[12] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, 1978.

[13] F. Mattern. Virtual time and global states of distributed systems. In *Proceedings of the Workshop on Parallel and Distributed Algorithms, Elsevier,*, pages 215–226, 1989.

[14] L.L. Peterson, N.C. Buchholz, and Schliching R.D. Preserving and using context information in interprocess communication. *ACM Transactions on Computer Systems*, 7(3):217–246, August 1989.

identified by the relation $\stackrel{mo}{\rightarrow}$ which is defined next. It must be noted that this is a minor limitation since it is offset by the reduced overheads (each message carries 1 bounded integer as opposed to $2N^2$) in the cheaper implementation proposed.

Definition 5 [**"$\stackrel{mo}{\rightarrow}$", to be read as "actual message ordering permitted"**]

Define $g \stackrel{mo}{\rightarrow} g'$ *iff* $gsend(g) \stackrel{lam}{\rightarrow} gsend(g') \bigwedge ((g.typ=GB \bigvee g.typ=GT) \bigvee (g'.typ=GF \bigvee g'.typ=GT))$.

Note that $\stackrel{n*}{\rightarrow} \subseteq \stackrel{mo}{\rightarrow}$.

To summarize, we have proposed an F-channel based implementation of the GS primitive, which has been found to be useful in a variety of problem domains [5], e.g., termination detection, consistent cuts, global snapshots, shared token implementation, replicated data updates, mutual exclusion, etc.. The proposed implementation requires each message to carry only one bounded integer to enforce the message receipt ordering.

References

[1] M. Ahuja. *Flush* primitives for asynchronous distributed systems. *Information Processing Letters*, 34(2):5–12, February 1990.

[2] Alain Sandoz Andre Schiper, Jorge Eggli. A new algorithm to implement causal ordering. *Proc 3rd Int Workshop on Distributed Algorithms, Nice, France*, Springer Verlag LNCS 392:219–232, Sept 1989.

[3] K.P. Birman and T.A. Joseph. Reliable communication in the presence of failures. *ACM Transactions on Computer Systems*, 5(1):47–76, February 1987.

[4] Birrel and Nelson. Implementing remote procedure calls. *ACM Transactions on Computer Systems*, 2.1, Feb 1984.

[5] A. Gahlot M. Ahuja T. Carlson. Global flush primitive for sending a message to a group of processes. *Submitted for publication*, 1992.

[6] K. M. Chandy and L. Lamport. Distributed snapshots: Determining global states of distributed systems. *ACM Transactions on Computer Systems*, 3(1):63–75, 1985.

[7] J. Chang and Maxemchuk N. F. Reliable broadcast protocols. *ACM Transactions on Computer Systems*, 2.3, Aug 1984.

[8] E. W. Dijkstra and C. S. Scholten. Termination detection for distributed computation. *Information Processing Letters*, 11:1–4, 1980.

[9] J. Fidge. Partial orders for parallel debugging. In *Proceedings of ACM SIGPLAN/SIGOPS workshop on parallel & Distributed Debugging*, pages 183–194, 1985.

[10] Gifford and Glasser. Remote pipes and procedures for efficient distributed communication. *ACM Transactions on Computer Systems*, 6.3, Aug 1988.

[11] Birman K., Schiper A., and Stephenson P. Fast causal multicast. Technical Report TR90-1105, Dept. of CS, Cornell University, Itacha, NY-14853-7501, April 1990.

context relationship.

The GS primitive is more closely related to the ISIS CBCAST primitive [3]. In CBCAST a label "clabel" is associated with each message. The ordering among messages is specified in terms of a "less than" relation, denoted by $\overset{isis}{\rightarrow}$, based on these labels. The kernel at each site knows this "less than" relation. Messages are received in an order that does not violate $\overset{isis}{\rightarrow} \bigcap \overset{lam}{\rightarrow}$ relation.

A message receipt ordering permitted by GS can also be permitted using CBCAST by appropriately defining the labels. [5] describes how to define such labels. [10]

In the new version of ISIS [11], the relation $\overset{isis}{\rightarrow}$ is the same as $\overset{lam}{\rightarrow}$. This implementation is similar to *Psync* except for process groups. The relation $\overset{isis}{\rightarrow}$ is encoded using vector clocks [9, 13]. Each message carries the value of the vector clock at the time of sending as its timestamp. This timestamp consists of N unbounded integers.

The implementation of GS suggested in [5] requires each message to carry $2N^2$ unbounded integers to enforce the required ordering and the amount of information kept at each site is also potentially unbounded.

As opposed to all the above broadcast primitives, the proposed implementation of the GS primitive requires each message to carry only 1 bounded integer, which is independent of the number of processes in the system. This bound on information carried by each message is not without cost. The number of message receipt orderings permitted by the proposed implementation is somewhat less than that permitted by the implementation proposed in [5]. It will become clear from the following example.

Consider a system of three processes, p_i, p_j, and p_k. Process p_i sends g which is a GO to p_j only and then sends a gt to p_k only. p_i then sends g' which is also a GO to p_j only. By definition, g and g' can be received in any order. Under the proposed implementation g' cannot be received before g at p_j, because the min-cut of g cannot cross the max-cut of gt and the max-cut of gt cannot cross min-cut of g'. Thus, limiting the number of permissible message receipt orderings to the order

[10] This assumes that the underlying kernel knows how to compare these label.

