design goal. Therefore, Neutrino chooses *asynchronous* updates of the UE state. Whenever a control procedure completes at the primary CPF, we replicate the user state onto the backup CPFs. We pick per-procedure state synchronization because it adds a smaller delay in the Procedure Completion Times (PCT) as compared to per-message synchronization (depicted in Figure15).

4.2.3 The CTA Message Log. Since the UE state updates from the primary CPF to the backup CPFs are asynchronous, there is no *inherent* consistency guarantee (similar to SCALE [14], also see section 3.1). Thus, there may be a case that on the failure of a primary CPF, none of the backup CPFs are up-to-date. In such a situation, *Read your Writes* consistency is ensured by maintaining a temporary message log at the control traffic aggregator (CTA). The complete process is as follows:

(1) On receiving each control message, the CTA associates with it a logical clock (for tracking all messages and keeping those in order) and writes it to volatile memory. After appending the logical clock to the message, it is also forwarded to the primary CPF.
(2) When a control procedure completes, the primary CPF sends state updates, for the particular UE, to all the backup CPFs along with the logical clock of the last message of the procedure; this logical clock is used to identify the end of a particular procedure in the log.
(3) Every replica node (backup CPFs) sends an ACK to the CTA after successful state synchronization. Reception of the ACK

[12]The primary CPF would block the state write acknowledgement and, therefore, will not be able to timely respond to the control plane messages from the UE/BS.

at the CTA ensures that the backup replica has up-to-date state for the UE which can be used to serve the UE in the event the primary CPF fails.

(4) CTA stores the ACK received from all the replicas with the last message of the procedure in the store.

To keep the CTA message log size in check, the CTA periodically scans the message store and *prunes* all the messages corresponding to a procedure for which ACKs have been received from all the backup CPFs.

4.2.4 Out-of-date Backup CPFs. We now describe the process of **marking a UE state as outdated** in a replica.

(1) The CTA periodically scans the last message of every procedure in the log store and if:
   (a) An ACK is not received from one or multiple replicas for a configurable timeout (e.g., 30 seconds in our case, because procedure completion times are at most a few seconds), it informs the corresponding replica(s) that the particular UE's state is outdated and also provides (i) the list of CPFs (if exists) having up-to-date UE's state, and (ii) the logical clock associated with the last message of the procedure (this is used to ignore the reception of outdated state).
   (b) Replica(s) mark the state of the particular UE outdated.
   (c) If a CPF is successful in fetching up-to-date UE's state from the list of CPFs provided by the CTA, it marks UE's state up-to-date.
   (d) CTA deletes all the messages belonging to this procedure.

(2) If a replica receives state update for a UE, which was previously marked outdated, UE's state in the CPF is marked up-to-date.

(3) If a CPF receives a request from a UE for which it does not have an up-to-date state, UE is asked to *Re-Attach*.

(4) If a second control procedure starts for a UE for which ACK is not received from one or multiple replicas, CTA informs the corresponding replica(s) that the UE's state is outdated and provides a list of CPFs having an up-to-date state.

4.2.5 Consistency and Failure Recovery. We describe the recovery process under each of the failure scenarios enumerated below (also depicted in Figure 5).

Failure Scenario 1 (Primary fails - backup up-to-date): In this scenario, primary CPF fails, however, there exists a backup replica which has successfully synchronized with the primary CPF on the completion of the last procedure for the UE. In addition, there are no ongoing procedures from the UE. In this case, the back-up replica is up-to-date and satisfies *Read your Write* consistency.

Failure Scenario 2 (Primary fails - message replay on backup): Primary CPF fails while there is an ongoing procedure from the UE. However, the backup replica has successfully synchronized with primary on completion of the previous procedure. In this case, the CTA replays all the stored messages on the back-up replica to make it up-to-date before serving the UE. After the messages are replayed, the backup replica is up-to-date, and *Read your Write* consistency is maintained.

Failure Scenario 3 (Primary fails - all replicas out-of-sync): Primary CPF fails while no backup replica exists for the UE which was synchronized with the primary on previous procedure completion. In this case, we avoid the UE from operating in an outdated state. Instead, we recreate a consistent UE state at the CPF, by asking the UE to execute the *Re-Attach* procedure. The *Re-Attach* procedure constructs consistent UE state, and any subsequent reads operate on this state, hence satisfying *Read your Writes* consistency.

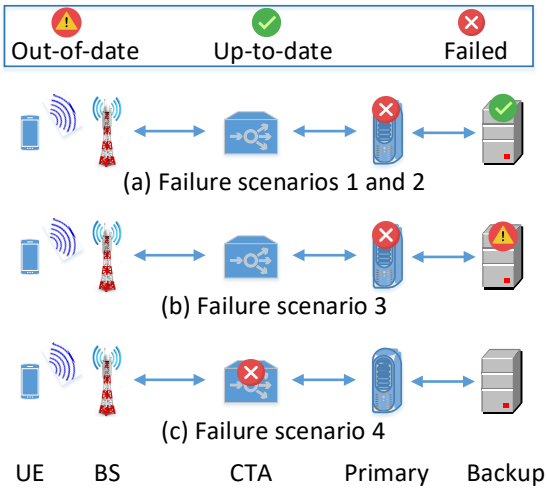

Figure 5: Failure scenarios.

Failure Scenario 4 (CTA fails): In this case, the failure recovery procedure is similar to one used in failure scenario 3. When a CTA fails, the UE executes the *Re-Attach* procedure, through a new CTA, creating (i) fresh state for the UE at new CPF(s) and (ii) a mapping of the UE to a specific CPF on the new CTA.

In failure scenarios 1 and 2, Neutrino completely masks failure from the UE. In failure scenario 3, we do not have an up-to-date state for the UE in the core network. However, we prevent the UE from operating on an outdated state, thus maintaining *Read your Writes* consistency. However, as UE is asked to *Re-Attach*, this can impact application performance because of the time required to execute *Re-Attach*, as shown in Figure 9. As we do not backup CTA state, recovery in failure scenario 4 is exactly similar to that of scenario 3. In summary, Neutrino provides consistency in all the above failure scenarios and significantly improves delay to recover from failures in scenarios 1 and 2.

4.3 Proactive Geo-replication

As discussed in the previous section, a single CPF serves as a primary for each UE, and the state is replicated on N number of backup replica(s). In this section, we discuss how this state can be used to expedite handovers.

