Membership-Insensitive Totally Ordered Multicast: Properties and Performance

Jerzy Konorski[1]

[1]Technical University of Gdansk
ul. Narutowicza 11/12, 80-952 Gdansk, Poland
jekon@pg.gda.pl

Abstract. Membership-insensitive group transport protocols are particularly desirable for large groups of fast-varying membership where the distributed application being served insists on collective output and survivability rather than producing a consistent record of successive group views. An overview of such a protocol, able to maintain agreed delivery order of multicast messages across the group except for scenarios that can be made arbitrarily improbable, is presented along with a proposed specification of the so-called '1C network service with ContiguityDetector' it is built upon. Key properties of the protocol are expressed through some graph-theoretic observations. LAN implementation experience and performance measurements are described to conclude that the group throughput remains high under constant group membership and, there being virtually no group reconfiguration overhead, varies gracefully in step with the number of active group members.

1 Group Communication Model

Group communication models extend the familiar point-to-point (unicast) paradigm by introducing

- a family of patterns of message transfer, involving a group of member sites rather than just two, of which multipoint-to-multipoint is the closest to the intuitive notion of group communication (the related *multicast protocols* [5] are focused on typically best-effort multicast routing and switching within the underlying communication network),
- a wider choice of QoS options that include, besides performance-oriented ones, group-wide consistency constraints like reliability, message ordering and membership control (the related *group transport protocols* [13] focus on implementation of these constraints, typically through multiple point-to-multipoint message transfers with coupled control); when referring to a *group communication protocol* (GCP) we shall understand a group transport protocol, the multicast protocol functionality being left to the underlying network service.

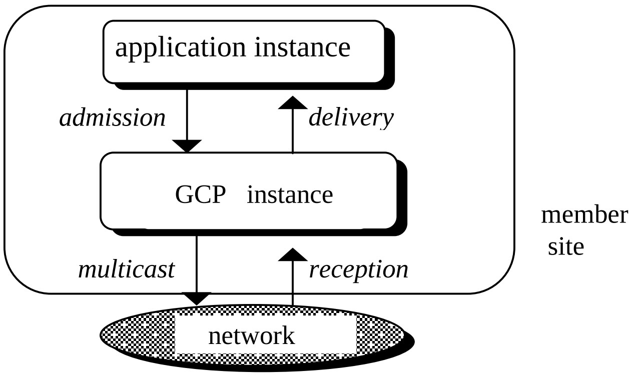

Fig. 1. Logical placement of a GCP instance

Logically, as Fig. 1 shows, a GCP instance at a member site is situated between the local instance of the distributed application it serves and the network service it uses.[1] We will carefully distinguish between message admission and delivery, and message multicast and reception, actions occurring across the application-to-GCP and GCP-to-network interfaces, respectively. With regard to the elements of the model in Fig. 1 we state the following assumptions and non-assumptions:

- **application instances**: generate messages at will, as determined by the (arbitrary) course of application execution; each message is destined for the whole of the group consisting of *N* member sites (we consider one group only),
- **GCP instance**: multicasts messages it has admitted, receives messages as they arrive from the network and delivers them to the application instance subject to prescribed group-wide consistency constraints.
- **network service**: employs any best-effort multicast protocol thus constituting an asynchronous environment with arbitrary message delays and losses.
- **sites**: may exhibit intermittent presence i.e., undergo alternate presence and absence spells (e.g., due to outages or switching to other tasks), thus constituting a potentially fast-varying membership with non-Byzantine site faults permitted.

The rest of the paper is organised as follows. In Sec. 2 the relevant consistency constraints are specified and a case is made for membership-insensitive GCPs. A suitable network service model, called 1C with ContiguityDetector, is introduced in Sec. 3 along with a few exemplary network settings. A membership-insensitive GCP is proposed in Sec. 4 and some key properties thereof stated in Sec. 5. Finally, Sec. 6 describes a LAN implementation experience with the proposed protocol and concludes with a brief discussion of its measured performance.

[1] Fig. 1 need not map directly onto the ISO-OSI layering e.g., 'GCP instance' may be either part of the transport or application layer or split between the two; accordingly, 'network' may or may not encompass transport layer functionality.

