Scalable Coordination of a Tightly-Coupled Service in the Wide Area

Kevin C. Webb[†], Bhanu C. Vattikonda, Kenneth Yocum, and Alex C. Snoeren

UC San Diego and [†]Swarthmore College

Abstract

Today's large-scale services generally exploit loosely-coupled architectures that restrict functionality requiring tight cooperation (e.g., leader election, synchronization, and reconfiguration) to a small subset of nodes. In contrast, this work presents a way to scalably deploy tightly-coupled distributed systems that require significant coordination among a large number of nodes in the wide area. Our design relies on a new group membership abstraction, *circuit breaker*, to preserve efficient pairwise communication despite transient link failures.

We characterize our abstraction in the context of a distributed rate limiting (DRL) service's deployment on a global testbed infrastructure. Unlike most distributed services, DRL can safely operate in separate partitions simultaneously, but it requires timely snapshots of global state within each. Our DRL deployment leverages the circuit breaker abstraction along with a robust gossip-based communication protocol to meet its demanding communication requirements. Through local and wide-area experiments, we illustrate that DRL remains accurate and fair in the face of a variety of failure scenarios.

1 Introduction

The last decade has seen the development and commercial deployment of a handful of services that scale to thousands of nodes while gracefully managing adverse events such as network disconnections, fail-stop failures, and node churn. Examples include peer-to-peer file dissemination, key-value storage, bulk data processing, and multiplayer gaming. To accomplish this feat, system designers generally restrict the dependencies of most operations through techniques such as partitioning and replication, thereby limiting the impact of individual failures on the service as a whole. For the remaining set of operations that require strongly coordinated execution—such as locking or leader election—system architects isolate them to a much smaller set of nodes that execute consensus protocols. This two-tiered design pattern underlies many large-scale services at Google and Yahoo!, which respectively employ Chubby [6] or Zookeeper [14] to help maintain correctness during synchronization, failure, or configuration events.

Not all distributed applications, however, fit well into this scale-out paradigm. In some cases each node must be aware of the actions of all other nodes in the system to make forward progress, ensure correctness, or to have acceptable performance. We call such services *tightly coupled*. This paper describes an efficient group communication abstraction that arose from an effort to deploy one such service, distributed rate limiting (DRL) [23], to control bandwidth usage on PlanetLab.

Throughout this work, we discuss our DRL deployment to provide context for our communication abstraction's requirements. DRL functions optimally when all nodes can communicate with each other. Unlike most tightly-coupled distributed systems, however, DRL has the potential to remain highly available even in the wide area because it can safely function without global coordination (i.e., operate independently in separate partitions). DRL continues, however, to require timely communication between all nodes within a partition.

Existing coordination facilities fail to accommodate DRL's unique requirements. In particular, partition-tolerant systems based upon quorums can endure node failures and network partitions, but they force the application to cease operation in minority partitions. Moreover, they may repeatedly deliver obsolete views during periods of network instability. Hence, we propose a new group membership abstraction, *circuit breaker,* that uses a resilient overlay to improve the system's ability

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.

TRIOS'13, Nov. 3, 2013, Farmington, Pennsylvania, USA.
Copyright 2013 ACM 978-1-4503-2463-2/13/11...$15.00.
http://dx.doi.org/10.1145/2524211.2524214

to agree on new views and reduce the perturbation of applications during periods of asymmetric connectivity.

Like FUSE [11], applications using a circuit breaker participate in failure detection, and the system ensures that group membership event notifications are delivered to all non-failed members of the group. If a node determines that it cannot reach another group member directly, it can attempt route messages to that destination over a resilient overlay [2] provided by the circuit breaker without immediately resorting to a heavyweight view recomputation. By incorporating a resilient overlay, we increase the likelihood that our underlying group membership service will reach agreement during network failure scenarios.

If it does not crash, the sender of a message can be sure that either 1) routing through the overlay will succeed or 2) the circuit breaker will begin computing a new view in which the sender or receiver (or both) may no longer be members. Once the connectivity between membership servers stabilizes, circuit breaker will deliver a new view, and all pairs of members in the group will be capable of communicating (perhaps indirectly via the overlay) with one another until the next failure occurs.

The circuit breaker abstraction enables an architecture for tightly-coupled distributed systems that integrates traditional view-based group membership techniques [5, 16, 20] with a robust, light-weight gossip protocol. Using this architecture, we have successfully deployed distributed rate limiting as an internal management component of the PlanetLab global testbed. The resulting system not only gracefully handles node failures and simple network partitions, but also functions efficiently in the face of the far more common case of many small cuts to the connectivity between limiters.

This paper makes the following contributions:

- **Circuit breaker abstraction:** We define and implement a new coordination abstraction for tightly-coupled distributed applications that conjoins group membership and messaging. The abstraction supports partitioned operation similar to traditional consensus-based view change protocols, but also provides lightweight, reliable communication within a view through a resilient messaging overlay that provides backup communication paths. The system begins computing a new view—i.e., the circuit "trips"—if the messaging overlay fails.
- **Robust gossip-based communication:** In tightly-coupled systems, connectivity cuts can delay or prevent efficient message delivery. For example, DRL requires the ability to rapidly disseminate updates to system state within a group, and transient failures disrupt the calculations necessary for the proper enforcement of the global rate limit. We extend the work of Kempe et al. [17] to provide a gossip protocol that maintains the lightweight, efficient distribution properties of the original while being resilient to transient packet loss. Our evaluation shows that our gossip variant provides accuracy similar to a centralized alternative while providing the scaling benefits of a fully distributed architecture.
- **Integration lessons, DRL case study, and evaluation:** We discuss how both DRL and its gossip protocol leverage the circuit breaker abstraction. In particular, both are *partition-aware*, having the ability to safely resume operation in multiple disjoint views simultaneously, and we describe techniques to enhance re-starts to ensure accurate and fair distributed rate limits. Finally, we compare and contrast two circuit breaker prototypes, each using a different underlying coordination protocol. The first leverages Yahoo!'s ZooKeeper [14], a replicated coordination service, while the second implements a proposed protocol for partitionable [3] group membership, Moshe [16]. We illustrate how their distinct design choices impact their ability to establish and maintain useful views in practical wide-area environments.

The remainder of this paper is organized as follows. We survey related work and briefly review distributed rate limiting in Section 2. Section 3 enumerates the specific challenges that must be addressed by any tightly coupled system that seeks to function well in wide-area environments. We present the circuit breaker abstraction in Section 4 before summarizing its implementation in Section 5. We describe our DRL implementation in Section 6 and demonstrate its correctness and effectiveness in both the local area and on PlanetLab in Section 7. Section 8 concludes the paper.

2 Background and related work

Before introducing the circuit breaker abstraction, we first briefly survey existing approaches to group membership, as well as describe the distributed rate limiting problem, which motivates our design requirements.

2.1 Related work

We are far from the first to attempt to deploy a tightly-coupled distributed service in the presence of failures. VAXclusters maintained a great deal of shared global state [18], but assumed the nodes of the system were in close physical proximity, with relatively reliable, high-speed connectivity between them. ISIS [5] and its follow-ons enabled group-based distributed computation through virtual synchrony, but assumed nodes were distributed among a relatively small number of sites. Within a site, nodes were expected to have abundant, reliable connectivity. In contrast, we seek to operate in an environment where no node can assume reliable connectivity to any significant fraction of other nodes.

More generally, there has been a great deal of work on group membership and distributed agreement protocols [8]. Perhaps the most famous is Paxos [19], which provides distributed consensus. While Paxos can certainly be used to implement a group membership protocol, it does not do so by itself. Most group membership protocols are not targeted for the wide area, and those that are typically leverage hierarchy [7], with Moshe [16] being a notable exception.

The FUSE [11] failure notification service guarantees that, once triggered, notifications reach every member of the FUSE group. Unlike our system, FUSE makes no guarantees about inter-node communication within a FUSE group. Furthermore, FUSE applications must explicitly re-create FUSE groups upon failure event notification, while our system automatically delivers new views when necessary.