We first discuss our deployment model. Figure 6 shows our deployment model. We divide the deployment area into regions. The unit region, which we call the level-1 region, consists of multiple BSs, one CTA, and a pool of CPFs.[13] There are multiple options for deploying CTA and CPFs. One option is co-locating the CTA with the pool of CPFs (e.g., in a central office) which would serve all the BSs in a level-1 region. Another option is deploying CTA and CPFs on different edge nodes (e.g., combination of central towers and

[13] As a CTA also performs load balancing of the control traffic from the BSs, deploying a CTA at each BS is not a viable option.

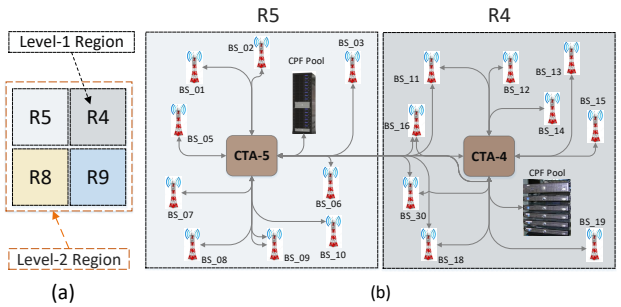

Figure 6: Deployment model. Left (a) shows regions defined through geo-hashing, and Right (b) zooms into two sub-regions containing CTAs and CPFs.

central office). For our evaluation, we assume CTA is co-located with the CPFs serving a region. This option simplifies deployment, and provides potentially lower propagation delays.

We assign each level 1 a geo-hash [4]. Each character in the geo-hash represents two bits (one bit for GPS longitude and one for latitude). If we take one character off from the geo-hash, we get a geo-hash which points to a four times bigger region which we call level-2 region. Geo-hash is not an essential requirement in our design and the operator can manually define these regions however our existing implementation is based on geo-hashes. Figure 6 (a) shows four level-1 regions (R5, R4, R8, and R9). All these four regions are collectively called the level-2 region. Figure 6 (b) shows our deployment model.

Multiple consistent hash rings: BSs route control traffic to the nearest CTA module. Each CTA implements two consistent hash rings; (i) level-1 hash ring consists of all the CPFs in the level-1 region and (ii) level-2 hash ring includes all the CPFs in the level-2 region.[14] When CTA receives a control message from the UE, it extracts a unique user ID,[15] and hashes it to the level-1 ring to determine the primary CPF for the UE. When a control procedure completes, the primary CPF replicates the user state on N consecutive replicas on a level-2 ring (not included in the level-1 ring) based on the hash of the user's ID. For example, if in Figure 6(a), a user is served by a CPF in the R5 region, its state will be replicated on any N replicas from region R5, R8, or R9 based on the hash of the user ID. Replicating state in other regions has certain advantages, which include, (i) when handing over to a BS in the nearby region, the user may find up-to-date state already present, and (ii) different regions may have different failure modes. Based on the description of the regions, we implement the following types of handovers.

Intra-region hand-over: If a UE moves from one BS to another BS within the same level-1 region in Neutrino, no CPF handover is required. Consider the example of the deployment described in Figure 6(b). If a UE move across the BSs within level-1 region where CTA of the source and target cell remains the same, e.g. from BS_02 to BS_06 in region R5, the CPF serving the UE does not change therefore no state migration is required. Such handovers are fast because only BS change is required without any UE state migration between CPFs.

Inter-CPF hand-over: Inter-CPF handover in the 4G/LTE system is costly in terms of latency as the UE state needs to be migrated to the target CPF. In edge deployment, where CPFs will be placed closer to the UEs, the frequency of inter-CPF handovers will increase. Neutrino provides a mechanism for faster handover between the CPFs by proactively replicating UE's state in the target region.

Consider the following example from the deployment scenario in Figure 6(b). A UE moves across the level-1 regions where CTA of the source cell is different from the target cell; e.g., a UE moves from BS_10 (R5, CTA-5) to BS_30 (R4, CTA-4). When BS_10 determines that handover (HO) to the target BS_30 is required and both BSs are sharing the same level-2 rings, UE state migration before the HO can be avoided. This is because Neutrino already keeps the state of each UE on a CPF replica in the level-2 ring. We call such a HO, *Fast Handover*.

4.4 Serialization Engine

We observe that FlatBuffers [28] provides the best trade-off in terms of performance and expressiveness. It significantly speeds-up message processing times and can represent different types of control messages, e.g., we observed cellular control message widely use unions and unsigned data types which are not supported by LCM. FlatBuffers (FBs) is fast because it provides (i) direct access to inner fields via pointers during decoding and (ii) needs no additional memory allocation during encoding.

However, the performance benefits of FBs come at a cost: large encoded message size. To provide fast access to inner fields, the FBs compiler couples a *vtable* containing pointers with each table containing data elements. The pointers in the vtable contain byte offsets which are used to directly access a field in the encoded message. While this speeds up the decoding time, these vtables contribute significantly to the encoded message size. ASN.1 PER on the hand follows a Length-Value encoding scheme and simply couples the length of a table with its contents (also encoded as Length-Value pairs). Hence resulting in a smaller encoded buffer size as compared to FBs.

Optimizing FlatBuffers: Cellular control messages[16] makes wide use of unions containing single data elements. However, FBs only supports tables in unions. When a single field needs to be added to a union, it first has to be wrapped in a table. This is inefficient because this wrapping generates a vtable for the single field. Since the union contains a single field, traversal using the vtable pointers is unnecessary. Ideally, we should be able to directly access the single field. To address this issue, we introduced a new data type, named *svtable*, that generates less metadata for single-value fields in unions. Our optimization reduces 10 bytes for single scalar fields in unions and 14 bytes for single variable length fields in tables while also reducing the encoding and decoding times (see Figure 19 and 20). Other possible optimizations, include, introducing a bit string data type (since FBs currently only supports byte strings) and variable-length data types, e.g., an integer in FBs is always encoded in 4 bytes, even if it could be represented using fewer bytes.

[14] One can potentially implement more than 2 consistent hash rings, however, there are tradeoffs. We leave this exploration for future work.

[15] We specifically use MME-temporary mobile subscriber identity (M-TMSI) when the UE is in idle state and the unique S1AP identifier for the UE in MME, when the UE is active. Similar to [31], CTA assigns these two identifiers the same value for a UE during the initial attach procedure.

[16] Specifically, message part of the S1AP protocol and NGAP protocol.

5 IMPLEMENTATION

We have implemented Neutrino and all supporting components and functions in 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 2.2 [2]) for fast I/O operations and is similar to the traffic generator used in [50]. It replays real cellular control traffic traces [45] and is also capable of serializing those control messages using ASN.1 and our modified serialization scheme. In our implementation, the base station communicates with the CTA using S1AP—the protocol currently used in 4G/LTE networks between the base stations and control plane functions.[17]

Message De/Serialization: We implemented the message serialization (both ASN.1 and our FlatBuffers-based modified serialization scheme) for all the control messages included in all the control procedures supported by our CPF implementation (see below). The ASN.1 compiler we used is the same as used for OpenAirInterface [49], while the implementation of our serialization code is built upon an open-source implementation of FlatBuffers [28].

