SoftCell: Taking Control of Cellular Core Networks

Xin Jin†, Li Erran Li⋆, Laurent Vanbever†, and Jennifer Rexford†
Princeton University†, Bell Labs⋆

ABSTRACT

Existing cellular networks suffer from inflexible and expensive equipment, and complex control-plane protocols. To address these challenges, we present SoftCell, a scalable architecture for supporting fine-grained policies for mobile devices in cellular core networks. The SoftCell controller realizes high-level service polices by directing traffic over paths that traverse a sequence of middleboxes, optimized to the network conditions and user locations. To ensure scalability, the core switches forward traffic on hierarchical addresses (grouped by base station) and policy tags (identifying paths through middleboxes). This minimizes data-plane state in the core switches, and pushes all fine-grained state to software switches at the base stations. These access switches apply fine-grained rules, specified by the controller, to map all traffic to the appropriate addresses and tags. SoftCell guarantees that packets in the same connection traverse the same sequence of middleboxes in both directions, even in the presence of mobility. Our characterization of real LTE workloads, micro-benchmarks on our prototype controller, and large-scale simulations demonstrate that SoftCell improves the flexibility of cellular core networks, while enabling the use of inexpensive commodity switches and middleboxes.

1. INTRODUCTION

The rapid proliferation of cellular devices (e.g., smart phones, tablets, and smart meters) is pushing existing cellular networks to their limits. New technologies like Long Term Evolution (LTE) are helping increase the capacity of radio access networks, placing even greater demands on cellular core networks to support many diverse devices and applications. Cellular core networks carry traffic between base stations and the Internet on behalf of user equipment (UE), as shown in Figure 1. The network relies on specialized equipment such as serving gateways (S-GWs) that provide seamless mobility when UEs move between base stations, and packet gateways (P-GWs) that perform a wide variety of functions like traffic monitoring and billing, access control, and parental controls. The base stations, serving gateways, and packet gateways communicate over GTP tunnels traversing a network of switches and routers.

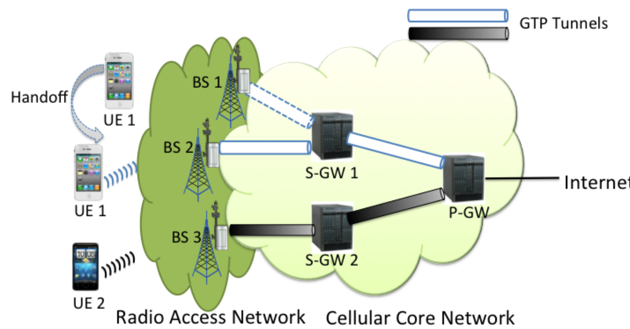


Figure 1: LTE network architecture

Cellular core networks are remarkably complex and inflexible [1, 2], an unfortunate legacy of their circuit-switched origins. Centralizing critical data-plane functionality at the boundary with the Internet forces all traffic to flow through the packet gateway—including device-to-device traffic and local Content Distribution Network (CDN) services within the same cellular network. With so much functionality in one box, it is not surprising that packet gateways are complex and expensive, and force carriers to buy functionality they do not need. Carriers cannot "mix and match" capabilities from different vendors (e.g., a firewall from one vendor, and a transcoder from another), or "scale up" the resources devoted to a specific function [2, 3]. Since the packet gateways are hard to change, carriers are forced to replace them to deploy new functionality, even when the existing equipment suffices for most purposes.

To make matters worse, growing link speeds and more diverse network policies will put even greater strain on packet gateways in the future. Cellular networks can apply customized policies based on a wide variety of *subscriber attributes* (e.g., the cell-phone model, the operating-system version, the billing plan, options for parental controls, whether the total traffic exceeds a usage cap, and whether a user is roaming), as well as the *application* (e.g., transcoding for video traffic, caching for Web traffic, and exemption from usage caps for applications that pay the carrier on the user's behalf) [2]. For example, the carrier may direct traffic for older cell phones through an echo-cancellation gateway, video traffic through a transcoder during times of congestion, and all traffic through a firewall, while applying different monitoring policies depending on the billing plan, usage cap, roaming status, and the application.

Rather than perform all these functions at the Internet boundary, we argue that cellular providers should adopt a network design more akin to modern data centers. The network should consist of a fabric of simple core switches, with most functionality moved to low-bandwidth access switches (at the base stations) and a distributed set of middleboxes that the carrier can expand as needed to meet the demands. These middleboxes could be dedicated appliances, virtual machines running on commodity servers [3], or simply packet-processing rules installed in the switches [4, 5]. A logically-centralized controller can direct traffic through the appropriate middleboxes, via efficient network paths, to realize a high-level service policy (e.g., directing a UE's video traffic through a transcoder and a firewall).

Cellular networks raise unique scalability challenges, compared to data-center and enterprise networks. Fine-grained policies can easily lead to an explosion in the data-plane state needed to direct traffic through the right middleboxes. This is especially true for the large volume of "north-south" traffic arriving from the Internet. In addition, stateful middleboxes require that all traffic in the same connection traverses the same middleboxes, even when a UE moves from one base station to another. The switches need to forward packets differently based on multiple factors (e.g., the UE and the application), which typically requires expensive TCAM (Ternary Content Addressable Memory) for packet classification. However, the merchant silicon chipsets used in commodity switches have just a few thousand to tens of thousands of TCAM entries. (See Table 2 in [6].) Supporting much larger packet classifiers would significantly increase the cost of the core switches.

To address these challenges, we present SoftCell, a scalable architecture for supporting fine-grained policies for mobile devices in cellular core networks. The SoftCell controller realizes high-level service polices by directing traffic through a sequence of middleboxes, optimized to the network conditions and UE locations. To ensure data-plane scalability, the core switches forward traffic on *hierarchical addresses* (grouped by base station) and *policy tags* (identifying middlebox paths). SoftCell pushes fine-grained packet classification to the access switches, which can be implemented easily in software. These access switches apply fine-grained rules, specified by the controller, to map UE traffic to the policy tags and hierarchical addresses. To ensure control-plane scalability, a local agent at the base station caches the service policy for each attached UE, to install rules in the access switch without involving the controller.

The SoftCell controller guarantees that packets in the same connection traverse the same sequence of middleboxes (*policy consistency*), and that bidirectional traffic traverses the same middleboxes in both directions (*policy symmetry*), even in the presence of mobility. SoftCell has an *asymmetric* edge architecture that does *not* require sophisticated packet classification of return traffic arriving at the gateway switches. SoftCell either *embeds* the policy tags in the UE IP address and port number (essentially "piggybacking" the information in the packets sent to the Internet), or *caches* them at the gateway (in a simple Network Address Translation table). This ensures return traffic flows through the right middleboxes, without requiring any support from the rest of the Internet. SoftCell also does not require any changes to UEs or the radio access network hardware, and can run on commodity switches and middleboxes.

In designing, prototyping, and evaluating SoftCell, we make the following contributions:

Fine-grained service polices: SoftCell supports fine-grained traffic steering based on applications and subscriber attributes, as well as flexible traffic engineering in selecting the network and middlebox paths.

Asymmetric edge design: SoftCell places most functionality at the many, low-bandwidth access switches, allowing the core network to use commodity hardware for the Internet gateway and other core switches.

Scalable data plane: SoftCell minimizes data-plane state in the core switches through multi-dimensional aggregation by policy tags, base station IDs, and UE IDs, and an algorithm for selecting policy tags.

Scalable control plane: To ensure control-plane scalability, access switches run local agents that cache service policies for the attached UEs, and the controller isolates the access switches from core topology changes.

Policy consistency and symmetry: SoftCell ensures that all traffic in the same TCP or UDP connection traverses the same sequence of middleboxes in both directions, even in the presence of mobility.

Realistic performance evaluation: We evaluate the scalability our architecture based on traces from a large LTE deployment, micro-benchmarks on a prototype controller, and large-scale simulation experiments.

We believe SoftCell significantly improves the flexibility of cellular core networks, while enabling the use of inexpensive commodity switches and middleboxes.