There have been many proposals for lightweight communication within a well-connected group, including gossip or epidemic routing protocols [9, 12]. The general problem of using and efficiently computing aggregates across a distributed set of nodes has been studied in a number of contexts, including distributed monitoring [10] and counting [25, 27]. Similar work developed probabilistic *synopses* for robust in-network computation in wireless sensor networks [22]. In contrast Kempe et al. does not require construction of an aggregating topology and, if updates quiesce, converges to the exact answer.

Previous work has been applied to the challenge of coordinating resource usage across a distributed set of nodes. GWiQ-P [15] constructs trees to balance the distribution of resource coins among a cluster of machines that share a global quota. It allows cluster administrators to tune the behavior of the protocol in response to failures between a reliable, heavyweight transaction protocol and an eventually consistent, lightweight coin recovery procedure. DQMP [4] combines a similar lightweight recovery protocol with a replicated, checkpointed management service to diffuse available resource capacity and balance local quotas across nodes. Neither system permits partitioned operation, and their resource balancing objectives may not be suitable for DRL, which tries to replicate max-min rate fairness between distributed TCP flows.

2.2 Cloud control

This work uses a case study of Distributed Rate Limiting (DRL) to provide context for the circuit breaker abstraction. DRL was originally introduced as a cost-control mechanism for cloud-based services [23]. Raghavan et al. observe that while bandwidth cost recovery is typically accomplished through burstable billing (e.g., 95th percentile charging) or metered pricing (e.g., Amazon's EC2 charges incrementally per gigabyte of traffic consumed [1]), many prefer to manage expenditures as fixed-cost overheads, not incremental expenses [13]. Instead, DRL provides the ability to control costs using a flat-fee model, ensuring a fixed price for an aggregate, global usage.

We examine a deployment of distributed rate limiting that addresses two thorny bandwidth management scenarios on the PlanetLab global testbed: In the first case, operators require a mechanism to limit the bandwidth consumed by a set of machines at a particular physical location without requiring that all nodes at the site use the same access link—or requiring the installation of traffic policing devices on the link(s). In the second case, management seeks to limit the bandwidth consumption of an application (or *slice*, in PlanetLab terminology) regardless of the set of nodes on which the slice is deployed. Administrators need such an ability, for example, to sandbox a prototype distributed service, e.g., a content distribution network, to control its aggregate bandwidth usage.

2.3 Distributed rate limiting

In DRL, a set of distributed traffic rate limiters collaborate to subject a class of network traffic (for example, the traffic from a particular slice) to a single, aggregate global limit. Critically, DRL apportions rate across these sites in proportion to *demand*, as if all traffic flowed through a single bottleneck link. Consider that PlanetLab has over 575 distinct physical locations, and may wish to limit the total traffic generated by a particular slice to 100 Mbps. Without DRL, administrators must either limit the slice to 100 Mbps at each location, or limit each site to a fixed portion (i.e., $100/575 \approx 0.173$ Mbps). While the first case allows any site to use the full limit, in the worst case the aggregate usage could reach 57.5 Gbps. While the second case ensures the aggregate won't exceed 100 Mbps, it can artificially constrain the capacity at individual sites if demand is not uniform. In contrast, DRL combines the aggregate limiting with the ability of individual sites to consume the entire limit.

Distributed rate limiting protocols treat traffic in a distributed system as if all packets logically pass through a single traffic policer. Individual flows compete dynamically for bandwidth not only with flows traversing the same limiter, but with flows traversing other limiters as well. Maintaining fairness between flows inside a traffic aggregate depends critically on accurate and timely apportioning of the global limit across limiters. The key technical challenge to providing this abstraction is measuring the *demand* of the aggregate at each limiter, and apportioning capacity in proportion to that demand.

Our implementation employs the flow proportional share (FPS) allocator due to Raghavan et al. [23]. Every

epoch (typically 50–500 ms) each limiter i uses FPS to decide how much of the global rate limit, L_{global}, it may use locally (L_i). To do so, each limiter calculates a single number[1] representing bandwidth demand, D_i. To calculate the local limit, each limiter computes its proportion of the aggregate demand across all n limiters each epoch:

$$L_i \propto (D_i / \sum_{1...n} D_j) * L_{global}$$

An effective DRL deployment ensures accurate ($\sum L_i = L_{global}$) and fair ($L_i \propto \frac{D_i}{D_{total}}$) global rate limits. Given the fundamental dependence of DRL on instantaneous aggregate demand, for the purposes of this paper one can reduce the problem to robustly distributing this aggregate demand information.

3 Challenges

Deploying DRL as a PlanetLab management facility is analogous to one of the authors' original uses cases of deploying DRL at a CDN and, therefore, presents similar real-world challenges. We find, however, that the initial design fails to adequately address several issues in practice. In particular, the original design relied on a scalable gossip-based protocol to take snapshots of aggregate bandwidth usage in the system. However, that protocol, as we discuss in Section 6.1, is highly sensitive to network cuts. Moreover, while DRL can theoretically operate in multiple network partitions simultaneously, the authors' implementation employs a simplistic mechanism for detecting that condition. Moreover, that mechanism also fails in the presence of isolated link cuts (incomplete partitions). We articulate key practical challenges that a robust implementation must address below.

3.1 Failures and partitions

One of the first issues that arises in any distributed system is the potential for nodes to fail or become partitioned from one another. In an asynchronous system—i.e., any real-world deployment—it is impossible to tell the difference between a node failure and a network partition; hence, distributed applications must account for such events in order to proceed safely.

There are two basic strategies for dealing with partitions. The classical approach taken in quorum-based systems [20] is to continue operation in only one (majority) partition—thereby ensuring a consistent view of any system state. An alternative exists, however, for those highly available systems that are capable of operating in multiple groups simultaneously: Nodes can be explicitly separated into disjoint groups on either side of the partition(s) which operate independently. These applications are known as *partition aware* [3].

[1] While FPS calculates demand in a more sophisticated manner, one may think about D_i as the number of active flows.

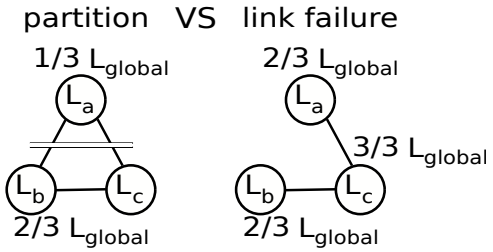


Figure 1: DRL adjusts the global rate limit for partitions based on their size. A partition of limiter a results in both b and c removing a third of the global rate limit they use (left). In contrast, a partial communication failure can result in some set of limiters (here limiter c) participating in both "partitions" (right).

Distributed rate limiting is an example of this latter class of applications. In particular, multiple nodes may be partitioned away from each other, but still be connected among themselves. DRL can continue to allocate rate dynamically among connected components while ensuring that total usage across all partitions remains $\leq L_{global}$. To support this strategy, each limiter needs to track current group membership, and for all limiters in a group (partition) v_i, proportionally adjust its notion of the global limit by $\frac{|v_i|}{n}$. This policy ensures that the global aggregate remains bounded correctly whether limiters truly fail or are simply partitioned away.

Consider, for example, the case of a network partition in a set of three limiters, as depicted in the left half of Figure 1. Initially, all three limiters will allocate rate according to their relative demands. After the partition occurs, neither side of the partition can know the total demand of the other. If, however, DRL nodes are aware of the size of their current group (on either side of the partition), they can leverage knowledge of the total number of limiters in the system (as opposed to the current group) to detect when its estimation of aggregate demand may be incomplete and apportion the limit appropriately.

3.2 Incomplete connectivity

Regardless of whether a partition-aware distributed system continues to operate in one group or many, the presumption is that connectivity is complete within a group and entirely absent between them. If this is not the case—if members of the same group cannot communicate with each other—even partition-aware designs may fail. In DRL, if link cuts do not completely partition the network, the simple strategy of maintaining local views of group membership can result in exceeding the global limit. The right side of Figure 1 illustrates this case. Here limiters a and b both believe they are in sepa-

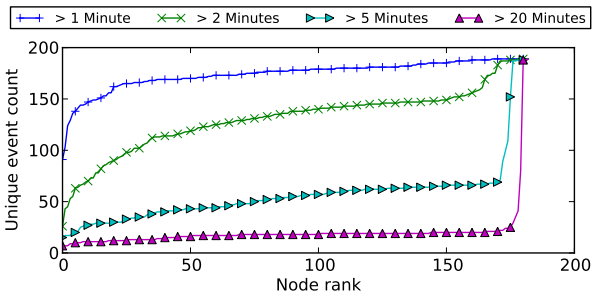


Figure 2: 182 PlanetLab hosts sorted by number of cuts over a six-day period. Each line plots the number of distinct link cuts longer than the specified duration.

rate partitions with c. If, for example, c has no demand, a and b can both acquire 2/3rds of the global limit.