Geo-Replication: For geo-replication, we implemented 2 bits per character version of the Geo Hashing similar to the one used in [4], thus causing a four-fold increase/decrease in the region size with each character.

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 a CPF instance that is selected based on the UE ID in the packet. We have implemented a message logging module at the CTA using the standard C++ STL map container. We have also implemented a consistent hashing based load balancing scheme within the CTA, obviating the need for separate load balancers.

Control Plane Function: Our CPF implementation supports the following four control procedures: (*i*) *initial attach*, (*ii*) *handover* with CPF change (*iii*) *FastHandover* and (*iv*) *service request*. To coordinate various control procedures, we have implemented state machines at both the CPF and the traffic generator. Our CPF implementation also includes the module responsible for state replication. All experiments and evaluations are performed with five CPF instances, each running on two CPU cores (one for processing requests and the second one for state synchronization).

6 EVALUATION

In this section, we show the following:

(1) **Impact on control procedure completion times:** Neutrino performs 2.3×, 3.4×, and 1.3× better in median PCT than existing EPC, SkyCore, and DPCM respectively.
(2) **Impact under failures:** In the case of CPF failure, it reduces median PCT by 5.6× as compared to the existing EPC.
(3) **Impact on control handovers:** Neutrino improves handover with CPF change by up to 3.1× in the median PCT. Neutrino also implements a *FastHandover* which expedites handover operation with proactive replication by up to 7×.
(4) **Impact on data applications:** Our evaluation shows the impact of frequent mobility on a self-driving car and AR application. We also evaluate impact on web browsing and video streaming applications.
(5) **Factor Analysis:** To better understand Neutrino's design choices and separate the benefits of Neutrino's different design ideas, we perform a series of micro-benchmarks. These include (i) comparison with different state synchronization schemes, (ii) overhead of message logging and (iii) comparison of different serialization techniques, to motivate the choice of FlatBuffers.

[17] A similar protocol is also used in 5G networks.

6.1 Setup and Methodology

Our test setup consists of two servers running Ubuntu 18.04.3 with kernel 4.15.0-74-generic. Each server is a dual-socket with 18 cores per socket, IntelXeon(R) Gold 5220 CPU @ 2.20GHz, and dual NUMA nodes having total memory 128GB. Both servers are also equipped with Intel X710 40 Gb (4 x 10) NIC. For testing control traffic, we use an implementation of the S1AP and NAS protocol [50] and implement the handling of request and response messages between the UE/BS and CPF for different control procedures. These experiments are run with real signaling traces from a commercial traffic generator and RAN emulator from ng4T [45]. We generate two different types of traffic patterns: (i) 10 Gbps bursty traffic to emulate a large number of IoT devices sending requests in a synchronized pattern, and (ii) uniform traffic to emulate a pre-specified number of control procedure requests per second. We run all experiments for 60 s.

6.2 Baselines

We compare the performance of Neutrino against the following designs:

Existing EPC: It is a modified version of the OpenAirInterface [49] codebase, uses ASN.1 based serialization, and requires UEs to *Re-Attach* on a CPF failure. Instead of kernel sockets, existing EPC uses DPDK [2] for fast I/O operations.

Neutrino: It is a modified version of the existing EPC which (i) uses optimized FlatBuffers-based serialization instead of ASN.1, (ii) uses fast failure recovery as in §4.2 and (iii) performs structured state replication.

DPCM: It is the same as existing EPC except control procedures are modified (BS receives state from the UE) as described in [61].

SkyCore: It is also a modified version of the existing EPC which synchronizes user state on each control message [40].

Next, we discuss the key evaluation results.

6.3 Latency Improvements in Procedure Completion Time (PCT) with Neutrino

This section presents PCT for `attach`, `handover`, and `service request` procedures in the non-failure scenario.

PCT - uniform traffic: Figure 7 shows service request PCT comparison of the existing EPC, DPCM, and SkyCore with Neutrino. The figure shows that for uniform traffic rate of up to 120K Procedures Per Second (PPS), Neutrino performs 2.3×, 1.3×, and 3.4× better than the existing EPC, DPCM, and SkyCore, respectively. Onward 140 KPPS, existing EPC, and SkyCore are unable to handle the arrival rate, resulting in a drastic increase in PCT. At 200 KPPS

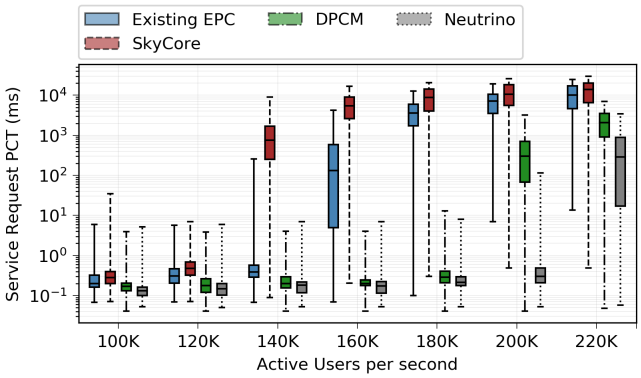

Figure 7: Impact in control procedure completion times.

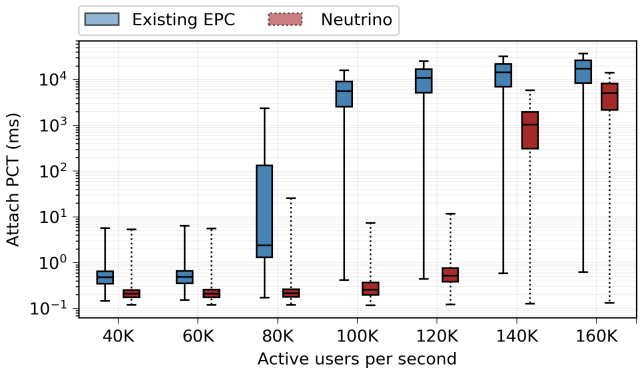

Figure 8: `attach` PCT by a varying number of procedures per second with uniform traffic.

