Technical Report 2012/04

Towards Cloud-assisted Software-defined Networking

Frank Dürr

Institute of Parallel and Distributed Systems
Universität Stuttgart

Universitätsstraße 38
70569 Stuttgart
Germany

August 2012

Towards Cloud-assisted Software-defined Networking

Frank Dürr
Institute of Parallel and Distributed Systems (IPVS)
Universität Stuttgart
Stuttgart, Germany
frank.duerr@ipvs.uni-stuttgart.de

Abstract

This paper introduces *cloud-assisted software-defined networking* (Ca-SDN), a networking paradigm for implementing flexible and highly optimized networks by combining the strengths of cloud computing and state of the art software-defined networking technologies. The basic idea is to utilize the vast resources of datacenters ("the cloud") for complex network management functionalities such as the management of network state and optimized route calculation. The software-defined network consists of standard OpenFlow switches whose flow tables are configured based on routes calculated by an external controller hosted in the datacenter. This approach promises a clear design, flexibility, scalability, high performance, ease of administration, and cost reduction.

In this paper, we present a cloud-assisted IP multicast service as a concrete example of Ca-SDN, and discuss the benefits, challenges, and research questions for implementing Ca-SDN by this example.

1 Introduction

In this paper, we introduce a novel networking paradigm called *cloud-assisted software-defined networking* (Ca-SDN) that combines the strengths of both cloud computing and state of the art software-defined networking technologies for the implementation of flexible and highly optimized communication systems.

The basic idea is to pull out computational complex and memory-intensive network management operations such as optimized route calculation, the management of network state information, multicast group management, accounting, etc. from the network and to implement them in one or—for instance, to increase reliability—multiple datacenters ("in the cloud"). The network itself is only responsible for forwarding packets.

To enable this separation of management and forwarding functionality, we built on the current trend of software-defined networking (SDN) [14, 17], in par-

ticular, the OpenFlow standard [16]. SDN allows for the separation of the control plane and data (forwarding) plane of switches and routers (since we assume multilayer switches, which de facto also implement routing functionality, we only speak of "switches" in the following). The forwarding tables (also called flow tables) of these switches are configured by an external controller process. In our Ca-SDN system, this controller is hosted in a datacenter to utilize the cloud resources for its scalable implementation and to enable the implementation of optimized routing algorithms. Packets without matching flow table entry are forwarded to the controller to calculate suitable routes on demand and configure the forwarding tables of switches on the calculated path accordingly. With Ca-SDN, the necessary network state information for route calculation is also gathered by the controller by querying the switches and stored in the datacenter to build a global view on the network. After setting a flow table entry, switches perform forwarding of matching packets without contacting the controller. Overall, this results in a network that is *centrally* managed and controlled from the datacenter.

Compared to current distributed "network-centric" approaches where relative complex protocols and functionalities like route calculation are executed by the routers themselves, this clear separation of management functionality (implemented in software and executed by the controller in the cloud) and forwarding (performed by switches in hardware) has clear advantages stemming either from SDN or the utilization of cloud resources:

- **Flexibility.** SDN offers a high degree of flexibility since the addition or modification of functionality only require a modification of the software implementation of the controller. For instance, novel routing algorithms can be installed without modifying switches just by adding or changing routing algorithms executed by the controller. The fact that the controller is typically implemented using modern standard languages such as Java or C++ together with the possibility to use the corresponding powerful developing environments further facilitates rapid changes.

- **Forwarding performance.** SDN enables flexibility without sacrificing the performance of forwarding. Since switches perform most forwarding operations in hardware using, for instance, ternary content-addressable memory (TCAM) for matching, forwarding can be performed at line rates beyond 10 Gbps in contrast to software routers. In case of missing forwarding table entries, also the time for setting up new entries can be reduced by utilizing the computational resources of the cloud. This time is particularly critical since, on the one hand, it slows down the communication, for instance, by delaying the communication setup of a TCP connection. On the other hand, long delays might lead to an overloaded controller or control network. For instance, in case of a UDP data stream, packets will be constantly forwarded to the controller as long as no forwarding table entry has been installed.

- **Scalability.** Central control naturally raises concerns about the scalability of the controller. However, the large computational resources of a datacenter promise a high degree of scalability by scaling up to multiple cores and scaling out to many machines. Moreover, we will show later that central control in fact often reduces network traffic compared to distributed protocols since the distribution of information often is based on flooding information instead of only forwarding it to a single entity.

- **Optimization.** With Ca-SDN, networks can be optimized using a global view on the network, for instance, to maximize the throughput or minimize communication latencies. This also includes constrained optimization problems such as guaranteeing a certain end-to-end latency—for instance, as part of a quality of service specification of a service-level agreement—while minimizing the network load. Due to the large computational resources of the datacenter, sophisticated optimization algorithms can be used to increase the quality of routing and to ensure the scalability of central online route calculation.