Clearly, each node must belong to exactly one group, and existing consensus-based group membership protocols effectively enforce this invariant. However, these protocols typically do not provide any facilities to handle incomplete connectivity *within* a group. They do not require full connectivity to successfully form a group, and rely upon the application itself to detect communication failures and request a new view that excludes the (partially) unreachable node. Alternatively, a conservative group membership scheme could require full connectivity before even forming a group. In either case, any link cut can lead to the costly formation of a new (set of) group(s).

This conservative approach might be practical if partial connectivity resulting from link cuts were a rare occurrence. Unfortunately, such cuts occur far more frequently in the wide area than true partitions or persistent node failures [2, 29]. We quantify this well-known [26] phenomenon in PlanetLab in Figure 2 by considering the connectivity between the 182 PlanetLab nodes we use in Section 7. We present the frequency of link cuts during a six-day period in September of 2010. Here, each node sends a small UDP probe to every other node every 15 seconds and then tracks the set of nodes from which it does not receive probes. Nodes record when a peer is silent for one, two, five, and 20 minutes. The figure orders the individual nodes by the total number of links to other nodes that failed at least once during the trace (the maximum being 181). We plot four (independently ordered) lines, each considering a different minimum cut duration.

While there is a diversity of reliability, ranging from a few nodes that rarely experience cuts to one node that was cut off from every other node at some point, no nodes are ever completely partitioned in this experiment. Almost all nodes, however, experience at least one short ($\geq$ 1 minute) or moderate ($\geq$ 2 minute) connectivity failure to a majority of the nodes. Moreover, while not shown here, nodes occasionally exhibit asymmetric connectivity, where a can send messages to b, but the reverse was not true. If each change in connectivity requires new groups to be formed, then with this rate of transient failures, the membership system never stabilizes.

4 Circuit breaker

We now present the circuit breaker abstraction, which explicitly addresses the challenges from the previous section. Namely, it improves the ability to reach consensus on group membership in the presence of partial connectivity and, critically, facilitates the continued operation of a group in which all nodes may not be capable of directly communicating with one another. Circuit breaker provides a consistent view of group membership through a *group membership service* while simultaneously ensuring that, even in the face of many (but not all) forms of asymmetric or non-transititive network cuts between them, any two members of a group will be able to communicate. Furthermore, partition-aware applications can employ a circuit breaker to operate independently in multiple, simultaneous groups with incomplete connectivity.

We realize the circuit breaker abstraction by combining two existing techniques: view-based group membership and overlay-based routing. The defining characteristic of our design is the separation of concerns—yet sharing of fate—between the mechanisms responsible for group communication and overlay routing. Communication between members of a group can employ any protocol of the application's choosing, and it need not employ the circuit breaker in the common case. Maintaining group membership information, however, is always performed by the circuit breaker.

In our design, the circuit breaker is implemented on a small set of well-positioned membership servers[2]. Each node in the system associates with exactly one membership server, and the servers form a full mesh between each other. Hence, regardless of whether they can contact each other directly, any two members of a group are indirectly connected through an overlay path of at most three hops consisting of the links to their respective membership servers and possibly a link between the membership servers (if they do not share the same server). Nodes leverage the overlay provided by the circuit breaker when they cannot communicate directly.

4.1 Membership views

The circuit breaker's group membership service runs an agreement (i.e., distributed consensus) protocol across a distributed set of servers to construct one or

[2]For our case study deployment of DRL on PlanetLab, we use the data underlying Figure 2 to position membership servers

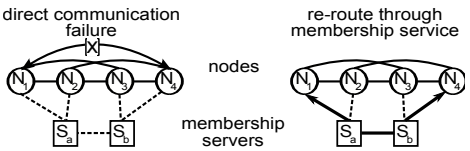


Figure 3: Our circuit breaker design incorporates logically distinct membership servers (S_a and S_b) connected to application nodes. Nodes in the same group normally communicate directly. When cuts occur, they leverage the connectivity between membership servers.

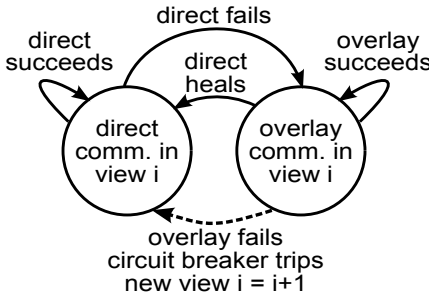


Figure 4: The sequence of events within a circuit breaker.

more *views* of the current group membership containing the currently active group members. There may be multiple, simultaneous views in the case of network partitions. Each participant associates exclusively with one server (though they may fail over to other servers) and receives group membership updates from that server only. Participants inform their server of potential changes in group membership with explicit join/leave requests. The membership service guarantees that views have unique, monotonically increasing identifiers within a partition.

The group membership service delivers a succession of views to all of the members in the system. There are, however, instances in which the service may not be able to agree on the next view or not support the existence of simultaneous views. The membership service will inform the impacted group members if either situation occurs, allowing the application to react by entering a safe state. For example, DRL limiters so notified enter a "panic" mode in which they safely enforce static local limits of $1/n$, as if they were running alone in a group of size one.

The particular choice of agreement protocol employed by the circuit breaker membership service directly influences whether multiple views can exist simultaneously, how quickly views are established, and the ability to reach consensus during cuts to connectivity between the servers providing the membership view service. Our current implementation supports two existing, scalable agreement protocols, Moshe [16] and Zookeeper [14]. We employ both to serve our needs, but their underlying designs (Section 5) provide contrasting performance and semantic guarantees.

4.2 Communication within views

All nodes receiving the same view—i.e., currently in the same group—expect to be able to reach each other directly. We rely on the application to report when direct-communication conditions deteriorate to the point of failure. Upon detecting such a failure in direct communication, group members communicate indirectly through the circuit breaker messaging overlay. In this section, we describe a circuit breaker's behavior during several common failure scenarios.

Figure 3 illustrates the distinct logical networks that connect nodes to one another and to the membership service. Here a link between two nodes has failed and the nodes use the messaging overlay to maintain connectivity. Critically, a circuit breaker's messaging overlay ensures that despite the separation of group communication and view maintenance, they share connectivity fate. The membership service depends on the messaging overlay to monitor the status of clients and other servers. Thus the messaging overlay will always succeed when the view is valid and fail when a new view must be delivered. Hence, if a node is unable to communicate through the circuit breaker's messaging overlay with any node in its current group, circuit breaker will try to form a new view in which, barring additional failures, it can communicate with all other members (although it is likely the previously troublesome node will no longer belong to the group in the new view). We say that the circuit breaker has *tripped*, and resets by computing a new view with the current set of stable members. Figure 4 depicts these events and summarizes the circuit breaker's reactions.

4.3 Failure scenarios

We illustrate how a circuit breaker reacts to a range of possible failure conditions through a canonical three-node graph and a representative set of cuts that demonstrate the various classes of (dis)connectivity. Figure 5 illustrates a system that has six nodes and three membership servers, each responsible for two nodes.

The first case we consider fully partitions a server, S_a, and its nodes, N_1 and N_2, from the other servers and nodes. Case 2 presents network cuts between any two nodes (though shown here only between nodes attached to different servers). Case 3 partitions the membership servers into $P_1 = \{S_a\}$ and $P_2 = \{S_b, S_c\}$, while Case 4 introduces a cut in the mesh connectivity of the membership servers. For our DRL deployment in the Planet-

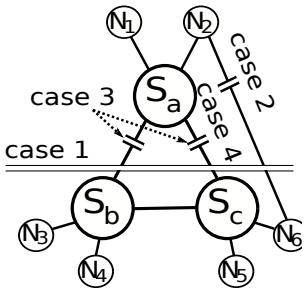


Figure 5: An example topology with three membership servers, S_i, each with two clients N_j, in which we assume that any pair of nodes can initially communicate. We use this topology to study four distinct failure cases.