2. SOFTCELL ARCHITECTURE

A SoftCell network consists of commodity middleboxes and switches managed by a controller. The controller supports flexible, high-level service policies by computing and installing rules in the switches to di-

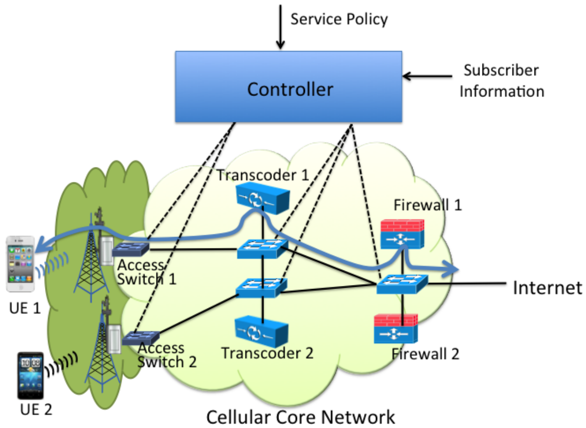


Figure 2: SoftCell network architecture

rect traffic through the right middleboxes and network paths. To support flexible policies without compromising scalability, SoftCell capitalizes on the unique properties of cellular core networks—particularly the fact that most traffic begins at the base-station edge, where the small number of flows and the small uplink bandwidth enable the use of flexible software switches.

2.1 SoftCell Core Network Components

The cellular core network connects to unmodified UEs (via base stations) and the Internet (via gateway switches), as shown in Figure 2. SoftCell does *not* require the specialized network elements (e.g., serving and packet gateways) or point-to-point tunneling (e.g., user-level GTP tunnels) used in today's LTE networks, as shown earlier in Figure 1.

Middleboxes: SoftCell supports commodity middleboxes implemented as dedicated appliances, virtual machines, or packet-processing rules on switches. Each middlebox function (e.g., transcoder, web cache, or firewall) may be available at multiple locations. Many middleboxes require all packets in both directions of a connection to traverse the same instance of the middlebox.

Access switches: Each base station has an *access* switch that performs fine-grained packet classification on traffic from UEs. Access switches can be software switches (such as Open vSwitch [7]) that run on commodity server hardware. The server can also run a local agent that caches service policies for attached UEs, to minimize interaction with the central controller.

Core switches: The rest of the cellular core consists of *core* switches, including a few *gateway* switches connected to the Internet. These core switches perform multi-dimensional packet classification at high speed, but only for a few thousands or tens of thousands of rules. We assume that the packet-processing hardware can perform arbitrary wildcard matching on the IP addresses and TCP/UDP port numbers (as in today's merchant silicon), or can cache flat rules after processing wildcard rules locally in software (as in DevoFlow [8]). Our gateway switches are much cheaper than P-GWs. They can be flexibly placed at many locations with access to the Internet. SoftCell enables a *"flatter"* core network architecture with more efficient routing than current LTE does.

Controller: The controller computes and installs switch-level rules that realize a high-level service policy, specified based on subscriber attributes and applications, by installing paths that direct traffic through middleboxes. The controller knows the attributes (e.g., billing plan, phone model, and usage cap) of each UE, allowing the controller to identify the appropriate clauses in the service policy for handling the UE's traffic.

The radio access networks consist of base stations that connect to unmodified UEs using existing protocols for mobility management, session management, and authentication. Just as today, a UE retains a single IP address as it moves between base stations in the same cellular core network; any changes our cellular core network makes to the IP addresses of packets are not visible to the UEs. We do not change the radio hardware at the base station, or common functions such as scheduling, radio resource management, and paging. SoftCell only changes how the base stations communicate with the core network, by having the base stations coordinate with the controller to enforce service policies. Similarly, SoftCell does not require changes to commodity middleboxes, or any support from the rest of the Internet.

2.2 Flexible, High-Level Service Policies

The SoftCell controller directs traffic over network and middlebox paths, based on the service policy. We believe that carriers should specify service policies at a high level of abstraction, based on subscriber attributes and applications, and rely on the controller to handle low-level details like ephemeral network identifiers, the locations of middleboxes and switches, and application identification. A service policy has multiple clauses that each specify which traffic (specified by a predicate) should be handled in what way (specified by an action):

Predicates: A predicate is a boolean expression on subscriber attributes, application type, and cell properties. Subscriber attributes consist of device type, billing plan, device capabilities, provider, etc. Application types include web browsing, real-time streaming video, VoIP, etc. Cell attributes include the air interface congestion level, capacity, etc.

Service action: An action consists of a set of middleboxes, along with quality-of-service (QoS) and access-control specifications. Specifying the set of middleboxes as a partial order allows the carrier to impose constraints (e.g., firewall before transcoder). The action

Pri	Predicates	Service Action
1	provider = B	Firewall
2	provider != A	Drop
3	app = video $\wedge$ plan = Silver $\wedge$ congestion > 7	[Firewall, Transcoder]
4	app = VoIP	QoS = expedited-forward $\wedge$ Firewall
5	*	Firewall

Table 1: Example service policy for carrier A

does not indicate a specific instance of each middlebox, allowing the controller to select middlebox instances and network paths that minimize latency and load.

Priority: The priority is used to disambiguate overlapping predicates. The network handles traffic using the highest-priority clause with a matching predicate.

Table 1 shows an example service policy that carrier A applies to traffic arriving at UEs, where outbound traffic follows the reverse sequence of middleboxes. Carrier A has a roaming agreement with carrier B, so the first clause directs traffic from B's subscribers through a firewall. The second clause disallows traffic from subscribers from all other carriers. The remaining clauses specify the handling of A's own subscribers, with all traffic going through a firewall. The third clause indicates that the video traffic to subscribers on the "silver" billing plan must go through a transcoder (after the firewall) when cell congestion at the base station exceeds a target level. The fourth clause specifies that VoIP traffic should be assigned to the "expedited forwarding" service class to protect this application from a heavy load of best-effort traffic. The fifth clause requires that all other traffic goes through a firewall. In this paper, we focus on middlebox service policies, since they require more sophisticated *traffic steering* rather than simple local processing to drop packets or mark the type-of-service bits.

2.3 Scalability Design Principles

The main challenge in SoftCell is to support flexible policies without compromising scalability. To motivate our main design decisions, we briefly discuss the main factors affecting the scalability of cellular core networks, and perform back-of-the-envelope calculations based on publicly-available statistics. We consider the design of a typical cellular core network serving a large metropolitan area with 1000 base stations [9].

Microflow rules in software access switches: A modern base station can serve around 1000 UEs [10]. Not surprisingly, most UEs have just a handful of active TCP or UDP connections at a time [11, 12]. A base station with 1000 UEs, each with (say) 10 active TCP/UDP connections, would have 10K simultaneously active flows. The backhaul link from the base station to the rest of the core network has a capacity of anywhere from 20 Mbps to 1 Gbps [9, 13]. A software switch like Open vSwitch [7] can easily store 100K microflows in a hash table, and perform packet forwarding at several gigabits per second [14], comfortably within these requirements. Open vSwitch can install around 1K flow entries every 100 msec [15], able to support a new flow from each UE every tenth of a second. These results suggest that a software switch can easily keep up with the number of flows, flow set up rates, and aggregate bandwidth at each base station.

Exploit the dominance of UE-initiated traffic: Most traffic in cellular networks is "north south", as opposed to data-center networks where most traffic is "east west". What's more, clients are usually (almost always) UEs in these "north south" traffic, which means traffic is first initiated from UEs. Actually, many cellular operators deploy NATs and stateful firewalls to forbid connections initiated from the Internet [16], as a way to protect their networks. This factors into our solution to place most key functionality at access switches.

Avoid fine-grained packet classifiers at the gateway switches: The gateway switches need to handle the traffic for 1000 base stations, each with (say) 10K active flows. This results in roughly 10 million active flows—too large for fine-grained packet classification using commodity switch hardware. To leverage merchant silicon with thousands to tens of thousands of TCAM entries, the data-plane state in the core switches should not be more than (say) an order of magnitude higher than the number of base stations. As such, the gateway switches should *not* perform fine-grain packet classification to identify the base station or service action associated with the incoming packets.

