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

Adrien Lebre, Brice Nédelec, Alexandre van Kempen

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

Adrien Lebre[1], Brice NÃľdelec[1], and Alexandre Van Kempen[2]

[1] Nantes UniversitÃľ, ÃĽcole Centrale Nantes, IMT Atlantique, CNRS, INRIA, LS2N, UMR 6004, F-44000 Nantes, France {adrien.lebre, brice.nedelec}@ls2n.fr

[2] Qarnot computing, 40/42 rue BarbÃĺs 92120 Montrouge, France alexandre.vankempen@qarnot-computing.com

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 for such a aim 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.
In this paper, we propose a decentralized implementation of content indexing called AS-cast. Using AS-cast, every node only indexes its closest replica; and all connected nodes with a similar index compose a *partition*. 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 closest replica despite concurrent operations. Our prototype, implemented on PeerSim, shows that AS-cast scales well in terms of generated messages and termination time.
As such, AS-cast can constitute a novel building block for geo-distributed services in need of efficient resource sharing in the vicinity of regions.

Keywords: Edge infrastructures, decentralized content indexing, scoped broadcast, logical partitioning protocol

1 Introduction

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 [13,16,35]. To favor this transition, new designs for data management across Edge infrastructures have been investigated [9,10,17,19]. They enable strategies to confine traffic by writing data locally and replicating content

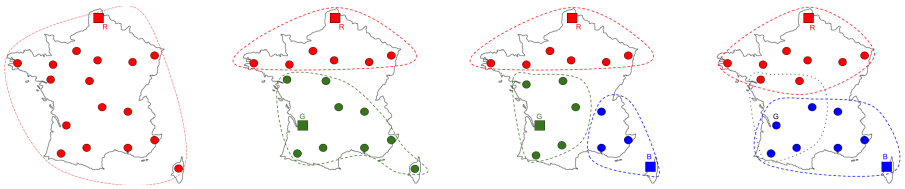
according to effective needs. However, locating content remains challenging. Retrieving a content location may actually take more time than retrieving the content itself. Indeed, these systems, when not using a centralized index hosted in a Cloud, rely on distributed hash tables (DHT) spread across different nodes composing the infrastructure [29]. 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 retrieving the content, the client can create another replica to improve the performance of future accesses, but it must recontact the indexing service to notify of the creation of this new replica.

Such 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 [3,12] 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 hosts, yet only keeping the fastest answer. In turn, clients either waste network resources, or face slower 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 broadcasting its operation [7,18]. 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.

To mitigate the broadcasting overhead, 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 rally their new closest partition. Some periodic advertisement protocols [20,36] already provide both creation and deletion. Yet, their functioning requires (i) to generate traffic even when the system is quiescent, and (ii) the use of synchronous communications [36] or physical-time-based intervals tuned according to network topology parameters such as network diameter [20].
In this paper, we address the content indexing challenge as a distributed partitioning problem. Our contribution 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.



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

Fig. 1: Partitions grow/shrink depending on creations/removals of replicas.

• 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 can join, leave, or 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-dependent predicate.

• We evaluate AS-cast through simulations using PeerSim [30]. Results empirically show that (i) AS-cast manages to quickly disseminate messages to the subset of relevant nodes; (ii) AS-cast's overhead decreases when the number of partitions increases; (iii) AS-cast does not generate traffic in quiescent systems; and (iv) AS-cast operates even in networks subject to physical partitioning.

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. Section 4 presents our evaluations. Section 5 reviews related work. Section 6 concludes and discusses future work.

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 content locations. More precisely, we propose to consider 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 [18] and epidemic propagation [14] to guarantee that every node eventually knows the content location 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). In Figure 1c, there are three logical partitions.

Dynamic partitioning: 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. A better approach would consist in triggering scoped broadcast at bordering nodes of red and blue partitions once again. In other words, the scope of scoped broadcast changes over receipts by nodes. This dynamic partitioning raises additional challenges related to concurrent operations where removed partitions could block the propagation of other partitions (see Section 3.3). Next Section details the properties of scoped broadcast and dynamic partitioning, and provides an implementation called AS-cast.

3 Adaptive scoped broadcast

To provide dynamic logical partitioning using scoped broadcast, all live nodes must collaborate to disseminate messages that notify new or removed sources to all and only interested nodes. This section reviews step-by-step the properties that allow nodes to converge to the desired state together. It first defines scoped

Table 1: Summary of notations.

Notation	Short	Description
G	Graph	Represents a network.
V	Vertices	Represents the set of nodes, or processes.
E	Edges	Represents the set of asynchronous communication links.
w_{xy}	weight	Positive weight of the edge $\langle x, y\rangle$.
π_{xz}/Π_{xz}	path/best Path	List of contiguous edges from Node x to Node z.
$\lvert\pi_{xz}\rvert/\lvert\Pi_{xz}\rvert$	sum of weights	Positive sum of weights of the path.
σ_x	state	The local state of Node x.
$b_x(m)$	broadcast	Node x creates a new message m that must be delivered by all nodes.
$d_x(m)$	deliver	Node x delivers the message m.
$r_y(m)/r_{yx}(m)$	receive	Node y receives the message m from any neighboring node/Node x.
$s_{xy}(m)$	send	Node x sends the message m to Node y.
$f_x(m)$	forward	Node x forwards the message m to its neighbors.
$m \oplus \sigma$	aggregator	Aggregates σ into the metadata of message m.
$\phi(\mu, \sigma)$	predicate	Checks the metadata μ using the state σ.
$\Diamond P$	eventually	Eventually predicate P is true.
$e_1 \rightarrow e_2$	happens before	The event e_1 happened before the event e_2. d, s, r etc. are events.
D_x	Delivered	Set of delivered messages by Node x.
α_x^d or $\alpha_{\pi_{xz}}^d$	add source	Message that notifies the adding of Source x in the network.
δ_x	source deletion	Message that notifies the possible deletion of Source x by Node x.
$\mathcal{S}(m)$	Stale message	Message m conveys stale control information.

broadcast, then uses it to guarantee consistent partitioning when a node can only become a new source in the system. It highlights the issue when a node can also remove its status of source. It shows that using local knowledge and scoped broadcast, nodes can still reach dynamic consistent partitioning when they are able to detect possible blocking conditions in the dissemination of required notifications, even in dynamic networks where nodes join, leave, or crash at any time. For convenience, Table 1 summarizes the notations used throughout the paper.

