TurboEPC: Leveraging Dataplane Programmability to Accelerate the Mobile Packet Core

Rinku Shah, Vikas Kumar, Mythili Vutukuru, Purushottam Kulkarni
Department of Computer Science & Engineering
Indian Institute of Technology Bombay
rinku@cse.iitb.ac.in,vikask@iitb.ac.in,{mythili,puru}@cse.iitb.ac.in

ABSTRACT

Recent architectures of the mobile packet core advocate the separation of the control and dataplane components, with all signaling messages being processed by the control plane entities. This paper presents the design, implementation, and evaluation of TurboEPC, a redesign of the mobile packet core that revisits the division of work between the control and data planes. In TurboEPC, the control plane offloads a small amount of user state to programmable dataplane switches, using which the switches can correctly process a subset of signaling messages within the dataplane itself. The messages that are offloaded to the dataplane in TurboEPC constitute a significant fraction of the total signaling traffic in the packet core, and handling these messages on dataplane switches closer to the end-user improves both control plane processing throughput and latency. We implemented the TurboEPC design using P4-based software and hardware switches. The TurboEPC hardware prototype shows throughput and latency improvements by up to 102× and 98% respectively when the switch hardware stores the state of 65K concurrent users, and 22× and 97% respectively when the switch CPU is busy forwarding dataplane traffic at linerate, over the traditional EPC.

CCS CONCEPTS

• **Networks** → **In-network processing**; **Programmable networks**; **Mobile networks**.

KEYWORDS

LTE-EPC, cellular networks, programmable networks, in-network compute, smartNIC

ACM Reference Format:
Rinku Shah, Vikas Kumar, Mythili Vutukuru, Purushottam Kulkarni. 2020. TurboEPC: Leveraging Dataplane Programmability to Accelerate the Mobile Packet Core. In *Symposium on SDN Research (SOSR '20), March 3, 2020, San Jose, CA, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3373360.3380839

SOSR '20, March 3, 2020, San Jose, CA, USA

ACM ISBN 978-1-4503-7101-8/20/03…$15.00
https://doi.org/10.1145/3373360.3380839

1 INTRODUCTION

Software Defined Networking (SDN) is a network design paradigm that advocates the separation of the control plane of a network element, which makes the decision on how to handle network traffic, from the dataplane that does the actual packet forwarding. With SDN, the more complex control plane functionality can be logically centralized and implemented in agile software controllers, while the dataplane can be implemented in simpler, efficient forwarding switches. This principle of separating the control and dataplane has also been applied to the design of the mobile packet core, and is referred to as *Control and User Plane Separation (CUPS)* [2] by the telecom community. The mobile packet core, e.g., the 4G LTE EPC (Long Term Evolution Evolved Packet Core), consists of networking elements that perform control plane functionality such as authenticating users and setting up data sessions, and dataplane functionality of forwarding user traffic from the wireless radio network to external data networks (§2). With recent releases of 4G (and the new 5G standards) espousing the CUPS principle, the dataplane can move closer to the end users, enabling applications that require low latency data forwarding (e.g., self-driving cars).

EPC procedure	Number of transactions/sec
Attach	9K
Detach	9K
S1 release	300K
Service request	285K
Handover	45K

Network load when total subscribers in the core=1M

Table 1: Sample EPC load statistics [43, 55].

In this paper, we revisit the boundary between the control plane and dataplane in the CUPS-based design of the mobile packet core. Our work is motivated by two observations pertaining to the signaling traffic in the core. First, signaling traffic is growing rapidly [8, 42], fueled by smartphones, IoT devices and other end-user equipment that connect frequently to the network in short bursts. In fact, the signaling load in LTE is 50% higher than that of 2G/3G networks [42]. This high signaling load puts undue pressure on the packet core, making it difficult for operators to meet the signaling traffic SLAs [29]. Second, the signaling procedures can be classified into two types based on their frequency (see Table 1) and nature of processing. A small percentage of the signaling traffic consists of procedures like the *attach* procedure (1–2% of total traffic, as per [43, 55]) that is executed when a user connects to the mobile network for the first time, or the *handover* procedure (~5%) that is executed when the user moves across regions of the mobile network. A significant fraction of the signaling traffic (~63–90%) is

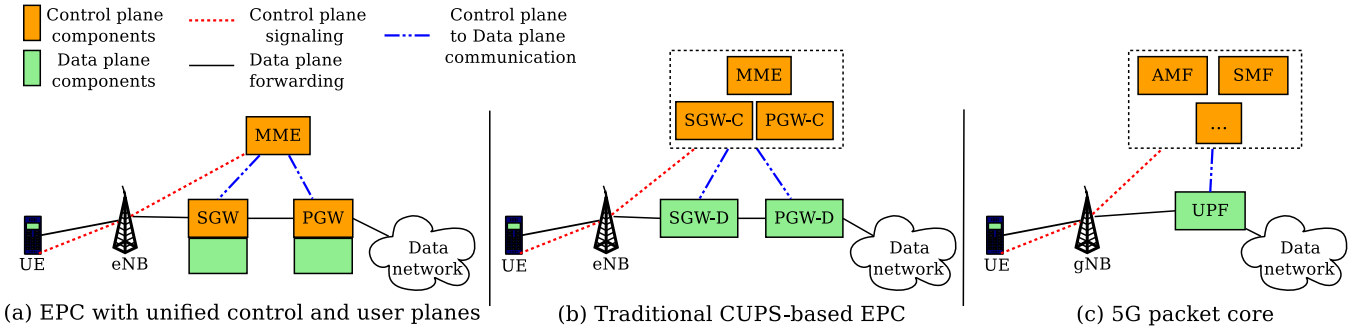


Figure 1: The mobile packet core.

made up of procedures like the *S1 release* that is invoked to release the forwarding state of the user when the user session goes idle for a brief period, and the *service request* procedure that restores the user's forwarding state when the user becomes active again. Further, these two classes of signaling procedures also require access to different types of state during their processing. Attaching a user to the network entails authenticating the user using a network-wide subscriber database, and setting up the forwarding path of the user under mobility requires access to the global network topology. However, frequent signaling procedures like the S1 release and service request access only the *user context* of a single subscriber, and not network-wide global state.

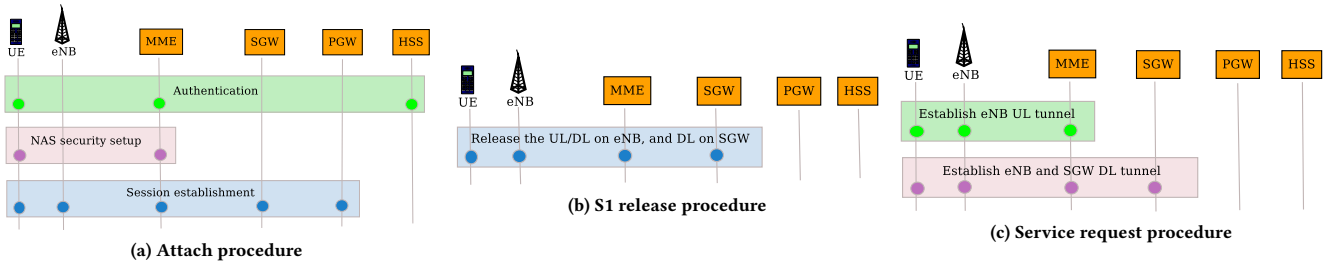
The key idea of our work is that we can improve control plane performance of the mobile packet core by offloading a subset of the control plane procedures like the S1 release and service request from the control plane onto the dataplane switches. Our idea is inspired by recent advances in dataplane technologies, where dataplane switches are evolving from fixed function hardware towards programmable components that can forward traffic at line rate while being highly customizable [41, 53]. Since the S1 release and service request procedures only access and modify user-specific context, the handling of these procedures can be embedded into the packet processing pipeline of programmable dataplane switches, provided the required user context is made available in the switches. Offloading these frequent signaling procedures to the dataplane switches improves both control plane throughput (by utilizing spare switch capacity for handling signaling traffic) and latency (by handling signaling traffic closer to the end user at the switches). We will use the term *offloadable* procedures to describe signaling procedures in the mobile control plane that can be easily offloaded to programmable dataplane switches.

This paper describes TurboEPC (§3), a redesign of the LTE EPC mobile packet core, where offloadable control plane procedures are handled at programmable switches in the dataplane for better control plane performance. TurboEPC modifies the processing of the non-offloadable messages (like the attach procedure) in the control plane in such a way that a copy of the user-specific context information that is generated/modified during such procedures is pushed to dataplane switches closer to the end user. This user context is stored in the switches along with the forwarding state needed for dataplane processing, and is used to process offloadable signaling messages within the dataplane switch itself.

There are several challenges in realizing this idea. First, the control plane state stored in switches may be modified locally by the offloaded signaling messages, causing it to diverge from the "master copy" in the centralized control plane. This inconsistency in state may impact the correctness of the processing of other non-offloadable signaling messages in the control plane, but synchronizing the two copies of the state continuously will erode the performance gains of the offload itself. TurboEPC overcomes this challenge by synchronizing the offloaded control plane state with its master copy *only* when such state is required for the processing of some non-offloadable message, and piggy-backs this state onto the said non-offloadable message itself. Second, dataplane switches have limited memory, and the contexts of millions of active users ([55, 57]) cannot possibly be accommodated within a single switch. To overcome this challenge, TurboEPC partitions user context across multiple switches as per operator policy, thereby increasing the probability that the user context can be stored within the dataplane. Third, switch failures can lead to loss of the latest version of the user context stored in switches, and result in inconsistencies in the control plane state of the user. TurboEPC overcomes this challenge by replicating user context across switches and implements a failover mechanism to tackle switch failures.

