AS-cast: Lock Down the Traffic of Decentralized Content Indexing at the Edge

Anonymous author
Anonymous affiliation

Anonymous author
Anonymous affiliation

Anonymous author
Anonymous affiliation

Abstract

Although the holy grail to store and manipulate data in Edge infrastructures is yet to be found, state-of-the-art approaches demonstrated the relevance of replication strategies that bring content closer to consumers: The latter enjoy better response time while the volume of data passing through the network decreases overall. Unfortunately, locating the closest replica of a specific content requires indexing *every live* replica along with its *location*. Relying on remote services enters in contradiction with the properties of Edge infrastructures as locating replicas may effectively take more time than actually downloading content. At the opposite, maintaining such an index at every node would prove overly costly in terms of memory and traffic, especially since nodes can create and destroy replicas at any time.

In this paper, we abstract content indexing as distributed partitioning: every node only indexes its closest replica, and connected nodes with a similar index compose a partition. Our decentralized implementation AS-cast is (i) efficient, for it uses partitions to lock down the traffic generated by its operations to relevant nodes, yet it (ii) guarantees that every node eventually acknowledges its partition despite concurrent operations. Our complexity analysis supported by simulations shows that AS-cast scales well in terms of generated traffic and termination time. As such, AS-cast can constitute a new building block for geo-distributed services.

2012 ACM Subject Classification Networks → Network protocols

Keywords and phrases Edge, decentralized content indexing, scoped broadcast, logical partitioning

Digital Object Identifier 10.4230/LIPIcs...

1 Introduction

In recent years, the data storage paradigm shifted from centralized in the cloud to distributed at the edges of the network. This change aims at keeping the data close to (i) its producers since data may be too expensive or sensitive to be transmitted through the network; and (ii) its consumers so data may quickly and efficiently reach them [12, 15, 35]. To favour this transition, new designs for data management across Edge infrastructures have been investigated [8, 9, 16, 18]. They enable strategies to confine traffic by writing data locally and replicating content according to effective needs. However, locating content remains challenging. Retrieving a content location may actually take more time than retrieving the content itself. Indeed, these systems, when not using a centralized index hosted in a Cloud, rely on distributed hash tables (DHT) [29] spread across the different nodes composing the infrastructure. When a client wants to access specific content, it requests a remote node to provide at least one node identity to retrieve this content from. After it retrieved the content, the client can create another replica to improve the performance of future accesses, but it must contact the content indexing service again to notify of such a change.

These approaches are in contradiction with the objectives of Edge infrastructures that aim at reducing the impact of latency as well as the volume of data passing through the network. First, accessing a remote node to request content location(s) raises hot spots and availability issues. But most importantly, it results in additional delays [3, 11] that occur even before the actual download started. Second, the client gets a list of content locations at the discretion of content indexing services. Without information about these locations, it often ends up downloading from multiple replica hosts, yet only keeping the fastest answer. In turn, clients either waste network resources, or face lower response time.

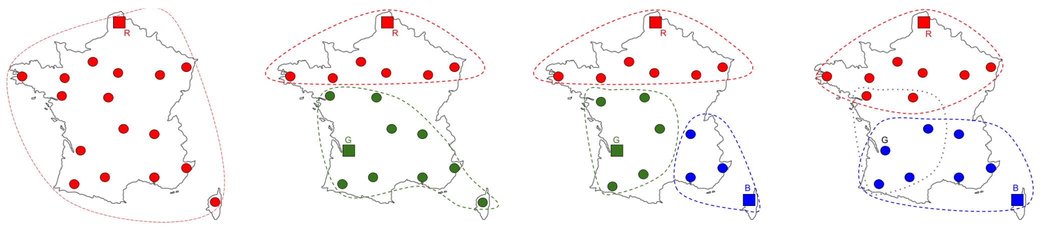
To address the aforementioned limitations, every node that might request or replicate content must also host its own content indexing service in a fully decentralized fashion [25]. At any time, it can immediately locate the closest replica of specific content. A naive approach would be that every node indexes and ranks *every live* replica along with its *location* information. When creating or destroying a replica, a node would notify all other nodes by efficiently broadcasting its operation [6, 17, 31]. Unfortunately, this also contradicts Edge infrastructure objectives, for such a protocol does not confine the traffic generated to maintain its indexes. A node may acknowledge the existence of replicas at the other side of the network while there already exists a replica next to it.

Instead, a node creating a replica should notify all and only nodes that have no closer replica in the system. This would create interconnected sets of nodes, or *partitions*, gathered around a *source* being their respective replica. A node deleting its replica should notify all members of its partition so they can identify and rally their new closest partition. A periodic advertisement protocol already provides both operations for routing purposes [19]. However, its functioning requires (i) to generate traffic even when the system is quiescent, and (ii) to finely tune physical-time-based intervals that depend on network topology parameters such as network diameter.

The contribution of this paper is threefold:

- We highlight the properties that guarantee decentralized consistent partitioning in dynamic infrastructures. We demonstrate that concurrent creation and removal of partitions may impair the propagation of control information crucial for consistent partitioning. Yet, nodes are able to purge stale information forever using only neighbor-to-neighbor communication, hence leaving room for up-to-date information propagation, and eventually consistent partitioning.
- We provide an implementation entitled AS-cast that uses aforementioned principles to adapt its partitioning to creations and deletions of partitions even in dynamic systems where nodes join, leave, and crash at any time. AS-cast's efficiency relies on a communication primitive called scoped-broadcast that enables epidemic dissemination of messages as long as receiving nodes verify an application-dependant predicate.
- We evaluate AS-cast with simulations. Our experiments empirically support our complexity analysis and the scalability of AS-cast. Using scoped broadcast, AS-cast quickly disseminate messages to a subset of relevant nodes. The higher the number of partitions, the smaller AS-cast's overhead. Most importantly, AS-cast does not generate traffic in quiescent systems. Our experiments also highlight the relevance of AS-cast in autonomous systems. AS-cast would allow to lock down the traffic generated by content indexing, even in contexts where *physical* partitioning may occur. As such, AS-cast can constitute a new building block for geo-distributed services.

The rest of this paper is organized as follows. Section 2 illustrates the motivation and problem behind our proposal. Section 3 describes dynamic consistent partitioning and its implementation along with an analysis of its complexity. Section 4 presents our evaluations. Section 5 reviews related work. Finally, Section 6 concludes and discusses future work.



(a) Node R efficiently advertises its content by epidemic propagation. Every node requests R if needed.

(b) Node G creates a second replica splitting the red set in two. Nodes request their closest replica host.

(c) Node B creates another replica. Node B needs to notify only a small subset of nodes.

(d) Node G destroys its replica. Nodes that belonged to its partition must find the closest partition they are in.

Figure 1 Partitions grow and shrink depending on creations and removals of replicas.

2 Motivation and problem

Numerous studies addressed content indexing in geo-distributed infrastructures ranging from centralized services to fully decentralized approaches [23]. We advocate for the latter where nodes host the service themselves so they do not need to request remote – possibly far away third-party – entities to retrieve their content location. In this paper, we tackle the content indexing issue as a dynamic logical partitioning problem. This section motivates our positioning and explains the shortcomings of existing implementations.