- **Ease of administration.** The administration of the network becomes more easy due to the central control and global view onto the network. For instance, the traffic of a certain source (or tenant in a cloud) can be monitored more easily at the central controller instance to determine excessive network traffic or to implement typical cloud pricing models such as "pay as you go", where the customer only pays for the actually used resources.

- **Cost reduction.** The example of cloud computing and datacenters shows that is often more cost efficient to employ a larger number of cheap machines (such as commodity PC hardware) than few expensive machines (such as powerful and highly reliable server hardware). Transferred to networking, it would be equally attractive to use inexpensive standard switches. By "outsourcing" complex functionality to standard PC hardware in the cloud, the complexity of switches can be reduced. Instead of implementing several protocols on a multilayer switch or router, the functionality of the switch can be reduced to mere forwarding and a few more helper functions such as collecting traffic statistics (e.g., packet counters). In the future, this might lead to standard switches, which can be produced cheap in larger numbers, and additional controller software implementing the logic of routing and other auxiliary functions.

Due to these advantages, we envision that Ca-SDN will have great influence on future network infrastructures and protocols, including the whole range of networks from local datacenter networks to wide-area networks. The necessary components and basic standards are already available today. Several providers offer powerful cloud computing infrastructures, and also network providers can

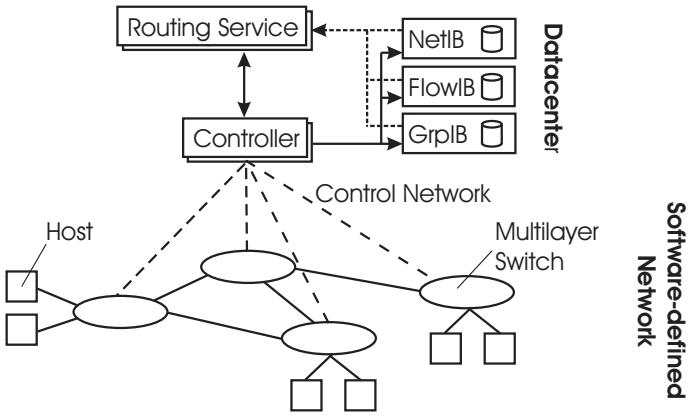

Figure 1: System Architecture.

setup their own datacenters (“private cloud”). Moreover, the OpenFlow standard enables the implementation of SDN. Companies such as IBM, NEC, and HP already offer first OpenFlow switches. The example of Google, which has adopted OpenFlow throughout their networks recently, shows that SDN is ready to be used in productive systems [9].

However, to enable Ca-SDN some challenges and research questions have to be addressed. Therefore, besides introducing the concept of Ca-SDN, the main contribution of this paper is the identification the associated challenges and research questions of Ca-SDN. In order to firm up this discussion and bring it to a technical level, we introduce a concrete communication system that benefits from Ca-SDN, namely, a cloud-assisted IP multicast service.

The rest of this paper is structured as follows. We start with a description of an architecture for CA-SDN systems in Section 2 using the concrete example of a cloud-assisted IP multicast system. Then, we describe the routing process, group management, and path calculation and optimization of the cloud-assisted IP multicast system in Section 3. Finally, we summarize the identified research questions in Section 4, and conclude the paper with a summary and outlook onto future work in Section 5.

2 System Architecture

In this section, we present a system architecture for Ca-SDN systems. Most of the described components and functions are generic. However, we will also show some specific components and give specific descriptions for our Ca-SDN example cloud-assisted IP multicast.

The components or our system are depicted in Figure 1. *Hosts* are the senders and receivers of communication flows. As one goal, we want to keep the current protocol stack implementation of hosts unmodified since otherwise

problems with a large number of legacy systems would arise. In our IP multicast example this means that we only rely on standard protocols like UDP/IP or the Internet Group Management Protocol (IGMP [4]).

Switches are responsible for message forwarding. We assume that these are multilayer switches that can interpret layer 2 to layer 4 header information to identify communication flows. According to the OpenFlow standard, flows can be identified by a 10 tuple including, for instance, source/destination MAC addresses, IP addresses, port numbers, or VLAN ids. For IP multicast, we can define source-specific distribution trees using destination multicast IP address and sender IP address, or shared trees using only the destination multicast IP address.

Furthermore, we assume that switches implement the OpenFlow protocol. In particular, the flow tables can be modified by an external *controller* process, which in case of Ca-SDN runs on a host in a datacenter as already mentioned. For our multicast example, we assume a single controller that is responsible for all switches of the network. The controller is connected to the switches through a control network using TCP and optionally TLS. This can be either a separate network, or control messages are forwarded "in-band" over the data network. The configuration of flow tables can be done either proactively or reactively. For instance, in the case of multicast, routes (distribution trees) and corresponding flow table entries can be installed proactively when hosts join a multicast group. Or they can be installed reactively "on the fly", when a switch receives a multicast packet without matching flow table entry. In the later case, when the first packet of a flow without matching flow table entry arrives at a switch, the switch sends it to the controller. The controller calculates a suitable route for this flow, and updates the flow tables of the switches along the calculated path—or multiple paths in case of multicast distribution trees—accordingly. Subsequent packets of the flow are forwarded at line rate without the need for contacting the controller again.

