Multicast with Aggregated Deliveries

Gregory Aaron Wilkin Patrick Eugster
Department of Computer Science
Purdue University
West Lafayette IN-47906, USA
{gwilkin, peugster}@cs.purdue.edu

Abstract

An increasing number of distributed systems relies on forms of message correlation, which result in atomic delivery of multiple messages aggregated by following process-specific criteria. Generally, more than one process is aggregating messages, implying that messages are multicast. While delivery guarantees for multicast scenarios with single message delivery are well understood, existing systems and models for aggregated deliveries either consider only unicast, centralized setups, or focus on efficiency thus providing only best-effort guarantees. This paper investigates the foundations of Multi-Delivery Multicast (MDMcast) in asynchronous distributed systems with crash-stop failures. We first describe a succinct aggregation model with a concise and generic predicate grammar for expressing conjunctions on messages and properties for a corresponding multicast primitive, which we term Conjunction-MDMcast (C-MDMcast). We show that for processes interested in identical conjunctions to achieve agreement on delivered messages, a total order on individual messages (or equivalent oracle) is not only useful but necessary, which is clearly opposed to single-message deliveries. We show this indirectly by exhibiting an algorithm implementing C-MDMcast on top of Total Order Broadcast (TOBcast) and vice-versa for a majority of correct processes. Then, we extend our predicate grammar with disjunctions, leading to the specification of Disjunction-MDMcast (D-MDMcast). We exhibit an algorithm implementing D-MDMcast, derived from our algorithm implementing C-MDMcast. We formalize several additional properties for both of our specifications, including ordering properties on aggregated messages and a notion of agreement capturing non-identical yet “related” conjunctions, and show how our respective algorithms implement these.

Keywords: aggregation, multicast, properties, order

1 Financially supported by NSF grants number 0644013 and 0834529, and DARPA grant number N11AP20014.
2 Contact Address: 305 N. University St, West Lafayette, IN Contact Phone: 765 494 0739
1 Introduction

Several fundamental models of distributed systems leverage relationships among events [22], and an increasingly large number of distributed applications are explicitly built on a form of message correlation. A traditional use of correlation consists in the verification of safety conditions for intrusion detection [21]. Network monitoring, more generally [20], also enables the improvement of resource usage, for instance, in data centers. Workflow monitoring and production chain management are further scenarios [24, 25]. More recent application environments for correlation include embedded and pervasive systems [16], and sensor networks [27].

Seminal investigations of message correlation were conducted in the context of active databases [8, 15, 14]. Since there can be many options in syntax and semantics of elementary correlation, early work in active databases research attempted to formalize those options. A message $m_k$ in the following represents a message of type $T_k$. A sequence of messages $m_1 \cdot m_2 \cdot m_3$ can be matched by a “subscription” correlating instances of two message types $T_1$ and $T_2$ as $[m_1, m_2]$ (first received first) or as $[m_2, m_1]$ (most recent first). However, such work, just like stream processing [3 11], considers events to be unicast, or focuses on individual processes, centralized setups, or synchronous systems: StreamCloud [17] ensures that operations split into sub-operations executing in parallel behave equivalently to non-parallel ones. Ordering is achieved based on timestamps assuming well-synchronized clocks.

Message aggregation has also been investigated in the context of content-based publish/subscribe systems [7], which focuses on multicast. Several systems including Gryphon [29], PADRES [24] and Hermes [26], have been extended to support some form of correlation, broadening their scope from the canonical stock quote dissemination example for publish/subscribe systems to expressive algorithmic stock trading. However, most such extensions focus on matching efficiency and complexity or on the number of possible aggregations and thus yield only best-effort guarantees on message delivery unless relying on centralized rendezvous nodes [26], which present a single point of failure. The absence of guarantees can have adverse affects [28]. Consider, for example, monitoring a network to decide which one of two gateways to route certain traffic through. If the first gateway receives the sequence $m_1 \cdot m_2 \cdot m_3$ outlined above, but the second one receives the sequence $m_2 \cdot m_1 \cdot m_3$ instead, each gateway might consider itself to be responsible for routing. Worse even, each can consider the other to be responsible. Of course, individual systems can be designed to deal with some of these issues (e.g., by using a proxy process to merge and multiplex streams to replicas), but corresponding solutions are hardly generic and can easily introduce bottlenecks to performance and dependability.

While in single event (message) delivery scenarios several families of properties have been proposed and rigorously investigated (e.g., agreed delivery [18], probabilistic delivery [6], ordering properties [13]), the feasibility of guarantees in the presence of atomic, aggregated deliveries of sets of messages which are multicast in asynchronous distributed systems remains unexplored.

This paper is, to the best of our knowledge, the first to fill this gap, namely through the following contributions:

- A simple model of multicast with aggregated message delivery is proposed for the crash-stop failure model. The model includes a basic predicate grammar for subscriptions of processes supporting message correlation in the form of message conjunctions. We term this specification Conjunction Multi-Delivery Multicast (C-MDMcast).
- We show that to achieve agreement on delivered messages (message aggregates) among processes subscribing with identical conjunctions, total order on individual messages is both useful (as intuitively conveyed by the example above) and necessary in addition to agreement. We show this by exhibiting an algorithm FRIP implementing C-MDMcast on top of Total Order Broadcast (TOBcast) and vice-versa with a majority of correct processes. This is opposed to single message deliveries where total order and agreement can be separated\(^1\) and constitutes a fundamental result since Total Order Broadcast has been shown to be equivalent to the basic Consensus problem, which is unsolvable in asynchronous distributed systems if a single process can fail [12].
- We specify a stronger agreement property on conjunctions, which formalizes the intuition that the aggregated messages delivered in response to a first subscription, which “covers” a second subscription, should include the set of messages delivered to the latter one. Such subsumption is trivial in single-message deliveries (and in fact is paramount to scalability in publish/subscribe systems [2]) but more involving when the delivery of a message depends on others. We prove that FRIP implements this stronger agreement property.

\(^1\)Agreement and (total) order are often intertwined by flavors of liveness in ordering properties [4] but they can be separated in single message delivery scenarios [13].
• We add disjunctions to our predicate grammar and define corresponding properties, leading to the problem of Disjunction Multi-Delivery Multicast (D-MDMcast). We exhibit a derivation D-FRIP of our algorithm FRIP, which implements D-MDMcast.

• We formulate total order properties for conjunctions and disjunctions: A total order being required on individual messages to achieve agreement on aggregated deliveries, we can leverage this order to establish a total order on aggregated deliveries for scenarios desiring agreement.

Note that the goal of this paper is not to exhibit the weakest failure detector [9] for correlation or to give guidelines on how to devise correlation multicast systems. The intent is more to explore (in-)feasibilities, e.g., to show that some ordering or equivalent oracle is required to achieve agreement (and not ordering) on aggregated deliveries.

Roadmap. Section 2 presents background information such as the system model assumed. Section 3 introduces our model of multi-message delivery and formalizes delivery properties for conjunctions. Section 4 compares C-MDMcast with TOBcast. Section 5 presents extensions including subsumption, disjunctions, and ordering properties on message sets. Section 6 concludes with final remarks. Appendix A presents proofs and issues of extending order properties.

2 Preliminaries

We first introduce the system model considered and notations.

2.1 System Model

We assume a system Π of processes, Π={p₁, ..., pₙ} interconnected pairwise by reliable channels [5] offering primitives to SEND (non-blocking) and receive (RECV) messages. We consider a crash-stop failure model [12], i.e., a faulty process may stop prematurely and does not recover. Further, we assume the existence of a discrete global clock to which processes do not have access and that an algorithm run R consists in a sequence of events on processes. That is, similar to other models [11], one process performs an action per clock tick which is either of (a) a protocol action (e.g., RECV), (b) an internal action, or (c) a “no-op”.

A failure pattern F is a function mapping clock times to processes, where F(t) gives all the crashed processes at time t. Let crashed(F) be the set of all processes ∈ Π that have crashed during R. Thus, for a correct process pi, pi ∈ correct(F) where correct(F) = Π − crashed(F) [10].