2 Consistency Constraints

The group-wide consistency constraints a GCP has to meet, Agreed Delivery Order (ADO) and Membership Control, are very different in nature, which gives rise to splitting a GCP into separate subprotocols. Although group communication need not be connection-oriented, we may use the term *group connection* to designate a self-contained exchange of a set M of multicast messages. Let the member sites be numbered 1 through N and let R_n and D_n be the sequences of messages respectively received and delivered at site n throughout the group connection. ADO states that, regardless of the R_n's, the following holds for the D_n's:

- **Atomicity**: for any $m \in M$, $m \in D_{n*}$ at some site $n*$ implies that $m \in D_n$ for all n,
- **Total Order**: a total order '$\rightarrow$' on M can be produced such that for any site n and $m,m' \in M$, ($m,m' \in D_n$ and $m \rightarrow m'$) implies that m precedes m' in D_n.

Note that neither of these statements implies the other; in conjunction they permit a distributed application to run as if supplied from a central event queue.[2]

Preserving ADO clearly requires that the member sites perceive themselves as members. Given that some of them may enter an absence spell at times, one wants to confine the above ADO specification to the constant membership case. The opposite case is covered by Membership Control. Let $e_{n,i}$ be the ith membership change event perceived at site n and let $V(e_{n,i})$ be the membership view implied by $e_{n,i}$ i.e., the set of sites perceived by site n as present upon $e_{n,i}$. Then the following is to hold:

- **Membership Consensus**: all sites $n' \in \mathrm{V}(\mathrm{e}_{n,i})$ perceive $\mathrm{e}_{n,i}$,
- **Virtual Synchrony** [2]: all sites $n' \in \mathrm{V}(\mathrm{e}_{n,i}) \cap \mathrm{V}(\mathrm{e}_{n,i+1})$ i.e., members of two consecutive membership views, deliver messages in ADO between perceiving $\mathrm{e}_{n,i}$ and $\mathrm{e}_{n,i+1}$; **Extended Virtual Synchrony** [11] enables to resume Virtual Synchrony when some previously absent sites have entered another presence spell.

ADO is costly in terms of message latencies and overhead, which may behove designers to consider quasi-ADO protocols instead, where violations of ADO are possible though materialise arbitrarily rarely. Membership Control, besides the well-known impossibility result [8], fails in that ultimately it is to unify the perception of successive group views, which is not a universal motivation. It is for collective accountancy-oriented applications (banking, reservation systems), but not for collective output-oriented ones, especially involving large and/or variable groups, where membership sensitivity is more a burden than a benefit. We illustrate this point by an example of a generic distributed resource sharing framework.[3]

[2] Some authors [16] add another dimension by distinguishing weak and strong Total Order. Others [9] define Validity and Integrity to ensure eventual delivery of multicast messages and prevent delivery of fake ones, respectively. The former is covered by the Eventual Receipt assumption (Sec. 4); the latter is only important for Byzantine site faults.

[3] A similar point can be made for some other parallel tasking frameworks e.g., parallel simulation with rollback.

Suppose that the member sites are to perform successive operations on a countable reusable resource and that the throughput (total operations per second) determines the output we get. Temporary exclusive control over resource units is transferred among the sites via multicast request, allocation and release messages, the former processed at each site in the FIFO order. To prevent the so-called wait-for condition and the resulting possibility of deadlock, a site that has not been allocated enough resource units for its operation is obliged, upon expiry of a time-out, to release the collected resource units and start requesting anew. Preserving ADO of request and allocation messages is important as any inconsistency would favour a situation of two or more sites collecting resource units in parallel and ending up with expired time-outs, which would reduce the throughput we are interested in maximising. Two observations emerge from the above example:

- Violation of ADO, although worsens the performance, need not be catastrophic to the application. Ad-hoc simulation experiments performed for several realistic WAN settings typically revealed a mere 10% operation throughput drop with as much as 5% messages violating ADO, which suggests there is a need for quasi-ADO protocols given the high cost of maintaining strict ADO in large groups.
- The sites are completely uninterested in tracking current membership. In fact, some of them may get uninterested at all and enter an absence spell; subsequently they may return and when they do, they are to promptly resume (quasi-)ADO instead of triggering costly rejoin-group procedures. For the TOTEM GCP [1], an ad-hoc simulation study shows that a single site out of 20, toggling periodically between presence and absence, causes reconfiguration overhead that halts the group connection for about 20% of the time and reduces the group throughput roughly by half for another 20%. Thus there is a need for membership-insensitive GCPs even if they are quasi-ADO rather than ADO. Note that membership insensitivity increases survivability in a non-protective or hostile environment.