We implemented TurboEPC (§4) over a simplified mobile packet core consisting of the ONOS [31] SDN controller in the control plane and P4-based bmv2 software switches [58] in the dataplane. The control and dataplane components communicate using P4Runtime [59]. Our P4-based dataplane switch pipeline was also ported to P4-programmable Netronome Agilio CX smartNICs [53], helping us realize an implementation of TurboEPC on programmable hardware. Evaluation of our prototype (§5) shows that the software prototype of TurboEPC improves LTE EPC control plane throughput by 2.3× and reduces latency by 90% over the traditional CUPS-based EPC design over realistic traffic mixes, by utilizing spare dataplane switch capacity for signaling message processing. Our hardware prototype shows throughput and latency improvements by up to 102× and 98% respectively when the switch hardware stores the state of 65K concurrent users, and 22× and 97% respectively when the switch CPU is busy forwarding dataplane traffic at linerate.

While prior work has proposed several optimizations to the mobile packet core architecture (§6), to the best of our knowledge, we are the first to show that the control plane of mobile data networks



(a) Attach procedure (b) S1 release procedure (c) Service request procedure

Figure 2: LTE EPC procedures.

can be accelerated by offloading signaling procedures on to programmable dataplane switches. Further, while we have evaluated our ideas over the 4G core, we believe that our contributions apply to the future 5G packet core [17] as well, because the design of the 5G dataplane maps closely to that in the CUPS-based 4G EPC. To summarize, the key contributions of this paper are:

- TurboEPC, a redesigned mobile packet core that offloads a significant fraction of signaling procedures from the control plane to programmable dataplanes, thereby improving performance.
- An implementation of TurboEPC over P4-based programmable software/hardware switches, to demonstrate the feasibility of our design.
- A quantification of the performance gains of TurboEPC over the traditional CUPS-based EPC design.

2 BACKGROUND & MOTIVATION

Mobile packet core architecture. The core of a mobile network connects the radio access network, consisting of user equipments (UEs) and the base stations (eNBs), with other packet data networks, including the Internet. Figure 1(a) shows the architecture of the traditional 4G packet core, also called the LTE EPC (Long Term Evolution Evolved Packet Core). The main components of the EPC are the control plane Mobility Management Entity (MME) that handles all signaling procedures, and the dataplane Serving and Packet Gateways (SGW and PGW) that forward user traffic. The SGW and PGW also participate in control plane procedures pertaining to establishing and tearing down user forwarding paths. In order to enable independent scaling of the control and dataplane logic in the S/P-GWs, later releases of 4G LTE espoused the Control and User Plane Separation (CUPS) principle. Figure 1(b) shows the LTE EPC architecture with CUPS; the S/P-GWs are separated into control and dataplane entities, which communicate using a standardized protocol called PFCP (Packet Forwarding Control Protocol [2]). The upcoming 5G standard fully embraces the CUPS principle, as shown in Figure 1(c). In the 5G core, the Access and Mobility Management Function (AMF), Session Management Function (SMF), and other components handle signaling traffic in the control plane, while the User Plane Function (UPF) forwards traffic in the dataplane. The control and dataplane components once again communicate via PFCP. We base our discussion of TurboEPC in the rest of the paper on the CUPS-based EPC architecture shown in Figure 1(b). We assume that the MME and the control plane components of the S/P-GWs are implemented atop an SDN controller, and the dataplane of the S/P-GWs is implemented in SDN switches. Our ideas easily generalize to the 5G architecture, as well as other CUPS-based EPC implementations, e.g., if the control plane components were to be standalone applications.

LTE EPC procedures. Figure 2 briefly illustrates a subset of the LTE EPC control plane procedures When a UE connects to a LTE network for the first time, the initial message sent from the UE via the eNB triggers the *attach* procedure in the core. During this procedure, the UE and the network mutually authenticate each other, by using the user state stored in Home Subscriber Subsystem (HSS), and establish security keys for use during future communication. Finally, the MME sets up the state required to forward user traffic through the core at the SGW and PGW that are on the path from the user to the external packet data network. The *detach* procedure reverses the processing of the attach procedure.

In the dataplane, user data packets are tunneled through the S/P-GWs using the GPRS Tunneling Protocol (GTP). The GTP header consists of Tunnel Endpoint Identifiers (TEIDs) that uniquely identify the path of a user's traffic through the core, and the S/P-GWs in the core network route dataplane traffic based on the TEID values. Separate TEIDs are generated for each of the two links on the datapath (eNB-SGW and SGW-PGW) and for each of the two directions of traffic (uplink and downlink). When a user's IP data packet arrives from the wireless network at the eNB, it is encapsulated into a GTP packet, which is then transmitted over UDP/IP, first between the eNB and the SGW, and then between the SGW and PGW. The egress PGW strips the GTP header before forwarding the user's data to external networks.

If the UE goes idle without sending data for a certain duration (usually 10-30 seconds [21]), a *S1 release* procedure (figure 2(b)) is invoked. During this procedure, the uplink/downlink forwarding rules for the user are deleted from the eNB, downlink forwarding rules are deleted from the SGW, and the connection state of the user changes idle. Later, when the UE becomes active again, the UE initiates a *service request* procedure (figure 2(c)) to restore the forwarding state that was released during the idle period at the dataplane gateways. The user state at the MME also changes back to being actively connected. When a UE moves from one network location to another, it triggers a *handover* procedure in the core. The handover procedure involves, among other things, releasing the user's forwarding context in the old dataplane gateways, and setting up the user's forwarding context along the new path. Note that the core network performs several other procedures beyond those discussed here; however, this description suffices to understand our work.

State	Description	Example	network-wide or local
Security keys	Used for user authentication, authorization, anonymity, confidentiality.	K_{ASME}, CK, IK, AV, K_{NASenc}, K_{NASint}	network-wide
Permanent identifiers	Identifies the user globally	IMSI, MSIN	network-wide
Temporary identifiers	Temporary identity for security	GUTI, TMSI	per-user
IP address	Identifies the user	UE IP address	network-wide
Registration management state	Indicates if the user is registered to the network	ECM-DEREGISTERED, ECM-REGISTERED	network-wide
Connection management state	Indicates if the user is currently idle or connected	ECM-IDLE, ECM-CONNECTED	per-user
User location	Tracks the current user location	Tracking Area(TA), TAI	per-user
Forwarding state	Used for routing data traffic	Tunnel end-point identifiers(TEID)	per-user
Policy/QoS state	Determines policies & QoS values	GBR, MBR	per-user

Table 2: Classification of LTE EPC state.

Message	Security keys	Permanent identifiers	Temporary identifiers	IP address	Registration management state	Connection management state	User location	Forwarding state	Policy /QoS state	Frequency (%)
Attach req	r+w	r	r+w	r+w	r+w	r+w	r+w	r+w	r+w	0.5 – 1
Detach req	—	r	r+w	r+w	r+w	r+w	r+w	r+w	—	0.5 – 1
Service req	—	—	r+w	r	—	r+w	—	r+w	—	30 – 46
S1 release	—	—	r+w	r	—	r+w	—	r+w	—	30 – 46
Handover req	r+w	r	r+w	r+w	r+w	r+w	r+w	r+w	r+w	4 – 5

Table 3: Classification of LTE EPC messages.

Motivation for TurboEPC. Table 2 shows the various components of the per-user state, or *user context*, that is accessed by LTE procedures [16]. One key contribution of our work is to identify parts of the user context that have *network-wide* scope (shaded rows in the table). A piece of user context has network-wide scope if it is derived from, or depends on, network-wide information. For example, the security keys of the user or the IP address are derived from information that is located in the centralized HSS database, and hence have network-wide scope. On the other hand, the connection state of a user (whether connected or idle) is only changed based on local events at the eNB (whether radio link is active or not), and hence has local scope.

Next, Table 3 shows the various user states that are accessed during the processing of each LTE EPC procedure, along with the relative frequencies of each procedure. The shaded columns represent the states with network-wide scope. We see from this table that the S1 release and service request procedures modify only the connection management state (from ECM-CONNECTED to ECM-IDLE and vice versa), forwarding state (GTP tunnel identifiers), and temporary user identifiers, none of which have network-wide scope. Therefore, if we offload this subset of per-user state to dataplane switches closer to the eNB edge, the S1 release and service request procedures can be processed within the dataplane itself, without being forwarded all the way to the centralized controller. *How do we identify which location on the edge to offload this state to?* Note that a given user is only connected to one eNB at a time, and any changes in user location are notified to the core via suitable signaling messages (e.g., handover). Therefore, it is safe to offload some parts of the user context to the edge close to the current eNB, without worrying about concurrent access to this state from other network locations. The offload of the S1 release and the service request procedures to the edge is particularly useful because of the high proportion of these messages in the already high LTE signaling traffic [8, 42, 43, 55]. Further, the latency targets for these signalling messages in future networks [1, 20, 28] is as low as 1ms. Therefore, if we process these high frequency signaling messages

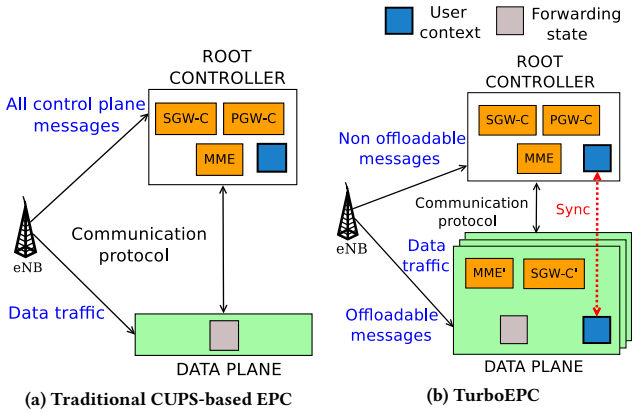


Figure 3: TurboEPC Design.

at the edge closer to the user, we can more easily achieve these stringent latency bounds, and protect the core from high signaling load.

