A Cross-Layer Multicast-Push Unicast-Pull (MPUP) Architecture for Reliable File-Stream Distribution

Shuoshuo Chen*, Xiang Ji*, Malathi Veeraraghavan*, Steve Emmerson†, Joseph Slezak‡, Steven G. Decker‡
* University of Virginia, Charlottesville, VA 22903, {sc7cq, xj4hm, mv5g}@virginia.edu
† University Corporation for Atmospheric Research, Boulder, CO, emmerson@ucar.edu
‡ Rutgers University, New Brunswick, NJ, jjs358@scarletmail.rutgers.edu

***Abstract*—The growing deployment of OpenFlow/SDN networks makes it increasingly possible to leverage network multicast services. This work proposes a novel cross-layer Multicast-Push Unicast Pull (MPUP) architecture that includes functionality in the application, transport and link layers to offer users a reliable file-stream distribution service to multiple subscribers. A prototype implementation of the MPUP architecture was realized in a new version of Local Data Manager (LDM), LDM7, a software program that has been in use since 1994 for real-time meteorology data distribution. LDM6, the currently deployed version, uses application-layer multicast. Experiments were run on the GENI infrastructure to compare LDM7 and LDM6. The two main findings are (i) LDM7 can be run at a higher sending rate than LDM6 allowing for improved performance (lower file-delivery latency), and (ii) to achieve the same performance, LDM7 uses significantly lower bandwidth and compute capacity. A three-fold improvement in performance improvement was possible with LDM7, and a bandwidth reduction from 350 Mbps to 21.4 Mbps was observed with 24 receivers.**

***Keywords*—Multicast; Internet applications; Software Defined Network; OpenFlow**

I. Introduction

There is a need in various domains to distribute file streams reliably to multiple receivers. A file stream is defined as a series of files of potentially varying sizes that arrive at random time intervals. Application-Layer Multicast (ALM) is commonly used for distributing data reliably to multiple receivers. For example, an application called Local Data Manager (LDM) [1], used to distribute near real-time meteorological data to multiple organizations, is an ALM solution.

While ALM is easier to deploy than network multicast solutions such as IP multicast, ALM consumes more network bandwidth and compute capacity at the sending hosts. For example, the University Corporation for Atmospheric Research (UCAR) uses LDM in a project called Internet Data Distribution (IDD) [2] to distribute 30 different types of file-streams, e.g., radar data and satellite data. UCAR receives 20 GB/hr from various input sources but transmits 1 TB/hr from its sending compute cluster because each file in each of multiple file-streams is repeated as many times as the number of subscribers. As the data volume and the number of subscribers in such data distribution projects grow, there is an increasing need to find solutions that scale the required resources (bandwidth and CPU capacity) more gradually.

Network multicast solutions have the advantage of requiring lower bandwidth and compute capacity because a sending host can transmit a single copy of a file in the form of packets, while a switch/router somewhere within the network can make multiple copies of the packets, and transmit these copied packets on to multiple ports. IP multicast has been the only network multicast solution available on Wide-Area Networks (WANs). However, distributed routing protocols, e.g., MSDP [3], which are used to spread reachability information for IP-multicast Class-D addresses, are complex [4], and have been difficult to deploy.

What has changed recently is the introduction of a new networking paradigm in the form of OpenFlow and Software Defined Networks (SDN) [5]. This paradigm promotes a more centralized approach in which a single SDN controller engages in control-plane communications with network switches using protocols such as OpenFlow. Inter-domain control protocols are being developed for SDN controllers in two different domains (autonomous systems) to communicate and jointly configure inter-domain paths [6]. This new paradigm could potentially be leveraged to create control-plane mechanisms for configuring flow-table entries within switches to realize multicast trees. User data can then be transferred from one sender to multiple receivers via such network multicast trees. The term OpenFlow Multicast (OFM) [7] has been used to describe this new network multicast solution.

This paper addresses the problem of what functionality is required at the transport and application layers to leverage network multicast solutions such as OFM for reliable file-stream distribution to multiple receivers. We propose a cross-layer Multicast-Push Unicast-Pull (MPUP) architecture that defines functions at three layers: (i) link layer, (ii) transport layer, and (iii) application layer. It combines (i) a multicast-push function at the transport layer, (ii) a unicast-pull function at the transport and application layers, and (iii) rate-guaranteed link-layer network multicast.

Contributions:

1) A cross-layer MPUP architecture for reliable file-stream distribution to multiple receivers.
2) A prototype implementation of the MPUP architecture called LDM7.
3) An experimental comparison of the performance and resource requirements of LDM6, which is an ALM implementation, with LDM7.

Novelty and significance: The proposed architecture is novel

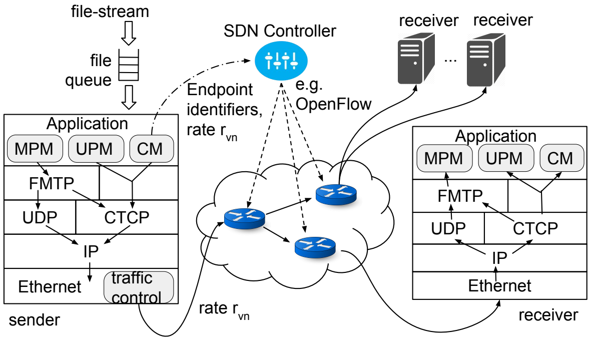


Fig. 1: Multicast Push Unicast Pull (MPUP) Architecture

in that it uses a cross-layer design to offer a high-performance solution for reliable file-stream distribution to multiple receivers by leveraging new network services on OpenFlow/SDN networks. Its significance lies in its ability to offer customers a solution in which bandwidth and compute capacity can be lowered while still achieving the same performance. Alternatively, for the same bandwidth and compute capacity, higher file-delivery performance (lower latency) is possible with the MPUP solution. For receivers who do not have access to the new OpenFlow/SDN network services, the sender can continue using the ALM solution for file-stream delivery. Therefore, the MPUP solution can be slowly expanded to cover an increasing number of receivers as the OpenFlow/SDN network services spread in availability.

Section II describes the cross-layer MPUP architecture. Section III describes a system model for the MPUP architecture, and defines metrics for performance evaluation of MPUP implementations. Section IV describes our prototype implementation of LDM7. Section V describes our experiments for a comparative evaluation of LDM6 (as a representative ALM solution) with LDM7 (a representative MPUP solution). Related work is reviewed in Section VI. Finally, key conclusions are provided in Section VII.

II. Cross-Layer Multicast-Push Unicast-Pull Architecture

Fig. 1 illustrates the cross-layer Multicast-Push Unicast-Pull (MPUP) architecture. The main systems in this architecture are the sender, multiple receivers, network switches, and an SDN controller. This solution assumes the availability of a rate-guaranteed multicast network service supported by the network switches and SDN controller. As shown in Fig. 1, a `file queue` is used to receive a stream of files from outside. A sender application reads and distributes files from the file queue to multiple receivers. The sender application has three modules: *Control Module (CM)*, *Multicast-Push Module (MPM)*, and a *Unicast-Pull Module (UPM)*.

