Sources of Instability in Data Center Multicast

Dmitry Basin
Dept. of Electrical Engineering
Technion, Haifa, Israel
sdimbsn@tx.technion.ac.il

Ken Birman
Cornell University
Ithaca, NY, USA
ken@cs.cornell.edu

Idit Keidar
Dept. of Electrical Engineering
Technion, Haifa, Israel
idish@ee.technion.ac.il

Ymir Vigfusson
IBM Research Haifa Labs
Haifa University Campus,
Haifa, Israel
ymirv@il.ibm.com

ABSTRACT

When replicating data in a cloud computing setting, it is common to send updates using reliable dissemination mechanisms such as network overlay trees. We show that as data centers scale up, such multicast schemes manifest various performance and stability problems.

Categories and Subject Descriptors

C.2.2 [**Computer Systems Organization**]: Computer-communication networks— *Network Protocols*; C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Distributed applications, Cloud computing*

General Terms

Reliability, Performance

Keywords

Reliable multicast, Cloud computing

1. INTRODUCTION

Data centers, and particularly massive ones that support cloud computing, e-commerce, and social networking, necessarily replicate data. Replication is done for many reasons: to provision financial servers that respond to read-mostly requests, to parallelize computation, to cache important data items, for fault-tolerance, and the list goes on.

Since updates to replicated data can be thought of as reliable multicasts, data center multicast is an important technology. Nonetheless, a series of recent keynote speeches at major conferences makes it clear that data center multicast is a troubled area [19, 1, 6]. One might expect data centers to use IP multicast, but in fact this is rare, in part due to concerns about flow control [18, 1]. Only TCP is really trusted today (because it backs down when loss occurs), and indeed, TCP is the overwhelming favorite among data center transport protocols [10, 6]. Using TCP to get reliable multicast with high throughput produces an implicit *TCP overlay tree*. But although such trees are easy to build, and were repeatedly shown to work well analytically, in simulations, or "in the lab" [2, 21, 3], in real data centers, they have been reported to exhibit low throughput [6].

In this paper we probe the root cause of this gap between predictions and reality. We study via mathematical analysis and simulations the use of overlay trees composed of reliable point-to-point links e.g., using TCP, for reliable multicast; our multicast model is defined in Section 2. We show that multicast oscillations and throughput collapse may result from infrequent short delays, (e.g., of a tenth of a second), which were not modeled or simulated before. Nodes can experience such disturbances for a variety of reasons, such as Java garbage collection pauses, Linux scheduling delays, or flushing data to disk.

In Section 3, we present an analytical model capturing such disturbances. We prove an upper bound on the aggregate multicast throughput achievable in overlay trees in the presence of short disturbances. The limiting factor for performance is back-pressure, which is caused when disturbed nodes fail to send acknowledgements upstream. This occurs even in the absence of message loss, and without any retransmissions. In Section 4, we conduct simulations to validate our analysis. For the purpose of again obtaining an upper bound, we simulate an environment with ideal flow and congestion control, without TCP's slow start or retransmissions.

In Section 5, we present results from our analysis and simulations. We find that in large trees, ($10K$-$60K$ nodes, a size not unreasonable in cloud settings), when each node is disturbed for one second every hour on average, throughput degradation (up to 90%) occurs even if message loss is negligible. Under default TCP configurations in low-latency networks, message loss can cause problems even in very small trees: we demonstrate that multicast throughput collapses with as few as $30 - 100$ nodes if switches become congested.

Our paper thus explains why overlay trees built from reliable point-to-point links (possibly, but not necessarily TCP links) are quite likely to perform poorly, at least if implemented naïvely. Note that we do not claim that data center multicast must inherently be slow; indeed, approaches to overcome the limitations shown by our analysis exist. Rather, we refine the model for reasoning about such sys-

LADIS '10 Zürich, Switzerland

tems so as to capture phenomena that have a severe impact on performance in practice, and were previously overlooked. This approach may thus prove useful in future work for analyzing solutions that remedy the problems we highlight. Our conclusions appear in Section 7.

2. MULTICAST MODEL

The multicast system consists of N nodes, interconnected by a high throughput low latency network with maximal bandwidth T_{max} per link. We assume that each node has one NIC, and hence can send at a combined rate of at most T_{max} on all links. We also studied the case of multiple NICs and obtained similar results, which are omitted due to lack of space. Consistent with previous work [2, 21, 15, 7], we assume that the nodes are structured into a balanced k-ary tree, and the root of the tree is the source of the multicast.

