Technical Report COMP-010-2008
Computing Department
Lancaster University

On the convergent detection of crashed regions in overlay networks

Barry Porter, François Taïani, Geoff Coulson
Computing Department
Lancaster University, Lancaster, UK
{barry.porter, francois.taiani, geoff}@comp.lancs.ac.uk

Abstract

This report presents a distributed service that allows nodes in an overlay network to agree locally on the extent of crashed regions in their neighbourhood. Our service is local *is that it only involves nodes in the vicinity of a crashed region. It is* convergent *as nodes that propose overlapping and hence conflicting regions are forced to reconcile their views before deciding. In this report, we motivate the need for such a service, formally specify its properties, propose an implementation, and prove its correctness.*

Computing Department
Infolab21, South Drive
Lancaster University
LANCASTER LA1 4WA
United Kingdom
Tl.: +44 (0) 1524 51 03 38
mail: francois.taiani@comp.lancs.ac.uk

1 Introduction

Overlay networks [9] have been proposed as a powerful mechanism to realise advanced large-scale services such as DHTs, stream-broadcasting, and file sharing, in ways that are usually both highly flexible, self-organising, and resilient [17, 2, 18, 6, 8].

Most of these approaches, however, assume that neighbouring nodes in the overlay topology do not share common failure modes, i.e. ignore so-called *correlated* failures. The argument is often that the overlay topology uses some randomising mechanism that maximises the diversity between topologically close nodes (see for instance [11]).

However, this might not always be true, particularly as more works consider optimisations that leverage knowledge about lower networking layers (e.g. cross-layer optimisation), for instance by positioning overlay neighbours on machines that are physically close in the underlying physical network. This diversity assumption may even cease to apply completely in wireless sensor networks and mobile ad-hoc systems, in which connected regions of the overlay are likely to disappear as the underlying physical system fails due to external hazards (flooding, fire, earthquake, terrorist attack) [15].

In this article we look at the particular problem of detecting such correlated crashed regions. Our premise is that the system can benefit from a collective response to the emergence of crashed regions, so that there is a need for the nodes in the locality of a crashed region to come to an agreement on the shape and extent of these regions.

Compared with traditional consensus, this form of agreement presents two key challenges: (i) the solution should be scalable, i.e. it should only involve nodes in the vicinity of a crashed region; (ii) because of ongoing crashes, nodes might disagree on the extent of a crashed region, but as they do so they'll also disagree on *who* should even take part in the agreement, leading to possibly competing proposals, and the needs for arbitration and convergence of views.

Contributions: In this article, we formally specify the convergent detection of crashed regions, present a solution that uses perfect failure detectors, and prove its correctness. The protocol itself was first outlined in an earlier work as part of a larger repair mechanism [16] but this is the first time that we provide a formal specification of the problem, along with a precise description of the protocol, and present a formal proof of its correctness.

Report organisation: We first specify the problem of the convergent detection of crashed regions in overlay networks in Section 2, then move on to describe our solution (Section 3). In Section 4, we present our proof of correctness, and finish with some related work (Section 5) and a conclusion.

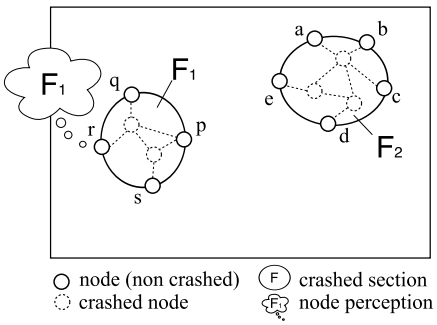
2 The problem

2.1 Overview

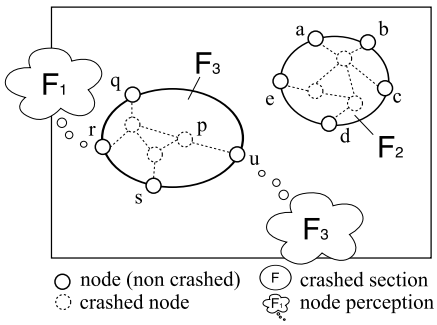
Figure 1-(a) shows an example of an overlay network in which the nodes in region F_1 and F_2 have crashed, and these crashes are being detected by the respective *border nodes* (i.e. the neighbouring nodes) of each crashed region: $\{p, q, r, s\}$ for F_1 and $\{a, b, c, d, e\}$ for F_2.

We're interested in a solution that allows each border set to reach an agreement about the extent of their crashed region. For scalability reasons, we'd also like the two agreements (on F_1 and F_2) only to involve neighbouring nodes.