Lab environment, Case 2 is the most prevalent/important type of disconnectivity.

Each of these cases presents a challenge that must be handled appropriately by the application and circuit breaker. In Case 2 the intra-group connectivity cut will cause the application to exploit the circuit breaker's messaging overlay while preserving the group membership. Cases 1 and 3, on the other hand, trip the circuit breaker, resulting in a partitioning of the group. Case 4 may or may not trip the circuit breaker, depending on the semantics of the membership service employed by the circuit breaker implementation, as discussed in the next section.

Note that we expect most deployments will co-locate circuit breaker membership servers with application nodes. This assumption implies that a full partition cannot occur between nodes without affecting the servers if each node remains connected to its membership server. Any node not connected to a membership server is removed from the current view, and the node's application is expected to respond appropriately (in DRL, nodes default to $1/n$ limiting).

While important for providing connectivity during failures, re-routing messages via the overlay isn't ideal. Doing so increases the latency of communication and puts an additional load on the membership servers. Hence, ideally the application should be able to to cope with transient packet loss itself. TCP is an obvious option, but may not be well suited for applications that require bounded delay. In such instances, alternate mechanisms must be employed to detect paths with sustained, heavy packet loss and ameliorate the impact of occasional packet loss on the remaining links. We describe the our DRL implementation's mechanism in Section 6.

5 Membership implementation

Our circuit breaker implementation can leverage either of two underlying agreement protocols, Moshe or Zookeeper. Moshe is a partitionable group membership service protocol, proposed by Keidar et al. [16], that has been optimized for the wide area. We implemented Moshe in 1,100 lines of Python. Our implementation combines the multiple, simultaneous views of the original Moshe protocol with an integrated messaging overlay. In contrast, Yahoo! designed Zookeeper [14] for use within the data center, and it presents its clients with a shared "file-system" abstraction for implementing a range of coordination primitives, including leader election, status propagation, and group membership. We employ an unmodified version of the publicly available Zookeeper distribution maintained by Yahoo!. We do not alter the API of either system.

While both systems provide group membership, they use different mechanisms to reach agreement on the next view. Thus, they behave quite differently in the event of network partitions and cuts in connectivity between servers. Moshe provides partitionable group membership, allowing servers to be partitioned from one another while continuing to deliver views to their connected clients. This feature allows partition aware systems to continue to operate in each partition as described in Section 3.1. Conversely, quorum-based designs, such as Zookeeper, maintain a single active view, removing nodes from the view when their membership server is not in the quorum. Both systems allow circuit breaker to cope with the common case of transient failures between clients without creating new views.

5.1 Moshe

View maintenance. Moshe servers monitor the liveness of their attached clients via periodic heartbeat messages. When a server detects a failure (or client join), it propagates the event to all other servers and starts a view change. The servers integrate these events into an `NSView`, which is the set of group members that are not currently suspect. A Moshe server uses the `NSView` as its proposal when executing its membership protocol, whose fast path can converge in a single communication round.

When a membership event arrives, each Moshe server issues a `startChange` event to its clients, indicating that the system is starting agreement on a new view. Moshe is designed to avoid delivering stale views and will thus wait until all servers within a connected group agree on the set of known failures. Upon receipt of this message, applications must proceed conservatively until a new view is established (for example, DRL enters a safe state in which it reverts to $1/N$ limiting).

Messaging overlay. Our implementation of Moshe uses a separate messaging layer to relay membership events, `NSViews`, and other control messages between clients and Moshe servers. Clients see an extended

Moshe interface that exports this layer as a simple blocking messaging API. This API allows client applications to send and receive datagrams between each other. When a client sends a datagram, Moshe relays the message through the client's membership server.

At the servers, the messaging layer implements a simplified form of the link-state routing protocol used in the Resilient Overlay Network (RON) [2] such that if they share a third, mutually-connected intermediate server, a pair of servers can relay messages to one another through it. This modification, which was proposed but not implemented in the original Moshe protocol, allows Moshe servers to reach agreement even if a significant fraction of the inter-server links are cut (but do not create a partition). In contrast, the original Moshe design fails to reach agreement if any link between servers fails. We refer the reader to [16] for further details regarding the conditions under which Moshe servers may not reach agreement.

5.2 Zookeeper

View maintenance. Group membership is easily layered on top of Zookeeper's "file-system" abstraction, which maintains a set of shared objects, called znodes, in a hierarchical namespace. Zookeeper serializes and applies client updates to znodes, returning the same sequence of changes to these shared objects to each connected client. For membership, clients add (or remove) themselves to the current view by creating (or deleting) a znode in the hierarchical namespace. This directory of znodes represents the current view of system membership. Clients may "watch" a directory of znodes, receiving alerts when any child znodes change.

To detect changes in membership, clients leverage *ephemeral* znodes, which are objects removed by Zookeeper when the client that created it has failed. Thus Zookeeper automatically removes a group member from the view when it fails to remain connected to its Zookeeper server. When re-connecting, clients re-insert their znode into the hierarchy to re-join the view.

To understand the semantics Zookeeper provides as a group membership service, it is useful to understand how it applies client updates. Zookeeper runs a leader election protocol to ensure that a quorum of servers has an elected leader. A client's server relays writes to the leader and the leader propagates those updates to all replicas (servers). It employs a totally ordered broadcast protocol, Zab [24], which implements a simplified form of Paxos. Note that up to f Zookeeper servers may fail (assuming $2f+1$ servers) and recover statefully.

This design has important implications for the performance of client systems during partitions and cuts. First, it is optimized for read-dominated workloads, where replicas can easily serve data without running the consensus protocol to update state. Second, to create a view the leader must be able to update (reach) a quorum of servers. If a server cannot reach the leader or find a new quorum to elect a new leader, then it assumes that it is not in the primary partition. In this case, the server will tell its clients that it is no longer able to return the latest view. Thus, cuts between servers not only force a new view to be delivered, but my also force clients to be without a view for the duration of the failure.

Messaging overlay. By design, the znode interface allows clients to coordinate with one another through the membership service. We built a generic message passing abstraction using znodes to emulate per-client mailboxes. In this case, there is an additional, but persistent znode for every group member that represents its inbound mailbox. Clients send messages by writing a new ephemeral znode with the message contents under the destination's mailbox znode. By using a "sequence" znode for the mailbox, the system assigns new message znodes monotonically increasing sequence numbers, ensuring ordered message delivery.

Having independent znodes per message avoids the overhead of Zookeeper's read-modify-write procedure for updating existing znodes. However, Zookeeper must still perform consensus for each message (providing the global message sequencing). While expensive, this design obviously meets our messaging requirement: a failure to deliver a message implies a disconnection that will force Zookeeper to deliver a new view.

6 DRL implementation

We used circuit breakers to implement a distributed rate limiting service for PlanetLab. Here, we describe the barriers to an effective wide-area deployment and how our circuit-breaker-based architecture addresses them. We also present a hardened gossip protocol that functions well in the wide area, and optimizations to improve the performance of gossip when used in conjunction with view-based membership services like circuit breakers.

6.1 Communication requirements

As described previously, the key requirement for efficient operation in DRL is to maintain an accurate demand aggregate as the network paths between limiters experience packet loss or connectivity cuts. DRL avoids single points of failure and performance bottlenecks through a peer-to-peer design, wherein each limiter maintains an approximation of the aggregate by periodically communicating with other limiters about their local demand. Moreover, DRL provides proportional partition tolerance, where each limiter only computes

the aggregate demand on its side of a partition, and each connected component apportions the fraction of the total limit in proportion to its membership as described in Section 3.1.

Within each connected component, the original DRL implementation employs a gossip protocol based upon the push-sum protocol proposed by Kempe et al. [17]. Like other gossip protocols, it scales well, sending a total of $O(n \log n)$ messages per communication round (a limiter epoch in our case), where n is the number of communicating limiters. The protocol works by having each node send a fraction of its current estimate of the aggregate bandwidth demand to its neighbors. To do so, nodes maintain two values, participant's portion p of the demand aggregate, and a protocol weight w. The protocol begins by setting p to the locally observed demand and w to one. During a protocol round, each limiter chooses b peers uniformly at random, where b is the *branching factor*. It then divides its portion and weight by $(b+1)$ and sends equal amounts to each of the selected peers, keeping the additional share for itself.

