More is Less: Reducing Latency via Redundancy

Ashish Vulimiri
UIUC
vulimir1@illinois.edu

Oliver Michel
University of Vienna
oliver.michel@univie.ac.at

P. Brighten Godfrey
UIUC
pbg@illinois.edu

Scott Shenker
UC Berkeley and ICSI
shenker@icsi.berkeley.edu

ABSTRACT

Low latency is critical for interactive networked applications. But while we know how to scale systems to increase capacity, reducing latency — especially the tail of the latency distribution — can be much more difficult.

We argue that the use of redundancy in the context of the wide-area Internet is an effective way to convert a small amount of extra capacity into reduced latency. By initiating redundant operations across diverse resources and using the first result which completes, redundancy improves a system's latency even under exceptional conditions. We demonstrate that redundancy can significantly reduce latency for small but critical tasks, and argue that it is an effective general-purpose strategy even on devices like cell phones where bandwidth is relatively constrained.

Categories and Subject Descriptors

C.2.0 [**Computer-Communication Networks**]: General

General Terms

Performance, Reliability

1. INTRODUCTION

Low latency is important for humans. Even slightly higher web page load times can significantly reduce visits from users and revenue, as demonstrated by several sites [21]. For example, injecting just 400 milliseconds of artificial delay into Google search results caused the delayed users to perform 0.74% fewer searches after 4-6 weeks [7]. A 500 millisecond delay in the Bing search engine reduced revenue per user by 1.2%, or 4.3% with a 2-second delay [21]. Human-computer interaction studies similarly show that people react to small differences in the delay of operations (see [12] and references therein).

Achieving consistent low latency is challenging. Modern applications are highly distributed, and likely to get more so as cloud computing separates users from their data and computation. Moreover, application-level operations often require tens or hundreds of tasks to complete — due to many objects comprising a single web page [19], or aggregation of many back-end queries to produce a front-end result [1,10]. This means individual tasks may have latency budgets on the order of a few milliseconds or tens of milliseconds, and *the tail* of the latency distribution is critical. Thus, latency is a difficult challenge for networked systems: How do we make the other side of the world feel like it is *right here*, even under exceptional conditions?

One powerful technique to reduce latency is *redundancy*: Initiate an operation multiple times, using as diverse resources as possible, and use the first result which completes. For example, a host may query multiple DNS servers in parallel to resolve a name. The overall latency is the minimum of the delays across each instance, thus potentially reducing both the mean and the tail of the latency distribution. The power of this technique is that *it reduces latency precisely under the most challenging conditions:* when delays or failures are unpredictable.

Redundancy has been employed in several past networked systems: notably, as a way to deal with failures in DTNs [15], and in a multi-homed web proxy overlay [3]. But beyond these specific research projects, redundancy is typically eschewed across the Internet. We argue this is a missed opportunity.

The contribution of this paper is to argue for redundancy as a general technique for the wide-area Internet. The combination of interactive applications, high latency, and variability of latency make redundancy well suited to this environment. Even in a well-provisioned

Hotnets '12, October 29–30, 2012, Seattle, WA, USA.
Copyright 2012 ACM 978-1-4503-1776-4/10/12 ...$10.00.

network where individual operations usually work, some amount of uncertainty is pervasive and the demand for consistent low latency outweighs the need to save bandwidth which is today comparatively cheap.

To support this argument, in §3 we examine the cost of redundancy. Since latency-bound tasks are likely to be small, the overall overhead is small when workloads are heavy-tailed; we show that flow-size distribution measurements confirm this. We next set a benchmark for when replication is useful from the perspective of impact on network bandwidth, showing that replication may be cost-effective even in extremely conservative (cell phone) scenarios as long as we can save more than 10 milliseconds per kilobyte of added traffic. In §4, we show the benefits of replication can be orders of magnitude larger than this threshold in a number of common application scenarios. For example, querying multiple DNS servers can reduce the fraction of responses later than 500 ms by 6.5×, while the fraction later than 1.5 sec is reduced by 50×. We also discuss applications to multipath routing, TCP connection establishment, and quality of service.