Because crashes are ongoing, crashed regions might grow while an agreement is being attempted, possibly leading to conflicting views about the same network area. In Figure 1-(b), for instance, p fails after r has detected F_1 as crashed, but before an agreement on F_1 has been reached. Region F_1 thus grows into F_3, and node u, p's still non-crashed neighbour, becomes involved. u detects the entirety of F_3 as crashed. Because the views of u and r overlap, but each node has different



(a) Two disjoint regions of the network have crashed.



(b) Because of ongoing crashes, region F_1 has grown into F_3, and nodes r and u now have conflicting views of the extent of their crashed region.

Figure 1. Protocol instances and conflicting views

local perceptions of the state of the network, they could possibly reach diverging conclusions on what must be recovered and how. Our protocol should prevent this, and insure that any decisions pertaining to the same part of the network *converge* to a unified view.

2.2 System model and assumptions

We model an overlay network as a undirected graph $\mathcal{G} = (\Pi, \mathrm{E})$ of asynchronous message-passing nodes $\Pi = \{p_1, p_2, ..\}$, where $\mathcal{G}$ represents the overlay topology. A node is *faulty* if it crashes at some point, *correct* if it never crashes. Any two nodes might exchange messages through asynchronous, reliable, and ordered (fifo) channels. We also assume that each node knows $\mathcal{G}$ (e.g. in a real system, that each node can reconstruct $\mathcal{G}$ on demand).

The *border* of a node p is the set of p's neighbours: $\mathsf{border}(p) = \{q \in \Pi \mid (p, q) \in \mathrm{E}\}$. By extension, the border of a subset $S \subseteq \Pi$ of nodes is the set of nodes who have a neighbour in S but do not belong to S: $\mathsf{border}(S) = \{q \in \Pi \backslash S \mid \exists p \in S : (p, q) \in \mathrm{E}\}$. If $p \in \mathsf{border}(S)$, we say that p is a *border node* of S or that p *borders* S.

A *region* is a subset R of nodes such that the subgraph $\mathcal{G}[R]$ induced by R in $\mathcal{G}$ is connected. A *crashed region* at a time t, is a region in which all nodes have crashed. A *faulty component* is a region in which all nodes are faulty, but whose border nodes are all correct. Note that by construction, two faulty components can only be either equal or disjoint (they cannot partially overlap).

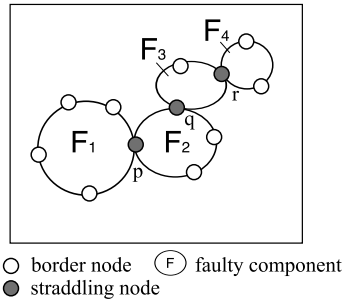


Figure 2. A cluster of adjacent faulty components

Two regions R and T are *adjacent* if their borders intersect: $\mathsf{border}(R) \cap \mathsf{border}(S) \neq \emptyset$. Note that adjacency is a reflective and symmetric relation. We call *fault-adjacency* the *restriction* of the adjacency relation to faulty components, and for two faulty components R and S we note

$\mathsf{clustered}(S, R)$, the reflective transitive closure of the *fault-adjacency* relation. We call *faulty cluster* an equivalent class of this closure relation. Two regions R and T are *adjacent* if their borders 'touch', and a *faulty cluster* is a maximal set of faulty components that 'transitively' touch each other's border. (Figure 2)

Although $\mathcal{G}$ and the number of faulty clusters might be infinite, we assume $\mathcal{G}$ is locally finite (i.e. every node has a finite number of neighbours), that each faulty component contains a finite number of nodes, and that each faulty cluster contains a finite number of faulty components.

2.3 Convergent detection of crashed regions: specification

Operations

We use a mono-threaded event-based programming model, as adopted by Guerraoui and Rodrigues in [13], to specify the convergent detection of crashed regions, and present our solution. Our service starts when a node detects one of its neighbours q as crashed ($\langle \mathbf{crash} \mid q \rangle$ event). It stops by raising a $\langle \mathbf{decide} \mid S \rangle$ event, where S is the crashed region decided by the local node. We call S the *view* of the deciding node.

Properties

A protocol implementing the convergent detection of crashed regions should fulfil the following properties, which capture the problem that we described informally in Section 2.1.

CD1 (Integrity) No node decides twice.

CD2 (View Accuracy) If a node p decides S, then $p \in \mathsf{border}(S)$, and S is a crashed region.

CD3 (View Convergence) If two correct nodes decide R and S, $(R \cap S \neq \emptyset) \Rightarrow (R = S)$.

CD4 (Uniform Border Agreement) :
If two nodes p and q decide, and p decides S, and $q \in \mathsf{border}(S)$, then q decides S.