We only consider the network of one operator. This could be a datacenter network—in this case, the cloud network is managed by a controller residing in the same cloud—, or up to a complete autonomous system. Question on network virtualization to offer the "network as a service" to the customer or to create a virtual network spanning multiple providers are beyond the scope of this paper, and we refer to the literature for these questions [13, 21, 12]. Ca-SDN as presented in this paper is rather focused on how a network operator can manage and optimize his network. Although we only consider interior gateway routing, we want to note that logically centralized network management has also proven to be beneficial for BGP using central Routing Control Platforms (RCP). For instance, an RCP based on OpenFlow was proposed in [20] that also utilizes central data stores to manage network information similar to the central management of network information in Ca-SDN.

Besides the controller, several other services are hosted by the datacenter. The *network information base (NetIB)* stores information about the configura-

tion of the network. In particular, it contains information about the physical network topology. To determine the topology automatically, the Link Layer Discovery Protocol (LLDP [10]) can be used. Besides connectivity information, the NetIB also manages additional link information such as link capacities, the load of links, and latencies. This advanced information is a prerequisite for calculating optimal routes or routes fulfilling given QoS guarantees. With OpenFlow, switches maintain several counters, for instance, for the number of received and sent bytes per port, which are queried by the controller to determine the dynamic link state. The result is a global view on the network topology and link state.

The *flow information base (FlowIB)* contains information about each flow. On the one hand, it stores the path of each flow using a global view on the flow table entries of each switch (global forwarding information base). On the other hand, it manages traffic statistics for each flow, in particular, flow data rates. Again, this dynamic information can be collected by querying the counters of switches.

The *group information base (GrpIB)* stores multicast group membership information. In more detail, it stores information about which switch has group members directly connected to a switch port. Theoretically, the GrpIB could also manage group memberships of individual hosts. However, as shown below it is sufficient for our approach to know at which port of a switch group members are connected. For this purpose, we can use the standard IGMP protocol, thus, no changes to hosts are required. Moreover, we will show below that switches do not have to implement IGMP since group management is moved to the controller in the datacenter.

In order to reduce the latencies when updating or querying these information bases, they are kept in main memory, possibly replicated on different machines running processes of the routing service (see below) to increase scalability and for fast error recovery in case of host failures.

The *routing service* is responsible for calculating routes of flows, i.e., distribution trees in the case of multicast. Routes can be calculated according to different optimization goals and QoS constraints (cf. Section 3.4). The routing service has access to all the information bases mentioned above. Moreover, several routing processes can run in parallel on one machine (with several processing cores) or separate machines in order to reduce the latency of route calculation and to scale with the number of flows needing route information.

Besides these core services, there can be further *auxiliary services* (not shown in the figure). For instance, an *accounting service* could track the consumed network resources of individual flows based on flow counters. Messages exceeding a certain quota could be interrupted by simply removing flow table entries at the source switch. Or according to the common cloud pricing model "pay as you go" the sender could be billed based on the actually induced network traffic as already motivated. Although we do not go into further detail on how such auxiliary services could be implemented, the central availability of the nec-

essary information in a single datacenter (possibly also storing the master data of the provider such as customer data) facilitates the implementation of such services.

3 Cloud-assisted IP Multicast Service

In this section, we present details about the implementation of a cloud-assisted IP multicast service to identify challenges and research questions of Ca-SDN. For this service, we describe the group management, the routing process, the optimization of route calculation, and how to optimize route updates and the collection of network state information. During the course of this section, we also highlight the advantages of cloud-assisted multicast compared to common distributed multicast protocols such as DVMRP [22] or MOSPF [15].

3.1 Group Management

Before messages can be forwarded, group members of a given multicast group have to be identified and recorded in the GrpIB. Since our goal is not to make any changes to hosts, we use standard IGMP for this purpose. With IGMP, hosts send host membership reports containing the IP multicast group address to the local router of their LAN. With cloud-assisted multicast, we want to collect group membership information in the datacenter to make it available to central route calculation and to relieve switches from managing group information. To this end, on receiving an IGMP message, the switch forwards it to the controller. The controller adds an entry to the group membership database to record (a) that the switch has directly connected group members, and (b) over which port of the switch directly connected group members can be reached.

The controller creates periodic IGMP membership query messages, which are sent over the control network to the switch and then over all ports to the "all systems" multicast group 224.0.0.1. This query reaches the hosts connected to the switch, which reply as described above with membership reports. If no reports for a group are received from a switch within a certain time period, the respective group membership entry is removed from the GrpIB (soft-state approach).