In summary, as system designers we typically build scalable systems by avoiding unnecessary work. But we argue that in the wide-area Internet, extra work is a useful and elegant way to achieve robustness to unexpected conditions and consistently low latency.

2. RELATED WORK

Replication is used pervasively to improve reliability, and in many systems to reduce latency. Distributed job execution frameworks, for example, have used task replication to improve response time, both preemptively [2, 11] and to mitigate the impact of stragglers [23].

Within networking, replication has been explored to reduce latency in several specialized settings, including replicating DHT queries to multiple servers [16] and replicating transmissions (via erasure coding) to reduce delivery time and loss probability in delay-tolerant networks [15, 20]. Replication has also been suggested as a way of providing QoS prioritization and improving latency and loss performance in networks capable of redundancy elimination [14].

Perhaps closest to our work, Andersen et al. [3]'s MONET system proxies web traffic through an overlay network formed out of multi-homed proxy servers. While the primary focus of [3] is on adapting quickly to changes in path performance, they replicate two specific subsets of their traffic: connection establishment requests to multiple servers are sent in parallel (while the first one to respond is used), and DNS queries are replicated to the local DNS server on each of the multi-homed proxy server's interfaces. We show that replication can be useful in both these contexts even in the absence of path diversity: a significant performance benefit can be obtained by sending multiple copies of TCP SYNs to the *same* server on the *same* path, and by replicating DNS queries to multiple public servers over the *same* access link.

Most importantly, unlike all of the above work, our point is that replication is a general technique that can be applied in a variety of common wide-area Internet applications. We argue this point by studying the overhead associated with replicating small flows, the necessary benefit for end-users, and several use cases.

3. MORE IS LESS

In this section we discuss the efficacy of the general approach of reducing latency via redundancy.[1] Later (§4) we discuss specific applications.

3.1 Avoiding uncertainty

The power of redundancy is to reduce uncertainty without having to anticipate the cause of that uncertainty. In general, assume we have multiple options available to obtain a service — such as multiple servers, multiple paths, or even multiple moments in time at which to request service. If we could predict which of these options will perform correctly with lowest latency, then we would simply execute that option.

But perfect prediction of exceptional conditions, especially in large networked systems, is somewhere between difficult and impossible. Momentarily high utilization on a server, delay due to virtualization [22], a slow disk, absence of an object in a web or DNS cache, network congestion, an attack like a fraudulent DNS reply, and other abnormalities all impact the mean and especially the tail of the latency distribution. Operations that fail can be retried, of course, but this adds greater delay especially in the wide-area. And designing systems which do not encounter such abnormalities is difficult and expensive.

If, however, we execute multiple options in parallel and use the fastest correct result, then we can avoid these exceptional conditions unless they occur to all options simultaneously.

There are two key difficulties. First, we may not have multiple truly independent options, so that exceptional conditions are correlated across options. We later give examples of wide-area applications that have sufficient diversity across options (§4).

Second, redundancy involves more work and expense. In §3.2, we argue that the overall increase in utilization may be small, since latency-sensitive tasks are often a small fraction of the total network load. In §3.3, we consider the perspective of an individual user, arguing that saving even a few milliseconds per extra KB of traffic is worthwhile.

[1] This discussion may repeat arguments in the introduction. But that would be rather apropos, wouldn't it?

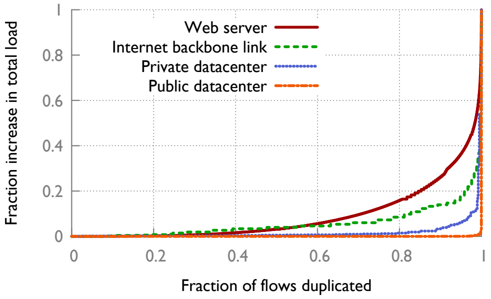

Figure 1: Overhead of duplicating traffic.

3.2 Overhead can be low

In general, operations that require consistent low latency are likely to be small in the work they expend per operation. Consider, for example, a flow across the wide-area Internet: if the flow is significantly larger than the network's delay-bandwidth product, the total time it needs to complete will be dominated by its throughput. When downloading a gigabyte movie which may have minutes of buffer, a few hundred milliseconds delay or loss of a few packets will likely go unnoticed. Alizadeh et al. [1] have also noted that in a number of data center applications, latency-critical jobs are small. In such systems, the flows that are the most likely to benefit from replication are also those that are the least expensive to replicate.