3 Network Service Model

GCP design requires specification of the underlying multicast service. Some well-known ADO protocols are built on top of an unreliable service [1], [15], reliable service [14] or reliable FIFO service [7]; quite a few rely on causal service [2], [6].[4] Single-access broadcast media (e.g., Ethernet, FDDI, Token Ring, single-channel wireless TDMA) make a favourable environment since their hardware- and MAC-level broadcast and message ordering naturally define '$\rightarrow$' as the transmission order. Some ADO protocols make explicit use of this property [10], [12]. An appropriate service abstraction goes by the name of *1-channel (1C) service* [12] and can be reformulated analogously to Total Order using the R_n's rather than D_n's:

[4] This does not contradict the standard argument that message ordering should be an application layer issue [4] since both reliable FIFO and causal multicast services can be emulated within a GCP as a form of ADO preprocessing.

- **1C**: a total order '$\rightarrow$' on M can be produced such that for any site n and messages $m,m' \in M$, ($m,m' \in R_n$ and $m \rightarrow m'$) implies that m precedes m' in R_n.

1C is not a reliable multicast service: due to possible reception misses (caused by transmission losses, buffer overflow etc.), Atomicity cannot be guaranteed at the network level. Nevertheless, a realistic performance-oriented enhancement of the 1C service is possible. Let '$\Rightarrow$' denote contiguity in '$\rightarrow$' i.e., $m \Rightarrow m'$ means that $m \rightarrow m'$ and there is no m'' such that $m \rightarrow m''$ and $m'' \rightarrow m'$. At each site an inference device called ContiguityDetector is placed that for any pair of consecutively received messages m and m' infers either $m \rightarrow m'$ or $m \Rightarrow m'$ as specified below:

- if $m \Rightarrow m'$ is inferred at a site then indeed $m \Rightarrow m'$,
- if $m \Rightarrow m'$ then the inference about m and m' ($m \rightarrow m'$ or $m \Rightarrow m'$) may vary from site to site depending on local conditions.

We now discuss several realistic settings that virtually provide the above service.

3.1 Single-Access LAN

Messages arriving at a site are stored in a reception buffer whence they are fetched by the local GCP instance for processing. Slow-fetch and the resulting message overwrite can be assumed the sole factor behind reception misses; other factors (e.g., transmission errors, receiver overruns, software bugs) are marginalised over time by the advances in LAN technology. The ContiguityDetector at a site reduces to a bit set whenever an arriving message overwrites a previous one in the reception buffer, and reset after reception of any message; if the bit is found reset at the instant of message m' reception then $m \Rightarrow m'$ is inferred, m being the latest message received. Alternatively, a timing-based ContiguityDetector infers $m \Rightarrow m'$ if the receptions of messages m and m' are less apart in time than what is needed to transmit a message.

3.2 Bounded-Delay WAN

Upper-bounding message delays by a Δ converts the network into a synchronous environment suitable for timed ADO multicast service [9]. Assuming that local clocks at sites run synchronously, messages are delivered in ADO based on increasing reception timestamps. By adding Δ to the current time, a sender obtains the reception timestamp for its message, whose lifetime is limited to Δ. To avoid excessive delivery latencies, a Δ' can be used instead ($\Delta' < \Delta$) at the cost of some messages using up their lifetime before reaching all the member sites. The ContiguityDetector at site n analyses current network delays to site n and for a received message m' determines whether the delays have remained under Δ' since the latest message m was received. In that case, $m \Rightarrow m'$ is inferred.

3.3 Tag Sequencer

Messages are tagged with unique identifiers and use a reliable multicast service. A copy of the tag is unicast to a special sequencer site which numbers arriving tags sequentially and multicasts them back to the group. At any site, messages whose numbered tags have been received are delivered in sequence. For security and/or privacy reasons, tags from different groups should appear indistinguishable to the sequencer; it must therefore use a broadcast rather than multicast service for numbered tags e.g., best-effort FIFO broadcast. Now multiple groups share the same tag sequence, thus when a site has filtered out other groups' tags then for two consecutive tags numbered i and j ($i<j$) corresponding to messages m and m', it can infer $m\Rightarrow m'$ only if the filtered out tags' numbers are contiguous between $i+1$ to $j-1$.