3.1 Scoped broadcast

In this paper, we consider Edge infrastructures as a set of interconnected autonomous systems comprising 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-dependent scope.

Definition 1 (Autonomous system). *An autonomous system is a network comprising nodes and communication links that we represent as a graph of vertices and edges: $G = \langle V, E\rangle$ with $E \in V \times V$. A path π_{xz} from Node x to Node z is a sequence of contiguous edges $[\langle x, y_1\rangle, \langle y_1, y_2\rangle, \ldots, \langle y_n, z\rangle]$.*

Definition 2 (Scoped broadcast). *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 [22,28,38]. It underlines the transitive relevance of messages. It also highlights that the functioning of epidemic propagation is well-aligned with the objectives of scoped broadcast. As consequence, we assume implementations relying on the forwarding of messages from neighbor-to-neighbor.

Definition 3 (Forwarding). *When x forwards $f_x(m)$ a message m, it sends it $(s_{xy}(m))$ to all its neighbors y accumulating and aggregating ($\oplus$) metadata that depends on its local knowledge σ_x: $f_x(m) \implies \forall \langle x, y \rangle \in E : s_{xy}(m \oplus \sigma_x)$.*

We use scoped broadcast to efficiently modify the state of each node depending on the partitions that exist in the system to reach consistent partitioning.

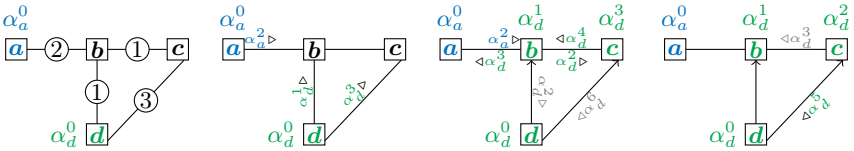
3.2 Consistent partitioning

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 are closer to this source than any other one. Such a distance (or *weight*) is application-dependent: in the context of maintaining distributed indexes, this would be about link latency that nodes could monitor by aggregating `pings`; or operational costs when dealing with multiple tenants.

Definition 4 (Consistent partitioning (CP)). *Assuming a set of sources $S \subseteq V$, a positive weight w_{xy} associated with each edge $\langle x, y \rangle \in E$, we define consistent partitioning as a set of logical partitions $P_{s \in S}$ where each node x belongs to the partition of its closest source s, i.e., there exists a path π_{sx} with a sum of weights $|\pi_{sx}| = \sum\{w_{pq} | \langle p, q \rangle \in \pi_{sx}\}$ smaller than any other path, with $|\pi_{xx}|$ being x's greatest lower bound.*

Unfortunately, nodes do not share a common global knowledge of the network state. For nodes to eventually ($\Diamond$) reach consistent partitioning (CP), each source s must send `Add` notifications α_s to all nodes that are closer to it than any other source, along with enough metadata to allow them to decide of their closest source. Based on Definition 2 and Definition 3, this consists in ensuring that each node x eventually receives $\alpha_s^{|\Pi_{sx}|}$ where Π_{sx} is the *best* path from any source to it; the best value $|\Pi_{sx}|$ being built over forwarding along this path.

Theorem 1 (Forwarding of Best (FB) $\implies \Diamond$ CP). *Assuming that each node x stores its outgoing weights ($\forall \langle x, y \rangle \in E : w_{xy} \in \sigma_x$), a total order on messages based on weights ($m_s^d < m_{s'}^{d'}$ when $d < d' \vee (d = d' \wedge s < s')$), reliable communication links ($s_{xy}(m) \iff \Diamond\, r_{yx}(m)$), nodes eventually reach consistent partitioning if each node delivers its best (i.e., smallest) message among received*



(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 improved with α_d^2.

Fig. 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.

messages R_x, and forwards its best message among delivered messages D_x such that $f_x(\alpha_s^d) \implies \forall \langle x, y \rangle \in E : s_{xy}(\alpha_s^d \oplus^{w_{xy}})$, i.e., by accumulating the respective weight of used edges.

Proof. Whenever a node s becomes a source, it broadcasts hence delivers its own message $d_s(\alpha_s^{|\pi_{ss}|})$. Whatever its set of received messages, it acknowledges that it belongs to its own partition P_s since $\alpha_s^{|\pi_{ss}|} = \min D_s$ and it remains forever since $|\pi_{ss}|$ is its greatest lower bound. Such a source forwards its notification to its neighbors. Every neighbor eventually receives its notification since communication links are reliable. Whatever the order of received messages R_x at neighboring node x, total order ensures that it delivers notifications α_s^d when $d = |\pi_{ss}| + w_{sx} = \min |\pi_{s'x}|$ where $\min |\pi_{s'x}|$ is the lightest weight of x to any source s', s' being s in this case. Among these neighbors, at least those that fulfill the latest condition forward their respective notification. By transitivity, the message originating from s reaches all nodes that belong to P_s at least through their respective lightest path: $\forall y \in V, s, s' \in S : \min |\pi_{sy}| < \min |\pi_{s'y}| \implies \Diamond\, d_y(\alpha_s^{\min |\pi_{sy}|})$. When the system becomes quiescent, i.e., no node becomes source anymore, every node eventually acknowledges the partition it belongs to, i.e., nodes eventually reach consistent partitioning together. In addition, the protocol terminates: a node never delivers hence forwards a message after it received, delivered, and forwarded the message of its closest source from its lightest path.