The sender *CM* collects subscription requests for a file-stream from receiver CMs. The sender CM has knowledge of the traffic characteristics of the file-stream, based on which, it can select an appropriate rate r_{vn} for the multipoint virtual network. Endpoint identifiers and the computed rate r_{vn} are sent in a request message by the sender CM to the SDN controller as shown in Fig. 1. The SDN controller computes the multicast tree topology based on the specified endpoints, and sends control-plane messages (e.g., OpenFlow) to the switches to configure the multipoint virtual network. The SDN controller also sends the rate parameter r_{vn} to the switches.

Switches can be configured to offer multicast service on different fields of the packet header. One example is the Ethernet IEEE 802.1Q VLAN ID. A switch can replicate frames incoming on a particular port with a particular VLAN ID, translate the VLAN ID in each frame replica to a potentially different value for each outgoing port (as per the forwarding-table entry), and then forward the corresponding frame replica to each outgoing port. Other header fields such as Destination IP address and MultiProtocol Label Switching (MPLS) label field could also be used to realize the multipoint virtual network. The rate parameter is used to configure QoS mechanisms such as traffic policing on ingress ports, and scheduling/shaping on egress ports.

The sender application's *Multicast-Push Module (MPM)* uses File Multicast Transport Protocol (FMTP) [8] to provide reliable multicast service over the multipoint virtual network. FMTP is a protocol in which a file is divided into blocks that are sent over a multicast network tree from the sender to the receivers. FMTP uses the services of UDP and Circuit TCP (CTCP) as shown in Fig. 1. FMTP uses UDP datagrams to send its data blocks to an IP-multicast Class-D address, which is configured at the sender and all receivers. Even if the multipoint virtual network is realized at Layer-2, the IP layer is used at the endpoints for ease-of-programming with sockets. The rate of the virtual network could be lower than the sender Network Interface Card (NIC) rate, in which case traffic control is required in the Ethernet layer to limit sending rate as shown in Fig. 1. The FMTP blocks carried within UDP/IP datagrams are rate limited by the traffic control module so as not to exceed the virtual network rate r_{vn}.

Even though the multipoint virtual network is rate-guaranteed, which means packets should not be dropped in switch buffers, bit errors and receive-buffer overflows can occur leading to dropped packets. Dropped FMTP blocks are identified from the Block Sequence Number field carried in FMTP headers since the multipoint virtual network is assumed to guarantee in-sequence delivery. A receiving FMTP can thus send a retransmission request for an errored/dropped FMTP block upon receiving an out-of-sequence block. A CTCP connection is used for sending these requests, and receiving retransmissions of errored/dropped FMTP blocks. CTCP is a variant of TCP in which congestion control is disabled as it is designed for use over rate-guaranteed circuits [9]. Thus, using unicast CTCP connections, an FMTP sender offers FMTP receivers the opportunity to request and receive retransmissions of individual blocks within a file.

For performance reasons, FMTP limits the duration for which it serves retransmissions of lost/errored packets to individual receivers. Without such a limit, an FMTP sender could expend CPU cycles serving a few receivers with high packet-loss rates, while adversely affecting the multicast delivery of new files. Given that this MPUP solution is designed for file-streams, the tradeoff between file-delivery latency and successful file-delivery ratio needs to be considered in the design. To handle this tradeoff, FMTP uses a sender timeout factor f_{snd} to set a maximum retransmission period $\tau_{snd}(n)$ for each file n, given by

$$\tau_{snd}(n) = \max(f_{snd} * S_n/r_{mc}, \max_{1 \leq i \leq m} RTT_i) \qquad (1)$$

where S_n is the size of file n, r_{mc} is the rate used for multicasting FMTP blocks in UDP/IP datagrams, RTT_i is the Round-Trip Time of receiver i, and m is the number of receivers. A round-trip time is needed for the last data block to reach all receivers and for a receiver that missed this data block to send a block retransmission request to the sender. In wide-area networks, RTT could be higher (e.g., tens of ms) than the the first term in (1) if files are small and sending rates are high. A receiver's requests for blocks of a file with an expired $\tau_{snd}(n)$ will be rejected by the sender.

The role of the *Unicast Pull Module (UPM)* (see Fig. 1) is to handle cases when an FMTP receiver is unable to deliver a whole file to the application. Upon receiving a rejection for retransmission requests for blocks of a file, the FMTP receiver will drop all successfully received blocks of the file and notify the application of a dropped file. The UPM at the receiver will send a "pull" request for the dropped file to the UPM at the sender. The latter sends the dropped file over a separate (unicast) CTCP connection to the receiver.

One final aspect of FMTP is that a receive-side timer was added as part of this work. It was not present in the original FMTP specification [8]. This timer is required because retransmission requests will not be generated fast enough if all blocks at the end of a file are dropped, and there is a large silence period to the next file. FMTP uses a Begin-of-File (BOF) block and End-of-File (EOF) block, both of which are also multicast. Upon receiving a BOF, a receiver timeout value $\tau_{rcv}(n)$ is computed for file n using a factor f_{rcv} as follows:

$$\tau_{rcv}(n) = f_{rcv} \times S_n/r_{mc} \qquad (2)$$

If EOF for file n is not received within $\tau_{rcv}(n)$ of its BOF arrival, the FMTP receiver will generate retransmission requests for all missing data blocks. The presence of a File Identifier (File ID) in each FMTP block header, which is incremented by 1 for each file in a file stream, allows FMTP receivers to detect complete loss of a file, and to notify the application UPM. RTT does not appear in (2) because the timeout interval is between the reception of BOF and reception of EOF. The receiver timeout factor f_{rcv} should be an integer value greater than 1 so that if individual packets are caught in switch buffers behind other packets, the receiver avoids sending premature retransmission requests. Furthermore, f_{snd} should be set to a value larger than f_{rcv} so that the receiver's block retransmission requests for a file reach the sender before the sender timer corresponding to the file times out.

In *summary*, MPUP requires a rate-guaranteed multipoint virtual network interconnecting the sender and all receivers, a multicast FMTP/UDP/IP session running over this multipoint virtual network, and $2m$ CTCP connections. The Multicast Push Module (MPM) of the application uses the multicast FMTP/UDP/IP session. The first set of m CTCP connections are used by FMTP to handle dropped blocks within files, while the second set of m CTCP connections are shared by the Unicast Pull Module (UPM) and Control Module (CM) of the application. Pull requests for files and retransmissions of complete files are sent between UPMs, while file-stream subscription requests are sent between CMs.

