ATHENA: Reliable Multicast for Group Communication in Data Centers

Abstract

Several data center applications such as Hadoop and OpenStack Nova VM provisioning service utilize group communication (one-to-many or many-to-many transfers). Since these applications require reliable delivery, they leverage TCP for all group communication. Even though multicast lends itself naturally to these group communication patterns, it has remained largely underdeployed in the Internet owing to concerns around reliability and security. However, data center networks with their structured topologies and centralized control using Software Defined Networking (SDN) present an opportunity to address these concerns. In this paper, we explore the feasibility of using SDN-based reliable multicast for all group communications inside data centers. We present the design and implementation of ATHENA– an SDN-based reliable multicast system for applications that use group communications in data centers. ATHENA enables reliable transfer to ensure that no data is lost, congestion control to ensure that other TCP flows in the network are not starved, multi-tenancy and security. We have implemented ATHENA as a module in OpenDayLight OpenFlow controller and also enhanced OpenStack Nova service to use ATHENA. Our experiments on real hardware demonstrate that ATHENA improves Nova VM provisioning time by upto 2.5 times.

1 Introduction

Data centers typically host back-end applications and services, that perform several tasks as execution, synchronization and replication. These services are extremely critical for seamless functioning of user facing applications. This back-end east-west traffic is replete with group communication patterns. HDFS in the case of Hadoop [2] and Google distributed file system [19] in the case of Google for distributed file storage; MapReduce in the case of Hadoop [2] for distributed data execution; IBM Websphere [4] for state synchronization across clustered application servers; Nova in the case of OpenStack [9] for batch VM image provisioning are some examples of applications that have inherent group communication patterns. Multicast lends itself naturally to these group communication patterns as it avoids data duplication in the network and enables optimal bandwidth usage, yet is under-deployed in data centers. These use cases, coupled with the unique setting that modern data center networks offer with their structured topologies, high link density and a managed environment (software defined environments or software defined data centers), have spurred renewed scrutiny in multicast schemes with a focus on adoption for data centers [23, 26, 28]. However, most back-end services that require group communication, still primarily use TCP. This is mainly because of their requirement to have a reliable and secure mode of communication that can recover from packet losses. Use of TCP for group communications is far from optimal, as it results in inefficient usage of the network as TCP would transfer data over multiple one to one connections, thereby resulting in same data being sent over certain links of the network multiple times.

Modern data centers are changing. Recent works such as B4 [24] by Google, ANANTA [30] and SWAN [21] by Microsoft and the strong industry backing for the Open Networking Foundation [5], all point towards the rapid emergence of Software Defined Networking (SDN) in data center networks. Software defined networking, when applied to managed structured topologies in data centers, provides a unique opportunity to provide a robust, reliable and secure mode of group communication in data center networks. In this context, we present ATHENA– a reliable, secure, scalable and TCP-friendly multicast scheme for SDN based data centers. The SDN architecture offers a centralized control plane to control all the switches from a central entity – the controller, which enables centralized global network visibility and centralized control – attributes we leverage in ATHENA.

We now discuss the high level requirements a multi-

cast scheme should fulfill in a software defined data center setting so as to be considered for adoption for group communications.

OPTIMAL ROUTING, SCALABILITY, STATELESSNESS. Traditional routing schemes, as PIM-SM [16], are based on localized switch level views. SDNs offer global network visibility and centralized control, which may result in more efficient multicast routing trees as shown in [23]. Further, not all commodity switches have support for multicast that requires enhanced state maintenance and packet handling logic. On the other hand, SDN switches can easily implement multicast routing through OpenFlow rules as shown by Iyer et al. [23]. In this work we leverage the routing scheme from [23], while adding support for reliability, congestion control, scalability, statelessness and security.

SECURITY. Security has always been a cited point of concern and a deterrent to practical deployment of multicast. SDNs offers a unique avenue to simplify threat detection and security enforcement due to its access to global network state.

TENANT SEPARATION. Data centers house multiple tenants. Tenant separation entails that multicast addresses should be reusable across tenants. There should not be any interference between tenants' multicast communication even if they employ an identical multicast address.

RELIABILITY. Data centers run several data critical applications that are extremely sensitive to packet losses. Typically, reliable delivery at high data rates is a necessity and cannot be compromised with. Moreover, packet losses in data centers are inevitable due to the unpredictable and bursty nature of data center traffic [13]. Thus, the multicast scheme should be able to recover from data losses even at high data rates.

CONGESTION CONTROL. Multicast communication should coexist with existing communications. TCP is the de facto choice of communication protocol used by applications in data centers. Thus, the multicast scheme should be TCP friendly in its congestion control scheme. This will also prevent unreasonable data losses.

Reliability and Congestion Control are top requirements when it comes to a multicast scheme's practical deployment for group communications as they ensure seamless adoption and peaceful coexistence with existing applications. However, at the same time, other requirements described above cannot be discounted. The design goal of ATHENA is to provide a data center deployment-ready solution. To achieve this goal, the design of ATHENA (§ 3) addresses all of the above requirements, with major focus on reliability and congestion control. Further, we implement ATHENA (§ 4) for an OpenFlow enabled SDN based data center. We evaluate ATHENA (§ **??**) on a small data center like topology with real hardware and demonstrate its usefulness by integrating it with OpenStack Nova image provisioning. Our results demonstrate that ATHENA can improve VM provisioning and file transfer times by up to 2.5 times.

2 Background

ATHENA is a reliable multicast scheme for SDN based data centers. It leverages the SDN architecture via the OpenFlow standard [29]. Moreover, ATHENA uses the Internet Group Management Protocol (IGMP) [34] for inferring multicast group memberships and is based on the Pragmatic General Multicast (PGM) protocol [18] to ensure reliability. In this section we give a brief background on SDNs, OpenFlow and PGM.

SDN AND OPENFLOW. Software Defined Networks (SDNs) decouple the network traffic forwarding management system, referred to as the control plane, from the network entities actually performing the traffic forwarding operations, referred to as the data plane. This enables logically centralized network intelligence in software-based controllers that maintain a global view of the dumb forwarding entities, namely the commodity hardware switches and/or the software-based virtual switches.

OpenFlow is an open communications protocol that is defined to interface between the control plane and the data plane of an SDN architecture. The OpenFlow protocol allows remote administration of a switch's packet forwarding table (also referred to as flow table) by adding, modifying and removing packet matching rules and actions. Typically the switch matches any incoming packet to its flow entries in the flow table and in case of no match forwards the packet to the controller. The controller, in turn, either updates its state about the abstracted global network or issues a `FLOW_MOD` message to make additions, deletion or modification to the switch's flow table. Each flow entry at a switch consists of a match field (*in-port*—*eth*-type-src-dst—*VLAN*-id-priority—*IP*-src-dst-proto-ToS—*TCP/UDP*-sport-dport) and a set of actions (drop–forward to controller–forward to ports–composition of above). Each attribute in the match field is populated with either a specific value to match against the corresponding attribute of an incoming packet or is a wildcard to signify omission of matching against that attribute.