As first advantages, we see that switches do not need to implement the IGMP protocol anymore and do not need to manage group memberships since the controller handles IGMP messages and manages the GrpIB. Moreover, compared to link-state multicast protocols like MOSPF, no group membership information has to be flooded throughout the network to create a global view on memberships on *every* router. Instead, network-wide flooding is replaced by unicast messages to the controller, which significantly reduces the communication overhead.

3.2 Reactive Routing

Next, we describe the cloud-assisted routing process. Following our principle not to modify the hosts, the first step in sending a multicast message is similar to traditional IP multicast. The sending hosts maps the IP multicast address to a MAC layer multicast address and sends the multicast packets using MAC layer multicast to the local switch.

Let us assume that a reactive routing strategy is used for installing flow table entries, and this is the first packet to this multicast group arriving at the the switch. Since the switch does not have a flow table entry for this destination IP multicast address, it forwards the packet to the controller. The controller is in charge of identifying group members, calculating a distribution tree, and installing the distribution tree by configuring the switches' flow tables along the calculated tree. Obviously, these steps are time critical since as long as the distribution tree is not installed, the switch cannot forward packets and relays them to the controller which might result in an overloaded controller or control network if routing takes too long.

The identification of group members is done using the global GrpIB. A lookup with the multicast address of the packet yields all switches having directly connected group members. These are the terminal nodes of the distribution tree. This lookup can be implemented efficiently in practical constant time using a trie with sub-microsecond lookup latency. Moreover, a parallel implementation using multiple machines for different requests is straightforward since lookups of different requests are independent.

Knowing the terminal nodes, the distribution tree can be calculated using topology and flow information from the NetIB and FlowIB, respectively. Since this is one of the most interesting steps w.r.t. efficiency and possibilities for optimization, we dedicate Section 3.4 to this topic. The controller updates the flow tables of switches along the distribution tree accordingly. Moreover, for switches with directly connected group members, actions to perform a MAC-layer multicast for the respective ports have to be installed on the switches to re-write the destination MAC address to the MAC-multicast address corresponding to the IP group address (if it is guaranteed that the destination MAC address does not change during forwarding, this step can be left out since the sender already used the corresponding MAC-layer destination address). Then, hosts will receive the multicast packets as usual per MAC-layer multicast.

As advantages compared to protocols based on flooding & pruning like DVMRP [22] or PIM-DM [1], no multicast packets have to be flooded throughout the network. Compared to link-state protocols like MOSPF, the distribution tree is calculated only once rather than (redundantly) at every multicast router along the distribution tree. Also no rendezvous or core routers have to be managed in contrast to CBT [3] or PIM-SM [6] (the controller knows the complete distribution tree anyway). In general, the centrally calculated tree can be optimized in various ways based on global network and flow state, and sophisticated algo-

rithms as shown in Section 3.4. It is even possible to reconfigure the multicast tree completely "on the fly", changing, for instance, from a source-based tree to a shared tree, just by centrally updating the flow tables. Finally, the OpenFlow switches do not need to implement any multicast routing protocol at all.

3.3 Proactive Routing

Reactive routing has the advantage to reduce the flow table sizes of switches. As long as no packet is actually sent to a certain group, no flow table entries are installed. However, it also leads to two drawbacks. First, an additional latency for the first packet(s) of a flow is induced by forwarding packets to the controller, route calculation, and configuring flow table entries. Second, the controller or control network might become overloaded if the whole process of setting up flow table entries takes too long. In particular, for connection-less UDP communication (as used by IP multicast), which does not require a communication setup and thus can start sending packets immediately at a possible high rate, this problem is significant.

This can be avoided by proactively setting up flow table entries when the first hosts join the multicast group. The rest of the described forwarding process remains unchanged. The proactively calculated distribution tree must include all group members. Moreover, it must contain source-specific paths from each possible sender to the distribution tree. Since IP multicast does not require senders to be group members nor to register before they start sending, proactively connecting senders to the distribution tree becomes a problem. If possible senders are known a priori (for instance, all machines of a tenant in a datacenter network), we simply can connect all machines to the distribution tree. However, this might not be possible in an open system. Then, introducing a primitive for signaling the intention to send to the controller can be a solution.

In general, handling UDP traffic flows with high data rates in a scalable manner remains a challenge in SDN.

3.4 Optimization of Distribution Trees

The great flexibility of Ca-SDN becomes most obvious when we look at routing and the calculation of multicast distribution trees in detail. Basically, the controller can calculate any kind of tree. For instance, it could calculate an individual source-based tree per sender, e.g., a single-source shortest path tree per sender that is optimal w.r.t. the latency between the sender and each group member. Or it can calculate a shared tree containing all group members using, for instance, a minimum Steiner tree that leads to the minimum network load. Instead of just optimizing one metric such as minimum latency or minimum network load, the controller could also solve a constrained optimization problem. For instance, it could try to find a Steiner tree with a given upper bound on end-to-end (sender-to-member) latency such that the induced network load

is minimal. Such constrained optimization problems are particular useful for ensuring QoS parameters.