Dissemination: Figure 1 depicts an infrastructure comprising 17 interconnected nodes spread across France. In Figure 1a, a single Node R hosts the content, so every other node downloads from this node when it needs it. In that regard, Node R only needs to disseminate a message notifying all other nodes of its new content. Node R can use uniform reliable broadcast [17] and epidemic propagation [13] to guarantee that (i) every node eventually knows the content location (ii) by efficiently using neighbor-to-neighbor communication.

Location: Then, in Figure 1b, another Node G creates a replica of this content. Similarly to Node R, it notifies other nodes of this new replica. However, to avoid that north nodes request its replica, and south nodes request the northern one, nodes hosting a replica must add *location information* along with their notifications. As a consequence, every node eventually knows every replica location and can download from its closest host. Red nodes would request Node R while green nodes would request Node G.

Scoped broadcast: Then, in Figure 1c, another Node B creates a replica of this content. Similarly to Node R and Node G, Node B can notify other nodes of this new replica. However, the set of nodes that could actually use this new replica is actually much smaller than the network size. Uniform reliable broadcast is not designed for such a context and would generate a lot of unnecessary traffic. Instead, nodes need a communication primitive that propagates notifications within a scope by evaluating a predicate, starting at its broadcaster (the *source*). In other terms, nodes propagate notifications as long as they consider them useful based on location information they carry. We call such a primitive *scoped broadcast* (see Section 3.1), for messages transitively reach a subset of interconnected nodes (the *scope*). Using this primitive, nodes can lock down the traffic of content indexing to relevant nodes.

Logical partitioning: Every node ends up with at least one known replica that is its closest. The set of interconnected nodes with the same closest replica is a *partition*. Every node belongs to one, and only one, partition (see Section 3.2).

Dynamic partitioning and adaptive scoped broadcast: Finally, in Figure 1d, Node G destroys its replica. Every node that belonged to its green partition must choose another partition to be in. While it makes sense for Node G to scoped broadcast its removal, Node B and Node R cannot afford to continuously advertise their replica to fill the gap left open by Node G. Instead, we want to trigger once again scoped broadcast at bordering nodes of red and blue partitions. The scope of scoped broadcast changes over receipts by nodes. Dynamic partitioning raises additional challenges related to concurrent operations where removed partitions would block the propagation of other partitions (see Section 3.3).

Next Section details the properties of scoped broadcast and dynamic partitioning in dynamic networks, and provides an implementation called AS-cast along with its complexity.

3 Adaptive scoped broadcast

To assign and maintain nodes to their best partition according to replica creations and removals, as well as dynamic infrastructure changes, we designed and implemented AS-cast. AS-cast stands for Adaptive Scoped broadcast. It relies on a primitive that allows a node to broadcast a message within a limited scope. We first use this primitive to guarantee consistent partitioning when a node can only create a new replica within the system. We highlight the issue when a node can also destroy a replica, and provide a second algorithm that handles replica removals as well as dynamic changes of the infrastructure. This section describes the communication primitive called scoped broadcast, then discusses the properties that guarantee consistent partitioning, and finally details our implementation AS-cast.

3.1 Scoped broadcast

In this paper, we consider Edge infrastructures with heterogeneous nodes interconnected by communication links. Nodes involved in the management of content may crash but are not byzantine. Nodes can reliably communicate through asynchronous message passing to other known nodes called neighbors. We define scoped broadcast as a communication primitive that propagates a message around its broadcaster within an application-dependant scope.

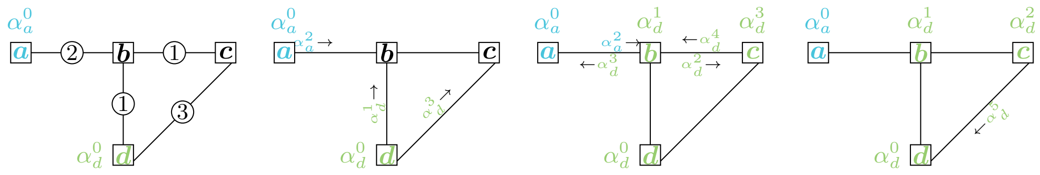
▶ **Definition 1** (Edge infrastructure). *An Edge infrastructure is a connected graph $G(V, E)$ of vertices V and bidirectional edges $E \subseteq V \times V$. A path π_{ij} from Node i to Node j is a sequence of vertices $[i, k_1, k_2, \dots k_n, j]$ with $\forall m : 0 \leq m \leq n, \langle \pi_{ij}[m], \pi_{ij}[m+1] \rangle \in E$.*

▶ **Definition 2** (Scoped broadcast (S-cast)). *When Node x scoped broadcasts $b_x(m)$ a message m, every correct node y within a scope receives $r_y(m)$ and delivers it $d_y(m)$. The scope depends on the state σ of each node, the metadata μ piggybacked by each message, and a predicate ϕ verified from node to node: $b_x(m) \wedge r_y(m) \implies \exists \pi_{xy} : \forall z \in \pi_{xy}, \phi(\mu_z, \sigma_z)$.*

This definition encompasses more specific definitions of related work [22, 28, 37]. It also highlights that epidemic propagation and scoped broadcast have well-aligned preoccupations. More precisely, it underlines the transitive relevance of messages, thus a node can stop forwarding messages as soon as its predicate becomes unverified. We use S-cast to efficiently modify the state of each node depending on the partitions that exist in the system.

3.2 Consistent partitioning

At any time, a node can decide to become a *source*, hence creating a new partition in the system by executing an `Add` operation. This partition includes at least its source plus neighboring nodes that estimate they are closer to this source than any other one. Such a distance (or *weight*) is application-dependant: in the context of maintaining distributed indexes, this would be about link latency that nodes could monitor by aggregating `pings`.



(a) Both a and d become sources. $w_{ab} = 2$; $w_{bc} = w_{bd} = 1$; $w_{cd} = 3$.

(b) Messages transit through links and carry increasing weights.

(c) b and c receive, deliver, and forward α_d^1 and α_d^3 respectively.

(d) a and b discarded their received messages. c still improved with α_d^2.

Figure 2 Efficient consistent partitioning using S-cast. Partition P_a includes a while Partition P_d includes b, c, and d. Node c and Node d never acknowledge the existence of Source a, for Node b stops the propagation of the latter's notifications.

▶ **Definition 3** (Partitioning). *Let $S \subseteq V$ be the set of sources, and P_s be the partition including at least Node s, each node belongs to at most one partition $\forall p, q \in V, \forall s_1, s_2 \in S : p \in P_{s_1} \wedge q \in P_{s_2} \implies p \neq q \vee s_1 = s_2$, and there exists at least one path π_{ps} of nodes that belong to this partition $\forall q \in \pi_{ps} : q \in P_s$.*

Definition 2 and Definition 3 share the transitive relevance of node states. However, we further constrain the partitioning in order to guarantee the existence of exactly one consistent partitioning that nodes eventually converge to.

