Li *et al.*'s algorithm

- Li *et al.*'s algorithm for recording a global snapshot in a non-FIFO system is similar to the Lai–Yang algorithm.
- multiple initiators
- The algorithm is not concerned with the contents of computation messages and the state of a channel is computed as the number of messages in transit in the channel.

- A process maintains two counters for each incident channel to record the number of messages sent and received on the channel and reports these counter values with its snapshot to the initiator.
- The initiator computes the state of Cij as: (the number of messages in Cij in the previous snapshot) + (the number of messages sent on Cij since the last snapshot at process pi) − (the number of messages received on Cij since the last snapshot at process pj).

- Snapshots initiated by an initiator are assigned a sequence number. All messages sent after a local snapshot recording are tagged by a tuple < init_id,MKNO >, where init_id is the initiator's identifier and MKNO is the sequence number of the algorithm's most recent invocation by initiator init_id; to insure liveness.
- The tuple <init_id,MKNO> is a generalization of the red/white colors used in Lai–Yang to accommodate repeated invocations of the algorithm and multiple initiators.

- The local state recording is done as described by rules 1–3 of the Lai–Yang algorithm.
- A process maintains input/output counters for the number of messages sent and received on each incident channel after the last snapshot.
- The algorithm is not concerned with the contents of computation messages and so the computation of the state of a channel is simplified to computing the number of messages in transit in the channel.

The channel states are recorded as described in rules 4–6 of the Lai–Yang algorithm:

4. Every white process records a history, as input and output counters, of all white messages sent or received by it along each channel after the previous snapshot (by the same initiator).
5. When a process turns red, it sends these histories (i.e., input and output counters) along with its snapshot to the initiator process that collects the global snapshot.
6. The initiator process computes the state of channel C_{ij} as follows:

$$\begin{aligned} SC_{ij} = {} & transit(LS_i, LS_j) = TRANSIT_{ij} \\ & + (\#\text{messages sent on that channel since the last snapshot}) \\ & - (\#\text{messages received on that channel since the last snapshot}). \end{aligned}$$

- If the initiator initiates a snapshot before the completion of the previous snapshot, it is possible that some process may get a message with a lower sequence number after participating in a snapshot initiated later. In this case, the algorithm uses the snapshot with the higher sequence number to also create the snapshot for the lower sequence number.
- The algorithm works for multiple initiators if separate input/output counters are associated with each initiator, and marker messages and the tag fields carry a vector of tuples, with one tuple for each initiator.

Mattern's Algorithm

Mattern's algorithm assumes a single initiator process and works as follows:

1. The initiator "ticks" its local clock and selects a future vector time s at which it would like a global snapshot to be recorded. It then broadcasts this time s and freezes all activity until it receives all acknowledgements of the receipt of this broadcast.
2. When a process receives the broadcast, it remembers the value s and returns an acknowledgement to the initiator.
3. After having received an acknowledgement from every process, the initiator increases its vector clock to s and broadcasts a dummy message to all processes. (Observe that before broadcasting this dummy message, the local clocks of other processes have a value $\not\geq s$.)
4. The receipt of this dummy message forces each recipient to increase its clock to a value $\geq s$ if not already $\geq s$.
5. Each process takes a local snapshot and sends it to the initiator when (just before) its clock increases from a value less than s to a value $\geq s$. Observe that this may happen before the dummy message arrives at the process.
6. The state of C_{ij} is all messages sent along C_{ij}, whose timestamp is smaller than s and which are received by p_j after recording LS_j.

One of the following schemes for Termination Detection

1. Each process i keeps a counter $cntr_i$ that indicates the difference between the number of white messages it has sent and received before recording its snapshot. It reports this value to the initiator process along with its snapshot and forwards all white messages, it receives henceforth, to the initiator. Snapshot collection terminates when the initiator has received $\sum_i cntr_i$ number of forwarded white messages.
2. Each red message sent by a process carries a piggybacked value of the number of white messages sent on that channel before the local state recording. Each process keeps a counter for the number of white messages received on each channel. A process can detect termination of recording the states of incoming channels when it receives as many white messages on each channel as the value piggybacked on red messages received on that channel.