3 TURBOEPC DESIGN

We begin with an overview of TurboEPC's basic design (§3.1) and then describe design features related to scalability (§3.2) and fault tolerance (§3.3).

3.1 Overview

The key idea of TurboEPC is to offload a subset of the user context, and a subset of LTE EPC procedures to the edge, on to dataplane switches closer to the eNB, so that the throughput and latency of processing such messages can be improved. We define an *offloadable* state as that which is accessed/modified by local events at the edge, and is never accessed/modified concurrently from multiple network locations. A particular LTE EPC procedure is offloadable if all the states that are needed to process the message are also offloadable. We will refer to messages that are not offloadable as non-offloadable messages. While this paper applies the concepts of offloading state

	User state (in bytes)	Forwarding state (in bytes)
eNB	0	32
SGW	64	28
PGW	0	19

Table 4: Size of state stored at TurboEPC switches.

and control plane messages to the EPC architecture, our ideas can be generalized to other systems where these definitions apply as well.

Figure 3 compares the CUPS-based traditional EPC design with TurboEPC. In the traditional CUPS-based EPC design (Figure 3(a)), the MME, SGW-C, and PGW-C components are implemented within a *root* SDN controller in the control plane, while the dataplane processing is performed in dataplane switches (SGW-D & PGW-D). The eNB forwards all control plane traffic to the root controller, which processes these messages and installs forwarding state at the S/P-GW switches. All control plane state, including the per-user context, is maintained only in the control plane. In contrast, in the TurboEPC design (shown in Figure 3(b)), the eNB forwards offloadable messages (e.g., S1 release and service request) to the dataplane S/P-GW switches.[1] To enable the processing of offloadable signaling messages in the dataplane, the root controller in TurboEPC pushes a subset of the generated/modified per-user context into the dataplane switches after the completion of every non-offloadable signaling message processing. The user context that is pushed to the dataplane consists of a mapping between the UE identifier and the following subset of information pertaining to the user: the tunnel identifiers (TEIDs) and the UE connection state (connected/idle). This user context is stored in dataplane switch data structures, much like the forwarding state, and consumes an additional ≈64 bytes of memory over and above the ≈32 bytes of forwarding state in our prototype (as shown in Table 4).

Offloadable signaling messages that arrive at the edge dataplane switches (close to the eNB) are processed within the switch dataplane itself, by accessing and modifying the offloaded per-user context. For example, the S1 release request processing requires the TurboEPC switch dataplane to delete the uplink/downlink TEIDs at the eNB and the downlink TEID at the SGW, change the user connection state to idle, and update GUTI if required. Because these offloadable messages reach the switch at least a few tens of seconds (idle timeout) after the context is pushed by the root controller, the state offload does not cause any additional delays while waiting for state to be synchronized. If the signaling message requires a reply to be sent back to the user, the reply is generated and sent by the switch dataplane as well.

Note that, after the user context has been modified by the offloadable signaling messages within the switch data structures, the latest copy of this state resides only in the dataplane. TurboEPC does not synchronize this state back to the root after every modification to the offloaded state, because doing so nullifies the performance gains due to offload in the first place. Instead, TurboEPC lazily synchronizes this state with its master copy at the root controller only when required. That is, all future offloadable messages will access the latest copy of the offloaded state within the dataplane itself, and non-offloadable messages that do not depend on this offloaded state will be directly forwarded to the root by the eNB. However, there are some non-offloadable messages in EPC (e.g., some types of handover messages) that require access to both the latest offloaded user context in the dataplane as well as the non-offloaded state stored in the root. These messages are first sent to the dataplane switches by the eNB, and any processing of the message that can be done without access to global state is performed at the switch itself. Next, the message is forwarded from the dataplane switch to the root controller, with a copy of the modified user context (that is subsequently invalidated at the switch) appended to the packet, in order to correctly complete the rest of the processing at the root.

We acknowledge that TurboEPC introduces a small amount of overhead during the processing of non-offloadable handover messages, since we need to piggyback the user context from the switch to the root controller, as described above. This overhead may be acceptable in current networks, because the handover messages comprise only 4–5% [43, 55] of all signaling traffic. However, the handover traffic can increase for future networks, e.g., with small cells in 5G. We plan to revisit our handover processing to reduce overhead in such usecases as part of our future work.

This basic design of TurboEPC faces two significant challenges, (i) A typical mobile core must handle millions of active connected users [55, 57], while switch memory is usually limited. For example, recent high-end programmable switches have a few tens of MB of memory available to store tables [41], which means that a single switch can only store user context information for a few 100K users in our current design. In fact, the Netronome programmable NIC hardware used in our prototype implementation [54] could only store user context information for 65K users. Therefore, it is unlikely that a single dataplane switch can accommodate the contexts of all users connected to an eNB. (ii) The latest version of the modified user context stored at the switches may be lost in the case of switch failures, making the UE's view and the network's view of the user's context inconsistent. We now describe how TurboEPC overcomes these challenges.

3.2 Partitioning for Scalability

In order to overcome single switch memory limitations, and maximize handling of offloadable messages at the dataplane, TurboEPC relies on multiple programmable switches in the core network. TurboEPC partitions the user context required to handle offloadable messages amongst multiple dataplane switches along the path from the eNB to S/P-GW (possibly including the S/P-GW itself). Further, if the dataplane switches cannot accommodate all user contexts even with partitioning, some subset of the user contexts can be retained in the root controller itself. With this design, any given dataplane switch stores the contexts of only a subset of the users, and handles the offloadable signaling messages pertaining to only those users. The switches over which the partitioning of user context state is done can be connected to each other in one of two ways, as we describe below.

Series design. In the *series design* shown in Figure 4(a), the contexts of a set of users traversing a certain eNB to S/P-GW path in the network are split amongst a series of programmable switches placed along the path. When an offloadable control plane message arrives at one of the switches in the series, it looks up the user context tables to check if the state of the incoming packet's user exists on

[1] We assume that the eNB is capable of analyzing the header of a signaling message to determine if it is offloadable or not.

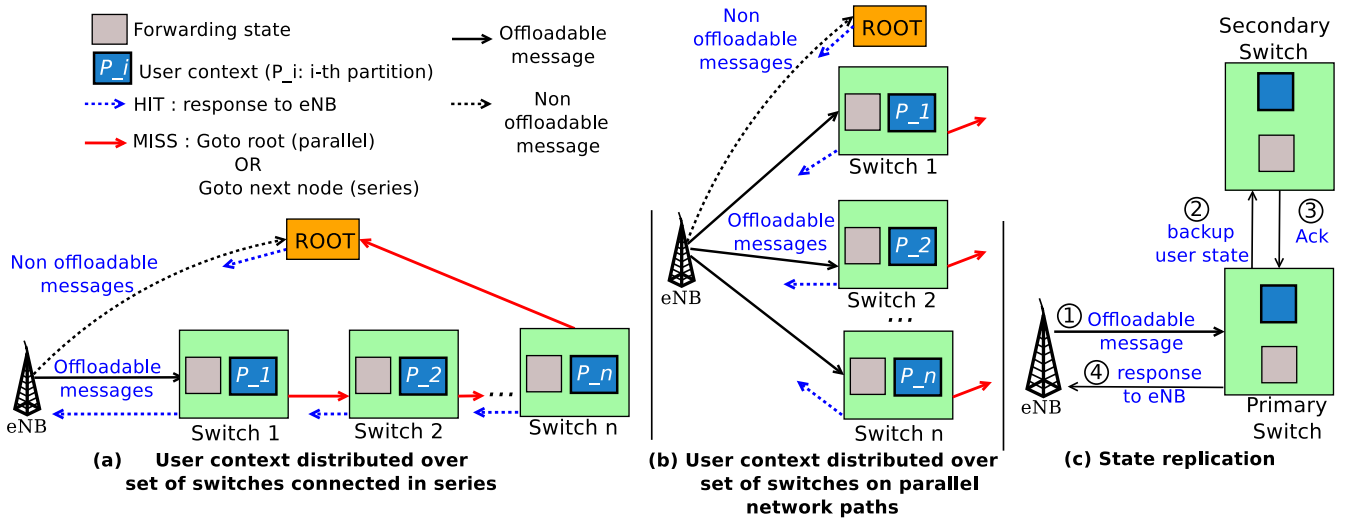


Figure 4: Scalability and fault tolerance in TurboEPC.

the switch. If it exists (a hit), the switch processes the signaling message as discussed in §3.1. If the user context is not found (a miss), the packet is forwarded to the next switch in the series until the last switch is reached. If the user context is not found even at the last switch, the message is forwarded to the root controller, and is processed like in the traditional EPC.