Furthermore, if latency-critical operations are small, then they will comprise a small fraction of the network's total work if the distribution of sizes is heavy-tailed, which is a pervasive property in the Internet. Figure 1 shows the percentage increase in total network load that would result if the smallest $x\%$ of all flows were duplicated in four different settings: flows in a public [13] and a private [6] data center, web service requests crossing a university wide-area uplink [5], and an Internet backbone link [18]. In all cases it would be possible to duplicate at least the 33% smallest flows while increasing the total load by only 2%.

Several caveats may limit or expand the scenarios in which replication is useful. (1) In contrast to the general rule above, some latency-sensitive tasks are large, such as real-time video streams. Replication may still be useful if these streams have low rate (§4.3) or are critical. (2) If overhead is high, a replicated request could be marked as lower priority, so it can be dropped if it interferes with other work, similar to [16]. Alternately, one could replicate only the most important operations; for example, a cloud service provider could charge tenants a higher rate for consistent low latency. (3) Even if overhead is high, replication may be acceptable when the system is under-utilized, as the wide-area Internet typically is. Thus, replication can reclaim value from otherwise unused resources.

3.3 End-user cost-benefit analysis

We can estimate when replication is useful for an end-user via the value of time and the cost of increased utilization. The following calculations analyze only the cost in terms of network bandwidth to users and providers, and the benefit to a user.

For replication to make sense to an end-user, we need $\ell v \geq b$, where ℓ is the average latency savings in milliseconds for each KB of added traffic, v is the dollar value of one millisecond of latency reduction, and b is the cost of added traffic per KB.

The value of time v is the most difficult to calculate. It may be highly application-specific, and may depend on mean or tail latency in ways best quantified by a human user study of quality of experience. But to obtain a first approximation, we assume the value of time is simply the US average earnings of \$23.50 per hour in June 2012 [17], which implies $v \approx 6.53 \cdot 10^{-6}$ \$/ms.

To estimate the cost per kilobyte b, we turn to advertised rates for end-user and cloud service bandwidth. For cell plans, we assume a user who has paid for basic connectivity already, and calculate the cost of bandwidth from overage charges. For example, AT&T's smallest wireless data plan incurs \$20 per 300 MB.

We can now solve $\ell \geq b/v$ to obtain the break-even point, in terms of the necessary latency savings per kilobyte of additional traffic:[2]

Connectivity plan	Cost b	Break-even benefit ℓ
	(\$/KB)	(ms/KB)
AT&T, small cell	$651.04 \cdot 10^{-7}$	9.970
AT&T, large cell	$95.37 \cdot 10^{-7}$	1.460
T-Mobile Austria	$9.67 \cdot 10^{-7}$	0.148
AT&T DSL	$1.91 \cdot 10^{-7}$	0.029
EC2 and Azure clouds	$1.20 \cdot 10^{-7}$	0.018
MaxCDN	$0.40 \cdot 10^{-7}$	0.006

Note that in a connection between an end-user and a service in the cloud, the latter's bandwidth costs are comparatively small. The table shows replication may be cost-effective even in the most conservative connectivity plan as long as we can save more than 10 milliseconds (in the mean or tail, depending on the goal) for each kilobyte of added traffic. We will see that the benefit can be at least an order of magnitude greater than this quantity.

4. EXAMPLES

4.1 Connection establishment

We start with a simple motivating example, demonstrating why replication should be useful even when the available choices are limited: we consider what happens when multiple copies of a packet are sent on the same path. It is obvious that this should help if all packet

[2] These prices exclude the cost of energy, taxes, and fees. Retrieved from the providers' websites on 5 Oct 2012.