2.2 Properties and Total Order Broadcast

For brevity and clarity, we adopt in the following a more formal notation for properties than common. Consider for instance the well-known problem of Total Order Broadcast (TOBcast) [18] defined over primitives TO-BCAST(m) and TO-DLVR(m), which will be used for comparison later on. We denote TO-DLVRᵢ(m) as the TO-delivery of message m by process pi at time t, and similarly, TO-BCASTᵢ(m) denotes the TO-broadcasting of m by pi at time t. We elide any of i, t, or m when not germane to the context. We write ∃e for an event e such as a SEND or TO-BCAST as a shorthand for ∃e ∈ R. The specification of Uniform TOBcast thus becomes:

TOB-No Duplication  ∃TO-DLVRᵢ(m) ⇒ \#TO-DLVRᵢ(m)' ∈ t' ≠ t
TOB-No Creation  ∃TO-DLVRᵢ(m) ⇒ ∃TO-BCASTᵢ(m)' ∈ t' < t
TOB-Validity  ∃TO-BCASTᵢ(m) ∧ pi ∈ correct(F) ⇒ ∃TO-DLVRᵢ(m)
TOB-Agreement  ∃TO-DLVRᵢ(m) ⇒ ∀pj ∈ correct(F)\{pi\} ∃TO-DLVRᵢ(m)
TOB-Total Order  ∃TO-DLVRᵢ(m)[j], TO-DLVRᵢ(m')[j], TO-DLVRᵢ(m)[j'], TO-DLVRᵢ(m')[j'] ⇒ (i < t ⇒ t_j < t_j')

TOB-Agreement is visibly a uniform property. Property TOB-Total Order corresponds to Strong Uniform Total Order (SUTO) in [4].

3 Conjunction Multi-Delivery Multicast (C-MDMcast)

In this section, we present a basic specification of multicast with message conjunctions introducing the problem of Conjunction Multi-Delivery Multicast (C-MDMcast).
3.1 Predicate Grammar

Sets of delivered messages — referred to as relations — represent messages aggregated according to specific subscriptions. Such subscriptions are combinations of predicates on messages expressed in disjunctive normal form (DNF) according to the following grammar:

\[
\begin{align*}
\text{Subscription} & : \quad \Phi \mid \Phi \lor \Psi \\
\text{Conjunction} & : \quad \rho \mid \rho \land \Phi \\
\text{Predicate} & : \quad T[i],a \ op \ v \mid T[i],a \ op \ T[i],a \mid T[i] \mid \top
\end{align*}
\]

\(T[i],a\) denotes an attribute \(a\) of the \(i\)-th instance of type \(T\ (T[i])\) and \(v\) is a value. As syntactic sugar, we can allow predicates to refer to just \(T.a\), which can be automatically translated to \(T[1].a\). We may use this in examples for simplicity. A type \(T\) is characterized by an ordered set of attributes \([a_1, \ldots, a_n]\) each of which has a type of its own – typically a scalar type such as Integer or Float. A message \(m\) of type \(T\) is an ordered set of values \([v_1, \ldots, v_n]\) corresponding to the respective attributes of \(T\). By abuse of notation but unambiguously, we write \(\rho_l \in \Phi \iff \Phi = \rho_1 \land \ldots \land \rho_k\) and \(l \in [1..k]\).

To simplify properties further on, we also introduce the empty predicate \(\top\) which trivially yields \(true\). A predicate that compares a single message attribute to a value or compares two message attributes on the same message, i.e., on the same instance of a same type (e.g., \(T_k[i],a \ op T_k[i],a\)) is a unary predicate. When two distinct messages (two distinct types or different instances of the same type) are involved, we speak of binary predicates \((T_k[i],a \ op T_l[j],a', \ k \neq l \lor i \neq j)\). Predicates comparing an attribute of a type instance to itself \((T_k[i],a \ op T_k[i],a)\) constitute useless operations and are prohibited. We also allow wildcard predicates of the form \(T\) (or \(T_1\)) to be specified; such predicates simply specify a desired type \(T\) of messages of interest. \(T[i]\) implicitly also declares \(T[k] \forall k \in [1..i-1]\) if these are not already explicitly declared as part of other predicates in the same subscription.

A process \(p_j\)'s subscription is referred to as \(\Psi(p_j)\). By abuse of notation but unambiguously, we sometimes handle disjunctions or conjunctions as sets (of conjunctions and predicates respectively). We write, for instance, \(\rho_l \in \Phi \iff \Phi = \rho_1 \land \ldots \land \rho_k\) with \(l \in [1..k]\), or \(\Phi_r \in \Psi \iff \Psi = \Phi_1 \lor \ldots \lor \Phi_n\) with \(r \in [1..n]\). For simplicity, we first consider a subscription to consist in a single conjunction in the context of C-MDMcast. We handle disjunctions later on.

3.2 Predicate Types and Evaluation

We assume that a deterministic order \(\prec\) exists within subscriptions based on the names of message types, attributes, etc., which can be used for re-ordering predicates within and across conjunctions. This ordering can be lexical or based on priorities on message types, and is necessary for even simplest forms of determinism and agreement. We consider subscriptions to be already ordered accordingly for simplicity of presentation.

The number of messages involved in a subscription is given by the number of types and corresponding instances involved. More precisely, the types involved in a subscription are represented as sequences. The same type can be admitted multiple times. Such sequences can be viewed as the signatures of predicates, defined as follows:

\[
\begin{align*}
\top(\Phi \lor \Psi) & = \top(\Phi) \lor \top(\Psi) & \top(T[i],a \ op v) & = \top(T[i]) \\
\top(\rho \land \Phi) & = \top(\rho) \land \top(\Phi) & \top(T[j]) & = \emptyset \\
\top(T_1[i],a_1 \ op T_2[j],a_2) & = \top(T_1[i]) \lor \top(T_2[j]) & \top(T[i]) & = [T_1, \ldots, T_i]
\end{align*}
\]

\(\lor\) stands for in-order union of sequences defined below:

\[
\emptyset \lor [T, \ldots] = [T, \ldots] \quad [T, \ldots] \lor \emptyset = [T, \ldots]
\]

\[
[T_1, \ldots, T_j, T'_1, \ldots] \lor [T_2, \ldots, T'_2, \ldots] = \begin{cases} 
[T_1, \ldots, T_j] \lor ([T_1', \ldots] \lor [T_2, \ldots, T'_2, \ldots]) & T_1 \prec T_2 \\
[T_2, \ldots, T'_2] \lor ([T_2', \ldots] \lor [T_1, \ldots, T_j, T'_1, \ldots]) & T_2 \prec T_1 \\
\max(i,j) \times [T_1, \ldots, T_i] \lor ([T_i', \ldots] \lor [T_2, \ldots]) & T_1 = T_2
\end{cases}
\]

Above, \(\lor\) represents simple concatenation.
Any subscription $\Phi$ thus involves a sequence of message types $T(\Phi)=[T_1, ..., T_n]$ where we can have for $i, j \in [1..n], i < j$ such that $\forall k \in [i..j] T_k = T_i = T_j$, that is, a subsequence of identical types. These represent a stream of messages of the respective type of length $j-i+1$.

A subscription is evaluated for an ordered set of messages $[m_1, ..., m_n]$, where $m_i$ is of type $T_i$. We assume that types of values in predicates are checked statically with respect to the types of messages. $T(m)$ returns the type of a given message $m$. Note that we do not introduce a set of uniquely identified types $\{T_1, T_2, ...\}$. This allows for the set of types to be unbound which does not violate our assumptions or guarantees, and keeps notation more brief in that we can use $[T_1, ..., T_k]$ to refer to a sequence of $k$ arbitrary types, as opposed to something of the form $[T_{i_1}, ..., T_{i_k}]$.