CD5 (Border Termination) If a node p decides S, then all correct nodes in $\mathsf{border}(S)$ eventually decide.

CD6 (Progress) In each faulty cluster, at least one correct node bordering a faulty component in the cluster eventually decides: $\forall S, S$ is a faulty component : $\exists R, p : R$ is a faulty component $\wedge\, \mathsf{clustered}(S, R) \wedge p \in \mathsf{border}(R) \wedge p$ decides.

CD7 (Locality) Communication is limited to faulty-components and their borders, i.e. a node p only exchanges messages with a node q if there is a faulty component S such that $\{p, q\} \subseteq S \cup \mathsf{border}(S)$.

Note that CD7 implies that nodes with no faulty neighbours do not take part in the protocol.

3 A protocol for the convergent detection of crashed regions

3.1 Dependencies: Failure detector, best effort broadcast, region ranking

Our algorithm uses a perfect failure detector, provided in the form of a *subscription-based* service: a node p subscribes to the crashes of a subset of nodes S by issuing the event $\langle \mathbf{monitorCrash} \mid S \rangle$ to its local failure detector. Our subscription-based failure detector is perfect and insures: *(i) Strong Accuracy*: if a node p receives a $\langle \mathbf{crash} \mid q \rangle$, then q has crashed, and p did subscribe to be notified of q's crash; and *(ii) Strong Completeness*: if a node q has crashed, and p has subscribed to be notified of q's crash, then p will eventually receive a $\langle \mathbf{crash} \mid q \rangle$ event.

For compactness, our implementation uses a best effort broadcast service [13], represented by the events $\langle \mathbf{bebBroadcast} \mid R, [\mathrm{m}] \rangle$ and $\langle \mathbf{bebDeliver} \mid s, [\mathrm{m}] \rangle$. This service simply loops through all recipients and sends them the message through the underlying 1-1 network. We assume a node receives the broadcast messages of a given sender in the order in which they were sent (i.e. the broadcast mechanism maintains the fifo property of the underlying channels).

We also use the following ranking relation between regions, denoted $\succ$: $R \succ S$ iff R and S are regions, and either (i) R contains more nodes than S, or (ii) they contain the same number of nodes but R's border contains more nodes than S's border , or (iii) R and S have the same size, and so do their respective borders, but R is greater than S according to some strict total order relation $\rhd$ on sets of nodes. The actual ordering relation $\rhd$ on node sets does not matter. One possibility is to use a lexicographic order on node IDs. The construction of $\succ$ is akin to that of a lexicographic order on $\mathbb{N} \times \mathbb{N} \times \mathcal{P}(\Pi)$ using the order $>$ on $\mathbb{N}$ and $\rhd$ on $\mathcal{P}(\Pi)$, and thus can be shown to be a strict partial order on regions. For a set $\mathcal{C}$ of regions, we define $\mathsf{maxRankedRegion}(\mathcal{C})$ as the highest ranked region in $\mathcal{C}$ according to $\succ$.

Finally, for a subset S of nodes, we define $\mathsf{connectedComponents}(S)$ as the set of the maximal regions that make up S, i.e.—formally—as the vertex sets of the connected components of the subgraph $\mathcal{G}[S]$ induced by S in $\mathcal{G}$.

3.2 Overview

The pseudo code of our algorithm is given in Figure 3. $\langle \mathbf{init} \rangle$ is implicitly executed by all nodes when the protocol starts. Each node then remains idle until one of its neighbours fails, as notified by a $\langle \mathbf{crash} \mid q \rangle$ event.

The bulk of the protocol is primarily a superposition of flooding uniform consensus instances (e.g. a simplified version of the consensus algorithm for strong failure detectors presented by Chandra et al [5]; our presentation follows closely the one proposed in [13]) between the border nodes of proposed views. This superposition is complemented by an arbitrating mechanism to deal with overlapping but conflicting views (line 30). Because of this arbitration, all consensus instances must be dealt with explicitly in the protocol, rather than as separate instances, and variables such as $\mathrm{opinions}[.][.][.]$ and $\mathrm{waiting}[.][.]$ are indexed by proposed views (in addition to rounds, and, for $\mathrm{opinions}$, participants).

A node starts a consensus instance when it detects that it sits on the border of a crashed region (line 19). The view proposed by a node has been incrementally built up when receiving $\langle \mathbf{crash} \mid . \rangle$ events (line 6). The advertised view myView is the highest ranked crashed region known to the node at this point. The view construction continues as the consensus unfolds, to be used as a new proposed view in case the attempt to reach an agreement fails.