At periodic intervals[3], a node updates its local demand measurement using flow-proportional share [23]. It then estimates the demand aggregate by computing $(\frac{p}{w} \cdot n)$. This estimate converges to the true demand aggregate in $O(\log n)$ messaging rounds and has the nice property that, when gossip starts, the initial reported aggregate at each limiter will be $(\frac{D_i}{1} \cdot n)$. Thus initial local allocations will be at most $1/n$ of the total limit. While Kempe's original protocol calculates the aggregate of a set of fixed values across the system, it also continues to converge as values change over time.

In theory, gossip protocols can function in partially connected environments. DRL's Kempe-inspired gossip protocol, however, depends upon the conservation of the total amount of weight in the system, which is equal to the number of limiters. Any gossip messages that do not successfully reach their destination take with them a fraction of the system's weight. If the local values (in our case local demand) do not change, the algorithm will continue to return the correct aggregate. However, if local demand changes *and* weight is not conserved, the reported aggregate quickly diverges.

6.2 Employing circuit breakers

Here, we describe how we leverage circuit breakers to address DRL's communication requirements. At a high level, limiters use view information provided by a circuit breaker to set the current aggregate limit in proportion to the fraction of the initial limiter set active in the current view as discussed in Section 3.1, and adjust parameters of the gossip protocol to make effective use of the existing communication links with each view. Additionally, we extend the gossip protocol to manage packet loss and leverage the circuit breaker's messaging overlay to deal with network cuts. While TCP could be employed to address packet loss, we eschew this approach for two reasons. First, we wish to avoid the overhead of connection set up and tear down at fine-grain time intervals or maintaining hundreds of open connections. Second, retransmission delays can negatively impact convergence[4].

6.2.1 A loss-resilient gossip protocol

Every epoch, DRL communicates its local demand to a set of peer limiters and recalculates its estimate of the aggregate demand based upon updates it receives. To generate this update, each limiter gossips its updated local demand values to a random set of peers in the current view as described in Section 6.1. To compensate for transient packet loss, the gossip protocol employs a cumulative acknowledgment mechanism that ensures that peers eventually receive all outstanding portion and weight values. Rather than "firing and forgetting" every message, each limiter records the amount of outstanding portion and weight (the gossip "values") for each peer. When next transmitting a message to a peer, a node explicitly acknowledges these values by piggybacking an acknowledgment in the message. This acknowledgment allows senders to "free" the outstanding values from their bookkeeping.

To do so, senders maintain separate sequence number spaces for each peer. For each destination, senders transmit the current round's gossip values plus the current unacknowledged weight from previous rounds. The messages also carry a lower-bound sequence number and a current sequence number. The lower-bound sequence number represents the last acknowledgment from this peer, and the current sequence number increases monotonically. The receiver uses the two sequence numbers in the message to determine how much, if any, of the gossip values contained in a message should be incorporated locally. Thus, if at any point in the protocol a message or acknowledgment is dropped or duplicated, neither weight nor portion is permanently lost, though it may require additional messaging rounds for the nodes to detect the loss and reclaim them. As a result, the weight conservation property is preserved and the protocol continues to operate correctly.

There are a few important observations to make about this protocol. First, gossip values may still be lost for some period of time, and the aggregate may diverge. Thus this protocol works well for short periods of loss, but not for longer-term link failures. In brief, the pro-

[3]For the PlanetLab deployment, we used a 500-ms interval.

[4]Because communication is intermittent, retransmission depends upon a timeout (RTO), not the fast retransmission mechanism (reception of duplicate acknowledgments).

tocol provides the scalability of gossip-based communication, but requires an accurate view of inter-limiter connectivity—it cannot function across cuts.

6.2.2 Leveraging the messaging overlay

DRL employs a two-part solution to handle link cuts. First, each limiter passively monitors inter-limiter links for persistent loss and adds them to a local "black list." Essentially an application-layer link failure detector, this monitoring mechanism piggy-backs upon existing gossip messages, avoiding the $O(n^2)$ messaging inherent in a straightforward all-pairs ping approach. A limiter suspects a link/peer is down/unreachable after it sends r messages without receiving an acknowledgment (we call this a *loss* event). This black listing is critical for gossip, since it prevents the protocol from losing weight by communicating on lossy links.

When a limiter blacklists a link to a peer limiter, it tries to reach the peer via the circuit breaker messaging overlay. If it succeeds, the two limiters *reconcile* any outstanding gossip protocol weight. Both limiters then continue as normal but exclude each other as potential gossip partners. They also periodically attempt to reestablish direct communication, and, if they succeed, remove the link from the blacklist. Using the connectivity provided by the circuit breaker as a backup allows DRL to avoid the cost of re-starting gossip when inter-limiter links fail.

Circuit breaker trips are also discovered passively. Limiters employ the messaging overlay only when the gossip protocol adds a link to a limiter's black list, and a failure is declared only upon repeated packet drops along a link in the overlay. (Recall that circuit breakers rely on the application to detect connectivity failures, so do not trip themselves.) As Figure 3 illustrates, membership servers must be able to connect to the limiters to deliver view changes, and to each other to reach consensus. Hence, any partition in the membership service topology will trip the circuit breaker. For instance, a limiter disconnection forces the circuit breaker to deliver a new view without that limiter. Similarly, a disconnection between membership servers implies that they cannot reach consensus. This condition may cause the circuit breaker to deliver new views and/or some limiters to enter a safe state [5] as they realize they have not yet received a new view. Hence, our use of the circuit breaker's messaging overlay as a last resort ensures a consistent view of connectivity.

To understand the benefits of this fate sharing, consider an alternate design that leverages a dedicated routing overlay running on the limiters themselves. If this overlay fails to find a path between two limiters whose direct path has failed, the gossip protocol cannot resolve lost weight and the aggregate will diverge. If this failure were decoupled from the delivery of views, the global rate limit may be over- (or under-) subscribed for an indeterminate period of time. However, a failure to communicate through the circuit breaker's messaging overlay will trip the circuit breaker, causing all limiters to enter a new view (or realize that a view cannot form). This event will reset communication to the new view, restoring the ability of DRL to maintain the global limit.

[5] In DRL, we refer to this is a "panic", and a limiter in this state sets its local limit to $1/n * L_{global}$

6.3 Managing constant change

In general, changes in view membership are disruptive to the limiting functionality of the DRL system. While, in principle, view-change events only need to adjust L_{global} to account for the size of the new view, practical considerations significantly impact the performance of the aggregation protocol. In particular, due to potential inconsistencies in limiter states across view changes, it is generally necessary to restart gossip in each new view.

In particular, due to the impossibility of determining whether gossip packets were successfully received at a now disconnected limiter—and, hence, the inability to ensure weight conservation—the DRL restarts its gossip protocol for each new view. Since the protocol converges in $O(\log n)$ rounds, it takes approximately 5 seconds to re-converge with 500 limiters, 500-ms epochs, and a branching factor of two. Recall that the initial aggregate computed is $(n \cdot D_i)$, however, which causes every local limit to be at most $1/nth$ of the global limit. Limiters with additional demand will drop packets, which will likely trigger backoffs for many TCP flows that will subsequently need to rediscover their fair share. We therefore seek to minimize view changes while ensuring the aggregate computed within the view remains accurate.

Our implementation contains several optimizations that attempt to improve performance with the current view and lessen the impact of view changes that do occur. We also suggest an extension to determine when new views may be advisable even if not strictly necessary.

Efficient gossip restart: Recall that restarting gossip—which happens every time the view changes—forces the reported aggregate to be $(n \cdot D_i)$. Thus, until gossip re-converges, it may report an aggregate value that is significantly different from the aggregate in the previous view, even if demand has not changed. While the restarted protocol will converge within a few rounds, we reduce the aggregate's fluctuation by blending the last aggregate value of the previous view with the aggregate returned by the new gossip instance for a number of gossip rounds equal to the number expected to reach

convergence, $log(n)$. We blend the prior aggregate, A_i, with the new one, A_{i+1} using an exponentially weighted moving average. EWMA smooths the transition from A_i to A_{i+1} over a few epochs.