The evaluation of a conjunction $\Phi$ on a relation is written as $\Phi[m_1, ..., m_n]$. Finally, for evaluation of an attribute $a$ on a message $m_i$, we write $m_i.a$. Evaluation semantics for predicates are defined as follows:

$$
(\Phi \lor \Psi)[m_1, ..., m_n] = \Phi[m_1, ..., m_n] \lor \Psi[m_1, ..., m_n]
$$

$$
(\rho \land \Phi)[m_1, ..., m_n] = \rho[m_1, ..., m_n] \land \Phi[m_1, ..., m_n]
$$

$$
(T[i].a \text{ op } v)[m_1, ..., m_n] = \begin{cases} m_{k+i-1}.a \text{ op } v & T(m_k) = T \land (T(m_{k-1}) \neq T \lor (k-1) = 0) \\ false & otherwise \end{cases}
$$

$$
(T_1[i].a_1 \text{ op } T_2[j].a_2)[m_1, ..., m_n] = \begin{cases} m_{k+i-1}.a_1 \text{ op } m_{l+j-1}.a_2 & T(m_k) = T_1 \land (T(m_{k-1}) \neq T_1 \lor (k-1) = 0) \\ \land T(m_l) = T_2 \land (T(m_{l-1}) \neq T_2 \lor (l-1) = 0) & otherwise \end{cases}
$$

$$
(T)[m_1, ..., m_n] = true
$$

$$
(\top)[m_1, ..., m_n] = true
$$

(by def. of signatures)

Parentheses are used for clarity. For brevity we write simply $\Phi[...]$ for $\Phi[...] = true$.

We consider the DLVR primitive to be generically typed, i.e., we write $DLVR_\Phi ([m_1, ..., m_n])$ for delivering a relation $[m_1, ..., m_n]$ where $m_i$ is of type $T_i$ such that $T(\Phi)=[T_1, ..., T_n]$. Analogous to TOBcast, $DLVR_\Phi ([m_1, ..., m_n])_t$ defines the delivery event of a message $m$ on process $p_i$ in response to $\Phi$ at time $t$ and $MCAST^{a}(m)_t$ defines the multicasting of a message $m$ by $p_i$ at time $t$ etc. may be omitted when not germane to the context.

### 3.3 Properties

Conjunction Multi-Delivery Multicast (C-MDMcast) is defined over primitives MCAST and DLVR, where DLVR is parameterized by a “subscription” $\Phi$ and delivers ordered sets of messages. In the remainder of this paper, deliver refers to DLVR (while TO-deliver refers to TO-DLVR), and multicast refers to MCAST (vs. TO-broadcast).

#### 3.3.1 Basic Safety Properties

Basic safety properties for C-MDMcast are MDM-NO DUPLICATION, MDM-NO CREATION and ADMISSION:

**MDM-NO DUPLICATION**  $\exists DLVR_\Phi([.., m, ..])_t \Rightarrow \#DLVR_\Phi([.., m, ..])_t' \mid t' \neq t$

**MDM-NO CREATION**  $\exists DLVR_\Phi([.., m, ..])_t \Rightarrow \#MCAST^{m}_t' \mid t' < t$

**ADMISSION**  $\exists DLVR_\Phi([m_1, ..., m_n]) \mid T(\Phi) = [T_1, ..., T_n] \Rightarrow \Phi \in \Psi(p_i) \land \Phi[m_1, ..., m_n] \land \forall k \in [1..n] : T(m_k) = T_k$

The MDM-NO DUPLICATION property implies that a same message is delivered at most once on any single process for a conjunction, which may be opposed to certain systems that allow a same message to be correlated multiple times. Our property could easily be substituted to allow a delivery for every instance of a type in a conjunction which would, however, make the guarantees and proofs more complicated without affecting the feasibilities explored in this paper.

#### 3.3.2 Liveness

ADMISSION can trivially hold while not performing any deliveries. We have to be careful about providing strong delivery properties on individually multicast messages though, as messages may depend on others to match a given conjunction. Nonetheless, we want to rule out bogus implementations, which simply discard all messages. We thus propose the two following complementary liveness properties:

**CONJUNCTION VALIDITY**  $\exists MCAST^a([m^k_j], k \in [1..n], l \in [1..\infty] \land p_i \in correct \land \exists \Phi \in \Psi(p_i) \mid \Phi[m^1_l, ..., m^n_l] \Rightarrow \exists DLVR_\Phi([..])_{t_j} \mid j \in [1..\infty]$
Conjunction-MDMcast
Disjunction-MDMcast
Enqueue, Match, Dequeue
Total Order Broadcast
Reliable Channels

Figure 1: Layered structure.

MESSAGE VALIDITY \( \exists \text{MCAST}(m^x), \text{MCAST}^k(m^x)_k, k \in [1,n], x, l \in [1,\infty] \mid \{p_i,p_j,p_k\} \subseteq \text{correct} \wedge \Phi \in \Psi(p_j) \wedge T(\Phi) = [T_1,...,T_n] \wedge \forall z \in [w,y]T_z = T(m^x) \wedge \exists(T(m^x)|x-w+1,a_1 \text{ op } T[r],a_2) \in \Phi \mid (T \neq T(m^x) \lor r \neq x-w+1) \wedge \Phi[m^x_1,...,m^x_{w-1},m^x,m^x_{w+1},...,m^x_t] \Rightarrow \exists DLVR^t_\Phi([...,m^x,...]) \)

These two properties deal with the two possible cases that can arise. The first property deals with dependencies across messages and can be paraphrased as follows: “If for a correct process \( p_i \), there is an infinite number of relations of matching messages that are successfully multicast, then \( p_i \) will deliver infinitely many such relations.” This property is reminiscent of the \( \text{FINITE LOSSES} \) property of fair-lossy channels \([5]\). It allows matching algorithms to discard \textit{some} messages for practical purposes such as agreement and ordering, yet ensures that when matching messages are continuously multicast a corresponding process will continuously deliver.

MESSAGE VALIDITY provides a property analogous to validity for single-message deliveries (e.g., TOBcast): If a message is multicast by a correct process \( p_i \), and its delivery in response to a conjunction on some correct process \( p_j \) is not conditioned by binary predicates with other message types, then the message must be delivered by \( p_j \) if matching messages of all other types are continuously multicast. This latter condition is necessary because the delivery of the message even in the absence of binary predicates requires the \textit{existence} of other messages. The condition also ensures that any unary predicates on the respective message type are satisfied. In the case of multiple instances of that type, for each of which there are only unary predicates that match, the property does not force the message to be delivered more than once as the position of the message is not fixed in the implied delivery.

Note that none of these properties is impacted by the presence of multiple instances of a same type in a conjunction. An infinite flow of messages of some type implies multiple (a finite number of) infinite flows of that type.

3.3.3 Agreement

The properties so far ensure that as long as there are matching messages being multicast, processes will eventually deliver relations. We are, however, interested in providing stronger properties for these delivered relations, especially across processes. We define \textsc{Conjunction Agreement} below:

\textsc{Conjunction Agreement} \( \exists DLVR^t_\Phi([m_1,...,m_n]) \Rightarrow \forall p_j \in \text{correct}(F) \setminus \{p_i\} \mid \Phi \in \Psi(p_j) : \exists DLVR^t_\Phi([m_1,...,m_n]) \)

\textsc{Conjunction Agreement} guarantees that two correct processes \( p_i \) and \( p_j \) with identical subscriptions expressed by the conjunction \( \Phi \) must deliver the same relation, \textit{without constraining the respective orders of such deliveries}. Quite obviously, it is a uniform property.

4 Comparison of C-MDMcast with Total Order Broadcast

In this section, we show that Total Order Broadcast, as defined in Section \([2.2]\) can be used to implement C-MDMcast and vice-versa with a majority of correct processes. This substantiates the intuition that a total order on messages or an equivalent oracle is not only useful to achieve agreement on conjoined messages, but also necessary.

4.1 Conjunction Multi-Delivery Multicast (C-MDMcast) using Total Order Broadcast (TOBcast)

We first present an algorithm to implement C-MDMcast using TOBcast (as specified in Section \([2.2]\), called FRIP (First received matching with infix+prefixed disposal). This algorithm exploits the total order on messages created by TOBcast to reach agreement on delivered relations. Fig. \([1]\) represents a layered structure for MDMcast. It includes D-MDMcast, which represents an extension of the basic C-MDMcast, which we introduce later.