We use a standard model for multicast with reliable communication channels each associated with an incoming (receive) buffer and an outgoing (send) buffer, and flow-control [2, 21]. These can be, e.g., TCP links, though this is not necessary. All incoming and outgoing buffers are of equal size, denoted B_0. The aggregate buffer size along the path from the root to a leaf is denoted $B_{max}(N) = \Theta(B_0 \cdot \log N)$. The application on the root node transmits packets, intermediate nodes forward them without any application-level buffering (which would otherwise be equivalent to having larger low-level buffers), and the leaf nodes consume the packets. Application threads at the root and at internal nodes stall to prevent overflow of outgoing buffers. The flow-control mechanism prevents overflow of incoming buffers.

3. ANALYTIC MODEL

In this section we model the system dynamics and illustrate oscillatory behavior. The key property captured in our model involves *disturbances* that nodes may experience. Disturbances prevent nodes from forwarding packets. This can happen for various reasons, e.g., when the application thread does not respond or when packets do not reach the node because of a link problem. By modeling disturbances, we capture dependencies among delays of messages that arrive close together. This differs from traditional models, which typically assume that message delays are *independently* sampled from some distribution (often exponential).

In order to model disturbances, we define two possible states of a node: *Good* and *Bad* (disturbed). The node states are described in Figure 1(a). Every node in the multicast tree alternates between these states. Nodes in the *Good* state forward packets from upstream nodes to downstream nodes (or consume them in case of leafs) as described in the multicast model, whereas nodes in the *Bad* state do not consume or forward packets. We assume that becoming disturbed is a memoryless process, which we capture by modeling the duration of time that a node remains in the Good state as exponentially distributed with expectation λ. Because data centers are typically homogeneous, we assume that the distributions at all nodes are identical and independent (iid). The time in the Bad state may have an arbitrary distribution, provided that the distribution is iid for all nodes and has a finite expectation, μ. We denote p_{Bad} and p_{Good} the probabilities that a node is in *Bad* and *Good* states, respectively ($p_{Bad} + p_{Good} = 1$). The process of node states transitions can be seen as regenerative process and in long term system run the probabilities are: $p_{Good} = \frac{\lambda}{\lambda+\mu}, p_{Bad} = \frac{\mu}{\lambda+\mu}$

The system throughput depends on the dynamics of all node states. To capture these dynamics, we introduce global system states. The global states transitions are described in Figure 1(b). When we start observing the system, we assume that there are *Good* nodes in the system and say that the system state is *Active*. When some node $\boldsymbol{u}$ becomes *Bad*, the system transitions to the *Blocking* state. When node $\boldsymbol{u}$ becomes *Good* again, the system moves back to the *Active* state. Note that at this point, other *Bad* nodes might exist in the system. The system transitions to *Blocking* state only when some new node changes its state from *Good* to *Bad* during the *Active* system state. This definition of system states allows us to divide the execution into time intervals, each spanning from the beginning of an occurrence of the *Active* state until the beginning of the successive *Active* state. We call these intervals *periods*.