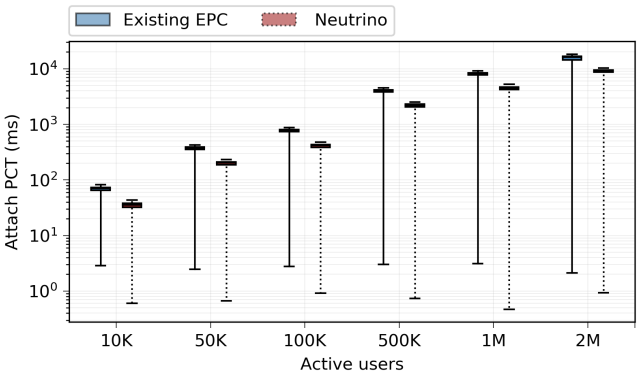

Figure 9: `attach` PCT with a varying number of active users with bursty control traffic.

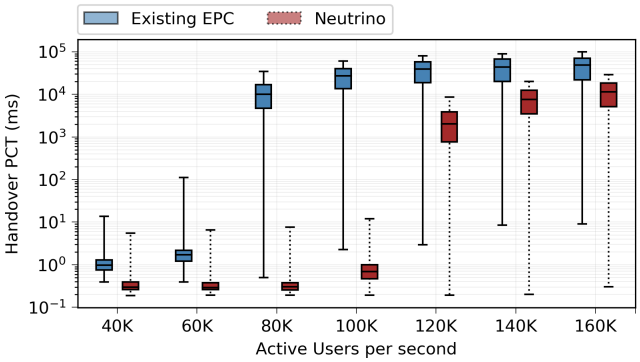

Figure 10: `handover` PCT under failure with varying number of active users with uniform traffic.

and higher rates, PCT drastically increases for all the schemes but Neutrino still performs better as compared to the other schemes.

We perform `initial attach` and `handover` PCT experiments with uniform control traffic. Figure 8 shows PCT distribution for the `initial attach` procedure. The figure shows that till 60 KPPS, Neutrino performs up to 2.3× better than the existing EPC in median PCT. Onward 60 KPPS, existing EPC fails to meet the arrival rate of the requests and queue starts building up. We called this region, existing EPC's saturation region. In existing EPC's saturation region, the median PCT for existing EPC drastically increases while it remains low for Neutrino. Onward 120 KPPS, Neutrino is unable to meet the message arrival rate and the queue builds up. We call this Neutrino's saturation region. In this region, Neutrino performs up to 3.4× better than the existing EPC in terms of median PCT. Neutrino can better meet high arrival rates as compared to all other schemes. In all these cases, the primary source of improvement is Neutrino's fast message serialization.

PCT - Bursty Traffic: Figure 9 shows PCT distribution for an `initial attach` procedure with varying number of active UEs, when using Neutrino and existing EPC. Due to the high arrival rate for bursty traffic model, queues immediately build-up for both Neutrino and existing EPC. The figure shows that Neutrino performs up to 2× better than the existing EPC for a bursty traffic model.

6.4 PCT under failure with Neutrino

We conduct experiments with CPF failures for both Neutrino and existing EPC. PCT under failure for existing EPC includes the time taken by the UE in executing the procedure before the failure, as well as the time taken to re-attach to another CPF after the failure. In the case of Neutrino, PCT under failure includes the time taken by the UE to execute the procedure before the failure and the time secondary CPF takes to replay the stored messages to recover the lost user state. In both cases, PCT does not include failure detection time.

Figure 10 shows PCT distribution under CPF failure for `handover` procedure with uniform traffic. We observe an improvement of up to 5.6× in median PCT when the procedure arrival rate is less than 60 KPPS. In addition, to faster serialization, this improvement is attributed to faster state recovery in Neutrino. Instead of re-attaching the UE on a CPF failure, CTA module sends logged messages to the replica CPF, which then replays them to reconstruct the state updates, saving multiple RTTs.

6.5 Fast Handover in Neutrino

Figure 11 shows the comparison of the PCT for handover in existing EPC, Neutrino - Default (in this case, user state migration is required before handover completion) and Neutrino - Proactive (user state is proactively replicated in the target region to implement Fast handover, as discussed in section 4.3). The Figure shows

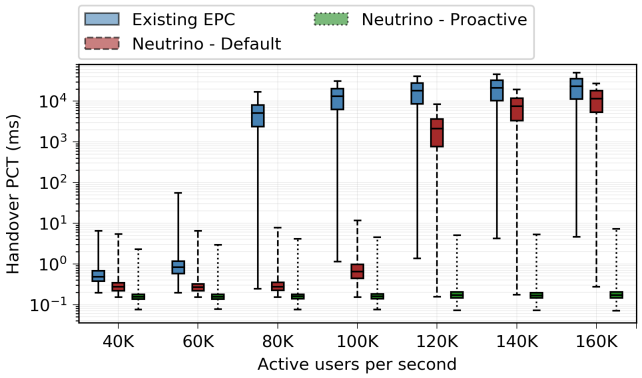

Figure 11: `Fast handover` procedure completion times with uniform traffic.

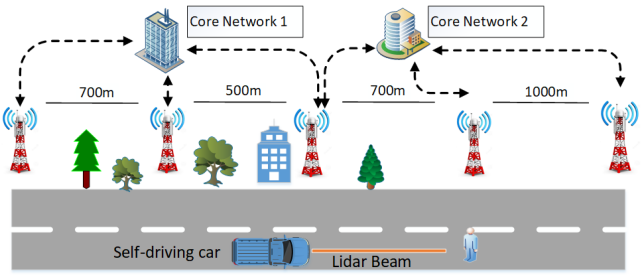

Figure 12: An example scenario of frequent control handovers with high mobility applications in an edge-based cellular core.

that Neutrino - Proactive improves median PCT by up to 7× over existing EPC, when the procedure arrival rate is less than 60 KPPS. Above 60 KPPS, existing EPC is unable to meet the arrival rate, and PCT increases drastically.

6.6 Impact on application performance

To measure the impact of Neutrino on application performance, we interface Intel's 5G UPF [29] with Neutrino. A UE connected to Neutrino can create a new session, delete an existing session, and modify existing bearer on the UPF through S11 interface [9].

To measure the impact of mobility on the application performance, we set up the client application (on UE) with CARLA self-driving car emulator [17, 18] and an edge application that processes sensors' data. Experiments are performed in two scenarios; (i) while executing a single handover and (ii) executing multiple handovers during a 5 minutes drive at 60 mph with the BS spacing similar to Figure 12. In both scenarios, we set a deadline for the application data. We generate sensor data at a frequency of 1KHz in the uplink direction. At the edge application, we note the the number of packets which missed their application-specific deadline.