Since there is no p_j in this $gt.dst$, this gt will not be received by any p_j and yet it can be used as a signal from S_i that S_i is resetting its message identifier id.

4 Related Work and Conclusions

In this section, we compare GS with other broadcast primitives, namely, the Psync primitive in x-kernel [14] and CBCAST primitive of ISIS [3]. We also compare the proposed implementation of the GS primitive with the implementation suggested in [5]. The comparison is in terms of the number of possible message receipt orderings permitted by the primitive and the amount of additional information carried by a message to enforce the message ordering.

The *Psync* primitive in x-kernel [14] generates a context relationship for each message g, where g is in the context of g' if $g' \in \mathcal{P}(g)$. g is received only after all message in $\mathcal{P}(g)$ have been received. The set of message receipt orderings permitted by the *Psync* primitive is a subset of message receipt orderings permitted by GS. That is, for any given sequence of message sends all message receipt orderings permitted when Psync is used are also achievable when GS primitive is used and there may be some message receipt orderings permitted when GS is used which are not achievable under Psync. If it is desired that only the message receipt orderings permitted by *Psync* be allowed under GS, then the following must be ensured:

$\forall g, g', j : g.dst[j]=1 \bigwedge g'$ is the last message received from $p_{g'.se}$ by p_j before sending g:

$$\langle g'.typ,\ g.typ\rangle \notin \{\langle \text{GO,GO}\rangle, \langle \text{GO,GB}\rangle, \langle \text{GF, GO}\rangle, \langle \text{GF,GB}\rangle\}.$$

If $g' \in \mathcal{P}(g)$ then g' must be received before g at all common destinations. Since, *gf*, *gb*, and *go* do not order this receipt with respect to receipt of all messages in the system, allowing the above message type pairs would violate the message receipt ordering permitted by *Psync*.

The other difference between *Psync* and GS is that in *Psync* each message is sent to all processes in the system, as opposed to a subset of processes in GS.

The implementation of *Psync* proposed in [14] requires that each message be uniquely identified among all messages sent by a process. Each message carries N unbounded integers to encode the

Figure 3 shows the complete protocol executed by S_i.

3.4 Bounding the Message Id

Note that in the above implementation, each global flush message is assumed to have a process wide unique message identifier. We now discuss ways to bound these identifiers by reusing them.

Lemma 2 *Global Flush messages g and g', sent by the same process, may have the same id iff their diffusion regions do not intersect at any process.*

This follows from the fact that at each S_j, $g.id$ is used to count the number of incoming channels on which the message has been received by S_j, so far. Once a message has been received on all incoming channels, message identifier is no longer needed to identify the message. □

Lemma 3 *Given three global flush messages g, gt, g' sent by p_i such that $g \in \mathcal{P}(gt)$ and $g' \in \mathcal{F}(gt)$. The diffusion regions of g and g' do not intersect at any site.*

Proof: We assume to the contrary and then show that Property G is violated. Assume that three global flush messages g, gt, and g', sent by p_i, exist such that $g \in \mathcal{P}(gt)$, $g' \in \mathcal{F}(gt)$, and the diffusion regions of g and g' intersect at S_j.

For the diffusion regions of g and g' to intersect at S_j either the max-cut of g must cross the max-cut of gt or the max-cut of gt must cross the min-cut of g'. If the max-cut of g crosses the max-cut of gt, then assume that g is a gf. Since gf is received at the max-cut, its receipt violates Gt. A contradiction. If max-cut of gt crosses the min-cut of g', then assume that g' is a go. Since go is received at the min-cut, its receipt violates Gt. A contradiction. □

Clearly, it follows from Lemmas 2 and 3 that the identifiers of messages, sent by $gt.se$, in $\mathcal{P}(gt)$ can be reused for messages sent by $gt.se$ in $\mathcal{F}(gt)$. Thus, to bound the range of values used by S_i for message identifiers, S_j should periodically initiate the diffusion process for gt s.t. $\forall j$ $gt.dst[j]$=0.

]
]
]

The function $InR(Infr)$ returns true if an entry for $Infr$ already exists in $\mathcal{D}$. Otherwise, it returns false. The function $InsertQ(x)$ inserts entry x in $\mathcal{Q}$. The function $Increment(Infr)$ increments the cnt field in the entry for $Infr$ in $\mathcal{D}$. The function pop removes and returns the entry at the top of the $\mathcal{Q}$.