PGM. Pragmatic General Multicast (PGM) is a single-sender, multiple-receiver reliable multicast computer network transport protocol which runs over a best effort datagram service, such as IP multicast. A PGM source multicasts sequenced data packets (`ODATA`) to the receivers. To ensure reliability, PGM uses special packets such as Negative Acknowledgement (`NAK`), NAK Confirmation (`NCF`) and Repair Data (`RDATA`). In case an `ODATA` is lost, a unicast `NAK` is sent by the receiver to

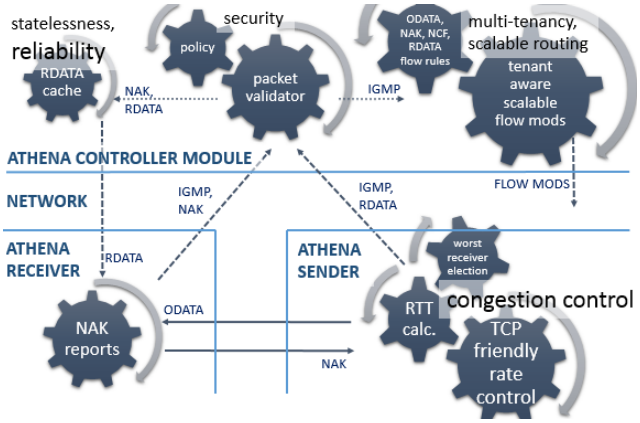


Figure 1: ATHENA Architecture

the sender, PGM-hop by PGM-hop. On receiving a NAK, a PGM-enabled network element multicasts an NCF on the interface on which the NAK was received to acknowledge receipt of the NAK and updates its repair state for that particular sequence number(s) and in turn forwards the NAK upstream to the next PGM hop. Finally on receipt of the NAK, the source, in addition to NCF, generates multicast RDATA. The PGM-enabled network elements maintain repair state with regards to the interfaces requesting repair packets for a particular sequence number and perform constrained-forwarding for that RDATA. In addition to constrained-forwarding, PGM also has optional provisions for NAK elimination, subnet-based NAK-suppression and Forward Error Correction to further enhance scalability. However, all these scalability features come with the added condition and cost of PGM-friendly state maintained at intermediate network elements.

3 ATHENA

ATHENA comprises of control-plane modules at the controller and data-plane modules at the end-hosts as shown in Figure 1. ATHENA's data-plane modules at end-hosts (multicast receivers and multicast senders) are based on the PGM protocol. Even though PGM is a reliable multicast protocol, ATHENA gives a new context for PGM in an SDN based setting. ATHENA preserves the PGM packet semantics, while adding novel modifications to the way these packets are handled to achieve reliability, congestion control, multi-tenancy, security and scalability in a stateless manner. In particular, ATHENA centralizes reliability but also devices a tcp-friendly transmit rate control scheme to substantially limit the involvement of the control channel for reliability purposes. Typical life cycle of a multicast session has the following three phases: multicast group establishment, multicast tree formation and finally multicast data transmission. In case of ATHENA, the life cycle of a multicast session has the following phases: multicast group establishment, security verification, tenant isolation, multicast route formation, congestion controlled reliable multicast data transmission and multicast group termination. All these phases are exclusively or inclusively orchestrated by the controller modules and the end-host modules of ATHENA.

3.1 Lifecycle of a Multicast Session

In the first phase, which is multicast group formation, end-hosts generate IGMP membership reports indicating join requests for a particular multicast group. Since no flow entries are added by the controller at the switches for IGMP packets, all these IGMP packets in the network are forwarded to the controller. The controller keeps track of the members in a multicast group by listening to all IGMP traffic and also other multicast traffic in the network. Other multicast traffic refers to the first few multicast packets (ODATA) sent by a new multicast sender that did not match any of the flow entries in the switches till the time the sender in the multicast group was identified and the corresponding multicast routing tree was installed by the controller in the network. The routing is done by the controller by first constructing the tree and then installing the corresponding flow entries. Since ATHENA's reliable multicast scheme is based on the PGM protocol, the flow entries installed at the switches are specialized for PGM packet types that the end-hosts generate. These flow entries, not only enable multicast data transmission, but also lay the foundation for reliability, congestion control and various stateless optimizations (such as RDATA constrained forwarding, NAK suppression) that ATHENA guarantees. In the last phase (multicast group termination), the controller queries idle multicast groups via IGMP membership queries and handles IGMP leave requests to prune or remove multicast groups from the network.

3.2 Routing

We leverage the Avalanche Routing Algorithm (AvRA), specifically designed for SDN based data centers, developed by Iyer et al. [23]. Optimal tree building is equivalent to solving the Steiner Tree problem [22]. On arbitrary graphs, such as the ones represented by traditional IP networks, the Steiner Tree problem is known to be NP-complete. However, AvRA takes advantage of the structured topologies that data centers offer to build near-optimal multicast trees in polynomial time, which for data center topologies like Tree and FatTree reduce to the optimal route solution. Moreover, AvRA tries to minimize the size of the routing tree created for each

sender multicast group and enables linear time addition or deletion of a multicast group member. AvRA also introduces path diversity and equitable switch distribution across multicast trees by randomizing the path of a new multicast group member to its nearest attachment point in the desired multicast tree. ATHENA uses the multicast tree formation algorithm proposed in [23], but makes it PGM aware and makes several changes to support reliability, congestion control, stateless optimizations, multi-tenancy and scalability.

3.3 Modifications to PGM

PACKET MODIFICATIONS. To enable PGM in an SDN setting, ATHENA uses slightly modified versions of the packets used by PGM. ATHENA needs to identify the ODATA, RDATA, NCF and NAK packets that are generated by PGM via OpenFlow match attributes. Since ODATA, RDATA, NCF have source match attributes that of the sender and destination match attributes that of the multicast group, there is no way to distinguish between these packets in their original form. Moreover NAK are unicast packets from the receiver to the sender and cannot be distinguished from other unicast traffic with the same source and destination. As a solution to this, ATHENA modules at the end-hosts change the PGM packet contents by embedding a unique value in the IP Type of Service (ToS) field for NAK, NCF and RDATA. Multicast addressed ODATA are identified via the absence of a value in the IP ToS field.

ROUTING AND STATELESS OPTIMIZATIONS. ATHENA does differential routing for the various packet types. ODATA are forwarded as is on the forward multicast tree routes. On the first-hop switches from the multicast senders, flow rules are installed to drop NCF and to forward RDATA to the controller. Flow rules are also installed on the first-hop switches from the multicast receivers, to forward NAK to the controller and optionally, if NAK suppression is enabled, to other receivers of this multicast group on the same first-hop switch. Additionally, reverse multicast tree routes are also installed for NAK from the receivers to the sender. Forwarding NAK and RDATA to the controller achieves constrained RDATA forwarding, whereas sending NAK to the peer receivers ensures NAK suppression. All this is achieved via simple OpenFlow-based flow rules without necessitating PGM-specific state maintenance requirements at the switches.