TABLE I: Model Parameters

Parameter	Symbol
File-stream	**F**
No. of receivers	m
Round Trip Time	RTT_i
Packet loss rate	p_i
Base rate	r
Virtual network rate	r_{vn}
Sender multicast (mc) rate	r_{mc}
Sender unicast (uc) rate	r_{uc}
Sender traffic-control mc buffer size	b_{mc}
Sender traffic-control uc buffer size	b_{uc}
Receiver UDP buffer size	b_{rcv}
Sender CTCP congestion window	f_{cwnd}
FMTP sender timeout factor	f_{snd}
FMTP receiver timeout factor	f_{rcv}
Sender file queue size	q_{snd}
Sender limit on # files in file queue	n_{snd}

III. System model and metrics

Section III-A describes a model for MPUP systems. Section III-B describes metrics used to characterize the performance of an MPUP system.

A. System model

Table I lists the parameters of an MPUP system model. The file-stream arrival process **F** is represented by the notation (t_n, S_n), where the file inter-arrival times t_n could be unevenly spaced [10], and file sizes S_n could vary.

The number of receivers m is 2 or more since MPUP is used for multicast. The network path from a sender to a receiver i is characterized by three properties: RTT, packet loss rate and virtual-network rate, denoted by $\{RTT_i, p_i, r_{vn}\}, 1 \leq i \leq m$. Since the virtual-network rate r_{vn} is the same for all receivers, it is not indexed by i. Before explaining r_{vn}, we explain the term *base rate* r used in Table I. The rates used for LDM7 and LDM6 experiments are multiplicative factors of this base rate.

The traffic-control module within the sender Ethernet layer will be configured to rate limit multicast packets to a rate r_{mc}, and unicast packets to a rate, r_{uc}, such that $r_{uc} \leq r_{vn} - r_{mc}$. Since the application could send bursts of packets at rates higher than r_{mc} and r_{uc}, buffers of size b_{mc} and b_{uc} should be allocated in the traffic-control module to absorb the bursts.

The receiver UDP buffer size b_{rcv} should be made large enough to hold packets if the rate at which the application removes packets from the UDP buffer is lower than the packet arrival rate.

The next parameter listed in Table I, the sender CTCP congestion window f_{cwnd} is held at a fixed value that is slightly larger than Bandwidth-Delay Product (BDP) on the path with the highest RTT so that the sender can keep sending packets without waiting for acknowledgments. The next two parameters, FMTP sender timeout factor f_{snd} and FMTP receiver timeout factor f_{rcv}, were explained in Section II.

At the application layer, the sender file-queue size q_{snd} should be large enough to absorb bursts in the file-stream given the fixed sending rate at the lowest layer. An application could also limit the number of files n_{snd} in the file queue.

B. Metrics

Four metrics are defined: throughput, FMTP file delivery ratio (FFDR), sender NIC bandwidth usage and sender CPU utilization.

Throughput: A seemingly simple measure to evaluate our multicast service is latency, which is the time taken for a receiver to fully receive a file, if successful. This latency measurement would include the time for the original multicast and for FMTP block retransmissions if any were needed. However, latency depends on file size.

Therefore, a better measure is per-file throughput, which is defined as file size divided by latency. But averaging per-file throughput values across a set of files effectively gives an equal weight for the throughput values of all files in the set irrespective of their sizes. This could result in scenarios in which the average throughput is misleading.

A better representative metric is to compute file-set throughput at each receiver by summing file sizes over a set of file indices and summing corresponding latencies, and dividing these two sums. Then an averaging operation is performed to compute the average file-set throughput values across all receivers. This average metric is referred to as *throughput*, and defined as follows:

$$\Gamma_{(I_1,I_2)} = \frac{1}{m}\sum_{i=1}^{i=m} \frac{\sum_{j\in\mathbf{Z}_i} S_j}{\sum_{j\in\mathbf{Z}_i} D_{ij}} \tag{3}$$

where indices I_1 and I_2 are chosen such that $\sum_{k=I_1}^{k=(I_2-1)} S_k < G$ and $\sum_{k=I_1}^{k=I_2} S_k \geq G$, where G is the aggregate file-set (group) size, $\mathbf{Z}_i$ is the subset of files within file-set (I_1, I_2) that were successfully received at receiver i, and D_{ij} is the latency incurred for the delivery of file j to receiver i. Latency is measured from the time instant when the application provides a file to the FMTP layer at the sender to the time instant when the receiver sends a final acknowledgment for the file to the sender confirming that the whole file was successfully received. Since these two time instants are logged at different hosts, some clock synchronization technique, e.g., Network Time Protocol (NTP), is required to determine latency. File indices are used to characterize (average file-set) throughput on a rolling basis since the file-arrival process **F** is a time series.

A fixed time interval is not used as is commonly done for time series because there could be variability in the file arrival process from one time interval to another if the selected interval is too small. We also considered using a fixed number of files in each file-set. But for the reasons cited earlier, the interpretation of mean throughput could be wrong if the total size of files varies from one file-set to the next. Therefore, we chose to use aggregate file-set size as the defining parameter for computing rolling throughput values while allowing the time intervals and number of files to vary between file-sets.

FMTP File Delivery Ratio (FFDR): The metric FFDR is a measure of the success of file delivery by FMTP from a single sender to multiple receivers. A file j is said to have been delivered successfully to receiver i by FMTP if all blocks of file j were received by receiver i either via multicast or via the CTCP connection between the FMTP layers at the sender and receiver. The presence of the application-layer UPM ensures successful delivery of all files to all receivers as long as receivers request files within the specified duration for which files are served by the UPM. However, this FFDR metric captures the extent to which FMTP is successful in delivering files without the application-layer UPM. FFDR is defined as follows:

$$\Lambda_{(I_1,I_2)} = \frac{1}{m}\sum_{i=1}^{i=m} \frac{|\mathbf{Z}_i|}{N_{(I_1,I_2)}} \tag{4}$$

where $N_{(I_1,I_2)}$ is the number of files sent by the multicast sender with indices in the range (I_1, I_2). Thus FFDR is an average metric, with the averaging done across the ratios computed for all receivers.

Sender NIC bandwidth usage: This metric offers a measure of how much traffic is generated by the sender to support file multicasts and retransmissions. It is defined as follows:

$$\Theta(t_1,t_2) = \frac{\mathbb{B}(t_2) - \mathbb{B}(t_1)}{(t_2 - t_1)} \tag{5}$$

where t_1, t_2 are time instants, and $\mathbb{B}$ is a counter that tracks the number of bytes transmitted out by the sender NIC. The difference in the counter values at t_2 and at t_1 yields the number of bytes sent in the interval (t_1, t_2).

Sender CPU utilization: The ratio of the aggregate share of CPU time used by the sender-side processes (MPM, UFM and CM, as described in Section II) within a time interval (t_1, t_2) to the total CPU time available in the interval (t_1, t_2), expressed as a percentage, is sender CPU utilization, $\Phi(t_1, t_2)$.