4 Distributed Precedence Graph Protocol

We now present a survivable membership-insensitive GCP named Distributed Precedence Graph (DPG) that operates on top of the 1C service with ContiguityDetector. The network service is also assumed to fulfil Eventual Receipt [3] i.e., infinitely many message remulticasts result in infinitely many receptions at each member site. The DPG protocol exhibits quasi-ADO and Extended Virtual Synchrony in that, except for scenarios that occur arbitrarily rarely, each member site delivers continuous segments of the ADO sequence in successive presence spells.

A generic DPG instance at site n will be denoted DPG-n for short. The basic data structures at DPG-n consist of admission and ordering buffers at the application-to-DPG interface, across which messages are exchanged, as well as a multicast queue, a reception buffer and the ContiguityDetector at the DPG-to-network interface, across which PDUs are exchanged. Two PDU types are distinguished:

- *'message'* PDU - contains a new message (multicast for the first time) along with a unique message tag; the tag is valid only for the message's lifetime and need not reflect any static site numbering nor message sequencing within a site,
- *'report'* PDU - helps construct ADO, recover lost messages, perform flow control and local clock rate synchronisation.

A local clock controls the multicast interval (for flow control), remulticast time-outs (for recovery of lost messages), as well as deadlines for message incorporation into ADO (to implicitly eliminate absent sites).

4.1 Construction of Quasi-ADO

DPG-n maintains an acyclic directed *precedence graph* **G** whose vertices map onto message tags (written symbolically X, Y, Z etc.) and arcs reflect the order '$\rightarrow$'. $X\in\mathbf{G}$ implies that DPG-n has received message X or has been notified of its reception at some other site via a *'report'* PDU. $(X,Y)\in\mathbf{G}$ means that messages X and Y have been consecutively received at some site, implying $X \rightarrow Y$ though not necessarily $X \Rightarrow Y$. Reception of message Z causes the inclusion into **G** of vertex Z and arcs (X,Z) for all X having no successor in **G** (while the message body is stored in an ordering buffer). Moreover, DPG-*n* maintains the tag (*L_REC*) of the latest received message and *marks* arc (*L_REC*,Z) if *L_REC*$\Rightarrow Z$ is inferred at the instant of message Z reception. DPG-*n* includes in *'report'* PDUs it issues a subgraph of **G** corresponding to messages not yet delivered locally. Supposing DPG-*n* has received graph **G'** in a *'report'* PDU, first it computes $\mathbf{G} := \mathbf{G} \cup \mathbf{G'}$ and marks each arc that was marked in **G** or **G'**. Next, **G** is *linearised* so as to take advantage of the newly acquired knowledge of '$\rightarrow$'. Linearisation continues as long as, for any arc (X,Y),

– there exists a vertex $Z \in \mathbf{G}$ such that either $(X,Z) \in \mathbf{G}$ and (X,Z) is marked or $(Z,Y) \in \mathbf{G}$ and (Z,Y) is marked; (X,Y) is then replaced by (Z,Y) or (X,Z) respectively;
– there exists a directed path from X to Y in **G** consisting of more than one arc; (X,Y) is then removed from **G**.

4.2 Message Incorporation into Quasi-ADO

Two vertices of **G**, *E_DELIV* and *L_DELIV*, correspond to the earliest and latest *deliverable* messages (already incorporated into quasi-ADO). Message X becomes deliverable when the arc (*L_DELIV*,X) is marked; when all immediate successors of *L_DELIV* have been included in **G** for at least *Deliv_Cycle* seconds, they become deliverable in any static order e.g., of decreasing tags. Local delivery of *E_DELIV* follows when either the corresponding message body is found in the ordering buffers or *Deliv_Cycle* seconds have elapsed since it became deliverable (in the latter case a 'dummy' message is delivered with the understanding that all protocol instances that have the original message are currently absent).