ADAPTIVE TIMEOUTS. PGM is rigid in its design in the sense that several parameters such as NAK timeout values are fixed and have to be manually tuned to best fit a particular deployment. ATHENA enables accurate RTT estimation as explained in 3.5.2. This serves as a barometer for the network and is used to dynamically change the various PGM parameters depending on the network conditions.

3.4 Reliability

PACKET LOSS DETECTION. ATHENA's reliability sets in when multicast receivers detect packet losses. We use the control loop that PGM protocol dictates for packet loss detection and associated NAK generation. Whenever an out of sequence ODATA, which leaves a gap in the receiver's sequence number space, is received, the receiver triggers a back-off timer with a random timeout value ($NAK\_RB\_IVL = \mathcal{O}(RTT)$) for the presumably lost sequence numbers in this gap. This back-off state ensures some amount of wait for the presumably lost data packets, which may have been set out of order by the network, until declared lost. If the back-off phase times out, the generation of NAK for the lost sequence number(s) takes place. After the generation of NAK, PGM protocol mandates a random back-off until it receives an NCF and thereafter a random back-off phase again till it receives RDATA. Time out of any of the above back-off phases mandates a NAK retransmission. In case of ATHENA, the receivers directly go into a back-off phase with random timeout ($NAK\_RDATA\_IVL = \mathcal{O}(RTT)$) waiting for RDATA. ATHENA does away with the NCF associated back-off phase because it is intended for the scenario where PGM-enabled intermediate network elements are present.

REPAIR CACHE UPDATE. After NAK generation by the receivers, NAK are forwarded to the controller along the control-plane. NAK are also forwarded along the data plane to the sender and to the peer receivers on the same first-hop switch. Forwarding the NAK packet to the peer receivers is an optional feature at the controller to ensure NAK suppression. NAK transmission to the peer receivers serves as an indication that loss notification of a particular sequence numbered data packet has already been sent and need not be retransmitted by any other receiver in the same multicast group on that particular first-hop switch. The controller maintains a sequence number indexed repair cache per sender per multicast group that stores the list of receivers requesting RDATA. On receipt of a NAK, the controller queries the repair cache for RDATA requested and, if present, selectively injects RDATA to the requesting receiver from the first-hop switch connected to it as an OpenFlow PacketOut. In case NAK suppression is enabled, it multicasts RDATA packet to all the receivers of that multicast group attached to this first-hop switch. If the controller doesn't have RDATA, it adds the requesting receiver to the corresponding list of waiting receivers. NAK suppression is best suited for scenarios wherein loss of a packet is more likely to occur at an intermediate link in the net-

work than at the link from a first-hop switch to a receiver. In such a scenario all the receivers on a particular switch will generate the same NAK. NAK suppression ensures such repeat transmits are minimized. The controller repair cache employs a least-recently-used (LRU) scheme that periodically (period = 1 minute) cleans up RDATA entries unused over this period.

CONSTRAINED RDATA FORWARDING. On receiving a NAK, the sender generates the requested RDATA. This RDATA is not immediately transmitted but instead added to the retransmit queue which is periodically serviced (*period* = $\mathcal{O}(RTT)$) for RDATA transmission. This is not mandated by the PGM protocol and is a new addition in ATHENA at the sender that aids in avoiding duplicate RDATA transmits. Typically, when a loss in ODATA occurs on a multicast tree link, all the receivers in that subtree generate NAK with the same sequence number(s). Even if NAK suppression is enabled, different receivers across different first-hop switches in that subtree will still generate the same NAK. Thus, at the sender, on receipt of a repeat NAK, the retransmit queue is checked for presence of corresponding RDATA and retransmission is avoided. Once the RDATA is generated, it is forwarded to the controller from the sender's first-hop switch. At the controller, the corresponding entry in the repair cache is checked and constrained RDATA forwarding is achieved as only the waiting receivers are selectively provided with RDATA by injection as OpenFlow PacketOut at their first-hop switches (individually or to all group receivers depending on whether NAK suppression is enabled or not). Thereafter the waiting list is purged and RDATA copy is stored in the repair cache for servicing any future requests.

3.5 Congestion Control

ATHENA modifies the PGM based data-plane modules at end-hosts (multicast receivers and multicast senders) to achieve congestion control. ATHENA's congestion control scheme is a rate-based scheme. Although TCP provides a window-based congestion control scheme, ATHENA tries to mimic TCP's scheme by appropriately setting transmit rates in accordance to those that TCP's window-based scheme would dictate in that particular setting. This makes our congestion control scheme TCP-friendly. Also, unlike TCP wherein ACK is an orchestrator for its congestion control scheme, ATHENA's congestion control scheme relies on NAK. ATHENA has a NAK-based single-rate multicast scheme, with added tasks of RTT estimation and worst receiver election to ensure TCP friendly congestion control. ATHENA's TCP-friendly congestion control scheme guarantees healthy coexistence with TCP flows. This also ensures that ATHENA's multicast flows are resilient even with UDP flows by occupying the remaining available bandwidth. This ensures that packet losses are kept at a bare-minimum and the involvement of the control-plane for packet repair is minimized.

3.5.1 Worst Receiver Election

ATHENA is a single-rate based multicast scheme. As a result, it needs to ensure that none of the receivers fall back drastically in its received sequence number space. To ensure this, ATHENA identifies the worst sender–to–receiver path and ensures congestion control along that path towards the worst receiver. ATHENA uses the instantaneous throughput, Γ (equation 1), observed at the receivers as a metric to identify the receiver path experiencing the maximum congestion in the multicast tree. The use of this metric is similar to the metric Throughput Rate at Congestion (TRAC) that ERMCC [27] uses. During a loss event, instantaneous throughput is a direct indication of the available bandwidth. In case of losses at multiple receivers, the comparison of this metric helps identify the receiver with the least available bandwidth and hence its election as the worst receiver. To achieve this, receivers convey the instantaneous throughput on a packet loss (Γ_{loss}) to the sender via NAK. Upon receipt of a NAK, the sender, among other operations, extracts this metric and updates its worst receiver and associated instantaneous throughput value.

$$\Gamma(t) = \frac{bytes_{(t+\delta t)} - bytes_t}{(t+\delta t) - t},\ \delta t = 20msec \qquad (1)$$