Routing can even be changed on the fly by switching from one distribution tree to another during message forwarding. This is essential to adapt to dynamic network state such as the changing load of links to satisfy the optimality objectives or QoS constraints. However, instead of just re-calculating the tree with the same routing algorithm, SDN has the flexibility to complete change the strategy of tree calculation and the associated routing algorithm dynamically. For instance, when the number of source-based flow table entries grows too large in case of a source-based distribution tree, it could fall back to shared trees with only one entry per group.

Moreover, the resources of the cloud can be used to calculate more optimal routes than would be possible with comparable weak routers. For instance, assume that the minimization of network load is the optimization goal. The optimal tree w.r.t. network load is a minimum Steiner tree. However, for larger networks calculating the exact solution is infeasible in general since the Minimum Steiner Tree Problem is an NP-complete problem. Due to the complexity of this problem, common multicast routers fall back to simpler algorithms like calculating a single-source shortest path tree using Dijkstra's algorithm and pruning the branches that do not lead to terminal nodes. Given the processing resources of a datacenter, more optimal heuristics for this problem could be applied [18]. This requires algorithms tailored to the large number of processing cores and machines available in the datacenter. For instance, the enumeration algorithm describe in [18] calculates a number of minimum spanning trees (MST) for different subnetworks of the complete network. Calculating these MSTs in parallel on different cores or machines is straightforward since these calculation are independent.

Moreover, it might be desirable to strictly limit the runtime of such calculations. In particular, if a reactive routing strategy is applied, the calculating should be in the sub-milliseconds range. The mentioned enumeration algorithm is a good example for an algorithm where this could be achieved. Instead of calculating the complete set of MSTs, one could set a deadline for the calculation of MSTs. If this deadline has passed, the final result (Steiner tree) is calculated based on the MSTs calculated so far. Given a weaker deadline or larger number of compute resources, this heuristic yields more optimal results. Therefore, we can trade-off optimality for runtime. In particular, for high volume data flows and/or proactive routing strategies, a better result and longer runtime pays off.

Beyond optimizing single flows individually, we can also consider global optimization problems involving *all* flows together. As a realistic example, consider the following constrained optimization problem: Maximize the overall network throughput under max-min fair bandwidth allocation for each flow. This problem has been studied in [5] for ATM networks and unicast flows (similar to TCP flows which also achieve fair bandwidth allocation), however, similar problems

could also be defined for multicast communication. Although TCP is known to achieve fair bandwidth allocation on links traversed by the flow, usually the route of flows is assumed to be given. In [5], the authors show that the overall throughput can be increased significantly by calculating optimized route. Also this problem is a (NP) hard problem [11] that could benefit from advanced optimization algorithms and the compute resources of the cloud. In [2], the authors demonstrated that central global route optimization using advanced optimization algorithms (in this example, a simulated annealing algorithm) can be used to optimize also larger networks (in this example a larger datacenter network) online.

3.5 Optimizing Network Updates

Not only the process of optimizing distribution trees according to given optimization goals needs to be considered for an efficient operation. Also the process of updating routes and flow table entries needs to be optimized to ensure scalable and consistent operation.

First of all, we should try to minimize the number of flow table updates since each update means management overhead. Considering the fact that it is undesirable to re-configure the whole network when only few flows are added, removed, or change their properties like data rates or group members and senders, we strive for routing algorithms that yield stable solutions which only require incremental changes of a small number of flow table entries. This way, the number of flow-table updates can be reduced.

Moreover, correct forwarding avoiding, e.g., loops and duplicates, becomes a challenge under dynamic routing. Although the calculated routes are correct – for instance, a multicast distribution tree connects the sender to all group members and is loop-free by definition –, the transition between routes might lead to inconsistent behavior if no precautions are taken.

There are different basic approaches to tackle this problem:

- Only strive for *eventual consistency*, which is often the consistency semantic guaranteed by distributed routing protocols. Obviously, if no further changes are induced, eventually the network again reaches a consistent state where the last (and correct) distribution tree is installed in the network. Since in general packet forwarding is not required to be loop-free, loss-free, etc., during transitions, one could argue that the resulting problem anyway have to be handled by the transport protocols or on the application layer, for instance, by detecting duplicates, retransmitting packets, etc. In other words, ensuring correct behavior during transitions is only an optimization w.r.t. eventual consistency.

- Find *update mechanisms that guarantee consistent behavior* also during transitions, for instance, by performing forwarding in phases as proposed

in [19]. Such mechanisms are facilitated by the global view and central control as available in Ca-SDN.

3.6 Optimized Collection of Network State

The optimization of routing relies on the global knowledge of dynamic network state information at the central controller. However, collecting this information induces communication overhead that could outweigh the benefits of optimization. Therefore, optimizing the collection of network state information is another objective.