Exploit locality to reduce data-plane state in the core switches: Fortunately, cellular core networks have natural geographic locality, with the access switches aggregating through metro networks to mobile switching offices, through the core to gateway switches. Since a *cluster* of around 10 base stations connect (in a ring, tree, or mesh topology) to the core [17], aggregating by base station clusters can reduce data-plane state by an order of magnitude. In addition, traffic for these base stations would often traverse the same nearby middlebox instances (to minimize latency and network load), offering further opportunities to reduce the state required to support service policies.

Avoid fine-grained events at the controller: While the access switches can maintain per-flow state, the *controller* cannot manage the network at the flow level. With an arrival rate of (say) 1-10K flows/second from each of 1000 base stations, a controller that processes microflows would need to handle 1M-10M events per second, roughly doable with today's SDN controller platforms [18], but only at the expense of flow set-up latency and high overhead. Instead, we believe the SoftCell controller should only handle coarse-grained events, such as UE arrivals at base stations or traffic requiring a new service action.

Avoid updating access switches after topology changes: In addition to satisfying the service policy, the controller must response in real time to network events such as link failures or congestion by computing and installing new rules in the switches. If the controller also needed to update all 1000 base stations, routing convergence time would suffer. Instead, routing changes should be isolated to the core switches, without updating the access switches.

The first four principles ensure that SoftCell has a scalable data plane, as discussed in Section 3. Then, Section 4 applies the last two principles to ensure the control plane scales.

3. SCALABLE DATA PLANE

To ensure the scalability of the data plane, the access switches apply fine-grained rules that map packets to hierarchical addresses and coarse-grained policy tags, with the help of the controller. The core switches direct traffic based on these large aggregates. By selecting base station address blocks and policy tags intelligently, the controller can enable aggregation across nearby base stations and related policy tags to further reduce the state. To avoid classifying packets arriving from the Internet, SoftCell either embeds the forwarding information in the IP+TCP/UDP header (essentially "piggybacking" the state in outgoing packets) or caches policy tags at the gateway. When a UE moves from one base station to another, the controller installs temporary rules in the core switches to direct in-progress flows to the new location while ensuring policy consistency.

The controller directs traffic over a *policy path*, a sequence of switches and middleboxes from one edge switch to another. To simplify the discussion, we initially assume that the controller handles the first packet of each flow, similar in spirit to Ethane [4]. This clearly would compromise the scalability of the controller—an issue we address in Section 4.

3.1 Core: Multi-Dimensional Aggregation

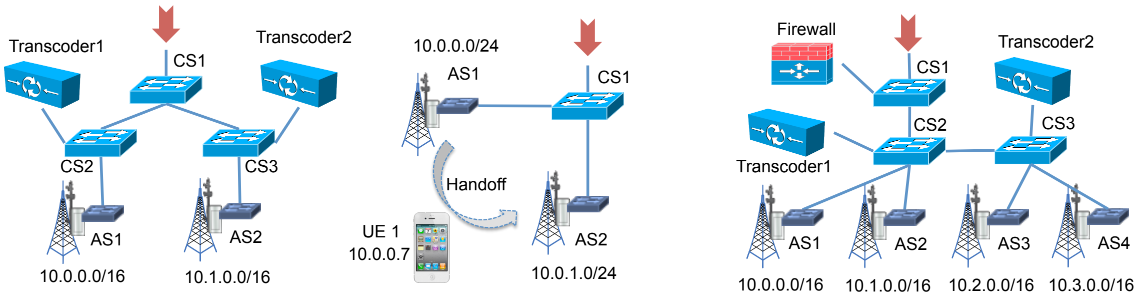
Delivering different traffic over different sequences of middleboxes is hard to achieve in a scalable way. Suppose we have 1000 base stations, each with 1000 UEs, where each UE has 1000 service policy clauses. Installing a path for each service policy clause would lead to 1 billion paths. If implemented naively, this would generate a huge amount of rules in the switches. The key idea to achieve scalability is to aggregate traffic on multiple dimensions, i.e., policies, base stations, and UEs.

Aggregation by policy (policy tag): Service policies defined on high-level attributes seem very compact. However, subscriber attributes are not easily translated or aggregated with network addresses. For example, since UEs with "Silver Plan" can have a variety of IP addresses, the third clause of the service policy in Table 1 may require a rule for each flow in the worst case. We could conceivably assign "Silver Plan" UEs IP addresses under the same subnet, allowing us to assign one rule that matches on the IP prefix. However, we cannot do this for every attribute, not to mention that many service policies are defined on combinations of attributes. To minimize the rules in core switches, we use a *policy tag* to aggregate flows on the same policy path. We associate packets with a policy tag at the access switch, allowing core switches to forward packets based on coarse-grained policy tags.

Aggregation by location (hierarchical IP address): In many core switches, traffic destined to the same base station would traverse the same output link, even if the packets go through different middleboxes. By including location information in the UE addresses, we can aggregate traffic by IP prefix. Furthermore, cellular core networks have a natural hierarchical structure. Therefore, we assign each base station an IP prefix, called *base station ID*, and IDs of nearby base stations can be further aggregated into larger blocks. We can aggregate even more by combining policy tags and IP addresses. Suppose two policy paths going to two base stations share a long path segment before branching. If assigned the same policy tag, a single rule matching on the tag can forward packets along the shared segment until the branching point, where traffic divides based on the base station prefix.

Aggregation by UE (UE ID): Packets also need a UE identifier (*UE ID*) that differs from other UEs at the same base station. For example, some middleboxes (like intrusion detection systems) need a way to identify groups of flows associated with the same UE which is impossible if all flows for the same base station share the same address. In addition, having a UE ID in each packet enables optimizations for handling mobility, by installing switch rules that forward in-progress flows to the UE at its new location. Together, the base station prefix and the UE ID form a hierarchical location-dependent address (LocIP) for the UE. Using hierarchical "care of" addresses to handle mobility is an old idea [19, 20]. However, prior work does not consider service policies, or the techniques described in the next two subsections to ensure policy symmetry and consistency.

Our key idea is to *selectively* match on the three di-



Switch	Match	Action
CS1	10.0.0.0/16	Forward to CS2
	10.1.0.0/16	Forward to CS3

(a) Location-based routing

Switch	Match	Action
CS1	10.0.0.7	Forward to AS2
	10.0.0.0/16	Forward to AS1
	10.1.0.0/16	Forward to AS2

(b) Mobility

Switch	Match	Action
CS1	tag1	Forward to Firewall
CS2	tag1, 10.0.0.0/15	Forward to Transcoder1
	tag1, 10.2.0.0/15	Forward to CS3
CS3	tag1, 10.2.0.0/15	Forward to Transcoder2

(c) Flexible policy

Figure 3: Examples of multidimensional aggregation rules for traffic arriving from the Internet

mensions to maximize aggregation of data-plane state:

Location-based routing: In Figure 3(a), core switch CS1 matches on the base-station prefix to forward traffic to CS2 and CS3. CS2 and CS3 decide whether to direct traffic to a transcoder based on the policy tag, but CS1 does not need to base its forwarding decision on the tag.

UE mobility: In Figure 3(b), CS1 forwards traffic to base stations based on the destination IP prefix. When UE1 moves from access switch AS1 to AS2, we install a high-priority rule at CS1 to match on both the base station prefix and the UE ID. This ensures that ongoing flows reach UE1 at AS2 over a direct path.

Flexible policy: Figure 3(c) illustrates how to implement the third clause in Table 1, using the tag "tag1." CS1 forward "tag1" packets to the Firewall[1]. Suppose we assign AS1 and AS2 traffic to Transcoder1, and AS3 and AS4 traffic to Transcoder2. Then CS2 matches on both the tag and the prefix (more precisely, aggregated prefix of two base stations) to forward AS1 and AS2 traffic to Transcoder1, and AS3 and AS4 traffic to CS3. CS3 finally forwards AS3 and AS4 traffic to Transcoder2.