3.5.2 RTT estimation

In addition to worst receiver election, ATHENA's data plane modules at receivers and sender also coordinate via NAK to estimate RTT. RTT estimation is done at the sender for the worst receiver only and is used for performing TCP-friendly rate control. To estimate RTT, in addition to Γ, every receiver embeds the maximum sequence number of ODATA seen at the time of generation of NAK as well as the delay (time elapsed) in between the time at which the NAK was generated and actually sent (t_{delay}). The sender on receiving NAK from the worst receiver, keeps track of the time at which it received that particular NAK (t_{in_time}), extracts the embedded sequence number from NAK and queries it's sender window for the time at which it had sent that particular ODATA (t_{out_time}). The sender, then, estimates RTT as:

$$RTT = t_{in\_time} - t_{delay} - t_{out\_time} \qquad (2)$$

3.5.3 Rate control at source

ATHENA's multicast sender also modulates the sending rate, γ (equation 3), in a TCP-friendly manner. If no NAK is received, sending rate is increased by 1 MSS/RTT every RTT, RTT being the estimated round trip time between the multicast sender and the current worst multicast receiver. In case NAK is received from the worst receiver, it is throttled to a high fraction of Γ_{loss} (embedded in NAK) if the rate was not already throttled in the most recent RTT.

$$\gamma = \begin{cases} \gamma + \dfrac{MSS}{RTT} & \text{no loss for a RTT} \\ min(\beta\Gamma_{loss}, \gamma) & \text{at loss, } \beta = 0.5 \end{cases} \quad (3)$$

Theorem 1 ATHENA*'s rate control scheme is TCP-friendly*

Proof: Refer the Appendix. ■

3.6 Multi-Tenancy and Scalability

Data centers typically house multiple cloud tenants. In an SDN based data center, the controller maintains a map between the packet context (ingress switch, source MAC of the packet) and the associated tenant-id. ATHENA's controller module makes use of this map via the controller exposed API's to get the tenant-id upon receiving a packet. ATHENA has a VLAN mapper that maps each tenant-id a unique VLAN-id and installs routes that are VLAN-id aware i.e. ATHENA pushes this VLAN-id into the related packets at the source switch and extracts it from the packets at the destination switch. Further, ATHENA uses VLAN-id and destination MAC based rules at the intermediate switches to install tenant distinguishable routes in the intermediate network.

Enabling multi-tenancy in this fashion also ensures scalability. Typical FLOW_MOD based rules get installed in the TCAM space of OpenFlow enabled switches. However, the TCAM sizes are limited. For example, IBM RackSwitch G8264 [3] has support for 1500 TCAM entries. This implies that even if installing a multicast tree route via a switch entails one flow entry, then that switch can support at most 1500 multicast trees. On the other hand, IBM RackSwitch G8264 has support for 97250 FDB (Forwarding DataBase, also known as L2 table) based rules. FDB based rules have a restriction that only destination MAC and VLAN-id are allowed as OpenFlow match attributes and the only supported OpenFlow action is output to a list of ports. In the case of ATHENA, VLAN-id and destination MAC suffice to uniquely identify a packet in the intermediate network and the only action required is output to a set of port(s) towards the destination(s). This allows ATHENA to install all multicast rules in the intermediate network as FDB based rules. The only bottleneck can be at the edge switches where the match involves matching on the IP ToS and the destination attributes and the corresponding actions involve composition of pushing/popping a VLAN-id or dropping a packet or forwarding a packet to the controller or forward on certain port(s). However, modern data centers invariably have virtual switches as the edge switches to enable network virtualization and virtual network provisioning so as to ensure an economic, scalable and efficient IaaS solution. Being software-based, virtual switches have no TCAM limitations, and therefore do not present any bottlenecks in terms of flow rule space and ensure scalability of ATHENA.

With regards to constraints on scalability from the point of view of using a logically centralized repair mechanism, ATHENA's congestion control scheme ensures that losses and hence involvement of the control-plane for repair are kept at a minimum. Moreover a typical data center SDN deployment consists of a cluster of logically centralized controllers managing different network slices [21, 24]. ATHENA's solution only requires a logically centralized repair cache and as substantiated in our implementation is possible to deploy over a cluster of controllers. In such a deployment, repair for losses in different parts of the networks serviced by different controllers over non-overlapping control-plane links further enhances the overall scalability of the soluton.

3.7 Security

Multicast security threats include data leakage due to third person IGMP group requests, multicast tree tampering using IGMP packet spoofing, denial of service using IGMP or NAK spoofing and data poisoning by ODATA/RDATA spoofing. Since all the IGMP traffic and some multicast traffic is forwarded to ATHENA's controller module from the edge switches, ATHENA does packet validation and policy enforcement for threat detection and mitigation. By default ATHENA enforces tenant isolation policy via its multi-tenancy module. Additional policies supported by ATHENA are multicast membership restrictions within a tenant that can be specified by the network administrator in an XML format specifying the allowed set or the disallowed set of end-hosts for a multicast group. ATHENA also does packet validation on all the incoming packets (IGMP, ODATA, NAK, RDATA) to identify instances of packet spoofing. ATHENA uses the controller APIs exposing the packet context to tenant-id map (ArpHandler in OpenDayLight) to validate incoming packets by querying for the packet context in this map. The absence of a corresponding tenant-id in the map, identifies an instance of a spoofed

packet. The packet context consists of the source switch identifier, the packet in-port on the switch and the source MAC. Although the source MAC can be spoofed, but the source switch identifier is a unique id assigned by the controller for each switch to controller control-plane TCP connection and cannot be spoofed.

4 Implementation and OpenStack Integration

ATHENA's design has been implemented as a controller module in the OpenDayLight (ODL-hydrogen-base-v1.0) controller [7] and the end-host based modules have been implemented by modifying the OpenPGM (v5.2.122) library [8]. To validate the design, we use OpenStack (havana-release-v2013.2.3) [9] cloud platform and enhance its Nova provisioning and Glance services to use ATHENA for enabling multicast-based VM image file transfer.

ODL. ATHENA's controller module is implemented in ODL and is cluster-aware. ATHENA uses ODL's clustering service using Infinispan distributed cache store. This enables ATHENA to scale over large network topologies in data centers by delegating repair responsibility to different controllers that manage different network slices.

OPENSTACK. OpenStack is an open-source cloud platform to provide Infrastructure-as-a-Service (IaaS) solution. It comprises of several interrelated components that manage compute (Nova), storage (Swift, Cinder), networking (Neutron) and image (Glance) resources in a data center and are controlled through REST APIs via a web-based dashboard (Horizon).

We modify OpenStack's Nova and Glance services to use ATHENA based multicast for batch VM provisioning. On receiving a request for batch provisioning of VMs, Nova Compute manager service is called with the request parameters. It initializes database records for the instances and forwards the request to the Nova Scheduler, which schedules a set of compute nodes to host the instances. Thereafter, spawn requests are sent out to Nova Compute worker services on the respective compute nodes via remote procedure calls (RPC), which individually send REST calls to Glance service to start receiving VM image. For multicast based VM image transfer using ATHENA, all the receiving nodes ideally need to synchronize and join a multicast group before Glance starts streaming VM image. To this end, we extended the Nova Scheduler service and Glance service to enable this synchronization. Nova Scheduler service now informs Glance service of the upcoming batch VM provisioning request via a REST call. In response, the Glance service generates a unique reservation-id and a unique multicast IP and passes these to the Nova Scheduler service. Nova Scheduler service piggy-backs the reservation-id and the multicast IP on the RPC to Nova Compute workers. Each Nova Compute worker checks for the presence of the VM image in its local cache and sends out a REST call to Glance service with the reservation-id and cache status. Each Nova compute worker then starts ATHENA based multicast receiver if the VM image is not found in its local cache. Meanwhile, Glance service waits for all the REST calls from all the Nova Compute workers and thereafter starts ATHENA's multicast sender to start streaming the VM image.