Our FRIP algorithm in Alg. \([1]\) can be broken down into several components for performing the aggregation of messages, namely (1) the buffering of TO-delivered messages (ENQUEUE), (2) the actual matching of messages (MATCH), and (3) the disposal of messages after matching (DEQUEUE).
Executed by every process $p_i$

1. init
2. $\Psi \leftarrow \Phi$
3. $\Phi \leftarrow p_1 \land \ldots \land p_n$ \{Composite subscription, $|\Phi| = m$\}
4. $Q[T(m)] \leftarrow \emptyset$ \{Msg queues; parameterized by type $T$\}
5. TO-MCAST($m$):
6. TO-BCAST($m$)
7. procedure DEQUEUE($[m_1, \ldots, m_l], Q$) \{Garbage collection\}
8. for all $Q[T(m)] = \ldots \oplus m_k \oplus m \oplus \ldots$, \(k \in [1..l]\) do
9. $Q[T(m_k)] \leftarrow m \oplus \ldots$
10. function ENQUEUE($m$, $Q$, $\Phi$) \{Queue management\}
11. win $\leftarrow \max(j \mid \exists \ldots T(m)[j].a \ldots \in \Phi)$ \{All type instances\}
12. if $\forall j = 1..\text{win}$, $\exists T(m)[j].a \text{ op } v \in \Phi \mid \neg(m.a \text{ op } v)$
13. $\lor \exists T(m)[j].a \text{ op } T(m)[j].a' \in \Phi \mid \neg(m.a \text{ op } m.a')$
14. return false \{Useless due to unary predicates\}
15. else
16. $Q[T(m)] \leftarrow Q[T(m)] \oplus m$ \{Append $m$\}
17. return true
18. upon TO-DLVR($m$) do
19. if $T(m) \in T(\Phi)$ then
20. if ENQUEUE($m$, $\Phi$, $Q$) then
21. $[m_1, \ldots, m_l] \leftarrow \text{MATCH}([\emptyset, \Phi, Q])$
22. if $l > 0$ then \{Not an empty set\}
23. DEQUEUE($[m_1, \ldots, m_l]$)
24. function MATCH($[m_1, \ldots, m_n], \Phi, Q$) \{Recursive msg-match\}
25. $T \leftarrow T_{n+1}$ \{Next type\}
26. $l \leftarrow \max(j \mid Q[T] = m_1 \oplus \ldots \oplus m_j \oplus \ldots)$ \{Last\}
27. for all $k = (l + 1)..h \mid Q[T] = m_1 \oplus \ldots \oplus m_h$ do
28. $\Phi = \text{MATCH}([m_1, \ldots, m_h])$ \{Found a match\}
29. return $[m_1, \ldots, m_h, m_k]$ \{Found a match\}
30. return $E$ \{Found a match\}
31. else if $E = \text{MATCH}([m_1, \ldots, m_n, \Phi, Q]) \neq \emptyset$ then
32. return $E$
33. return $\emptyset$

Alg. 1: First received matching with infix+prefix disposal (FRIP) algorithm.

Every process $p_i$ has a subscription of one conjunction $\Phi$. $p_i$ maintains exactly one queue $Q$ per message type appearing in its conjunction (regardless of the number of instances of that type in its subscription). When TO-delivering a message, $p_i$ first checks whether the type of the message is in its subscription and, if so, attempts to ENQUEUE it. $Q[T(m)] \oplus m$ denotes appending a message $m$ to the queue of $m$’s type $T(m)$. The ENQUEUE primitive returns true if the message has been ENQUEUED, indicating it satisfies all unary predicates on the respective type in the conjunction. This tells the algorithm to proceed to MATCHING, as any received message can potentially complete a relation.

It is important that this matching is triggered deterministically on every process, and that the matching itself is deterministic. The procedure attempts to find the first instance of the first type in $\Phi$ for which there are messages of the remaining types with which all predicates in $\Phi$ are satisfied. Among all such possibilities, if any, the algorithm recursively seeks for a match with the first instance of the second type in $\Phi$ etc., until a match is found or all possibilities are exhausted. In the case of messages of the same type, a first instance of that type is recursively matched with the first follow-up instance of the same type until the number of messages needed for that type are matched, or until all possibilities in the queue for that type are exhausted. Thus in Alg. 1 on Line 28 $l$ denotes the last matched instance of the currently matched type. These semantics can be termed first received matching semantics (lending the first two letters of the algorithm name FRIP). Consider the example of Section 1 where messages of two types $T_1$ and $T_2$ are matched with wildcard predicates of the respective types. A message $m_{2 \ldots n}$ in the following represents a message of type $T_2$. A sequence $m_1^1 \cdot m_2^1 \cdot m_3^2$ received by a process $p_i$ will lead to the match $m_1^1, m_2^1, m_3^2$ with the above matching semantics, while a permutation of the sequence, $m_1^1 \cdot m_2^1 \cdot m_3^2$ will lead to the match $m_1^2, m_2^1, m_3^1$. A simple permutation across processes can thus lead to delivery of distinct relations, which intuitively conveys the need for total order.

The described matching algorithm performs an exhaustive search and is thus not efficient; however, it suffices to illustrate the relevant properties and can be represented concisely. More elaborate and efficient matching algorithms exist, which offer the same semantics. A common approach consists in storing partial matches in specialized data-structures for matching a given message effectively (e.g., [19]). As mentioned the goal of this paper is not to give guidelines on how exactly correlation-enabled multicast systems should be devised but to explore the bounds of correlation.

Upon a successful match, our FRIP algorithm in Alg. 1 discards not only consumed, matched messages, but also predating buffered ones. We refer to these semantics as infix+prefix disposal. More precisely, upon a successful match $[m_1, \ldots, m_n]$, for each message $m_i$, all messages of the same type received prior to $m_i$ are discarded via the garbage collection mechanism DEQUEUE. This algorithm, thus, achieves agreement since it is triggered deterministically and also behaves deterministically. Fig. 2 shows such an example for a conjunction $\Phi = \rho_1 \land \rho_2$ where $\rho_1 = T_1.a_1 < T_2.a_1$ and $\rho_2 = T_3.a_1 < 20$ (recall that $T_1.a_1$, $T_2.a_1$ and $T_3.a_1$ are shorthand for $T_1[1].a_1$, $T_2[1].a_1$ and $T_3[1].a_1$ respectively). The marked line shows a matched relation. The latest message received is of type $T_2$ with value 7. All messages in the respective queues in front of the matched messages are DEQUEUED.
Executed by every process \( p_i \)

1: \textbf{init}
2: broadcasts ← ∅ \{Output message buffer\}
3: tbdelivered ← ∅ \{To be delivered\}
4: seq ← 0 \{Last own message created\}
5: last_sent ← 0 \{Last own message sent\}
6: last_reply ← 0 \{Last own message received\}
7: last_delivered[i] ← 0 \{Seq # hash parameterized by proc ID\}

8: \textbf{task SENDER}
9: if last_recv = last_sent then \{no \( p_i \) msg correlated\}
10: if broadcasts ≠ ∅ then
11: C-MDMCAST([m, seq’, \( p_i \)]) \{\text{seq’} = \min\{seq” | [..., seq”] ∈ broadcasts\}
12: broadcasts ← broadcasts ∪ \{[m, seq’]\}
13: last_sent ← seq’
14: else
15: seq ← seq + 1
16: C-MDMCAST([⊥, seq, \( p_i \)])
17: last_sent ← seq

18: \textbf{To TO-BCAST(m)}:
19: seq ← seq + 1
20: broadcasts ← broadcasts ∪ \{[m, seq]\}
21: upon C-MDMCAST\( \Phi \rightarrow \)MIP \( \cup \) MIP \( \{[m, seq’, \( p_i \)]\}_{1..(f+1)} \) do
22: tbdelivered ← tbdelivered ∪ \{[m, seq’, \( p_i \)]\}_{1..(f+1)}
23: if \( \exists k \in [1..(f+1)] \) \( p_jk = p_i \) then
24: last_recv ← seq’
25: \textbf{task RECEIVER}
26: if \( \exists [m, seq’, \( p_j \)]\)_{1..(f+1)} ∈ tbdelivered \( \forall k \in [1..(f+1)] : seq’ = last_delivered[p_jk] + 1 \) then
27: tbdelivered ← tbdelivered \( \{[m, seq’, \( p_j \)]\}_{1..(f+1)} \)
28: for all \( k = 1..(f+1) \) do
29: last_delivered[p_jk] ← last_delivered[p_jk] + 1
30: if \( m_k \neq ⊥ \) then
31: TO-DLVR(m_k)

Alg. 2: Algorithm \( T_{C \rightarrow MDMcast \rightarrow TOBcast} \) implementing TOBcast using C-MDMcast.