Parallel design. Figure 4(b) depicts a *parallel design*, where the user context is distributed amongst programmable switches located on multiple parallel network paths between the eNB and the S/P-GW in the network. The difference from the series design is that the eNB now needs to maintain information on how the user contexts are partitioned along multiple paths, and must forward offloadable messages of a certain user along the correct path that has the user's state. This entails the extra step of parsing the signaling message header to identify the user, and an additional table lookup to identify the path to send the message on, at the eNB. Offloadable signaling messages that do not find the necessary user context at the switches on any of the parallel paths are forwarded to the root. While the series design leads to simpler forwarding rules at the eNB, the parallel design lends itself well to load balancing across network paths. Note that, while our current implementation supports only the simple series and parallel designs described above, a network could employ a combination of series and parallel designs, where user contexts are partitioned across multiple parallel paths from the eNB to the S/P-GWs, and are further split amongst multiple switches on each parallel path. Across all designs, the root controller installs suitable rules at all switches, to enable forwarding of signaling message towards the switch that can handle it. §5 compares the performance of both designs, and evaluates the impact of partitioning state on TurboEPC performance.

Partitioning user context. The question of how best to partition user contexts across multiple programmable switches in a large network depends upon many factors, including the number of active users, the size of the core network, the capacity of the programmable switches, and the routing and traffic engineering policies employed within the network, and is beyond the scope of this paper. Another interesting question that we defer to future work is deciding which users should be handled at which switches. With the advent of new use cases such as vehicular automation, IoT, smart sensors, and AR/VR in next generation networks, it is becoming important to provide ultra-low latency and ultra-high reliability in processing signaling traffic of some users. Subscribers who require low latency for their frequent signaling requests, but are not highly mobile (e.g., smart sensors), are ideal candidates to offload to the dataplane. It is also conceivable to think that an operator would wish to offload the contexts of premium subscribers. TurboEPC can support any such operator-desired placement policy.

3.3 Replication for Fault Tolerance

In TurboEPC, a subset of the user context is pushed into the dataplane switches during the attach procedure. This context is then modified in the dataplane tables during the processing of subsequent offloadable signaling messages. For example, the S1 release message changes the connection state in the context from connected to idle. In the case of a switch failure, such modifications could be lost, leaving the UE in an inconsistent state. For example, a UE might believe it is idle while a stale copy of the user context at the root controller might indicate that the user is actively connected.

To be resilient to such failure scenarios, TurboEPC stores the user context at one *primary* dataplane switch, and another *secondary* switch. During the processing of non-offloadable messages such as the attach procedure, the root controller pushes the user context to the primary as well as the secondary switch of that user. The root also sets up forwarding paths such that offloadable signaling messages of a user are directed to the primary switch of the user. Upon processing an offloadable message, the primary switch first synchronously replicates the updated user context at the secondary switch, before generating a response to the signaling message back to the user, as shown in Figure 4(c). Our current implementation uses simple synchronous state replication from the primary to one other secondary switch, and is not resilient to failures of both the

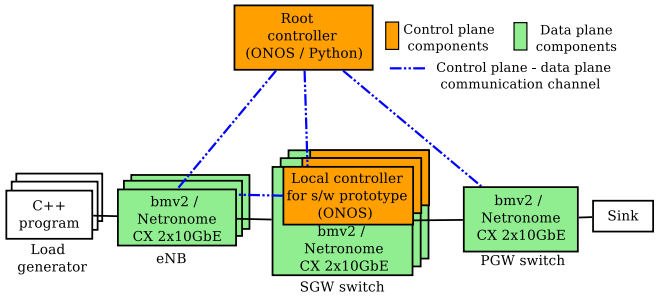


Figure 5: TurboEPC implementation.

primary and secondary switches in quick succession. We plan to evolve our design for replication across multiple secondary switches as part of future work, using techniques from recent research [24].

If a primary switch fails before replication completes, no response is sent to the user, the user will retry the signaling message, and will be redirected to a new switch after the network repairs the failure. If the primary switch fails after successful replication, the SDN controller will be notified of the failure in the normal course of events, e.g., in order to repair network routes, and the TurboEPC application installs forwarding rules to route subsequent offloadable messages of the user to the secondary switch. The root controller also synchronizes itself with the latest copy of user context from the now primary (former secondary) switch, and repopulates this context at another new secondary switch. Users served by the failed switch may see a temporary disruption in offloadable message responses (along with a disruption in dataplane forwarding) during the time of failure recovery and we evaluate the impact of such disruptions in §5.

4 IMPLEMENTATION

We implemented simplified versions of the CUPS-based traditional EPC and TurboEPC in order to evaluate our ideas. We have built our prototype by extending the SDN based EPC implementation available at [47] & [22]. Our implementation supports a basic set of procedures: attach, detach, handover, S1 release, and service request in the control plane, and GTP-based data forwarding. While our implementation of these procedures is based on the 3GPP standards, complete standards compliance was not our goal, and is not critical to our evaluation. The source code of TurboEPC is available at [48].

Figure 5 shows the various components of our implementation. A load generator emulates control and dataplane traffic from multiple UEs to the core, a simplified eNB switch implements only the wired interface to the core, and a sink consumes the traffic generated by the load generator. The load generator is a multi-threaded raw-sockets based program of 5.3K lines, that generates EPC signaling messages and TCP data traffic. The load generator can emulate traffic from a configurable number of concurrent UEs. Further, the emulated traffic mix (i.e., the relative proportions of the various signaling and dataplane messages) is also configurable.

The control plane components of the packet core (MME, SGW-C, PGW-C) are implemented within an SDN controller. The dataplane switches (eNB, SGW-D, PGW-D) are implemented as P4-based packet processing pipelines in approximately 3K lines of P4 code While the dataplane performs only GTP-based forwarding in the traditional CUPS-based EPC prototype, it also performs additional processing of offloadable signaling messages in TurboEPC. We have compiled our TurboEPC P4 code to run on two targets: the bmv2 simple_switch_grpc [58] software switch target, and the Netronome CX 2x10GbE [54] smartNIC hardware target. We now describe these hardware and software switches.

TurboEPC software switch. In the software switch based TurboEPC prototype, the SDN application that forms the EPC control plane is implemented in the ONOS controller [31] in 10K lines of Java code. The offloadable message processing is implemented within a local ONOS controller that is co-located with the P4-based software dataplane switches. This local controller configures and modifies the P4 software switch tables that contain the offloaded state. We use P4Runtime [59] as the communication protocol between the ONOS controller and the P4 software switch. However, the current P4Runtime v1.0.0 does not support multiple controllers (e.g., local and root controllers) configuring the same dataplane switch. Therefore, we built custom support for this feature by modifying the *proto/server* package of the *P4Runtime* [59] to send/receive packets to/from multiple controllers. While our initial implementation simply broadcast control plane messages to all the controllers, this resulted in unnecessary message processing overhead at the controllers. Therefore, we further modified the P4Runtime agent at the bmv2 switch and the ONOS controller to enable a switch to identify the specific controller where the control packet should be forwarded to. This optimization required significant code changes but also improved performance.

TurboEPC hardware switch. Our hardware-based TurboEPC switch did not integrate with the ONOS SDN controller used in the software prototype, due to limitations of the control-dataplane communication mechanisms in the programmable hardware we used. Therefore, we used a separate Python based controller as the root controller in our hardware TurboEPC prototype. Another difference with the software switch is in how offloadable messages are processed. The software prototype stores offloaded user context and forwarding state in switch tables, and the local controller is invoked to modify these tables when processing offloadable messages. However, this local controller can consume the limited switch CPU available in hardware switches. Therefore, the hardware prototype stores offloaded state not in switch tables but in switch register arrays, which are distinct from switch tables. While a switch table can only be modified from the root/local control plane, a register can be modified by P4 code running within the dataplane itself. Therefore, we modified our design so that the switch tables only store a pointer from the user identifier to this register state, and not the actual state itself. The root controller takes care of maintaining the free and used slots in the register arrays of the switches, and creates the table entries that map from user identifiers (which are either available in packet headers, or can be derived from the packet headers) to register array indices when the user context is first created during the attach procedure. After the entries are created, offloadable messages that change the offloaded state do not require to invoke the switch control plane (that consumes switch CPU) to modify the tables, but can fetch the register index from the table and directly modify the registers from within the dataplane itself.

TurboEPC packet processing pipeline. We now briefly describe the P4-based packet processing pipeline in the TurboEPC dataplane switches (Figure 6). Incoming packets in an EPC switch are first run through a *message redirection table* that matches on various header fields to identify if the incoming message is a signaling message, and if yes, where it should be forwarded to. This table is populated by the root controller to enable correct redirection of non-offloadable signaling messages to the root, and offloadable messages to the switch that has the particular user's context. Packets that do not match the message redirection table continue along the pipeline, and are matched through multiple *GTP forwarding tables* for GTP-based dataplane forwarding.