Fast view propagation: While gossip converges in approximately the same number of rounds everywhere, each limiter may become aware of the new view at different times. To facilitate a rapid changeover to a new view, we allow limiters to learn of new views not only from their circuit breaker membership server, but also from their peers. Specifically, when a limiter receives a gossip message with a higher instance number than is current (indicating a gossip restart due to a view change), the limiter will stop participating in its current gossip instance and begin gossiping only with the source of this message (who must also be in the new view) and any others it receives with the new instance. Eventually, limiters learn of the new view from the circuit breaker and update their neighbor set.

Choke point detection: Finally, though beyond the scope of this paper, we observe that it may be desirable to proactively create a new view based on connectivity. For example, there may be instances where only a few links exist between two otherwise well-connected components. Consider the (albeit unlikely) event when the only connectivity is through the circuit breaker messaging overlay. In the event that the system is sufficiently large, this lack of connectivity could severely degrade the communication protocol's ability to converge to an accurate global demand. We conjecture that DRL could passively monitor inter-limiter connectivity (collecting link failed events) and then use min flow/max cut algorithms could be used to determine when such a "choke point" exists. At this point, DRL could trip the circuit breaker to create an artificial partition to improve performance.

7 Evaluation

We now evaluate the utility of circuit breakers and our resulting architecture by considering how effectively our DRL implementation manages changes in connectivity between limiters. In particular, we quantify the price DRL pays for its fully distributed design by comparing its performance to a centralized design in the absence of network partitions.

First, however, we compare the differing behavior of alternative circuit breaker membership services in response to the four canonical failure classes described in Section 4.2 using a local area testbed. Continuing in the local area, we then illustrate the robustness properties of our gossip aggregation protocol during packet loss and link failures. Finally, we demonstrate the ability of the system to maintain accurate limits by running

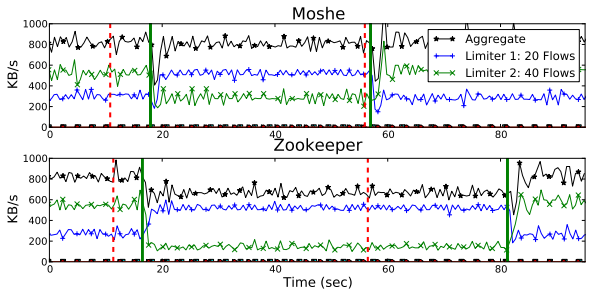


Figure 6: The behavior of DRL, running with a circuit breaker backed by Moshe and Zookeeper in Case 3 (membership server partition), both with blacklisting enabled.

DRL across 182 world-wide PlanetLab nodes[6]. To separate the effectiveness of our view-based group membership and communication mechanism from the DRL algorithm itself (which is due to Raghavan et al.), we measure the accuracy of the demand aggregate computation independently from DRL's ability to police traffic.

7.1 Local-area validation

All local-area experiments run on six physical machines connected via Gigabit Ethernet. They run the 2.6.22.19 Linux kernel included in the 4.2 MyPLC release. Four machines have 3 GB of memory and 2.33-GHz Xeons, while the other two have 1 GB of memory and 1.266-GHz PIII processors. There is a limiter on each physical machine and three of them are co-located with membership servers (on the faster CPUs). None of our experiments are CPU limited. Both Zookeeper and Moshe operate as independent servers in their own slices.

7.1.1 Comparing membership services

We consider the performance of Zookeeper- and Moshe-based circuit breakers on each of the four failure modes shown in Figure 5. All of our local experiments share the same static traffic pattern: L_2 services forty TCP flows and L_6 services twenty TCP flows with a shared, global limit of 6 Mbps. All flow sources send as fast as possible for the entirety of the experiment to an off-testbed sink. Thus, without failures, we expect L_2 to acquire 2/3rds of the global limit, as DRL sets limits proportional to the flow count if all flows have effectively infinite demand.

Figure 6 illustrates the benefits of partitionable group membership when the membership servers experience a partition (Case 3). Here, solid green vertical lines indicate the time(s) at which the membership services delivers a new view. Note the perturbation in rates as both

[6] 24 in Asia, 48 in Europe, and 110 in the Americas.

Case	Optimal Rates	Optimal Behavior	Zookeeper	Moshe
Case 1: Full partition	$L_2 \propto \frac{1}{3}, L_6 \propto \frac{2}{3}$	2 View	Minority Panic	2 View
Case 2: Limiter cut	$L_2 \propto \frac{2}{3}, L_6 \propto \frac{1}{3}$	1 View	1 View	1 View
Case 3: Server partition	$L_2 \propto \frac{1}{3}, L_6 \propto \frac{2}{3}$	2 View	Minority Panic	2 View
Case 4: Server cut @ S_a and S_c	$L_2 \propto \frac{2}{3}, L_6 \propto \frac{1}{3}$	1 View	Minority Panic	1 View

Table 1: How the membership services react to the four failure scenarios. **1 View**: DRL active in one view. **2 View**: DRL in 2 views $\{L_1, L_2\}$ and $\{L_3, L_4, L_5, L_6\}$. **Minority Panic**: Panic in $\{L_1, L_2\}$.

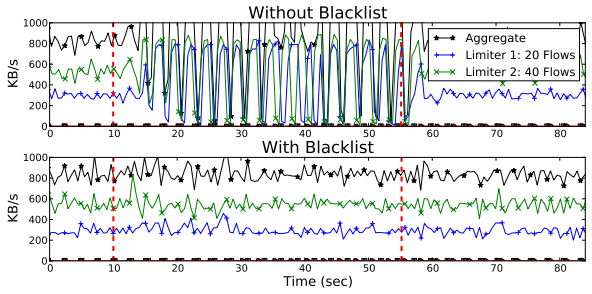
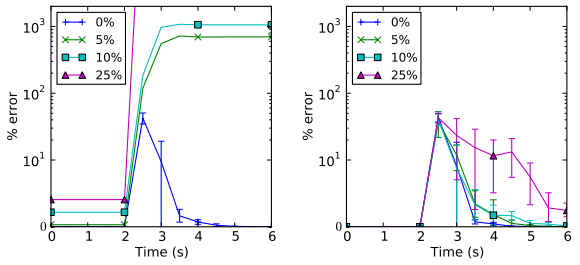


Figure 7: The behavior of DRL, running with a Moshe-backed circuit breaker, with and without blacklisting in failure scenario 2 (client-to-client network cut).



(a) Kempe (no ACKs). (b) Hardened gossip (ACKs).

Figure 8: The effect of packet loss with and without the acknowledgment protocol. Initially the protocol is converged, but with high packet loss and no acknowledgements, the initial convergence may report a slightly incorrect value (a).

systems restart gossip after delivering new views. Here Moshe delivers two views, $\{L_1, L_2\}$ and $\{L_3, L_4, L_5, L_6\}$, which allow L_2 to receive a 1/3rd of the global limit (2 Mbps) and L_6 to acquires the remaining 2/3rd's (4 Mbps). In contrast, Zookeeper forces L_2 (and L_1) to enter the safe "panic" state, since they are in the minority partition, leaving 1/6th of the global limit unused. Finally, Zookeeper reacts more slowly to the healed partition because it exponentially backs off when probing server liveness.

Table 1 summarizes the outcome of all combinations of the four failure scenarios and group membership services. In each row we have shaded the systems with desirable results. Both Zookeeper and Moshe maintain the original view during inter-limiter network cuts, the most common scenario (Case 2). In scenarios 1 and 3, Moshe avoids the static panic modes of Zookeeper, which force limiters in the minority partition to panic. Finally, because Moshe can route around single inter-server cuts, Moshe will continue to deliver the same view in Case 4.

7.1.2 Robust gossip

For the next experiment, Figure 7 shows the results of running DRL with Moshe with and without using the blacklist/overlay for a single cut between two limiters (Case 2). The red vertical dashed lines represent both the onset (at time 10) and resolution (at time 55) of the cut event, in this case between L_2 and L_4. Without the use of a blacklist, gossip diverges, causing both L_2 and L_6 to synchronously oscillate between believing they each have the entire limit and believing they have none of it. By routing through the circuit breaker messaging overlay to recover gossip weight, DRL maintains accurate limits despite the cut.