**Theorem 1** FRIP implements C-MDMcast.

**Proof.** See Appendix A.1

### 4.2 Total Order Broadcast using Conjunction Multi-Delivery Multicast

Is total order on messages necessary for solving C-MDMcast? Alg. 2 describes an algorithm \( T_{C \rightarrow MDMcast \rightarrow TOBcast} \) to implement Total Order Broadcast over C-MDMcast assuming a majority \((f + 1)\) of correct processes, where \( f \) is the maximum number of processes that can fail in a run.

In short, the algorithm uses a single type of multicast message \( MIP \): This message type contains the actual application message of type \( M \), the sending process’s current sequence number as an integer \( I \) as well as the process’s identifier of type \( P \). Each process is interested in conjunctions consisting in a number of instances of \( MIP \) equal to the size of the majority partition of processes in the system. That is, \( \Phi = \bigwedge_{i=1..(f+1)} MIP[i] \), or more simply \( MIP[f+1] \).

We must ensure a total order among all processes. To do so, each process proceeds in lock-step manner. More precisely, every process at every time has a message that is “under correlation”, i.e., a message it has multicast but not yet delivered as part of a relation. This is ensured by task SENDER. If a process does not have any pending TO-broadcast application messages (a TO-BCAST message is simply added to a queue \( \text{broadcasts} \) of messages to be broadcast), it simply uses an empty message \( ⊥ \). This is necessary to ensure validity (i.e., an infinite sequence of messages) while a single process only multicasts a single message at a time – less than a majority of processes might be TO-broadcasting.

Since a process can very well deliver several relations that do not contain any of its own messages, and these relations are not necessarily delivered by the underlying C-MDMcast layer in the same order on all processes, they are stored in a buffer upon arrival. The internal messages of the relation are only TO-delivered by the RECEIVER task when certain conditions hold. That is, the next relation of messages to be TO-delivered must contain messages for which each message sequence number is next in sequence for each respective process. In fact, it is easy to see that any two relations must respectively contain a message from at least one common process – only one message of a given process is being under correlation at a time and every relation contains \((f + 1)\) messages. It is straightforward to use those sequence numbers to break ties (trivially enforcing FIFO order also). Given that at any point in time there is only one message per process under correlation, we cannot have two relations with messages from two processes with inverse respective sequence number orders. This argument can be extended to any number of transitively connected relations.

**Theorem 2** Alg. 2 implements Total Order Broadcast.

**Proof.** See Appendix A.2
5 Extensions to C-MDMcast

This section first discusses stronger agreement for conjunctions. Then we add disjunctions to the predicate grammar of Section 3.3.3 and discuss respective properties. Finally, we present ordering properties for relations.

5.1 Covering Agreement

We now introduce Covering Conjunction Agreement, a stronger property than Conjunction Agreement presented previously in Section 3.3.3. This new property captures the intuition that if a first subscription “covers” another one, i.e., is more generic, then the messages delivered in response to the former subscription should include those delivered for the latter one.

Formalizing such a property is not trivial because one would also want to retain agreement on (sub-)relations, i.e., that messages delivered together as part of the more specific subscription are delivered together as well for the more generic one. This leads to fundamental limitations. Covering Conjunction Agreement only holds for conjunctions which are respectively “extended to the right” with respect to the subscription order \( \prec \), and the condition on disjointness of the sets of types, e.g., between \( \Phi \) and \( \Phi' \), makes the sub-conjunctions independent:

\[
\text{Covering Conjunction Agreement} \quad \exists \text{DLVR}_{\Phi, \Phi'}^i(m_1, ..., m_n, ...) \mid ((\Gamma(\Phi) = [T_1, ..., T_n]) \cap \Gamma(\Phi')) = \emptyset \Rightarrow \\
\forall p_j \in \text{correct}(F) \setminus \{p_i\} \mid \Phi \in \Psi(p_j) : \exists \text{DLVR}_{\Phi}^i(m_1, ..., m_n)
\]

Covering Conjunction Agreement is not defined as a symmetric implication (with \( \Phi(p_j) = \Phi \land \Phi' \)). The presence of a matching set of messages for a sub-relation given by \( \Phi' \) namely does not imply a timely or even eventual occurrence of a matching set for \( \Phi' \) conjoined by \( p_j \) with \( \Phi \), not even by Conjunction Validity. Covering Conjunction Agreement becomes trivially symmetric if \( \Phi' = \top \) (thus subsuming Conjunction Agreement).

Also note that not only must the types of the conjunction \( \Phi \) be equal, but the predicates must also be equivalent, i.e., no process may extend \( \Phi \) with another predicate of the same respective types. Consider that process \( p_j \) has defined a predicate \( \Phi_j = T_1 \land T_2 \) which could simply mean to deliver the first found instance of a message of type \( T_1 \) with the first instance of a message of type \( T_2 \). Second, a process \( p_i \) has defined a predicate \( \Phi_i = T_1 \land T_2, a_1 < 3 \). Now suppose, as shown in Fig. 3, that a sequence of messages of types \( T_1 \) and \( T_2 \) arrive in the following order: \( m_1 \cdot m_2 \cdot m_1' \cdot m_2' \). It is clear that \( \Phi_j[m_1, m_2] \) holds, but assume that \( m_1^2 a_1 = 4 \) \((> 3)\) and \( m_2^2 a_1 = 2 \) \((< 3)\). Process \( p_j \) would then deliver \([m_1, m_1^2] \) followed by \([m_2, m_2^2] \) but process \( p_i \) would deliver \([m_1, m_2^2] \). Since \( m_2^2 \) is matched with different messages in both cases, neither the agreement property of Section 3.3.3 nor Covering Conjunction Agreement is met.

![Diagram illustrating order of reception of messages](image)

Figure 3: Graph illustrating the order of reception of messages (e.g., \( m_1 \)) vs. when they are delivered as part of a relation (e.g., \([m_1, m_2^2] \)).

Thus, by example, if process \( p_j \) defines a conjunction \( \Phi_j = T_1 a_1 = v \) and a second process \( p_i \) wishes to extend the conjunction \( \Phi_j \) with another predicate, it could be such that \( \Phi_i = \Phi_j \land T_2 a_2 = v' \) but not be of the form (a) \( \Phi_i = T_1 a_1 = v' \land \Phi_j \), (b) \( \Phi_i = \Phi_j \land T_2 a_2 = T_1 a_1 \), or (c) \( \Phi_i = T_1 a_1 \leq v \). (a) is impossible since matching on several message types at any given process must proceed in a deterministic order, and any choice for a given type will affect all the choices for subsequent types. (b) and (c) would require all processes to know of the subscriptions of all other processes (and many messages to be discarded), which we deem overconstraining.

**Theorem 3** FRIP ensures Covering Conjunction Agreement.

**Proof.** See Appendix A.3
5.2 Disjunction Multi-delivery Multicast (D-MDMcast)

We now extend C-MDMcast to support disjunctions, leading to specifying the problem of Disjunction Multi-delivery Multicast (D-MDMcast) defined over MCAST and DLVR.

