SoftMoW: Recursive and Reconfigurable Cellular WAN Architecture

Mehrdad Moradi† Wenfei Wu ‡ * Li Erran Li⋆ Z. Morley Mao†
Bell Lab, Alcatel-Lucent⋆ University of Michigan–Ann Arbor† University of Wisconsin–Madison‡

ABSTRACT

The current LTE network architecture is organized into very large regions, each having a core network and a radio access network. The core network contains an Internet edge comprised of packet data network gateways (PGWs). The radio network consists of only base stations. There are minimal interactions among regions other than interference management at the edge. The current architecture has several problems. First, mobile application performance is seriously impacted by the lack of Internet egress points per region. Second, the continued exponential growth of mobile traffic puts tremendous pressure on the scalability of PGWs. Third, the fast growth of signaling traffic known as the signaling storm problem poses a major challenge to the scalability of the control plane. To address these problems, we present SoftMoW, a recursive and reconfigurable cellular WAN architecture that supports seamlessly inter-connected core networks, reconfigurable control plane, and global optimization.

To scale the control plane nation-wide, SoftMoW recursively builds up the hierarchical control plane with novel abstractions of both control plane and data plane entities. To enable scalable end-to-end path setup, SoftMoW presents a novel label swapping mechanism such that each controller only operates on its logical topology and each switch along the path only sees at most one label. SoftMoW supports new network-wide optimization functions such as optimal routing and inter-region handover minimization. We demonstrate that SoftMoW improves the performance, flexibility and scalability of cellular WAN using real LTE network traces with thousands of base stations and millions of subscribers. Our evaluation shows that path inflation and inter-region handovers can be reduced by up to 60% and 44% respectively.

Categories and Subject Descriptors

C.2.1 [**Network Architecture and Design**]: Distributed networks; C.2.5 [**Local and Wide-Area Networks**]: Applications

*This work was performed when Wenfei was an intern at Bell Labs and was supported by NSF grant CNS 1218668.

CoNEXT'14, December 02-05 2014, Sydney, Australia

http://dx.doi.org/10.1145/2674005.2674981.

Keywords

Cellular Networks; Software-Defined Networking; Hierarchical Controllers

1. INTRODUCTION

The current LTE network architecture is organized into very large and rigid regions. Each large region has a core network and a radio access network. The core network contains an Internet edge comprised of packet data network gateways (PGWs). The radio network consists of only base stations. In this architecture, there are minimal control plane and data plane interactions among regions other than distributed interference management at radio access networks and limited coordination for mobility (e.g. no inter-PGW mobility [25]). All users' outgoing traffic must traverse a PGW and possibly go through the Internet.

This rigid architecture is becoming harder and harder to support new trends of mobile traffic. First, mobile application performance is seriously impacted by the lack of Internet egress points per region. Specifically, as shown by a recent study [24], the lack of sufficiently close Internet egress points is a major cause of path inflation, suboptimal routing, and QoS degradation in large operators. Second, the continued exponential growth of mobile traffic puts tremendous pressure on the scalability of PGWs. Third, the fast growth of signaling traffic known as the signaling storm problem [4] poses a major challenge to the scalability of the control plane.

Rather than organizing mobile wide area networks as rigid regions with no direct traffic transit, we argue that cellular networks should have a seamlessly *inter-connected core network* with a logically centralized control plane. The interconnected core network should consist of a fabric of simple core switches and a distributed set of middleboxes (software or hardware). The control plane directs traffic through efficient network paths that might cross region boundaries rather than exiting to the Internet directly from the origin region. The control plane should also globally support seamless UE mobility and optimize the performance of mobile traffic. For example, mobile traffic routing should be globally optimized; regions should be reconfigured to adapt to its workload.

Such an architecture raises unique challenges in scalability in comparison with data-center networks [12, 20] and inter-data center WANs [16, 15] since the cellular WAN has its own unique properties and challenges. First, the logically centralized control plane needs to control tens of thousands of switches and middleboxes, and hundreds of thousands of base stations in the data plane. A control plane with many controller instances in one data center cannot effectively han-

dle the signaling load (e.g., connection setups and handovers) from hundreds of millions of subscribers distributed throughout a continent. Second, without global network states and a single controller exerting control, it will be hard to perform network wide routing optimization and inter-region handover minimization.

To address these problems, we present SoftMoW, a scalable network-wide control plane that supports global optimization and control plane reconfiguration. SoftMoW makes the following contributions.

- First, SoftMoW *recursively* builds up the hierarchical control plane with novel abstractions consisting of both control plane and data plane entities. Key to our SoftMoW architecture is the controller. It is designed to be *modular* which consists of the network operating system (NOS), operator applications and the recursive abstraction application (RecA). NOS provides core services such as routing and path implementation. NOS does not handle cellular specific functions. Operator specific functions (e.g. mobility management) are implemented as applications on top of NOS. All recursive abstraction functions are implemented in RecA.
- Second, to enable scalable end-to-end path setup, SoftMoW presents a *novel label swapping mechanism* such that each controller only operates on its logical topology and each switch along a flow's path only sees at most one label. This new mechanism reduces the states in the switches.
- Third, SoftMoW designs new network-wide optimization functions such as optimal routing and region optimization to minimize inter-region handover.
- Fourth, we demonstrate that SoftMoW improves the performance, flexibility and scalability of cellular WAN using real LTE network traces with thousands of base stations and millions of subscribers. Our evaluation shows that path inflation and inter-region handovers can be reduced by up to 60% and 44% respectively.

2. SoftMoW DESIGN OVERVIEW

SoftMoW's goal is to design a scalable cellular WAN architecture (both the control plane and data plane) to enable network-wide optimizations. We introduce the components in a SoftMoW network, the design challenges and our solutions.

2.1 SoftMoW Components

SoftMoW does not require expensive, inflexible and specialized devices (e.g., PGWs and SGWs) that integrate control and data plane operations with middlebox functions. SoftMoW does not change the LTE protocols used in the user equipment (UE) and the protocols between UE and base stations. SoftMoW has the following high-level architectural components.

Nation-wide inter-connected core networks. SoftMoW distributes and inter-connects programmable switches nation-wide. The network in one region should have enough egress points through a subset of the switches. An egress point can connect to other regions of the same carrier, other carriers' mobile networks, Internet service providers or content providers at peering points to exchange traffic. This eliminates the internal path inflation problem caused by the lack of sufficiently close egress points and enhances end-to-end QoS metrics by offering better diversity of external paths.

Radio access networks. Radio access networks consist of base stations which are organized and inter-connected into base station group (BS group) with different topologies (e.g., ring, mesh, and spoke-hub) to ensure intra-BS-group fast-path communications. BS groups are connected to core network switches locally. We assume each base station has an *access switch* performing fine-grained packet classifications on traffic from UEs.

Middleboxes and service policies. SoftMoW departs from the centralized policy enforcement at PGWs and utilizes middleboxes which can be flexibly placed throughout the cellular WAN. For scalability, middlebox functions will be mostly limited to edge networks of the cellular WAN. Middlebox instances can potentially implement any sophisticated network functions. The functions can be specific to application types (e.g., noise cancellation function and video transcoding function) and operators (e.g., charging and billing), and security (e.g., firewall, and IDS). A service policy is then met by directing traffic through a *partially ordered set* (also known as poset) of middlebox types. Given the location and utilization of middlebox instances, the controller can implement a poset using various combinations of physical instances.

Controller. The controller enforces a rich set of service policies on subscribers' network access through new global network applications. These applications are based on a global view of the inter-connected core networks, which are not available in current LTE networks or recently proposed cellular architectures such as SoftCell [23]. Specifically, the controller sets up end-to-end optimal paths for aggregate flows and minimizes the number of inter region handovers.

2.2 Design Challenges and Solutions

Challenge 1: scalable control plane. The logically centralized control plane needs to control tens of thousands of switches and middleboxes, hundreds of thousands of base stations in the data plane. A control plane with many controller instances in one data center (e.g., [17, 6]) will not effectively handle the signaling loads (e.g., connection setups and handover events) from hundreds of millions of subscribers distributed throughout a continent. Also, a flat decentralized architecture where local controllers only communicate with their neighbors (e.g., [22]) is not scalable enough to support fast and global optimizations. It requires distributed algorithms that involve many rounds of message exchanges.