The opinion vectors received from other nodes in a round are gathered at line 21. However, because a node might be involved simultaneously in multiple conflicting consensus instances, messages related to conflicting views are also gathered and processed. The resulting opinion vectors, indexed by round and proposed view (line 27) are stored in $\mathrm{opinions}[.][.][.]$. Lines 22-26 dynamically initialise $\mathrm{opinions}[.][.][.]$ and $\mathrm{waiting}[.][.]$ on demand.

If a node becomes aware of a conflicting view with a lower rank (line 30), it sends a special reject vector to this view's border nodes, and subsequently ignores any message related to this view.

Rounds are completed in the last event (line 45) when all non-crashed border nodes of myView have replied: if no more rounds are needed (line 39), the local node has completed its current consensus instance, and the proposed view is decided upon if its final vector only contains accept, otherwise the whole process is reset (line 43), and restarts at line 15 with a new consensus instance as soon as a new crashed node is detected on the border of $\mathrm{locallyCrashed}$.

1: **upon event** $\langle$ **init** $\rangle$
2: decided $\leftarrow \perp$; newCandidate $\leftarrow$ FALSE ; proposed $\leftarrow$ FALSE
3: locallyCrashed, candidateView, myView, received, rejected $\leftarrow \emptyset$
4: **trigger** $\langle$ **monitorCrash** | border(p) $\rangle$
5: **end event**

6: **upon event** $\langle$ **crash** | $q\rangle$ ▷ View construction
7: locallyCrashed $\leftarrow$ locallyCrashed $\cup \{q\}$
8: **trigger** $\langle$ **monitorCrash** | border(q)\locallyCrashed$\rangle$
9: $\mathcal{C}$ = connectedComponents(locallyCrashed)
10: **if** candidateView $\prec$ maxRankedRegion($\mathcal{C}$) **then**
11: candidateView = maxRankedRegion($\mathcal{C}$)
12: newCandidate $\leftarrow$ TRUE
13: **end if**
14: **end event**

15: **upon event** proposed = FALSE $\wedge$ newCandidate = TRUE ▷ New consensus instance
16: myView $\leftarrow$ candidateView ; newCandidate $\leftarrow$ FALSE ; proposed = TRUE
17: $V_{\text{accept}}[p_k] \leftarrow \perp$ **for all** $p_k \in$ border(myView)\{self}
18: V_{accept}[self] $\leftarrow$ accept ; round $\leftarrow 1$
19: **trigger** $\langle$ **bebBroadcast** | border(myView), $[1, \text{myView}, \text{border}(\text{myView}), V_{\text{accept}}]\rangle$ ▷ myView is being proposed
20: **end event**

21: **upon event** $\langle$ **bebDeliver** | p_i,$[r, S, B, V]\rangle \wedge S \notin$ rejected ▷ Updating my opinion array
22: **if** $S \notin$ received **then**
23: received $\leftarrow$ received $\cup \{S\}$
24: opinions$[S][r][p_k] \leftarrow \perp$ **for all** $p_k \in B \wedge 1 \leq r < |B|$
25: waiting$[S][r] \leftarrow B$ **for all** $1 \leq r < |B|$
26: **end if**
27: **for all** p_k **such that** (opinions$[S][r][p_k] = \perp \wedge V[p_k] \neq \perp$) **do** opinions$[S][r][p_k] \leftarrow V[p_k]$
28: waiting$[S][r] \leftarrow$ waiting$[S][r]\setminus (\{p_i\} \cup \{p_k | V[p_k] =$ reject$\})$
29: **end event**

30: **upon event** $\exists L \in$ received : $L \prec$ myView ▷ Rejecting a lower ranked view
31: $V_{\text{reject}}[p_k] \leftarrow \perp$ **for all** $p_k \in$ border(L)\{self}
32: V_{reject}[self] $\leftarrow$ reject
33: received $\leftarrow$ received\{L}
34: rejected $\leftarrow$ rejected $\cup \{L\}$
35: **trigger** $\langle$ **bebBroadcast** | border(L), $[1, L, \text{border}(L), V_{\text{reject}}]\rangle$
36: **end event**

37: **upon event** myView $\in$ received $\wedge$ waiting[myView][round]\locallyCrashed $= \emptyset \wedge$ decided $= \perp$
38: **if** round = |border(myView)| **then** ▷ Consensus instance completed
39: **if** $\forall p_i \in$ border(myView) : opinions[myView][round][p_i] = accept **then**
40: decided $\leftarrow$ myView ▷ Consensus succeeded, decision
41: **trigger** $\langle$ **decided** | myView$\rangle$
42: **else**
43: proposed $\leftarrow$ FALSE ▷ Consensus attempt failed, reset
44: **end if**
45: **else** ▷ New round
46: round $\leftarrow$ round + 1
47: **trigger** $\langle$ **bebBroadcast** | border(myView), [round, myView, border(myView), opinions[myView][round−1]]$\rangle$
48: **end if**
49: **end event**