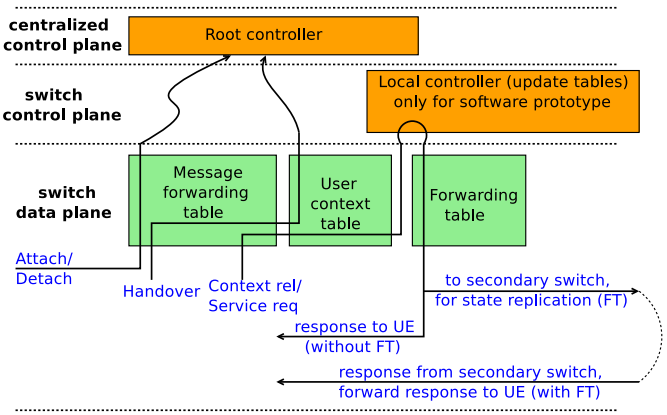


Figure 6: Packet processing pipeline in TurboEPC.

Offloadable signaling messages destined to the current switch are first run through the *user context table* to find any existing offloaded user context. The signaling message is processed by modifying or deleting the user context and/or GTP forwarding state stored on the switch. The switch data structures are either updated by the local controller (software prototype) or within the dataplane itself (hardware prototype). After message processing, the packet may be forwarded to the secondary switch for state replication. On successful replication (within the dataplane), the secondary switch generates the response packet for the user, and forwards it to the primary switch as an acknowledgement for successful state replication. The primary switch dataplane forwards the response packet back to the user, indicating successful execution of the signaling message. If the signaling message processing could not complete at the switch (e.g., user context is not found, or the handover message requires further processing at the root), the packet is forwarded to the root controller for further processing. In the case of series design (not last switch), if user context is not found, the message is forwarded to the next switch on the path.

5 EVALUATION

We now evaluate the TurboEPC software and hardware switch prototypes, and quantify the performance gains over the traditional CUPS-based EPC.

Traffic Mix	Attach, Detach %	S1 release, Service request %	Handover %
Att-1	1	99	0
Att-5	5	95	0
Att-10	10	90	0
Att-50	50	50	0
HO-5	10	85	5
Typical [43]	1–2	63–94	5

Table 5: LTE-EPC traffic mix used for experiments.

5.1 TurboEPC software prototype

We first evaluate the TurboEPC prototype implemented on P4-based software switches. We primarily aim to evaluate the benefits of our TurboEPC design as compared to the traditional EPC design. Further, we also seek to demonstrate the correctness and efficacy of the various mechanisms for scalability and fault tolerance in our design.

Setup. The components in our evaluation setup include the load generator, a sink node, ONOS v1.13 SDN controller, and multiple P4-based programmable bmv2 software switches (simple_switch_grpc) for the eNB, SGW, and PGW components of LTE EPC. We use multiple "forwarding chains" of load generators and switches in the dataplane, to generate enough load to saturate the root SDN controller. All components run on Ubuntu 16.04 hosted over separate LXC containers to ensure isolation. The root controller container is hosted on an Intel Xeon E5-2697@2.6GHz (24GB RAM) server, and the rest are hosted on an Intel Xeon E5-2670@2.3GHz (64GB RAM) server. The root/local controllers, and all P4 software switches are allocated 1 CPU core and 4GB RAM each. Our load generator is a closed loop load generator which emulates multiple concurrent UEs generating signaling and dataplane traffic. The number of concurrent emulated UEs in our load generator is tuned to saturate the control plane capacity (root or local or both) of the system in all experiments, and is varied between 4 and 100.

Parameters and metrics. We generate different workload scenarios by varying the mix of offloadable (S1 release and service request) and non-offloadable (attach, detach, and handover) signaling messages in the control plane traffic generated by the load generator. Table 5 shows the relative proportions of the various signaling messages in the traffic mixes used, along with a typical traffic mix found in real user traffic [43]. All results reported are averaged over three runs of an experiment conducted for 300 seconds, unless mentioned otherwise. The performance metrics measured are the average control plane throughput (number of control plane messages processed/sec) and average response latency of control plane requests, as measured at the load generator over the duration of the experiment.

TurboEPC vs. Traditional EPC. We first quantify the performance gains of the basic TurboEPC design as compared to the traditional EPC design. In these set of experiments, we assume (and ensure) that all UE context state fits in the memory of a single switch. We also do not perform any replication of dataplane state for fault tolerance. As we are interested in measuring maximum control plane capacity, our load generator does not generate any

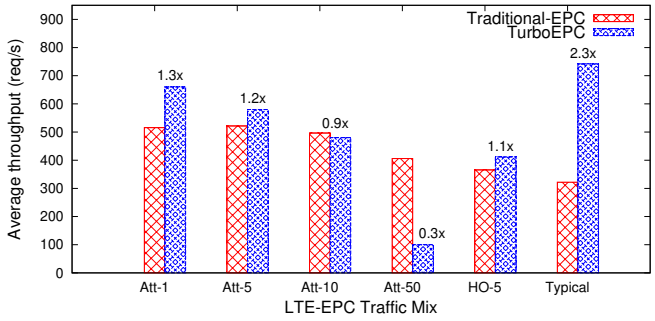


Figure 7: TurboEPC vs. traditional EPC: Throughput.

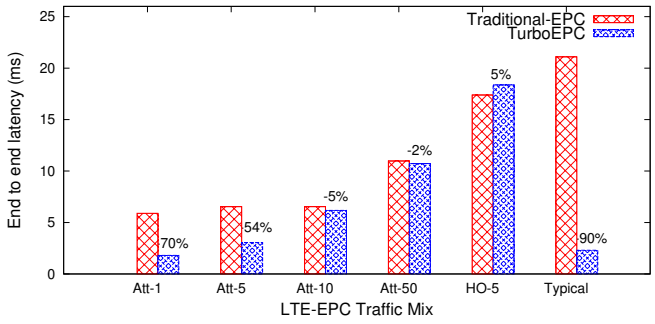


Figure 8: TurboEPC vs. traditional EPC: Latency.

dataplane traffic. Figures 7 and 8 show the control plane throughput and latency respectively of the traditional EPC and TurboEPC, for various traffic mixes of Table 5. As can be seen, performance gains of TurboEPC over traditional EPC are higher for traffic mixes with a greater fraction of offloadable messages. For example, for the typical traffic mix, we observe that TurboEPC improves control plane throughput by 2.3× over traditional EPC, while control plane latency is reduced by 90%. Further, we note that the root controller was fully saturated in the traditional EPC experiments, while CPU utilization was under 20% with TurboEPC, because most signaling traffic was processed using dataplane switch CPU. However, when the traffic consists of a high proportion of non-offloadable messages (e.g., mix Att-50, which is unrealistic), TurboEPC has slightly lower throughput than traditional EPC, because it incurs an additional overhead of pushing user context to the dataplane switches during the processing of non-offloadable messages. In summary, we expect TurboEPC to deliver significant performance gains over the traditional EPC over realistic traffic mixes which contain a high proportion of offloadable signaling messages.

The performance gains of TurboEPC are more pronounced when the distance between the "edge" and "core" of the network increases, and with increasing number of switches that can process offloadable messages in the dataplane, both of which are likely in real-life settings. Figures 9 and 10 show the performance of TurboEPC as a function of the distance to the root controller (emulated by adding delay to all communications to the root) and the number of forwarding chains of dataplane switches. We see from the figures that TurboEPC with 4 chains provides 4× – 5× throughput over traditional EPC. We also observe that TurboEPC latency does not

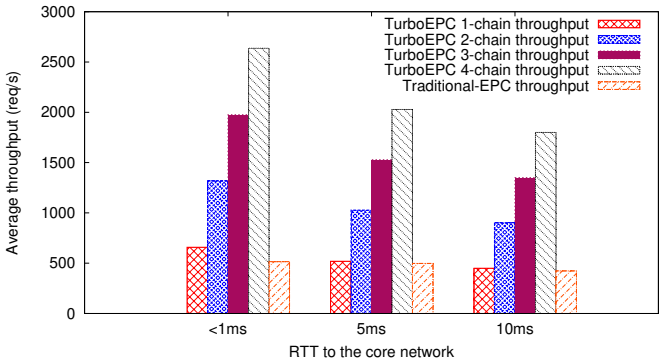


Figure 9: Throughput with varying distance to core, and varying number of dataplane switches.

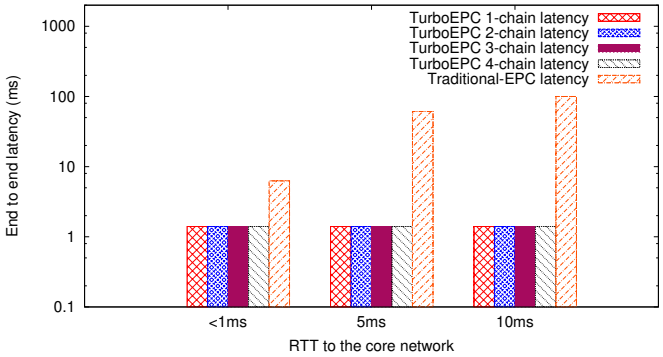


Figure 10: Latency with varying distance to core, and varying number of dataplane switches.

increase with the distance to the core network, and the latency is reduced by two orders of magnitude compared to traditional EPC when the round trip latency to the core is greater than 5ms.