0: S_i:*[p_i is ready to execute GS($g.typ$, $g.dst$, $g.body$) $\bigwedge$ $\neg$ $Inhibit$ $\rightarrow$
[$Infr$:= $\langle id$, i, $g.typ$, $g.dst$, $g.body\rangle$
$g.typ$=GO$\rightarrow$ $mtyp$= o
$g.typ$=GF$\rightarrow$ $mtyp$= f
$g.typ$=GB$\rightarrow$ $mtyp$= b
$g.typ$=GT$\rightarrow$ $mtyp$= t; $Inhibit$=$True$
$diffuse(mtyp, Infr)$
id=id+1
]
[]
1: p_i is ready to execute Rcv($g.se$, $g.typ$, $g.body$) $\bigwedge$ $\neg$empty($\mathcal{Q}$)$\rightarrow$ return(pop($\mathcal{Q}$)) to p_i
[]
2: S_i receives m carrying $Infr$ $\rightarrow$
[$\neg$ InR($Infr$)$\rightarrow$
[InsertR($Infr$)
send m carrying $Infr$ on all outgoing channels
($Infr.typ$=GO $\bigvee$ $Infr.typ$=GB) $\bigwedge$ $Infr.dst[i]$=1 $\rightarrow$ InsertQ($\langle Infr.se$, $Infr.body$, $Infr.typ\rangle$)
]
[]
InR(Infr)$\rightarrow$
[Increment($Infr$)
$\exists r \in \mathcal{D}$ $\bigwedge$ $r.cnt$=inc_i $\rightarrow$
[R:=R $-$ r
($Infr.typ$=GF $\bigvee$ $Infr.typ$=GT) $\bigwedge$ $Infr.dst[i]$=1 $\rightarrow$ InsertQ($\langle r.se$, $r.body$, $r.typ\rangle$)
$Infr.typ$=GT $\bigwedge$ $Infr.se$=i $\rightarrow$ $Inhibit$=False
]
]
]
]
Initially, id=0, empty($\mathcal{Q}$)=True, $\mathcal{D}$={}, $Inhibit$=False

Figure 3: Protocol executed at S_i.

$g.typ$=GT$\rightarrow$ $mtyp$= t; $Inhibit$=$True$
$diffuse(mtyp,\ Infr)$
id=id+1
]

Initially, id=0; $Inhibit$ = $False$

Whenever p_i is ready to execute Rcv($g.se$, $g.typ$, $g.body$) and $\mathcal{Q}$ is not empty, S_i returns the element at the top of $\mathcal{Q}$.

1: p_i is ready to execute Rcv($g.se$, $g.typ$, $g.body$) $\bigwedge$ $\neg$empty($\mathcal{Q}$)$\rightarrow$ return(pop($\mathcal{Q}$)) to p_i

Whenever a flush message is delivered at S_i, it does the following: If an entry for this diffusion process does not already exist in $\mathcal{D}$, an entry is created with the cnt field initialized to 1. S_i then forwards the message on all outgoing channels. If the flush message is an o or b and p_i is a destination, then it inserts $\langle g.se,\ g.typ,\ g.body\rangle$ in $\mathcal{Q}$, i.e., g becomes receivable. As shown in Table 1, gb and go become receivable at min-cut. If an entry for this diffusion process already exists, the cnt field in incremented. When cnt becomes inc_i, where inc_i is the number of incoming F-channels for S_i, the entry is deleted. Note that the max-cut for a diffusion is attained when cnt = inc_i. If the flush message is an f or t and p_i is a destination, then it inserts $\langle g.se,\ g.typ,\ g.body\rangle$ in $\mathcal{Q}$, i.e., g becomes receivable. As indicated in Table 1, gf and gt become receivable at the max-cut of their diffusion process. Additionally, if $g.se$=i and $g.typ$=GT the Inhibit flag is set to false as this event marks the end of the diffusion region on S_i.

2: S_i receives m carrying $Infr$ $\rightarrow$
[$\neg$ InR($Infr$)$\rightarrow$
[InsertR($Infr$)
send m carrying $Infr$ on all outgoing channels
($Infr.typ$=GO $\bigvee$ $Infr.typ$=GB) $\bigwedge$ $Infr.dst[i]$=1 $\rightarrow$ InsertQ($\langle Infr.se,\ Infr.body,\ Infr.typ\rangle$)
]
[]
InR(Infr)$\rightarrow$
[Increment($Infr$)
$\exists r \in \mathcal{D}$ $\bigwedge$ $r.cnt$=inc_i $\rightarrow$
[R:=R $-$ r
($Infr.typ$=GF $\bigvee$ $Infr.typ$=GT) $\bigwedge$ $Infr.dst[i]$=1 $\rightarrow$ InsertQ($\langle r.se,\ r.body,\ r.typ\rangle$)
$Infr.typ$=GT $\bigwedge$ $Infr.se$=i $\rightarrow$ $Inhibit$=False

process, to determine the S_j's event at the max-cut. To implement this, we assume that each global flush message sent by a process has a process-wide unique [8] identity denoted by *g.id*. All flush messages, sent by a diffusion process invoked to send *g*, carry *g.id* as a part of *Infr*. Thus, *Infr*:= $\langle$*g.id*, *g.se*, *g.typ*, *g.dst*, *g.body*$\rangle$.

Each S_j maintains a set $\mathcal{D}$ which contains information about the messages whose min-cut has been attained at S_j (i.e., at least one copy of flush message has been received by S_j) but the max-cut has not been attained at S_j (i.e., a copy of the flush message has not been received on all incoming channels). Each element r in $\mathcal{D}$ is a 6-tuple $\langle$*id*, *se*, *typ*, *dst*, *body*, *cnt*$\rangle$. The *cnt* field is used to count the number of copies received of the message which is uniquely identified by $\langle$*id*, *g.se*$\rangle$.