Solution: recursively build up a hierarchical and reconfigurable control plane. SoftMoW hierarchically constructs a *network-wide control plane* that is *reconfigured* in response to the signaling loads and traffic patterns. The control plane consists of geographically distributed controllers that are organized into a tree structure. Recursively from the leaf level, each controller (except the root) exposes a small number of logical and reconfigurable data plane entities to its immediate parent. These entities aggregate many switches, middleboxes and base stations. To enable global optimization such as routing optimization by ancestor controllers, the exposed logical switches and their interconnections are described as a *virtual fabric* with annotated bandwidth, latency, and hop count information.

Challenge 2: scalable end-to-end path implementation. Our cellular WAN provides connections between millions of UEs and thousands of Internet egress points, the number of routing states in the core network switches is

tremendous. One way to implement the routes is to aggregate flows traversing the same path, assign them one label and route on labels (e.g. MPLS). In a decentralized flat control plane, implementing a label-switched path involves all controllers and switches on the path. To do this, each controller has to know the global state. Keeping entire data plane states consistent at each controller or storing them into a central data base is not scalable. In SoftMoW, each controller has a limited summarized view over a set of logical entities to improve scalability, but this makes the state management and path implementation more challenging.

Solution: scalable recursive label swapping. SoftMoW leverages its tree structured control plane architecture. Using a novel recursive label swapping approach, SoftMoW implements end-to-end paths while keeping per packet overhead minimal. An ancestor controller pushes labels onto packets of matching flows traversing its logical and reconfigurable switches. Recursively, these labels will be replaced with local labels by each lower level controllers. At the physical data plane switches, only a local label is pushed onto packets of matching flows, which each represents a local regional path segment. When packets leaving a region, the local label is popped off and an ancestor's label is pushed.

Challenge 3: scalable topology discovery and maintenance. Topology discovery is easy in flat multi-controller settings. Each switch is controlled by one controller instance. A controller sends discovery messages from all ports of registered switches. When a switch receives a discovery message, it forwards the message to the controller. The controller then maintains the link between the source and destination switches and stores link-specific information (e.g., port name, link capacity). In SoftMoW, detecting links is more challenging because each cross-region link is visible to only one non-leaf controller; the non-leaf controller needs to discover it without breaking the abstraction.

Solution: recursive discovery protocol. We design and introduce a novel global discovery protocol allowing recursive discovery of topologies by each controller. Each leaf controller first discovers its own physical topology. Then the parent controller is exposed with a logical topology and can discover the cross-region links it controls. This process continues until the root controller discovers its topology. Controllers at the same level can perform discovery at the same time in parallel. The sequential process only applies to the bootstrapping phase. During normal operations, periodical discovery messages will be carried out concurrently.

Challenge 4: network-wide optimization. SoftMoW's goal is to enable global optimizations for control plane and data plane functions such as optimal routing and inter-region handover minimization. Maintaining and performing optimization with global network states for a country-wide network is not scalable.

Solution: design algorithms on abstract topologies of hierarchical controllers. SoftMoW supports global network optimization without a global network state at each controller. We demonstrate this feature using two important network functions. First, application traffic may have its own requirements on the path (e.g. low-latency path for delay-sensitive VoIP). In SoftMoW, the path is computed by controllers from the leaf to the root. If a local optimal path meeting the application requirements is found, it is used without further delegating to ancestor controllers. We show that the root controller is guaranteed to find an optimal path in terms of performance metrics (e.g., latency and hop count). Second, an inter-region handover requires the involvement of an ancestor controller, the source controller and destination controller. In this procedure, new paths have to be implemented and in-flight packets have to be diverted to the target base station. To minimize control plane load, SoftMoW performs inter region handover optimization. The optimization is done from the root controller to leaf controllers. We show that the process converges if handover traffic pattern does not change during the optimization.

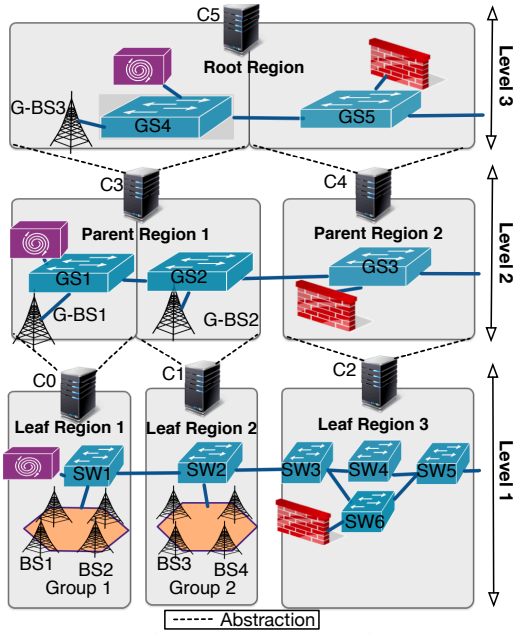


Figure 1: A 3-level SoftMoW Architecture

3. SoftMoW CONTROL PLANE

We first give an overview of how we recursively construct the control plane and the logical data plane, and present the design of the controller architecture.

3.1 Recursive Constructions

As shown in Figure 1, SoftMoW hierarchically builds a reconfigurable network-wide control plane. The control plane consists of geographically distributed controllers that are organized into a tree structure, each controller associated with a level number and a globally unique ID. The topmost node is the *root* controller which can make coarse-grained decisions for the entire network, and level 1 nodes are *leaf* controllers close to the physical data plane. The number of levels, the number of children per node, and the geographical location of each node can be determined based on fine-grained latency budgets of control functions [3] as well as the density and size of the physical topology.

SoftMoW partitions the physical data plane network into logical regions whose borders can change over time based on traffic and failure patterns. Each leaf region is managed by a leaf controller. In Figure 1, leaf controllers (level-1) discover their physical switches and build the level-1 data plane, and they also abstract some entities for level-2 controllers. The level-2 controllers obtain logical network entities from the leaf controllers, discover their logical level-2 data plane and also make logical network entities. Finally, the root controller (level-3) obtains logical network entities from the

level-2 controllers and builds the level-3 data plane. When building these data planes recursively from the leaf level, each controller simplifies its topology and exposes the following three types of *logical* and *reconfigurable* data plane entities to its parent.

- **Gigantic Switch (G-switch)** aggregates a number of physical or gigantic switches and the controller. A G-switch is programmable and characterized by an ID, ports, and a virtual fabric (will be clear in Section 3.2) and flow table. Each port of a G-switch corresponds to border ports of its constituent switches, i.e. is connected to either Internet domains (e.g., ISP) or neighboring regions.
- **Gigantic Middlebox (G-middlebox)** hides physical or G-middlebox instances of the same type and function (e.g., light weight DPI) and their controller. A G-middlebox can be attached to G-switches, and is identified with the sum of the processing capacities and utilization of constituent instances.
- **Gigantic Base station (G-BS)** summarizes one or more adjacent BS groups or G-BSes, and their controller. A G-BS inherits the union of the radio coverage of underlying base stations and connects to ports of a G-switch.

Abstracting the logical region for the parent. To build the first logical data plane (level 2), each leaf controller builds and exposes a single G-switch for all switches, a G-middlebox for all middlebox instances of the same type, a G-BS for one or more adjacent BS groups (will be clear in Section 5.2). Intuitively, a parent's logical region is the union of regions exposed from its children in the tree. Recursively, non-leaf controllers except the root (e.g., level-2 controllers in Figure 1) perform the same procedure on G-middleboxes, G-switches, and G-BSes located in their logical region.

Reconfiguration of logical data plane devices. Each non-leaf controller can reconfigure logical entities exposed from its children. This gives each controller the ability to optimize its descendants' control plane hierarchy and data plane operations without a global state, solely based on its partial view and abstract topology. Any non-leaf controller can initiate a reconfiguration that indirectly causes controllers in its subtree to level-by-level from bottom-to-top interact with each other to modify the exposed logical entities. This new feature enables interesting global applications such as minimizing "east-west" control load in the cross-controller handovers (see Section 5).

3.2 G-Switch Virtual Fabric

To enable global optimization, e.g., traffic engineering and optimal routing, each controller in the tree hierarchy should know a few pieces information about the internal inter-connections behind its G-switches. SoftMoW exposes a virtual switch fabric for each G-switch. A virtual switch fabric (vFabric) is a succinct representation allowing the parent controller to have three pieces of information per G-switch port pair:*latency*, *hop count*, and *available bandwidth*.