As mentioned for the NetIB and FlowIB, we would like to know the current data rates of each communication flow and load of each link. The switch traffic counters can be used to this end. The first question is, what is the required granularity of data rates to perform sufficiently good optimizations? Obviously, collecting more fine-grained information results in a higher overhead for collecting it. Another question is, what is the right mechanism for collecting traffic statistics? With the current OpenFlow standard, the controller has to poll switches. Using events instead to only notify the controller of significant changes can reduce the overhead, but also requires new concepts to specify events and event functionality on the switch. As a workaround, a local "proxy" controller can be used that is co-located with the switch, and which creates events from the values queried from the switch using the standard OpenFlow protocol. This would at least reduce the load on the control network and central controller.

In many cases, the efficiency of collecting traffic statistics can already be improved by considering the specific topology of the network and selecting strategic locations (switches) in the topology to collect traffic information. As an example, consider the datacenter network topology depicted in Figure 2. This is a typical Clos network topology as proposed in [7], which could also be used for our cloud-assisted IP multicast service if deployed in a datacenter.

This topology connects a large number of 103,680 physical hosts using 5405 switches (77 Intermediate (Int), 144 aggregate (Agr), 5184 top-of-rack (ToR) switches). Let's assume it is sufficient to collect the data rates of links every 10 s (actually, whether this rate is sufficient is an open question depending on the concrete optimization quality). Further assume that we are only interested in the current load of links and not the data rates of each individual flow, for instance, to calculate a Minimum Steiner Tree as mentioned above for multicast routing. First of all, we see that in this topology, there is only one link from each host to its ToR switch. Collecting the data rates of these links is irrelevant for routing since a distribution tree has to traverse this link anyway if a group member or sender is located on this host. Since every other link is connected to an Aggr switch, it is sufficient to query the traffic statistics (number of bytes sent and received) for all ports of each Aggr switch with a query rate of $q = \frac{1}{10}$ s. According to the OpenFlow standard, every query is sent via TCP/IP to the Aggr switch, which leads to a protocol overhead of $H_{2-4} = 82$ bytes (42 bytes

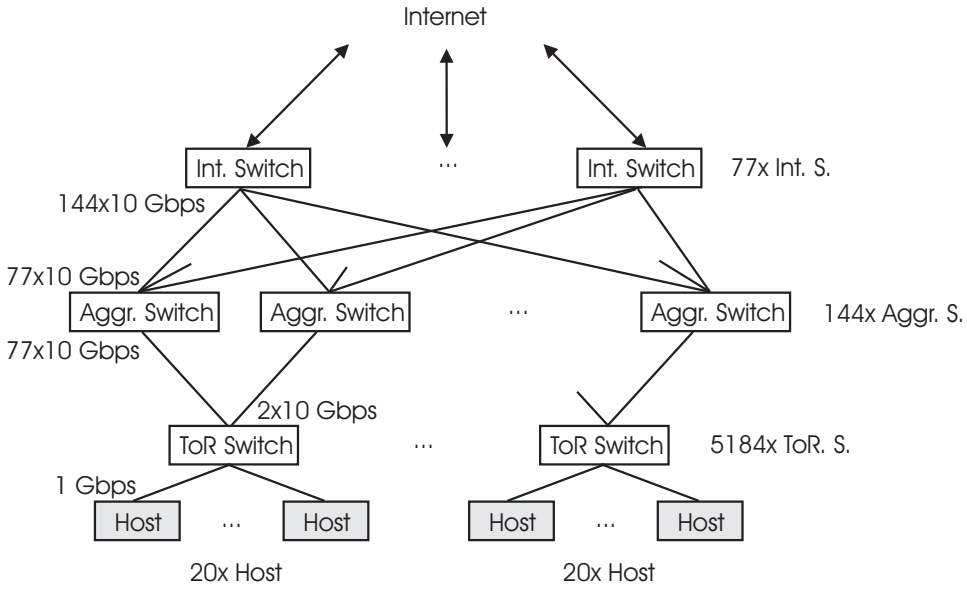

Figure 2: Datacenter Network Topology.

layer 2 header, 20 bytes IP header, 20 bytes TCP header) per packet. The application layer protocol part of an OpenFlow port statistics request consists of $S_{req} = 32$ bytes (8 bytes general header, 16 bytes status request header, 8 bytes port request header). The application layer part of an OpenFlow port statistics response for one switch with $p = 144$ ports consists of $S_{res} = 15000$ bytes (8 bytes general header, 16 bytes status response header, $104p$ bytes response body). Note that this response contains more information than only the bytes sent and received since currently the OpenFlow standard returns all traffic statistics per port including number of received/sent packets, packet drops, etc. Thus, the overhead could be further reduced by implementing more selective query mechanisms in the OpenFlow standard. Overall, this leads to the following total data rate $r = 1.8$ Mbps at the central controller assuming a maximum transmission unit of $M = 1500$ bytes and $A = 144$ Aggr switches:

$$r = 8A(H_{2-4} + S_{req} + S_{res} + H_{2-4}\lceil \frac{S_{res}}{M - H_{3-4}} \rceil) q \text{ [bps]}$$

If the control network is a separate network, a 1 Gbps network would be more than sufficient w.r.t. collecting traffic statistics. Also if the control network is implemented "in-band" over the data network, such low rates could easily be handled in the example datacenter network.