IV. Prototype implementation

This section describes an implementation of the cross-layer MPUP architecture. Specifically, a new version of the Local Data Manager (LDM) application used for real-time meteorology data distribution was implemented. The new version is

LDM7. The currently used version LDM6 uses unicast TCP connections from the sender to each receiver. In other words, LDM6 is an ALM solution.

Three terms used in LDM are introduced here for usage in the rest of the paper: (i) "product" is synonymous with "file," (ii) "Product-Queue (PQ)" describes the file queue of the MPUP architecture (see Fig. 1), and (iii) "feedtype" is used to describe a file-stream.

To implement LDM7, the following modifications were made to LDM6: (i) added the interface to FMTP for reliable multicast (LDM7 is the application layer of the MPUP architecture of Fig. 1), (ii) modified the PQ component to support access to the product-queue from multiple threads, and (iii) added the UPM module of the MPUP architecture to allow a receiver to request a single product from the sender.

The FMTP design and implementation described in our prior work [8] was the first version of FMTP, FMTPv1. Modifications were required in FMTPv1 to support LDM7. Therefore, a new version FMTPv2 was implemented. It includes the following changes: (i) the addition of a receive-side timer, whose purpose was described in Section II, (ii) modification of the FMTP Application Programming Interface (API) at the sender to allow for FMTP to serve block retransmissions for a product directly from the PQ without creating its own local copy of the product, (iii) elimination of one user-space copy within LDM at the receiver, and (iv) improved support for delivery of file-streams. Further details on the FMTPv2 implementation are provided in a thesis [11].

In addition to coding FMTPv2 and LDM7, for testing purposes, a utility called pq_insert was created to emulate the real-world generation of data-products, both in terms of their creation-times and their sizes. First the notifyme utility was executed on a local host to collect IDD feedtype metadata (product creation-times and sizes). This utility connects to a UCAR LDM server and obtains the creation times (time instant when a product was injected into the IDD system) and product size for live feedtypes. The received metadata was stored in log files at the local host. The pq_insert utility reads these existing LDM log files, and uses the metadata size for each product to create a file filled with random bits (dummy data), which is then added to the PQ at the creation time for the product.

The pq_insert utility emulates LDM data-product ingesters, which receive data from radar sites or other such installations. These ingesters then create data products and insert these products into the PQ. The PQ is a memory-mapped structure that is shared by all processes of an LDM process group. Linux signals such as SIGCONT are used by the ingesters to notify the LDM process group when a new data product has been inserted into the PQ. The upstream LDM7 server receives the signal and then reads the product from the PQ and calls the SendProduct library function of FMTP to initiate multicast.

V. Evaluation

Section V-A describes the experimental testbed. Section V-B describes the input parameter values used in our experiments and the experimental workflow. Section V-C describes an experiment to determine an appropriate value to use for aggregate file-set size G, which is required for throughput computation (see Section III-B). The second metric, FMTP File Delivery Ratio (FFDR) (see Section III-B) is the focus of the experiment described in Section V-D. Section V-E compares LDM6 and LDM7 throughput and finds an operating point at which LDM6 and LDM7 achieve the same average throughput for use in the next experiment, which compares resource requirements between LDM6 and LDM7. This comparison is described in Section V-F.

A. Experimental setup

GENI [12], an NSF supported network testbed, was used to run all the experiments. Users are offered an interface to request slices, each of which consists of one or more virtual machines (VMs) that are interconnected by rate-specified VLANs. For our experiments, we created a slice consisting of VMs at five different racks located at (i) Wayne State University (WSU), Detroit, MI, (ii) StarLight (SL), Chicago, IL, (iii) Oakland Scientific Facility (OSF), Berkeley, CA, (iv) University of Massachussetts (UMass), Amherst, MA, and (v) University of Houston (UH), Houston, TX. The number of VMs at each rack was varied in our experiments to change the number of receivers in our multicast experiments. While the machines had different characteristics at the various racks, we provide an example of the type of resources used in these experiments. One of our VMs was assigned 2 cores (out of the 20 cores of an Intel Xeon E5-2660v2 processor in the physical machine), 6 GB RAM (out of 96 GB RAM in the physical machine), and 50 GB disk space (out of a large disk array on the physical machine). The physical machine had a 40 Gbps network interface card. A multipoint VLAN was stitched between the top-of-rack switches at these five racks, and the rate of the VLAN was set to 1 Gbps. In other words, r_{vn} was 1 Gbps.

The software used in our experiments consists of: (i) LDM6, (ii) LDM7, (iii) Linux traffic-control (tc) utility to control sending rate, (iv) Linux iptables to inject artificial packet losses, (v) Linux sar utility of the sysstat package to measure bandwidth usage, (vi) Linux ps to measure CPU utilization, (v) Python scripts to parse LDM log files for throughput and FFDR, and to parse log files created by the sar and ps programs for bandwidth usage and CPU utilization, respectively, and (vi) R programs to create graphs.

B. Experiment execution

Experiments were run to measure the performance and resource requirements of (i) LDM7: MPUP implementation, and (ii) LDM6: ALM implementation. We first explain how values were chosen for the input parameters, and then explain the experimental workflow.

Table II shows values used for the parameters in our experiment. The reader is referred to Table I for interpretations of the symbols.

TABLE II: Values for input parameters

Symbol	Value
F	NGRID 06/15/15 00:00-01:00
m	$\{4, \cdots, 24\}$
RTT_i	$\{36, 41, 50, 90\}$ ms
p_i	$\{0, 1\}$ %
r	$\{10, \cdots, 60\}$ Mbps
r_{vn}	$\geq r_{mc} + r_{uc}$ (LDM7) and $\geq m \times r$ (LDM6)
r_{mc}	r
r_{uc}	r
b_{mc}	600 MB
b_{uc}	600 MB
b_{rcv}	b_{mc}
f_{cwnd}	$1.2 \times r_{uc} \times (\max_{1 \leq i \leq m} RTT_i)$
f_{snd}	5000
f_{rcv}	20
q_{snd}	5 GB
n_{snd}	35000

A data analysis of five IDD feedtypes showed that both file inter-arrival times and file sizes have long-tailed right-skewed distributions [13]. Of the analyzed feedtypes, we chose NGRID as a representative file-stream with which to compare LDM7 and LDM6 performance and resource requirements. Specifically, we collected metadata for 7 hours (12 AM to 7 AM) of the NGRID feedtype on June 15, 2015. As described in Section IV, the real metadata collected for the NGRID feedtype was used as input to the pq_insert program to create dummy data products with the corresponding creation times and sizes.