Using standard shortest path algorithms, each child controller constructs these metrics by computing multiple shortest paths for each port pair in its topology. Note that, for the bandwidth metric, different port pairs of the G-switch can share bottleneck links. In this case, if the available bandwidth exposed for a port pair in the child controller's data plane changes more than a predetermined threshold, the child controller will recompute new bandwidths, update the vFabric and notify the parent controller.

3.3 Controller Architecture

We design a modular controller architecture as shown in Figure 2. A SoftMoW controller consists of a network operating system (NOS), operator applications and an application called RecA that implements the recursive abstraction.

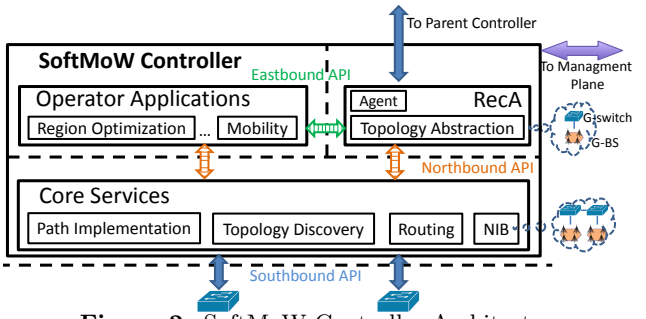


Figure 2: SoftMoW Controller Architecture

Network operating system. SoftMoW expects a number of core services: *path implementation*, *topology discovery*, *routing* and *network information base (NIB) query*. SoftMoW NOS can reuse any existing controller platforms that expose these services through a northbound API. SoftMoW NOS is agnostic of cellular specific functions and other controllers in the hierarchy. NOS communicates with switches (logical or physical) using a southbound API, e.g. OpenFlow API extended to support our virtual fabric feature.

Operator applications. SoftMoW cellular specific functions are implemented as operator applications on top of the NOS, e.g. functions similar to LTE such as home subscriber server (HSS), policy charging and rule functions (PCRF), mobility and new functions such as region optimization and routing optimization. Applications can use the northbound API to get network information (e.g. topology) and set up their configurations (e.g. path setup, sending messages).

Recursive abstraction application (RecA). To implement the recursive abstraction, we design a NOS application called RecA. RecA encapsulates all functions related to the recursive abstraction and provides an eastbound API for operator applications. RecA has two basic modules: *agent* and *topology abstraction*. RecA's topology abstraction module queries the NIB using the NOS northbound API. It abstracts a network topology (including switches, base stations and middleboxes) as one G-switch, a number of G-BSes (border BS groups need to be exposed in a specific way, will be clear in Section 5) and one G-middlebox of each type. The RecA agent communicates with a parent controller (if any). For each logical device, the agent establishes a channel to its parent controller. This way logical devices act as physical ones (e.g., a G-switch acts as a physical switch).

RecA provides the eastbound API to other operator applications. An operator application can register its message type in RecA, and give messages that it cannot handle to RecA; then RecA will send the message to its parent controller as a Packet-In event. The agent also handles messages from the parent. If a message is about path implementation, the agent sends it to the topology abstraction module, which translates the message to multiple messages using the current network view of NIB; if the message is of a type registered by an operator application, it is sent to the application. RecA and operator applications use the northbound API to send messages to logical (child controllers) and physical data plane entities.

Management plane. The management plane bootstraps the recursive control plane. It configures all controllers in the hierarchy via dedicated channels (e.g. assigns IP addresses, and region identifier, and configures the tree structure). The RecA at each controller exports its topology to the management plane. The region optimization applications communicate with the management plane to reconfigure local or physical network devices. The management plane also coordinates UE state transfer during region optimization.

4. CORE SERVICES

SoftMoW core services provided by the network operating system includes the NIB, topology discovery, routing and path implementation. Similar to the NIB in other controller designs [17], SoftMoW's NIB consists of network devices, device type (e.g. base station, middlebox, switch), links and their metrics. We assume standard mechanisms (e.g. those in [17]) to gather NIB and maintain NIB's consistency. The NOS has visibility of its own local network topology (physical or logical), does not maintain UE state, is not aware of any ancestor or descendant controllers (may communicate with peer controllers). Now we proceed to present the other three core services.

4.1 Recursive Topology Discovery

SoftMoW presents the first topology discovery protocol in a recursively built control plane architecture. Topology discovery in SoftMoW is much more challenging than in flat architectures. This is because only leaf controllers have direct control over physical switches. Yet each inter G-switch link is physical and is only visible to the ancestor controller of both endpoints of the link.

In SoftMoW, each controller discovers a subset of total links of the physical topology. Data plane switches and links (logical and physical) are discovered sequentially from bottom to top; controllers at each level can discover their (inter G-switch) links in parallel. We now proceed to describe the procedures of topology discovery: *G-switch discovery*, *inter G-switch link discovery* and *Computation of G-switch abstraction*. These procedures are performed by RecA and the topology discovery module. Base stations, middleboxes and links with them as endpoints can also be discovered similarly. If base stations and middleboxes do not implement our discovery protocol, they can also be configured by the management plane.

4.1.1 G-switch Discovery

Similar to physical switches, the RecA agent of each non-root controller connects to the parent controller. After a controller starts, its topology discovery module first discovers all switches (G-switches or physical switches) in its region. If the switch type is G-switch, the controller also performs a feature request to obtain the virtual fabric information. The G-switch device information is stored in NIB. The controllers use the southbound API (e.g., Openflow) to get the G-switch information.

4.1.2 Inter G-switch Link Discovery

Link discovery message. After G-switch discovery, a controller uses inter G-switch Link Discovery Protocol to find the links between its G-switches. For each G-switch port, it initiates a link discovery message, which has a meta data field and a stack field. The link discovery message traverses through the controller hierarchy down to the physical data plane, goes through a physical link, and is reported from the receiving switch back to its origin along the controller hierarchy. The meta data field carries the properties of the traversed physical link (e.g., latency and loss rate), which is filled by the leaf controller on its path. The stack stores the traversed path in the controller hierarchy with the format of (Controller ID, G-switch ID, G-switch port).

Origination path. In more detail, when the topology discovery module in a controller discovers inter G-switch links, link discovery messages are sent out from each port of G-switches (which is actually received by the corresponding child controller). Intuitively, the link discovery message recursively is passed to lower-level child controllers and finally sent out of a port of a physical switch. The initiating controller pushes its ID, the G-switch ID and the port onto the stack. When the RecA agent of a child controller receives the message from its parent, the message is forwarded to the RecA topology abstraction module. This module extracts the G-switch and port from the top of the stack, and maps them to one of its G-switches and its port. Then, RecA pushes its ID, the G-switch ID and port onto the stack. If the controller is a leaf controller, it also encodes meta data of the physical link into the meta data field. RecA calls the northbound API SendMsg(switch, port, msg) to send the message.

Return path. Both topology discovery module and RecA register the link discovery Packet-In message from the lower-level. When a controller receives a link discovery message from one of its G-switches with an incoming port. It pops the stack to get the (Controller ID, G-switch ID, G-switch port). In the topology discovery module, if the popped controller ID is its ID, the link discovery message has been originated by itself, so a new inter G-switch link is discovered. This inter G-switch link is added to the NIB of the current controller. In RecA, if the controller ID is not its ID and the stack is not empty, the link discovery message is reported to the parent by the RecA agent; if the stack is empty, the link discovery message is dropped indicating the link discovery message can not return to the initiating controller and there is no inter G-switch link on the path.

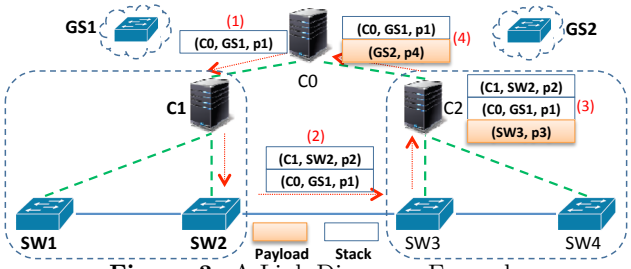


Figure 3: A Link Discovery Example

Example. Figure 3 shows an example of inter G-switch link discovery. The root controller intends to discover the link between G-switch GS1 and GS2 on its logical data plane. The link discovery protocol finishes in 4 steps. (i) The root controller C0 initiates a link discovery message. It populates the stack with its own ID C0 and the G-switch ID GS1 and port number p1. (ii) The child controller C1 receives the link discovery message. It translates the G-switch ID and port number into the physical switch ID SW2 and port number p2. Then, C1 pushes (C1, SW2, p2) onto the stack. (iii) Physical switch SW3 receives the message at port p3 and

passes it to its controller C2. C2 encodes the receiving (SW3, p3) into link discovery message. C2 pops the stack and find the controller ID at the top of the stack is C1, which is not its ID. So it translates the (SW3, p3) to corresponding ID and port number of its abstract G-switch, which is (GS2, p4), and passes the link discovery message to its parent C0. (iv) C0 pops the stack and find the controller ID at the top of the stack is its ID. In this way, it finds the inter G-switch link between its G-switches (i.e. GS1 and GS2).