The set of messages in $\mathcal{D}$ which become receivable may not be received in any arbitrary order because such a receipt might violate Property *G*. If messages in this set are received by p_j in the order they become receivable at p_j, then, as proved by Lemma 1, Property *G* would not be violated [9]. To implement this, each S_j also maintains a queue $\mathcal{Q}$ which contains information about messages that have not yet been received but can be received, in the order specified by $\mathcal{Q}$, without violating Property *G*. Each element in $\mathcal{Q}$ is a triplet $\langle$*g.se*, *g.typ*, *g.body*$\rangle$.

To implement the inhibition of message sends as identified in Table 1, each site maintains a boolean *inhibit* initialized to false.

Implementation Details

Whenever p_i is ready to execute GS(*g.typ*, *g.dst*, *g.body*), S_i initiates the diffusion process. This gives the 0^{th} statement of the algorithm.

0: p_i is ready to execute GS(*g.typ*, *g.dst*, *g.body*) $\bigwedge \neg$ *Inhibit* $\rightarrow$
[*Infr*:= $\langle$*id*, i, *g.typ*, *g.dst*, *g.body*$\rangle$
g.typ=GO$\rightarrow$ *mtyp*= *o*
g.typ=GF$\rightarrow$ *mtyp*= *f*
g.typ=GB$\rightarrow$ *mtyp*= *b*

[8] Later we weaken this restriction to put a bound on *g.id*.

[9] Such a total ordering is sufficient but not necessary. A weaker ordering can be imposed, such that Property *G* is not violated, by using elaborate data structures. Such enhancements, although interesting, are not relevant to the central idea presented in this paper and hence are not pursued.

We now prove the correctness of the implementation.

3.2 Correctness Proof

Lemma 1 *In a system which supports the flush messages only, where each global message send and receive is implemented according to rules summarized in Table 1, a message when received satisfies Property G.*

Proof:

Property *Go* is trivially satisfied.

For each $g \in \mathcal{P}(gf)$, max-cut for g cannot cross min-cut of gf (due to the property of the forward flush). Thus, such an g would be received before gf. Hence, satisfying Property *Gf*.

For each $g \in \mathcal{F}(gb)$, min-cut for gb cannot cross max-cut of g (due to the property of the backward flush). Thus, gb would be received before any such g. Hence, satisfying Property *Gb*.

The requirement that initiation of diffusion process in g's diffusion region on $S_{g.se}$, if g is a gt, be inhibited is necessary. Any g' whose diffusion process is initiated by $S_{g'.se}$ in this region would be in $\mathcal{F}(g)$. Max-cut of g may cross min-cut of g', thus potentially permitting the receipt of g' before g. This would violate Property *Gt*(b). [7] The rest of the proof for gt is similar to the proof for gf and gb.

Hence the implementation is correct. □

3.3 Discussion

Recall that each gf and gt become receivable at the max-cut of their corresponding diffusion processes. Since for a diffusion process, each S_j receives exactly one copy of the message on each incoming channel, S_j can count the number of copies of a flush message received for each diffusion

[7] This is not the case with gb since a gb is received at its min-cut. The min-cut of gb can never cross the min-cut of any such g' (due to the backward flush property).