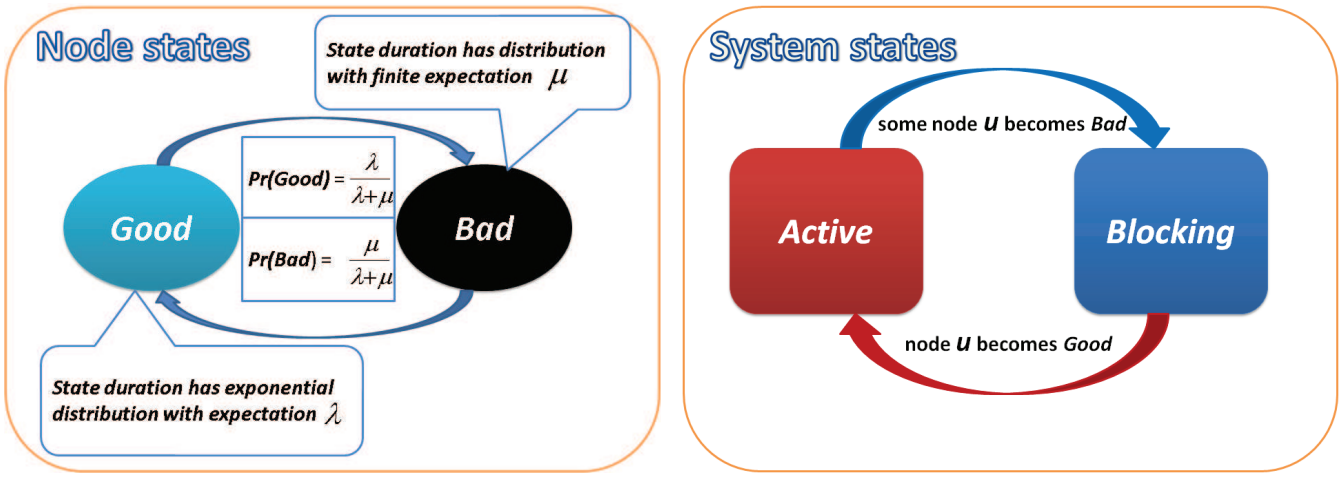
Bad nodes cause throughput oscillations at the root. In our model, oscillations arise as follows: Suppose we are in the Active state without *Bad* nodes, and some internal node $\boldsymbol{u}$ becomes *Bad* (the system transitions to the Blocking state). As a result, $\boldsymbol{u}$ stops forwarding packets. For a while, the root continues sending, so there are still incoming packets from the upstream link, which fill $\boldsymbol{u}$'s buffers. When $\boldsymbol{u}$'s incoming buffer fills up, the underlying flow control mechanism causes $\boldsymbol{u}$'s parent node to stop sending, which in turn causes its buffers to fill up. If $\boldsymbol{u}$'s disturbance persists, then eventually all the buffers on the path from the root to $\boldsymbol{u}$ become full, and the root's sending throughput drops to zero[1]. Thus, overall this state, the root can send at most the amount of data that fills the path to $\boldsymbol{u}$, i.e., B_{max}, and must then block. Only after the system returns to the Active state, the root may resume sending packets. We make the best case assumption that buffers become empty at this stage.

For each period k, let A_k (B_k) denote the number of bits transmitted by the root during the period in the Active (Blocking) state, respectively, and let t_{A_k} (t_{B_k}) denote the respective state's duration in seconds. The aggregate throughput of the root during m successive periods is

$$\textsc{Aggr}(m) \triangleq \frac{\sum_{k=1}^{m}(A_k + B_k)}{\sum_{k=1}^{m}(t_{A_k} + t_{B_k})}.$$

Our main result is a bound on aggregate throughput. For each period k, we can bound A_k by $\frac{T_{max}}{2} \cdot t_{A_k}$. Note that $\frac{T_{max}}{2}$ is the maximal achievable throughput for a binary overlay tree of nodes with a single NIC. On the other hand, B_k is bounded by B_{max}. To analyze the duration in each state, recall that every node's duration in the *Good* state is drawn iid from exponential distribution with expectation λ, and the duration of its *Bad* state is drawn iid from some distribution with a finite expectation μ. We define t_{Good} to be the probability that a node is in the *Good* state at an arbitrary time, and p_{Bad} to be $1-p_{Good}$. Intuitively, the time the system will remain in the *Active* state depends on the number of *Good* nodes, because it will become blocked when the first of these becomes *Bad*. This number is a random variable with expectation $N \cdot p_{Good}$. We bound t_{A_k}, we define some threshold δ and consider separately (1) the case that the number of *Bad* nodes in the *Active* state exceeds its

[1]This phenomenon of congestion moving upstream is known in queueing networks [4], though the behavior is different in our case, because all traffic through a node stops when some upstream node is blocked.



(a) *Life-cycle of a node in the multicast system. In Good state, nodes follow the protocol. When some node experiences disturbance it transits to Bad state. In Bad state, the node's application thread does not forward packets from incoming to outgoing buffers.*

(b) *Life-cycle of the entire system. When we start observing the system, we assume that all nodes are Good and the system state is Active. When some node* **u** *becomes Bad, the system transitions to the Blocking state. When* **u** *again becomes Good, the system transitions back to the Active state. Note that when system returns to the Active state other Bad nodes can still exist.*