5.2.1 Predicate Grammar, Types, and Evaluation

Our grammar of Section 3.1 supports subscriptions consisting in disjunctions. Similarly, we defined the predicate types of disjunctions as well as the evaluation semantics.

For simplicity, but without loss of generality, we rule out the case of a disjunction that contains several identical conjunctions, i.e., $\forall \Psi = \Phi_1 \lor ... \lor \Phi_n$, $l, k \in [1..n]$: $\Phi_k = \Phi_l \Rightarrow k = l$. In practice, we can remove all but one copy.

DLVR is still parameterized by a conjunction $\Phi_k$ for a given invocation, which can be, however, any $\Phi_k \in \Psi(p_j)$ for a given process $p_j$’s subscription $\Psi(p_j)$.

It is important to emphasize at this point that $\lor$ is not interpreted as an eXclusive OR. Our validity and agreement properties introduced in Section 5.3 as well as the stronger property introduced in Section 5.1 thus remain valid for disjunctions since conjunctions within a disjunction are handled independently with respect to the messages delivered.

5.2.2 Disjunction-FRIP Algorithm

We now present an algorithm D-FRIP (Disjunction-FRIP) to implement the properties provided by D-MDMcast using TOBcast. This algorithm, depicted in Alg. 3, reuses the auxiliary functions ENQUEUE, DEQUEUE, and MATCH from FRIP in Alg. 1. In D-FRIP, however, every process maintains one message queue per message type per conjunction. For example, for a disjunction $\Psi = \Phi_1 \lor \Phi_2$ where $T(\Phi_1) = T(\Phi_2) = [T_1, T_2]$, $\Phi_1 = \rho_1 \land \rho_2$ where $\rho_1 = T_1.a_1 < T_2.a_2$ and $\rho_2 = T_1.a_1 < 20$ and $\Phi_2 = \rho_3 \land \rho_4$ where $\rho_3 = T_1.a_1 > T_2.a_2$ and $\rho_4 = T_2.a_1 < 20$, a process maintains two queues for type $T_1$ and $T_2$, one each for $\Phi_1$ ($Q_1[T_1]$) and $Q_1[T_2]$) and $\Phi_2$ ($Q_2[T_1]$ and $Q_2[T_2]$).

The primary change with respect to FRIP consists in a new response to a TO-delivery. The new primitive dispatches a given message to conjunctions in a deterministic order. In contrast to C-MDMcast, a same message can now lead to multiple MATCHES and DLVRs.

Theorem 4 D-FRIP implements D-MDMcast.

Proof. See Appendix A.4

Executed by every process $p_i$. Reuses ENQUEUE, DEQUEUE, and MATCH from FRIP.

<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1:</td>
<td>init</td>
</tr>
<tr>
<td>2:</td>
<td>$\Psi \leftarrow \Phi_1 \lor ... \lor \Phi_o$</td>
</tr>
<tr>
<td>3:</td>
<td>$\Phi_i \leftarrow \rho_1 \land ... \land \rho_o$</td>
</tr>
<tr>
<td>4:</td>
<td>$Q_i[T] \leftarrow \emptyset$ {MQs parameterized by type $T$ for $\Phi_i$}</td>
</tr>
</tbody>
</table>
| 5: | To MCAST($m$):
| 6: | TO-BCAST($m$) |
| 7: | upon TO-DLVR($m$) do |
| 8: | for all $\Phi_i \in $ in order do |
| 9: | if $T(m) \in T(\Phi_i)$ then |
| 10: | if ENQUEUE($m$, $\Phi_i$, $Q_i$) then |
| 11: | $[m_1, ..., m_k] \leftarrow$ MATCH($\emptyset$, $\Phi_i$, $Q_i$) |
| 12: | if $k \neq 0$ then \{Not an empty set\} |
| 13: | DEQUEUE($[m_1, ..., m_k]$, $Q_i$) |
| 14: | DLVR$_i$($[m_1, ..., m_k]$) |

Alg. 3: D-FRIP algorithm implementing D-MDMcast using TOBcast.

5.3 Total Order

Section 4 showed that total order is required when propagating single messages to achieve some form of agreement on relations in C-MDMcast. The same mechanisms for achieving such ordering might, however, help provide total order properties for relations. We define three types of total order properties for MDMcast below:

A trivial optimization of the algorithm for the case where $p$ has several conjunctions $\Phi_1, ..., \Phi_l$ with overlapping message types would consist in sharing queues across conjunctions. A message is tagged by an index $k$ of $\Phi_k$ to indicate that the message has previously been used in a match and delivered for conjunction $k$. Earlier messages of that type should then also be tagged with $k$. Messages with tags $\{1, ..., l\}$ may then be discarded.
deliveries. Alas, total order on individual messages is a prerequisite for agreement on delivery of relations; this order is
of conjunctions. (DNF), corresponding to most work on publish/subscribe systems and derived models, disjunctions are handled on top present paper follows the model of subscriptions proposed. Considering subscriptions to be in disjunctive normal form (sub-)conjunctions are delivered in a total order. An implementation which never delivers no two same relations on two processes with identical (sub-)conjunctions, could still ensure MESSAGE TOTAL ORDER. Perhaps more obvious is that, inversely, MESSAGE TOTAL ORDER does not imply CONJUNCTION TOTAL ORDER. DISJUNCTION TOTAL ORDER further sets our model apart from many single-message delivery multicast settings (e.g., traditional publish/subscribe [7]), where subscriptions are conjunctions, and disjunctions are handled independently through multiple conjunctions. Our property strives for total order across relations delivered to distinct conjunctions in a disjunction. Appendix A.6 elaborates on the difficulty of defining further ordering properties, in particular a property combining both CONJUNCTION - and DISJUNCTION TOTAL ORDER.

Theorem 5 FRIP ensures MESSAGE TOTAL ORDER.
Proof. See Appendix A.5

Theorem 6 FRIP ensures CONJUNCTION TOTAL ORDER.
Proof. See Appendix A.5

Theorem 7 D-FRIP ensures MESSAGE TOTAL ORDER.
Proof. See Appendix A.5

Theorem 8 D-FRIP ensures DISJUNCTION TOTAL ORDER.
Proof. See Appendix A.5

6 Conclusions
In their seminal paper on broadcasts, Hadzilacos and Toueg [18] propose a modular architecture for the three main ordering properties FIFO, causal and total order. Causal order is layered on top of FIFO order, and total order is a complementary property (though its implementation can be used towards causal order). The decomposition in the present paper follows the model of subscriptions proposed. Considering subscriptions to be in disjunctive normal form (DNF), corresponding to most work on publish/subscribe systems and derived models, disjunctions are handled on top of conjunctions.

As we prove in this paper, ordering and agreement are intertwined in aggregated deliveries unlike in single message deliveries. Alas, total order on individual messages is a prerequisite for agreement on delivery of relations; this order can, however, be exploited to order relations. While specific correlation and stream processing models have more expressive subscription grammars through operations which could give rise to more specific guarantees, there is little agreement on the fundamental operations of such models, and our feasibility results are of a general nature which apply to more specific models. Inversely, we have prioritized uniform properties in this paper for simplicity. Replacing certain properties by non-uniform ones for TO-Broadcast may lead to losing uniformity of certain properties for C-MDMcast based on FRIP, replacing other properties may hamper more than just uniformity.

In practice, using a Consensus-based TO-Broadcast to implement correlation yields high availability yet is very expensive; inversely, a pragmatic sequencer-based approach exposes a single point of failure and a performance bottleneck. We have thus devised and implemented a pragmatic scalable algorithm which uses a distributed hash-table to determine merger processes which handle specific conjunctions or disjunctions among given message types. These merger processes are replicated at a small scale to achieve some degree of fault tolerance which is weaker but far less expensive than that achieved by solving TO-Broadcast in a peer-based manner.

We are currently investigating further guarantees, such as aggregation-specific FIFO and causal ordering guarantees for messages and relations.
References


Appendix A

This appendix presents proof sketches for the lemmas and theorems presented in Sections 4 – 5.

A.1 Correctness of FRIP

Lemma 1 FRIP ensures MDM-No Duplication.