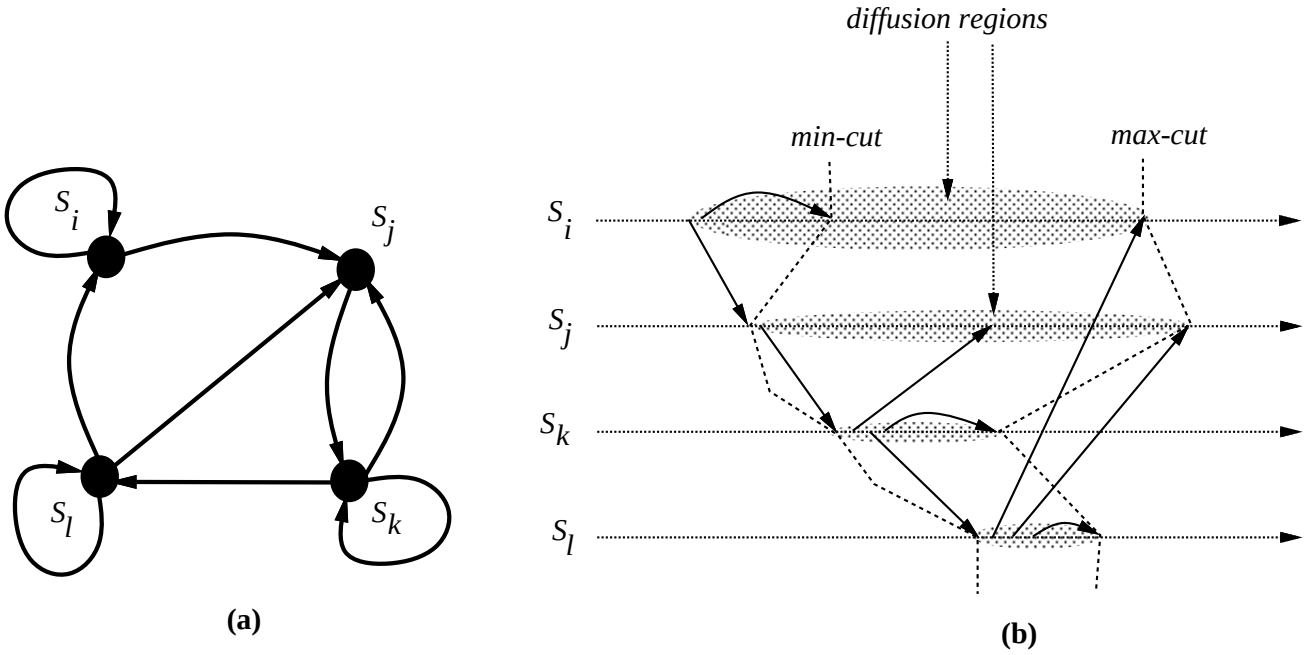

Figure 2: Figure (a) shows an F-channel network topology for a 4-site system. The solid directed lines denote F-channels. Figure (b) identifies the message exchange due to the diffusion process initiated by S_i, the min-cut, the max-cut, and the diffusion region on S_i.

forwards a copy of the flush message along each outgoing channel, if it has not already done so.

Since each site forwards a flush message only the first time it receives the flush message, a site receives exactly one copy of each flush message on each incoming channel.

The set of events of receiving the first flush message, one at each site, due to a diffusion process identifies a cut [6] to be referred to as *min-cut* for the diffusion process. Similarly, the set of events of receiving the last flush message due to the diffusion process identifies the cut referred as the *max-cut* for the diffusion process. Given any two cuts C1 and C2, we say that C1 *crosses* C2 iff $\exists$ $e_i \in$ C1 s.t. $e_i \notin$ C2. Note that the min-cut of a diffusion process never crosses its max-cut.

The interval between the first and the last event at site S_j due to a diffusion process is termed as the *diffusion region on* S_j. Figure 2(b) identifies the messages exchanged due to a diffusion process, its min-cut, its max-cut, and its diffusion region on sites when an underlying F-channel network topology of Figure 2(a) is assumed.

Whenever p_i executes GS(*g.typ*, *g.dst*, *g.body*), S_i initiates a diffusion process with *Infr*= $\langle$*g.typ*, *g.dst*, *g.body*$\rangle$. When *g.typ* is a GF, GB, GT, or GO, then *mtyp* is a f, b, t, or o, respectively. If g is a *gf* or *gt*, then for each p_j in *g.dst* g becomes receivable at max-cut of the diffusion process. Otherwise, for each p_j in *g.dst* g becomes receivable at the min-cut. Additionally, if g is a *gt*, then message sends by $p_{g.se}$ are inhibited in g's diffusion region on $S_{g.se}$. Table 1 summarizes the implementation rules for the global flush messages.

To implement GS(*g.typ*, *g.dst*, *g.body*) where *g.typ*=	Initiate *diffuse process* with *Infr*=$\langle$*g.typ*, *g.dst*, *g.body*$\rangle$ and *mtyp*=	For each p_j in *g.dst* g becomes receivable at	Inhibit initiation of diffusion process in the diffusion region on $S_{g.se}$
GF	f	max-cut	No
GB	b	min-cut	No
GT	t	max-cut	Yes
GO	o	min-cut	No

Table 1. An implementation of Global Flush Primitive based on diffusion process.

[6] A *cut* [6] is a set of events, containing an event from each site, such that if event e_j on S_j is in the cut, then all events on S_j before e_j are also in the cut.

F-channel network.

Property TC [Topological Constraint]

Consider the set of all global flush messages in the distributed system.

If there exists two global flush messages g and g′ in this set that have p_j as one of their destinations s.t. ((g.typ=GF $\bigvee$ g.typ=GT) $\bigwedge$ $g' \in \mathcal{P}(g)$) $\bigvee$ ((g.typ=GB $\bigvee$ g.typ=GT) $\bigwedge$ $g' \in \mathcal{F}(g)$),

then there must be a path of F-channels in the F-channel network topology from $S_{g.se}$ to S_j through $S_{g'.se}$. □

This topological constraint follows directly from the global flush properties and the underlying principle of the implementation. Global flush properties state that *gf* and *gb* are ordered with respect to messages in their past and future, respectively. A *gt* is ordered with respect to messages in its both past and future. The conditions in the underlying principle state that if message g must be received before/after g' at p_j then they must be inserted along F-Channel paths to p_j in that order. The topological constraint, thus, states that a path of F-Channels must exist to facilitate insertion of each message before/after messages in its future/past. Since the set of global flush messages in a distributed system may not be known apriori, **TC** can be stated as: if there is a path of F-channels from S_i to S_j, then there must be a path of F-channels from S_j to S_i. In the rest of the paper we would assume that there is a path of F-channels from each site to every other site.