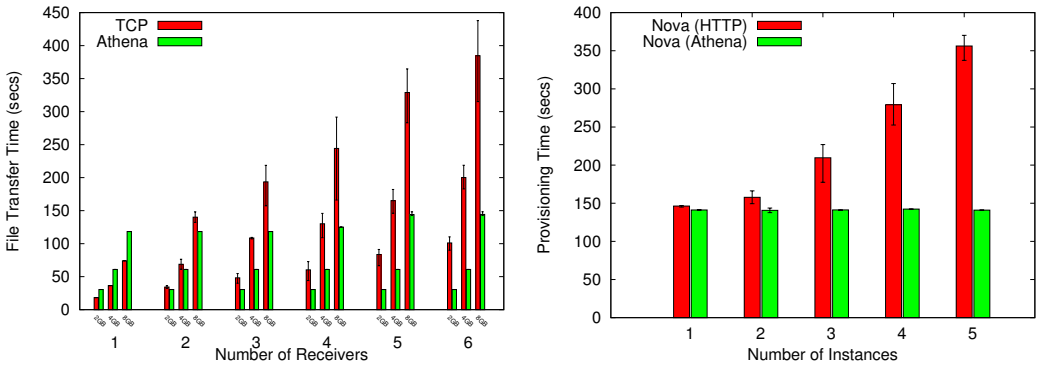
5 Evaluation

5.1 Experimental setup

All of ATHENA's experiments have been done on a physical testbed. Our physical testbed is a small data center consisting of 14 switches (IBM RackSwitch G8264 [3]) arranged in a three-tiered fat tree design with 2 core, 4 aggregate and 8 edge switches. Each edge switch connects to a server. Each server has an Open vSwitch (OVS version 2.0.2) [6] via which it connects to the edge switch. Thus OVS's are the eventual edge switches. All the hardware switches and the OVS's are OpenFlow enabled and are connected in a separate management network via a hub to the controller server. All of our servers are IBM x3650 M3 machines having 2 Intel Xeon x5675 CPUs with 6 cores each and 2 threads per core (24 logical cores in total) at 3.07 GHz, and 128 GB of RAM, running 64 bit Ubuntu Linux v12.04. The servers are connected to the edge switches (data-plane) via 1 Gbps NICs and to the management network (control-plane) via 100 Mbps NICs. To scale the experiments so as to achieve larger multicast groups, multiple receivers are run on each server and isolation is ensured by using cgroups and network namespaces. Each virtual interface on each of these cgroups connects to a port on the OpenFlow enabled OVS bridge on the server. In all the following experiments a *multicast session* refers to ATHENA's multicast flow for a file transfer with 1 sender and 50 receivers. We also disable NAK suppression in ATHENA's controller module in all of the following experiments to do a worst-case analysis.

5.2 Performance Comparison of ATHENA and TCP

In order to compare file transfer time between ATHENA and regular TCP connection, we wrote custom TCP based sender and receiver *C* programs to transfer files between any two servers. We configured our systems to use the TCP Reno congestion control algorithm though any other congestion control algorithm could have been



(a) Large File Transfers.

(b) OpenStack Provisioning Time.

Figure 2: Performance comparison of ATHENA and TCP.

used. For the experiments that require background TCP connections running between servers we used Iperf utility to generate that TCP traffic.

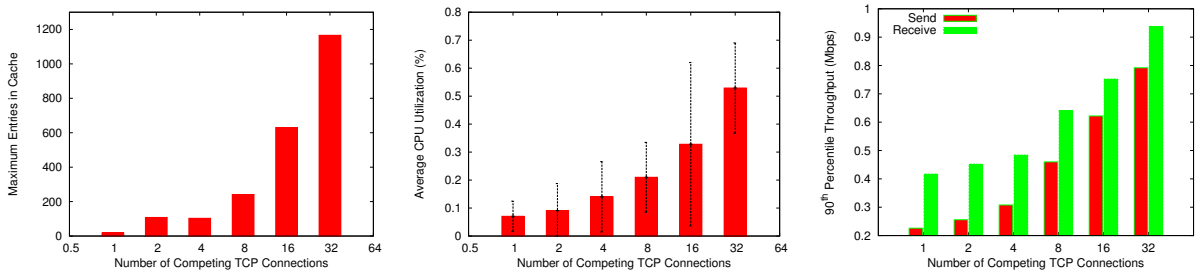
FILE TRANSFERS. In Figure 2a we compare the time to transfer files with ATHENA and with TCP individually from a single sender to increasing number of receivers with negligible background traffic. The file sizes for the experiment vary from 2GB to 8GB. We ensure optimal TCP file transfer times by using parallel TCP programs at the sender that run as separate processes and minimize context switching by distributing them uniformly across the available set of CPUs using the *taskset* utility. We observe that the file transfer times for ATHENA are identical across receivers whereas TCP has a higher standard deviation in the time for the same file transfer across receivers.

OPENSTACK INSTANCE PROVISIONING TIME. In Figure 2b we compare the time taken to batch provision VM images (time elapsed between REST call requesting VM provisioning and VM starting bootup), using OpenStack's out-of-box TCP-based implementation with OpenStack using ATHENA. The typical size of a batch operation is less than 5 images [32]. We increased the size of the batch from 1 to 5. The plot shows the average, minimum and maximum of the time taken to spawn each instance. The result shows that the VM provisioning time for ATHENA is independent of the number of instances in the batch and also uniform across instances for a single batch. For five instances, the provisioning time reduction is about 2.5 times and should improve further with larger batches. In our physical testbed, we observed the same 2.5x gain even on increasing the number of VM's spawned on each server as in OpenStack the transferred image is locally cached and multiple VM instances of the same type end up spawning from a copy of the same locally cached image.