Figure 1: State diagrams

expectation by a factor of $1+\delta$ or more, and (2) the case that it does not. We use this to bound AGGR(m). We define

$$\text{AGGR}_{norm}(N,\delta) \triangleq$$

$$\min_{1\le n\le N} \frac{\frac{\lambda}{n-n\cdot p_{Bad}\cdot(1+\delta)}+\lambda\cdot\left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right)^{N\cdot p_{Bad}}+\frac{2\cdot B_0+2\cdot B_{max}}{T_{max}}}{\frac{\lambda}{n-n\cdot p_{Bad}\cdot(1+\delta)}+\lambda\cdot\left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right)^{N\cdot p_{Bad}}+\mu}$$

in the full paper [5] and we prove the following normalized aggregate throughput bound for binary trees:

$$\forall\delta: 0<\delta<\frac{p_{Good}}{p_{Bad}}, \frac{\lim_{m\to\infty}\text{AGGR}(m)}{T_{max}/2} \le \text{AGGR}_{norm}(N,\delta). \quad (1)$$

We note that this bound is not tight due to several best-case assumptions, most notably the fact that buffers are empty in the *Active* state.

4. SIMULATION MODEL

We use simulations to validate and complement our analytic results. Recall that Equation 1 only provides a coarse upper bound on aggregate throughput. While the bounds can predict a potential problem, they do not capture the full severity of the problem. Our simulation model eliminates some of the best-case assumptions made in the analysis.

We construct a complete binary tree where each node has incoming and outgoing buffers according to the multicast model of Section 2. The simulation runs in time slots with a duration of $\frac{packet\ size}{T_{max}}$ s.t. a node can send at most one packet per slot. Links do not lose messages. Nodes alternate between *Good* and *Bad* states, as in the analytical model: the number of time slots that a node remains in a given state is generated as an iid random sample from an exponential distribution with an appropriate mean. These states are related only to the application thread – they determine whether a node can move packets from its incoming to outgoing buffers, or consume packets in case of a leaf.

As a best case assumption, we implement perfect flow control and load balancing among outgoing links. Each node transmits at most one packet per slot from its outgoing buffer, choosing the target child in a round-robin manner among those that have at least one free slot in their incoming buffer. A sent packet is inserted to the appropriate incoming buffer after $\frac{RTT}{2}$ time and is removed from the outgoing buffer $\frac{RTT}{2}$ time later.

By modeling perfect flow control and loss-free links, our simulation still provides an upper bound on attainable throughput. This is roughly equivalent to a scenario where TCP's slow start is disabled. Note however, that in contrast to the analytic model, we do not assume that all buffers become immediately empty upon return to an Active state, and hence provide a finer upper bound.

5. RESULTS

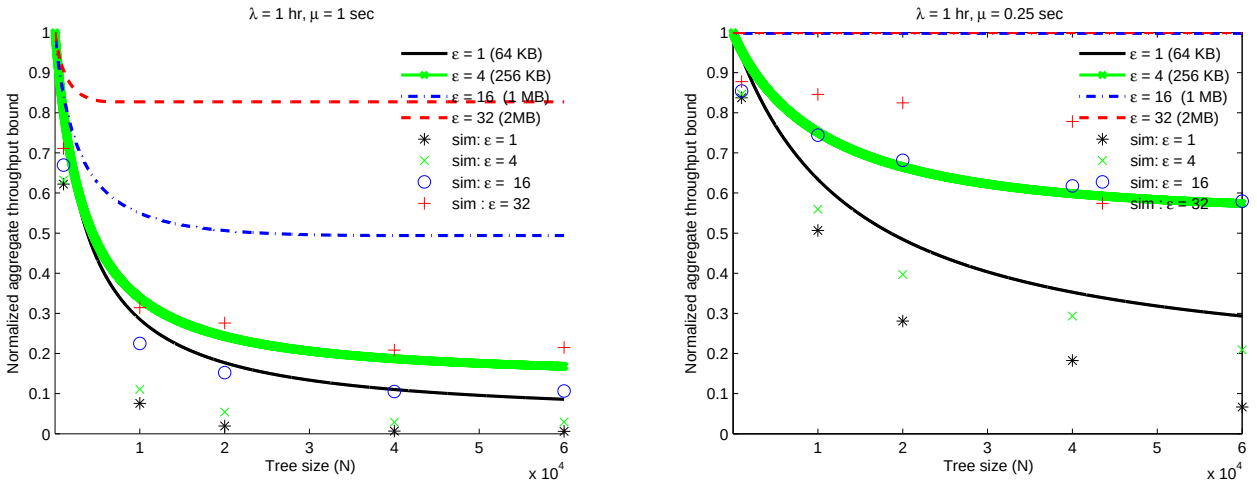
Now we use our analytical model and simulations to investigate multicast in a data center environment.