Impact on autonomous vehicles and AR/VR: The time budget for a self-driving car to make a decision based on sensors' data is in the order of 100 ms [55]. Figure 13 shows in both single and multiple handover scenarios, Neutrino performs up to 2.8× better than the existing EPC.

Virtual reality (VR) applications, that use head-tracked systems, require a latency of less than 16 ms [53] to achieve perceptual

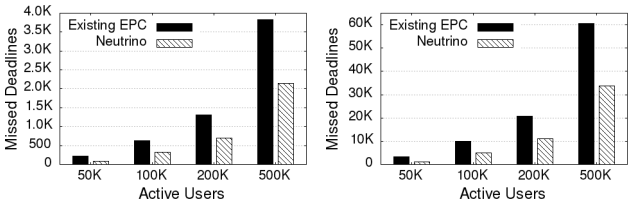

Figure 13: Effect of mobility (LHS: single HO, RHS: multiple HOs) on a self-driving car.

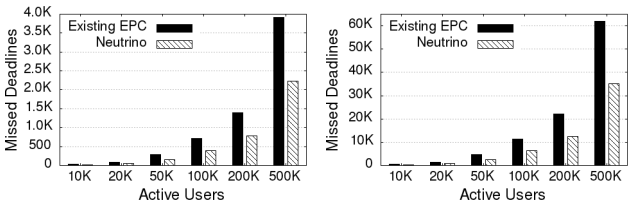

Figure 14: Effect of mobility (LHS: single HO, RHS: multiple HOs) on VR application.

stability. We measure the number of application packets that miss the VR deadline requirement in both single and multiple mobility scenarios. Figure 14 shows that Neutrino performs up 2.5× better than the existing EPC.

Impact on video startup latency and page load: The second set of experiments are for a stationary UE, in idle state, starting a web browsing or video streaming application. To get data access, UE needs to execute `service request` procedure to set up a data channel for the application. Application startup latency in this scenario is a function of `service request` PCT. This experiment measures the average (i) video startup delay and (ii) page load time (PLT). To avoid network variation in the video startup delay, Apache webserver replays locally stored videos. Video startup delay is measured using DASH player. Figure 3 shows a video startup delay comparison between the Neutrino and existing EPC while CPF is handling a varying number of active users. The Figure shows that Neutrino performs up to 37× better than existing EPC in terms of median video startup delay. Page load time is equal to (i) `service request` PCT plus (ii) average page load time of the top 10 Alexa pages. To filter out network variations, MITM proxy [6] is used to replay locally stored web pages. A Firefox web browser extension *Load Time* is used to measure page load time. Figure 3 shows that Neutrino performs up to 3.2× better than the existing EPC in terms of median PLT.

6.7 Factor Analysis

Below, we discuss the results of our micro-benchmark experiments.

6.7.1 Impact of state synchronization on PCT. We compare the overhead of different replica synchronization schemes on control plane latency. Figure 15 shows the attach PCT distribution for three different schemes; (i) *No Rep*: no message logging and state replication, (ii) *Per Msg Rep*: with message logging and per-message state replication and (iii) *Per Proc Rep*: with message logging and per procedure state replication. Figure 15 shows that per-message state replication has the highest median PCT, due to frequent state locking for check-pointing. Per-procedure state replication has a

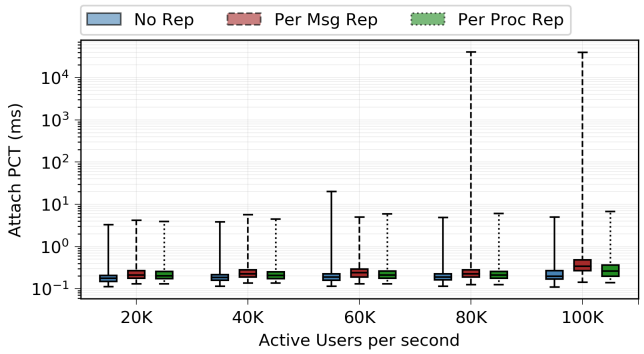

Figure 15: Effect of state synchronization on attach PCT.

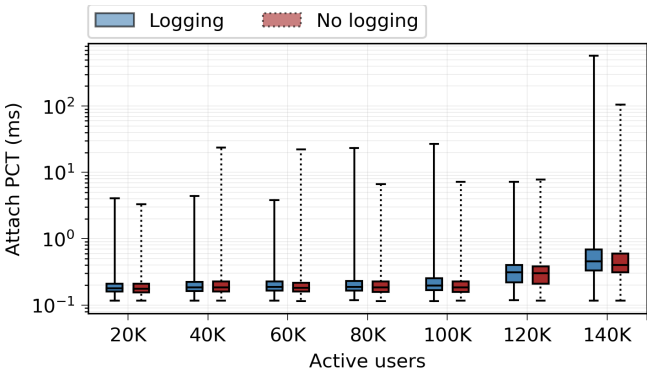

Figure 16: Impact of message logging on attach PCT.

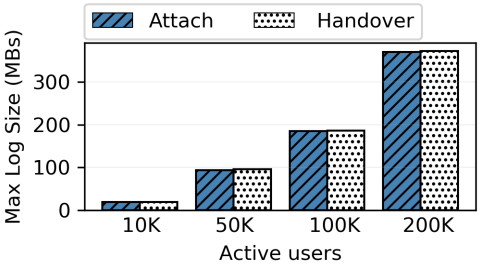

Figure 17: Log size variation with the number of active users.

slightly higher median PCT as compared to *No Rep*, but it provides the best trade-off between consistency and overhead in PCT.

6.7.2 Impact of message logging on PCT. We performed *attach* procedure with and without message logging enabled. Figure 16 shows PCT distribution for the attach procedure. The figure shows message logging has a negligible impact on the PCT in Neutrino and the reason is in-memory logging is fast.

6.7.3 Message log size at the CTA. Figure 17 shows the maximum log size at CTA with varying total number of active users and the type of procedures being performed with per-procedure synchronization. The figure shows log size grows by increasing the number of active users, however even with 200K active users, it remains less than 400 MBs.

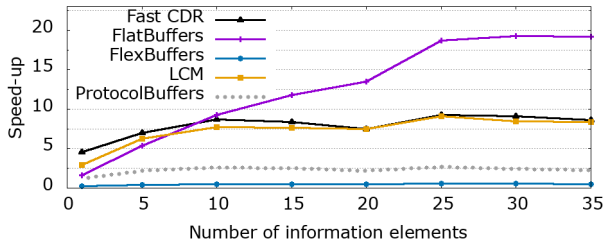

Figure 18: Improvement in encoding + decoding times in comparison to ASN.1.