losses on the path are independent. In this case, sending two back-to-back copies of a packet would reduce the probability of it being lost from p to p^2. In practice, of course, back-to-back packet transmissions are likely to observe a correlated loss pattern. But Chan et al. [8] measured a significant reduction in loss probability despite this correlation. Sending back-to-back packet pairs between PlanetLab hosts, they found that the average probability of individual packet loss was ≈ 0.0048, and the probability of *both* packets in a back-to-back pair being dropped was only ≈ 0.0007 – much larger than the $\sim 10^{-6}$ that would be expected if the losses were independent, but still 7× lower than the individual packet loss rate.[3]

As a concrete example, we quantify the improvement that this loss rate reduction would effect on the time required to complete a TCP handshake. The three packets in the handshake are, by the metrics discussed in §3, ideal candidates for replication: they make up an insignificant fraction of the total traffic in the network, and there is a high penalty associated with their being lost (Linux and Windows use a 3 second initial timeout for SYN packets; OS X uses 1 second [9]). We use the loss probability statistics discussed above to estimate the expected latency savings on each handshake.

We consider an idealized network model. Whenever a packet is sent on the network, we assume it is delivered successfully after $(RTT/2)$ seconds with probability $1 - p$, and lost with probability p. Packet deliveries are assumed to be independent of each other. p is 0.0048 when sending one copy of each packet, and 0.0007 when sending two copies of each packet. We also assume TCP behavior as in the Linux kernel: an initial timeout of 3 seconds for SYN and SYN-ACK packets and of $3 \times RTT$ for ACK packets, and exponential backoff on packet loss [9].

With this model, it can be shown that duplicating all three packets in the handshake would reduce its expected completion time by approximately $(3 + 3 + 3 \times RTT) \times (4.8 - 0.7)$ ms, which is at least 25 ms. If we assume each packet is 50 bytes long, this implies a savings of around 170 ms/KB, which is more than an order of magnitude larger than the break-even latency savings identified for the most expensive plan in §3.3. The benefit increases with RTT, and is even higher in the tail. Duplication would improve the 99.9th percentile handshake completion time, for instance, by at least 880 ms, for a latency savings of around 6000 ms/KB.

4.2 DNS

An ideal candidate for replication is a service that involves small operations and which is replicated at multiple locations, thus providing diversity across network paths and servers, so that replicated operations are quite independent.

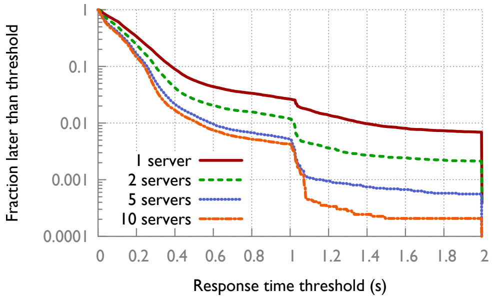

Figure 2: DNS response time distribution.

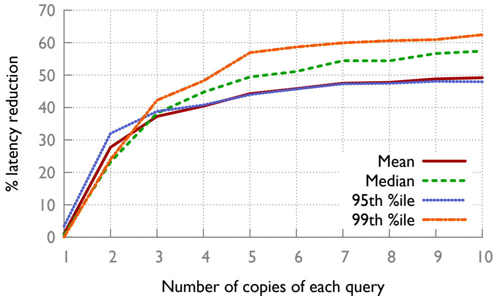

Figure 3: Reduction in DNS response time, averaged across 15 PlanetLab servers.

We began with a list of 10 DNS servers[4] and Alexa.com's list of the top 1 million website names. At each of 15 PlanetLab nodes across the continental US, we ran a two-stage experiment: (1) Rank all 10 DNS servers in terms of mean response time, by repeatedly querying a random name at a random server. Note that this ranking is specific to each PlanetLab server. (2) Repeatedly pick a random name and perform a random one of 20 possible trials — either querying one of the ten individual DNS servers, or querying anywhere from 1 to 10 of the best servers in parallel (e.g. if sending 3 copies of the query, we send them to the top 3 DNS servers in the ranked list). In each of the two stages, we performed one trial every 5 seconds. We ran each stage for about a week at each of the 15 nodes. Any query which took more than 2 seconds was treated as lost, and counted as 2 sec when calculating mean response time.