3.2 Asymmetric Edge: Packet Classification

To minimize data-plane state in the core, we want to classify packets and associate them with tags as they enter the network. The access switch maintains microflow rules that, upon receiving a packet from a UE, rewrite the IP address (to the location-dependent address) and tags the packet with the policy tag. Similarly, upon receiving packets from the core network, the access switch rewrites the IP address back to the value the UE expects. The access switch learns the appropriate rules from the controller. For example, the access

[1]Traffic *from* middleboxes is identified based on the inport.

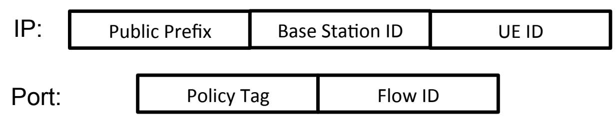


Figure 4: Embedding location and policy information in source IP address and source port number. Thus the information can be implicitly piggybacked in return traffic.

switch could send the first packet of each flow to the controller, and have the controller install the appropriate rule to direct the remaining packets over a chosen policy path. For better control-plane scalability, the controller can provide a local agent at the base station with appropriate classifiers for handling any traffic from this UE, as discussed in more detail in Section 4.

Performing fine-grain packet classification is acceptable at the access switch, due to the low link speeds and the relatively small number of active flows. However, gateway switches must handle orders of magnitude more flows, and more flow-arrival events, so they should not perform such fine-grained packet classification. As such, we adopt an asymmetric design where the gateway switches can easily determine the appropriate IP address and tag to use, in one of two ways:

Embedding state in packet headers: Rather than *encapsulating* packets, as is commonly done in data-center networks, we can *embed* the policy tag, base station ID, and UE ID in the packet header. This ensures that the return traffic carries these fields. For example, we could encode the state as part of the UE's IP address (e.g., in IPv6), or a combination of the UE's IP address and TCP/UDP port number (e.g., in IPv4) as shown in Figure 4. The access switch rewrites the source IP address to the location-dependent IP address (i.e., the carrier's public prefix, as well as the base station and UE IDs), and embeds the policy tag as part of the source port. UEs do not have many active flows, leaving plenty of room for carrying the policy tag in the port-number field. With this embedding mechanism, our three identifiers are implicitly "piggybacked" in return traffic arriving from the Internet[2]. The gateway switch can simply make forwarding decisions based on the destination IP address and port number of incoming packets.

Caching state in gateway switches: Instead of embedding state in packet headers, the gateway switch can cache the state when forwarding outgoing packets, and associate the state with the return traffic arriving from the Internet. In this scheme, the gateway switch performs network address translation, and caches the tag in the process. In practice, network address translation may be necessary anyway, if the cellular provider does not have a large enough public IP address block to allocate a unique address for each UE. While NATing does introduce per-flow state, the gateway switch does *not* need to contact the controller or translate the UE's address into the subscriber attributes. While the gateway would need a larger table than the other core switches, supporting microflow rules does not require expensive TCAM or any sophisticated processing.

3.3 Policy Consistency Under Mobility

Seamless handling of device mobility is a basic requirement for cellular networks. UEs move frequently from one base station to another, and carriers have no control over when and where a UE moves. In addition to minimizing packet loss and delay, carriers must ensure that ongoing flows continue traversing the original sequence of middleboxes (though not necessarily the same switches), while reaching the UE at its new location. Such *policy consistency* is crucial for traffic going through stateful middleboxes, like firewalls and intrusion prevention systems. However, new flows should traverse middlebox instances closer to the UE's new location, for better performance. As such, SoftCell must differentiate between old and new flows, and direct flows on the appropriate paths through the network

Differentiate between old and new flows: Incoming packets from old flows have a destination IP address corresponding to the UE's old location, so these packets naturally traverse the old sequence of middleboxes. SoftCell merely needs to direct these packets to the new base station, which then remaps the old address to the UE's permanent address. During the transition, the controller does not assign the old location-dependent address to any new UEs. For the traffic sent *from* the UE, the old access switch has a complete list of microflow rules for the active flows. Copying these rules to the new access switch ensures that packets in these flows continue to be mapped to the old IP address, to avoid a disruption in service. Each UE has a relatively small number of active connections (say, 10), limiting the overhead of copying the rules. To minimize handoff latency, the SoftCell controller could copy these rules in advance, as soon as a UE moves near a new base station.

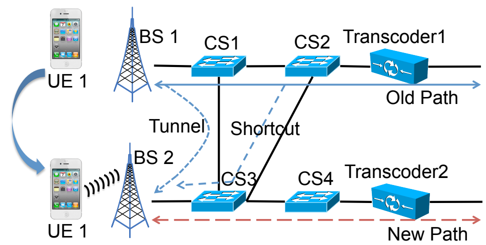


Figure 5: Tunnels and shortcuts for old flows

Efficiently reroute the old flows: To handle ongoing connections during mobility events, SoftCell maintains long-lived *tunnels* between nearby base stations, as shown in Figure 5. These tunnels can carry traffic for *any* UEs that have moved from one base station to another. This "triangle routing" ensures policy consistency and minimizes packet loss, at the expense of higher latency and bandwidth consumption. The many short-lived connections would not experience any significant performance penalty. To handle long-lived connections more efficiently, the controller can establish temporary *shortcut* paths for directing traffic between the new base station and the old policy path, as shown in Figure 5. The controller can learn the list of active microflows from the access switch at the old base station, and install rules in the core switches to direct incoming packets over the shortcut paths. A single UE may need multiple shortcuts, since different traffic may go through different middleboxes[3]. As such, these shortcut paths are created when a UE moves, and removed when a soft timeout expires—indicating that the old flow has ended.

3.4 Rule Minimization in Core Switches

We have shown that we can reduce the number of core switch rules by relying on multi-dimensional aggregation. We now present an online algorithm that performs policy path implementation in real time on a

[2]This approach raises some security and privacy challenges. Malicious Internet hosts may spoof policy tags and congest network links or middleboxes, though these attacks can be blocked using conventional firewalls. In addition, a UE 's IP address changes upon moving to a new base station, making it easier for Internet servers to infer the UE's location. Network address translation can reduce these concerns.

[3]No short-cut paths are needed in the common case when a UE moves to another base station in the same *cluster*, since these base stations connect to the same core switch. In this case, simply adding the microflow rules at this core switch is sufficient.

per policy path basis. For ease of description, we first describe our path implementation algorithm assuming the policy path is a simple path, as shown in Algorithm 1. We then discuss how to deal with loops.

Simple tag reuse rules: To reduce the amount of switch rules, we want to maximize the reuse of existing rules which match policy tags and base station IDs. Our first step is to pick a tag that is already used in switches where the new policy path includes. To ensure correctness, we impose the constraint that different policy paths originated from the same destination access switch to have different tags. Otherwise, we would not be able to distinguish among different policy paths from the same base station. As shown in Algorithm 1, we enumerate the candidate tags and choose the tag that results the minimal number of new rules we need to install (line 1-8). It is possible that the candidate tags are an empty set, in which case we will choose a random unused tag (line 8).

Safe aggregation: When we iterate over each switch for a given tag t along the policy path, for each switch, we calculate how many rules we need to install with the candidate tag and base station prefix (line 4). This is done in method $sw_1.getNewRule(t, prefix, sw_2)$. This method performs *safe aggregation*. In particular, the method looks at all switch rules with tag t with an action that forwards to the same next-hop switch sw_2. It will try to aggregate the base station prefixes of the rules. By safe aggregation, we mean that resulting prefix of the aggregate rule contains the exact number of component prefixes. For example, if there are three /16 base station prefixes, we can not aggregate them into a /14 prefix. On the other hand, if we have all four component prefixes, we can aggregate them into the /14 prefix. After picking the tag tag^* to use, the algorithm installs the path with the prefix and the tag. We only need to install rules to switches where we cannot utilize existing rules to reach the correct next hop (line 10-15). We can install aggregate rules if the aggregation is safe (line 13). Otherwise, we just install this rule (line 15). Note that safe aggregation is done *atomically* to prevent inconsistencies introduced by race condition.