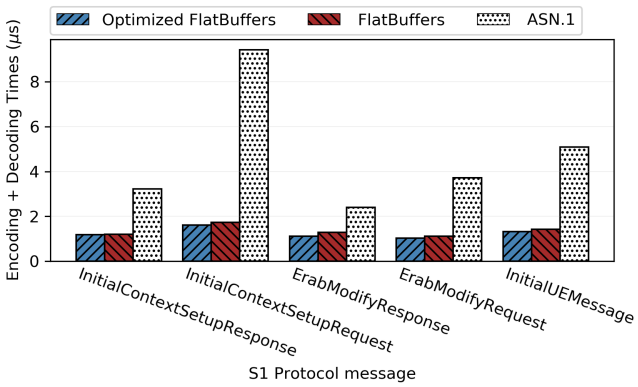

Figure 19: Encoding + decoding with Optimized FlatBuffers.

6.7.4 Serialization benefits. We motivate the choice of FlatBuffers (FBs) for serializing cellular control messages over several serialization schemes; FlexBuffer [26], Protocol Buffers [27], Fast-CDR [3] and LCM [5] with ASN.1 [1]. We compare the time to decode and encode the control messages. For these experiments, we construct a custom message with varying number of data elements/fields.

Encoding + Decoding times: Figure 18 shows the speedup in the total encoding plus decoding time as compared to the ASN.1 serialization scheme, for a custom control message with varying number of data fields. For messages with data elements less than 7, Fast-CDR and LCM perform better. When data elements increase beyond 7, FBs is the clear winner. For 25 data elements, the total speedup in encoding + decoding time for FBs is twice that of the next best scheme. The speedup in comparison with ASN.1 is between 1.6× to 19.2×. We note here that all cellular control messages we tested, contained a minimum of 8 data elements.

Tests with real control messages: We next compare Optimized FBs with FBs and ASN.1 over a subset of real control messages. We specifically quantify both the encoding + decoding times as well as the increase in encoded message size with FBs. In Figure 19 we observe a decrease of up to 5.9× in encoding + decoding times with FlatBuffers over ASN.1. There is a further decrease with Optimized FBs in some cases. However, this decrease does come at a cost; the encoded message size in FBs can add up to 300 bytes of more metadata than ASN.1 (Figure 20). With Optimized FBs, we can save up to 32 bytes of data per message.

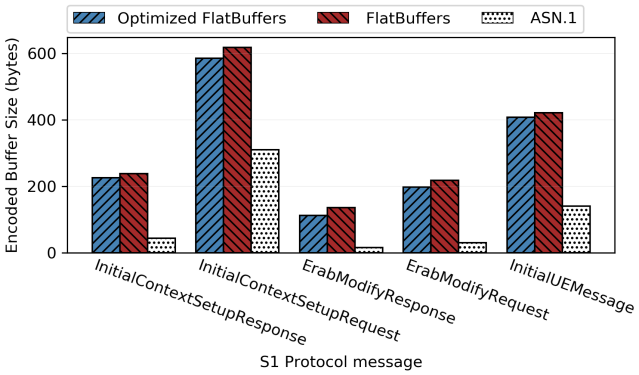

Figure 20: Size comparison of encoded messages between Optimized FBs, FBs and ASN.1.

7 DISCUSSION

In this section, we discuss Neutrino's deployability and serialization tradeoffs.

Deployability: Neutrino is designed for next-generation cellular networks like 5G and 6G. It only requires minimal changes to BS; instead of ASN.1, BS use Neutrino's FBs-based serialization engine. In rolling out 5G deployments, cellular providers are expected to make major upgrades on BS [13], hence, we expect upgrading the serialization engine would not be a hindrance in adopting Neutrino. However, Neutrino's serialization engine is not compatible with previous generations of cellular networks (4G/LTE and earlier), although Neutrino's consistency protocol (§4.2) and proactive state replication scheme (§4.3) can seamlessly work with 4G/LTE.

Serialization tradeoff: Neutrino's serialization scheme reduces encoding/decoding time by a factor of up to 19.2× at the cost of an increase in message size. With increasing available bandwidth in cellular networks, we believe this is an acceptable tradeoff for the cellular providers. We are currently investigating further optimizations in Neutrino's FB-based compiler to reduce the overhead in message sizes while retaining its latency benefits.

8 RELATED WORK

Scaling distributed MMEs: There are recent proposals for scaling MMEs [12, 14, 31, 43]. SCALE [14] proposes mechanisms for scaling a software 4G/LTE MME to handle increasing signaling load. It uses state replication to handle MME failures and consistent-hashing based load balancing. MMLite [43] proposes a load balancing solution for MMEs, leveraging skewed consistent hashing to distribute incoming connections more efficiently. However, as we discussed earlier (§2), both SCALE and MMLite do not provide any consistency guarantees. MobileStream [31] proposes a programmable mobile core control platform which decomposes control plane in multiple stateless and one stateful node and externalizes user state. MobileStream provides better programmability of the control plane than other schemes, however, in an edge-deployment scenario, lookups on a remote state store can become a source of increased latency. In comparison to these schemes, Neutrino provides (i) consistency guarantees, (ii) faster serialization, and (iii) faster failure recovery. There are other general proposals for externalizing NF state [32, 33]. However, they are targeted towards centralized cloud deployments.

Reducing control traffic latency: Recent work DPCM [61] also aims to reduce control plane latency. DPCM proposes a client-side solution, which reduces control plane latency by initiating and executing some control operations in parallel by using the device side user state. With DPCM's client-side modifications, Neutrino can further speed-up the processing of control traffic.

Centralized cellular control plane architectures: There are several proposals for architecting an SDN-based cellular core [30, 36, 42]. A common theme in all these works is to have a logically centralized cellular control plane (including all the MME functionality) with a programmable data plane. However, unlike Neutrino, these proposals do not aim to address control plane latency and centralized control plane architectures may not suitable for achieving low latency control traffic in edge deployments.

Consolidated cellular core architectures: There are several proposals for consolidating EPC designs [40, 41, 50]. PEPC [50] slices EPC by the user, consolidating UE state, and refactoring EPC functions. PEPC improves the overall EPC performance, however, it does not consider control plane fault tolerance. Similarly, SoftBox[41] proposes a consolidated EPC architecture but does not provide fault tolerance. In contrast, Neutrino's goal is to design a faster and consistent cellular control plane, but Neutrino can be incorporated in consolidated EPC architectures like PEPC. Skycore [40] consolidates RAN and EPC. Skycore is designed for a specific deployment scenario where the base stations are deployed on unmanned aerial vehicles (UAV). Skycore broadcasts user state updates to the neighbor nodes, which as we show in our evaluation (§6) does not lead to a scalable design.