The number of receivers was varied from 4 to 24 in steps of 4. One VM on a UMass server was used as the upstream LDM server (sender) and the VMs at the remaining four sites were used for the downstream LDM servers (receivers). If the total number of receivers m used in an experiment was four, one VM was used in each rack. Correspondingly for $m = 24$, six VMs were used in each rack. The RTT values from a UMass VM to VMs in WSU, SL, UH, and OSF, were, as indicated in Table II, 36, 41, 50, and 90 ms, respectively. To compare the performance of LDM6 and LDM7 under lossy conditions, random artificial packet drops were injected at all receivers using Linux iptables using the loss rates specified in Table II.

The Linux tc utility was used for the traffic control module shown in Fig. 1. A combination Hierarchical Token Bucket (HTB) and Bytes First In First Out (BFIFO) queueing disciplines of tc were used for LDM7, while Token Bucket Filter (TBF) was used for LDM6. For LDM7, two queues were needed for multicast and retransmissions, and hence HTB was used. The UDP datagrams were directed to one queue, while packets from all $2m$ CTCP connections were directed to the second queue. Borrowing of bandwidth between the queues was disabled because FMTP does not implement flow control, and therefore a high sending rate could overwhelm the UDP buffer at the receivers.

Analysis of the NGRID feedtype metadata in our prior work [13] showed that a sending rate of 10 Mbps and a sending buffer size of 300 MB was sufficient to meet a specified throughput value. Therefore, we used 10 Mbps as a starting value for r_{mc}, but doubled the buffer size b_{mc} to 600 MB to ensure that no packets were dropped by the tc buffer at the sender. A dropped packet at the sender will require retransmissions for all the receivers, and is hence avoided. The rate r_{mc} was set as the HTB rate and ceil parameters for both queues, while the 600 MB buffer size was set in the BFIFO parameter. The same values were used for rate r_{uc} and buffer size b_{uc} with a rough estimation that if loss rate was 1% and there were 100 receivers, on average, all blocks of all files would be retransmitted once.

For LDM6, the TBF parameters were set as follows: rate = $m \times r$, burst = 50 KB, and limit = $m \times 2 \times r \times (\max_{1 \leq i \leq m} RTT_i)$ [14]. All $2m$ CTCP connections from the sender to the m receivers were directed to the same queue, whose size is given by the limit TBF parameter minus the burst (token-bucket) size. Even for the high r rate of 60 Mbps, when $m = 24$, the TBF limit value is only 32.4 MB, since the maximum RTT is 90 ms. Compare this 32.4 MB value with the 600 MB value used for the tc queues for LDM7. The difference is because UDP and TCP react differently to a full Ethernet-layer tc queue. Experiments showed that in the case of UDP, if the tc queue is full, packets are simply dropped, whereas TCP will block and hold the packets in its own buffer. Furthermore, if FMTP tries writing to a TCP socket with a full TCP buffer, FMTP will be blocked. Therefore, in LDM6, queueing delays occur only in the PQ, but in LDM7, queueing delays also occur in the tc queue. Through experimentation, we found that 600 MB was sufficient to ensure 0 dropped packets by the sender tc module for the NGRID 1-hour feedtype used.

The UDP buffer size b_{rcv} at the receiver was also set to 600 MB for LDM7. This large value was chosen to limit packet losses due to flow-control problems at the receiver. With this choice, there were no packet drops at any of the receivers in the experiments in which no artificial packet losses were injected.

The fixed congestion window (f_{cwnd}) parameter of CTCP was set to 20% more than the maximum bandwidth-delay product to ensure continuous sending of segments without waiting for an acknowledgment. The same f_{cwnd} value is applied to all (system-wide) CTCP sockets since this value is set in Linux sysctl. The Linux setsockopt function can be used to set (possibly different) per-socket values for f_{cwnd}, but this requires modification of the application code.

Values for the two FMTP time-out factors, f_{snd} and f_{rcv}, were selected as follows. The receiver factor f_{rcv} was set to 20 with the expectation that even if many packets from other flows become interspersed between two packets of a given

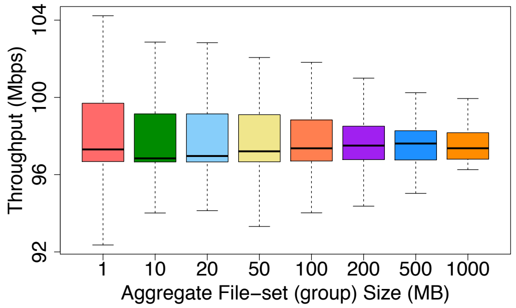


Fig. 2: Impact of Aggregate File-Set (Group) Size G on Throughput

product, a multiplicative factor of 20 applied to the product transmission delay is sufficient for delivery of all blocks of a product. The sender time-out factor f_{snd} was selected after some experimentation. To avoid packet loss at the sender `tc` queue, f_{snd} was chosen to be the large value of 5000.

The LDM PQ has two parameters: q_{snd} and n_{snd}, which represent the maximum size of the PQ in bytes, and the maximum number of files that can be stored in the PQ. If a newly arriving file causes either of these limits to be exceeded, one or more of the oldest files will be deleted to make space for the new file. We selected the values for these parameters to hold approximately 1-hour of the NGRID file-stream.

The *experimental workflow* consists of four steps: (i) upload LDM6 and LDM7 software, and configuration files (for tc, FMTP, CTCP, and LDM), to the remote GENI VMs from a local host, (ii) run the software and monitoring tools on the GENI VMs, (iii) download collected logs from the GENI VMs to a local host, and (iv) run the log parsers to extract performance measures. A script was used for automated execution of this workflow. The monitoring tools, `sar` for bandwidth usage, and `ps` for CPU utilization, run as a Linux `cron` job that is executed every min while the LDM processes are running. The log files include bandwidth logs, CPU logs, and LDM logs. The log parsers extract the four metrics: throughput, FFDR, bandwidth usage, and CPU utilization.

C. Experiment 1: Determine file-set size for throughput metric

Section III-B defined a metric called throughput on file-sets rather than single files. Per-receiver file-set throughput was defined as the effective rate at which files within a file-set of size G were received. The purpose of Experiment 1 is to determine an appropriate size for G.

An LDM7 run was executed to send products whose sizes and creation times were extracted from 7 hours of the NGRID feedtype[1]. A single receiver was used in this run. Specifically, the data was sent by LDM7 from a UMass host to an SL host. No artificial losses were injected. The rate r_{mc} was set to 100 Mbps.

The throughput metric was computed using (3) for different values of G, starting from 1 MB. The side-by-side boxplots in Fig. 2 shows the throughput variability across file-sets (groups) for each setting of G. When G was 1 MB, there were 17631 groups within the 7-hour input file-stream. For larger values of G, the total number of groups decreases because the aggregate size of all files in the 7-hour file-stream is fixed (31.5 GB). When $G = 200$ MB, there were only 150 groups. While the median throughput did not change much for different values of G, the Inter-Quartile Range (IQR) decreased as G increased. For example, IQR was 3.02 Mbps when G was 1 MB, while it dropped to 1.73 Mbps for a G value of 200 MB.