Design	Attach, Detach	S1 release, Service request	Handover
RTT to the core is less than 1*ms*			
Centralized	10.72	10.28	17.38
TurboEPC	10.98	1.44	18.36
RTT to the core is 10*ms*			
Centralized	200	38	549
TurboEPC	205	2.4	580

Table 6: Average end-to-end latency for LTE-EPC (in *ms*).

While TurboEPC improves average control plane performance, it can (and does) degrade performance for some specific non-offloadable messages. For example, as discussed in §3.1, processing non-offloadable messages like the attach request incurs the extra cost of pushing offloaded user context to dataplane switches. Similarly, handover message processing incurs a higher overhead with TurboEPC because we need to piggyback the offloaded state and synchronize it with the root. Table 6 shows the average processing latency of various individual signaling messages in TurboEPC and the traditional EPC, in the setup with a single forwarding chain. The generated

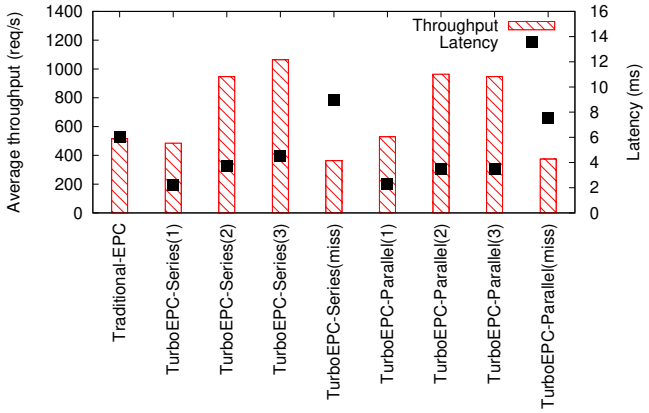


Figure 11: Series vs. parallel partitioning.

load followed the typical traffic distribution as shown in Table 5. Table 6 shows the latency results for two scenarios: (i) when the EPC core is close to the edge ($RTT < 1ms$), and (ii) when the EPC core is far from the edge ($RTT = 10ms$). We see that the processing latency reduces by up to 86–94% for offloadable messages like S1 release and service request, but increases by 2–5% for non-offloadable messages like attach requests and handovers. Because offloadable messages form a significant fraction of signaling traffic, TurboEPC improves the overall control plane performance of the mobile packet core, even though a small fraction of signaling messages may see slightly degraded performance.

Series vs. parallel partitioning. Next, we perform experiments with the series vs. parallel state partitioning design variants of the TurboEPC software switch prototypes, to evaluate the performance impact of the additional complexity of these designs. This experiment was performed with traffic mix Att-1 of Table 5 (1% attach-detach requests), and results for other traffic mixes were similar. We use multiple (up to 3) TurboEPC switches in series and parallel configurations, and partition 100 active users uniformly over these switches. Besides these 100 users, our load generator also generates traffic on behalf of an additional 20 users whose contexts were not stored in the dataplane switches, to emulate the scenario where all contexts cannot be accommodated in the dataplane. Figure 11 shows the average control plane throughput and latency of the TurboEPC-Series(n) and TurboEPC-Parallel(n) designs, for varying number of switches n in series and parallel, both when the context of the users is found within one of the switches (hit) and when it is not (miss). We see from the figure that the TurboEPC throughput scales well when an additional switch becomes available to process offloadable signaling messages. The scaling is imperfect when there are 3 switches in series or parallel, because the eNB switch became the bottleneck in these scenarios. This eNB bottleneck is more pronounced in the case of the parallel design, because the eNB does extra work to lookup the switch that has the user's context in the parallel design. We hope to tune the eNB software switch to ameliorate this bottleneck in the future.

While the throughput increases with extra TurboEPC switches, the control plane latency also increases due to extra hop traversals and extra table lookups, as compared to the basic TurboEPC design. This impact on latency is more pronounced in the series designs,

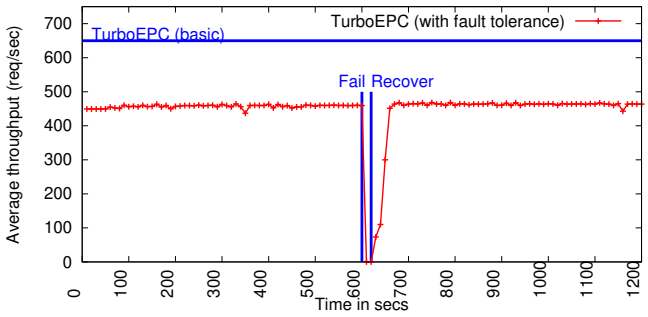


Figure 12: TurboEPC throughput during failover.

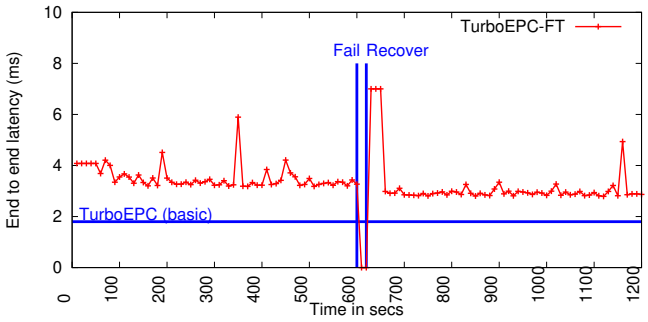


Figure 13: TurboEPC latency during failover.

where each switch adds an extra hop to latency. However, even with 3 switches in series or parallel, TurboEPC latency is still lower than that of the traditional EPC. We also see from the figure that the miss latency of offloadable message processing is worse than the message processing latency of the traditional EPC, because the messages undergo multiple table lookups within the dataplane before eventually ending up at the root controller.

TurboEPC fault tolerance. Next, we evaluate the fault tolerance of the TurboEPC design, by simulating a failure of the primary switch in the middle of an experiment and observing the recovery. Figure 12 shows the average throughput and Figure 13 shows the average latency of the fault-tolerant TurboEPC for an experiment of duration 1200 seconds, where the primary switch was triggered to fail after 600 seconds. Also shown in the graphs are the throughput and latency values of the basic TurboEPC without any fault tolerance for reference. We see that the throughput of the basic TurboEPC is 40% higher and the latency is 33% lower than the fault tolerant design due to lack of the overhead of replication. After the failure of the primary switch, we found that the root controller takes about 15 seconds to detect the primary switch failure, ~2 ms to push rules to eNB that would route incoming packets to the secondary switch, and ~30 ms to restart offloadable signaling message processing at the secondary switch. During this recovery period, we observed ~200 signaling message retransmissions, but all signaling messages were eventually correctly handled by TurboEPC after the failure.

5.2 TurboEPC hardware prototype

We now evaluate our hardware-based TurboEPC prototype, built using the P4-programmable Netronome Agilio smartNIC [54].

Setup. The TurboEPC hardware setup was hosted on three Intel Xeon E5-2670@2.3GHz (128GB RAM) servers, each connected to one Netronome Agilio CX 2x10GbE smartNIC. The three servers hosted the single chain of the load generator+eNB, SGW, and PGW+sink respectively. A python based controller is hosted on the SGW switch, and served as the root control plane.

Parameters and Metrics. Our load generator generated a mix of offloadable/non-offloadable signaling messages and dataplane traffic (using iperf3) in the experiments. The smartNIC hardware could accommodate the user contexts of 65K users within the switch hardware tables, and the load generator generated traffic for up to 65K users in all experiments. The maximum forwarding capacity of our smartNICs (without any TurboEPC changes) was measured at 8 Gbps, so our load generator also limited its maximum dataplane traffic rate to 8 Gbps. All experiments were run for 300 seconds, and we report the maximum throughput and latency of processing offloadable signaling messages in the hardware prototype.

Capacity of TurboEPC hardware switch. First, we measure the maximum control plane capacity of our hardware TurboEPC switch, without any interfering dataplane traffic. Figure 14 shows the offloadable message processing throughput and latency of a single TurboEPC hardware switch. We evaluate the maximum throughput with the smartNIC loaded with user state size varying from 100 to 65K. We found that the throughput does not vary when we add state of more users to the smartNIC. Also shown are the throughput and latency numbers for the traditional CUPS-based EPC (RTT to the root < 1ms) for reference. We see from the table that our TurboEPC hardware switch can successfully serve upto 65K users, while providing 102× higher throughput and 98% lower latency than traditional EPC.

Performance with dataplane traffic. TurboEPC improves control plane throughput over the traditional EPC by leveraging the extra capacity at dataplane switches for offloadable signaling message processing. However, the performance gains of TurboEPC may be lower if the switch is busy forwarding dataplane traffic. We now measure the impact of this dataplane cross traffic on the control plane throughput of TurboEPC. We pump increasing amounts of dataplane traffic through our TurboEPC hardware switch (with state for 65K users) and measure the maximum rate at which the switch can process offloadable signaling messages while forwarding data traffic simultaneously. Figure 15 show the signaling message throughput and latency respectively, as a function of the dataplane traffic forwarded by the TurboEPC hardware dataplane switch. We see from the figure that as the data traffic rate increases, the offloadable signaling message throughput decreases, and response latency varies between 100μs to 180μs. The throughput and latency of the traditional EPC (RTT to the root < 1ms) is also shown for reference in the figures. We observe that when the switch is idle, the hardware based TurboEPC throughput is 102× higher, and the latency 98% lower, as compared to the traditional EPC. However, even when the switch is forwarding data at line rate (8Gbps), we observe throughput to be 22× higher and latency 97% lower than traditional EPC, confirming our intuition that spare switch CPU can be used for handling offloaded signaling traffic. As part of future work, we plan to explore an adaptive offload design that offloads signaling message processing to the dataplane only when switches have spare processing capacity, an idea we have explored in our prior work [49].