Network function failure recovery: There are proposals on middlebox failure recovery such as [51, 54]. Neutrino's failure recovery is in part inspired by these prior proposals in the general middlebox context. However, the specific failure recovery scheme in Neutrino; per-procedure checkpointing and message logging mechanisms, are designed to satisfy the consistency and latency requirements in the cellular context.

9 CONCLUSION

Next-generation cellular networks aim to support new and emerging applications with ultra-low latency and high reliability requirements. In this work, we identify the key issues in meeting these requirements in existing cellular control plane. We design Neutrino, a new edge-based cellular control plane that provides users an *abstraction of reliable access to cellular services while ensuring lower latency.*[18] We show Neutrino can lead to substantial improvements in the performance of latency-sensitive applications, while tolerating control plane failures. With discussions about 5G core architectures actively under-way, we hope this paper will contribute to an important discourse in control plane designs.

ACKNOWLEDGMENTS

We thank our shepherd Sanjay Rao and the anonymous SIGCOMM reviewers for their valuable feedback.

[18]This work does not raise any ethical issues.

REFERENCES

[1] [n. d.]. ASN.1. https://asn1.io/. ([n. d.]). [Online; accessed 19-Sep-2019].

[2] [n. d.]. Data Plane Development Kit. https://www.dpdk.org/. ([n. d.]). [Online; accessed 19-Sep-2019].

[3] [n. d.]. Fast-CDR. https://github.com/eProsima/Fast-CDR. ([n. d.]). [Online; accessed 19-Sep-2019].

[4] [n. d.]. Geo Hash. http://geohash.gofreerange.com/. ([n. d.]). Online; accessed 19-Sep-2019.

[5] [n. d.]. Lightweight Communications and Marshalling. https://lcm-proj.github.io/. ([n. d.]). [Online; accessed 19-Sep-2019].

[6] [n. d.]. MITM Proxy. https://mitmproxy.org/. ([n. d.]). [Online; accessed 28-Jan-2020].

[7] 3GPP Ref #: 23.401. 2016. *General Packet Radio Service (GPRS) enhancements for Evolved Universal Terrestrial Radio Access Network (E-UTRAN) access.* Retrieved 09/05/2016 from http://www.3gpp.org/DynaReport/23401.htm

[8] 3GPP Ref #: 23.501. 2019. *System architecture for the 5G System (5GS).* Retrieved 20/01/2020 from https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3144

[9] 3GPP Ref #: 24.301. 2016. *Non-Access-Stratum (NAS) protocol.* Retrieved 09/05/2016 from www.3gpp.org/dynareport/24301.htm

[10] 3GPP Ref #: 29.272. 2016. *Mobility Management Entity (MME) and Serving GPRS Support Node (SGSN) related interfaces based on Diameter protocol.* Retrieved 09/05/2016 from www.3gpp.org/DynaReport/29272.htm

[11] 3GPP Ref #: 36.413. 2016. S1 Application Protocol (S1AP). www.3gpp.org/dynareport/36413.htm. (2016).

[12] Xueli An, Fabio Pianese, Indra Widjaja, and Utku Gunay Acer. 2012. DMME: A Distributed LTE Mobility Management Entity. *Bell Labs Technical Journal* 17, 2 (Sept. 2012), 97–120. https://doi.org/10.1002/bltj.21547

[13] J. G. Andrews, S. Buzzi, W. Choi, S. V. Hanly, A. Lozano, A. C. K. Soong, and J. C. Zhang. 2014. What Will 5G Be? *IEEE Journal on Selected Areas in Communications* 32, 6 (June 2014), 1065–1082. https://doi.org/10.1109/JSAC.2014.2328098

[14] Arijit Banerjee, Rajesh Mahindra, Karthik Sundaresan, Sneha Kasera, Kobus Van der Merwe, and Sampath Rangarajan. 2015. Scaling the LTE Control-Plane for Future Mobile Access. In *CoNEXT'15*.

[15] Sebastian Burckhardt. 2014. Principles of Eventual Consistency. (2014). Retrieved 17/06/2020 from http://research.microsoft.com/apps/pubs/default.aspx?id=230852

[16] Sebastian Burckhardt. 2015. Consistency in Distributed Systems, Microsoft Research. In *Springer International Publishing Switzerland'15*.

[17] carla.org. [n. d.]. CARLA. ([n. d.]). Retrieved Jan 27, 2020 from https://github.com/carla-simulator/carla.git

[18] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. 2017. CARLA: An Open Urban Driving Simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*. 1–16.

[19] Ramakrishna Rao Kotla Mahesh Balakrishnan Marcos K Aguilera Douglas B. Terry, Vijayan Prabhakaran and Hussam Abu-Libdeh. 2013. Consistency-based service level agreements for cloud storage. In *SOSP '13*.

[20] Ericsson. 2016. *A vision of the 5G core: flexibility for new business opportunities.* Retrieved 09/05/2016 from https://goo.gl/yRfxkG

[21] ETSI. 2018. GS MEC 002: Multi-access Edge Computing (MEC); Framework and Reference Architecture. https://www.etsi.org/committee/1425-mec. (2018).

[22] ETSI. 2018. GS MEC 002: Multi-access Edge Computing (MEC); Phase 2: Use Cases and Requirements. https://www.etsi.org/committee/1425-mec. (2018).

[23] ETSI. 2018. MEC in 5G networks. https://www.etsi.org/images/files/ETSIWhitePapers/etsi_wp28_mec_in_5G_FINAL.pdf. (2018).

[24] FierceTelecom. 2019. AT&T: Virtualized Flexware service already leverages edge computing. https://www.fiercetelecom.com/telecom/at-t-virtualized-flexware-service-already-leverages-edge-computing. (2019). [Online; accessed 03-April-2019].

[25] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall and Werner Vogels. 2007. Dynamo: Amazon's Highly Available Key-value Store. In *SOSP'07*.

[26] Google. [n. d.]. FlexBuffers. https://google.github.io/flatbuffers/flexbuffers.html. ([n. d.]). [Online; accessed 19-Sep-2019].

[27] Google. [n. d.]. ProtocolBuffers. https://developers.google.com/protocol-buffers/. ([n. d.]). [Online; accessed 19-Sep-2019].

[28] Google/GitHub. [n. d.]. FlatBuffers. https://google.github.io/flatbuffers/. ([n. d.]). [Online; accessed 19-Sep-2019].

[29] Intel. [n. d.]. UPF-EPC. https://github.com/omec-project/upf-epc.git. ([n. d.]). [Online; accessed 27-Jan-2020].