Ideally, the selected group size should have a small IQR; however, the number of groups drops as G increases, which then limits the number of computed throughput values within a single run of the experiment. With a G value of 200 MB, the number of groups in the 1-hour file-stream used in the remaining experiments, was 27, which is still large enough for averaging operations. Therefore, we chose $G = 200$ MB as a good compromise between these two opposing factors.

D. Experiment 2: FMTP File Delivery Ratio (FFDR)

FFDR is a metric that only applies to LDM7. In this experiment, LDM7 was executed with 8 and 16 receivers, using a base rate r of 20 Mbps. With the default setting for f_{snd} of 5000, when no artificial packet drops were injected at the receivers, FFDR was 100% in all our experiments. However, when artificial packet drops were injected at receivers, FFDR dropped below 100%, which means receivers will need to seek retransmissions of files from the UPM in the sending application.

Fig. 3 shows when packet loss rate was set to 1%, FFDR was 88.7% and 75.9% when the number of receivers was 8 and 16, respectively.

E. Experiment 3: Throughput measurement

The goals of this experiment were two-fold: (i) compare throughput of LDM6 and LDM7, (ii) find a suitable value for the base rate r to achieve the same throughput with LDM6

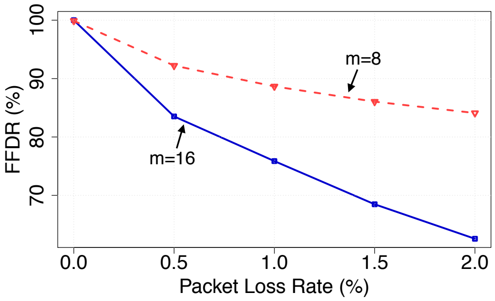
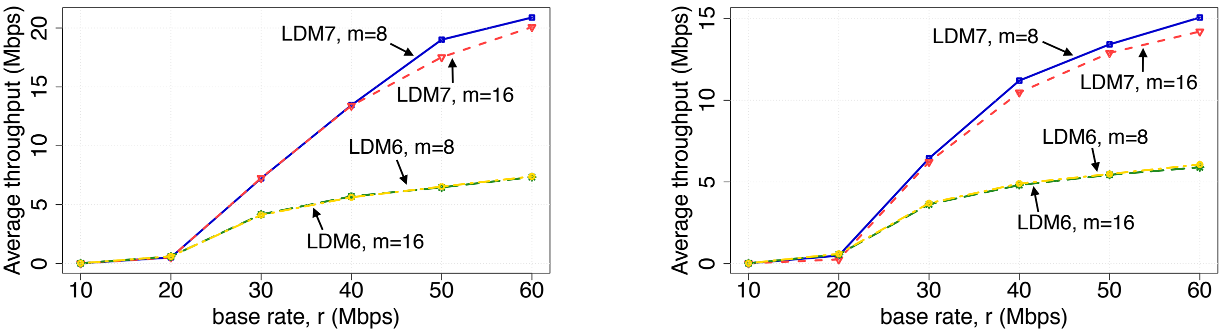


Fig. 3: FMTP File Delivery Ratio (FFDR) for LDM7

[1] For only this experiment, we used a 7-hour clip of the NGRID feedtype, specifically 00:00-07:00 on June 15, 2015, while for all other experiments only the first hour data was used as listed in Table II.



(a) No artificial packet loss injections ("lossless")

(b) Packets dropped randomly at 1% at each receiver ("lossy")

Fig. 4: Impact of tc rate limiting on throughput under lossless/lossy conditions; m: number of receivers

and LDM7 so as to enable a fair comparison of resource requirements. The main parameter that was varied in this experiment was the base rate r.

This experiment used four base configurations: (i) LDM7 with 8 receivers, (ii) LDM7 with 16 receivers, (iii) LDM6 with 8 receivers, and (iv) LDM6 with 16 receivers, with two variants: lossless and lossy, for a total of 16 configurations. In the lossless variant, no artificial packet drops were injected in these runs, while in the lossy variant, 1% random packet loss was injected at all receivers. For each of these 8 configurations, 6 runs were executed corresponding to base rate settings from 10 Mbps to 60 Mbps, for a total of 48 runs. For each of these 48 runs, throughput values were obtained for each of the 27 file-sets of size 200 MB in the 1-hour file-stream. The average throughput across the 27 file-sets was computed for each setting and plotted against the base rate r in Fig. 4.

Towards meeting our first goal of comparing LDM6 and LDM7 throughput, Fig. 4 shows that LDM7 is able to take better advantage of higher rates than LDM6. This is because LDM6 needs to create copies of each product and send each copy individually to each receiver, while LDM7 sends out a single file-stream to all receivers.

When the base rate is 60 Mbps, and the number of receivers is 16, the total TBF `tc` rate ($m \times r$; see Section V-B) is 960 Mbps for LDM6. Recall the 1 Gbps limit for r_{vn} in our GENI experimental slice as mentioned in Section V-A. On the other hand, with LDM7, the sender multicast rate r_{mc}, which is equal to the base rate r, could be increased to values above 60 Mbps. But already at the base rate setting of 60 Mbps, average throughput is three-fold better for LDM7 than LDM6 as see in Fig. 4a. Increasing the base rate further yielded an average throughput of 75 Mbps, which is a ten-fold increase. This observation becomes important in applications that require a low file-delivery latency. ALM solutions can only support relatively low rates per receiver because of file-stream replication. The higher the number of receivers, the smaller the bandwidth available for any single receiver, which in turn adversely affects file-delivery latency. On the other hand, LDM7 can use a significantly higher-rate virtual network to deliver files with lower latency.

Our second goal was to find an appropriate setting of parameters at which LDM6 and LDM7 achieve the same average throughput for use in our resource-requirements comparison experiment. Fig. 4 shows that the 20 Mbps base-rate setting is a good choice as it yields the same throughput for LDM6 and LDM7 under both lossless and lossy conditions.

F. Experiment 4: Resource requirements comparison

LDM6 and LDM7 were executed with several combinations of parameter settings. The number of receivers was varied from 4 to 24 as described in Section V-B. The same lossless and lossy settings described in Section V-E were used in these experiments.

Three sets of results are shown: (i) per-min comparison of bandwidth usage by LDM6 and LDM7, (ii) time-averaged bandwidth comparison between LDM6 and LDM7 as a function of the number of receivers, and (iii) time-averaged CPU utilization as a function of the number of receivers.