4.3 Lost Message Recovery

DPG-*n* appends a list *LM* of lost message tags to issued *'report'* PDUs as well as a list *RM* of messages (tags and bodies) remulticast in response to earlier received *'report'* PDUs. Message X loss is detected upon: (1) reception of a *'report'* PDU with graph **G'** containing a vertex $X \notin \mathbf{G}$ and/or a remulticast message X not found in the ordering buffers, which cannot be stored because of overflow, or (2) reception of a *'message'* PDU while the ordering buffers overflow. In case (2), all the ordering buffers might be occupied by successors of message X. To prevent such a deadlock, message X is permitted to seize the ordering buffer occupied by any successor Y, and the latter is lost instead of message X. Accordingly, X or Y is put on *LM* in the next issued *'report'* PDU to act as a remulticast request. If necessary, remulticast requests are repeated every *Retr_Cycle* seconds.

4.4 ’Report’ PDU Heartbeat

The protocol attempts to maintain a uniform *’report’* PDU cycle throughout the group. To do that, upon reception of some other protocol instance’s *’report’* PDU, DPG-*n* delays the multicast of its own for *Report_Cycle* seconds. However, DPG-*n* can waive this delay if graph **G’** in the received *’report’* PDU (say issued by DPG-*n’*) contains an arc that would be replaced or removed in **G**$\cup$**G’**. In that case DPG-*n* views the current knowledge of ’$\rightarrow$’ at DPG-*n’* as incomplete regardless of the fact that some messages recently received by DPG-*n* may not have been received and accounted for at DPG-*n’* at the instant it issued the *’report'* PDU.

4.5 Flow Control

The flow control mechanism in DPG-*n* is supposed to counteract ordering buffers overflow and premature removal of messages from **G**. Locally delivered messages concluded to have also been delivered at all currently present sites are removed from **G** at a limited rate depending on the current throughput bottleneck estimate, called *Message_Cycle*, that determines the minimum interval between successive multicasts of *'message'* PDUs. *Message_Cycle* is contained in *'report'* PDUs and incremented by a fixed amount *Delta* whenever an ordering buffer at DPG-*n* overflows; it is set to max{*Message_Cycle*, *Message_Cycle'*} upon reception of a *'report'* PDU containing *Message_Cycle'*, and decremented by *Delta* periodically every *Fc_Cycle* seconds (down to a predetermined lower bound).

5 Protocol Properties

Typically, a message *X* becomes deliverable upon marking the arc (*L_DELIV*,*X*), which means *L_DELIV*$\Rightarrow$*X* was inferred at some member site. The other possibility i.e., all immediate successors of *L_DELIV* reaching the limit of *Deliv_Cycle*, implies that *Deliv_Cycle* should be large enough to allow formation of identical successor sets at each member site. Granted that, the question is whether there is enough inter-site consistency regarding successors of *L_DELIV*. Observations 1 through 3 below state that **G** remains consistent with ’$\rightarrow$’ across a succession of linearisations. Observations 4 and 5 are conjectures based on the protocol specification and simulation, yet to be proved rigorously. $\mathbf{G}^{+}(X)$ and $\mathbf{G}^{-}(X)$ denote the subgraphs of successors and predecessors of *X* in **G**, and lin(**G**) the linearised graph.

Observation 1. Linearisation preserves consistency of $G^{+}(L\_DELIV)$ with ’$\rightarrow$’ in that $(X,Y)\in G^{+}(L\_DELIV)$ implies $X\rightarrow Y$.

Proof: Suppose that $(X_0,Y_0)\in \mathbf{G}^{+}(L\_DELIV)$ at DPG-*n*. Observe that the marking of the arc (X_0,Y_0) would mean that at some site, messages X_0 and Y_0 were received consecutively and without an intervening reception miss; consequently $X_0 \Rightarrow Y_0$. Assume then that arc (X_0,Y_0) is not marked, which leaves two possibilities:

1) Messages X_0 and Y_0 were received consecutively at some DPG-*n'* which thus included arc (X_0,Y_0) into its precedence graph **G'** subsequently disseminated in a *'report'* PDU. Then, by virtue of the 1C specification (Sec. 3), $X_0 \rightarrow Y_0$.