Figure 3. Convergent detection of crashed regions

4 Proof of correctness

CD1 is fulfilled by construction. For CD2, connectedComponents at line 9 and the strong accuracy of the failure detector insure that proposed views are crashed regions. Using recursion on $\langle \mathbf{crash} \mid . \rangle$ events, a node p can be shown to respect the two invariants *(i)* $p \in \mathsf{border}(\mathrm{locallyCrashed}_p)$ and *(ii)* $\{p\} \cup \mathrm{locallyCrashed}_p$ is connected, thus yielding that p is on the border of any view it proposes. CD7 follows from CD2, and the fact that two nodes only exchange messages when on the border of a region detected as crashed by one of them.

Our proof of the remaining four properties reuses elements of the proof of the consensus algorithm presented in [5] for strong failure detectors (S), of which the flooding uniform consensus is derived. The difficulty lies in that our protocol uses multiple overlapping consensus instances, each indexed by the view they propose, and that there is no prior agreement on the consensus instances that need to be run, neither in terms of their participants, nor in terms of their sequence. Instead, a node initiates new consensus instances depending on the perception delivered by its local failure detector (line 19).

In addition, our arbitrating mechanism means a node can first propose and then reject the same view, thus complicating the *uniform border agreement*, as we shall see.

Our proof uses the happened-before relation $\rightarrow$ defined by Lamport (with the slight change that we use broadcast communication rather than point-to-point messages). Our happened-before relation is defined over *execution points*: an execution point e is the point in time when a given node initiates the execution of one of the lines of the protocol[1]. If $e \rightarrow d$ we say that e was executed *at some earlier point* than d.

When a node p executes $\langle \mathbf{bebBroadcast} \mid \mathsf{border}(S), [1, S, \mathsf{border}(S), V_{\mathsf{accept}}] \rangle$ at line 19 we say that p *proposes* S. Similarly when p executes $\langle \mathbf{bebBroadcast} \mid \mathsf{border}(L), [1, L, \mathsf{border}(L), V_{\mathsf{reject}}] \rangle$ at line 35 we say that p *rejects* L.

Finally, we use a subscript notation to distinguish between the same protocol variable at different nodes: e.g. myView_p denotes the variable myView used by node p.

4.1 Lemma 1

At any execution point the vectors $\mathrm{opinions}_p[S][r][.]$ *maintained by a node* p *are such that* $\forall q \in \mathsf{border}(S)$:

i) $\mathrm{opinions}_p[S][r][q] = \mathsf{accept} \Rightarrow q$ *accepted* S *at some earlier point* $\wedge$

ii) $\mathrm{opinions}_p[S][r][q] = \mathsf{reject} \Rightarrow q$ *rejected* S *at some earlier point*

This lemma follows from a recursive data-flow argument on the values of opinions[S][r][.], the properties of the best-effort broadcast and the transitivity of the happened-before relation.

4.2 Lemma 2

A node proposes (resp. rejects) a given view S *at most once. A node never proposes a view it has previously rejected.*

The uniqueness of rejection follows from the use of the $\mathrm{rejected}$ and $\mathrm{received}$ variables. The use of the strict ranking relation $\prec$ at line 10 means the series of values taken by $\mathrm{candidateView}$ at a given node is strictly monotonic according to $\prec$, and by construction that this is also true of myView, thus completing the lemma.

[1] Execution points are often called events in the literature, but we could not reuse the term here.

4.3 Lemma 3

If two nodes p and q complete a consensus instance at line 39 with $\text{myView}_p = \text{myView}_q = S$ *they obtain the same opinion vector:*

$$\text{opinions}_p[S][N][.] = \text{opinions}_q[S][N][.] \textit{ where } N = |\mathsf{border}(S)|$$

We prove this lemma by contradiction. Let's assume $\exists k \in \mathsf{border}(S) : \text{opinions}_p[S][N][k] \neq \text{opinions}_q[S][N][k]$.

The first case, where one of the two values is $\perp$, uses the well-known argument on cascading node crashes, identifying $N-1$ distinct nodes in $\mathsf{border}(S)$ that did not complete the consensus instance, which contradicts the fact that p and q did complete it.