Dealing with loops: Ideally, we should only compute and install loop-free paths. However, due to the flexibility of service policies and placements of middleboxes, loops are sometimes unavoidable. For instance, in Figure 2, there is no way to avoid a loop in the path if a service policy clause requires outbound video traffic to go through a firewall before a video transcoder. A loop that enters a switch twice but from different links can be easily differentiated by input ports. However, a loop that enters a switch twice from the *same* link is more difficult to handle. In such a case, we use additional tags to help switches make forwarding decisions. More specifically, we break a loop into two segments; each segment uses one tag for forwarding. At the switch that connects these two segments, we install a rule to "swap" these two tags. This approach can be generalized to support nested loops.

Algorithm 1 Install A New Policy Path

Input:
- $path$: the policy path to install
- $prefix$: the IP prefix of the base station
- $candTag$: the set of candidate tags for the base station
- $usedTag$: the set of tags used by the base station

Output: *switch rules* and a *tag* for this policy path

Step 1: Choose a tag to minimize new rules

1: **for** t in $candTag$ **do**
2: $\quad newRule[t] = 0$ ▷ new rules needed if tag t is used
3: $\quad$ **for** (sw_1, sw_2) in $path$ **do**
4: $\quad\quad newRule[t] += sw_1.getNewRule(t, prefix, sw_2)$
5: **if** $candTag \mathrel{!}= \emptyset$ **then**
6: $\quad tag^* = \arg\min_t \{newRule[t]\}$
7: **else**
8: $\quad tag^* = random\{t | t \notin usedTags\}$
9: $usedTag = \{tag^*\} \cup usedTag$

Step 2: Install the path with the prefix and tag

10: **for** (sw_1, sw_2) in $path$ **do**
11: $\quad$ **if** $sw_1.getNextHop(tag^*, prefix) \mathrel{!}= sw_2$ **then**
12: $\quad\quad$ **if** $sw.canAggregate(tag^*, prefix, sw_2)$ **then**
13: $\quad\quad\quad sw.aggregateRule(tag^*, prefix, sw_2)$
14: $\quad\quad$ **else**
15: $\quad\quad\quad sw.installRule(tag^*, prefix, sw_2)$

4. SCALABLE CONTROL PLANE

Sending the first packet of every flow to the central controller would introduce a high overhead. Instead, a local agent at each base station offloads part of the control-plane functionality. In this section, we first present the design of the local agent and then describe how the control plane handles network dynamics.

4.1 SoftCell Local Agent

Each base station runs a local software agent equipped with the computing power to conduct various management tasks, including radio resource allocation for UEs. The local agent caches a list of *packet classifiers* for each UE at the behest of the central controller. The packet classifiers are a *UE-specific* instantiation of the service policy that matches on header fields in the packet and identifies the appropriate policy tag, if a policy path already exists. When the UE arrives at the base station, the controller computes the packet classifiers based on the service policy, the UE's subscriber attributes, and the current policy tags. When the UE starts a new flow, the local agent consults these classifiers to determine the right policy tag for these packets, and installs a microflow rule in the access switch, similar to the "clone" function in DevoFlow [8]. The local agent only contacts the controller if no policy tag exists for this traffic—that

	Access Switch	Core Switch	Controller w/o Local Agent	Controller w/ Local Agent	
				Central Controller	Local Agent
UE Arrival	Yes	No	Yes	Yes	Yes
Flow Arrival	Yes	Sometimes	Yes	Sometimes	Yes
UE Handoff	Yes	Yes(Relevant)	Yes	Yes	Yes
Topology Change	No	Yes(Relevant)	Yes	Yes	No
Dynamic Policy	No	Yes(Relevant)	Yes	Yes	No

Table 2: How network events affect data plane and control plane. *Yes* means the switch/controller is involved in the event, *No* means not involved, *Sometimes* in *Flow Arrival* means only involved when the policy path has not been installed, and *Relevant* means only relevant switches (a small number of the whole) are involved. Central controller offloads most *Flow Arrival* events to local agents.

is, if the packet is the first traffic at this base station, across all UEs, that need a particular policy path.

Let's use an example to illustrate this. Suppose UE7 arrives at base station 1 with prefix 10.0.0.0/16. The local agent first assigns a UE ID *10* to the UE. Now UE7 is associated with the location-dependent address 10.0.0.10. The local agents also contacts the controller to fetch a list of packet classifiers for this UE. Suppose the list includes two packet classifiers:

1. match:dst_port=80, action:tag=2
2. match:dst_port=22, action:send-to-controller

When a packet with destination port 80 from UE7 arrives, the access switch find any existing microflow rule, and directs the packet to the local agent. The local agent determines that the traffic matches the first packet classifier. Since the policy path already exists, the local agent simply installs a microflow rule in the access switch which (i) rewrites the UE IP address to 10.0.0.10 and (ii) pushes "tag=2" to the source port number, without contacting the central controller. Suppose another packet arrives from UE7 with destination port 22. This flow matches the second packet classifier and the action is "send to controller". This means the policy path to base station 1 has *not* been installed yet. The local agent sends a request to the central controller to install a new policy path and returns the policy tag. Then, the local agent can update the packet classifier and install a microflow rule for the packets of this flow.

In this way, local agents cache UE-specific packet classifiers and process most microflows locally, significantly reducing the load on the controller.

4.2 Handling Network Dynamics

Next, we discuss how the control plane deals with network dynamics, as summarized in Table 2. We already discussed UE and flow arrival in Section 4.1, and UE handoff in Section 3.3 respectively. Here, we just briefly discuss topology change and dynamic policy.

Topology change: When the network topology changes (due to a link, switch, or middlebox failure), the controller calculates and installs new paths for affected policies. In some cases, like a stateful middlebox crash without any state saved, in-progress flows may experience significant packet loss or even have to terminate. The topology change is handled by the controller and only affects relevant switches. There is no need to update all access switches to change their flow table.

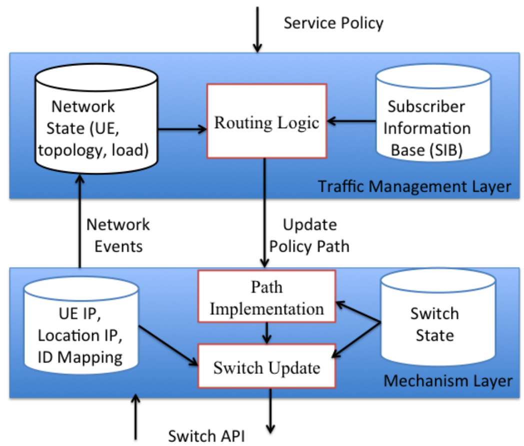


Figure 6: SoftCell controller

Dynamic policy: In addition to static policy, SoftCell supports dynamic policy which change the policy path during the lifetime of a flow. For example, when the air interface of a base station is congested, the service policy may require video traffic to go through a transcoder. In this case, the controller must install a new path for video traffic. The central controller updates the policy paths in the core network, based on the policy requirements, without changing the policy tags.

5. EXTENSIBLE CONTROLLER DESIGN

In addition to supporting service policies, carriers need to manage their network and middlebox resources, to minimize latency and balance load. Our controller design cleanly separates traffic management from the low-level mechanisms for installing rules and minimizing data-plane state, as shown in Figure 6.

Traffic-management layer: The traffic-management layer computes policy paths through the switches and middleboxes, to satisfy both the service policy and traffic-management goals. This layer determines the service attributes for a UE from the Subscriber Information Base (SIB), and consults the service policy to compute policy paths that traverse the appropriate middleboxes and optimize traffic-management objectives.

Mechanism layer: The mechanism layer realizes the policy paths by installing rules in the underlying switches, using the techniques proposed in the previous two sections. This layer hides all the details of location-dependent addresses, the encoding of policy tags, the path implementation algorithm, and assuring path consistency during mobility. The mechanism layer could also poll traffic counters in the switches and aggregate them to the level of policy tags to enable the traffic-management layer to operate on a coarser-grain view of the traffic.