Satisfied that our implementation properly handles clean cuts in network connectivity, we now explore the behavior of our gossip protocol under adverse environments with varying degrees of packet loss. These experiments study the ability of the communication protocol to report accurate aggregates. Instead of traffic demand, limiters in this experiment advertise a fixed value that changes at pre-defined times, allowing us to know the aggregate.

Figure 8(a) shows the effect of packet loss on DRL's original gossip protocol by reporting the average relative error from the true aggregate across all six limiters. Initially the system has converged; at 2 seconds all limiters report a new value. Even with a modest rate of packet loss, the reported aggregate experiences a substantial relative error. In contrast, Figure 8(b) illustrates the same experiment employing our acknowledgment scheme presented in Section 6.2.1. Even when there is 25% packet loss, the system converges to the true aggregate.

The next experiment illustrates how blacklisting gossip links and re-routing through the circuit breaker messaging overlay can assuage persistent connectivity issues. Here limiters report new values every 9 seconds, requiring gossip to re-converge to the new aggregate. Figure 9 shows the average relative error in the system during the experiment. After a warm up period, we cut

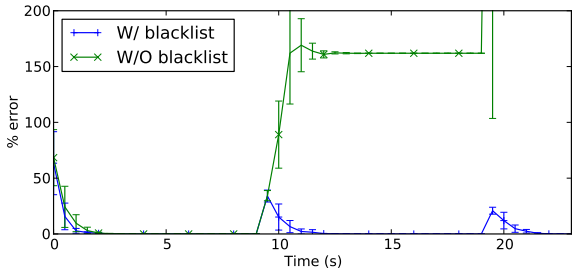


Figure 9: A link is cut at five seconds, which affects aggregate accuracy when limiters report new values (every 9 seconds). Without reconciling through the overlay (blacklisting), the aggregate quickly diverges.

one link at 5 seconds. Since the limiters do not change their advertised value, the aggregate remains stable regardless. However, when the limiters change their values at 9 seconds they converge to the wrong value if they attempt to gossip across the cut. By blacklisting the cut link, the system reconverges by resolving outstanding weight through the circuit breaker messaging overlay.

7.2 Wide-area experiments

We now explore the behavior of our DRL implementation in the absence of catastrophic node failures or network partitions—i.e., a case in which a (simpler, likely higher-performance) centralized alternative might suffice. If a fully decentralized implementation is similarly performant, its increased robustness and availability properties make it attractive for world-wide deployment.

DRL (with Moshe) has been deployed as a trusted component on PlanetLab nodes since late January 2010. These experiments use 182 PlanetLab nodes, and deploy three membership servers, each co-located with a particular node. We place a membership server on each continent where PlanetLab nodes exist: North America, Asia, and Europe. We set the gossip branching factor to $\lceil \log_2 182 \rceil$. An ordinary slice on each limited node runs a simple traffic generator to which we can connect to create bandwidth demand.

7.2.1 A centralized alternative

As an optimistic comparison point, we also compute aggregates using a centralized aggregator, similar to tree-based schemes in large-scale system monitoring [21, 28]. We implement a scheme in which individual nodes relay their local demands to a single, collector node in each view that computes the sum. While this approach converges in $O(1)$ rounds and has a lower per-round cost ($2n$ versus $n \log n$) than gossip, it also exhibits a single point of failure (or scalability bottleneck) in each view, i.e., the collector.

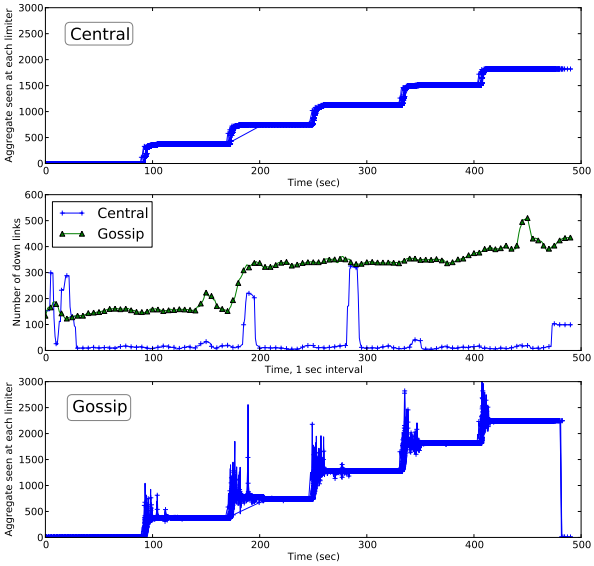


Figure 10: The performance of the gossip and centralized aggregation protocol leveraging Moshe-based circuit breakers across 182 PlanetLab nodes. The middle graph plots the number of failed links encountered by each protocol.

Like gossip, the centralized aggregator must restart on a new view. However, in this case the limiters use the view to re-elect a new aggregator, delaying accurate snapshots of global demand for approximately two epochs. This communication protocol also blacklists inter-limiter links and leverages the circuit breaker messaging overlay. However, unlike our gossip protocol, it will continuously send messages through the overlay until the link heals or a new view arrives.

7.2.2 Detecting and healing cuts

Before demonstrating traffic rate limiting, we first validate the behavior of our gossip aggregation protocol on 182 nodes in the wide area using a simple 6-minute experiment. Like our initial local-area tests, there is no traffic; instead, each limiter reports a fixed demand value at each epoch, allowing us to compare reported aggregates to ground truth. We benchmark the gossip aggregation protocol against the centralized aggregation alternative described in Section 7.2.1.

Initially, all limiters report the value 0. Every 80 seconds, 38 additional limiters change their value to 10. This change results in the "stair-step" shape graph that both the centralized aggregator (top) and gossip protocol (bottom) produce in Figure 10. While centralized converges almost instantaneously, gossip exhibits a short period of instability before it converges.

In fact the stable performance of both protocols belies that fact that connectivity is not perfect. The middle graph plots the count of failed (directed) links ob-

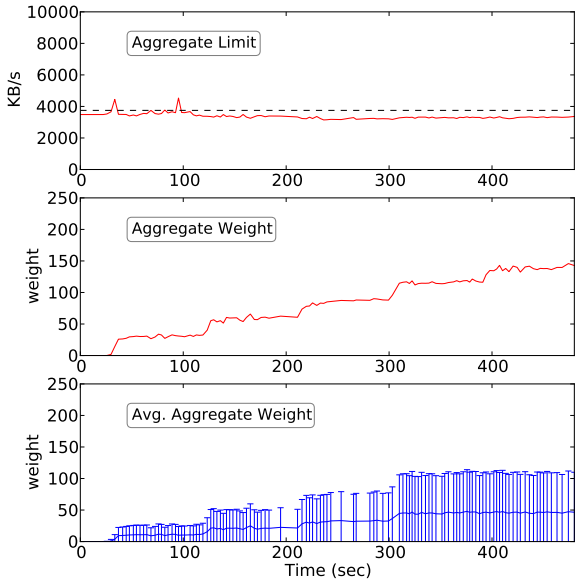


Figure 11: DRL enforcing a 30-Mbps global limit across 182 limiters worldwide.

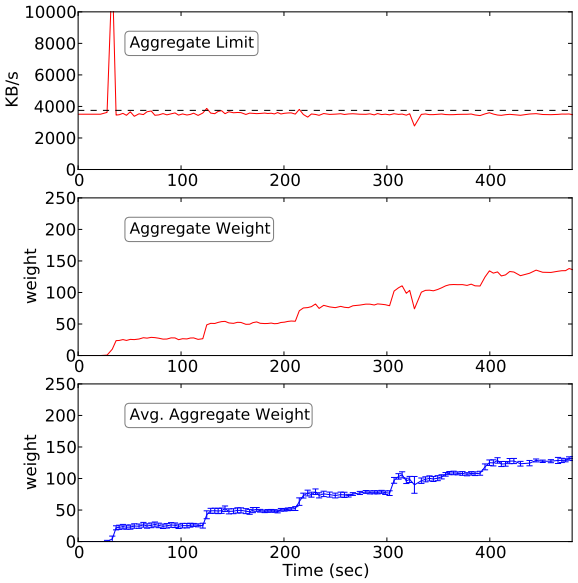


Figure 12: Enforcing a 30-Mbps global limit across 182 limiters, but using a centralized aggregation protocol.

served during the experiment. Since the centralized protocol uses order $O(n)$ links compared to gossip's $O(n^2)$, it sees a lower level of failed links. Even though there are large spikes of disconnectivity, routing through the circuit breaker messaging overlay during those periods ensures stable, accurate aggregates for both protocols. During the gossip experiment a node was removed from the view at 175 seconds, causing its link to all 181 other limiters to be considered cut.