Figure 2 shows the distribution of query response times across all the PlanetLab nodes. The improvement is substantial, especially in the tail: Querying 10 DNS servers, the fraction of queries later than 500 ms is reduced by 6.5×, and the fraction later than 1.5

[3] It might be possible to do even better by spacing the transmissions of the two packets in the pair a few milliseconds apart to reduce the correlation.

[4] The default local DNS server, plus public servers from Level3, Google, Comodo, OpenDNS, DNS Advantage, Norton DNS, ScrubIT, OpenNIC, and SmartViper.

sec is reduced by 50×. Averaging over all PlanetLab nodes, Figure 3 shows the average percent reduction in response times compared to the best fixed DNS server identified in stage 1. We obtain a substantial reduction with just 2 DNS servers in all metrics, improving to 50-62% reduction with 10 servers. Finally, we compared performance to the best single server *in retrospect*, i.e., the server with minimum mean response time for the queries to individual servers in Stage 2 of the experiment, since the best server may change over time. Even compared with this stringent baseline, we found a result similar to Fig. 3, with a reduction of 44-57% in the metrics when querying 10 DNS servers.

Is this improvement worthwhile? Sending 10 DNS queries instead of 1 costs $<$ 4500 extra bytes for a latency savings of 0.1 sec in the mean or 0.9 sec in the 99th percentile. This is roughly 20× better than the most pessimistic break-even benefit in §3.3, meaning the bandwidth cost is negligible. We also note that querying multiple servers increases caching, a side-benefit which would be interesting to quantify.

We note that prefetching — that is, preemptively initiating DNS lookups for all links on the current web page — makes a similar tradeoff of increasing load to reduce latency, and its use is widespread. (Note, however, that redundancy is complementary to prefetching, since some names in a page will not have been present on the previous page.)

4.3 Multipath routing

We now consider replication of packets in a large-scale multipath routing setting. This setting may be available in a future Internet environment, but also mimics fairly closely (in topology and data rate) the Skype overlay network [4], where consistent low latency is beneficial.

We conducted experiments on three different overlay topologies, each consisting of a source and a destination node connected via 8 intermediate nodes, yielding 8 distinct end-to-end paths. We used two topologies spanning the US and one with the source and half the intermediate nodes in Europe and the destination and the remaining intermediate nodes in the US. Most of these nodes were on PlanetLab, but in one topology we used exclusively reserved ProtoGENI nodes for the source and destination because the PlanetLab nodes we used at first with were too heavily loaded (see further discussion below). P2P applications such as Skype use similar overlay topologies [4], relaying traffic through intermediate nodes to circumvent NAT and firewall issues.

Over the course of roughly 48 hours, we sent UDP data packets at rates of $r = 32$kbit/s and $r = 56$kbit/s over the $m \in \{1, 2, 3\}$ best paths at that point in time. The combinations of r and m were randomly chosen every minute. We used a simple-moving-average (SMA)[5] with a window-size of 5 to rank the paths.

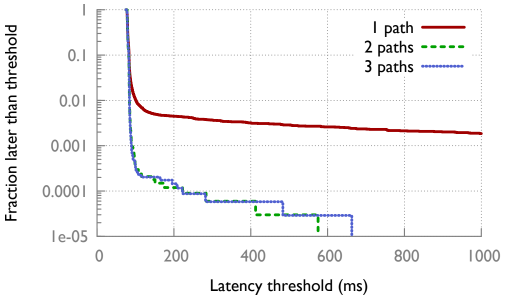

Figure 4: Multipath routing: Latency distribution in representative intra-US topology at $r = 56$kbit/s.

We observed a significant performance improvement when using replication, despite the variation in factors such as the topology, the data rate, and the shape of the latency distribution on each path. However, we note that in one prior measurement (results not reported here) replication significantly *degraded* system performance. We believe this was due to unusually heavy load on the sending PlanetLab node: when we switched it out for a lightly loaded ProtoGENI node we again observed the same level of performance as in the other experiments. The latter results are reported here, but this underscores the fact that a larger scale study is necessary to characterize the conditions under which replication helps.

Fig. 5 and 4 show that most or all of the improvement we observe comes from using two paths instead of one. While the improvement in the mean latency is relatively small, replication significantly improves the latency tail and reduces the packet loss probability. Compared to a single path we see better latencies starting at the 95th percentile, and with just one additional path the 99th percentile latency falls by about 60%.