4.1.3 *Computation of G-switch Abstraction*

The RecA application in a controller uses the northbound API topo=GetTopology() to get its G-switches and inter G-switch links, and then it computes one abstract G-switch. In an abstract G-switch, all internal ports (i.e. ports between G-switches) are hidden, and all border ports are exposed. SoftMoW also computes other properties between G-switch port pairs, such as latency, bandwidth and hop count as discussed in Section 3.2. The parent controller requests the G-switch features (e.g. virtual fabric) from the RecA agent in the child controller via the southbound API. G-BS and G-middleboxes can also be computed similarly. We do not go into the details in this paper.

4.2 Route Computation

SoftMoW must provide UEs with Internet access. The routing service computes end-to-end optimal paths through the northbound API (path, match fields)=Routing(request, service policy). The inputs are a routing request and a service policy. The outputs are a computed path and match fields to classify the flow. The computed paths are implemented using the path implementation service.

Interdomain routes. To perform routing, SoftMoW interacts with ISPs and content providers through an interdomain routing protocol (e.g., BGP) at egress points. Similar to a RCP server [7], leaf controllers run the route selection procedure on behalf of their gateway switches, each keeping a session with an eBGP speaking router in a neighbor ISP. For each gateway switch, leaf controllers select interdomain routes for all prefixes. In addition, the network performance of each selected route is measured (e.g., hops, latency) [24]. Leaf controllers forward the selected routes to their parent as Packet-In messages, each is associated with performance metrics. The routing module in each controller registers for interdomain routing messages, and puts them into NIB. Recursively, the RecA agent reads the interdomain routes from NIB and sends it to the parent (with translation to the G-switch). This procedure finishes once the root receives interdomain routes from its G-switches.

Recursive routing. When a controller has a routing request from one of its operator's applications (e.g., bearer request), it first checks if its logical region has an interdomain route to the destination on the Internet and the end-to-end internal path satisfies the performance constraints if specified in the request (e.g., latency). In addition, it checks whether the middlebox poset can be met in its logical region if specified in the service policy field. If so, the routing module returns the path and match fields, and then the application implements the path. If no path is found, the operator application delegates the request to RecA agent, which creates a routing request and sends to the parent, where the application in the parent controller registers in the core to get the message and process it. The application also registers for the response in RecA (e.g., to store in local caches). The delegation procedure increases the chance of satisfying the request since the parent has a better global view due to having a larger logical region. We will explain the routing service usage in handling bearer requests (Section 5.1).

Optimality discussion. Each controller might need to compute internal paths to interdomain routes through its own egress points. Using the virtual fabric of G-switches, the routing service can find a shortest path between the logical or physical gateway switches and base stations. We can guarantee a shortest path computed by a controller is the shortest is in the controller's region and its corresponding physical topology. We call such paths **locally optimal**. However, the shortest path in a controller's region may not be the global shortest path in the entire topology. We define shortest paths computed by the root controller in its global abstract topology as **globally optimal**. In general, a controller at a higher level is able to compute more optimal paths compared to any controller in its subtree.

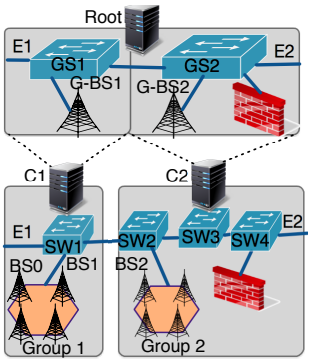


Figure 4: Local Optimal v.s. Global Optimal

Example. Using the interdomain routing messages, we know egress points E1 and E2 are 10 hops away from the address prefix A in Figure 4. The leaf controller C2 receives routing request (BS group= Group 2, destination=A) with the constraint of the maximum end-to-end hop count of 14. C2 computes the shortest path (SW2, SW3, SW4) going through E2 since it satisfies the performance requirement. This path is a local optimal path in C2's region. With the global network view, path (SW2, SW1) is one hop closer to the destination. The virtual fabric of a G-switch contains performance metrics for all port pairs. The root has the virtual fabric of G-switches GS1 and GS2, so it can easily compute the globally optimal path exiting from GS1.

4.3 Global Path Implementation

In SDN architectures where a controller has full visibility of its physical data plane topology [16, 15], path setup is straight-forward. The controller installs a match-action rule on each switch along the path. The match-action rule can match IP prefixes, VLAN tags, MPLS labels or some combinations of them. In SoftMoW, a controller aggregates flows on the same path, assigns them the same label and sets up routing on labels. So the states in switches can be significantly reduced. However, non-leaf controllers do not have full visibility of the physical data plane topology. We present a scalable mechanism that enables non-leaf controllers to implement paths in their abstract topology onto the underlying physical data plane. A northbound API PathSetup(match fields, path) is provided to applications to set up an input path with certain match fields.

In SoftMoW, a leaf controller can simply implement any intra-region paths. Similar to SoftCell [23], the access switch of base stations can perform fine-grained packet classification and push labels onto packets matching flow rules. Then, switches along the path are programmed to forward traffic based on specified labels. A non-leaf controller does not have control over physical switches, and multiple descendant controllers make partial forwarding decisions; so its path setup is more challenging. Similar to leaf controllers, a non-leaf controller should be able to instruct the access G-switch attached to each G-BS to classify packets and push virtual labels into the traffic, and program its G-switches along any desired path to operate based on pushed virtual labels.

To implement this operation, intuitively, when RecA agent in each child controller receives virtual label switching or packet classification rules, it translates them using its own topology. Each virtual label switching rule is mapped onto internal paths between the egress and ingress ports of the child controller's logical region, and the path computation is performed by the routing module. During the recursive translations, descendant controllers can establish any desired number of internal shortest paths between the ingress and egress points as long as the performance metrics of computed paths comply with the parent's virtual fabric. A descendant controller should be able to push a separate local label on top of the parent's label to establish each local path. Accordingly, the classification rule should be updated for each local path and installed into constituent access switches, each attached to a component G-BS.

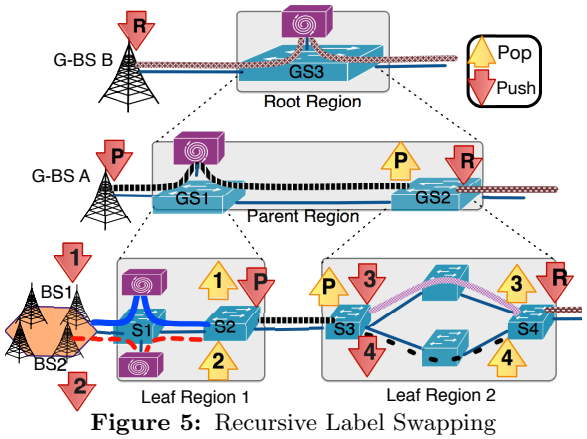


Figure 5: Recursive Label Swapping

High-overhead label stacking. To implement the recursive translations of virtual rules onto physical switches in underlying topology, a simple approach is to recursively stack k labels in the packets where k is the level of the controller initiating the path setup. Label stacking allows a label specified by an ancestor controller to be visible and available in the packets traversing across physical inter-G-switch links detected by the controller itself. Label stacking approach gives the illusion of packets traversing through the region of controllers at different level. When traffic enters a logical region at any level, the controller reads the label in the stack at the same level. This approach is not scalable in nation-wide mobile networks since it increases per-packet overhead due to encapsulating k labels in each packet, which exacerbates the bandwidth consumption as the number of levels in the SoftMoW architecture increases.