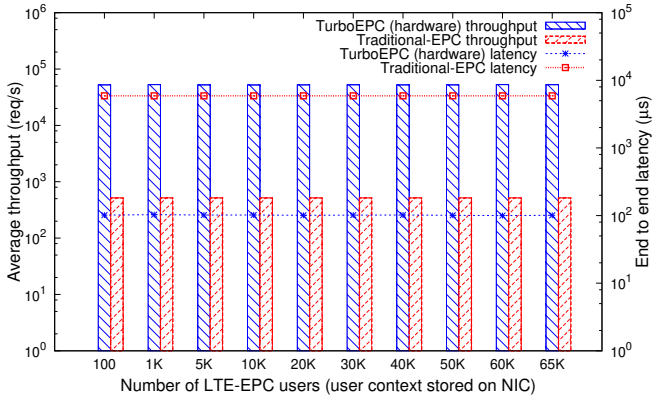


Figure 14: TurboEPC throughput vs. Number of Users

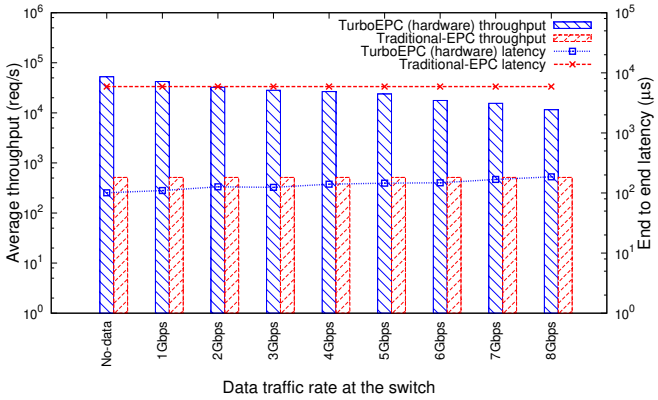


Figure 15: TurboEPC throughput with data traffic interference.

6 RELATED WORK

Optimizations to the Packet Core. Prior work has several proposals that redesign the mobile packet core to achieve a diverse set of goals. Softcell [23] proposes the solution to accelerate the 4G dataplane forwarding via offload of the packet route installation task to the edge switch. They further minimize the forwarding table size by aggregating the flow rules within the switch. While this work is primarily focused on optimizing the dataplane processing, TurboEPC accelerates the control plane via offload of signaling message processing to the edge switch. CleanG [37], PEPC [45], SCALE [5], DMME [4], MMLite [39], MobileStream [10], DPCM [33], and other similar proposals [34, 44, 46] optimize 4G/5G control plane processing, much like TurboEPC. CleanG [37] and PEPC [45] refactor the EPC control plane processing, to reduce the overhead of state transfer across components. SCALE [5] proposes a distributed design of the control plane, and horizontally scales the EPC control plane by distributing signaling load across multiple replicas. MMLite [39] proposes a stateless scalable MME design by storing the user specific state in shared memory. MobileStream [10] decomposes the traditionally monolithic control plane components and proposes the use of a streaming framework for scalability. DPCM [33] modifies

the EPC protocol by reducing the number of messages exchanged and by starting dataplane forwarding before completion of control plane processing. While these proposals advocate optimized architectures of the EPC control plane, none of them revisit the boundary between the EPC control and dataplanes. On the other hand, TurboEPC revisits the split of functionality between the control plane software and dataplane switches, and proposes a refactoring of the mobile core with the goal of offloading a subset of control plane processing to programmable dataplane switches closer to the end user. Therefore, this body of work is orthogonal and complementary to our work, and TurboEPC can leverage these control plane optimizations for the processing of non-offloadable messages at the root controller.

Programmable Dataplanes. While the first wave of SDN research decoupled the control plane from the dataplane and made the control plane highly programmable, the second wave of SDN research has made even the dataplanes highly programmable, realizing the true vision of software defined networking. Today, dataplanes can be customized using P4 [6], a programming language to define packet processing pipelines. These software-defined dataplanes can then be compiled to run on diverse targets, e.g., software switches [50, 58], hardware programmable switches guaranteed to work at line rate [7, 11, 41, 51], FPGAs [60], and smart programmable NICs [54]. Further, these programmable dataplanes can be configured from software SDN controllers using standard protocols [35, 59]. Programmable dataplanes have enabled a variety of new applications within the dataplane, e.g., in-band network telemetry (INT) [27], traffic engineering [52], load balancing [12, 36], consensus [14, 15], traffic monitoring [40], key-value stores [25, 32], congestion control [26], and GTP header processing [3, 9]. Molero.E [38] demonstrate the possibility of accelerating the control plane functions like failure detection/notification via offload to programmable dataplanes. TurboEPC takes this line of work one step further, and proposes the offload of frequent and simple signaling procedures to programmable switches.

Control Plane Scalability. With the SDN paradigm, a logically centralized control plane can potentially become a performance bottleneck and prior work has identified two broad approaches to solve this control plane scalability challenge. Some SDN controllers [30, 56, 61] use the technique of *horizontal scaling*, where the incoming control plane traffic is distributed amongst multiple homogeneous SDN controllers, which cooperate to maintain a consistent view of the common global network wide state amongst themselves using standard consensus protocols. In contrast, other SDN controllers [13, 18, 19, 49, 62] use *hierarchical scaling* to offload control plane functionality to lower levels of "local" SDN controllers that perform different functions. Our work is inspired by hierarchical SDN controllers but is quite different from them—we apply the idea of offloading computation from SDN controllers to dataplane switches in the CUPS-based mobile packet core.

7 CONCLUSION

This paper described TurboEPC, a mobile packet core design where a subset of signaling messages are offloaded to programmable dataplane switches in order to improve control plane performance. TurboEPC dataplane switches store a small amount of control plane state in switch tables, and use this state to process some of the more frequent signaling messages at switches closer to the edge. We implemented TurboEPC on P4-based software switches and programmable hardware, and demonstrated that offloading signaling messages to the dataplane significantly improves control plane throughput and latency.

ACKNOWLEDGEMENTS

We thank our shepherd Sonia Fahmy and the anonymous reviewers for their insightful feedback.

REFERENCES

[1] 3GPP. 2017. *5G 3GPP specifications.* https://www.3gpp.org/ftp/Specs/archive/23_series/23.502/
[2] 3GPP. 2017. *Control and User Plane Separation.* http://www.3gpp.org/cups
[3] Ashkan Aghdai et al. 2018. Transparent Edge Gateway for Mobile Networks. In *IEEE 26th International Conference on Network Protocols (ICNP).*
[4] X. An, F. Pianese, I. Widjaja, and U. G. Acer. 2012. DMME: A distributed LTE mobility management entity. *Bell Labs Technical Journal* 17, 2 (2012), 97–120.
[5] 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 *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies.*
[6] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, and David Walker. 2014. P4: Programming Protocol-independent Packet Processors. *SIGCOMM Computer Communication Review* 44 (2014).
[7] Pat Bosshart, Glen Gibb, Hun-Seok Kim, George Varghese, Nick McKeown, Martin Izzard, Fernando Mujica, and Mark Horowitz. 2013. Forwarding Metamorphosis: Fast Programmable Match-action Processing in Hardware for SDN. In *Proceedings of the ACM SIGCOMM Conference.*
[8] Gabriel Brown. 2012. *On Signalling Storm.* Retrieved November 10, 2018 from https://blog.3g4g.co.uk/2012/06/on-signalling-storm-ltews.html
[9] Carmelo Cascone and Uyen Chau. 2018. Offloading VNFs to programmable switches using P4. In *ONS North America.*
[10] Junguk Cho, Ryan Stutsman, and Jacobus Van der Merwe. 2018. MobileStream: A Scalable, Programmable and Evolvable Mobile Core Control Plane Platform. In *Proceedings of the 14th International Conference on Emerging Networking EXperiments and Technologies.*
[11] Sharad Chole, Andy Fingerhut, Sha Ma, Anirudh Sivaraman, Shay Vargaftik, Alon Berger, Gal Mendelson, Mohammad Alizadeh, Shang-Tse Chuang, Isaac Keslassy, Ariel Orda, and Tom Edsall. 2017. dRMT: Disaggregated Programmable Switching. In *Proceedings of the ACM SIGCOMM Conference.*
[12] Eyal Cidon, Sean Choi, Sachin Katti, and Nick McKeown. 2017. AppSwitch: Application-layer Load Balancing Within a Software Switch. In *Proceedings of the APNet.*
[13] Andrew R. Curtis et al. 2011. DevoFlow: Scaling Flow Management for High-performance Networks. In *Proceedings of the ACM SIGCOMM.*
[14] Huynh Tu Dang et al. 2018. Consensus for Non-Volatile Main Memory. In *IEEE 26th International Conference on Network Protocols (ICNP).*
[15] Huynh Tu Dang, Daniele Sciascia, Marco Canini, Fernando Pedone, and Robert Soule. 2015. NetPaxos: Consensus at Network Speed. In *Proceedings of the the ACM SIGCOMM SoSR.*
[16] ETSI. 2017. *The Evolved Packet Core.* http://www.3gpp.org/technologies/keywords-acronyms/100-the-evolved-packet-core
[17] ETSI. 2018. *5G standards specification (23.501).* https://www.etsi.org/deliver/etsi_ts/123500_123599/123501/15.02.00_60/ts_123501v150200p.pdf
[18] Luyuan Fang, Fabio Chiussi, Deepak Bansal, Vijay Gill, Tony Lin, Jeff Cox, and Gary Ratterree. 2015. Hierarchical SDN for the hyper-scale, hyper-elastic data center and cloud. In *Proceedings of the SoSR.*
[19] Soheil Hassas Yeganeh and Yashar Ganjali. 2012. Kandoo: A Framework for Efficient and Scalable Offloading of Control Applications. In *Proceedings of the HotSDN.*
[20] R. E. Hattachi. 2015. *Next Generation Mobile Networks, NGMN.* https://www.ngmn.org/fileadmin/ngmn/content/downloads/Technical/2015/NGMN_5G_White_Paper_V1_0.pdf
[21] Open Air Interface. 2016. *EPC: S1 release.* https://gitlab.eurecom.fr/oai/openairinterface5g/issues/16
[22] Aman Jain, Sunny Lohani, and Mythili Vutukuru. 2016. *Opensource SDN LTE EPC.* https://github.com/networkedsystemsIITB/SDN_LTE_EPC
[23] Xin Jin, Li Erran Li, Laurent Vanbever, and Jennifer Rexford. 2013. SoftCell: Scalable and Flexible Cellular Core Network Architecture. In *Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies.*