The two data sending rates we tested, which are realistic for real-time audio traffic, yielded more or less identical behavior, but we expect that testing a larger range of rates will reveal that replication ceases to help when the data rate is sufficiently high. We leave an investigation of this question to future work.

We note that the experiments we have conducted so far are too small-scale to be representative – a much more thorough evaluation would be necessary to properly quantify the performance benefits that replication can yield. However, our results suggest that there are reasonable scenarios in which replication can help.

[5] In other experiments, not discussed here, we found that SMA performed better than other common adaptive estimators, including the exponentially-weighted moving average and several standard learning-theoretic algorithms.

Topology	mean latency			99% latency			99.9% latency		
	1 path	2 paths	3 paths	1 path	2 paths	3 paths	1 path	2 paths	3 paths
Intra-U.S. 1	103.5ms	7.28%	6.30%	292.1ms	61.61%	59.62%	loss	1846.8ms	1997.0ms
Intra-U.S. 2	86.0ms	6.63%	6.41%	101.9ms	17.19%	17.19%	1888.5ms	95.04%	95.22%
Trans-Atlantic	142.1ms	4.02%	4.19%	233.7ms	37.26%	37.27%	594.9ms	69.52%	71.81%

Figure 5: Multipath routing: Measured RTTs in 56kbit/s experiment for 1 path and *percent reduction* when replicating over 2 or 3 paths (packets later than 4.5 seconds were considered lost).

4.4 Quality of service

Low latency is an important aspect of quality of service (QoS) on the Internet. Traditional approaches to providing QoS (e.g. IntServ) involve relatively complex signaling and packet scheduling mechanisms, lack a proven pricing model, and have not seen widespread deployment. Even if they were deployed, these mechanisms deal with only one cause of high latency (congestion) to the exclusion of other unpredictable events (router failure, buggy QoS implementations, physical layer corruption, etc.).

Replication may be an effective means to obtain some of the benefits of QoS. If multipath routing is available—with multiple network providers, tunnels, or more advanced mechanisms in future Internet architectures—then latency-sensitive traffic can be replicated along these paths. Compared with traditional QoS, this scheme (1) eases deployment, by requiring no data plane QoS support and reusing existing pricing models based on bandwidth volume; (2) can be flexibly applied to a subset of traffic so only a small additional load is introduced where it is most effective; (3) deals with any unpredictable events, such as router failures, as long as they are not correlated across the independent paths.

5. RESEARCH DIRECTIONS

Using redundancy as a common technique in the Internet raises several interesting research directions.

Automating redundancy. Our analysis suggests that redundancy is worthwhile even with relatively small improvements in delay, and thus, it may be worthwhile to automate. One simple way would be, for example, to replicate the first k packets of each TCP flow, for some small k. Beyond this, can we automate redundancy to reduce latency in wide-area application-level services?

Path selection. Instead of choosing paths (or servers) based on their mean performance, it may be beneficial to pick options that are as independent as possible. Choosing a set of k paths whose latency-tails are as "maximally independent" as possible is an interesting systems and algorithmic problem.

Selfishness. Suppose users choose how much to replicate while selfishly minimizing their latency. We have constructed examples showing that the Nash equilibrium of such a process in a queueing system can be worse than without replication. But can performance seriously degrade? In practical systems, are end-host costs (e.g., energy or uplink constraints) sufficient to prevent poor outcomes?

Security. Beyond latency, there are broader architectural reasons for multiplicity. Just as redundancy makes it harder for nature to cause a problem, so it is harder for attackers to cause a problem. By replicating DNS queries, for example, one may detect malicious or incorrect responses. Can we design an Internet with *pervasive* redundancy, improving both latency and security?

6. ACKNOWLEDGEMENTS

We gratefully acknowledge the support of NSF grants CNS 10-50146 and CNS 11-49895.

7. REFERENCES

[1] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan. Data center TCP (DCTCP). In *SIGCOMM*, 2010.

[2] G. Ananthanarayanan, A. Ghodsi, S. Shenker, and I. Stoica. Why let resources idle? Aggressive cloning of jobs with Dolly. In *USENIX HotCloud*, 2012.