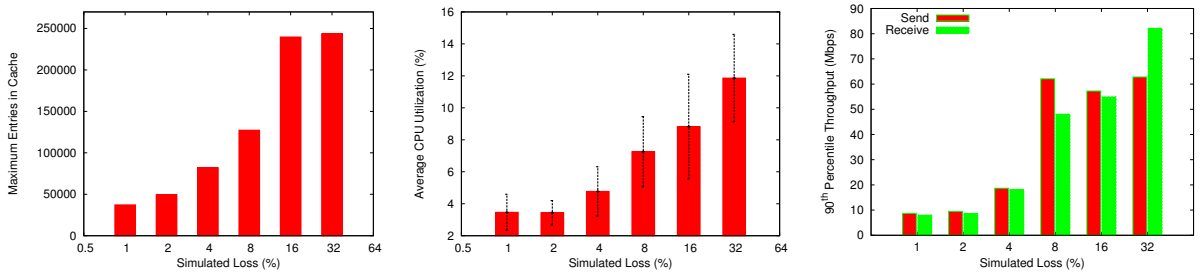
5.3 Controller Overhead Evaluation

In these experiments we evaluate ATHENA induced overheads at the controller. The first experiment is reflective of real case scenarios where multiple TCP connections compete with ATHENA's multicast flow whereas the second experiment is to stress test ATHENA's controller module with increasing severity (increasing link loss rates) and observe associated overheads. To get an idea of the ATHENA induced overheads, the performance metrics we observe include the cache size at the controller module of ATHENA, the CPU utilization of the controller and the network bandwidth usage to and from the management network (control-plane) at the controller server.

CONTROLLER PERFORMANCE IN PRESENCE OF MULTIPLE TCP CONNECTIONS. In this experiment we measure controller performance when a *multicast session* for a 2GB file transfer competes with increasing number of TCP flows on a single sender to receiver path. We observe an increasing trend for cache size, CPU utilization and network bandwidth at the controller in Figure 3a, 3b and 3c respectively. Increasing number of competing TCP connections leads to a higher frequency of packet losses, which in turn implies higher repair traffic to and from the controller and hence higher CPU utilization and repair cache size at the controller. These performance metrics are bound to increase with increasing data-plane traffic but even with 32 competing TCP sessions the values observed for these performance metrics are miniscule. Even on scaling this experiment to an uncontrolled size with 4000 random TCP connections competing with the *multicast session* we observed a peak network throughput of only ∼90 Mbps which suggests that ATHENA is scalable with increasing data-plane traffic. Also, all the file transfers done as part of this experiment completed reliably irrespective of the number of competing TCP connections.



(a) Cache Size trend with TCP connections. (b) CPU utilization with TCP connections. (c) Controller bandwidth with TCP connections.



(d) Cache Size trend with Lossy Link Rates. (e) CPU utilization with Lossy Link Rates. (f) Controller bandwidth with Lossy Link Rates.

Figure 3: Controller overhead evaluation of ATHENA.

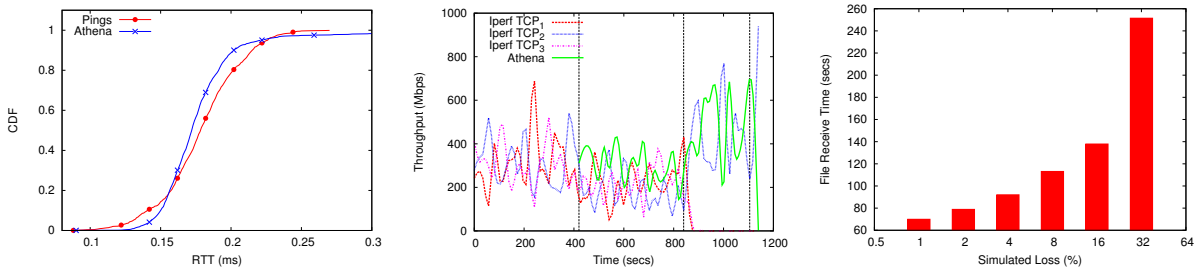
CONTROLLER PERFORMANCE IN PRESENCE OF LOSSY LINKS. In this experiment we measure the controller performance when a *multicast session* for a 2GB file transfer is subjected to a worst-case scenario of unending sustained packet losses. We simulate the effects of a lossy link by randomly dropping incoming packets at one of the receivers from the *multicast session* with a particular loss probability (loss rate) relentlessly till the file is transferred. Figure 3d, 3e and 3f give the observed performance metrics for cache size, CPU utilization and network bandwidth at the controller respectively with increasing sustained loss rates. We observe an increasing trend with increasing loss rates which is justifiable as repair traffic to and from controller increases and hence CPU utilization and cache size increases. All the file transfers done as part of this experiment completed reliably. Note that this experiment is just a stress test and does not simulate the momentary losses observed during traffic bursts in a data center.

5.4 Athena Accuracy and Guarantees

RTT ESTIMATION ACCURACY. Accuracy of RTT estimation is important for the congestion control mechanism in ATHENA to work properly. In this experiment, we have a *multicast session* with a pre-elected receiver randomly dropping 1% of the incoming packets. Since the NAK driven RTT estimates from ATHENA could not be one-to-one time synchronized with periodic RTT estimates from ping between the sender and the lossy receiver, we compare via a CDF. Figure 4a highlights accuracy of RTT estimates. Accuracy of RTT estimates also validates that ATHENA correctly identifies the lossy receiver as the worst receiver (cross-validated with logs at the multicast-sender).

TCP FRIENDLINESS. In Figure 4b we have a *multicast session* with competing TCP traffic on a single path from the sender to a particular receiver. Initially we have 3 parallel TCP sessions competing within themselves on this path. At 400 seconds, we start the *multicast session* for a 16GB file transfer. ATHENA correctly identifies the worst receiver for TCP-friendly rate control on this worst path it shares with 3 TCP sessions and adjusts its sending rate to co-exist with the other 3 TCP sessions. At around 800 seconds, two of the TCP sessions terminate. ATHENA quickly detects the reduced congestion along the path and increases its sending rate to occupy the available bandwidth while still co-existing with the remaining TCP session. At around 1120 seconds, the 16GB multicast file transfer completes reliably and the *multicast session* terminates and the TCP session occupies the maximum available bandwidth.

RELIABILITY. In this experiment we have a *multicast session* for a 2 GB file transfer with one of the receivers continuously dropping packets with a particular loss probability (loss rate). In Figure 4c we report



(a) RTT Estimate Accuracy. (b) TCP friendliness. (c) Reliability Stress Test.

Figure 4: ATHENA accuracy and guarantees evaluation.

time taken to complete file transfers despite the sustained packet loss rate. Even with a sustained 32% loss rate, a 2GB file transfer completes successfully in 251.44 seconds. Typically, maximum loss rates observed in data centers are around 12% [13] that too only during traffic bursts. This shows that ATHENA guarantees reliability.

SECURITY. In this experiment we attempt DoS from a malicious host by sending forged IGMP leave request for an unsuspecting receiver in a particular multicast group. ATHENA successfully thwarts this malicioius attempt. ATHENA detects attacks in real-time. On our physical testbed the attack detection (which is the single default policy) takes $\sim$124μsec. We also subject ATHENA to increasing number of policies (membership constraints) and analyze the time to verify an IGMP leave/join request. We observe that as the policies increase from 1 to 1K, the validation time increases from 124μsec to 345μsec. Even with 10K policies, ATHENA takes just 923μsec to complete verification of the corresponding IGMP request.

6 Related Work

Related research can be broadly categorized into multicast protocols designed for adoption in the Internet and those designed for adoption in data centers.