Algorithm 1 shows the instructions that implement a *reactive* forwarding of best to reach consistent partitioning in a static network where nodes never join, crash, nor leave the system. As soon as a node receives a better message, it delivers and forwards it. The trade-off consists in decreasing termination time at cost of increased traffic. Figure 2 illustrates the behavior of this algorithm on a system comprising 4 nodes a, b, c, and d. Both a and d become sources. They scoped broadcast notifications α_a^0 and α_d^0. They initialize their own state with the lowest value 0 (see Line 3), and send a message to each of their neighbors

Algorithm 1: Add-only CP protocol at Node p.

```
1 O_p, W_p                                              // set of neighbors and weights
2 A_s^d → α_∅^∞                                         // smallest distance d to Source s
3 func Add( )  receiveAdd(∅, α_p^0)                     // become source
4 func receiveAdd(q, α_{s'}^{d'})                       // notification from q
5 |  if α_{s'}^{d'} < A_s^d then                        // check predicate (φ)
6 |  |  A_s^d ← α_{s'}^{d'}                            // deliver and update (σ)
7 |  |  foreach n ∈ O_p do  send_n(α_{s'}^{d'} ⊕^{w_pn})   // forward to neighbors
```

by accumulating the corresponding edge weight (see Line 7). In Figure 2c, b receives α_d^1. Since it improves its own partition distance, it keeps it and forwards it to its neighbors. In Figure 2d, b discards α_a^2, for it does not improve its partition distance. Neither c nor d will ever acknowledge that Source a exists. The protocol lacks of obvious traffic optimization, e.g., grey messages are useless in this scenario. Nevertheless, the system discards last transiting messages after it reached consistent partitioning.

Adding logical partitions to a static network is straightforward and lightweight. Unfortunately, removing partitions or introducing dynamic network membership increases complexity and costs caused by concurrent operations.

3.3 Dynamic consistent partitioning

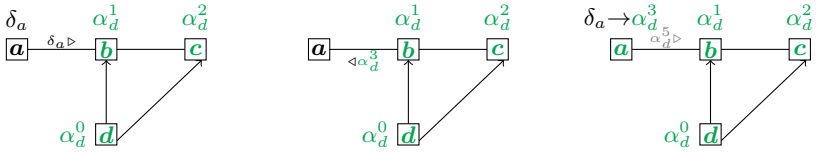
At any time, a source can revoke its status of source by executing a `Del` operation, hence deleting its partition from the system. All nodes that belong to this partition must eventually choose another partition to belong to.

Definition 5 (Dynamic consistent partitioning (DCP)). *A DCP protocol enables nodes to join or leave the set of sources at any time while ensuring eventual consistent partitioning.*

Delete operations bring a new notion of order between events, and most importantly between message deliveries. A node s that performs a `Del` operation implicitly states that its preceding `Add` operation becomes obsolete. Their results – scoped broadcast in the form of `Add` notifications α_s – should be canceled by the corresponding `Del` notifications of staleness δ_s. We focus on the last delivery of each node, since the best is also the last, as highlighted by Algorithm 1.

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 (Stale messages). *Only the latest broadcast of a node matters. A message m conveys stale control information $\mathcal{S}(m)$ as soon as its broadcaster broadcasts another message: $\mathcal{S}(\alpha_x) = \exists \delta_x : b_x(\alpha_x) \rightarrow b_x(\delta_x)$. A node only delivers or sends messages that it assumes up-to-date. For convenience, we define* $\text{last}(D) = \alpha_s \in D : \nexists \alpha_{s'} \in D : d_x(\alpha_s) \rightarrow d_x(\alpha_{s'})$.



(a) a deletes its partition. It notifies all nodes that belong to its partition.

(b) δ stops when it encounters another partition. b answers with its partition.

(c) a receives, delivers, and forwards the message α_d^3. Every node belongs to P_d.

Fig. 3: Efficient removal of a partition using scoped broadcast.

A naive attempt at implementing DCP resembles echoes in acoustics: a sound propagates in the air before crashing into surrounding walls to finally come back altered. Following the principles of scoped broadcast as stated in Definition 2, a node x that receives a staleness notification δ_s forwards it only if the latter targets its current partition α_s. These messages propagate as long as they remain in the deleted partition. When they reach the bordering nodes of the deleted partition, it creates an echo of bordering partitions that will go backward to fill the gap left open by removals using forwarding of best (FB) as stated by Theorem 1.

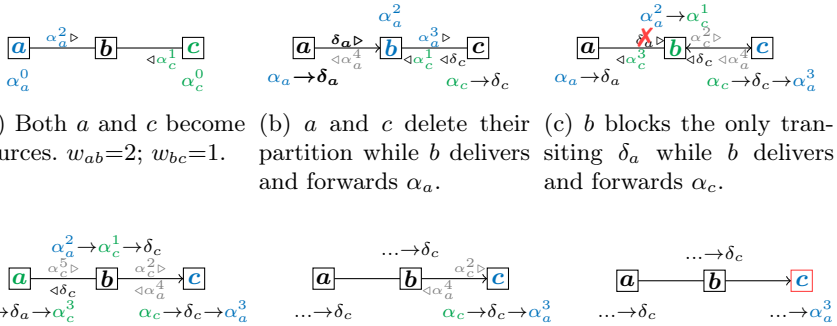
Definition 8 (Forwarding of staleness (FS)). *Any source can broadcast a staleness notification at any time. Every node x delivers and forwards a received staleness notification δ_s if it targets its best up-to-date delivered message:*
$(\text{last } D_x = \alpha_s \wedge d_x(\alpha_s) \to r_x(\delta_s)) \implies d_x(\delta_s) \to f_x(\delta_s)$.

Definition 9 (FB^+: echos). *In addition to FB, a node x that receives but does not deliver a staleness notification δ_s replies by – or echoes – its best up-to-date delivered message:*
$(\text{last } D_x = \alpha_{s'}^d \wedge \delta_{s'} \notin D_x \wedge d_x(\alpha_{s'}^d) \to r_{xy}(\delta_s)) \implies r_{xy}(\delta_s) \to s_{xy}(\alpha_{s'}^d \oplus^{w_{xy}})$.