▶ **Definition 4** (Consistent partitioning (CP)). *Let $W_{xy} = W_{yx}$ be the positive symmetric weight between x and y, Π_{xz} be the shortest path from x to z the weight of which $|\Pi_{xz}|$ is lower than any other path weight, with $|\Pi_{xx}|$ being the greatest lower bound of x, the only consistent partitioning $\mathcal{P}$ is a set of partitions $P_{s \in S}$ such that each node belongs to a logical partition comprising its closest source: $\forall p \in P_{s_1} : \nexists P_{s_2}$ such that $|\Pi_{s_2 p}| < |\Pi_{s_1 p}|$.*

Unfortunately, nodes do not share a common global knowledge of the network state. For nodes to reach consistent partitioning together, a Node s adding a partition must send a notification α_s to every node that is closer to it than any other source. Since epidemic dissemination and scoped broadcast have well-aligned preoccupations, we assume implementations relying on message forwarding from neighbor-to-neighbor.

▶ **Theorem 5** (Eventual Forwarding of Best $(EFB) \implies CP)$. *Assuming reliable communication links where a correct node q eventually receives the message m sent to it by a node p $(s_{pq}(m) \implies r_q(m))$, nodes eventually reach consistent partitioning if each node eventually forwards $(f_p(m) \implies \forall \langle p, q \rangle \in E : s_{pq}(m))$ its best known partition.*

Proof. When a node s_1 becomes a source, it belongs to its own partition, for there exists no better partition than its own: $\forall p \in V : |\Pi_{s_1 s_1}| < |\Pi_{s_1 p}|$. It delivers, hence forwards such an α message to its neighbors. Since communication links are reliable, neighboring nodes eventually receive such a notification $\forall \langle s_1, q \rangle \in E, s_1 \in S \iff \Diamond\, r_q(\alpha_{s_1}^{w_{s_1 q}})$. Most importantly, whatever the order of received messages, every node q' in this neighborhood – such that there exists no better partition s_2 than the received one – delivers and forwards it: $\forall s_2 \in S : |\Pi_{q' s_1}| < |\Pi_{q' s_2}| \implies \Diamond\, d_{q'}(\alpha_{s_1}^{w_{s_1 q'}})$. By transitivity, the message originating from s_1 reaches all such nodes through their shortest paths: $\forall q'' \in V, s_1, s_2 \in S : |\Pi_{q'' s_1}| < |\Pi_{q'' s_2}| \implies \Diamond\, d_{q''}(\alpha_{s_1}^{|\Pi_{s_1 q''}|})$. Since there exists only one best sum of weights per node that can never be retracted, nodes eventually reach consistent partitioning. ◀

Algorithm 1 Adding a partition by Node p.

```
1  $O_p$, $W_p$                                  // set of neighbors and weights
2  $s \leftarrow \varnothing$                    // best source of partition ($\sigma$)
3  $d \leftarrow \infty$                         // smallest distance to $s$ ($\sigma$ and $\mu$)
4  func Add ( )                                  // generates notification message: $\alpha_p^0$
5      receiveAdd($\varnothing$, $p$, 0)         // $b_p(\alpha_p^0)$
6  func receiveAdd($q$, $s'$, $d'$)              // $r_p(\alpha_{s'}^{d'})$ from $q$
7      if $d' < d$ then                          // distance predicate ($\phi$)
8          $s \leftarrow s'$                     // $d_p(\alpha_{s'}^{d'})$
9          $d \leftarrow d'$                     // update local state ($\sigma$)
10         foreach $n \in O_p \setminus q$ do    // forward to neighbors $f_p(\alpha_{s'}^{d'})$
11             send$_n(s', d' + W_{pn})$         // with updated metadata ($\mu$) $s_{pn}(\alpha_{s'}^{d'+w_{pn}})$
```

Algorithm 1 shows the instructions that implement consistent partitioning when (i) weights are scalar values, (ii) nodes only add new partitions to the system, (iii) and nodes never crash nor leave the system. Figure 2 illustrates its behavior on a system comprising 4 nodes a, b, c, and d. Node a and Node d become the sources of their partition. They S-cast a notification add message: α_a^0 and α_d^0. They initialize their own state with the lowest possible bound 0 (see Line 5), and send a message to each of their neighbors by accumulating the corresponding edge weight (see Line 11). In Figure 2c, Node b receives α_d^1. Since it improves its own partition distance, it keeps it and forwards it to its neighbors. In Figure 2d, Node b discards α_a^2, for it does not improve its partition distance. Nodes c and d will never acknowledge that Source a exists. Ultimately, nodes discard last transiting messages. Despite the obvious lack of traffic optimization, the system reaches consistent partitioning.

While only adding logical partitions to the distributed system is straightforward, removing them introduces additional complexity caused by concurrent operations.

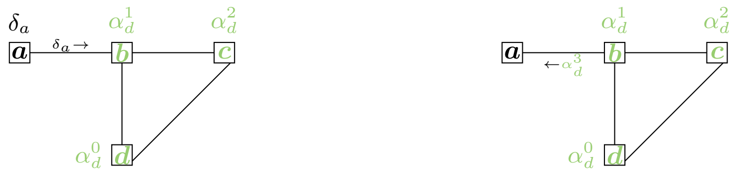
3.3 Dynamic consistent partitioning

At any time, a source can revoke its self-appointed status of source by executing a `Del` operation, hence deleting its partition from the system. All nodes that belong to this partition must eventually choose another partition to belong to. In Figure 3, two partitions exist initially: P_a and P_d that respectively include $\{a\}$, and $\{b, c, d\}$. In Figure 3a, Node a deletes its partition. It notifies all neighboring nodes – here only Node b – that may belong to its partition using S-cast. Upon receipt, Node b discards the delete notification δ_a since δ_a does not target the former's partition P_d. Node b sends its own best partition α_d^3 that may be the best for Node a. Eventually, every node belongs to Partition P_d. In this scenario, they reach consistent partitioning.

Delete operations add a new notion of order between events, and most importantly between message deliveries. Delete operations implicitly state that all preceding events become obsolete, and that all messages originating from these preceding events convey stale control information.

▶ **Definition 6** (Happens-before $\rightarrow$ [27]). *The transitive, irreflexive, and antisymmetric happens-before relationship defines a strict partial order between events. Two messages are concurrent if none happens before the other.*

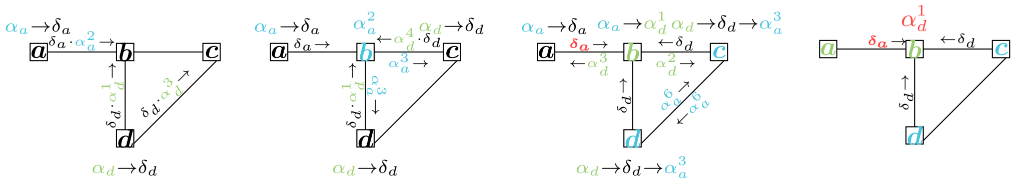
▶ **Definition 7** (Message staleness). *A message m conveys stale control information if the node that broadcast m later broadcast another message m': $S(m) = \exists m' : b_p(m) \rightarrow b_p(m')$.*



(a) Node a deletes its partition. It notifies all nodes that belong to its partition using S-cast.