6.1 Internet Multicast

SRM [17], LBRM [20], RMTP [31], ARM [25], PGM [18], PGMCC [33], TFMCC [36], ERMCC [27] are some examples of the rich and varied literature that addresses multicast adoption in the Internet. Some of these Internet multicast schemes exclusively address reliability (SRM, LBRM, ARM, PGM) while some exclusively address congestion control (TFMCC, ERMCC) making them incomplete. Some of these Internet multicast schemes require state maintenance at intermediate network elements (ARM, PGM, PGMCC) and most assume multicast support to be offered by the intermediate switches. This is not the case in data centers and especially SDN based data centers where switches are forwarding entities with little intelligence. On a more general qualitative basis, Internet multicast schemes have a distributed locus of control leading to associated control overheads wherein network elements or end hosts have to exchange control traffic, via the data-plane, with each other to build data transmission multicast trees (as in IP multicast routing algorithms), identify repair delegates (as in RMTP, LBRM), build repair multicast trees etc. Consequently, Internet multicast schemes are not best suited as-is for data centers and particularly SDN based data centers wherein all the control decisions can be taken by the controller via the control-plane, entailing the need for a scheme that takes advantage of this by doing away with the overhead data-plane control traffic and also discarding the various requirements and assumptions inherent in them.

Moreover the rate control scheme of PGMCC, which proposes a window-based rate control mechanism, is much more prone to drop-to-zero problem in the presence of repeated worst-receiver changes. PGMCC measures RTT relatively in terms of advancement of the senders sequence number space with respect to the received `NAK` packets sequence number and does not yield accurate results. TFMCC requires a complex control loop involving echoing feedback from the receivers according to some priority order to achieve TCP-friendly rate adjustment and worst-receiver election.

6.2 Data Center Multicast

TCP is the de facto standard for reliable data delivery. Several reliable multicast schemes exist, which leverage TCP in the form of TCP overlay trees [10, 11, 35]. These schemes claim to perform well through analytical studies or simulations in the lab. However, in real data centers deployments, such overlay schemes, wherein synchronization (like TCP window synchronization across participating nodes) is necessary, have been found to

yield low throughput [14]. Basin et al. [12] identify inappropriate mathematical models that did not take into consideration infrequent short delays at nodes resulting in failure to send acknowledgments upstream and compounding synchronization failures, as the root cause of this marked difference between prediction and reality. RDCM [26] is a recent reliable data center multicast scheme that uses overlay point-to-point links for reliability. The congestion control scheme in RDCM is window-based and, in effect, requires window-based-acknowledgments from all the receivers to the sender every RTT. The scalability impact study by Chaintreau et al. [15] applies to the congestion control scheme in RDCM, which shows analytically that the throughput decreases in proportion to the inverse of the logarithm of the number of receivers in the presence of light tailed random noise. In essence, a congestion control scheme based on aggregated acknowledgments defeats any advantages obtained from an overlay based reliable multicast scheme as it hampers scaling of the multicast group due to increased compute-load on the sender with increasing number of receivers and increased network-overload on the last-hop link to the sender due to aggregated acknowledgment traffic from all the receivers.

Li et al. [28] address the scaling problem by partitioning the multicast address space and aggregating multicast based flow rules to enable switches to support 100000 multicast groups (as opposed to 1000 before). ATHENA also achieves similar improvement in scalability but by a different scheme that employs FDB based flow rules.

None of the prior works in data center multicast, to the best of our knowledge, address issues of multi-tenancy and security.

7 Conclusion

ATHENA enables secure, tenant-aware, reliable multicast for group communication while ensuring friendly coexistence with other applications by providing TCP-friendly congestion control. ATHENA makes a compelling case for data center deployment by improving the performance of OpenStack Nova VM provisioning service by upto 2.5 times.

A Appendix

Proof of ATHENA*'s TCP-friendliness:* Consider k *TCP* flows competing with *1* ATHENA flow. At time t_0 the complete network bandwidth (B) gets utilized and the intermediate network queues start buffering packets till at t_1 the queue is full and loss occurs. γ_{tcp}, γ and γ'_{tcp}, γ' denote sending rates before and after loss respectively.

$$\gamma(t_0) + \gamma_{tcp}(t_0) = B \tag{4}$$
$$\gamma(t) = \gamma(t_0) + (t - t_0)\Delta \tag{5}$$
$$\gamma_{tcp}(t) = \gamma_{tcp}(t_0) + (t - t_0)\Delta_{tcp} \tag{6}$$
$$\Delta_{tcp} = k\Delta \tag{7}$$
$$\Gamma_{loss} = \left(\frac{\gamma(t_1)}{\gamma(t_1) + \gamma_{tcp}(t_1)}\right) B \tag{8}$$
$$\gamma'(t_1) = \beta\Gamma_{loss} \tag{9}$$
$$\gamma'_{tcp}(t_1) = \frac{\gamma_{tcp}(t_1)}{2} \tag{10}$$

Using the above equations:

$$\begin{aligned}
&\frac{\gamma'(t_1)}{\gamma'_{tcp}(t_1)} - \frac{\gamma(t_0)}{\gamma_{tcp}(t_0)} \\
&= \frac{\beta\Gamma_{loss}}{\gamma'_{tcp}(t_1)} - \frac{\gamma(t_0)}{\gamma_{tcp}(t_0)}, \qquad \textit{now using } (8,\ 9,\ 10) \\
&= \frac{2\beta\gamma(t_1)B}{\gamma_{tcp}(t_1)(\gamma(t_1) + \gamma_{tcp}(t_1))} - \frac{\gamma(t_0)}{\gamma_{tcp}(t_0)} \\
&= \frac{2}{\gamma_{tcp}(t_1)}\left(\frac{\beta\gamma(t_1)B}{\gamma(t_1) + \gamma_{tcp}(t_1)} - \frac{\gamma(t_0)\gamma_{tcp}(t_1)}{2\gamma_{tcp}(t_0)}\right) \\
&= \frac{2}{\gamma_{tcp}(t_1)}\left(\frac{\beta(\gamma(t_0) + (t_1 - t_0)\Delta)B}{\gamma(t_1) + \gamma_{tcp}(t_1)}\right. \\
&\quad \left. - \frac{\gamma(t_0)(\gamma_{tcp}(t_0) + (t_1 - t_0)k\Delta)}{2\gamma_{tcp}(t_0)}\right), \textit{from } (5,\ 6,\ 7) \\
&= \frac{2}{\gamma_{tcp}(t_1)}\left(\gamma(t_0)\left[\frac{\beta B}{\gamma(t_1) + \gamma_{tcp}(t_1)} - \frac{1}{2}\right]\right. \\
&\quad \left. + \Delta(t_1 - t_0)\left[\frac{\beta B}{\gamma(t_1) + \gamma_{tcp}(t_1)} - \frac{k\gamma(t_0)}{2\gamma_{tcp}(t_0)}\right]\right)
\end{aligned} \tag{11}$$