A modular controller design allows each layer to evolve independently, to adopt new innovations in how to manage traffic and data-plane state, respectively.

6. PERFORMANCE EVALUATION

In this section, we demonstrate the scalability and performance of our SoftCell architecture. First, we measure the workload that SoftCell would face in a typical cellular core network by analyzing a trace from a large LTE network. We then show that SoftCell is able to sustain several times of this workload by performing micro benchmark. Finally, we show that SoftCell can handle thousands of service policy clauses on commodity switches trough large simulations.

6.1 LTE Workload Characteristics

As a first step towards SoftCell deployment, we measured the workload of a real cellular network to understand the practical performance requirements of the controller.

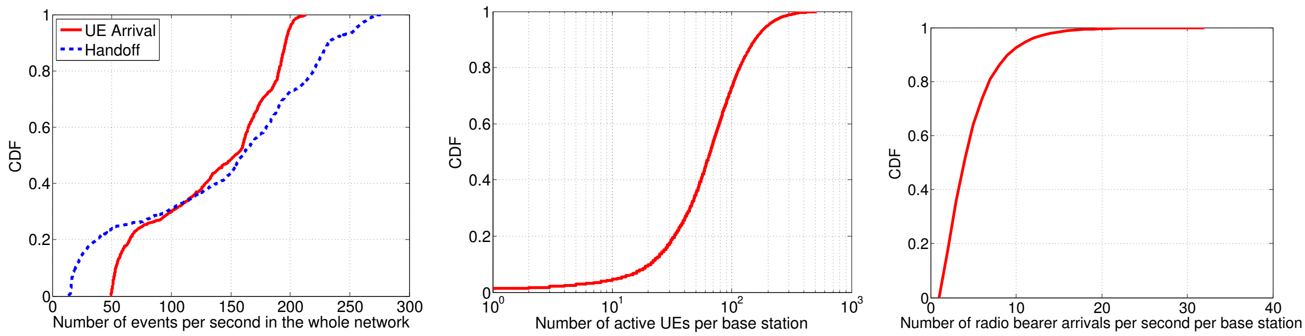
Dataset Description: We collected about 1TB traces from a large ISP's LTE network during one week in January 2013. The dataset covers a large metropolitan area with roughly 1500 base stations and 1 million mobile devices (including mobile phones and tablets). The trace is bearer-level and includes various events such as radio bearer creation, UE arrival to the network, UE handoff between base stations, etc. A radio bearer is a communication channel between a UE and its associated base station with a defined Quality of Service (QoS) class. When a flow arrives and there is an existing radio bearer with the same QoS class, the flow will use the existing radio bearer. Since radio bearers timeout in a few seconds, it is possible that a long flow may trigger several radio bearer creation and deletion. Since we do not have flow-level information, we use radio bearers as an estimation of flow activity. We present measurement results for a typical week day.

Network wide characteristics: Figure 7(a) shows the CDF of UE arrival events and handoffs in the whole network. A UE arrival event means a new UE first attaches to the network, e.g., after a UE is powered on. When a UE arrives at the network, the central controller fetches the UE attributes from the SIB and send the UE's packet classifiers to the local agent. A UE handoff event means a UE transfers from one base station to another. Upon handoff, the controller has to copy state from the old access switch to the new access switch with the help of local agents, and set up shortcuts for long flows. We do not account for UE handoffs between cells of the same base station as they do not cause forwarding changes. From the figure, we can see that the 99.999 percentile of UE arrival and handoff events per second are 214 and 280, respectively. While each of these events requires the central controller to contact local agents or update core switches, it is not a problem as today's commodity servers and software switches can easily handle tens of thousands of these events per second. Actually, even if we account for the exponential growth of mobile data [21] (18 times in the next five years), the workload can still be easily handled by commodity servers and software switches.

Load on each base station: Figure 7(b) shows the CDF of active UEs per base station. We see that a typical base station handles hundreds of active UEs simultaneously, with a 99.999 percentile of 514. Figure 7(c) depicts the radio bearer arrival rate at each base station. The number is relatively small, i.e. only 34 for the 99.999 percentile. As one radio bearer typically carries a handful of concurrent flows [11, 12], we expect the actual flow arrival rate to be around several hundred flows per second. These results imply that the local agent has to keep state for several hundred of UEs and process a maximum of tens of thousands new flows per second. However, as most of the time policy paths would have already been installed in the network, new flow requests only require the local agent to install packet classification rules at the access switch. Again, tens of thousands of these events per second can be easily handled by today's software switches.

6.2 Controller Micro Benchmark

We have implemented a SoftCell control plane prototype on top of the popular Floodlight [22] OpenFlow controller. The prototype implements both SoftCell central controller and SoftCell local agent. To perform topology discovery, we rely on the "TopologyService" module provided by Floodlight. Since there is no support for middlebox discovery in Floodlight, we encode middlebox placement and specification in a configuration file that is provided to the controller. The local agent fetches packet classifiers from the global controller upon every new UE arrival, then it uses the packet classifiers to handle all the following flows from the new UE. The communication between a local agent and the



(a) Events in the whole network (b) Active UEs per base station (c) Radio bearer arrivals per base station

Figure 7: Measurement Results of a LTE network

global controller is implemented with the Floodlight REST API.

In the following, we perform micro benchmark on the prototype, then we compare the results with the measurement results obtained earlier to demonstrate the ability of our controller to sustain the workload. We benchmark the prototype using Cbench [23]. Cbench emulates a number of switches, generates packet-in events to the tested controller, and counts how many events the controller processes per second (throughput). Each test server has an intel XEON W5580 processor with 8 cores and 6GB of RAM.

Central controller performance: First, we evaluate the throughput of the controller. Recall that the controller has to send packet classifiers to local agents when a UE attaches or moves to a base station. We use Cbench to emulates 1000 switches and let these switches keep sending packet-in events to the controller. From the controller viewpoint, these packet-in events correspond to packet classifier requests coming from 1000 local agents. The controller then replies to these requests with packet classifiers as fast as it can.

Our results show that the controller can process 2.2 million of requests per second with 15 threads. These results clearly demonstrate that the SoftCell controller can sustain the load of a large LTE network as, from our measurement study, we know that only hundreds to thousands of such events are encountered per second. Also, these results are similar to the ones reported by [18], the difference being mainly due to different settings and platforms.

Local agent performance: Second, we evaluate the throughput of the local agent. Recall that the local agent needs to fetch packet classifiers from the controller when processing events. The throughput of the local agent therefore depends on how frequently it needs to contact the controller which itself depends on the cache hit ratio. Table 3 shows the evolution of the local agent throughput in function of the cache hit ratio. A cache hit ratio of 80% means that the local agent can handle 80% of events locally and need to contact the controller for the remaining 20% of the events. To measure the throughput, we use Cbench to emulate the access switch connected to the local agent and let it keep sending packet-in events to the local agent. Upon the reception of a packet-in event, the local agent performs a lookup in its cache and contact the controller upon cache misses. The local agent and the controller run on two separate servers connected by the department LAN.

Cache Hit Ratio	0%	20%	40%	60%	80%	100%
Throughput	1.8K	2.3K	3.0K	4.5K	8.6K	505.8K

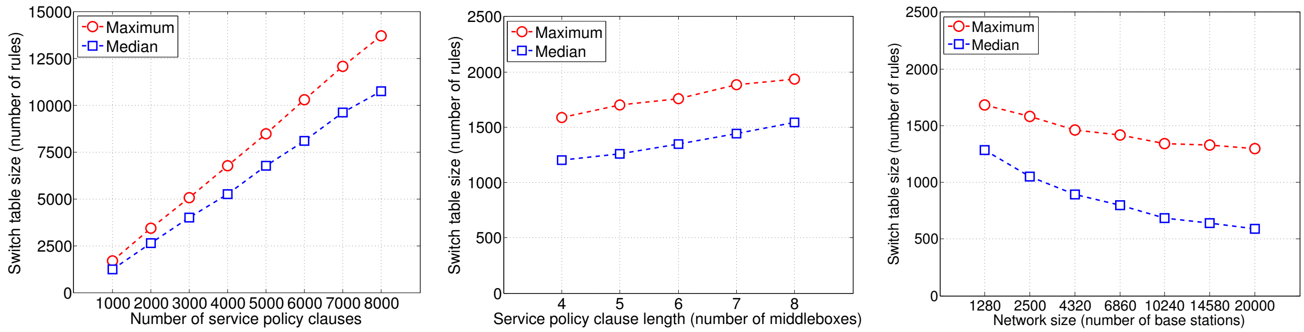
Table 3: Effect of cache hit ratio on local agent throughput

Again, the local agent throughput is sufficient to handle the number of new flows measured at a base station (a few to tens of thousands per second). Indeed, even in the worst case where the local agent has to contact the controller for every event, it is still able to handle 1.8K events per second. Not to mention that we can also optimize the cache hit ratio, e.g., by prefetching packet classifiers from the controller.