(b) δs stop as soon as they encounter another partition. Node b answers with its partition.

Figure 3 Efficient removal of a partition using S-cast. a eventually knows that it belong to P_d.



(a) Both Node a and Node d add then delete their partition. For the recall, $w_{ab} = 2$; $w_{bc} = w_{bd} = 1$; $w_{cd} = 3$.

(b) b delivers and forwards the α message about P_a while Node c delivers and forwards both α and δ messages about P_d.

(c) b already blocks the propagation of δ_a despite it propagated α_a before; and c does not deliver α_d, for it already deleted it before.

(d) Nodes discard remaining messages. Since b cannot deliver δ_a, c and d remain in the deleted P_a.

Figure 4 Naive propagation of α and δ messages is insufficient to guarantee consistent partitioning.

Unfortunately, stale control information as stated in Definition 7 may impair the propagation of both (i) notifications about actual sources, and (ii) notifications about deleted partitions. Figure 4 highlights such consistency issues with dynamic partitions, even in contexts where nodes have FIFO links, i.e., where nodes receive the messages in the order of their sending ($s_{pq}(m) \rightarrow s_{pq}(m') \implies r_q(m) \rightarrow r_q(m')$). Both a and d add then delete their partition concurrently. Node c receives, delivers, and forwards α_d^3 followed by δ_d. In the meantime, b receives, delivers, and forwards α_a^2. In Figure 4c, both c and d deliver the message about Partition P_a, for they did not have any known partition at receipt time. On the contrary, b delivers α_d^1, for it improves its distance to a known source. Node b then blocks a's removal notification δ_a. It never reaches c nor d. Also, c does not deliver α_d and δ_d since it already delivered it. The system converges to an inconsistent state where some nodes assume they belong to a partition while it does not exist anymore.

▶ **Theorem 8** (EFB$^+$: EFB $\wedge$ (Purge $\rightarrow$ EFB) $\implies$ <u>D</u>ynamic <u>c</u>onsistent <u>p</u>artitioning (DCP))**.** *When a node can* **`Add`** *and* **`Delete`** *its partition, to ensure consistent partitioning, eventual forwarding of best (see Theorem 5) additionally requires (i) eventual purging of stale messages ($S(m) \implies (d_q(m) \implies \lozenge\, d_q(m) \rightarrow d_q(m'))$); (ii) then the eventual forwarding of best such that m' ends up originating from the best up-to-date source for q.*

Proof. Assuming EFB as default propagation behavior where nodes deliver and forward the best sum of weights they received. We must prove that despite stale messages, the last delivery of every node is the message that followed the shortest path to its closest source.

EFB without Purge: From Figure 4d, a node may receive messages in such an order ($d_b(\alpha_d^1) \rightarrow r_b(\delta_a) \rightarrow d_b(\delta_d)$) that it does not forward messages required by other nodes (δ_a). The last delivery of some nodes (c and d) is wrong (α_a^3 instead of δ_a). Nodes never reach consistent partitioning.

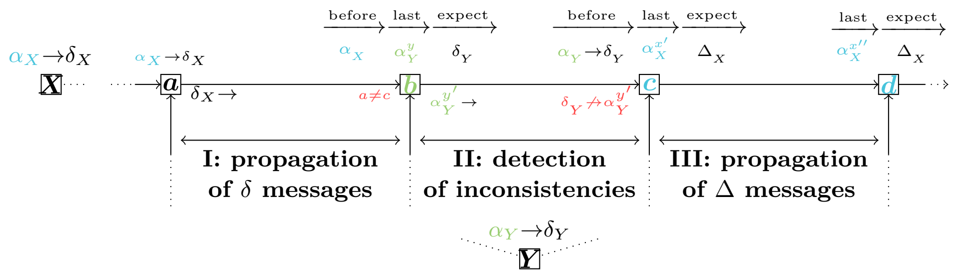

before
last
expect
X
a
b
c
d
Y
I: propagation of δ messages
II: detection of inconsistencies
III: propagation of Δ messages
a≠c

Figure 5 Stale α messages may stop δ messages from reaching nodes with the corresponding partition. Consistent partitioning requires that all nodes propagate δ messages as often as possible (I); detect possible inconsistencies when parents' α messages contradict history or state (II); notify nodes of possible inconsistencies (III).

Purge not followed by EFB: Assuming that every node removes control information as soon as it becomes stale ($S(\alpha_a) \implies d_b(\alpha_a^x) \rightarrow d_b(\delta_a)$) and cannot deliver stale messages anymore ($S(\alpha_a^x) \implies d_c(m) \rightarrow \ldots \rightarrow d_c(m' \notin \alpha_a)$), staleness does not impair up-to-date messages propagation. However, even in such settings, depending on receipt order ($d_b(\alpha_a^x) \rightarrow r_b(\alpha_c^{y:y>x}) \rightarrow d_b(\delta_b)$), a node may never receive again, hence deliver and forward a message that was received and discarded before (α_c^y). When such a message contains its best up-to-date partition, the node never ends up in its best partition. Therefore, nodes do not reach consistent partitioning.

Purge then EFB: Assuming the eventual and definitive removal of stale messages at each node ($S(\alpha_a) \implies (d_b(\alpha_a) \implies \Diamond\, d_b(\alpha_a) \rightarrow d_b(m') \not\rightarrow d_b(\alpha_a))$); and assuming that every removal eventually triggers the forwarding of best delivered messages ($d_c(\delta_x) \implies \Diamond\, f_d(\alpha_y)$), nodes may deliver other stale messages that will be eventually removed to never be delivered again. In the worst case, all nodes deliver and remove all stale messages to ultimately deliver and forward only up-to-date messages ($\Diamond\, d_c(m') \implies \neg S(m')$). As in Theorem 5, they eventually deliver and forward their best up-to-date message.
Using EFB$^+$, nodes reach consistent partitioning together even under dynamic partitioning settings where nodes can `Add` and `Delete` their partition. ◀

The main challenge consists in implementing a purging mechanism that ensures the *eventual and definitive* removal of stale control information. One could guarantee consistent partitioning by always propagating the δ messages corresponding to the α messages it propagated before. In Figure 4, it means that as soon as Node b forwards α_a^2, it assumes that its neighbors Node c and Node d may need the notification of removal δ_a if it exists. However, such a solution also implies that nodes generate traffic not only related to their current partition, but also related to partitions they belonged to in the past. This would prove overly expensive in dynamic systems where nodes join and leave the system, create and delete partitions, at any time. Instead, we propose to use the delivery order at each node to detect possible inconsistencies and solve them.

▶ **Theorem 9** (Three-phase purge). *Three phases are sufficient to eventually purge the system from stale control information: (i) propagation of delete notifications, (ii) detection of possibly blocking conditions, and (iii) propagation of delete notifications that were possibly blocked.*

Proof. We must prove that every node that delivered a stale α_X eventually (i) delivers a better α_Y, or (ii) delivers a removal notification of α_X. Figure 5 summarizes the issue of purging in scenarios involving concurrent operations:

If every node (such as a) starting from the source delivers and forwards a removal notification δ_X when their last delivery is α_X, then every such node eventually delivers the removal notification δ_X except nodes (such as b) that delivered α_X from a node that delivered a message from another partition α_Y since then.