Let's now assume both values are non-$\perp$, e.g. without loss of generality, $\text{opinions}_p[S][N][k] = \mathsf{accept}$ and $\text{opinions}_q[S][N][k] = \mathsf{reject}$. From lemma 1 we conclude that k both proposed and rejected S, both at some earlier point. Let's call e^k_{accept} and e^k_{reject} the corresponding execution points, and accept^k_S and reject^k_S the corresponding messages. Because of lemma 2, e^k_{accept} and e^k_{reject} are unique, and $e^k_{\mathsf{accept}} \rightarrow e^k_{\mathsf{reject}}$.

Because the best-effort broadcast is fifo, does not create messages, and events are processed in the order they are raised, we conclude that on any process $r \in \mathsf{border}(S)$ the reject value is always preempted by an accept, and hence $\forall r \in \mathsf{border}(S) : \text{opinions}_r[S][1][k] \in \{\perp, \mathsf{accept}\}$. A recursive data-flow argument similar to that of lemma 1, leads to $\text{opinions}_q[S][N][k] \in \{\perp, \mathsf{accept}\}$, and yields the contradiction.

4.4 Uniform border agreement (CD4) & Border termination (CD5)

Let's assume p and q decide, p decides S, and $q \in \mathsf{border}(S)$. If p decides S, then p completed the corresponding consensus instance with only accept values, and since $q \in \mathsf{border}(S)$ we have $\text{opinions}_p[S][N][q] = \mathsf{accept}$. By lemma 1, q proposed S. Since by construction a node *(i)* cannot propose any new view once it has decided on one, and *(ii)* cannot start a new consensus instance before completing the current one, q proposed S and completed the corresponding consensus instance before deciding. By lemma 3, q obtained the same vector as p on S, and hence decided S, thus proving CD4.

CD5 follows the same line, with the observation that if a node p completes a consensus instance on a view S, then all other nodes in $\mathsf{border}(S)$ either took part in each round or crashed, implying that all correct nodes eventually complete the instance with the same opinion vector as p (by way of lemma 3).

4.5 View convergence (CD3)

Let's consider two correct nodes p and q that decide on overlapping crashed regions S_p and S_q: $S_p \cap S_q \neq \emptyset$. If one node is in the border set of the other's region (e.g. $p \in \mathsf{border}(S_q)$), then *Uniform Border Agreement* (CD4) and *Integrity* (CD1) give us $S_p = S_q$.

Let's now assume $p \notin \mathsf{border}(S_q) \wedge q \notin \mathsf{border}(S_p)$, and use a proof by contradiction. Since $S_p \cap S_q \neq \emptyset$, there exists a node $a \in S_p \cap S_q$ (Figure 4). S_p being a region bordered by p (CD2), there exists a path $(n_0 = p, n_1, n_2, ..., n_i, ..., n_{k-1}, n_k = a)$ that links a to p through S_p: $\{n_1, ..., n_k, a\} \subseteq S_p$. Since $a \in S_q$, we can consider the point when this path "penetrates" for the first time into q's crashed region, i.e. we can consider the first node in this path n_{i_0} that belongs to S_q ($n_{i_0} \in S_q$ and $\forall i < i_0 : n_i \notin S_q$). Since p is correct, $n_{i_0} \neq p$, i.e. $i_0 \geq 1$, and we can look at n_{i_0-1}, the node in the path just before n_{i_0}. Let's call this node r (see Figure 4). Because n_{i_0} is the first node in the path to belong to S_q, we have $r \in \mathsf{border}(S_q)$, and since $p \notin \mathsf{border}(S_q)$, $r = n_{i_0-1}$ cannot be p ($i_0 > 1$). Because, with the exception of p, the path connecting p to a

is embedded in S_p, this means that r is in fact located in p's crashed region. This reasoning thus yields us a node (r) that is both on $\mathsf{border}(S_q)$ *and* in p's crashed region: $r \in S_p \cap \mathsf{border}(S_q)$. Using an identical argument, we can find a node s such that $s \in S_q \cap \mathsf{border}(S_p)$ (Figure 4).

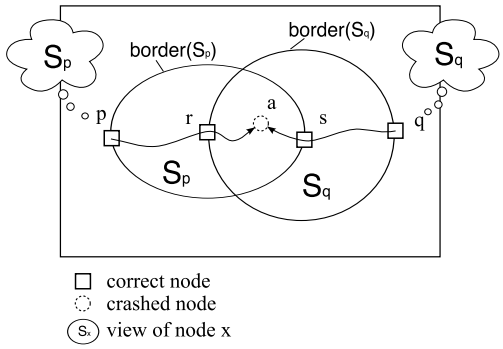


Figure 4. Convergence between overlapping views

To complete our proof by contradiction we now look at the happen-before relationships between events related to the nodes r and s.