Fig. 5 shows the per-min bandwidth usage on the sender NIC as a function of time, and the aggregate size of all products created in each min. LDM6 uses more bandwidth

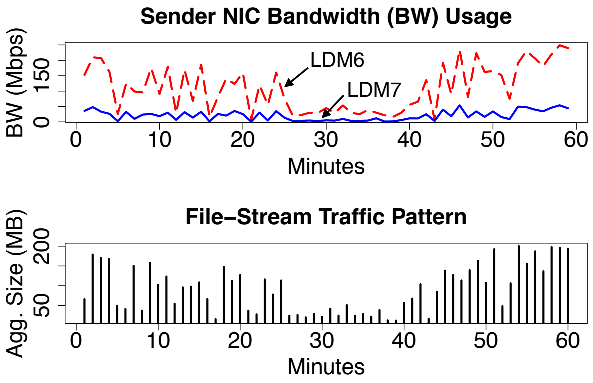


Fig. 5: Experimental settings: m = 8 receivers, lossless setting

than LDM7 as seen in the top graph of Fig. 5. This is simply because LDM6 is sending eight copies of each product, while LDM7 is sending only a single copy of each product along with a few block retransmissions. A second observation is that bandwidth usage for both LDM6 and LDM7 drops between mins 25 and 40. This drop is explained in lower graph of Fig. 5, which shows that the NGRID traffic pattern had smaller aggregate per-min size of products between these mins than in other time intervals. Having shown this per-min variation in bandwidth usage, we can now use the time-averaged bandwidth in the next figure.

Fig. 6a shows a time-averaged bandwidth comparison between LDM6 and LDM7 as a function of the number of receivers. Since LDM6 needs to send m copies of all products, it consumes more bandwidth than LDM7, and further the difference in bandwidth requirement between LDM6 and LDM7 increases with the number of receivers. For LDM7, bandwidth usage is almost independent of the number of receivers and is about 12.4 Mbps and 21.4 Mbps for the 0% and 1% cases settings, respectively. While the multicast rate r_{mc} is 20 Mbps, because of silence periods between products, and smaller product sizes in some mins as seen in Fig. 5, the time-averaged bandwidth usage is less than r_{mc}. Data analysis of the 1-hour NGRID file-stream shows that while the third-quartile for file inter-arrival time is 34 ms, the 90% is 297 ms, and 1% of files arrive more than 1.8 sec after their predecessors. In other words, there are gaps between file arrivals, which explains why the NIC was not used all the time even at the low rate setting of 20 Mbps. Finally, we observe that LDM6 bandwidth usage increases linearly, reaching 350 Mbps for 24 receivers. NGRID is just one of 30 feedtypes distributed by the IDD project, and the number of subscribers, which is currently 240, is growing. Therefore the total bandwidth required on the UCAR WAN access link for the IDD project is already high, and growing. Use of LDM7 will offer a significant reduction in the access-link bandwidth and cluster needed to support real-time data distribution.

Figs. 6b plots time-averaged sender CPU utilization for LDM6 and LDM7 as a function of number of receivers. Under both lossless and lossy settings, LDM6 enjoys lower sender CPU utilization when the number of receivers, m, is 4. This is because FMTP runs in user space, which incurs more CPU cycles than CTCP used by LDM6, which runs in the kernel. However, with larger numbers of receivers, CPU utilization increases rapidly for LDM6, while it stays almost flat for LDM7. This is because there is a separate upstream LDM process corresponding to each receiver in LDM6. While LDM7 also requires one sender process per receiver, these sender processes just handle retransmission requests and hence do not consume much CPU time. Thus, the total CPU utilization for LDM6 is an almost linear function of m, while for LDM7, CPU utilization is almost independent of m. In the lossy setting, higher CPU time is needed for both LDM6 and LDM7. Since the random packet drops were injected at all receivers, the additional CPU time needed at the sender increased linearly with the number of receivers.

In *summary*, it is clear that LDM7 requires fewer bandwidth and CPU resources than LDM6, with the resource savings increasing almost linearly with the number of receivers.

VI. Related work

A Software Defined Network Aware Pub/Sub (SAPS) solution that uses a hybrid approach with both Application Layer Multicast (ALM) and OpenFlow based multicast (OFM) was proposed for IoT/M2M and other applications [7]. The messages sent to OFM use UDP, while ALM messages use TCP. Message latency and the total number of messages exchanged were the parameters of interest. Our work addresses reliability aspects in FMTP and in the UPM module of the application to guarantee delivery of files.

Another recent solution called Rateless Code based Reliable Multicast (RCRM) [15] protocol builds on Data Distribution Service (DDS), which uses the Publish-Subscribe model. RCRM removes the use of heartbeats in DDS, and instead uses ACKs that are sent after many received messages are decoded. It relies on a form of FEC coding for reliability. In contrast, FMTP does not use coding as it is designed for smaller numbers of receivers (in the hundreds); instead it uses negative acknowledgments to obtain retransmissions on unicast connections.

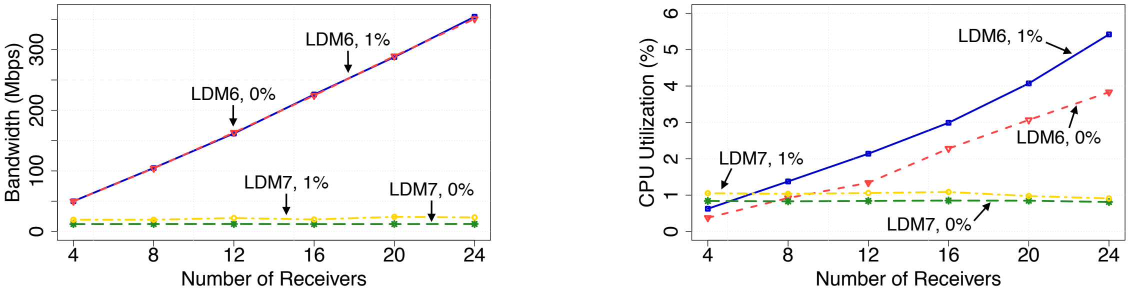
There are two IETF reliable multicast solutions: NACK-Oriented Reliable Multicast (NORM) [16] and Asynchronous Layered Coding (ALC) [17]. NORM is an NACK-based reliable multicast protocol that uses multicast in the first transmission attempt, and then either multicast or unicast for retransmitting lost packets. NORM is designed for IP-multicast, and since IP networks do not offer rate guarantees, NORM includes a rate-based congestion control mechanism. FMTP avoids data-plane congestion control by using SDN controllers to configure a rate-guaranteed multipoint VLAN.

Asynchronous Layered Coding (ALC) [17] is a massively scalable reliable content delivery protocol. The data is sent on multiple channels at different rates, and encoded with FEC for reliability. Receivers can obtain packets from multiple channels. There are no positive or negative ACKs from receivers. Hence this solution scales to a million receivers. FLUTE [18] is a protocol for unidirectional delivery of files to multiple receivers. It is built on ALC and is therefore massively scalable. FMTP is designed for scenarios with hundreds of receivers, not millions of receivers, and hence it avoids the overhead of sending to multiple channels and FEC.