6.3 Large-Scale Simulations

We now demonstrate the scalability of SoftCell data plane through large scale simulations. In particular, we show that SoftCell only requires a few thousand TCAM entries to support thousands of service policy clauses for thousands of base stations.

Methodology: We generate hierarchical topology composed following the description of cellular core networks in [9, 17]. Each topology is composed of three layers: *access*, *aggregation* and *core*. The access layer is composed of a cluster of 10 base stations interconnected in a ring fashion. Among these 10 base stations, two of them is connected to the aggregation layer [17]. The aggregation layer is composed of k *pods*, each of which is composed of k switches connected in full-mesh. In each pod, $k/2$ switches are connected to $k/2$ base station clusters. The remaining $k/2$ switches are connected to $k/2$ switches residing in the core layer. The core layer is itself composed of k^2 switches connected in full-mesh.



(a) Effect of the number of service policy clauses (b) Effect of service policy clause length (c) Effect of network size

Figure 8: Large-scale simulation result. With multi-dimensional aggregation, SoftCell data plane is able to support thousands of service policy clauses on commodity switches.

Each core switch is furthermore connected to a gateway switch. The whole topology is composed of $10k^3/4$ base stations. For example, $k = 8$ (resp. $k = 20$) gives a network with 1280 (resp. 20000) base stations. For each topology, we assume that they are k different types of middleboxes. We randomly connect one instance of each type in each pod composing the aggregation layer and two instances of each type in the core layer. On top of this topology, we generate n policy paths for *each* base station to the gateway switch. A policy path traverses m randomly chosen middlebox instances. Finally, we measure the number of rules in each switch flow table. In the base case, we consider $n = 1000$, $m = 5$ and $k = 8$. We vary k, n and m to show how the switch state is affected by the number of service policy clauses, the policy length and the network size, respectively.

Effect of number of service policy clauses: Figure 8(a) shows the maximum and median size of the switch forwarding table with respect to the number of service policy clauses. We can see that switch table size increases linearly with the number of service policy clauses with a small slope (less than 2). In particular, to support 1000 service policy clauses (1.28 million policy paths!), switches store a median of 1214 rules and a maximum of 1697 rules. Even to support 8000 service policy clauses, the maximum table size is only 13682. Observe that in practice ISPs may only need tens or hundreds of service policy clauses, meaning that SoftCell can be easily implemented on commodity switches. The good performance of SoftCell data plane is a direct consequence of its multi-dimensional aggregation (see Section 3) capability. Indeed, even if a service policy clause instantiates a policy path to each base station, the corresponding forwarding entries can be aggregated by prefix in the core layer provided that they traverse the same middlebox instance like CS1 in Figure 3(c). Similarly, in the aggregation layer, the forwarding entries corresponding to paths traversing the same middlebox instance in a pod can be aggregated by prefix like CS2 and CS3 in Figure 3(c). As such, to install a new service policy clause, each switch only installs a handful of new rules in average.

Effect of service policy clause length: Figure 8(b) shows the switch table size with respect to the policy length. When the maximum service policy clause length is 8, the maximum switch table size is 1934. As before, we see that switch table size increases linearly with the length of the service policy clause with a small slope. Indeed, when a service policy clause is longer, the policy paths traverse more middleboxes and require more rules for forwarding. However, most affected switches on the path only need one additional rule to match on the tag; only a few switches are connected to multiple middleboxes and therefore need to dispatch traffic to multiple middlebox instances. Thus the switch table size increases slowly across all policy clauses. Again, observe that a service policy clause length of 8 (traversing 8 middleboxes) is an aggressive number, while in practice 4 or 5 is sufficient.

Effect of network size: Figure 8(c) shows the switch table size with respect to the network size. We see the table size decreases as the network grows. It is true that with more base stations, we have to install more policy paths for the same service policy clause, thus need more rules. But remember that we can do aggregation on policy tags and base station prefixes, and when the network increases, we have more switches. The increase of rules is small due to aggregation and all rules are distributed over the more switches. This leads to the result that when the network grows, switches maintain smaller tables for the same number of service policy clauses.

In summary, SoftCell can support thousands of service policy clauses in a network of thousands of base stations with a few thousand TCAM entries, which can be easily achieved by commodity switches. The gain essentially comes from the ability to selectively match on multiple dimensions.

7. DISCUSSION

Traffic initiated from the Internet: Although most traffic in cellular networks today are initiated from UEs, some carriers [24] also offer various public IP address options. When a gateway switch receives packets destined to these special public IP addresses, the gateway will act like an access switch. It will install packet classifiers that translate the public IP addresses and the port numbers (with which UEs provide service to the Internet) to LocIPs and policy tags. Note that these packet classifiers are not microflow rules and don't require communication with the central controller for every microflow. They are coarse grained (match on the UE public IPs and port numbers) and can be installed once.

Asymmetric Internet routing: For ease of description, we have assumed that flows leaving a gateway switch return to the same gateway switch. However, Internet routing is not guaranteed to be symmetric. If gateway switches are not border routers peering with other autonomous systems, border routers can be configured to route return traffic to the same gateway switch. Alternatively, the controller can install corresponding switch rules for return traffic in all possible gateway switches (mostly a small fraction of the total number of gateway switches).

Exact rule matching switches: Our design and evaluation of SoftCell has assumed that switches can do prefix-matching on IP address and port number. To extend SoftCell to handle exact rule matching switches, there are two cases. In the case we embed state in packet headers, SoftCell requires a special gateway switch that can copy location IP prefix and tag information from IP headers to fixed fields (if exists and not used for other purpose or append a header like MPLS) these switches can match. This is a very simple function (copy some bits from some fields to other fields) that doesn't need to store any state for execution and can be implemented in hardware with line speeds. In the case of caching state at gateway switches, our wild card rule can be in the control plane of the gateway switches. We can use mechanism like Devoflow [8] to install micro flow rules on demand.

On-path middleboxes: The only problem with on-path middleboxes is that it is unavoidable to traverse them in some cases. If service policy specifies that certain flows can not traverse certain middleboxes (which we have not considered in our service policy), then our path computation has to avoid these middleboxes. In case no feasible path exists, the policy path request will be denied.

Radio resource control state tracking, paging and roaming: Base stations keep track of UE Radio Resource Control State (RRC) state and the SoftCell controller keeps track of the current location area of a UE. Our handling of RRC state tracking and paging is in principle the same as current LTE. Roaming traffic are handled the same way as native traffic albeit with different service policy. How to obtain roaming subscriber information for authentication, and how to do billing etc are coordinated among controllers of carriers. We do not discuss the details in this paper.

8. RELATED WORK

Our quest is to build a scalable architecture to support fine-grained policies for mobile devices in cellular core networks. SoftCell differs from prior work on cellular network architecture, scalable data center, software defined networks, and middleboxes.

Cellular network architecture: Recently work has exposed the complexity and inflexibility of current cellular data networks [1, 2]. There are several efforts [1, 2, 25, 26, 27] attempting to fix the problem. However, only [27, 26] have concrete designs. OpenFlow Wireless [27] focuses on virtualizing data path and configuration. [26] proposes an integration of OpenFlow with LTE control plane so that GTP tunnels can be setup using OpenFlow. None of them present scalable network architecture for fine-grained policy.