These exceptions eventually receive and deliver α_Y since α_Y is better than α_X through this path, except if they already received and delivered the removal notification of P_Y (such as c). These nodes may never receive hence deliver δ_X, and may never receive hence deliver a better message than α_X. They need an additional mechanism to eventually purge α_X that cannot rely on the eventual purging of P_Y at nodes like b, to avoid deadlocks.

Assuming that every node keeps an history of its past deliveries, nodes (such as c) can detect the inconsistency, since they receive from their parent an already deleted partition P_Y. This notification means that the parent discards any δ_Z with $\alpha_Z^z < \alpha_Y^y$, and most importantly, if δ_X exists, it discards it. To ensure the purge of stale messages, a detecting node must assume the worst case that such δ_X exists, and send another kind of message, noted Δ, that notify the possible removal of P_X.

A node (such as d) whose last delivery is α_X, but whose parent is neither inconsistent (like b) nor receiving δ_X (like a), eventually receives Δ from a detecting node, either directly or transitively, for such a child node (d) trusts the possible removal of P_X by delivering and forwarding such Δ. Δ suffers from identical blocking conditions (between b and c) than δ, leading to the same solutions of detection and propagation of Δ. Eventually, every node whose last delivery is α_X receives and delivers either a better α_Z, or δ_X, or Δ_X. ◀

3.4 Implementation and complexity

Algorithm 2 provides the few instructions of AS-cast that implement three-phase purge and eventual forwarding of best to enable dynamic consistent partitioning. For the sake of simplicity, it does not handle multiple sessions and associated optimisation such as data structure sharing and message packing.

As stated in Theorem 8, every node must purge stale messages. To identify staleness, each node maintains a vector of versions that associates the respective known local counter of each known source, or has-been source. Its size increases linearly with the number of sources that the node has ever known, which is the number of nodes in the system $\mathcal{O}(V)$ in the worst case. Nevertheless, following the principles of scoped broadcast, we expect that every node only acknowledges a small subset of sources. Using such a vector, every message m carries its source s along with its version c that we subscript $m_{s,c}$. Each node delivers and forwards (i) only newest messages (see Line 10) that (ii) improve their best known partition (see Line 17). This vector of versions constitutes a summary of local deliveries. It allows a node to detect inconsistencies in message delivery: the parent delivered a stale message (see Line 10), or the parent delivered a worse message (see Line 11).

A node (c) detecting an inconsistency disseminates a Δ message. Such a Δ must not modify any vector of versions since the partition of the corresponding α may still exist. Therefore, it additionally includes and maintains a path π initialized to α's one. Such a $\Delta_{s,c}^{\pi}$ transitively tracks and purges α messages that were forwarded by the detecting node. The path π also enables a quick stop of looping messages and in turns ensures termination. However, carrying paths in messages is costly. In the worst case, a message piggybacks the identity of all nodes in the system. Fortunately, message paths and AS-cast synergyze well with each other, for paths tend to be smaller as the number of sources in the system increases. Contrarily to α and Δ messages, $\delta_{s,c}$ messages do not carry a path, and are therefore more efficient. They (i) solely rely on versions to operate, and (ii) use the whole dissemination

Algorithm 2 AS-cast at Node p in dynamic networks.

```
1  $O_p$, $W_p$                                   // set of neighbors and weights
2  $A^{d,\pi}_{s,c} \leftarrow \alpha^{\infty,\varnothing}_{\infty,0}$    // best $\alpha$ so far
3  $V \leftarrow \varnothing$;  $V[p] \leftarrow 0$                    // vector of versions

4  func Add ( )
5      receiveAdd($p$, $\alpha^{0,\varnothing}_{p,V[p]+1}$)

6  func Del ( )
7      receiveDel($p$, $\delta_{p,V[p]+1}$)

8  func receiveAdd($q$, $\alpha^{d',\pi'}_{s',c'}$)
9      if  $q = \pi[|\pi|-1]$ and // from parent?
10         ($c' < V[s']$  or // history or state?
11         ($\alpha^{d',\pi'}_{s',c'} > A^{d,\pi}_{s,c}$ and $p \notin \pi'$)) then
12         receiveDel($q$, $\delta^{\pi}_{s,c}$) // (II) send $\Delta$
13     else if $A^{d,\pi}_{s,c} < \alpha^{d',\pi}_{s',c'}$ and $p \notin \pi'$ then
14         $V[s'] \leftarrow c'$
15         $A^{d,\pi}_{s,c} \leftarrow \alpha^{d',\pi'}_{s',c'}$
16         foreach $n \in O_p$ do
17             send$_n$($\alpha^{d'+W_{pq},\pi' \cup p}_{s',c'}$) // forward

18 func receiveDel($q$, $\delta^{\pi'}_{s',c'}$)
19     if  $p \notin \pi'$  and // looping?
20         (($\delta_{s',c'}$ and $\alpha_{s',c'-1} = A_{s,c}$) or // $\delta$?
21         ($\delta^{\pi'}_{s',c'}$ and $\alpha^{-,\pi'}_{s',c'} = A^{-,\pi}_{s,c}$)) then //$\Delta$?
22         $V[s'] \leftarrow \max(V[s'], c')$
23         $A^{d,\pi}_{s,c} \leftarrow \alpha^{\infty,\varnothing}_{\infty,0}$
24         foreach $n \in O_p$ do
25             if $\delta_{s',c'}$ then  send$_n$($\delta_{s',c'}$)  // (I)
26                          else send$_n$($\delta^{\pi' \cup p}_{s',c'}$) // (III)
27     else if $A^{d,\pi}_{s,c} \neq \alpha^{\infty,\varnothing}_{\infty,0}$ then
28         send$_q$($\alpha^{d+W_{pq},\pi \cup p}_{s,c}$) // compete again

29 func edgeUp($q$) // new link to $q$
30     if $A^{d,\pi}_{s,c} \neq \alpha^{\infty,\varnothing}_{\infty,0}$ then
31         send$_q$($\alpha^{d+W_{pq},\pi \cup p}_{s,c}$) // compete with $q$

32 func edgeDown($q$) // link to $q$ removed
33     if $q = \pi[|\pi|-1]$ then
34         receiveDel($q$, $\delta^{\pi}_{s,c}$) // (III) send $\Delta$
```

graph to propagate in the system, as opposed to Δ that must follow the dissemination tree of the corresponding α.

As stated in Theorem 8, dynamic consistent partitioning not only requires eventual purging of stale messages, but also the retrieval of the best up-to-date partitions. In that regard, both δ and Δ messages have dual use: since they already reach the borders of their partition by design, they trigger a competition when reaching bordering nodes (see Line 28).

Algorithm 2 also enables dynamic consistent partitioning in dynamic networks where nodes can join, leave, or crash at any time. Removing a node is equivalent to removing all its edges, and adding a node is equivalent to add as many edges as necessary. Adding an edge between two nodes may only improve the shortest path of one of these nodes. Therefore, it triggers a competition between the two nodes only. If one or the other node improves, the propagation of α messages operates normally. Removing an edge between two nodes may invalidate the shortest path of one of involved nodes plus downstream nodes. As a side effect, removing an edge may also impair the propagation of a partition delete. To implement this behavior, AS-cast uses its implementation of Δ messages. This prove costly but enables AS-cast even in dynamic networks subject to physical partitioning.