Figure 3 illustrates the behavior of this implementation ($\text{FB}^+ \wedge \text{FS}$). Two partitions initially exist: P_a and P_d that respectively include $\{a\}$, and $\{b, c, d\}$. In Figure 3a, a deletes its partition. FS in Definition 8 states that a must notify all its neighbors – here only b – that may belong to its partition. In Figure 3b, b receives but does not deliver δ_a since δ_a does not target its current partition P_d. FB^+ in Definition 9 states that b must echo back its own best up-to-date message α_d^3, for it may be the best for a. Figure 3c shows that every node eventually belongs to P_d, therefore reaching consistent partitioning. Unfortunately, this protocol does not ensure consistent partitioning in every scenario.

Lemma 1 ($\text{FB}^+ \wedge \text{FS} \not\Longrightarrow$ DCP). *Forwarding best up-to-date, staleness, and echoes is not sufficient to guarantee dynamic consistent partitioning.*

Proof. Stale control information (see Definition 7) may impair the propagation of both (i) notifications about actual sources, and (ii) notifications about deleted



(a) Both a and c become sources. w_{ab}=2; w_{bc}=1.

(b) a and c delete their partition while b delivers and forwards α_a.

(c) b blocks the only transiting δ_a while b delivers and forwards α_c.

(d) a receives, delivers, and forwards α_c^3.

(e) a receives, delivers, and forwards δ_c.

(f) c is stuck in a stale partition P_a.

Fig. 4: The naive propagation of messages is insufficient to guarantee consistent partitioning. If c had children, they would stay in the wrong partition too.

partitions. In Figure 4, a, b, c are chained with FIFO links, i.e., nodes receive the messages in the order of their sending. In Figure 4a, a and c become sources, sending their respective notification message to b. In Figure 4b, a and c delete their partition while b receives, delivers, and forwards α_a^2. In Figure 4c, b receives, delivers, and forwards α_c^1, for it improves its best partition. Then it receives and discards δ_a, for its best partition does not match the deleted one. It echoes back α_c^3 to a. In Figures 4d and 4e, a and b handle transiting notifications. Finally, Figure 4f shows that c is stuck in the deleted P_a for not having received δ_a that b blocked earlier. The system does not reach consistent partitioning.

The issue is that each node trusts its parent to forward all staleness notifications relevant to it. When this fails, as in Figure 4, not only a node (c) may wrongfully stay in a partition (P_a) when its source (a) already deleted it, but it may contribute to its inconsistency by not forwarding farther but up-to-date messages from live sources.

A first step to avoid staying in inconsistent state is to extend the behavior of each node so nodes such as c, that would remain in a wrong partition, use their history of receipt and delivery to detect the possible blocking of staleness notifications that can hinder reaching consistent partitioning. In Figure 4, b blocked the staleness notification δ_a that c needs since it belongs to a ephemeral partition P_c that c acknowledges to be stale before b does. Therefore, c can detect the possible blocking of δ_a since it acknowledges that its parent b received, delivered, and forwarded the stale notification α_c^1. In other terms, Node c acknowledges that between the delivery of α_c^1 and the delivery of δ_c, Node b blocked all other δ messages and therefore, it could have blocked the most important staleness notification: δ_a.

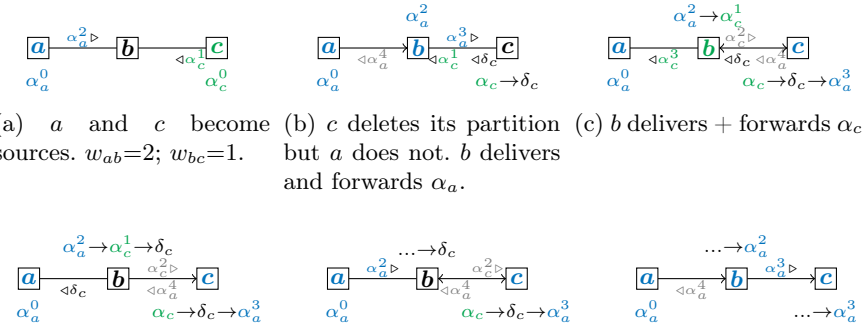
Lemma 2 ($\mathbf{FB^+} \wedge \mathbf{FS} \implies$ Detector existence). *Assuming* FB^+ *and* FS*, a chain of delivery on* π_{xz} *of* α_x *with* $\alpha_x = \text{last } D_z$*, if* x *broadcasts a staleness notification* δ_x*, either (A)* z *eventually delivers it, or (B) a node* y *in such a chain of delivery* π'_{xz} *eventually detects the possible blocking of* δ_x*.*

Proof. Assuming a chain of delivery $\pi_{xz} = [x, \ldots, z]$ of α_x with $\alpha_x = \text{last } D_z$ and $d_x(\delta_x)$, we must prove that whatever the possible states of nodes that belong to this chain, it either leads to outcome (A) or (B).

(I) Same last partition: $\forall y \in \pi_{xz} : \alpha_x = \text{last } D_y$, with forwarding of staleness (Definition 8), $f_{\pi_{xz}[k]}(\delta_x) \implies \Diamond\, d_{\pi_{xz}[k+1]}(\delta_x)$ except if $\delta_x \in D_{\pi_{xz}[k+1]}$ meaning that $\pi_{xz}[k+1]$ already delivered and forwarded δ_x following another delivery chain. Therefore, z eventually receives, hence delivers δ_x (outcome A).