While space limitations prevent us from showing graphs, we note that DRL is unable to leverage Zookeeper-based circuit breakers on PlanetLab because the messaging overlay becomes the bottleneck for computing the aggregate. Moreover, Zookeeper's quorum-based group membership semantics are more restrictive than Moshe's, which enable DRL to be partition aware.

7.2.3 Limiting

Finally, to demonstrate the ability of DRL to effectively limit traffic in the wide area, we install a global rate limit of 30 Mbps for a traffic-generating slice installed on the same 182 nodes. Here our traffic demand follows the same "stair-step" pattern as before: An off-testbed packet sink opens connections to 38 limiters every 80 seconds.

Figure 11 shows three graphs: the aggregate limit, the true aggregate weight (the demand aggregate, computed via post processing), and the average aggregate weight observed by each limiter (error bars show one standard deviation). Each value is averaged over the last two seconds; c.f. Figure 10, which is instantaneous. Despite the variations in measured demand—resulting in variation in average weight shown in the bottom graph—the computed aggregate weight remains quite stable, resulting in an accurate and responsive limit as shown in the top graph.

Figure 12 shows the same experiment using centralized aggregation. While there are slight differences—notably the computed aggregate weight is somewhat more stable—the resulting limiting performance is not markedly better. Here a tightly-coupled gossip-based protocol can provide similar performance to a centralized design while providing the scaling and robustness properties of decentralization.

8 Conclusion

This paper introduces the circuit breaker membership abstraction to ensures that the group members in a tightly-coupled distributed system can efficiently communicate with one another despite transient connectivity failures. Its design combines traditional view-based membership techniques with a reliable messaging overlay. This approach enabled a tightly-coupled service, DRL, to provide accurate global rate limits during the myriad network cuts found in relatively hostile environments like PlanetLab and the Internet in general.

Acknowledgments

The authors are indebted to Marc Fuiczynski, Larry Peterson, Sapan Bhatia, and S. Çağlar Onur for their as-

sistance in deploying our DRL prototype on PlanetLab, and Barath Raghavan, Keith Marzullo, and our shepherd, Mike Spreitzer, for their feedback on earlier drafts of this manuscript. This work was supported in part by the National Science Foundation through grant CNS-0627167.

References

[1] AMAZON. Elastic compute cloud. `http://aws.amazon.com/ec2`, 2009.

[2] ANDERSEN, D. G., BALAKRISHNAN, H., KAASHOEK, M. F., AND MORRIS, R. Resilient overlay networks. In *SOSP* (2001).

[3] BABAOGLU, O., DAVOLI, R., AND MONTRESOR, A. Group communication in partitionable systems: Specification and algorithms. *IEEE Trans. Softw. Eng. 27* (April 2001), 308–336.

[4] BEHL, J., DISTLER, T., AND KAPITZA, R. DQMP: A Decentralized Protocol to Enforce Global Quotas in Cloud Environments. In *SSS* (2012).

[5] BIRMAN, K. P. Replication and fault-tolerance in the ISIS system. In *Proceedings of SOSP* (1985), pp. 79–86.

[6] BURROWS, M. The chubby lock service for loosely-coupled distributed systems. In *Proceedings of USENIX OSDI* (Nov. 2006).

[7] CAO, J., WANG, G., AND CHAN, K. C. C. A fault-tolerant group communication protocol in large scale and highly dynamic mobile next-generation networks. *IEEE Trans. Comput. 56*, 1 (2007).

[8] CHOCKLER, G. V., KEIDAR, I., AND VITENBERG, R. Group Communication Specifications: A Comprehensive Study. *ACM Computing Surveys 33*, 4 (2001).

[9] DEMERS, A., GREENE, D., HAUSER, C., IRISH, W., LARSON, J., SHENKER, S., STURGIS, H., SWINEHART, D., AND TERRY, D. Epidemic algorithms for replicated database maintenance. In *PODC* (1987), pp. 1–12.

[10] DILMAN, M., AND RAZ, D. Efficient reactive monitoring. In *Proceedings of IEEE INFOCOM* (2001).

[11] DUNAGAN, J., HARVEY, N. J. A., JONES, M. B., KOSTIĆ, D., THEIMER, M., AND WOLMAN, A. FUSE: lightweight guaranteed distributed failure notification. In *OSDI* (2004).

[12] GOLDING, R. A. A weak-consistency architecture for distributed information services. *Computing Systems 5*, 4 (1992), 379–405.

[13] HINCHCLIFFE, D. 2007: The year enterprises open thier SOAs to the Internet? *Enterprise Web 2.0* (Jan. 2007).

[14] HUNT, P., KONAR, M., JUNQUEIRA, F. P., AND REED, B. Zookeeper: Wait-free coordination for internet-scale systems. In *USENIX ATC* (2010).

[15] KARMON, K., LISS, L., AND SCHUSTER, A. GWiQ-P: An Efficient Decentralized Grid-Wide Quota Enforcement Protocol. *ACM SIGOPS Operating Systems Review 42*, 1 (2008).

[16] KEIDAR, I., SUSSMAN, J., MARZULLO, K., AND DOLEV, D. Moshe: A group membership service for wans. *ACM Trans. Comput. Syst. 20*, 3 (2002), 191–238.

[17] KEMPE, D., DOBRA, A., AND GEHRKE, J. Gossip-based computation of aggregate information. In *IEEE FOCS* (2003).

[18] KRONENBERG, N. P., LEVY, H. M., AND STRECKER, W. D. Vaxclusters (extended abstract): a closely-coupled distributed system. *ACM Transactions on Computer Systems 4*, 2 (1986).

[19] LAMPORT, L. The part-time parliament. *ACM Transactions on Computer Systems 16*, 2 (1998), 133–169.

[20] LISKOV, B., GHEMAWAT, S., GRUBER, R., JOHNSON, P., AND SHRIRA, L. Replication in the harp file system. In *SOSP* (1991).

[21] MADDEN, S., FRANKLIN, M., HELLERSTEIN, J., AND HONG, W. TAG: a tiny aggregation service for ad-hoc sensor networks. In *OSDI* (2002).

[22] NATH, S., GIBBONS, P., SESHAN, S., AND ANDERSON, Z. R. Synopsis diffusion for robust aggregation in sensor networks. In *SenSys* (November 2004).

[23] RAGHAVAN, B., VISHWANATH, K., RAMABHADRAN, S., YOCUM, K., AND SNOEREN, A. C. Cloud control with distributed rate limiting. In *ACM SIGCOMM* (Aug. 2007).

[24] REED, B., AND JUNQUEIRA, F. P. A simple totally ordered broadcast protocol. In *Workshop on Large-Scale Distributed Systems and Middleware* (2008), pp. 1–6.

[25] SHAVIT, N., AND ZEMACH, A. Diffracting trees. *ACM Transactions on Computer Systems 14*, 4 (1996).

[26] STRIBLING, J. All-pairs-pings for PlanetLab. `http://pdos.csail.mit.edu/~strib/pl_app/`, 2005.

[27] WATTENHOFER, R., AND WIDMAYER, P. An inherent bottleneck in distributed counting. In *PODC* (1997).

[28] YALAGANDULA, P., AND DAHLIN, M. A scalable distributed information management system. In *ACM SIGCOMM* (September 2004).

[29] ZHANG, M., ZHANG, C., PAI, V., PETERSON, L., AND WANG, R. PlanetSeer: Internet path failure monitoring and characterization in wide-area services. In *OSDI* (2004).