In terms of number of messages, in the average case, a node i chosen uniformly at random among all nodes creates a logical partition. Its messages α_i spread through the network until reaching nodes that belong to another partition closer to them. This splits partitions in half on average. Overall, the a^{th} new partition comprises $\mathcal{O}(\frac{|V|}{2^{\lceil \log_2 a \rceil}})$ nodes. This decreases every new partition until reaching one node per new partition: its source. The average number of messages per node is $\mathcal{O}(\frac{|O|}{2^{\lceil \log_2 a \rceil}})$, where $\overline{|O|}$ is the average number of neighbors per node. Deleting the a^{th} partition generates twice as many messages as creating the a^{th} partition: δ messages travel through the partition, then α messages compete to fill the gap. Overall, the communication complexity shows that AS-cast scales well to the number of partitions. Next Section provides the details of our simulations that assess the proposed implementation.

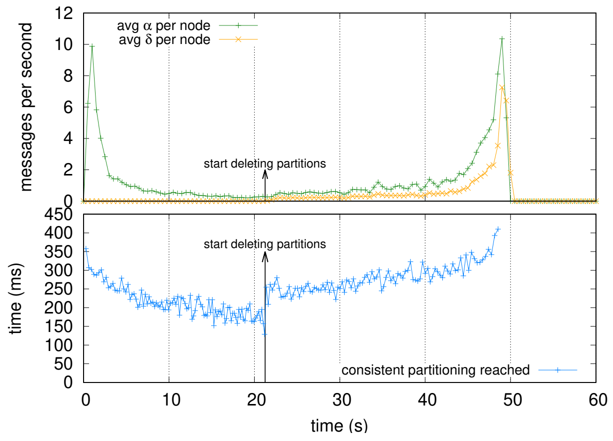


Figure 6 Overhead of dynamic consistent partitioning using AS-cast.

4 Experimentation

This section aims to answer the following questions about AS-cast: What is the overhead of reaching consistent partitioning using AS-cast in large scale networks? Is AS-cast relevant in Edge infrastructures where nodes are geo-distributed into clusters? We performed the experimentation using PeerSim, a simulator written in Java that allows researchers to evaluate and assess distributed algorithms in large scale networks [30]. The code is available on the Github platform at `https://anonymous.4open.science/r/peersim-partition-5592`.

Objective: Confirm the complexity analysis stating that communication and time overheads depend on the current partitioning of the system.

Description: We build a network comprising 10k nodes. First, we chain nodes together, then we add another communication link per node to another random node. Since links are bidirectional, each node has 4 communication links on average. We set the latency of links between 20 and 40 ms at random following a uniform distribution. We set the weight of links between 5 and 15 at random following a uniform distribution. Weights and latency are different, hence the first α received by a node may not be that of its shortest path. It implies more concurrent operations, hence more traffic to reach consistent partitioning.
We evaluate dynamic consistent partitioning. First, we create 100 partitions one at a time: nodes reach consistent partitioning before we create each new partition. Second, we remove every partition one at a time, in the order of their creation, hence starting from the first and oldest partition that had been added.
We focus on the complexity of AS-cast. We measure the average number of messages generated per node per second during the experiment; and the time before reaching consistent partitioning after adding or removing a partition.

Results: Figure 6 shows the results of this experiment. The top part shows the average traffic per node per second divided between α and δ messages. The bottom part shows the time before reaching consistent partitioning.
Figure 6 confirms that AS-cast's overhead depends on the size of partitions. The first partition is the most expensive, for α's must reach every node which takes time and generate more traffic. AS-cast quickly converges towards consistent partitioning in only 350 milliseconds. The last and 100^{th} partition added around 21 seconds is the least expensive. By using scoped broadcast, control information only reaches a small subset of the whole network.
Figure 6 also confirms that AS-cast's delete operations are more expensive than add operations.

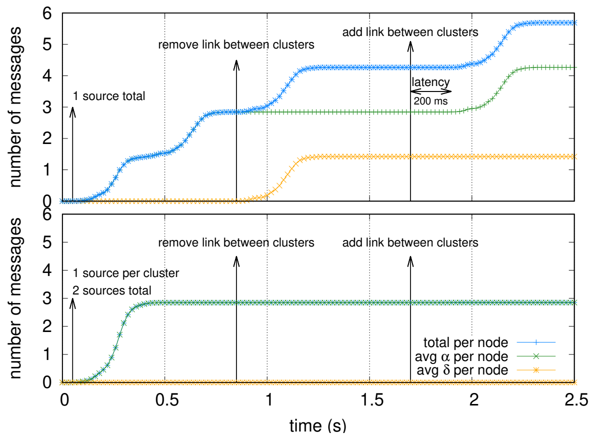


Figure 7 Overhead of AS-cast in the partitioning of 2 clusters connected by a long distance link.

Indeed, the top part of the figure shows that after 21 seconds, when the experiment involves removals, traffic includes both α's and δ's. The latter aims at removing stale information and triggering competition while the former aims at updating shortest paths. As corollary, the convergence time increases, for δ messages must reach the partition borders before sound competitors propagate their partition. It is worth noting that this delete operation involves concurrency: removals still propagate while the competition has started.

Figure 6 shows that the overhead of adding the 1^{st} partition does not correspond to the overhead of deleting this 1^{st} partition. When adding it, messages must reach all nodes while when removing it, messages must reach a small subset of this membership. AS-cast's overhead actually depends on current partitioning as opposed to past partitionings.

Finally, Figure 6 highlights that after 49 seconds, i.e., after the 100 adds and the 100 deletes, nodes do not generate traffic anymore. Being reactive, AS-cast has no overhead when there is no operation in the system. AS-cast's overhead actually depends on its usage.

Objective: Confirm that AS-cast locks down the traffic of decentralized content indexing in the context of dynamic inter-autonomous system communications.

Description: We build a network by duplicating the GÉANT topology [26] – a real infrastructure spanning across Europe – and by connecting these two clusters with a high latency link: 200 ms simulating cross-continental communications such as Europe-America. The experiments comprise $2 \times 271 = 542$ nodes and we derive intra-cluster latency from their nodes' geographic location.

We evaluate the traffic of AS-cast by measuring the average number of messages per node over the experiments. In the first experiment, at 50 ms, only one node becomes source, hence there is only one partition for the whole distributed system. In the second experiment, at 50 ms, two nodes become sources, one per cluster. Afterwards both scenarios are identical. At 850 ms, we remove the link between the two clusters. At 1.7 s, we insert back this link.

Results: Figure 7 shows the results of this experimentation. The top part displays the results with one source while the bottom part displays the results with one source per cluster. Figure 7 confirms that concurrent `Adds` may reach consistent partitioning faster. In particular, the top part of Figure 7 depicts a slow down in traffic around 500 ms due to the high latency inter-continental link. The first plateau shows the source's autonomous system acknowledging this source, while the second shows the other autonomous system catching up. The inter-continental link is a bottleneck that does not exist when each cluster has its own source.