Let's first consider node s. Since $s \in \mathsf{border}(S_p)$ and p decided on S_p, s itself did propose S_p (lemma 1). Since $r \in S_p$, s did detect r as crashed as some point. By a similar reasoning, we can conclude that r did propose S_q, and hence detected s as crashed as some point.

We thus end up with a set of 6 events that form a circular chain of happen-before events: $s\_detects\_r \rightarrow s\_proposes \rightarrow s\_crashes \rightarrow r\_detects\_s \rightarrow r\_proposes \rightarrow r\_crashes \rightarrow s\_detects\_r$ ... This provides our contradiction.

4.6 Progress (CD6)

Again we use a contradiction: consider a cluster of adjacent faulty components (Figure 2), and assume none of its correct border nodes ever decide. Since this situation lasts indefinitely, we can consider the case where all crashed regions are maximal and all remaining nodes are correct.

Because the views proposed by a node are strictly monotonic according to $\prec$, and we have assumed faulty components and faulty clusters are both finite, a node cannot propose an infinite sequence of views. A correct border node p that does not decide falls therefore into two cases: either (**C1**) p is blocked in a consensus round waiting indefinitely for the reply of another node q (line 37); or (**C2**) the last view proposed by p failed (line 43), and p does not detect any new crashed node (line 6).

Case C1 : If p is indefinitely blocked in a consensus round waiting for the reply of some other node q, q must be correct (if it were not, q would eventually crash, which would eventually be picked up by the failure detector, thus unblocking p). Since there's a path of crashed nodes from p to q (since p is waiting for q), q is on the border of the same faulty component as p, so q never decides (by assumption).

As for p, q falls in either case **C1** or **C2**. Let's first assume that the last view S_q^{max} proposed by q failed, and q does not detect any new crashed node (**C2**). Since we've assumed that all faulty nodes have crashed, by strong completeness of the failure detector, S_q^{max} is a faulty component, and because of the use of maxRankedRegion (line 11) and the fact that $\prec$ subsumes set inclusion, S_q^{max} is higher ranked than any crashed region bordered by q.

Since p is waiting for q, $S_p \neq S_q^{\mathsf{max}}$, and since q is on the border of both S_p and S_q^{max}, S_p is lower-ranked than S_q^{max}: $S_p \prec S_q^{\mathsf{max}}$. q has received a round-1 message proposing S_p (line 21), and should have rejected it (line 35), thus ending p's wait on q, which contradicts our assumption.

We therefore conclude that q cannot fall in case **C2**, and instead is blocked in a consensus round proposing a crashed region S_q (case **C1**). q received p's proposal message, and did consider it for

rejection (line 30). Because p is waiting for q, we know it did not receive any rejection message from q, and therefore, $S_p \succeq S_q$. Since p is waiting for q, q is not proposing the view as p, yielding a strict ordering between the two views $S_p \succ S_q$.

This construction can be repeated recursively, first for q, and then for the node q is waiting on, etc, each time yielding an infinite number of pairwise distinct crashed regions (via CD2) that are strictly ordered by the ranking relationship: $S_{p_1} \succ S_{p_2} \succ ... \succ S_{p_i} \succ ...$ This contradicts our assumption that each faulty cluster contains a finite number of faulty components, each containing a finite number of nodes.

Case C2 : Let's now assume the last view S_p^{max} proposed by p failed, and p does not detect any new crashed node. As above S_p^{max} is a faulty component, and all its border nodes are correct. Because the failure detector is strongly accurate, for p's proposal to fail, one node $q \in \mathsf{border}(S_p^{\mathsf{max}})$ must have rejected S_p^{max} because it was proposing an higher-ranked view. By assumption, q never decides, so must either fall in case **C1** or **C2**. If the former, we're back to the case above. If **C2**, q's last view S_q^{max} is higher than any view q ever proposed, implying $S_p^{\mathsf{max}} \prec S_q^{\mathsf{max}}$.

By recursively applying this argument, we either come back to case **C1** at some point, or obtain an infinite sequence of strictly ordered faulty components $S_{p_1}^{\mathsf{max}} \prec S_{p_2}^{\mathsf{max}} \prec S_{p_3}^{\mathsf{max}} \prec ...$, with the same kind of contradiction as in case **C1** above, which concludes our proof by contradiction.

5 Related work

As already mentionned this work is strongly related to that of Chandra and Toueg in [5].

Our work has some similarities with consensus with unknown participants, where the set of participants is fixed, but unknown to the nodes involved [12, 4, 3]. These works introduce the notion of a participant detector (PD) and study the properties this detector should fulfil to permit consensus under different assumptions.

These works are however quite different from what we are proposing, in that in our case participants are not only unknown, but evolve as failures occur. Our work also puts a strong focus on scalability with the *locality* property.