5.1 System parameters

Data center applications may easily run on tens to hundreds of nodes, so we investigate tree sizes between 10-100. Given that data center infrastructures are rapidly scaling out, we also experiment with $1K-60K$ node trees. We assume that nodes are connected by a high-throughput low-latency network with $T_{max}=1$Gbit/sec. According to [22], the default TCP buffer size of $B_{def}=64\ KB$ is good for such low latency networks ($RTT\le 1\ msec$). Unless explicitly mentioned, we use a scaling factor, ε, of 1, i.e., input and output buffers of size $B_0=\varepsilon\cdot B_{def}=64KB$. For simplicity, we assume that the multicast overlay is a complete binary tree, and thus $B_{max}=2\cdot(\log_2(N+1)-1)\cdot B_0$.

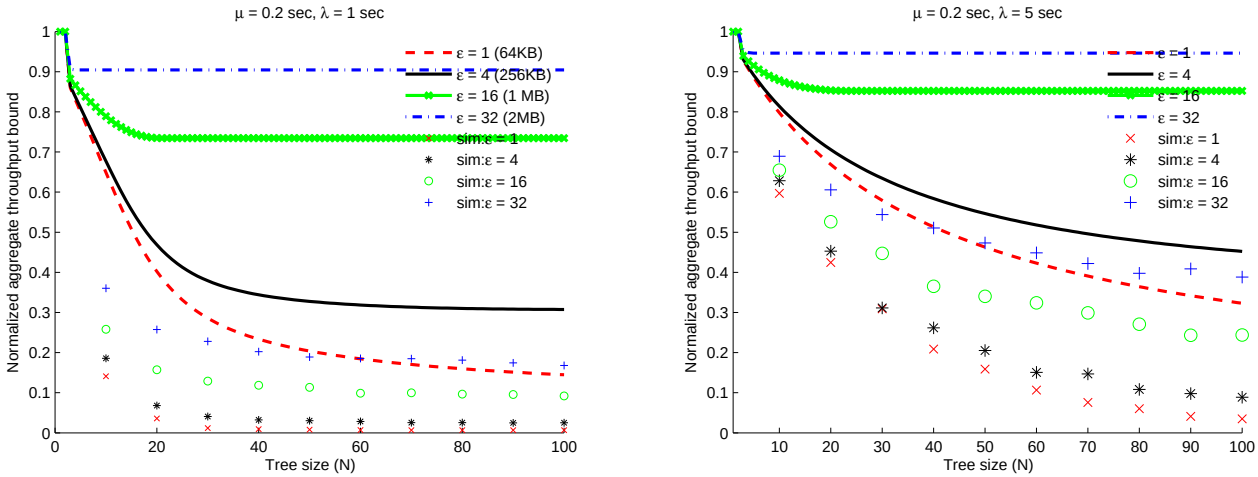
5.2 The effect of disruptions

As we all experience in our daily interaction with computers, and as many empirical studies have shown [19, 12, 20],



(a) (b)

Figure 2: Coarse-grained disturbances. Theoretical and simulated normalized aggregate throughput bounds for disturbances of $\mu = 1sec$ (2(a)) and $\mu = 0.25sec$ (2(b)) every $\lambda = 1hour$. The parameter ε is the scaling factor, so the node buffer size of $B_0 = \varepsilon \cdot 64KB$. The parameter δ used in the analytical bound is $0.01/p_{Bad}$. Each simulation data point in the graphs represents an average over five experiments, each lasting 1 minute. Repeated trials showed at most 3% error within the 95% confidence interval. The error bars are omitted for clarity.



(a) (b)

Figure 3: Fine-grained disturbances. Theoretical and simulation normalized aggregate throughput bound results for disturbances of $\mu = 0.2sec$ every $\lambda = 1sec$ (3(a)) and $\lambda = 5sec$ (3(b)) on average. The parameter ε is the scaling factor, so the node buffer size of $B_0 = \varepsilon \cdot 64KB$. The parameter δ used in the analytical bound is $0.3/p_{Bad}$. Each data point in the graph represents an average over five experiments, each lasting 1 minute. Repeated trials showed at most 3% error within the 95% confidence interval. The error bars are omitted for clarity.

computers are subject to disturbances that cause response delays once in a while. First, we consider coarse-grained events lasting $250 - 1000ms$ (μ in the analytical model) and occurring only once per hour per machine on average (λ in the analytical model). Figure 2 shows analytical and simulation results of our aggregate throughput bound for this case. Figure 2(a) considers one-second-long disturbances, and shows drastic throughput degradation – up to 90% for $60K$ trees. Figure 2(b) shows that disturbance duration is critical – hourly disruptions of $0.25sec$ (instead of $1sec$) degrade throughput by 70%.