However, it might be harder to monitor other network topologies. And if detailed per flow statistics are required for each link (including the host links), then the overhead increases significantly in our example due to the large number of

103,680 hosts (assuming several virtual machines connected to a virtual software switch on each physical host, this number even increases more). Therefore, how to optimize the collection of global network state remains another research question.

4 Research Questions

In this section, we summarize the research questions identified in the previous sections. Some of these questions are relevant in general for SDN, and some in particular for Ca-SDN:

- **Minimizing flow table size.** The number of flow table entries is limited by the TCAM size of the switches (typically, up to 150,000 entries). Therefore, the number of flow table entries needs to be minimized at least for larger networks such as the large datacenter network used in the example above. Considering the multicast example, shared trees consume less entries than source-based trees. Since for some applications source-based trees might be better suited due to the optimization criteria (for instance, minimum latency using a single-source shortest path tree), one has to decide carefully, when to use which kind of distribution tree to find a tradeoff between satisfying application requirements and the minimization of TCAM space. With respect to the routing strategy, the central question is, which flows to install proactively (possibly risking to waste flow table entries for inactive flows), and for which a reactive strategy (using flow table entries only on-demand) is sufficient? This also affects the next question, the low latency setup of flows.

- **Low latency flow setup.** In particular for the reactive routing strategy, achieving a low latency for setting up flow table entries is essential. Therefore, how to utilize the resources of the datacenter for fast routing through parallel algorithms becomes an interesting question. If very low latency is required, a proactive flow setup strategy might be the better choice. However, this required a priori knowledge of the communication relations. We have seen that for IP multicast such a primitive to signal the intent to send to a certain group is missing.

- **Optimized routing on global view.** The multicast example has demonstrated the flexibility to calculate routes according to many different objectives. The question is, how much can we improve routing according to objectives like throughput, fairness, latency, etc. using a fine-grained global view on the whole network? This direct leads to the next question of how to collect the relevant network state efficiently.

- **Efficient collection of global network state.** Global optimization requires a global view on the network. However, the collection of dynamic

global information must not outweigh the benefits of global optimization. Therefore, the first question is, what is the best trade-off between the granularity of collected state and the resulting benefit? This question has to be answered individually, depending on the optimization objective and the application requirements. We have also mentioned the possibility to increase the efficiency of collection by using more controllers co-located with the switches and/or eventing mechanisms for traffic statistics.

- **Consistent forwarding during transitions.** We have seen that during transitions forwarding can be incorrect leading to loops, duplicates, etc. Also SDN does not prevent inconsistencies in forwarding automatically. However, Ca-SDN with its global view and central control seems to facilitate solutions. First approaches have already been proposed, e.g. [19]. However, we expect to see more formalisms and mechanisms in the future.

- **High reliability.** A single controller raises concerns about the reliability of the system. Achieving reliability within a datacenter can be based on redundant controllers using the large number of redundant machines, and as the exemplary datacenter network shows, also the networks of datacenters are offering redundancy. However, recent incidents have shown that it is not unlikely that complete datacenters become unavailable, for instance, due to power supply problems. Therefore, using multiple datacenters would increase reliability significantly.

- **Distributed control.** Some of the above research questions pointed to using a set of distributed controllers as possible solution. In the case of Ca-SDN, this would mean that multiple datacenters could be used, or additional controllers could be co-located with the switches or groups of switches. However, distributing control among several controllers raises more research questions. How should distributed controllers be coordinated for a consistent and efficient (globally optimal) operation? Possible controller topologies include control hierarchies or flat peer-to-peer topologies (or combinations of both). Other questions are, how many controllers do we need and where to place them [8]? This certainly depends on the objective to achieve. For instance, to achieve reliability, few controllers in few datacenters would be sufficient. To achieve low latency and decrease the load onto the control network, a larger number of co-located controllers might be ideal. One of the key questions is, therefore, how to find the right balance between distribution and centralization?

5 Summary and Future Work

In this paper, we have presented cloud-assisted software-defined networking (Ca-SDN) as a combination of software-defined networking and cloud technolo-

gies. By outsourcing complex management functionality to powerful datacenters and using OpenFlow switches for fast flow-based forwarding, we tried to achieve a clear design that benefits from the strengths of both technologies. The benefits of Ca-SDN include high flexibility; fast low-latency forwarding; high scalability; optimization of the network w.r.t. to various objectives; ease of administration; and cost reduction. Moreover, we presented the design of a cloud-assisted IP multicast service to showcase the benefits, challenges, and research questions of Ca-SDN.