Label stacking example. Figure 5 shows logical regions of two leaf controllers, their parent, and the root controller (controllers are excluded for simplicity). The root has a single-path service policy for rate-limiting bidirectional traffic between G-BS B and a destination address prefix. To satisfy this policy, the root pushes label R at access switch of G-BS B and then installs the corresponding virtual rule into G-switch GS3 to forward traffic specified by label R. At the level below, the parent controller receives the rules. Based on its local view, it decides to stack label P on R (i.e., pushes [R P]) onto the packets. It programs the G-switches GS1 and GS2 to process incoming traffic with label P. In this approach, leaf controller 1 should at *least* push the stack [R P] onto each packet at the base stations. This allows leaf region 2 to read P from the stack and perform the forwarding. Then the rest of the network reads label R of the egress traffic from region 2. Intuitively, this gives the illusion of packets traversing up to the parent region at S2, and traversing down at S3. Also, the packet traverses up to the root level at S4. It is easy to imagine an increase in the packet header space and network bandwidth consumption, as SoftMoW levels increases, due to stacking multiple labels in packets.

Scalable recursive label swapping. We propose a novel *recursive label swapping mechanism* eliminating the high bandwidth overhead per-packet. In our approach, each packet has only one label at any given time. We have observed that a label specified by a non-leaf controller only needs to be visible across physical inter G-switch links detected by the controller itself. Thus we instruct controllers to perform *label pop* and *label push* operations. Each controller at the ingress switch (physical or gigantic) of its logical region pops the label (specified by an ancestor who controls the just traversed link) of the traffic. It then pushes an internal label corresponding to each internal path. Finally, it programs switches along each path. At the egress switch of its logical region, the controller aggregates the internal paths by popping their label. It then pushes back the ancestor's label onto packets of the flow. This mechanism guarantees the global coordination between the controller by having the necessary label at each switch while it minimizes the bandwidth overhead.

Recursive label swapping example. In Figure 5, the root adds label R to the traffic group at access switch of G-BS A similar to the previous example. It then programs G-switch GS3 to forward traffic based on label R to the rest of network. In this step, the controller of parent region receives the classification and forwarding rules. Using the push operation, it only pushes its local label P due to the local preference and does not mark the traffic with label R. Using the pop operation, it pops P and pushes backs the root's label R at G-switch GS2 where it loses its control on the egress traffic.

In the leaf region 1, the leaf controller decides to load balance the packets between two rate limiters, so it implements two local paths with label 1 and 2. With the push operation, it pushes label 1 and 2 at access switches of BS1 and BS2 respectively. With the pop operation, these two labels are replaced with the parent's label P at egress switch S2, so the next leaf region can process the traffic. In the leaf region 2, switch S3 is programmed to perform load balancing on ingress traffic from region 1. The leaf controller implements two separate paths by pushing local labels 3 and 4, and popping P at switch S3. These paths are aggregated at egress switch S4. The local labels are popped off and the root's label R is pushed back onto the packets. As shown in

the physical data plane, packets always carry a single label denoted with different patterns while many controllers make partial decisions.

5. OPERATOR APPLICATIONS

A key cellular network function is mobility management which includes setting up bearers (a bearer provides network connectivity service to the UE) and handovers. Mobility management is performed by the Mobility Management Entity (MME) in LTE whereas it is done by the mobility application in SoftMoW. The key differences are: (1) mobility application is simpler because of the use of the controller's northbound API which is not available in LTE; (2) it supports mobility better (e.g., LTE does not support inter-PGW handovers [25]). LTE mobility management has many procedures, due to the lack of space, we only discuss main functions that highlight the differences. Besides the mobility management, we present a new application, the region optimization application, to reduce the handover load of the controllers.

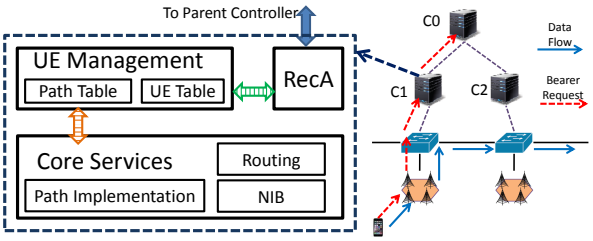


Figure 6: UE Management Application

5.1 UE Bearer Management

In each SoftMoW controller, the mobility application registers for the bearer request message type in the core. It also registers for the bearer response from the parent in RecA. The mobility application maintains two tables (Figure 6): (i) UE table where each row contains a bearer request and a local path ID. (ii) Path table that maps path IDs to the their details. A bearer request can be in the format of (UE ID, BS ID, SRC IP, DST IP, REQ) where the "DST IP" is the destination address on the Internet and "REQ" contains QoS. For example, some UE applications can request for better QoS on the end-to-end latency.

When a UE sends a bearer request to the base station, the request is forwarded to the leaf controller as a Packet-In message. The mobility application receives the request from the core and associates a service policy (i.e., a middlebox chain) with it if necessary. If there is no precomputed path in the path table, the mobility application calls the routing service using the northbound API (path, match fields)=Routing(request, service policy). Then it calls the northbound API (pathID, pathInfo)=PathSetup(path, matching fields) provided by the path implementation service. Finally, the path information is cached in the path table and the mobility application asks the base station to allocate the resources. As discussed in Section 4.2, if the routing service cannot find an end-to-end path that satisfies the bearer request and service policy, the mobility application sends the bearer request to RecA, which is forwarded to the parent controller.

Example. In Figure 6, the UE requests for a path with a larger bandwidth, which cannot be found by the routing service of C1 in its region; the request is sent to the root controller C0. C0 computes the path, stores the UE and path information, and sends the UE bearer response to C1's RecA. The C1's RecA implements the local path in it region, and C1's mobility application registers in its RecA to get the path information. Also, C2's RecA implements the rest of path in its region once it receives the virtual rule from C0.

The bearer state is synchronized between the UE and the mobility application. If the UE becomes idle, its bearers will be deactivated. We add two more fields to the UE table indicating whether a UE is active or idle, and whether the UE request has been handled locally or by the parent. When the mobility application deactivates a bearer, it updates the tables and also asks the path implementation service with the northbound API deactivatePath(pathID, pathInfo) to deactivate its path. If the UE bearer has been handled by the parent controller, the mobility application continues to request bearer deactivation from its parent via RecA.

5.2 UE Mobility

LTE has many handover procedures depending whether the source base station and target base station has a direct connection or not and whether the UE's current associated MME, or the serving gateway needs to be changed or not, etc. Similarly, there are many handover procedures in SoftMoW. We only discuss two main types of handovers: intra region and inter region. The handovers are performed through the coordination of the mobility application, RecA, the routing service and the path implementation service.

The Intra region type is used to handover a UE between a source base station and a target base station when both of them are in the same leaf region. This type of handover is easy, so we focus on complex inter region handovers. In inter region handovers, the source and target base stations are located in different leaf regions. Thus each corresponding border G-BS is exposed by a separate leaf controller. To simplify the inter region handover procedure and allow fine-grained region optimizations, we assume controllers do not aggregate gigantic stations and physical BS groups sitting at the border of their logical region with others in the recursive abstraction procedure. A leaf controller abstracts each border BS group as a single G-BS for its parent, and non-leaf controllers expose a single G-BS for each G-BS located at their region's boundaries. However, controllers can group, abstract, and expose their internal G-BSes and BS groups in different ways.

To handover a UE from the source base station to the target base station in inter region handover, SoftMoW only requires base stations abstracted as a border G-BS to advertise the corresponding *G-BS ID* along with other information periodically through the physical broadcast channel. When the source leaf controller and the UE agree on the handover target, the source leaf controller sends a handover request to its parent. The request contains at least source and target *G-BS IDs* and *BS IDs*. The mobility application registers for the handover request in the core. If the current controller is the ancestor of both the source and target leaf controllers, it starts a procedure to handle the request; otherwise the request is sent to RecA and forwarded to the parent controller recursively. For simplicity, we explain the inter region handover procedure through an example.

Example. To handover a UE from BS1 to BS2 in Figure 4, C1 sends a handover request from (G-BS1, BS1) to (G-BS2, BS2) to the root. The root requests G-BS2 to allocate the

resources at the BS2 to the UE. Then, it implements a new path between G-BS1 and G-BS2 to transfer in-flight packets and establishes some paths E2 and G-BS2 for new flows. Once the handover finishes, the root asks G-BS1 to release the resources. It then removes old paths between G-BS1 and E1 as well as between G-BS1 and G-BS2.