[24] Xin Jin, Xiaozhou Li, Haoyu Zhang, Nate Foster, Jeongkeun Lee, Robert Soule, Changhoon Kim, and Ion Stoica. 2018. NetChain: Scale-Free Sub-RTT Coordination. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*.

[25] Xin Jin, Xiaozhou Li, Haoyu Zhang, Robert Soule, Jeongkeun Lee, Nate Foster, Changhoon Kim, and Ion Stoica. 2017. NetCache: Balancing Key-Value Stores with Fast In-Network Caching. In *Proceedings of the SOSP*.

[26] Naga Katta, Mukesh Hira, Changhoon Kim, Anirudh Sivaraman, and Jennifer Rexford. 2016. HULA: Scalable Load Balancing Using Programmable Data Planes. In *Proceedings of the the SoSR*.

[27] Changhoon Kim, Anirudh Sivaraman, Naga Katta, Antonin Bas, Advait Dixit, and Lawrence J Wobker. 2015. In-band network telemetry via programmable dataplanes. In *ACM SIGCOMM*.

[28] Dr. Kim. 2017. *5G stats.* https://techneconomyblog.com/tag/economics/

[29] P. Kiss, A. Reale, C. J. Ferrari, and Z. Istenes. 2018. Deployment of IoT applications on 5G edge. In *IEEE International Conference on Future IoT Technologies*.

[30] Teemu Koponen et al. 2010. Onix: A Distributed Control Platform for Large-scale Production Networks. In *Proceedings of the OSDI*.

[31] Open Networking Lab. 2017. *ONOS SDN controller.* https://github.com/opennetworkinglab/onos

[32] Bojie Li, Zhenyuan Ruan, Wencong Xiao, Yuanwei Lu, Yongqiang Xiong, Andrew Putnam, Enhong Chen, and Lintao Zhang. 2017. KV-Direct: High-Performance In-Memory Key-Value Store with Programmable NIC. In *Proceedings of the SOSP*.

[33] Yuanjie Li, Zengwen Yuan, and Chunyi Peng. 2017. 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*.

[34] Heikki Lindholm et al. 2015. State Space Analysis to Refactor the Mobile Core. In *Proceedings of the AllThingsCellular*.

[35] Nick McKeown et al. 2008. OpenFlow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review* 38, 2 (2008).

[36] Rui Miao, Hongyi Zeng, Changhoon Kim, Jeongkeun Lee, and Minlan Yu. 2017. SilkRoad: Making Stateful Layer-4 Load Balancing Fast and Cheap Using Switching ASICs. In *Proceedings of the the ACM SIGCOMM Conference*.

[37] Ali Mohammadkhan, KK Ramakrishnan, Ashok Sunder Rajan, and Christian Maciocco. 2016. CleanG: A Clean-Slate EPC Architecture and ControlPlane Protocol for Next Generation Cellular Networks. In *Proceedings of the 2016 ACM Workshop on Cloud-Assisted Networking*.

[38] Edgar Costa Molero, Stefano Vissicchio, and Laurent Vanbever. 2018. Hardware-Accelerated Network Control Planes. In *Proceedings of the 17th ACM Workshop on Hot Topics in Networks (HotNets)*.

[39] 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*.

[40] Srinivas Narayana, Anirudh Sivaraman, Vikram Nathan, Prateesh Goyal, Venkat Arun, Mohammad Alizadeh, Vimalkumar Jeyakumar, and Changhoon Kim. 2017. Language-Directed Hardware Design for Network Performance Monitoring. In *Proceedings of the the ACM SIGCOMM Conference*.

[41] Barefoot networks. 2018. *NoviWare 400.5 for Barefoot Tofino chipset.* https://noviflow.com/wp-content/uploads/NoviWare-Tofino-Datasheet.pdf

[42] Nokia Siemens Networks. 2012. *Signaling is growing 50% faster than data traffic.* https://docplayer.net/6278117-Signaling-is-growing-50-faster-than-data-traffic.html

[43] David Nowoswiat. 2013. *Managing LTE Core Network Signaling Traffic.* https://www.nokia.com/en_int/blog/managing-lte-core-network-signaling-traffic

[44] M. Pozza, A. Rao, A. Bujari, H. Flinck, C. E. Palazzi, and S. Tarkoma. 2017. A refactoring approach for optimizing mobile networks. In *2017 IEEE International Conference on Communications (ICC)*.

[45] 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 *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*.

[46] M. T. Raza, D. Kim, K. Kim, S. Lu, and M. Gerla. 2017. Rethinking LTE network functions virtualization. In *IEEE 25th International Conference on Network Protocols (ICNP)*.

[47] Rinku Shah. 2018. *Cuttlefish open source project.* https://github.com/networkedsystemsIITB/cuttlefish

[48] Rinku Shah, Vikas Kumar, Mythili Vutukuru, and Purushottam Kulkarni. 2015. *TurboEPC github code.* https://github.com/rinku-shah/turboepc

[49] Rinku Shah, Mythili Vutukuru, and Purushottam Kulkarni. 2018. Cuttlefish: Hierarchical SDN Controllers with Adaptive Offload. In *IEEE 26th International Conference on Network Protocols (ICNP)*.

[50] Muhammad Shahbaz, Sean Choi, Ben Pfaff, Changhoon Kim, Nick Feamster, Nick McKeown, and Jennifer Rexford. 2016. PISCES: A Programmable, Protocol-Independent Software Switch. In *Proceedings of the ACM SIGCOMM Conference (SIGCOMM)*.

[51] Anirudh Sivaraman, Alvin Cheung, Mihai Budiu, Changhoon Kim, Mohammad Alizadeh, Hari Balakrishnan, George Varghese, Nick McKeown, and Steve Licking. 2016. Packet Transactions: High-Level Programming for Line-Rate Switches. In *Proceedings of the ACM SIGCOMM Conference*.

[52] Vibhaalakshmi Sivaraman, Srinivas Narayana, Ori Rottenstreich, S. Muthukrishnan, and Jennifer Rexford. 2017. Heavy-Hitter Detection Entirely in the Data Plane. In *Proceedings of the the SoSR*.

[53] Netronome Systems. 2017. *vEPC Acceleration Using Agilio SmartNICs.* https://www.netronome.com/media/documents/SB_vEPC.pdf

[54] Netronome systems. 2018. *Agilio CX SmartNIC.* https://www.netronome.com/m/documents/PB_NFP-4000.pdf

[55] Sami Tabbane. 2016. *Core network and transmission dimensioning.* https://www.itu.int/en/ITU-D/Regional-Presence/AsiaPacific/SiteAssets/Pages/Events/2016/Aug-WBB-Iran/Wirelessbroadband/core%20network%20dimensioning.pdf

[56] Amin Tootoonchian and Yashar Ganjali. 2010. HyperFlow: A Distributed Control Plane for OpenFlow. In *Proceedings of the the INM/WREN*.

[57] TRAI. 2017. *Highlights of Telecom Subscription Data.* https://main.trai.gov.in/sites/default/files/PR_60_TSD_Jun_170817.pdf

[58] P4 working group. 2017. *Behavioral-model.* https://github.com/p4lang/behavioral-model/tree/master/targets/simple_switch_grpc

[59] P4 working group. 2018. *P4Runtime.* https://github.com/p4lang/PI

[60] Xilinx. 2018. *Xilinx FPGA.* https://www.xilinx.com/products/silicon-devices/fpga.html

[61] Soheil Hassas Yeganeh and Yashar Ganjali. 2016. Beehive: Simple Distributed Programming in Software-Defined Networks. In *Proceedings of the SoSR*.

[62] Minlan Yu et al. 2010. Scalable Flow-based Networking with DIFANE. In *Proceedings of the ACM SIGCOMM*.