Scalable data centers: Our addressing scheme shares some similarity to prior work on scalable data center. VL2 [28] assigns servers IP addresses that act as names alone. PortLand [29] assigns internal Pseudo MAC addresses to all end hosts to encode their position in the topology. Nicira [30]'s virtual data center networks require intelligent Internet gateways. Our gateway switches are much simpler because we "embed" policy and location information in the packet header, rather than relying on the controller to install fine-grain packet-classification rules.

Software defined networks: Recent work [8, 31] improves upon Ethane [4] to avoid maintaining per micro flow state in switches. DevoFlow [8] which handles most micro-flow rules in the data plane. DIFANE [31] distributes pre-installed OpenFlow wildcard rules among multiple switches and ensures all decisions can be made in the data-plane. Unlike SoftCell, they do not support policy policy symmetry and policy consistency. SoftCell architecture conforms to [32]. SoftCell distinguishes edge from core. The core routes on tags and IP prefixes that are different from the UE addresses. In addition, SoftCell differentiates access edge from gateway edge. SoftCell minimizes state kept at gateway switches.

Middleboxes: Prior work has focused on (1) middlebox design, e.g. a single box with modular capabilities that can implement a vast variety of services (for instance, see [33, 34]); (2) mechanisms to enforce middlebox traversals [35]. However, they do not present any scalable network architecture for fine-grained policy.

9. CONCLUSION

Today's cellular core networks are expensive and inflexible. In this paper, we propose SoftCell, a scalable architecture for supporting fine-grained policies in cellular core networks. SoftCell achieves scalability in the data plane by (i) pushing packet classification to low-bandwidth access switches and (ii) minimizing the state in core network through effective, multi-dimensional aggregation of forwarding rules. SoftCell achieves scalability in the control plane by caching packet classifiers and policy tags at local agents that update the rules in the access switches. We further design a modular controller that decouples high-level traffic management from the low-level details of computing and installing switch-level rules. Our prototype and evaluation demonstrate that SoftCell significantly improves the flexibility of future cellular core networks, while reducing cost through the use of commodity switches and middleboxes.

10. REFERENCES

[1] B.-j. Kim and P. Henry, "Directions for future cellular mobile network architecture," *First Monday*, vol. 17, no. 12, 2012.

[2] S. Elby, "Carrier vision of SDN and future applications to achieve a more agile mobile business," October 2012. Keynote address at the SDN & OpenFlow World Congress, `http://www.layer123.com/sdn-live`.

[3] "Network functions virtualization: Introductory white paper," October 2012. `http://www.tid.es/es/Documents/NFV_White_PaperV2.pdf`.

[4] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. Gude, N. McKeown, and S. Shenker, "Rethinking enterprise network control," *IEEE/ACM Trans. Networking*, vol. 17, August 2009.

[5] R. Wang, D. Butnariu, and J. Rexford, "OpenFlow-based server load balancing gone wild," in *Hot-ICE Workshop*, March 2011.

[6] B. Stephens, A. Cox, W. Felter, C. Dixon, and J. Carter, "PAST: Scalable Ethernet for data centers," in *ACM SIGCOMM CoNext Conference*, December 2012.

[7] "Open vSwitch." `http://openvswitch.org/`, 2013.

[8] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "DevoFlow: Scaling flow management for high-performance networks," in *ACM SIGCOMM*, August 2011.

[9] M. Howard, "Using carrier ethernet to backhaul LTE." `http://tinyurl.com/bdxl6wo`, 2011. Infonectics Research, White Paper.

[10] Alcatel-Lucent, "Alcatel-Lucent 9926 digital 2U eNode B."

[11] A. Rahmati, C. Shepard, C. Tossell, A. Nicoara, L. Zhong, P. Kortum, and J. Singh, "Seamless flow migration on smartphones without network support," *IEEE Transactions on Mobile Computing*, 2013. To appear.

[12] Y. Zhang and A. Arvidsson, "Understanding the characteristics of cellular data traffic," in *ACM SIGCOMM CellNet Workshop*, August 2012.

[13] L. Whitney, "Ericsson demos faster lte speeds of almost 1Gbps." `http://tinyurl.com/aml5vt`.

[14] "The rise of soft switching, part II: Soft switching is awesome," June 2012. `http://tinyurl.com/bjz8469`.

[15] C. Rotsos, N. Sarrar, S. Uhlig, R. Sherwood, and A. W. Moore, "OFLOPS: An open framework for OpenFlow switch evaluation," in *Workshop on Passive and Active Measurement*, March 2012.

[16] Z. Wang, Z. Qian, Q. Xu, Z. Mao, and M. Zhang, "An untold story of middleboxes in cellular networks," in *ACM SIGCOMM*, August 2011.

[17] R. Nadiv and T. Naveh, "Wireless backhaul topologies: Analyzing backhaul topology strategies," *Ceragon White Paper*, 2010.

[18] "Controller performance comparisons," May 2011. `http://www.openflow.org/wk/index.php/Controller_Performance_Comparisons`.

[19] R. Ramjee, K. Varadhan, L. Salgarelli, S. Thuel, S.-Y. Wang, and T. La Porta, "Hawaii: A domain-based approach for supporting mobility in wide-area wireless networks," *IEEE/ACM Trans. Networking*, vol. 10, June 2002.

[20] A. Campbell, J. Gomez, and A. Valko, "An overview of cellular IP," in *IEEE Wireless Communications and Networking Conference*, 1999.

[21] Cisco, "Cisco visual networking index forecast projects 18-fold growth in global mobile internet data traffic from 2011 to 2016." `http://tinyurl.com/7gn9x9s`.

[22] "Floodlight OpenFlow Controller." `http://floodlight.openflowhub.org/`.

[23] "Cbench OpenFlow Controller Benchmark." `http://www.openflow.org/wk/index.php/Oflops`.

[24] AT&T, "Wireless IP options for mobile deployments." `https://www.wireless.att.com/businesscenter/solutions/connectivity/ip-addressing.jsp`.

[25] L. Li, Z. Mao, and J. Rexford, "Toward software-defined cellular networks," in *European Workshop on Software Defined Networking (EWSDN)*, October 2012.

[26] J. Kempf, B. Johansson, S. Pettersson, H. Luning, and T. Nilsson, "Moving the mobile evolved packet core to the cloud," in *IEEE WiMob*, October 2012.

[27] K.-K. Yap, R. Sherwood, M. Kobayashi, T.-Y. Huang, M. Chan, N. Handigol, N. McKeown, and G. Parulkar, "Blueprint for introducing innovation into wireless mobile networks," in *ACM VISA Workshop*, August 2010.

[28] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, "VL2: A scalable and flexible data center network," in *ACM SIGCOMM*, August 2009.

[29] R. Niranjan Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat, "PortLand: a scalable fault-tolerant layer 2 data center network fabric," in *ACM SIGCOMM*, August 2009.

[30] Nicira, "It's time to virtualize the network: Network virtualization for cloud data centers." `http://tinyurl.com/c9jbkuu`.

[31] M. Yu, J. Rexford, M. J. Freedman, and J. Wang, "Scalable flow-based networking with DIFANE," in *ACM SIGCOMM*, August 2010.

[32] B. Raghavan, M. Casado, T. Koponen, S. Ratnasamy, A. Ghodsi, and S. Shenker, "Software-defined Internet architecture: Decoupling architecture from infrastructure," in *ACM SIGCOMM HotNets Workshop*, October 2012.

[33] A. Gember, P. Prabhu, Z. Ghadiyali, and A. Akella, "Toward software-defined middlebox networking," in *ACM SIGCOMM HotNets Workshop*, 2012.

[34] V. Sekar, S. Ratnasamy, M. K. Reiter, N. Egi, and G. Shi, "The middlebox manifesto: Enabling innovation in middlebox deployment," in *ACM SIGCOMM HotNets Workshop*, 2011.

[35] D. Joseph, A. Tavakoli, and I. Stoica, "A policy-aware switching layer for data centers," in *ACM SIGCOMM*, August 2008.