5.3 Region Optimization and Reconfiguration

Inter region handovers increase "east-west" control plane load because they require the intervention of at least three controllers: the source and target leaf controllers, and the ancestor controller. Allocating more resources to busy nodes in the controller hierarchy is difficult due to the geographical distribution and also increases the intra-node coordination costs. Thus the regions should be refined to reduce this type of load; each non-leaf controller should reconfigure its own logical region to minimize the inter region handover load it handles. To achieve this goal, the region optimization application changes borders between sub-regions, each exposed by an immediate child controller, based on handover patterns. Handover patterns vary across time-of-day. Thus it is difficult to find static borders using an offline and static approach, so each controller should be able to perform optimizations periodically and on a slow time-scale. In particular, we are interested in minimizing inter region handovers at the root (level L) first because a handover request processed and handled by the root goes through more controllers. Similarly, the controllers at the level $n-1$ have a higher priority compared to the controllers at the level $n-2$. Hence we should run the handover optimization algorithm first at the root. Once the root is done, all controllers at level $n-1$ can run the optimization in parallel, and similarly for the levels below.

5.3.1 Region Optimization Algorithm

We now discuss the optimization algorithm for a non-leaf controller which we call the initiator controller.

Handover graph input. When the mobility application processes handover requests, it can log these processing. Then a *handover graph* can be computed, in which each node of the graph is a G-BS and an edge shows the number of handover in the past time window (e.g., several hours) between two nodes. The region optimization application can fetches all handover graphs from the mobility application. The two applications can communicate through mechanisms such as inter-process communication. We do not provide any further details for lack of space.

Example. For a two-level SoftMoW architecture, Figure 7b represents a global handover graph built by the root through aggregating histories. Figure 7a shows the leaf regions' BS group-level handover graph. As discussed earlier, to allow the root to run fine-grained optimization at the site-group level, leaf controllers have abstracted each border BS group (e.g., BS groups 3 and 2) as a single G-BS (e.g., G-BS 3 and 2) and have exposed to the root. However, they have abstracted adjacent internal BS groups all together. A similar rule applies to any other non-leaf controllers.

Greedy algorithm. Using the handover graph, the region optimization application in the initiator controller computes the reconfiguration of its logical data plane by refining sub-regions, each exposed from a child controller. The region optimization informs the management plane about the changes. The management plane performs the actual reconfiguration. In handover-specific reconfiguration, the initiator detaches a border G-BS connected to a source G-switch and then re-associates it with a destination G-switch. The source and destination G-switches are connected through an inter G-switch links (discovered by the initiator). This operation transfers the control of the border G-BS to new descendant controllers in the initiator's subtree. We propose a simple greedy local search algorithm to decide which border G-BS should be reconfigured by the initiator. In our algorithm, the initiator at each step selects a border G-BS connected to a G-switch, which yields the maximum gain. The gain is defined as the reduction in the amount of inter region handovers requiring the intervention of the initiator.

Example. Figure 7b shows the root level handover graph before the optimization showing the root handles 900 inter region handovers between G-switches A and B or the corresponding leaf regions shown in Figure 7a. Based on the gain function, the controller selects border G-BS 3 for the reconfiguration since it gives the maximum gain 200 (=500-200-100). The root associates the G-BS with G-switch GSA.

Constraints. We assume we have the lower bound LB_i and the upper bound UB_i on the amount of control plane loads (e.g., UE arrival) that each G-switch (or actual child controller) can handle. When the initiator picks the maximum gain border G-BS, it avoids reducing the load of a G-switch GS_i to below LB_i or increasing it to above UB_i, assuming the load of each type of control plane events (e.g., bearer arrival) incurred by a G-BS is given.

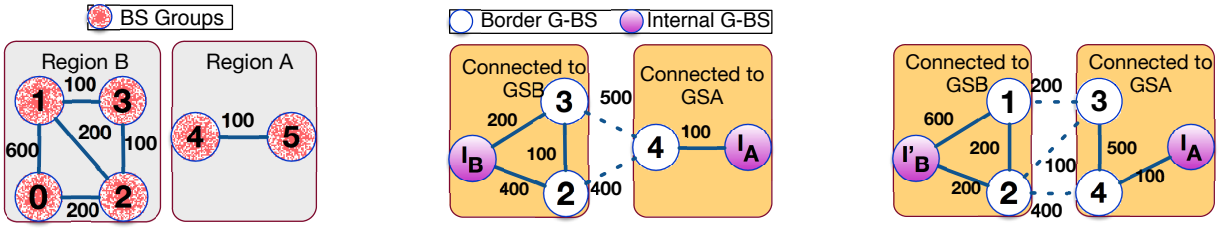
Termination and Convergence After the above steps, the initiator controller can enter into a new iteration of reconfiguration computation by selecting the next G-BS. The algorithm terminates when there is no more positive gain. *The sequential-parallel approach converges* because the handover optimization at an initiator controller, which is done by refining its logical sub-regions under its control, neither produces nor removes any gains for ancestor controllers except for the initiator itself, and controllers in its subtree. This is because a controller cannot affect inter region handovers seen at ancestor controllers.

5.3.2 Reconfiguration Protocol

Region optimization application computes the reconfiguration and sends reconfiguration messages to the management plane.

Finding leaf controllers. The management plane subscribers to topologies changes from NIB and abstraction changes from RecA. Using the topology information and configuration information, the management plane finds the source and destination leaf controllers, and instructs them to fulfill the G-BS re-association request from the region optimization application.

Reconfiguration. At this step, the source leaf controller finds a cut containing switches that are necessary to transfer the border BS group (abstracted as a single G-BS to allow fine-grained optimization) to the target leaf's region. It then communicates with the switches and component base stations to seamlessly add the target leaf controller as their new controller. In this procedure, the source leaf controller sets the role of the target leaf controller to the equal role (e.g., OpenFlow "OFPCR_ROLE_EQUAL"). This role means both the source and target leaf controllers receive all events generated by data plane devices (i.e., BS group, switches, and middleboxes). The management plane instructs: (i) the source leaf controller to handle events generated by existing



(a) Leaf controllers' graph

(b) Root's graph before the optimization

(c) Root's graph after optimization

Figure 7: Inter Region Handover Optimization

rules and avoid installing new rules. (ii) the target leaf controller to process all new requests (e.g., handover, routing, UE arrival, and path implementation). To make states consistent, the source controller transfers existing UE states and path information to the target controller in advance. When old communications finish, the source controller disconnects itself from the data plane devices and the new controller gets the master role.

Updating logical data planes. After a successful control transfer at the leaf level, the logical regions are updated from bottom to top in a recursive fashion to reflect new abstract topologies. Recursively, each RecA agent along the path modifies the G-switch ports and the virtual fabric for its parent. Next, the parent automatically discovers new inter G-switch links. Also, the RecA agents need to update, register, or deregister G-BSes. This is because some internal BS groups in the source leaf region become border BS groups, which should be reflected recursively. Figure 7c shows the root's handover graph after reconfiguring G-BS 3. The procedure transfers the control of BS group 3 from the source region B to the target region A. As a result, the new border BS group 1 is separated from I_B, abstracted as border G-BS 1 and exposed to the root. This leads to updating the internal G-BS I_B to I'_B which has lost BS group 1. Also, the target leaf controller might need to treat previous border BS groups as internal BS groups due to an expansion of its region.

6. DISCUSSION

We discuss how a basic SoftMoW can handle the controller, switch, and link failures, and implement consistent paths.

Controller failure recovery. To guarantee the reliability of the control plane, each logical node in the tree structure contains master and hot standby instances. For each node, NIB is decoupled from the controller logic and stored in a reliable storage system (e.g. Zookeeper [5]). The NIB is shared between the master and standby. The standby uses a heartbeat protocol to detect the failure of its master. Also, each physical or logical (i.e., master and standby) switch connects to both master and standby instances. All messages from a physical or a gigantic switch are duplicated and delivered to both instances. If a master is alive, the standby does not do anything. Otherwise, it takes over the master's work immediately. When the master controller receives an event, it first logs the event arrival in the NIB, and then processes it. When the master fails, the hot standby detects this and immediately checks the event logs and redo unfinished events.

Switch and link failure recovery. When a link failure occurs, the leaf or ancestor controller, which discovered the link, is notified through our recursive discovery protocol. If the failure affects the exposed G-switch and virtual fabric in a way that cannot be masked from the ancestor controllers, changes are reflected bottom up which may cause upper-level controllers to recompute new paths. Otherwise the controller finds affected local paths and implements alternative shortest paths with the same performance.