We now describe a *diffusion process* [8] which causes a flush message, m, carrying information *Infr*, to be sent exactly once along each F-channel in the underlying network. By appropriately defining the flush message type and information *Infr*, we would ensure the two underlying principles described in the beginning of this section.

Diffusion process initiation rule: When invoked as *diffuse*(*mtyp*, *Infr*), the initiating site sends a flush message of type *mtyp* carrying information *Infr* along each outgoing F-channel.

Diffusion process propagation rule: On the receipt of a flush message carrying *Infr*, the site

flush messages is implemented by communication between sites using *flush messages*. Execution at process at S_i can be viewed as a sequence of events of sending and receiving flush messages. *fsend*(m) is the event of sending a flush message m along an F-channel. *frecv*(m) is the event of receiving the flush message m along an F-channel.

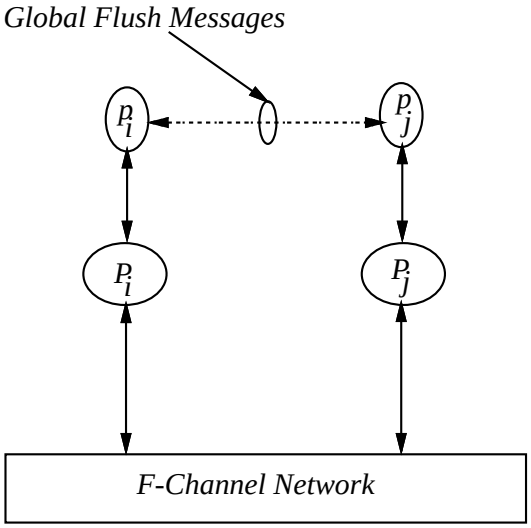

Figure 1: General Architecture of the system: The communication between processes using global flush messages is implemented by communication between sites using flush messages.

3 An F-channel Based Implementation

In this section, we discuss an implementation of GS primitive that is based on the F-channels.

3.1 Basic Idea

The underlying principle for the proposed implementation is as follows: *"if g and g' are two global flush messages such that g must be received at some p_j before g' is received then (1) a flush message m corresponding to g must be sent on all F-channel paths in the F-channel network from $S_{g'.se}$ to S_j before the flush message m' corresponding to g' is sent on these paths and (2) this order must be maintained along all paths until they are received."* This identifies the following constraint on the

by p_j. A global flush message can be *received* by p_j only when its receipt by p_j would not violate Property G. Such a global flush message (one which has not been received yet by p_j and its receipt would not violate Property G) is said to be *receivable* at p_j. A message that has arrived at p_j from the communication medium is said to be *delivered* at p_j. A message *delivered* at p_j may not yet be receivable at p_j. A global flush message g can be received by a process only when g satisfies the Property G with respect to other global flush messages in the system.

The relation $\stackrel{n*}{\rightarrow}$, defined next, describes the message receipt ordering permitted by the Property G.

Definition 4 *[“$\stackrel{n*}{\rightarrow}$”, to be read as “permitted message receipt ordering”]*
Define $g \stackrel{n*}{\rightarrow} g'$ *iff* $\exists j \; g \stackrel{j*}{\rightarrow} g'$, *where* $\stackrel{j*}{\rightarrow}$ *is defined as follows:*
$g \stackrel{j*}{\rightarrow} g' \Leftrightarrow gsend(g) \stackrel{lam}{\rightarrow} gsend(g') \bigwedge g.dst[j]{=}g'.dst[j]{=}1 \bigwedge$
$((g.typ{=}GB \bigvee g.typ{=}GT) \bigvee (g.typ{=}GF \bigvee g.typ{=}GT))$

The relation $\stackrel{n*}{\rightarrow}$ identifies the permitted message receipt orderings between various message types such that Property G is not violated. $g \stackrel{n*}{\rightarrow} g'$ means that g must be received before g'. Note that *gb* and *gt* satisfy $\stackrel{n*}{\rightarrow}$ relation with all messages in their future that have at least one common [5] destination. Similarly, *gf* and *gt* satisfy $\stackrel{n*}{\rightarrow}$ relation with all messages in their past that have at least one common destination.

The primitive Rcv(*var g.se*, *g.typ*, *g.body*) is used to receive a global flush message. The Rcv is a blocking primitive. On an invocation of Rcv(*g.se*, *g.typ*, *g.body*) by p_j, the sender *g.se*, the type *g.typ*, and the body *g.body* of g is returned to p_j, whenever a g exists at p_j such that it is receivable at p_j. Each invocation of Rcv(*g.se*, *g.typ*, *g.body*) by p_j corresponds to a $grecv(g)_j$ event.

For each p_i we define S_i (read as site i) as the site of p_i. These S_i's are connected by a network of F-channels such that there is a path consisting of F-channel, from each site to every other site. Later, in Section 3, we identify the minimal requirement on the F-channel network for the correct functioning of the proposed implementation. The communication between processes using *global*

[5] Both messages have a process as one of their destinations.

A message g', that is sent either by $p_{gb.se}$ after sending gb or by a p_j after p_j has received gb or a message that is in the future of gb, cannot be received by any process in $gb.dst$ before gb is received. Thus, a gb can be viewed as the *fastest* possible message in the system.