2) Arc (X_0,Y_0) was included into **G** via replacing some arc (X_1,Y_0) or (X_0,Y_1). Consequently, (X_1,X_0) is marked (i.e., $X_1 \Rightarrow X_0$) or (Y_0,Y_1) is marked (i.e., $Y_0 \Rightarrow Y_1$). Considering in turn how each of the two replaced arcs could have been included into **G**, we have again two possibilities analogous to 1) and 2). Repeating this reasoning $i+j$ times we conclude the existence in **G** of vertices X_0, X_1, ..., X_i and Y_0, Y_1, ..., Y_j such that $X_i \Rightarrow ... \Rightarrow X_1 \Rightarrow X_0$ and $Y_0 \Rightarrow Y_1 \Rightarrow ... \Rightarrow Y_j$ and $(X_i,Y_j) \in \mathbf{G}^+(L\_DELIV)$. Supposing that (X_i,Y_j) was included into **G** via possibility 2), one should similarly deduce the existence of another vertex, either X_{i+1} or Y_{j+1}. Yet graph **G** remains finite within a group connection, thus eventually for some i, j we will see that (X_i,Y_j) was included into **G** via possibility 1). Therefore $X_i \rightarrow Y_j$ and so $X_0 \rightarrow Y_0$.

Observation 2. Suppose that a reception of a *'message'* PDU results in extending graph **G** by a subgraph $\mathbf{G}_e$, then lin($\mathbf{G} \cup \mathbf{G}_e$) contains **G**. (This property guarantees that the local quasi-ADO stabilises in time despite reception of new messages.)

Proof: Suppose that DPG-*n* has received message *Z*. Then arcs of the form (X,Z) are included into **G**, where $X \in \mathbf{G}$ and $\mathbf{G}^+(X)=\varnothing$. This may create a directed path to *Z* and so lead to removing some arcs, but only ones of the form (Y,Z) i.e., not in **G**. If (L_REC,Z) is marked then some arcs can be replaced, but in this case $\mathbf{G}^+(L\_REC)=\varnothing$ and thus again only arcs of the form (X,Z) i.e., not in **G**, are so endangered.

Observation 3. Linearisation is commutative in the sense that lin[lin($\mathbf{G} \cup \mathbf{G'}$)$\cup \mathbf{G''}$] = lin[lin($\mathbf{G} \cup \mathbf{G''}$)$\cup \mathbf{G'}$]. (This property guarantees that eventually, a locally obtained quasi-ADO is independent of the order of reception of *'report'* PDUs.)

Proof: We show that the order of replacing or removing arcs is immaterial i.e., the arc set of lin(**G**) is fully determined by that of **G**. From the proof of Observation 1 we conclude that $(X_0,Y_0) \in$ lin(**G**) iff $\mathbf{G}^+(X_0) \cap \mathbf{G}^-(Y_0)=\varnothing$ and there exist vertices $X_1,...,X_i$ and $Y_1,...,Y_j$ such that $(X_i,Y_j) \in \mathbf{G}$, (X_k,X_{k-1}) is marked, for all $k=1..i$ and (Y_{k-1},Y_k) is marked, for all $k=1..j$. Neither of these conditions has to do with the order of operations performed during a linearisation.

Observation 4. On return after an absence period, a protocol instance is free to resume the local construction of quasi-ADO once it has reset the connection status. (To minimise the risk of a message tag collision, prior reception of a number of *'report'* PDUs is recommended.)

Observation 5. Given Eventual Receipt and proper DPG configuration, the D_n's remain consistent with quasi-ADO for an arbitrarily long period of time with Extended Virtual Synchrony preserved.

6 Implementation and Performance Measurements

We shall focus on group throughput defined as the average message delivery rate throughout a group connection. Note that the DPG protocol (in fact, any GCP) largely reduces the PDU transport potential of the underlying network since construction of (quasi-)ADO across a large group is a more demanding task than providing a FIFO point-to-point service.To investigate the group throughput, it was decided to implement the DPG protocol in an available LAN environment.

DPG-*n* has been implemented as about 2000 lines of ANSI C code to run on a Pentium 100..333 MHz workstation under Linux (kernel version 2.0.36). Up to 17 such workstations were configured into a group interconnected by a switched 10 Mb/s Ethernet. The 1C service specification was found to hold, and the single-access LAN version of the ContiguityDetector was adopted. *Report_Cycle* was set so as to keep the *'report'* PDU traffic within 20% of the total message traffic, while the flow control was optimised towards maximum group throughput.