Consistent path setup. In SoftMoW, path implementations by a controller are pushed top-down. However, the topology updates propagate bottom-up. If we want to provide strong consistency between controllers in neighboring levels, messages needs to be ordered (e.g., paxos, locks) which impacts the agility of path implementations. SoftMoW guarantees eventual consistency. If a failure happens due to inconsistency (e.g., path implementation during topology changes), SoftMoW's controllers recomputes new paths. To guarantee a packet goes through a consistent path during path updates, the new path and packets are assigned a new version number. The packets with the old version number can still use old rules to guarantee reachability.

7. IMPLEMENTATION AND EVALUATION

We prototype the architecture of SoftMoW to show the performance gains of SoftMoW compared with current rigid LTE architecture and evaluate the scalability of our topology discovery protocol. Finally we show the effectiveness of inter region handover optimization using trace-driven simulations.

7.1 Prototype and Methodology

Data plane. We prototype SoftMoW on top of the Floodlight [1] and Mininet [18]. Leaf controllers use the OpenFlow protocol to communicate with switches while other controllers interact with logical data plane elements through a custom API similar to OpenFlow. We build realistic data plane topologies using the RocketFuel dataset [21]. We present the results for a data plane containing 321 software switches. To attach radio access networks, we use our LTE data set. We connect each BS group to an access switch. Each BS group contains at most 6 inferred base stations organized in a ring topology. The minute-level uplink and downlink traffic rates of BS groups is obtained from the dataset. We set the delay and bandwidth of links to 5ms and 1Gbps respectively.

LTE dataset. We collected about 1TB traces from a large ISP's LTE network during one week in the summer of 2013. The dataset covers a large metropolitan area with more than 1000 base stations and 1 million mobile devices. The trace is *bearer*-level. A radio bearer is a communication channel between a UE and its associated base station with a defined Quality of Service (QoS) class. The trace includes various events such as radio bearer creation, UE arrival to the network, UE handover between base stations. From the trace, we compute the uplink and downlink traffic per minute per base station. When a flow arrives and there is an existing radio bearer with the same QoS class, the flow will use the existing radio bearer. Radio bearers time out in a few seconds, so a long flow may trigger several radio

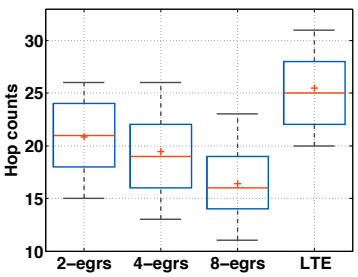


Figure 8: End-to-End Hop Count

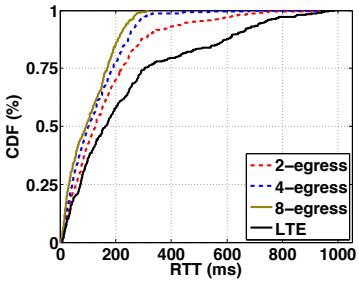


Figure 9: End-to-End Latency

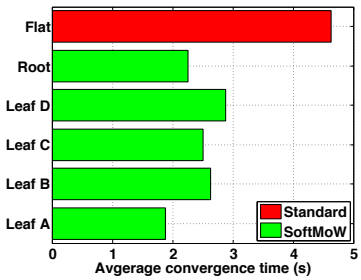
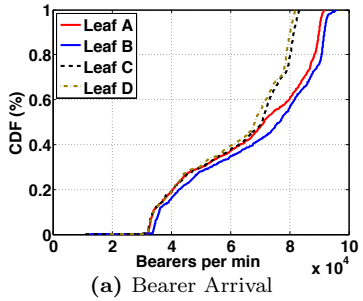
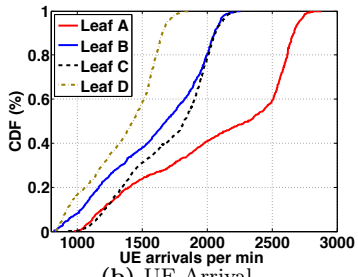


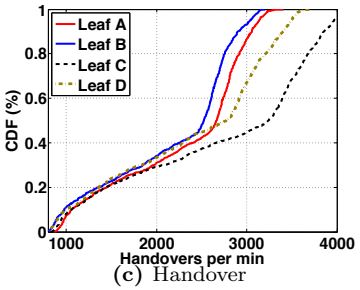
Figure 10: Convergence Time



(a) Bearer Arrival



(b) UE Arrival



(c) Handover

Figure 11: Cellular loads on Balanced Regions

bearer creation and deletion events. Because the data set does not contain flow-level information, we use radio bearers to estimate flow activities.

BS group inference. Our LTE dataset does not contain BS-group level information, so we infer BS groups by a simple algorithm. We assume each group has at most 6 base stations organized based on the ring topology. Our algorithm aims to find groups maximizing the weight of intra-group edges in the global handover graph. The optimal solution is NP-hard, so we design a greedy algorithm. In each iteration, the edge with the lowest weight is removed and then strongly connected components with fewer than 6 base stations are computed. We remove the components from the working graph and mark each as a new BS group. Finally, inferred BS groups are partitioned to form approximately equal-sized logical regions with similar cellular loads. We carefully assign a geographical location to each BS group to preserve the neighborhood relationship among them.

7.2 Routing Performance

We first focus on a two-level architecture with 4 leaf regions. We approximately place the leaf controllers in the center of their region. The root controller runs in the middle of the complete topology. SoftMoW's inter-connected core network increases the choices of Internet egress points so that the control plane can compute optimal end-to-end paths. We compare the two-level SoftMoW architecture with an existing rigid LTE region for the same number of base stations. To model egress points, we use iPlane [2] consisting of traceroute information from PlanetLab [10] nodes to Internet destinations. To consider routing changes, we replay the hop counts and latencies from multiple snapshots. The root implements internal shortest paths for traffic by taking into account both internal hop counts (from the G-BS to an egress point) and external hop counts (from an egress point to the destination).

Figure 8 illustrates the distribution of end-to-end hop counts as a function of the number of egress points for 11590 destinations on the Internet. We observe the average hop count decreases from 20.83 to 16 as the number of egress points increases from 2 to 8. This is because internal path inflation disappears since the traffic is directed through sufficiently close egress points, and also diversity of external paths improves the Internet access performance. In particular, SoftMoW with 8 egress points can reduce the average end-to-end hop count by 36% compared to LTE network. In addition, SoftMoW can also reduce end-to-end latencies by computing globally optimal paths at the root. Figure 9 depicts the CDF of RTT latency. We observe the 75th and 85th percentile RTT latencies reduce by 43% and 60% when we switch from the LTE network to the 8-egress point SoftMoW.

7.3 Discovery Protocol Performance

In the same setting, we now measure the convergence time of our recursive discovery protocol. The convergence time is measured per controller and starts from the beginning of a discovery period until all links and ports are discovered and become stable. We compare our results to the standard discovery protocol (e.g., LLDP) when a single controller is placed at the root's location and discovers all the links and ports.

Figure 10 shows the average convergence time for different controllers in our architecture and the flat control plane. We observe SoftMoW's controllers detect their topology between 44% and 58% faster compared to the flat discovery by the single controller. We identified the queuing delay at controllers is the root cause of such differences and the propagation delays between the controllers and switches have insignificant effects. The queuing delay is in proportion to the number of ports and links in topology.

Basically, SoftMoW is more scalable and can detect faults faster compared to flat single controller deployments because a large portion of links and ports are masked from each controller. Table 1 shows the leaf controllers on average have exposed 20.75% of total ports discovered in their logical

Table 1: SoftMoW Controller Abstractions

	Discovered			Exposed	Exposed
	SW	Ports	Links	Ports	Ports (%)
Leaf A	55	218	80	58	26
Leaf C	79	250	99	52	20
Leaf B	68	213	87	39	18
Leaf D	98	416	167	81	19
Root	4	230	115	-	-

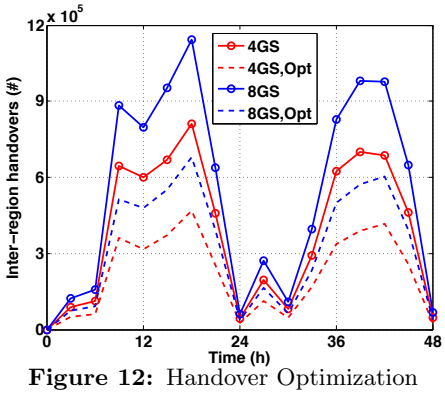


Figure 12: Handover Optimization

region to the root controller. Also, 73% of total links are hidden at the root level.