Proof sketch. TOB-No Duplication ensures that a message cannot be TO-delivered and thus enqueued more than once. If the message results in a successful match, the corresponding message is removed from the queue in the procedure DEQUEUE (Lines 7 - 9 in Alg. 1) and, therefore, will not be delivered more than once. Line 28 further ensures that for each matching instance of a same type, after the instance l, each subsequent instance message is also delivered and dequeued only once.

Lemma 2 FRIP ensures MDM-No Creation.

Proof sketch. TOB-No Creation ensures that a message will only be TO-delivered if TO-broadcast. A message is only TO-broadcast if multicast by Lines 5 - 6. A message may therefore only be delivered if it has been TO-delivered.

Lemma 3 FRIP ensures Admission.

Proof sketch. The function ENQUEUE (Lines 10 - 17) filters out all messages which do not satisfy the unary predicates in the subscription \( \Phi \). MATCH (Lines 25 - 34) iterates through the queues to find the first instance of the first type in \( \Phi \) for which there are messages of the remaining types (or further messages of the same type in such cases) with which all predicates in \( \Phi \) are satisfied. Hence, any relation \([m_1, \ldots, m_n]\) that is delivered matches the subscription \( \Phi \).

Lemma 4 FRIP ensures Message Validity.

Proof sketch. For a given type \( T \) of a message \( m \) which is not dependent on any other type in a conjunction through a binary predicate, given an infinite number of messages of each of the conjoined types, if \( m \) is TO-broadcast, it will eventually be TO-delivered by all correct processes. Further, \( m \) will not be DEQUEUED by some later message being matched prior since MATCH (Lines 25 - 34) looks for the first found instance of a type which satisfies the conjunction. \( m \), as part of only a unary predicate, will always be a first found instance; even when multiple messages of the same type such as \( m \) belong to a predicate, each message will be matched according to the order in the queue and none will be DEQUEUED due to some later message being matched.

Lemma 5 FRIP ensures Conjunction Validity.

Proof sketch. If for any process’s conjunction, infinitely many matching messages are multicast, then Conjunction Validity is ensured. Every multicast namely leads to a TO-broadcast, and since DEQUEUE is only called after a match, it cannot keep an infinite subset of matching TO-broadcast messages from being correlated and matched. Every time messages are discarded from the buffer, including those not delivered, there will still be an infinite number of matching messages TO-broadcast in the future.

Lemma 6 FRIP ensures Conjunction Agreement.

Proof sketch. The underlying Total Order Broadcast guarantees that no two (correct or faulty) processes TO-deliver the same two messages in different orders through TOB-Agreement and TOB-Total Order. Hence, no two processes with the same subscription \( \Phi \) have message queue contents which diverge in time with respect to their (identical) streams of TO-delivered messages. The deterministic matching of messages performed in the MATCH function (Lines 25 - 34) ensures that the same relations are delivered at all processes with subscription \( \Phi \).

Theorem 1 FRIP implements C-MDMcast.

Proof. By Lemmas 1 - 6.
A.2 Correctness of Algorithm $T_C\hspace{.1cm}MDMcast\hspace{.1cm}TOBcast$

Lemma 7 Alg. 2 ensures $\text{TOB-NO DUPLICATION}$.

**Proof sketch.** MDM-NO DUPLICATION ensures that no message can be delivered more than once. Each message multicast is added to $tbdelivered$ at most once. Thus, each message is TO-delivered at most once since once a message is TO-delivered, the relation containing that message is removed from $tbdelivered$.

Lemma 8 Alg. 2 ensures $\text{TOB-NO CREATION}$.

**Proof sketch.** MDM-NO CREATION ensures a message is delivered only if multicast. Each message is only multicast once it is placed in $broadcasts$ by Lines 18 and 20 and correspondingly placed in $tbdelivered$ if delivered within a relation. Only messages (except $⊥$ messages) in $tbdelivered$ are TO-delivered.

Lemma 9 Alg. 2 ensures $\text{TOB-VALIDITY}$.

**Proof sketch.** This proof will proceed in two steps. First, it will be shown that relations will be delivered by correct processes in a lock-step manner, which assures that messages of some form are delivered. Then, it will be shown that a relation containing message $m$ which a process $p_i$ has multicast will eventually be delivered by $p_i$ and thus TO-delivered.

Each process will multicast application (TO-broadcast) messages (Line 11 of Alg. 2) when present, or $⊥$ messages (Line 16 of Alg. 2) when there are no application messages to send. Because there is a majority $(f + 1)$ of correct processes, there will always be at least $(f + 1)$ messages which may be correlated at any given time. Since each process only multicasts one message at a time, each message a process receives of its own that is delivered in a relation will be a message in sequence; and that process may therefore multicast another message. If a process delivers a relation containing any number of in-sequence messages, there may be other messages in the same relation that are out-of-sequence from the respective processes. However, a process will not TO-deliver any messages in a relation unless all the messages are next in respective sequence by Line 26 of Alg. 2. Between any two relations, there will always be at least one message of a same process $p_k$ in each of the relations. There is, therefore, transitively an order that may be determined by the relation that $p_k$ delivered first. Therefore, there are further relations that contain the messages which precede each of the out-of-sequence messages that the corresponding processes have already delivered. By CONJUNCTION AGREEMENT, those relations will eventually be delivered and all the internal messages will therefore be TO-delivered.

When a correct process $p_i$ multicasts a message $m$, $m$ may only be delivered when it is correlated with $f$ additional messages. By MESSAGE VALIDITY, since subscriptions $\Phi$ have no unary (or binary) predicates on any of the messages, $m$ will eventually be delivered. When the relation containing $m$ is delivered, there may be other messages in that relation that are out of sequence from the respective processes. Since the relations containing those messages are guaranteed to be delivered and thus all the preceding in-sequence messages TO-delivered (as shown above), $m$ will thus be TO-delivered ensuring TOB-VALIDITY.

Lemma 10 Alg. 2 ensures $\text{TOB-AGREEMENT}$.

**Proof sketch.** All processes have the same subscription $\Phi$. By TOB-VALIDITY, if one process $p_i$ multicasts a message $m$, $m$ will be correlated with $f$ other messages, matching $\Phi$, and thus be TO-delivered by $p_i$. By CONJUNCTION AGREEMENT, all processes will deliver the relation containing $m$ and place that relation in $tbdelivered$. As was first shown for TOB-VALIDITY, if for some process, there are messages out-of-sequence in the same relation as $m$, the in-sequence messages will eventually be TO-delivered so that $m$ and all the messages in the same relation may also be TO-delivered by that process. By CONJUNCTION AGREEMENT, all processes will eventually deliver all the same relations. Through the deterministic order (as shown in Lemma 9) in which relations are delivered, all messages in the respective delivered relations will be eventually be TO-delivered, thus, Alg. 2 ensures TOB-AGREEMENT.

Lemma 11 Alg. 2 ensures $\text{TOB-TOTAL ORDER}$.
Proof sketch. Correct processes deliver the same relations (by Lemma 10), and these can be ordered deterministically (as argued in Lemma 9). By TO-delivering the messages within these relations deterministically (Lines 26 - 31 of Alg. 2) Alg. 2 ensures TOB-TOTAL ORDER.

Theorem 2 Alg. 2 implements Total Order Broadcast.

Proof. By Lemmas 7 – 11.

A.3 Covering Agreement

Theorem 3 FRIP ensures COVERING CONJUNCTION AGREEMENT.

Proof sketch. COVERING CONJUNCTION AGREEMENT is provided as messages of individual types are handled independently by the matching in Alg. 1. If two processes $p_i$ and $p_j$ define conjunctions $\Phi \land \Phi_i$ and $\Phi$ respectively, as long as $\Phi_i$ is disjoint with $\Phi$ (thus messages that match with $\Phi$ are independent of any messages that match with $\Phi_i$), then if a match is found for $p_i$, there is a subset $s$ of the relation for which $\Phi$ is true.