The implementation uses standard UDP broadcast interface, as shown in Fig. 2. One BSD socket is used for multicast and a separate one, configured as an asynchronous I/O port, for reception. At the top, a message generator/absorber inputs a stream of messages, thereby assuming the role of an application instance.

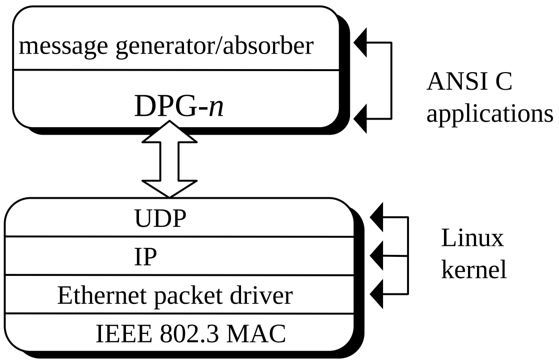

Fig. 2. DPG implementation layering

A few lessons (not necessarily applicable to WAN environments) have been learned regarding the DPG protocol performance in the specific LAN environment:

- Multiple reception buffers are needed instead of just one assumed for argument in Sec. 3.1 (20 in our implementation), with the ContiguityDetector inference rules appropriately redefined; the induced miss rate remains then well below 1%, and the resulting ADO violation rate is on order of one message in many thousands.
- Local clocks sometimes prove not accurate enough for flow control; in a recent version of our implementation, *Message_Cycle*, *Deliv_Cycle*, *Retr_Cycle* and *Report_Cycle* are all expressed in terms of the number of *'message'* PDUs received since the event that triggered the respective timer.

- Standard MAC interfaces do not receive what they are transmitting – each site behaves as if it misses own messages, which reduces the benefits of the ContiguityDetector and may lead to the graph **G** growing too large to be processed in real time; we have tried a Suspended Delivery option whereby upon multicast of a message *Z*, *X*⇒*Y* is inferred for all subsequently received messages *X*, *Y*, ... (disregarding the apparent miss of message *Z*), whose delivery is however suspended until reception of a *'report'* PDU with a graph **G'** containing vertex *Z*.

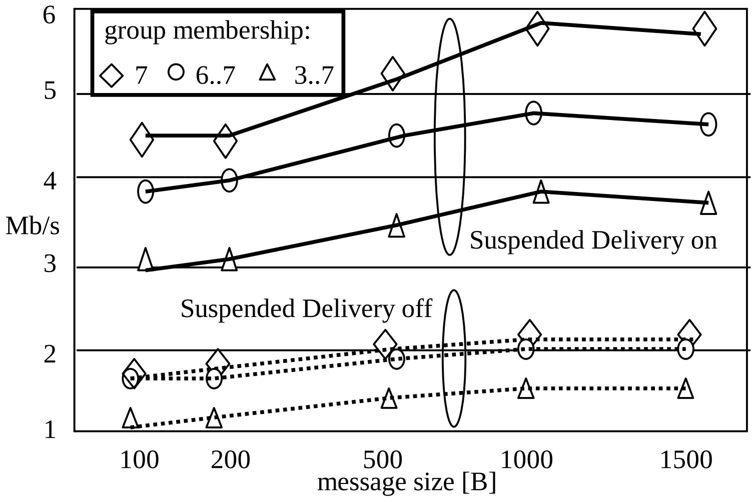

Fig. 3 Group throughput measurements for the DPG protocol

Measurements for a 7-site group reveal the significance of the latter option (Fig. 3). With it, the graphs **G** at sites rarely exceed 30 vertices and the group throughput is practically limited only by the PCs' processing rates. For example, over 740 1000-byte messages are delivered in ADO per second, with a graceful drop under variable membership, reflecting only the diminishing contribution of the fewer sites present, and not the reconfiguration overhead typical of existing ADO protocols [1], [6], [14], [15]. Switching off the Suspended Delivery option was not unlike switching off the ContiguityDetector altogether and typically resulted in reducing the group throughput to one-third of the previous value or less.