7.4 Handover Optimization

We characterize the cellular load on the leaf controllers and the effectiveness of inter region handover optimization through network measurement and simulation. We simulate a SoftMoW with two levels. In the first level, we define four and eight roughly equal-sized logical regions, each assigned to a leaf controller. In the second level, the root controller manages the abstract topology.

Cellular loads. Each leaf controller should handle three types of cellular events in addition to exposing logical devices to the root: *bearer arrival*, *UE arrival*, *handover request*. In practice, each type of cellular event can triggers multiple rounds of message passing between the controller and the logical data plane. Figure 11a shows the CDF of bearer arrivals. We observe each leaf controller handles as high as 10^5 bearer arrivals per minute. We use the bearer arrivals as the estimate of the number of packet-in messages received by the leaf controllers. Figure 11b shows leaf controllers receive and process between 1000 and 3000 attachment requests from UEs connecting to a base station in their region, which are triggered when users turn on their device. Figure 11c depicts the aggregate intra region and inter region handover requests processed by leaf controllers that varies between 1000 and 4000 per minute.

Optimization results. Periodically, the root refines the abstract sub-regions exposed from the leaf controllers based on its global handover graph. It strives to reduce the load of inter region handovers, which also improves the handover performance. In the optimization, we avoid drastically unbalancing the three cellular loads on each leaf controllers. Figure 12 shows the number of inter region handovers handled by the root over 48 hours for 8-region and 4-region settings. We observe the number of handover requests increases (i) in peak hours and (ii) by doubling the number of logical regions. The root runs the reconfiguration algorithm every 3 hours by collecting local handover graphs. We assume each GS (i.e., leaf controllers) should not handle more (less) than 30% of their maximum (minimum) initial cellular loads per minute. Given these constraints, Figure 12 depicts the root can reduce the load of inter region handovers by 38.08% to 44.61% using our iterative greedy reconfiguration algorithm.

8. RELATED WORK

Scalable control planes. Maestro [8] utilizes parallelism to achieve high scalability on multi-core machines. SoftMoW can benefit from the proposed techniques to make logical and physical rule installations faster at each node. HyperFlow [22] and Onix [17] are multi-controller designs without any explicit hierarchical structure. Kandoo [14] improves HypeFlow by leveraging a two-level controller. Unlike SoftMoW, Kandoo cannot be extended to more than two levels and can run specific applications such as elephant flow detection. In contrast to SoftMoW, these systems do not offer sufficient scalability to support continent-wide global applications.

Scalable data planes. To scale the data plane, SoftMoW, PNNI [11], XBar [19] hierarchically abstract a given network as logical entities. To control their specific target network and satisfy requirements, each of them offers different abstractions. PNNI's abstractions is designed for ATM networks. SoftMoW is the first complete recursive and reconfigurable architecture with richer abstractions suitable for cellular WAN operators. Unlike XBar and PNNI, SoftMoW builds virtual fabrics for its G-switches to enable network-wide optimization such as routing. In addition, SoftMoW runs a novel recursive label swapping mechanism to minimize the bandwidth overhead and data plane states.

Inter-DC control plane. Control plane architectures for data center WANs such as B4 [16] and SWAN [15] are specific to inter-DC traffic engineering. Inter-DC WAN topologies have several order of magnitudes fewer nodes and edges compared to the cellular WAN topologies [9]. SoftMoW's recursive and reconfigurable abstraction scales the network much better.

Cellular network control plane. Recently, researchers have also proposed flexible control plane architectures for cellular networks. SoftRAN [13] is a design specific to radio access networks. SoftRAN handles intelligent resource block allocation to optimize utilities. SoftCell [23] focuses on providing operators with fine-grained policies and compresses data plane rules. In contrast to prior work, SoftMoW handles inter-connected cellular core networks.

9. CONCLUSION AND FUTURE WORK

Cellular wide area networks have become an integral part of our society. However, they are remarkably inflexible and inefficient. This is exacerbated by the continued exponential growth of mobile data. To address this important problem, in this paper, we present SoftMoW, a scalable architecture that is based on on effective recursive and reconfigurable abstractions for both control plane and data plane. We designed a recursive link discovery protocol and virtual fabrics to allow automatic topology construction and support global resource management. SoftMoW optimizes network-wide objectives such as inter-region handover, path implementation, and routing. SoftMoW achieves these goals using novel algorithms benefiting from our scalable abstractions. Our evaluation results show that SoftMoW is very efficient and scalable. For future work, we would like to deploy SoftMoW in a large testbed.

Acknowledgments

We would like to thank the anonymous reviewers for providing valuable feedback on our work and Gordon Wilfong at Bell Labs for helpful discussions on BGP. This research was supported in part by the National Science Foundation under grants CNS-1039657 and CNS-1345226.

10. REFERENCES

[1] Floodlight. http://goo.gl/eXUprV.

[2] iplane dataset. http://goo.gl/JZWdK2.

[3] LTE design and deployment. http://goo.gl/DMKymH.

[4] Managing the signaling storm. http://goo.gl/lkTyb1.

[5] Zookeeper: Wait-free coordination for internet-scale systems. In *Proc. USENIX ATC*, 2010.

[6] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow, et al. ONOS: towards an open, distributed SDN OS. In *Proc. workshop on Hot topics in software defined networking (HotSDN)*, 2014.

[7] M. Caesar, D. Caldwell, N. Feamster, J. Rexford, A. Shaikh, and J. van der Merwe. Design and implementation of a routing control platform. In *Proc. USENIX NSDI*, 2005.

[8] Z. Cai and et al. The preliminary design and implementation of the Maestro network control platform, 2008.

[9] Y. Chen, S. Jain, V. K. Adhikari, Z.-L. Zhang, and K. Xu. A first look at inter-data center traffic characteristics via yahoo! datasets. In *Proc. IEEE INFOCOM*, 2011.

[10] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman. Planetlab: an overlay testbed for broad-coverage services. In *ACM Computer Communication Review*, 2003.

[11] T. A. Forum, M. Ahmed, and J. H. Rus. Private network-network interface specification version 1.0 (pnni 1.0), 1996.

[12] 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.

[13] A. Gudipati, D. Perry, L. E. Li, and S. Katti. SoftRAN: Software defined radio access network. In *Proc. workshop on Hot topics in software defined networking (HotSDN)*, 2013.

[14] S. Hassas Yeganeh and Y. Ganjali. Kandoo: A framework for efficient and scalable offloading of control applications. In *Proc. workshop on Hot topics in software defined networking (HotSDN)*, 2012.

[15] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer. Achieving high utilization with software-driven WAN. In *Proc. ACM SIGCOMM*, 2013.

[16] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, et al. B4: Experience with a globally-deployed software defined WAN. In *Proceedings of the ACM SIGCOMM*, 2013.

[17] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, et al. Onix: A distributed control platform for large-scale production networks. In *Proc. USENIX OSDI*, 2010.

[18] B. Lantz, B. Heller, and N. McKeown. A network in a laptop: rapid prototyping for software-defined networks. In *Proc. workshop on Hot topics in software defined networking (HotSDN)*, 2010.

[19] J. McCauley, A. Panda, M. Casado, T. Koponen, and S. Shenker. Extending SDN to large-scale networks, 2013.

[20] 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 *Proc. ACM SIGCOMM*, 2009.

[21] N. Spring, R. Mahajan, and D. Wetherall. Measuring ISP topologies with Rocketfuel. In *ACM Computer Communication Review*, 2002.

[22] Tootoonchian, Amin and Ganjali, Yashar. Hyperflow: A distributed control plane for OpenFlow. In *Proc. INM/WREN*, 2010.

[23] Xin Jin and Li Erran Li and Laurent Vanbever and Jennifer Rexford. SoftCell: Scalable and flexible cellular core network architecture. In *Proc. ACM CoNEXT*, 2013.

[24] K. Zarifis, T. Flach, S. Nori, D. Choffnes, R. Govindan, E. Katz-Bassett, M. Mao, and M. Welsh. Diagnosing path inflation of mobile client traffic. In *Proc. International Conference on Passive and Active Network Measurement (PAM)*, 2014.

[25] J. C. Zuniga, C. J. Bernardos, A. de la Oliva, T. Melia, R. Costa, and A. Reznik. Distributed mobility management: A standards landscape. *Communications Magazine, IEEE*, 2013.