Property *Gf* **[Global forward flush]**

For each "gf", at each process p_j *in* $gf.dst$ $\forall g'$: $g' \in \mathcal{P}(gf) \bigwedge g'.dst[j]=1$: $grecv(g')_j \overset{lam}{\rightarrow} grecv(gf)_j$. □

A message g', that is in the past of gf will be received by all processes in $gf.dst$ before gf is received. Thus a gf can be viewed as the *slowest* possible message in the system.

Property *Gt* **[Global two-way flush]**

For each "gt", at each process p_j *in* $gt.dst$ *the following two properties hold:*

$Gt(a)$ $\forall g'$: $g' \in \mathcal{F}(gt) \bigwedge g'.dst[j]=1$: $grecv(gt)_j \overset{lam}{\rightarrow} grecv(g')_j$, *and*

$Gt(b)$ $\forall g'$: $g' \in \mathcal{P}(gt) \bigwedge g'.dst[j]=1$: $grecv(g')_j \overset{lam}{\rightarrow} grecv(gt)_j$. □

Thus, a gt satisfies both Gb and Gf properties.

Property *Go* **[Global ordinary flush]**

No ordering, with respect to any other global flush messages, is imposed on receipt of a "go". □

A go satisfies neither Property Gb nor Property Gf. It can be received in any order with respect to other messages in the distributed system as long as the global flush properties dictated by other messages are not violated.

Properties Gb, Gf, Gt, and Go are collectively referred to as Property G (or the *global flush property*). A global flush message is said to be *received* by p_j when it's body is actually processed

was received by $g.se$ before $g.se$ sent g or were in the past of messages satisfying the previous two conditions.

Definition 3 ["$\mathcal{F}(g)$", **to be read as "the future of** g"]

For any global flush message g, define $\mathcal{F}(g) = \{g' \text{: } gsend(g) \overset{lam}{\rightarrow} gsend(g')\} = \{g' \text{: } g \in \mathcal{P}(g')\}$.

The future of a message g includes all messages g' in the system such that the event of sending of g "happened before" the event of sending of g'. Note that these g's need not be sent by $g.se$. The future of g, thus, includes all messages that are either sent by $g.se$ after sending g or are sent by p_j after receiving a message (which may be g) from $g.se$ that was not sent before $g.se$ sent g or are in the future of messages satisfying the previous two conditions.

Global Flush Primitive

Global flush messages can be of the following types: GB, GF, GT, or GO (to be read as *global backward flush*, *global forward flush*, *global two-way flush*, and *global ordinary* messages, respectively). *Global flush messages* are sent using the global send primitive, GS($g.typ$, $g.dst$, $g.body$), where $g.typ$, $g.dst$, and $g.body$ are defined as follows: The message type $g.typ$, which could be any one of GB, GF, GT, or GO, specifies that Properties *Gb*, *Gf*, *Gt*, or *Go* (defined later), respectively, should be ensured between receipt of g and receipt of messages in $\mathcal{P}(g)$ and/or messages in $\mathcal{F}(g)$. The message destination $g.dst$ specifies the group of processes to which the body $g.body$ of g is to be sent. The sender of a message can also name itself as one of the recipients of the message. $g.dst$ is represented as a vector of N integers such that $g.dst[j]$=1 if $g.body$ is to be sent to p_j and $g.dst[j]$=0 otherwise. We say that p_j is in $g.dst$ iff $g.dst[j]$=1. Each invocation of GS($g.typ$, $g.dst$, $g.body$) by p_i corresponds to a $gsend(g)$ event on p_i.

We refer to g as *gb*, *gf*, *gt*, or *go* message if $g.typ$ takes values of GB, GF, GT, or GO, respectively.

Property *Gb* **[Global backward flush]**

For each "gb", at each process p_j *in* $gb.dst$ $\forall g'$: $g' \in \mathcal{F}(gb) \bigwedge g'.dst[j]=1$: $grecv(gb)_j \overset{lam}{\rightarrow} grecv(g')_j$. □

2 System Model

The system under consideration consists of a set of N processes, p_0, ..., p_{N-1}, which communicate with each other solely by passing *global flush messages* [3], denoted by g. A process is a sequence of events. An event on a process could either be an internal event or an event of sending or receiving a *global flush message*. An internal event, denoted by *Int*, does not directly affect events on other processes. Let $gsend(g)$ denote the event of atomically sending g to its specified group [4] of processes in the system. We use $g.se$ to denote the sender of g. The event of p_j receiving g is denoted by $grecv(g)_j$. For every $grecv(g)_j$ event, there is a corresponding $gsend(g)$ event. In this paper, we concern ourselves only with events of type $gsend(g)$ and $grecv(g)_j$ because they directly affect events on other processes. We use e_i and e_i' to denote events on p_i. Lamport's "happened before" relation [12], denoted by $\stackrel{lam}{\rightarrow}$, describes a partial order on these events.