The service we propose is also related to group membership [7]. Deciding on a view in our protocol can be seen as the equivalent of installing a view. The link is particularly true with partitionable group membership (PGM) services [14, 10, 1], which look at how successively installed views should evolve to insure that both reachability and unreachability between nodes are reflected in their installed views.

As in partitionable group membership, our service requires views held by nodes to converge when these nodes enter a particular relationship. This relationship is defined in terms of reachability in PGM, while ours arise when two nodes propose views that overlap (CD3).

The key difference however is that, whereas PGM services are defined in terms of eventual convergence of installed views, our specification is stricter in that nodes can only decide once (CD1), and must therefore detect when they have reached a convergent state, while insuring liveliness in the system (CD6).

6 Conclusion

In this report we have formally specified a service for the convergent detection of crashed regions, where the nodes of an overlay attempt to reconcile their views of neigbouring crashed regions. We have described a fault-tolerant solution to this problem, and proved its correctness. One key aspect of our specification is that it only involves nodes bordering a crashed region (*locality*), and requires nodes to explicit decide when they've converged on a unified view.

This form of agreement could be seen as a particular case of a wider class of algorithms that

attempt to create local collective knowledge about some distributed condition in a scalable manner. Being crashed could for instance be seen as a particular case of a stable property, and this work might be extensible first to stable predicates (say a particular stable state), and then possibly, as in the fail-recovery model, to unstable properties.

References

[1] O. Babaoglu, R. Davoli, and A. Montresor. Group communication in partitionable systems: Specification and algorithms. Technical Report UBLCS-98-01, University of Bologna, 1998.

[2] M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron. SCRIBE: A large-scale and decentralized application-level multicast infrastructure. *IEEE Journal on Selected Areas in communications (JSAC)*, 2002.

[3] D. Cavin, Y. Sasson, and A. Schiper. Consensus with unknown participants or fundamental self-organization. In *Proc. of the 3rd Int. Conf. on AD-NOC Networks & Wireless (ADHOC-NOW'04)*, pages 135–148, Vancouver, 2004.

[4] D. Cavin, Y. Sasson, and A. Schiper. Reaching agreement with unknown participants in mobile self-organized networks in spite of process crashes. Research Report IC/2005/026, EPFL, 2005.

[5] T. D. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2):225–267, 1996.

[6] Y. Chawathe, S. McCanne, and E. A. Brewer. RMX: Reliable multicast for heterogeneous networks. In *INFOCOM*, pages 795–804, Tel Aviv, Israel, March 2000. IEEE.

[7] G. Chockler, I. Keidar, and R. Vitenberg. Group communication specifications: a comprehensive study. *ACM Computing Surveys*, 33(4):427–469, 2001.

[8] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. Freenet: a distributed anonymous information storage and retrieval system. In *International workshop on Designing privacy enhancing technologies*, pages 46–66. Springer-Verlag New York, Inc., 2001.

[9] D. Doval and D. O'Mahony. Overlay networks: A scalable alternative for p2p. *IEEE Internet Comp.*, 7(4):79–82, 2003.

[10] A. Fekete, N. Lynch, and A. Shvartsman. Specifying and using a partitionable group communication service. *ACM Trans. Comput. Syst.*, 19(2):171–216, 2001.

[11] A. J. Ganesh, A.-M. Kermarrec, and L. Massoulié. SCAMP: Peer-to-peer lightweight membership service for large-scale group communication. In *Proceedings of the 3rd International workshop on Networked Group Communication*, 2001.

[12] F. Greve and S. Tixeuil. Knowledge connectivity vs. synchrony requirements for fault-tolerant agreement in unknown networks. In *DSN '07: Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 82–91, Edinmburg, UK, 2007.

[13] R. Guerraoui and L. Rodrigues. *Introduction to Reliable Distributed Programming*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[14] I. Keidar, J. Sussman, K. Marzullo, and D. Dolev. Moshe: A group membership service for wans. *ACM Trans. Comput. Syst.*, 20(3):191–238, 2002.

[15] B. Porter, G. Coulson, and D. Hughes. Intelligent dependability services for overlay networks. In *Proceedings of Distributed Applications and Interoperable Systems 2006 (DAIS'06)*, volume 4025 of *LNCS*, pages 199–212, 2006.

[16] B. Porter, F. Taïani, and G. Coulson. Generalised repair for overlay networks. In *Proceedings of the 25th IEEE Symposium on Reliable Distributed Systems (SRDS 2006)*, pages 132–14, 2006.

[17] A. I. T. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware '01: Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms*, pages 329–350. Springer-Verlag, 2001.

[18] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, UC Berkeley, April 2001.