Our analytic bound is not tight due to a number best-case assumptions. In particular, when buffers are large, our equation does not provide a meaningful bound. Simulations show that the systems continues to perform poorly even with large buffers. We believe the reason to be that, disturbances cause buffers to fill up and they often remain full in later periods. Our model makes the optimistic assumption that

all buffers are empty at moment the Blocking state begins, though in reality, this is often not the case. Note that if the buffers are very big, it may take a while for them to fill up and cause instability. Second, we consider fine-grained disruption events that last $200ms$, occurring once in a few seconds. The catalyst of such disturbances might be heavy load on nodes, causing scheduling delays of application threads (e.g., due to Java garbage collection pauses[2]). Such disturbances may also stem from heavy load on network switches. The latter has also been observed to cause the well-known Incast problem [9, 8, 17, 14] in data centers. According to [11], under high utilization, an average node in the data center has about ten concurrent flows more than 50% of the time, and at least 5% of the time it has more than 80 concurrent flows. Therefore, packet drops at the node's switch port once every few seconds are realistic. The resulting TCP time-out prevents packets flow through the node, which is equivalent to a disturbance lasting $200ms$ in our analytical model. Figure 4 shows that such disruptions can cause throughput degradation of up to 80% even for trees of 60 nodes. The simulations show that, in practice, the situation is even worse and, again, increasing the buffer sizes does not help as expected. Reducing the frequency of disruption events (Figure 3(a) vs. Figure 3(b)) can improve the system performance.

6. RELATED WORK

While the earliest reliable multicast protocols (notably RMTP [16]) ran directly on UDP, at present it is more common to employ TCP for link-level reliability and flow control [21, 2, 15, 13]. Previous work [21] has shown that a back-pressure-based model like ours and [2] is more practical and achieves better performance than end-to-end congestion control, where NACKs are sent directly from all nodes to the sender, as in [7]. Performance is typically evaluated by running the real system (in modest configurations under controlled stress), and then simulating it (to explore larger ones). Such testing can easily miss the problems our work highlights. For example, although the authors of [2] used the same multicast model we employ here, their tests (in configurations with 30-60 nodes) did not evaluate the conditions that trigger throughput collapse in Section 5.2.

Simulations, on other hand typically model link delays and message losses as iid samples from some probabilities [21, 2, 7]. But they rarely account for the possibility of scheduling disturbances in nodes. For example in [2], whose multicast model is similar to ours, when there is no message loss in a system of size 126 the throughput is very close to the (high) throughput of a single link. In contrast we find that even without message loss, if the nodes in a 100-node overlay tree experience disruptions, throughput plunges by 80% relative to the peak. Moreover, our finding confirms the reports of cloud infrastructure owners, who have struggled to deploy trees of this sort [6].

7. CONCLUSION

Our work responds to a widely observed (but poorly explained) problem: data centers replicate information using overlay trees constructed from reliable links, but sometimes perform poorly even when everything seems to be operating smoothly. We show that this is to be expected: a reliable multicast constructed in this manner is overwhelmingly likely to experience throughput collapse and the problem grows worse as a system scales up.

We should emphasize that our work only applies to this particular style of multicast. One can also replicate data using hardware multicast, gossip, or other mechanisms. Thus, our findings are negative for today's most common replication solutions, but do not in preclude the emergence of better solutions tomorrow. Indeed, we see our work as a contribution in two respects: First, it explains a serious and widely noted problem. Second, it suggests a style of analysis that might be valuable in the future, as other solutions are proposed.

[2] `http://java.sun.com/developer/technicalArticles/Programming/turbo/`

Acknowledgments

We thank Ben Black, who briefed us on the issue of multicast oscillatory behavior observed in use at Amazon. We also thank Isaac Keslassy, Eyal Zohar and Boris Oklander for their helpful comments.

This work was supported, in part, by the Technion Funds for Security Research, by the Israeli Ministry of Industry, Trade and Labor Magnet Consortium and grants from the NSF and AFRL.