Figure 7 highlights that removing the inter-continental link generates additional traffic. The cluster cut off from the source must purge all control information about this source. Most importantly, Figure 7 shows that AS-cast still operates well when *physical* partitions appear. Nodes from the cluster without source do not belong to any partition while nodes from the other cluster belong to the same partition.
Figure 7 shows that, when adding back the inter-continental link, the two clusters can communicate again. In the experiment involving one source for two clusters, it generates traffic. After a 200 milliseconds delay corresponding to the link latency, the cut off cluster starts to communicate α messages again. Eventually, all nodes belong to the same partition. However, in the experiment involving one source *per* cluster, the new link does not trigger noticeable traffic, for nodes already know their closest source located in their cluster.
Overall, this experimentation empirically demonstrates the benefits of AS-cast in the context of geo-distributed infrastructures, e.g., inter-connected autonomous systems where nodes and communications links may be added and removed at any time. AS-cast coupled with replication strategies designed for Edge infrastructures would allow locking down the traffic of replica indexing, while providing every node with quick access to replicated content.

5 Related Work

Content indexing services in geo-distributed infrastructures allow nodes to retrieve specific content while leveraging the advantages of replication. These systems mostly rely on dedicated location services hosted by possibly remote third-party nodes; but cutting out the middleman requires that each node maintains its own index in a decentralized fashion.

Third-party: Dedicated services possibly hosted by remote third-party nodes are popular, for they propose the simplest mean to deploy such a service. They must maintain (i) the list of current replicas along with their respective location; and (ii) the network topology to determine the closest replica for every requesting node.
A central server that registers both these information facilitates the computation, for it gathers all knowledge in one place [1, 2, 16, 33]. However, it comes with well-known issues such as load balancing, or single point of failure.
Distributing this index among nodes circumvents these issues [4, 5, 10, 23, 36, 38], but still raises locality issues where nodes (i) request to possibly far away location services the content location, and then (ii) request the actual content from possibly far away replicas. For instance, using distributed hash tables (DHT) [5, 10, 23], each node stores a part of the index defined by an interval between hash values. Hash values are keys to retrieve the associated content location. Before downloading any content, a node requests its location using its key. After round-trips between possibly distant DHT servers, the node gets available replicas. Contrarily to AS-cast, such services do not ensure to include the closest replica.
In addition, they often do not include distance information along with replica location. Determining where resides the closest replica for every requesting node necessarily involves some knowledge about the *current* topology. Maintaining a consistent view of an ever changing topology across a network is inherently complicated [7, 32]. As a consequence, the requesting node ends up downloading from multiple replica hosts, yet only keeping the fastest answer. Nodes either waste network resources, or face lower response time.

Fully decentralized: Cutting out the middleman by hosting a location service on each and every node improves response time of content retrieval. However, it still requires every node to index every live replica as well as their respective distance in order to rank them. Named Data Networking (NDN) [21] broadcasts information about cache updates to all nodes in the system. Having the entire knowledge of all the replica locations along with distance information carried into messages, each node easily determine where the closest copy of specific content resides, without contacting any remote service. Conflict-free replicated datatype (CRDT) [34] for set data structures could also implement such a location service. Nodes would eventually have a set of all replicas, assuming eventual delivery of messages. Such solutions imply that every node receives every message, which contradicts the principles of Edge infrastructures that aim at reducing the volume of data passing through the network. On the opposite, each node running AS-cast only registers its closest replica. This allows AS-cast to use scoped broadcast as a communication primitive to lock down the traffic generated by content indexing based on distances.

Scoped flooding: Some approaches also confine the dissemination of messages. Distance-based membership protocols such as T-Man [24] make nodes converge to a specific network topology depending on targeted properties. Following periodic exchanges, they add and remove communication links to other nodes to converge towards a topology ensuring the targeted properties. While membership protocols and AS-cast share common preoccupations, AS-cast does not aim at building any topology and never modifies neighbors of nodes. The most closely related approaches to AS-cast come from information-centric networking (ICN) [14, 19]. They periodically advertise their cache content, and advertisements stop as soon as they reach uninterested nodes. However their operation requires that (i) they constantly generate traffic even in quiescent systems where nodes do not add nor destroy replicas; and (ii) they define physical-clock-based timeouts the value of which depends on network topology properties such as network diameter. Unfitting timeouts lead to premature removals of live replicas; or slow convergence where nodes wrongly believe that their closest replica is live while it was destroyed. On the opposite, AS-cast quickly informs each node of its closest replica even in large networks; and has no overhead when the system is quiescent.

6 Conclusion

With the advent of the Edge and IoT era, major distributed services proposed by our community should be revised in order to mitigate as much as possible traffic between nodes. In this paper, we tackled the problem of content indexing, a key service for storage systems. We proposed to address this problem as a dynamic logical partitioning where partitions grow and shrink to reflect content and replicas locations, as well as infrastructure changes. Using an efficient communication primitive called scoped broadcast, each node composing the infrastructure eventually knows the node hosting the best replica of specific content. The challenge resides in handling concurrent operations that may impair the propagation of messages, and in turn, lead to inconsistent partitioning. We highlighted the properties that solve this problem and proposed an implementation called AS-cast. Our simulations confirmed that nodes quickly reach consistent partitioning together, and that the generated traffic remains locked down to partitions.

As future work, we plan to enable lazy retrieval of partitions when a large number of contents is involved. Indeed, using AS-cast, every node partakes in the propagation of all indexes which raises scalability issues. We would like to leverage the hierarchical properties of autonomous systems to limit the propagation of such indexes only to interested systems. We also would like to evaluate AS-cast within a concrete storage system such as InterPlanetary File System (IPFS) [20]. This would allow us to assess the relevance of AS-cast in a real system subject to high dynamicity, and compare it against its current DHT-based indexing system that does not include distance in its operation.

References

1 A. Afanasyev, C. Yi, L. Wang, B. Zhang, and L. Zhang. SNAMP: Secure namespace mapping to scale NDN forwarding. In *2015 IEEE Conference on Computer Communications Workshops*, pages 281–286, April 2015.

2 V. Aggarwal, A. Feldmann, and C. Scheideler. Can ISPS and P2P users cooperate for improved performance? *SIGCOMM Comput. Commun. Rev.*, 37(3):29–40, July 2007.

3 A. S. Asrese, S. J. Eravuchira, V. Bajpai, P. Sarolahti, and J. Ott. Measuring web latency and rendering performance: Method, tools & longitudinal dataset. *IEEE Transactions on Network and Service Management*, 16(2):535–549, June 2019.

4 O. Beaumont, A.-M. Kermarrec, L. Marchal, and E. Rivière. VoroNet: A scalable object network based on Voronoi tessellations. In *2007 IEEE International Parallel and Distributed Processing Symposium*, pages 1–10, 2007.

5 J. Benet. IPFS - content addressed, versioned, P2P file system. *CoRR*, abs/1407.3561, 2014.

6 K. P. Birman, M. Hayden, O. Ozkasap, Z. Xiao, M. Budiu, and Y. Minsky. Bimodal multicast. *ACM Transactions on Computer Systems*, 17(2):41–88, May 1999.