Now if,

$$\frac{\beta B}{\gamma(t_1) + \gamma_{tcp}(t_1)} = \frac{1}{2}, \textit{ now using } (4,\ 5,\ 6,\ 7)$$
$$\beta B = \frac{B + \Delta(1 + k)(t_1 - t_0)}{2}$$
$$\textit{we get, } \beta = \frac{1}{2} + \frac{\Delta(1 + k)(t_1 - t_0)}{2B}$$

If value of β is 0.5 (second term negligible [1]), as also in our scheme, then by substitution in (11),

$$\begin{aligned}
&\frac{\gamma'(t_1)}{\gamma'_{tcp}(t_1)} - \frac{\gamma(t_0)}{\gamma_{tcp}(t_0)} = \frac{\Delta(t_1 - t_0)}{\gamma_{tcp}(t_1)}\left(1 - \frac{k\gamma(t_0)}{\gamma_{tcp}(t_0)}\right) \\
&\implies \textit{if } \frac{\gamma(t_0)}{\gamma_{tcp}(t_0)} > \frac{1}{k} \textit{ then } \frac{\gamma'(t_1)}{\gamma'_{tcp}(t_1)} < \frac{\gamma(t_0)}{\gamma_{tcp}(t_0)} \\
&\textit{and if } \frac{\gamma(t_0)}{\gamma_{tcp}(t_0)} < \frac{1}{k} \textit{ then } \frac{\gamma'(t_1)}{\gamma'_{tcp}(t_1)} > \frac{\gamma(t_0)}{\gamma_{tcp}(t_0)}
\end{aligned}$$

Thus *1* ATHENA*:k TCP* throughput approaches the ratio 1:k. ■

References

[1] Extended Paper Manuscript. https://www.dropbox.com/s/1yapbqjz02pbxms/paper.pdf.

[2] Hadoop. http://hadoop.apache.org/.

[3] IBM System Networking RackSwitch G8264. http://www-03.ibm.com/systems/networking/switches/rack/g8264/.

[4] IBM Websphere. http://www-01.ibm.com/software/websphere/.

[5] Open Networking Foundation. https://www.opennetworking.org/.

[6] Open Virtual Switch. http://openvswitch.org/.

[7] OpenDaylight. http://www.opendaylight.org/.

[8] OpenPGM. https://code.google.com/p/openpgm/.

[9] OpenStack. https://www.openstack.org/.

[10] BACCELLI, F., CHAINTREAU, A., ET AL. The one-to-many TCP overlay: A scalable and reliable multicast architecture. In *INFOCOM 2005*.

[11] BACCELLI, F., CHAINTREAU, A., LIU, Z., RIABOV, A., AND SAHU, S. Scalability of reliable group communication using overlays. In *INFOCOM 2004*.

[12] BASIN, D., ET AL. Sources of instability in data center multicast. In *Workshop on Large Scale Distributed Systems and Middleware 2010*.

[13] BENSON, T., ANAND, A., AKELLA, A., AND ZHANG, M. Understanding data center traffic characteristics. *SIGCOMM 2010*.

[14] BIRMAN, K., ET AL. Toward a cloud computing research agenda. *ACM SIGACT News 40*, 2 (2009), 68–80.

[15] CHAINTREAU, A., ET AL. Impact of network delay variations on multicast sessions with TCP-Like congestion control. In *INFOCOM 2001*.

[16] FARINACCI, D., LIU, C., DEERING, S., ET AL. Protocol independent multicast-sparse mode (PIM-SM): Protocol specification.

[17] FLOYD, S., JACOBSON, V., MCCANNE, S., LIU, C.-G., AND ZHANG, L. A reliable multicast framework for light-weight sessions and application level framing. In *SIGCOMM 1995*.

[18] GEMMELL, J., MONTGOMERY, T., SPEAKMAN, T., ET AL. The PGM reliable multicast protocol. *Network, IEEE 17*, 1 (2003), 16–22.

[19] GHEMAWAT, S., GOBIOFF, H., AND LEUNG, S.-T. The Google file system. In *SIGOPS 2003*.

[20] HOLBROOK, H. W., ET AL. Log-based receiver-reliable multicast for distributed interactive simulation. In *SIGCOMM 1995*.

[21] HONG, C.-Y., KANDULA, S., MAHAJAN, R., ET AL. Achieving high utilization with software-driven WAN. In *SIGCOMM 2013*.

[22] IMASE, M., AND WAXMAN, B. M. Dynamic Steiner tree problem. *SIAM Journal on Discrete Mathematics 1991*.

[23] IYER, A., KUMAR, P., AND MANN, V. Avalanche: Data center Multicast using software defined networking. In *COMSNETS 2014*.

[24] JAIN, S., KUMAR, A., MANDAL, S., ONG, J., ET AL. B4: Experience with a globally-deployed software defined WAN. In *SIGCOMM 2013*.

[25] LEHMAN, L.-W. H., GARLAND, S. J., AND TENNENHOUSE, D. L. Active reliable multicast. In *INFOCOM 1998*.

[26] LI, D., XU, M., CHEN ZHAO, M., GUO, C., ZHANG, Y., AND WU, M.-Y. RDCM: Reliable data center multicast. In *INFOCOM 2011*.

[27] LI, J., YUKSEL, M., AND KALYANARAMAN, S. Explicit rate multicast congestion control. *Computer Networks 2006*.

[28] LI, X., AND FREEDMAN, M. J. Scaling IP multicast on datacenter topologies. In *CoNEXT 2013*.

[29] MCKEOWN, N., ET AL. OpenFlow: enabling innovation in campus networks. *SIGCOMM 2008*.

[30] PATEL, P., BANSAL, D., YUAN, L., MURTHY, A., GREENBERG, A., MALTZ, D. A., KERN, R., KUMAR, H., ZIKOS, M., WU, H., ET AL. Ananta: cloud scale load balancing. In *SIGCOMM 2013*.

[31] PAUL, S., SABNANI, K. K., ET AL. Reliable multicast transport protocol (RMTP). *IEEE Journal on selected Areas in Communications 1997*.

[32] PENG, C., KIM, M., ZHANG, Z., AND LEI, H. Vdn: Virtual machine image distribution network for cloud data centers. In *INFOCOM 2012*.

[33] RIZZO, L. pgmcc: a TCP-friendly single-rate multicast congestion control scheme. In *SIGCOMM 2000*.

[34] THYAGARAJAN, A., CAIN, B., DEERING, S., ET AL. Internet Group Management Protocol, Version 3. Tech. rep., RFC 3376, October, 2002.

[35] URVOY-KELLER, G., AND BIERSACK, E. A congestion control model for multicast overlay networks and its performance. In *NGC* (2002).

[36] WIDMER, J., AND HANDLEY, M. TCP-friendly multicast congestion control (TFMCC): Protocol specification. Tech. rep., RFC 4654, August, 2006.