As part of ongoing work, we are implementing the introduced cloud-assisted multicast system and different route optimization algorithms to explore the practical limits and gains of Ca-SDN.

Moreover, we are going to investigate how different communication systems such as advanced group communication systems (semantic multicast, geocast, etc.), or event and publish/subscribe middleware can be implemented using SDN in general, and Ca-SDN in particular. We argue that these systems, which are typically implemented in overlay networks today, can increase their performance significantly by using Ca-SDN. A specific question for these systems is how to map their application-specific addresses to flows to utilize the efficiency of switches for forwarding.

References

[1] A. Adams, J. Nicholas, and W. Siadak. Protocol independent multicast – dense mode (PIM-DM). IETF, RFC 3973, Jan. 2005.

[2] M. Al-fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat. Hedera: Dynamic flow scheduling for data center networks. In *Prococeedings of the Seventh USENIX Symposium of Networked Systems Design and Implementation (NSDI '10)*, 2010.

[3] T. Ballardie, P. Francis, and J. Crowcroft. Core based trees (CBT). *SIGCOMM Comput. Commun. Rev.*, 23(4):85–95, Oct. 1993.

[4] B. Cain, S. Deering, I. Kouvelas, B. Fenner, and A. Thyagarajan. Internet group management protocol, version 3. IETF, RFC 3376, Oct. 2002.

[5] S. Chen and K. Nahrstedt. Maxmin fair routing in connection-oriented networks. In *Proceedings of the Second European Parallel and Distributed Systems Conference (Euro-PDS '98)*, pages 163–168, Vienna, Austria, July 1998.

[6] B. Fenner, M. Handley, H. Holbrook, and I. Kouvelas. Protocol independent multicast – sparse mode (PIM-SM). IETF, RFC 4601, Aug. 2006.

[7] 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 *Proceedings of the ACM SIGCOMM 2009 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 51–62, Aug. 2009.

[8] B. Heller, R. Sherwood, and N. McKeown. The controller placement problem. In *Proceedings of the First Workshop on Hot Topics in Software-defined Networks (HotSDN 2012)*, pages 7–12, Helsinki, Finland, Aug. 2012.

[9] U. Hölzle. OpenFlow @ Google. Open Networking Summit 2012, Apr. 2012. Slides available online `http://opennetsummit.org/talks/ONS2012/hoelzle-tue-openflow.pdf`.

[10] IEEE Computer Society. *802.1AB – Station and Media Access Control Connectivity Discovery*, 2005.

[11] J. Kleinberg, Y. Rabani, and E. Tardos. Fairness in routing and load balancing. *Journal of Computer and System Sciences*, 63(1):2–20, 2001.

[12] K. Lakshminarayanan, I. Stoica, and S. Shenker. Routing as a service. Technical Report UCB/CSD-04-1327, Computer Science Division (EECS), University of California Berkeley, 2004.

[13] J. Marias, E. Jacob, D. Sanchez, and Y. Demchenko. An OpenFlow based network virtualization framework for the cloud. In *Third IEEE International Conference on Cloud Computing Technology and Science (CloudCom '11)*, pages 672–678, Nov. 2011.

[14] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. OpenFlow: Enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74, Apr. 2008.

[15] J. Moy. Multicast extensions to OSPF. IETF, RFC 1584, Mar. 1994.

[16] Open Networking Foundation. *OpenFlow switch specification*, June 2012.

[17] Open Networking Foundation. *Software-defined Networking: The New Norm for Networks*, Apr. 2012.

[18] H. J. Prömel and A. Steger. *The Steiner Tree Problem*. Vieweg, 2002.

[19] M. Reitblatt, N. Foster, J. Rexford, C. Schlesinger, and D. Walker. Abstractions for network update. In *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 323–334, Helsinki, Finland, Aug. 2012.

[20] C. E. Rothenberg, M. R. Nascimento, M. R. Salvador, C. Corrêa, S. Lucena, and R. Raszuk. Revisiting routing control platforms with the eyes and muscles of software-defined networking. In *Proceedings of the ACM SIGCOMM*

Workshop on Hot Topics in Software Defined Networking (HotSDN '12), Aug. 2012.

[21] G. Schaffrath, C. Werle, P. Papadimitriou, A. Feldmann, R. Bless, A. Greenhalgh, A. Wundsam, M. Kind, O. Maennel, and L. Mathy. Network virtualization architecture: Proposal and initial prototype. In *Proceedings of the First ACM Workshop on Virtualized Infrastructure Systems and Architectures*, pages 63–72, 2009.

[22] D. Waitzman, C. Partridge, and S. Deering. Distance vector multicast routing protocol. IETF, RFC 1075, Nov. 1998.