(II) Different last partitions: $\exists y \in \pi_{xz} \setminus \{x, z\} : \alpha_s = \text{last } D_y$, by (I), the staleness notification reaches the first of such a node $y = \pi_{xz}[k]$. Forwarding of staleness (Definition 8) states that the forwarding stops when y already delivered δ_x ($\delta_x \in D_y$ covered by (I)) or delivered a better message $\alpha_s = \text{last } D_y$. With $y' = \pi_{xz}[k+1]$, forwarding of best (Definition 9) implies three possible results:
(i) y' equivalent to y: $\alpha_{s'} = \text{last } D_{y'}$ with $\alpha_{s'} \neq \alpha_x$ hence $y' \iff y$ which leaves two possible results as follows:
(ii) y' in P_x from another parent: $r_{y'}(\alpha_s) \wedge \alpha_x = \text{last } D_{y'} \wedge \alpha_x < \alpha_s$ which means that a shorter path of delivery π'_{xz} exists that either forwards appropriate staleness notifications (covered by (I)) or detects a possible blocking of the latter as follows:
(iii) y' in P_x with y as parent but does not deliver the y's last partition for it already delivered the corresponding staleness notification: $r_{y'}(\alpha_s) \wedge \delta_s \in D_{y'} \wedge \alpha_x = \text{last } D_{y'}$ which detects a possible blocking of δ_x (outcome (B)). Without global knowledge, y' assumes it belongs to the shortest and only path of delivery of α_x thus it cannot further delegate the detection to another node.

As a second step, detecting nodes must proactively purge the system from their forwarded notifications. For instance in Figure 4f, c detects the blocking of the staleness notification δ_a. It could broadcast δ_a in order to acknowledge the staleness of its own α_a^3 and inform neighboring nodes that delivered it as well. Unfortunately, there exists false positive in the blocking detection. Lemma 2's proof shows that the detection at y' does not depend on δ_x. Therefore δ_x may not exist but y' still receives α_s after the delivery of δ_s from the path it received and delivered α_x. Figure 5 highlights this behavior. In Figure 5b, a does not delete its partition. In Figure 5f and because of scoped broadcast, c received the same series of messages as in Figure 4f. Yet, c must assume the existence of δ_a and act accordingly by forcing the staleness of its best delivered message and disseminate this information to its neighbors. To avoid flooding the system with false positive staleness notifications, we reduce the scope of staleness notifications by including only downstream nodes. In Figure 5f, false positives would generate traffic at c and in turns, to its children if it had any. Nevertheless, this would not affect a nor b.



(a) a and c become sources. w_{ab}=2; w_{bc}=1.

(b) c deletes its partition but a does not. b delivers and forwards α_a.

(c) b delivers + forwards α_c.

(d) b delivers + forwards δ_c.

(e) a echoes back α_a^2 to b.

(f) a, b, c belong to P_a.

Fig. 5: From c's perspective, Figure 4e and Figure 5e are similar in terms of received messages, but the outcomes eventually differ. Yet, c must act on Figure 5e, and acknowledge then propagate the *possible* staleness of Partition P_a.

Definition 10 ($\mathbf{FS}^+$: forwarding staleness downstream). *Any* node *can broadcast a staleness notification at any time. A* child node x *delivers and forwards a received staleness notification if it comes from the path of delivery of its best up-to-date delivered message.*

It is worth noting that forwarding of staleness as stated in Definition 8 becomes a specific case of Definition 10 where the source itself forwards a staleness notification downstream. The notification must reach all nodes in its partition since the source has no nodes upstream, and belongs to the delivery path of all nodes that delivered this message.

Theorem 2 ($\mathbf{FB}^+ \wedge \mathbf{FS}^+ \implies$ DCP). *A protocol guarantees dynamic consistent partitioning if it implements forwarding of best up-to-date messages with echos, forwarding of staleness messages downstream, and the detection of possible blocking triggers the forwarding of staleness notifications downstream.*

Proof. Detection triggers forwarding of staleness downstream which completes the case study of Lemma 2 by ensuring that, when a source broadcasts its staleness notification, all nodes that belong to its partition eventually deliver such a staleness notification. All downstream bordering nodes also eventually receive such a staleness notification and echo back their best delivered message. This triggers another competition as in Theorem 1 for the nodes that delivered the staleness notification.

Algorithm 2 shows the instructions of our implementation called AS-cast that enables dynamic consistent partitioning. It implements forwarding of best (see Line 13) and echos (see Line 21). To implement forwarding of staleness

Algorithm 2: AS-cast: DCP protocol at Node p.

```
1  O_p, W_p                                          // set of neighbors and weights
2  A_π^d ← α_∅^∞                                     // best α so far
3  V ← ∅;  V[p] ← 0                                  // vector of versions
4  func Add( )                                       7  func Del( )
5  |  V[p] ← V[p] + 1                               8  |  V[p] ← V[p] + 1
6  |  receiveAdd(p, α_∅^0)                          9  |  receiveDel(p, δ_{p,V[p]})
10 func receiveAdd(q, α_{π'}^{d'})                   // notification of source creation
11 |  if α_{π'}^{d'} < A_π^d and ¬isStale(α_{π'}^{d'}) and ¬isLoop(α_{π'}^{d'}) then
12 |  |  A_π^d ← α_{π'}^{d'} ∪ ⟨p, V[p]⟩
13 |  |  foreach n ∈ O_p do send_n(A_π^d ⊕^{w_pn})
14 |  else if isParent(q) and isStale(α_{π'}^{d'}) then
15 |  |  receiveDel(q, δ_{p,V[p]+1})              // detection of possible inconsistency
16 |  updateVersions(π')
17 func receiveDel(q, δ_{s,c})                     // notification of a possible source removal
18 |  if ∃⟨s, c'⟩ ∈ π : c' < c then
19 |  |  A_π^d ← α_∅^∞
20 |  |  foreach n ∈ O_p \ q do send_n(δ_{s,c})
21 |  else if A_s^d ≠ α_∅^∞ then send_q(A_π^d ⊕^{w_pq})
22 |  updateVersions([⟨s, c⟩])
23 func onEdgeUp(q)                                 // new communication link to q
24 |  if A_π^d ≠ α_∅^∞ then send_q(A_π^d ⊕^{w_pq})
25 func onEdgeDown(q)                              // link to q removed (crash or leave)
26 |  if isParent(q) then receiveDel(q, δ_{p,V[p]+1})
27 func isStale(α_{π'}^{d'}) return ∃⟨q, c⟩ ∈ π' : c < V[q]
28 func isLoop(α_{π'}^{d'}) return ⟨p, _⟩ ∈ π'
29 func isParent(q) return ⟨q, _⟩ = π[|π| − 1]
30 func updateVersions(π') for ⟨q, c⟩ ∈ π' do V[q] ← max(V[q], c)
```