[3] D. G. Andersen, H. Balakrishnan, M. F. Kaashoek, and R. N. Rao. Improving web availability for clients with MONET. In *USENIX NSDI*, pages 115–128, Berkeley, CA, USA, 2005. USENIX Association.

[4] S. A. Baset and H. G. Schulzrinne. An analysis of the Skype peer-to-peer internet telephony protocol. In *IEEE INFOCOM*, pages 1 –11, april 2006.

[5] N. Basher, A. Mahanti, A. Mahanti, C. Williamson, and M. Arlitt. A comparative analysis of web and peer-to-peer traffic. In *WWW*, WWW '08, pages 287–296, New York, NY, USA, 2008. ACM.

[6] T. Benson, A. Akella, and D. A. Maltz. Network traffic characteristics of data centers in the wild. In *IMC*, pages 267–280, New York, NY, USA, 2010. ACM.

[7] J. Brutlag. Speed matters for Google web search, June 2009. http://services.google.com/fh/files/blogs/google_delayexp.pdf.

[8] E. W. Chan, X. Luo, W. Li, W. W. Fok, and R. K. Chang. Measurement of loss pairs in network paths. In *IMC*, pages 88–101, New York, NY, USA, 2010. ACM.

[9] J. Chu. Tuning TCP parameters for the 21st century. http://www.ietf.org/proceedings/75/slides/tcpm-1.pdf, July 2009.

[10] P. Dixon. Shopzilla site redesign – we get what we measure, June 2009. http://www.slideshare.net/shopzilla/shopzillas-you-get-what-you-measure-velocity-2009.

[11] C. C. Foster and E. M. Riseman. Percolation of code to enhance parallel dispatching and execution. *IEEE Trans. Comput.*, 21(12):1411–1415, Dec. 1972.

[12] W. Gray and D. Boehm-Davis. Milliseconds matter: An introduction to microstrategies and to their use in describing and predicting interactive behavior. *Journal of Experimental Psychology: Applied*, 6(4):322, 2000.

[13] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta. VL2: a scalable and flexible data center network. In *ACM SIGCOMM*, pages 51–62, New York, NY, USA, 2009. ACM.

[14] D. Han, A. Anand, A. Akella, and S. Seshan. RPT: re-architecting loss protection for content-aware networks. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, NSDI'12, pages 6–6, Berkeley, CA, USA, 2012. USENIX Association.

[15] S. Jain, M. Demmer, R. Patra, and K. Fall. Using redundancy to cope with failures in a delay tolerant network. In *ACM SIGCOMM*, 2005.

[16] J. Li, J. Stribling, R. Morris, and M. Kaashoek. Bandwidth-efficient management of DHT routing tables. In *USENIX NSDI*, 2005.

[17] U. D. of Labor. Economy at a glance. http://www.bls.gov/eag/eag.us.htm.

[18] F. Qian, A. Gerber, Z. M. Mao, S. Sen, O. Spatscheck, and W. Willinger. TCP revisited: a fresh look at TCP in the wild. In *IMC*, pages 76–89, New York, NY, USA, 2009. ACM.

[19] S. Ramachandran. Web metrics: Size and number of resources, May 2010. https://developers.google.com/speed/articles/web-metrics.

[20] E. Soljanin. Reducing delay with coding in (mobile) multi-agent information transfer. In *Communication, Control, and Computing (Allerton), 2010 48th Annual Allerton Conference on*, pages 1428–1433. IEEE, 2010.

[21] S. Souders. Velocity and the bottom line. http://radar.oreilly.com/2009/07/velocity-making-your-site-fast.html.

[22] J. Whiteaker, F. Schneider, and R. Teixeira. Explaining packet delays under virtualization. *ACM SIGCOMM Computer Communication Review*, 41(1):38–44, January 2011.

[23] M. Zaharia, A. Konwinski, A. D. Joseph, R. Katz, and I. Stoica. Improving MapReduce performance in heterogeneous environments. In *USENIX OSDI*, pages 29–42, Berkeley, CA, USA, 2008. USENIX Association.