8. REFERENCES

[1] A. Greenberg. Networking the Cloud, 2009. Keynote address at ICDCS 2009, Montreal, Canada.

[2] F. Baccelli, A. Chaintreau, Z. Liu, and A. Riabov. The one-to-many TCP overlay: a scalable and reliable multicast architecture. In *INFOCOM*, pages 1629–1640. IEEE, 2005.

[3] F. Baccelli, A. Chaintreau, Z. Liu, A. Riabov, and S. Sahu. Scalability of reliable group communication using overlays. In *INFOCOM*, 2004.

[4] S. Balsamo, V. de Nitto Persone, and R. Onvural. *Analysis of Queueing Networks with Blocking.* Springer, 2000.

[5] D. Basin, K. Birman, I. Keidar, and Y. Vigfusson. Sources of Instability in Data Center Multicast. Technical Report CCIT 768, Department of Electrical Engineering, Technion, July 2010.

[6] K. Birman, G. Chockler, and R. van Renesse. Toward a cloud computing research agenda. *SIGACT News*, 40(2):68–80, 2009.

[7] A. Chaintreau, F. Baccelli, and C. Diot. Impact of TCP-like congestion control on the throughout of multicast groups. *IEEE/ACM Trans. Netw.*, 10(4):500–512, 2002.

[8] Y. Chen, R. Griffith, J. Liu, R. H. Katz, and A. D. Joseph. Understanding TCP incast throughput collapse in datacenter networks. In *WREN '09: Proceedings of the 1st ACM workshop on Research on enterprise networking*, pages 73–82, New York, NY, USA, 2009. ACM.

[9] D. Nagleand, D. Serenyi and A. Matthews. The Panasas ActiveScale Storage Cluster: Delivering Scalable High Bandwidth Storage. In *SC '04: Proceedings of the 2004 ACM/IEEE conference on Supercomputing*, page 53, Washington, DC, USA, 2004. IEEE Computer Society.

[10] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel. The cost of a cloud: research problems in data center networks. *SIGCOMM Comput. Commun. Rev.*, 39(1):68–73, 2009.

[11] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta. Vl2: a scalable and flexible data center network. *SIGCOMM Comput. Commun. Rev.*, 39(4):51–62, 2009.

[12] J. Hamilton. Internet-Scale Service Efficiency. In *LADIS '08*.

[13] J. He and A. Chaintreau and C.Diot. A performance evaluation of scalable live video streaming with nano data centers. *Computer Networks*, 53(2):153–167, 2009.

[14] E. Krevat, V. Vasudevan, A. Phanishayee, D. G. Andersen, G. R. Ganger, G. A. Gibson, and S. Seshan. On application-level approaches to avoiding TCP throughput collapse in cluster-based storage systems. In G. A. Gibson, editor, *PDSW*, pages 1–4. ACM Press, 2007.

[15] G.-I. Kwon and J. W. Byers. Roma: Reliable overlay multicast with loosely coupled tcp connections. In *INFOCOM*, 2004.

[16] S. Paul, K. K. Sabnani, J. C.-H. Lin, and S. Bhattacharyya. Reliable Multicast Transport Protocol (RMTP). *IEEE Journal on Selected Areas in Communications*, 15(3):407–421, 1997.

[17] A. Phanishayee, E. Krevat, V. Vasudevan, D. G. Andersen, G. R. Ganger, G. A. Gibson, and S. Seshan. Measurement and Analysis of TCP Throughput Collapse in Cluster-based Storage Systems. In *FAST '08*, San Jose, CA, Feb. 2008.

[18] S. Ratnasamy, A. Ermolinskiy, and S. Shenker. Revisiting IP multicast. In *SIGCOMM '06*, pages 15–26, New York, NY, USA, 2006. ACM.

[19] M. Theimer. Some Lessons Learned from Running Amazon Web Services. In *LADIS '09*.

[20] L. A. Torrey, J. Coleman, and B. P. Miller. A comparison of interactivity in the Linux 2.6 scheduler and an MLFQ scheduler. *Softw. Pract. Exper.*, 37(4):347–364, 2007.

[21] G. Urvoy-Keller and E. W. Biersack. A Multicast Congestion Control Model for Overlay Networks and its Performance. In *NGC*, October 2002.

[22] E. Weigle and W. C. Feng. A comparison of TCP automatic tuning techniques for distributed computing. *International Symposium on High-Performance Distributed Computing*, 0:265, 2002.