downstream as stated in Definition 10: (A) each node maintains a vector of versions that associates the respective known local counter of each known source, or has-been source. It constitutes a summary of known progress of other nodes; (B) each notification message α carries the list of identifier and counter of each node that forwarded it. In the worst case, both these structures grow linearly with the number of nodes in the system $\mathcal{O}(V)$. Nevertheless, following the principles of scoped broadcast, we expect that nodes only acknowledge a small subset of sources and messages; (C) each staleness notifications δ only carry the identifier and counter of the node – source or detector – that generated it. Only downstream nodes may deliver such message, since they carry the identifier of the generator. To implement the detection as stated in Lemma 2, each node only requires to know the direct parent of its best delivered message which is already included in the piggybacked path of this message. Receiving a message known to

be stale from this parent triggers the generation of staleness notifications that can only be delivered by downstream nodes (see Line 14).

By reusing AS-cast's default behavior of echos and downstream staleness, Algorithm 2 also enables dynamic consistent partitioning even in dynamic networks subject to physical partitioning where nodes can join, leave, or crash at any time. Adding a node is equivalent to add as many edges as necessary. Since adding an edge may improve shortest paths, the protocol triggers a competition using echos of Definition 9. Removing a node is equivalent to removing all its edges. Removing an edge between two nodes may invalidate the shortest path of one of involved nodes plus downstream nodes, or impair the propagation of staleness notifications. Therefore, the protocol reuses its detectors of Lemma 2 to remove irrelevant messages from the system.

AS-cast follows the principles of scoped broadcast to efficiently and consistently propagate notifications to nodes that need them despite concurrent operations. Next Section aims at providing the empirical evidence of scalability and properties of the proposed approach.

4 Experimentation

In this section, we discuss the evaluations of AS-cast we conducted on top of PeerSim, a simulator to evaluate and assess distributed algorithms in large scale networks [30]. All the code of this experimentation section is available at: `https://gitlab.inria.fr/STACK-RESEARCH-GROUP/as-cast`.

4.1 Scalability and trade-off of AS-cast

Description: We build a network comprising 10k nodes. First, we chain nodes together, then we add another 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. To favor more concurrent operations, we set weights and latency to different values: each link has a weight between 5 and 15 set randomly using a uniform distribution. This allows nodes to receive messages in different orders, therefore generating more messages to reach consistent partitioning.
To evaluate dynamic consistent partitioning, we first 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 measure the average number of messages per node per second; 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. This corresponds to a complexity in terms of number of messages of $\mathcal{O}(\frac{|E|}{|V|\cdot|P|})$ where

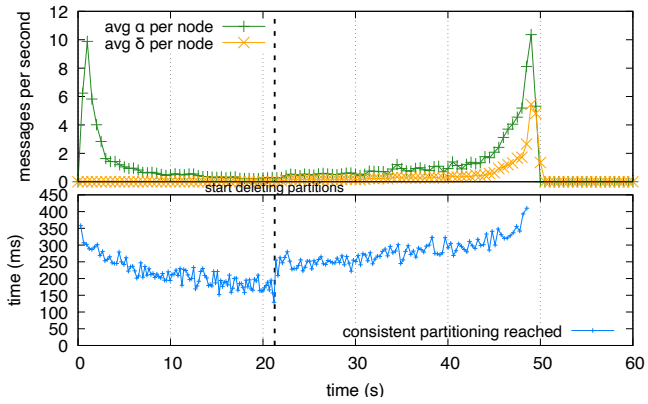


Fig. 6: Dynamic consistent partitioning overhead of AS-cast.

E is the number of links, V is the number of nodes, and P is the number of partitions. In other terms, the p^{th} partition contains $1/p$ nodes and reduces the size of closest partitions so every partition has $1/p$ nodes on average.
Therefore, the first partition is the most expensive, for α messages must reach every node which takes time and generate more traffic. AS-cast quickly converges towards consistent partitioning in 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 roughly twice as expensive as creation ones. Indeed, the top part of the figure shows that after 21 seconds, when the experiment involves removals, traffic includes both α and δ messages. 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. 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 partitions as opposed to past partitions.
Finally, Figure 6 highlights that after 49 seconds, i.e., after the 100 creations 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.

4.2 Traffic containment in dynamic inter-autonomous systems

Description: We build a network by duplicating the GÉANT topology [26] – an infrastructure of 271 nodes spanning across Europe – and we link these two

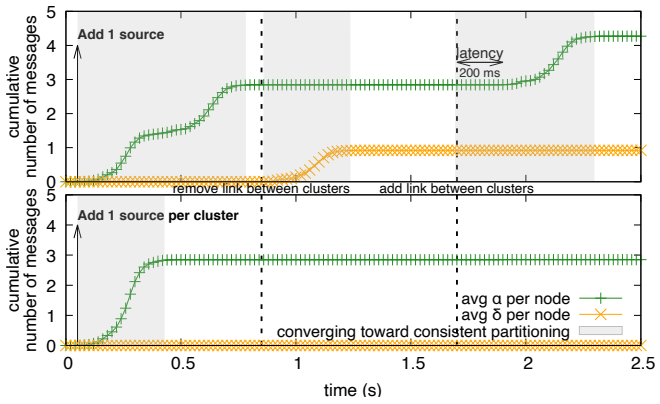


Fig. 7: Partitioning overhead of 2 clusters connected by a long distance link.