7 Conclusion

GCPs designed to serve collective output-oriented applications should be concerned about ADO, but much less about Membership Control. Under variable group membership, such protocols should ideally exhibit a group throughput drop that reflects only the diminishing contribution of the fewer member sites. We have proposed a membership-insensitive quasi-ADO protocol, built on top of the 1C network service complete with a conceptual inference device called ContiguityDetector, tailored to communication environments where message sequencing comes as a 'natural' feature e.g., single-access LANs, bounded-delay or tag-sequencing WANs. The proposed protocol has been implemented in a 10 Mb/s LAN environment to verify that it is able to maintain a high group throughput of messages delivered in quasi-ADO. A challenging issue is running the protocol in a larger group (50 or more sites); at the moment this is only possible via simulation.

References

1. Amir Y., Moser, L.E., Melliar-Smith, P.M., Agarwal, D.A., Ciarfella, P.: Fast Message Ordering and Membership Using a Logical Token-Passing Ring. In: Proc. 14th Int. Conf. on Distributed Computing Systems (1993) 551-560
2. Birman, K.P., Schiper, A., Stephenson, P.: Lightweight Causal and Atomic Group Multicast. ACM Trans. on Computer Systems 9 (1991) 272-314
3. Bruck, J., Dolev, D., Ho, C., Orni, R., Strong, R.: PCODE: An Efficient and Reliable Collective Communication Protocol for Unreliable Broadcast Domains. ftp://cs.huji.ac.il/pub/TRANSIS (1994)
4. Cheriton, D.R., Skeen, D.: Understanding the Limitations of Causally and Totally Ordered Communication. In: Proc. 14th ACM Symp. on Operating System Principles (1993) 44-57
5. Diot, C., Dabbous, W., Crowcroft, J.: Multipoint Communication: A Survey of Protocols, Functions and Mechanisms. IEEE J. on Selected Areas in Comm. 15 (1997) 277-290
6. Dolev D., Kramer, S., Malki, D.: Early Delivery Totally Ordered Multicast in Asynchronous Environments. In: Proc. 23rd Int. Symp. on Fault-Tolerant Computing (1993) 544-553
7. Ezhilchelvan, P.D., Macedo, R.A., Shrivastava, S.K.: Newtop: A Fault-Tolerant Group Communication Protocol. In: Proc. 15th Int. Conf. on Distributed Computing Systems (1995) 336-356
8. Fischer, M., Lynch, N.A., Paterson, M.S.: Impossibility of Distributed Consensus with One Faulty Process. J. of the ACM 32 (1985) 374-382
9. Hadzilacos, V., Toueg, S.: Fault-Tolerant Broadcasts and Related Problems. In: Mullender, S. (ed.): Distributed Systems, Addison-Wesley, Wokingam Reading (1993) 97-145
10. Melliar-Smith, P.M, Moser, L.E., Agrawala, V.: Broadcast Protocols for Distributed Systems. IEEE Trans. on Parallel and Distributed Systems 1 (1990) 17-25
11. Moser L.E., Amir, Y., Melliar-Smith, P.M., Agarwal, D.A.: Extended Virtual Synchrony. In: Proc. 14th Int. Conf. on Distributed Computing Systems (1994) 56-65
12. Nakamura, A., Takizawa, M.: Priority-Based and Semi-total Ordering Broadcast Protocols. In: Proc. 12th Int. Conf. on Distributed Computing Systems (1992) 178-185
13. Obraczka, K.: Multicast Transport Protocols: A Survey and Taxonomy. IEEE Comm. Magazine 36 (1998) 94-102
14. Rodrigues, L., Fonseca, H., Verissimo, P.: Totally Ordered Multicast in Large-Scale Systems. In: Proc. 16th Int. Conf. on Distributed Computing Systems (1996) 503-510.
15. Whetten, B., Montgomery, T., Kaplan, S.: A High-Performance Totally Ordered Multicast Protocol. In: Theory and Practice in Distributed Systems, Lecture Notes in Computer Science, Vol. 938, Springer-Verlag, Berlin Heidelberg New York (1994)
16. Wilhelm, U., Schiper, A.: A Hierarchy of Totally Ordered Multicasts. http://www-uk.research.ec.org/broadcast/ trs/papers/85.ps