7 Y. Breitbart, M. Garofalakis, C. Martin, R. Rastogi, S. Seshadri, and A. Silberschatz. Topology discovery in heterogeneous IP networks. In *Proc. IEEE INFOCOM 2000. Conference on Computer Communications. 19th Annual Joint Conference of the IEEE Computer and Communications Societies (Cat. No.00CH37064)*, volume 1, pages 265–274 vol.1, 2000.

8 B. Confais, A. Lebre, and B. Parrein. An object store service for a fog/edge computing infrastructure based on IPFS and a scale-out NAS. In *2017 IEEE 1st International Conference on Fog and Edge Computing (ICFEC)*, pages 41–50. IEEE, 2017.

9 B. Confais, A. Lebre, and B. Parrein. Performance analysis of object store systems in a fog and edge computing infrastructure. In *Transactions on Large-Scale Data-and Knowledge-Centered Systems XXXIII*, pages 40–79. Springer, 2017.

10 M. D'Ambrosio, C. Dannewitz, H. Karl, and V. Vercellone. MDHT: A hierarchical name resolution service for information-centric networks. In *Proc. of the ACM SIGCOMM Workshop on Information-centric Networking*, ICN '11, pages 7–12, New York, NY, USA, 2011.

11 T. V. Doan, L. Pajevic, V. Bajpai, and J. Ott. Tracing the path to YouTube: A quantification of path lengths and latencies toward content caches. *IEEE Communications Magazine*, 57(1):80–86, 2019.

12 U. Drolia, K. Guo, J. Tan, R. Gandhi, and P. Narasimhan. Cachier: Edge-caching for recognition applications. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pages 276–286, June 2017.

13 P. T. Eugster, R. Guerraoui, A.-M. Kermarrec, and L. Massoulié. Epidemic information dissemination in distributed systems. *Computer*, 37(5):60–67, 2004.

14 J. J. Garcia-Luna-Aceves, J. E. Martinez-Castillo, and R. Menchaca-Mendez. Routing to multi-instantiated destinations: Principles, practice, and applications. *IEEE Transactions on Mobile Computing*, 17(7):1696–1709, 2018.

15 P. Guo, B. Hu, R. Li, and W. Hu. FoggyCache: Cross-device approximate computation reuse. In *Proc. of the 24th Annual International Conference on Mobile Computing and Networking*, MobiCom '18, pages 19–34, New York, NY, USA, 2018. ACM.

16 H. Gupta and U. Ramachandran. Fogstore: A geo-distributed key-value store guaranteeing low latency for strongly consistent access. In *Proc. of the 12th ACM International Conference on Distributed and Event-based Systems*, DEBS '18, pages 148–159, New York, NY, USA, 2018.

17 V. Hadzilacos and S. Toueg. A modular approach to fault-tolerant broadcasts and related problems. Technical report, USA, 1994.

18 J. Hasenburg, M. Grambow, and D. Bermbach. Towards a replication service for data-intensive fog applications. In *Proc. of the 35th Annual ACM Symposium on Applied Computing*, pages 267–270, 2020.

19 E. Hemmati and J. J. Garcia-Luna-Aceves. A new approach to name-based link-state routing for information-centric networks. In *Proc. of the 2nd ACM Conference on Information-Centric Networking*, ACM-ICN '15, page 29–38, New York, NY, USA, 2015.

20 S. Henningsen, M. Florian, S. Rust, and B. Scheuermann. Mapping the interplanetary filesystem, 2020.

21 A K M M. Hoque, S. O. Amin, A. Alyyan, B. Zhang, L. Zhang, and L. Wang. NLSR: Named-data link state routing protocol. In *Proc. of the 3rd ACM SIGCOMM Workshop on Information-centric Networking*, ICN '13, pages 15–20, New York, NY, USA, 2013.

22 H.-C. Hsiao and C.-T. King. Scoped broadcast in dynamic peer-to-peer networks. In *Proc. of the 29th Annual International Computer Software and Applications Conference (COMPSAC 2005)*, volume 1, pages 533–538, Los Alamitos, CA, USA, jul 2005. IEEE Computer Society.

23 S. Iyer, A. Rowstron, and P. Druschel. Squirrel: A decentralized peer-to-peer web cache. In *Proc. of the Twenty-first Annual Symposium on Principles of Distributed Computing*, PODC '02, pages 213–222, New York, NY, USA, 2002. ACM.

24 M. Jelasity and O. Babaoglu. T-man: Gossip-based overlay topology management. volume 3910, pages 1–15, 05 2006.

25 A.-M. Kermarrec and F. Taïani. Want to scale in centralized systems? think P2P. *Journal of Internet Services and Applications*, 6(1):1–12, 2015.

26 S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan. The internet topology zoo. *IEEE Journal on Selected Areas in Communications*, 29(9):1765 – 1775, 2011.

27 L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565, July 1978.

28 M.-Y. Lue, C.-T. King, and H. Fang. Scoped broadcast in structured P2P networks. In *Proc. of the 1st International Conference on Scalable Information Systems*, InfoScale '06, page 51–es, New York, NY, USA, 2006.

29 P. Maymounkov and D. Mazières. Kademlia: A peer-to-peer information system based on the xor metric. In *Peer-to-Peer Systems*, pages 53–65. Springer Berlin Heidelberg, 2002.

30 A. Montresor and M. Jelasity. PeerSim: A scalable P2P simulator. In *Proc. of the 9th International Conference on Peer-to-Peer (P2P'09)*, pages 99–100, Seattle, WA, 2009.

31 M. Raynal. *Distributed algorithms for message-passing systems*, volume 500. Springer, 2013.

32 RFC2328. OSPF Version 2. `https://tools.ietf.org/html/rfc2328`, 1998.

33 J. Seedorf, S. Kiesel, and M. Stiemerling. Traffic localization for P2P-applications: The ALTO approach. In *2009 IEEE 9th International Conference on Peer-to-Peer Computing*, pages 171–177, Sept 2009.

34 M. Shapiro, N. Preguiça, C. Baquero, and M. Zawirski. Conflict-free Replicated Data Types. Research Report RR-7687, July 2011.

35 W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu. Edge computing: Vision and challenges. *IEEE Internet of Things Journal*, 3(5):637–646, 2016.

36 S. Triukose, Z. Wen, and M. Rabinovich. Measuring a commercial content delivery network. In *Proceedings of the 20th International Conference on World Wide Web*, WWW '11, page 467–476, New York, NY, USA, 2011. ACM.

37 L. Wang, S. Bayhan, J. Ott, J. Kangasharju, A. Sathiaseelan, and J. Crowcroft. Pro-diluvian: Understanding scoped-flooding for content discovery in information-centric networking. In *Proc. of the 2nd ACM Conference on Information-Centric Networking*, ACM-ICN '15, page 9–18, New York, NY, USA, 2015.

38 J. Xie, C. Qian, D. Guo, M. Wang, S. Shi, and H. Chen. Efficient indexing mechanism for unstructured data sharing systems in edge computing. In *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, pages 820–828, 2019.