We already compared LDM7, an implementation of our MPUP architecture to an ALM implementation, LDM6. However, there are other ALM approaches such as P2P, an example of which is Bit Torrent [19]. For file-streams carrying new data, P2P solutions are not ideal as they require multiple downloads before all receivers can receive all blocks. If latency is a consideration, a network multicast solution such as our MPUP architecture is more suitable.

VII. Summary and Conclusions

This paper proposed and evaluated a cross-layer multicast-push unicast-pull (MPUP) architecture for reliable file-stream



(a) Time-averaged sender NIC bandwidth usage

(b) Time-averaged sender CPU utilization

Fig. 6: Resource utilization; r = 20 Mbps; (LDM version, loss rate) are shown for each plot

distribution. The architecture combines a rate-guaranteed multipoint virtual network service, a reliable File Multicast Transport Protocol (FMTP) that uses a multicast-push with UDP and unicast-pull with Circuit TCP (CTCP), and an application-layer unicast-pull module. A new version of the Local Data Manager (LDM) application, LDM7, was implemented based on the MPUP architecture. LDM6, the current version, uses application-layer multicast to send near real-time file-streams of meteorology data to 240 institutional subscribers. LDM7 and LDM6 performance and resource requirements were compared in an experimental evaluation. First, LDM7 achieves higher throughput (lower latency) in file delivery when compared to LDM6 since the sending rate can be higher for LDM7. Second, for a given sending rate, at which both LDM7 and LDM6 yield the same throughput performance, the sender NIC bandwidth usage and sender CPU utilization for LDM6 increase linearly with the number of receivers in the multicast group, while both metrics stay almost constant for LDM7. For example, with 24 receivers, to serve a particular filestream, LDM6 needed 350 Mbps, while LDM7 needed only 12.4 Mbps in a lossless setting and 21.4 Mbps in a lossy setting.

Acknowledgment

This work is supported by NSF grants CNS-1116081, OCI-1127340, ACI-1340910, CNS-1405171, and CNS-1531065, and U.S. DOE grant DE-SC0011358.

References

[1] Local Data Manager. [Online]. Available: http://www.unidata.ucar.edu/software/ldm/

[2] UCAR. Unidata Internet Data Distribution. [Online]. Available: http://www.unidata.ucar.edu/software/idd/

[3] B. Fenner and D. Meyer, "Multicast Source Discovery Protocol (MSDP)," RFC 3618 (Experimental), Internet Engineering Task Force, Oct. 2003. [Online]. Available: http://www.ietf.org/rfc/rfc3618.txt

[4] S. Ratnasamy, A. Ermolinskiy, and S. Shenker, "Revisiting IP multicast," in *Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications*, ser. SIGCOMM '06. New York, NY, USA: ACM, 2006, pp. 15–26.

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

[6] On-Demand Secure Circuits and Advance Reservation System (OSCARS). http://www.es.net/OSCARS/docs/index.html.

[7] T. Akiyama, Y. Kawai, Y. Teranishi, R. Banno, and K. Iida, "SAPS: Software defined network aware pub/sub – a design of the hybrid architecture utilizing distributed and centralized multicast," in *Computer Software and Applications Conference (COMPSAC), 2015 IEEE 39th Annual*, vol. 2, July 2015, pp. 361–366.

[8] J. Li, M. Veeraraghavan, S. Emmerson, and R. Russell, "File multicast transport protocol (FMTP)," in *Cluster, Cloud and Grid Computing (CCGrid), 2015 15th IEEE/ACM International Symposium on*, May 2015, pp. 1037–1046.

[9] A. Mudambi, X. Zheng, and M. Veeraraghavan, "A transport protocol for dedicated end-to-end circuits," in *Communications, 2006. ICC '06. IEEE International Conference on*, vol. 1, June 2006, pp. 18–23.

[10] L. Nieto-Barajas and T. Sinha, "Bayesian interpolation of unequally spaced time series," *Stochastic Environmental Research and Risk Assessment*, vol. 29, no. 2, pp. 577–587, 2015. [Online]. Available: http://dx.doi.org/10.1007/s00477-014-0894-3

[11] S. Chen, "A cross-layer architecture and protocols for reliable file-stream distribution," diploma thesis, University of Virginia, May 2016.

[12] M. Berman, J. S. Chase, L. Landweber, A. Nakao, M. Ott, D. Raychaudhuri, R. Ricci, and I. Seskar, "GENI: A federated testbed for innovative network experiments," *Computer Networks*, vol. 61, pp. 5 – 23, 2014, special issue on Future Internet Testbeds Part I. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128613004507

[13] X. Ji, Y. Liang, M. Veeraraghavan, and S. Emmerson, "File-stream distribution application on software-defined networks (SDN)," in *Computer Software and Applications Conference (COMPSAC), 2015 IEEE 39th Annual*, vol. 2, July 2015, pp. 377–386.

[14] S. Tepsuporn, F. Al-Ali, M. Veeraraghavan, X. Ji, B. Cashman, A. J. Ragusa, L. Fowler, C. Guok, T. Lehman, and X. Yang, "A multi-domain SDN for dynamic layer-2 path service," in *Proceedings of the Fifth International Workshop on Network-Aware Data Management*, ser. NDM '15. New York, NY, USA: ACM, 2015, pp. 2:1–2:8. [Online]. Available: http://doi.acm.org/10.1145/2832099.2832101

[15] K.-h. Lee, C.-k. Kim, S.-H. Lee, and W.-t. Kim, "Rateless code based reliable multicast for data distribution service," in *Big Data and Smart Computing (BigComp), 2015 International Conference on*. IEEE, 2015, pp. 150–156.

[16] B. Adamson, C. Bormann, M. Handley, and J. Macker, "NACK-Oriented Reliable Multicast (NORM) Transport Protocol," RFC 5740 (Proposed Standard), Internet Engineering Task Force, Nov. 2009. [Online]. Available: http://www.ietf.org/rfc/rfc5740.txt

[17] M. Luby, M. Watson, and L. Vicisano, "Asynchronous Layered Coding (ALC) Protocol Instantiation," RFC 5775 (Proposed Standard), Internet Engineering Task Force, Apr. 2010. [Online]. Available: http://www.ietf.org/rfc/rfc5775.txt

[18] T. Paila, R. Walsh, M. Luby, V. Roca, and R. Lehtonen, "FLUTE - File Delivery over Unidirectional Transport," RFC 6726 (Proposed Standard), Internet Engineering Task Force, Nov. 2012. [Online]. Available: http://www.ietf.org/rfc/rfc6726.txt

[19] B. Cohen, "Incentives build robustness in BitTorrent," in *Workshop on Economics of Peer-to-Peer systems*, vol. 6, 2003, pp. 68–72.