Definition 1 ["$\stackrel{lam}{\rightarrow}$", **to be read as "happened before"**]

The relation $\stackrel{lam}{\rightarrow}$ on the set of events in a distributed system is the transitive closure of relation $\rightarrow$, where $\rightarrow$ is the relation defined as follows: (a) If e_i and e_i' are events such that e_i occurs before e_i' then $e_i \rightarrow e_i'$. (b) If e_i is the event of sending a global flush message and e_j is the event of receiving the same global flush message, then $e_i \rightarrow e_j$.

Definition 2 ["$\mathcal{P}(g)$", **to be read as "the past of** g"]

For any global flush message g, define $\mathcal{P}(g) = \{g'$: $gsend(g') \stackrel{lam}{\rightarrow} gsend(g)\}$.

The past of a global flush message g is the set of messages, in the system, whose send events "happened before" the send event of g. Note that the past of g may include messages sent by processes other than $g.se$. In other words, the past of g includes all messages that were either sent/received by $g.se$ before g was sent or were sent by p_j before p_j sent the last message g' that

[3] *Global flush messages*, sent using the GS primitive, satisfy the *global flush property* defined later in this section.

[4] The size of the *group* can vary from 1 to N. When the *group* is of size 1, then the message is a one-to-one message. When this *group* contains all processes in the system, the message is a broadcast.

its *past* [2] and/or in its *future*. Second, it takes as a parameter a named group or a set of processes as the destination of the message. This primitive provides an elegant way to reason about the past and the future of a message. It also provides the ability to permit higher communication level concurrency, measured in terms of number of permissible message receipt orderings, when compared to a system with FIFO channels. Use of the GS primitive simplifies algorithm development in a distributed environment because the receipt ordering is ensured by the communication layer as opposed to by the application itself. Several applications of the GS primitive are discussed in [5].

In this paper we propose an implementation of the GS primitive using F-channels. This implementation uses a *diffusion process* [8] which results in a flush message to be sent along each F-channel in the system. In the proposed implementation, each message carries just 1 bounded integer and the information kept at each site is bounded by the size of this integer. The overhead, both in terms of the amount of information carried by a messages and the amount of information kept at each site, is independent of the number of processes in the system. It must be noted that implementations of causal broadcast in related works, e.g., ISIS [2, 3, 11] and Psync [14] have unbounded overheads in terms of information carried by each message. This is due to the unbounded clocks used to encode this information. The counter based implementation of the GS primitive suggested in [5] also has unbounded overhead in terms of both, the information carried by each message and the amount of information kept at each site.

The rest of the paper is organized as follows: In Section 2, we discuss the system model, develop the notation, and describe the GS primitive. In Section 3, we describe an F-channel based implementation of the GS primitive and prove its correctness. In Section 4, we discuss related work and conclusions.

[2] The terms "past" and "future" are based on Lamport's "happened before" relation and will be defined later in Section 2.

1 Introduction

A distributed computation consists of a set of processes that do not share a common memory or a global time and which cooperate and compete to achieve a common goal. Interprocess communication mechanisms provide an abstraction through which such processes exchange messages. Various mechanisms for one-to-one communication, e.g., datagrams, virtual circuits, remote procedure calls [4, 10], F-channels [1] have been studied. Mechanisms for broadcast communication have also been studied [2, 3, 7, 11, 14]. These mechanisms provide primitives which support abstractions helpful in designing distributed algorithms.

F-channels have been proposed as a mechanism for one-to-one communication between pairs of processes in a distributed system [1]. An F-channel is a single non-FIFO channel that ensures the *flush property* [1] which is based on ordering the receipt of a message with respect to receipt of messages sent before and/or after[1] it along the same F-channel. It supports four message types, namely, *forward flush*, *backward flush*, *two-way flush*, and *ordinary* (message of these types will be denoted by f, b, t, and o, respectively). An f sent along an F-channel is received after all messages sent along the F-channel before sending f have been received. A b sent along an F-channel is received before any message sent along the F-channel after sending b is received. A t sent along an F-channel is received after all messages sent along the F-channel before sending t have been received and before any message sent along the F-channel after sending t is received. An o sent along an F-channel can be received at any time as long as its receipt does not violate requirements specified by other messages.

Global flush communication primitive GS, stands for Global Send, was recently proposed as a mechanism for one-to-many communication [5]. The global communication primitive has two properties: First, it ensures the *global flush property* (formally defined later) specified by the message type. The global flush property is an extension of the flush property to one-to-many communication and is based on ordering receipt of a message with respect to receipt of messages in

[1] According to senders local time.

An Implementation of Global Flush Primitive using F-channels

Ashwani Gahlot and Mukesh Singhal
Department of Computer and Information Science
The Ohio State University
Columbus, OH 43210

October 1, 1992

Abstract

The Global Flush communication primitive allows the sender to order receipt of a message with respect to receipt of other messages. Use of this primitive provides an elegant way to reason about message orderings (as it allows a process to deduce the receipt event orderings at other processes), simplifies algorithm development, and permits higher communication level concurrency than some communication primitives proposed in the literature. In this paper, we discuss an implementation of the Global Flush communication primitive that is based on F-channels.

Keywords: Asynchronous distributed systems, Communication architecture, Group communication, Distributed algorithms.