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 `Add`s may reach consistent partitioning faster. In particular, the top part of Figure 7 depicts a slow down in traffic around 300 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 AS-cast operates well even when *physical* partitions appear. Indeed, the disconnection of the inter-continental link existing between the two clusters either leads to (i) additional traffic when the nodes do not have any source in their cluster because nodes need to purge all indexes about the remote unreachable source until reaching consistent partitioning; or (ii) a status quo as nodes of each cluster already target the source in their respective cluster.

Finally, 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 inter-continental link latency, the cut off cluster starts to communicate α messages again. Eventually, all nodes belong to the same partition. However, in

Table 2: Related work summary.

Approaches	Without Third-party	Update	Eventually Consistent	Reactive or Cyclic
Centralized [1,2,17,33]	✗	one	✓	R
DHT [6,11,23]	✗	few	✓	R
Vector-based routing [21,32]	✓	all	✓	R/C
Replicated store [34]	✓	all	✓	R/C
Timeout-based [15,20]	✓	scope	✗	C
Random walks [36]	✓	scope	✓	C
Spanning Forest [4]	✓	scope	✓	C
This paper	✓	**scope**	✓	**R**

the experiment involving one source *per* cluster, the new link does not trigger noticeable traffic, for nodes already know their closest source in their cluster.

Overall, these experiments highlight the scalability of AS-cast in dynamic networks such as Edge infrastructures. Next Section reviews state-of-the-art approaches that can index content in geo-distributed infrastructures.

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. Table 2 summarizes the pros and cons of state-of-the-art approaches.

Third-party: Dedicated services 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 [1,2,17,33]. However, it comes with well-known issues such as load balancing, or single point of failure.
Distributing this index among nodes circumvents these issues [5,6,11,23,37,39], 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) [6,11,23], 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 [8,32]. 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. In a routing context, distance-vector routing protocols such as BGP or OSPF [32] similarly broadcast information to all other nodes to infer a topology map of the network and derive routing tables. 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) [15,20] and distributed clustering [36]. The sources advertise themselves periodically. Advertisements either stop as soon as they reach uninterested nodes [15,20], or propagate through most likely interested nodes [36]. However their cyclic operation requires that (i) they constantly generate traffic even in quiescent systems where nodes do not add nor destroy replicas; and (ii) they either operate in synchronous systems [4,36] or must define physical-clock-based timeouts the value of which depends on network topology properties such as network diameter [15,20]. 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 and dynamic networks with asynchronous communications; and has no overhead in quiescent systems.

6 Conclusion

With the advent of the Edge and IoT era, major distributed services proposed by our community should be revised to mitigate as much as possible traffic between nodes. In this paper, we addressed the challenge of content indexing as a dynamic logical partitioning problem where partitions grow and shrink to reflect content locations, and infrastructure changes. Using an efficient communication primitive called scoped broadcast, each node composing the infrastructure eventually knows the node hosting the closest replica of a 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 for Adaptive Scoped broadcast. Simulations confirmed that nodes quickly reach consistent partitioning together while the generated traffic remains locked down to partitions.

As future work, we plan to leverage the hierarchical properties of interconnected autonomous systems [31] to further limit the propagation of indexes within interested systems only. We expect that such an improvement would greatly benefit the autonomous systems, and particularly those hosting a large number of contents. Indeed, AS-cast ensures that every node eventually acknowledges its closest content replica. This feature unfortunately becomes an issue when each node is only interested by a small portion of content.

We also plan to evaluate our proposal within a concrete storage system such as InterPlanetary File System (IPFS) [6]. This would assess the relevance of AS-cast in real systems subject to high dynamics, and compare it against its current DHT-based indexing system that does not include distance in its operation. More generally, we claim that AS-cast and its extension can constitute novel building blocks for geo-distributed services. For instance, AS-cast could complement content delivery infrastructures [37] by efficiently sharing between nodes attached to the same CDN server the locations of web objects that have already been downloaded. This would further improve the containment of web traffic in our networks, and in turns, reduce the overall traffic footprint.

Acknowledgements Most of the material presented in this article such as our prototype are available on the STACK Research Group. Activities have been done within the framework of Inria/QarnotComputing DÃľfi, an initiative to *push carbon-neutral services towards the edge.*

References

1. Afanasyev, A., Yi, C., Wang, L., Zhang, B., Zhang, L.: SNAMP: Secure namespace mapping to scale NDN forwarding. In: IEEE Conference on Computer Communications Workshops (2015)
2. Aggarwal, V., Feldmann, A., Scheideler, C.: Can ISPS and P2P users cooperate for improved performance? SIGCOMM Comput. Commun. Rev. (2007)