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

[31] Ryan Junguk Cho and Z. Jacobus Van. 2018. MobileStream: A Scalable, Programmable and Evolvable Mobile Core Control Plane Platform. In *MobiCom'18*.

[32] Murad Kablan, Azzam Alsudais, Eric Keller, and Franck Le. 2017. Stateless Network Functions: Breaking the Tight Coupling of State and Processing. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. USENIX Association, Boston, MA, 97–112. https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/kablan

[33] Junaid Khalid and Aditya Akella. 2019. Correctness and Performance for Stateful Chained Network Functions. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. USENIX Association, Boston, MA, 501–516. https://www.usenix.org/conference/nsdi19/presentation/khalid

[34] Raheel Khalid. 2018. Envrmnt by Verizon: Cloud XR Experience on 5G with Mobile Edge Networks. https://www.slideshare.net/AugmentedWorldExpo/raheel-khalid-envrmnt-by-verizon-cloud-xr-experience-on-5g-with-mobile-edge-networks. (2018).

[35] Nour Kouzayha, Mona Jaber, and Zaher Dawy. 2017. Measurement-Based Signaling Management Strategies for Cellular IoT. In *IEEE Internet of Things Journal*.

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

[37] Yuanjie Li, Zengwen Yuan, and Chunyi Peng. 2017. A Control-Plane Perspective on Reducing Data Access Latency in LTE Networks. In *MobiCom'17*.

[38] LightReading. [n. d.]. An Inside Look at Verizon's Edge Computing Capabilities. https://www.lightreading.com/the-edge/an-inside-look-at-verizons-edge-computing-capabilities/d/d-id/749548. ([n. d.]). [Online; accessed 19-Sep-2019].

[39] M-CORD. 2016. Mobile CORD: Enabling 5G on CORD. http://opencord.org/wp-content/uploads/2016/03/M-CORD-March-2016.pdf. (2016). Online; accessed 19-Sep-2019.

[40] Eugene Sampath Mehrdad, Karthikeyan and Z. Morley Mao. 2018. SkyCore: Moving Core to the Edge for Untethered and Reliable UAV-based LTE Network. In *MobiCom'18*.

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

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

[43] Vasudevan Nagendra, Arani Bhattacharya, Anshul Gandhi, and Samir R. Das. 2019. 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)*.

[44] SEVEN Networks. 2016. Operators Urge Action Against Chatty Apps. https://www.seven.com/press_releases/2012/wireless_network_worst_case_scenario.php. (2016). [Online; accessed 09-May-2016].

[45] ng4T. 2016. *Next generation Telecommunication Technology Testing Tools.* Retrieved 09/05/2016 from http://www.ng4t.com/

[46] Binh Nguyen, Tian Zhang, Bozidar Radunovic, Ryan Stutsman, Thomas Karagiannis, Jakub Kocur, and Jacobus Van. 2018. ECHO: A reliable distributed cellular core network for hyper-scale public clouds. In *Mobicom '18*. ACM, New York, NY, USA, 163–178. https://dl.acm.org/doi/10.1145/3241539.3241564

[47] Nokia. 2016. Signaling is growing 50% faster than data traffic. (2016). Retrieved 09/05/2016 from https://www.nokia.com/blog

[48] ONOS. [n. d.]. Introducing ONOS - a SDN network operating system for Service Providers. https://www.opennetworking.org/onos/. ([n. d.]). [Online; accessed 01-Jan-2019].

[49] OpenAirInterface. 2016. OpenAirInterface: A 5G software alliance for democratising wireless innovation . http://www.openairinterface.org/. (2016). [Online; accessed 19-Sep-2019].

[50] Zafar Ayyub Qazi, Melvin Walls, Aurojit Panda, Vyas Sekar, Sylvia Ratnasamy, and Scott Shenker. 2017. A High Performance Packet Core for Next Generation Cellular Networks. In *SIGCOMM'17*.

[51] Shriram Rajagopalan, Dan Williams, and Hani Jamjoom. 2013. Pico Replication: A High Availability Framework for Middleboxes. In *SOCC'13*.

[52] Sylvia Ratnasamy, Brad Karp, Li Yin, Fang Yu, Deborah Estrin, Ramesh Govindan, and Scott Shenker. 2002. GHT: A Geographic Hash Table for Data-Centric Storage. In *ACM WSNA'02*.

[53] Mahadev Satyanarayananz. 2017. The Emergence of Edge Computing. *IEEE Computer Society* (2017).

[54] Justine Sherry, Peter Xiang Gao, Soumya Basu, Aurojit Panda, Arvind Krishnamurthy, Christian Maciocco, Maziar Manesh, João Martins, Sylvia Ratnasamy, Luigi Rizzo, and Scott Shenker. 2015. Rollback-Recovery for Middleboxes. In *SIGCOMM'15*.

[55] Chang-Hong Hsu Matt Skach Md E. Haque1 Lingjia Tang Jason Mars Shih-Chieh Lin, Yunqi Zhang. 2018. The Architectural Implications of Autonomous Driving: Constraints and Acceleration. In *ACM SIGPLAN'18*.

[56] Tarik Taleb and Konstantinos Samdanis. 2011. Ensuring Service Resilience in the EPS: MME Failure Restoration Case. *2011 IEEE Global Telecommunications Conference - GLOBECOM 2011* 36 (2011), 438–456. https://ieeexplore.ieee.org/document/6133654

[57] Tencent. 2018. Intelligent network optimization. https://cloud.tencent.com/product/ino. (2018). https://cloud.tencent.com/product/ino

[58] Maarten van Steen and Andrew S. Tanenbaum. 2017. *Distributed Systems: Principles and Paradigms (3rd edition).* Maarten van Steen.
[59] Paolo Viotti and Marko Vukolić. 2016. Consistency in Non-Transactional Distributed Storage Systems. In *ACM Computing Surveys'16.*
[60] Qiang Xu, Junxian Huang, Zhaoguang Wang, Feng Qian, Alexandre Gerber, and Zhuoqing Morley Mao. 2011. Cellular Data Network Infrastructure Characterization and Implication on Mobile Content Placement. In *SIGMETRICS'11.*
[61] Zengwen Yuan Yuanjie Li and Zengwen Yuan. 2017. A Control-Plane Perspective on Reducing Data Access Latency in LTE Networks. In *MobiCom'17.*
[62] Kyriakos Zarifis, Tobias Flach, Srikanth Nori, David Choffnes, Ramesh Govindan, Ethan Katz-Bassett, Z. Morley Mao, and Matt Welsh. 2014. Diagnosing Path Inflation of Mobile Client Traffic. In *PAM'14.*