Because of TOB-AGREEMENT and TOB-TOTAL ORDER, no two processes enqueue the same two messages in different orders. Thus, for every type in $\Phi$, both $p_i$ and $p_j$ will have queue contents which remain identical since any messages received by $p_i$ of any type in $\Phi_i$ will be placed in different queues. Thus, if $p_i$ delivers a relation, one of two possibilities occur: either the last message received that triggered the match on $p_i$ is in $s$, thus of a type in $\Phi$, or the last message received is not in $s$, thus of a type in $\Phi_i$.

If the last message received by $p_i$ is in $s$, then because of TOB-AGREEMENT and TOB-TOTAL ORDER, $p_i$ and $p_j$’s queues over the set of types for $\Phi$ were identical before that message was received by either $p_i$ or $p_j$. Further, by TOB-AGREEMENT, if a message is received by $p_i$, $p_j$ will also receive that message making the queues identical again. Because of the deterministic matching on Lines 25-34 of Alg. 1, $p_j$ will also deliver $s$.

Conversely, if the last message received by $p_i$ is not in $s$, then there are messages already in the queues for the types of $\Phi$ which match for $s$. Thus, by TOB-AGREEMENT and TOB-TOTAL ORDER, $p_j$ will have already received the messages in $s$ which would have triggered a match on $p_j$. The messages which match for $\Phi_i$ do not affect the order of matching or cause any of the messages in $s$ to be dequeued on $p_i$ when delivering the messages corresponding to $\Phi \land \Phi_i$. Thus, COVERING CONJUNCTION AGREEMENT is ensured.

A.4 Correctness of D-FRIP

Lemma 12 D-FRIP ensures MDM-NODUPLICATION.

Proof sketch. From Lemma 1 no message will be enqueued, delivered and/or dequeued more than once for any conjunction. D-FRIP holds a separate queue per conjunction. The primitives ENQUEUE and DLVR are each called at most once per message, per conjunction in Lines 8-14 in Alg. 3 and DEQUEUE is called per conjunction only after a match. Therefore, D-FRIP ensures MDM-NODUPLICATION for each conjunction.

Lemma 13 D-FRIP ensures MDM-NO CREATION.

Proof sketch. No message is TO-broadcast and hence TO-delivered unless multicast, and Alg. 3 only delivers TO-delivered messages.

Lemma 14 D-FRIP ensures ADMISSION.

Proof sketch. By Lemma 3 FRIP ensures ADMISSION for any one conjunction. Since ENQUEUE (Line 10) and MATCH (Line 11) are called per conjunction upon the reception of a message in Alg. 3 only valid relations that match at least one conjunction in a subscription are delivered in D-FRIP.

Lemma 15 D-FRIP ensures MESSAGE VALIDITY
Proof sketch. By Lemma 4, FRIP ensures MESSAGE VALIDITY for any one conjunction. Since individual messages are matched in a first received order for a conjunction, and since D-FRIP calls the match in Line 11 of Alg. 3 for each conjunction in a subscription, D-FRIP ensures MESSAGE VALIDITY.

Lemma 16 D-FRIP ensures CONJUNCTION VALIDITY.

Proof sketch. By Lemma 5, as long as infinitely many matching messages are multicast, FRIP ensures CONJUNCTION VALIDITY for any single conjunction. Since a match for each conjunction within a subscription is independently triggered upon receiving matching messages in Line 11 of Alg. 3, D-FRIP, therefore, ensures CONJUNCTION VALIDITY.

Lemma 17 D-FRIP ensures COVERING AGREEMENT.

Proof sketch. By Theorem 3, FRIP ensures COVERING CONJUNCTION AGREEMENT. If two processes $p_i$ and $p_j$ define conjunctions $\Phi \land \Phi_i$ and $\Phi$ respectively, then since MATCH is reused from Alg. 1 and is called deterministically in Alg. 3, the conjunctions will be evaluated independently and thus deliver matching messages when they are received. Since these conjunctions are matched independently of one another, and since FRIP ensures COVERING CONJUNCTION AGREEMENT per conjunction, D-FRIP ensures COVERING CONJUNCTION AGREEMENT.

Theorem 4 D-FRIP implements D-MDMcast.

Proof. By Lemmas 12 – 17.

A.5 Proofs for Ordering Guarantees

Theorem 5 FRIP ensures MESSAGE TOTAL ORDER.

Proof sketch. MESSAGE TOTAL ORDER is ensured in that TO-bcast determines a total order for the messages of any specific type, and that first received matching and infix+prefix disposal retain this order.

Theorem 6 FRIP ensures CONJUNCTION TOTAL ORDER.

Proof sketch. CONJUNCTION TOTAL ORDER is ensured because the matching deterministically proceeds along types in order of their occurrence in conjunctions, and by respecting orders for individual message types, thus the order for identical independent (sub-)relations.

Theorem 7 D-FRIP ensures MESSAGE TOTAL ORDER.

Proof sketch. The proof for MESSAGE TOTAL ORDER follows that for FRIP since the queueing and matching (used from FRIP) are performed deterministically after receiving each message.

Theorem 8 D-FRIP ensures DISJUNCTION TOTAL ORDER.

Proof sketch. DISJUNCTION TOTAL ORDER is ensured when two processes, $p_i$ and $p_j$, define two separate but equivalent conjunctions. D-FRIP ensures that after receiving each message, both processes deterministically perform matching on those respective conjunctions in the same order (left to right). Since it has been shown that FRIP ensures CONJUNCTION TOTAL ORDER, and D-FRIP reuses MATCH (Line 11) from FRIP per conjunction, D-FRIP, therefore, ensures DISJUNCTION TOTAL ORDER.
A.6 Junction Total Order

One might imagine extending CONJUNCTION - and DISJUNCTION TOTAL ORDER to a similar property as below:

\[
\text{Junction Total Order} \quad \exists \text{DLVR}^i_{\Phi_1 \land \Phi'_1}([m_1, ..., m_n])_{t_i}, \text{DLVR}^i_{\Phi_2 \land \Phi'_2}([m'_1, ..., m'_{n'}])_{t'_i}, \text{DLVR}^j_{\Phi_1}([m_1, ..., m_n])_{t_j}, \text{DLVR}^j_{\Phi_2}([m'_1, ..., m'_{m_n}])_{t'_j}
\]

\[
\text{DLVR}^i_{\Phi_1}([m_1, ..., m_m])_{t_i} \mid ((T(\Phi_1) = [T_1, ..., T_n]) \cap T(\Phi'_1)) = \emptyset \land ((T(\Phi_2) = [T'_1, ..., T'_m]) \cap T(\Phi'_2)) = \emptyset \Rightarrow (t_i < t'_i \iff t_j < t'_j)
\]

However, due to the (left-to-right) deterministic order in which disjunctions are evaluated, \(p_i\) and \(p_j\) could deliver commonly received messages in different orders. If a message \(m_2\) of type \(T_2\) is received by both processes, followed by a message \(m_1\) of type \(T_1\), where \(\Phi_1 = T_1 \lor T_2\), then \(p_j\) delivers \(m_2\) before \(m_1\). However, if message(s) are received by \(p_i\) that trigger \(\Phi'_1\) before any that satisfy \(\Phi'_2\), then \(p_i\) will deliver \(m_1\) before \(m_2\). Even in a more constraining case, when \(\Phi'_1 = \Phi'_2\) as in Fig. 4 where \(\Phi'_1 = \Phi'_2 = \Phi = T_3\), when the message arrives that triggers \(\Phi\) after receiving messages for \(\Phi_2\) followed by \(\Phi_1\), the matching for \(p_i\) is performed left to right and thus, \(p_i\) delivers \(m_1\) before \(m_2\). Defining the combination of total orders on conjunctions and disjunctions is different from simply ensuring both properties. It is difficult since processes can each have conjunctions which extend conjunctions of the other.

**Figure 4:** Example showing difficulty/issue of defining generalized JUNCTION TOTAL ORDER: \(\Phi_1 = T_1, \Phi_2 = T_2, \Phi'_1 = \Phi'_2 = \Phi = T_3\).