3. Asrese, A., Eravuchira, S., Bajpai, V., Sarolahti, P., Ott, J.: Measuring web latency and rendering performance: Method, tools & longitudinal dataset. IEEE Transactions on Network and Service Management (2019)
4. Barjon, M., Casteigts, A., Chaumette, S., Johnen, C., Neggaz, Y.M.: Maintaining a spanning forest in highly dynamic networks: The synchronous case. In: International Conference On Principles Of Distributed Systems (2014)
5. Beaumont, O., Kermarrec, A.M., Marchal, L., Rivière, E.: VoroNet: A scalable object network based on Voronoi tessellations. In: IEEE International Parallel and Distributed Processing Symposium (2007)
6. Benet, J.: IPFS - content addressed, versioned, P2P file system. CoRR **abs/1407.3561** (2014)
7. Birman, K.P., Hayden, M., Ozkasap, O., Xiao, Z., Budiu, M., Minsky, Y.: Bimodal multicast. ACM Transactions on Computer Systems (1999)
8. Breitbart, Y., Garofalakis, M., Martin, C., Rastogi, R., Seshadri, S., Silberschatz, A.: Topology discovery in heterogeneous IP networks. In: IEEE INFOCOM. Conference on Computer Communications (2000)
9. Confais, B., Lebre, A., Parrein, B.: An object store service for a fog/edge computing infrastructure based on IPFS and a scale-out NAS. In: IEEE International Conference on Fog and Edge Computing (2017)
10. Confais, B., Lebre, A., Parrein, B.: Performance analysis of object store systems in a fog and edge computing infrastructure. In: Transactions on Large-Scale Data-and Knowledge-Centered Systems (2017)
11. D'Ambrosio, M., Dannewitz, C., Karl, H., Vercellone, V.: MDHT: A hierarchical name resolution service for information-centric networks. In: ACM SIGCOMM Workshop on Information-centric Networking (2011)
12. Doan, T.V., Pajevic, L., Bajpai, V., Ott, J.: Tracing the path to YouTube: A quantification of path lengths and latencies toward content caches. IEEE Communications Magazine (2019)
13. Drolia, U., Guo, K., Tan, J., Gandhi, R., Narasimhan, P.: Cachier: Edge-caching for recognition applications. In: IEEE 37th International Conference on Distributed Computing Systems (ICDCS) (2017)
14. Eugster, P.T., Guerraoui, R., A.M.Kermarrec, Massoulié, L.: Epidemic information dissemination in distributed systems. Computer (2004)
15. Garcia-Luna-Aceves, J.J., Martinez-Castillo, J.E., Menchaca-Mendez, R.: Routing to multi-instantiated destinations: Principles, practice, and applications. IEEE Transactions on Mobile Computing (2018)
16. Guo, P., Hu, B., Li, R., Hu, W.: FoggyCache: Cross-device approximate computation reuse. In: Annual International Conference on Mobile Computing and Networking. MobiCom (2018)
17. Gupta, H., Ramachandran, U.: Fogstore: A geo-distributed key-value store guaranteeing low latency for strongly consistent access. In: ACM Conference on Distributed and Event-based Systems. DEBS (2018)
18. Hadzilacos, V., Toueg, S.: A modular approach to fault-tolerant broadcasts and related problems. Tech. rep., USA (1994)
19. Hasenburg, J., Grambow, M., Bermbach, D.: Towards a replication service for data-intensive fog applications. In: ACM Symposium on Applied Computing (2020)
20. Hemmati, E., Garcia-Luna-Aceves, J.J.: A new approach to name-based link-state routing for information-centric networks. In: Proc. of the 2nd ACM Conference on Information-Centric Networking. ICN (2015)

21. Hoque, A.K.M.M., Amin, S.O., Alyyan, A., Zhang, B., Zhang, L., Wang, L.: NLSR: Named-data link state routing protocol. In: ACM SIGCOMM Workshop on Information-centric Networking (2013)
22. Hsiao, H.C., King, C.T.: Scoped broadcast in dynamic peer-to-peer networks. In: International Computer Software and Applications Conference (2005)
23. Iyer, S., Rowstron, A., Druschel, P.: Squirrel: A decentralized peer-to-peer web cache. In: Symposium on Principles of Distributed Computing. PODC (2002)
24. Jelasity, M., Babaoglu, O.: T-man: Gossip-based overlay topology management. In: Comput. Networks (2006)
25. Kermarrec, A.M., Taïani, F.: Want to scale in centralized systems? think P2P. Journal of Internet Services and Applications (2015)
26. Knight, S., Nguyen, H.X., Falkner, N., Bowden, R., Roughan, M.: The internet topology zoo. IEEE Journal on Selected Areas in Communications (2011)
27. Lamport, L.: Time, clocks, and the ordering of events in a distributed system. Commun. ACM **21**(7), 558âĂŞ565 (Jul 1978)
28. Lue, M.Y., King, C.T., Fang, H.: Scoped broadcast in structured P2P networks. In: Conference on Scalable Information Systems (2006)
29. Maymounkov, P., Mazières, D.: Kademlia: A peer-to-peer information system based on the xor metric. In: Peer-to-Peer Systems. Springer Berlin Heidelberg (2002)
30. Montresor, A., Jelasity, M.: PeerSim: A scalable P2P simulator. In: Proc. of the 9th International Conference on Peer-to-Peer (2009)
31. Nur, A.Y., Tozal, M.E.: Geography and routing in the internet. ACM Transactions on Spatial Algorithms and Systems **4**(4) (Sep 2018)
32. RFC2328: OSPF Version 2. `https://tools.ietf.org/html/rfc2328` (1998)
33. Seedorf, J., Kiesel, S., Stiemerling, M.: Traffic localization for P2P-applications: The ALTO approach. In: 2009 IEEE 9th International Conference on Peer-to-Peer Computing. pp. 171–177 (Sept 2009)
34. Shapiro, M., Preguiça, N., Baquero, C., Zawirski, M.: Conflict-free Replicated Data Types. Research Report RR-7687 (Jul 2011)
35. Shi, W., Cao, J., Zhang, Q., Li, Y., Xu, L.: Edge computing: Vision and challenges. IEEE Internet of Things Journal **3**(5), 637–646 (2016)
36. Sohier, D., Georgiadis, G., Ere, S., Papatriantafilou, M., Bui, A.: Physarum-inspired self-biased walkers for distributed clustering. In: International Conference On Principles Of Distributed Systems (2012)
37. Triukose, S., Wen, Z., Rabinovich, M.: Measuring a commercial content delivery network. In: Conference on World Wide Web (2011)
38. Wang, L., Bayhan, S., Ott, J., Kangasharju, J., Sathiaseelan, A., Crowcroft, J.: Pro-diluvian: Understanding scoped-flooding for content discovery in information-centric networking. In: ACM Conference on Information-Centric Networking (2015)
39. Xie, J., Qian, C., Guo, D., Wang, M., Shi, S., Chen, H.: Efficient indexing mechanism for unstructured data sharing systems in edge computing. In: IEEE Conference on Computer Communications (2019)