**Periodic synchronization**. To synchronization state, a user would follow the steps illustrated in Figure 4. A critical first step in periodic synchronization is to determine a global synchronization point *without inter-user communication*. We exploit the following property of vector clocks:

* The state of different users would be the same if they have the same vector clock value.

sync

Figure 4. Main steps of periodic synchronization.

We set a user-configurable parameter, *n*, as the interval for synchronization. The rules to determine the synchronization point at each user is given below:

* When a user receives an operation from the session server, if the sum of the local and remote timestamps is equal to *n* or multiple of *n*, it would initiate a new round of synchronization *after* applying the operation assuming that the user has not submitted local operations concurrently. Otherwise, the concurrent local operations are first undone and then the remote operation is applied to ensure a consistent state with the remote user.
* The user that submitted the operation which triggers the round of synchronization at other users would not know it is time to start the round of synchronization because it cannot use the sum of its own vector clock value. Doing so would risk inconsistent state due to concurrent operations at other participants. In this case, the session server must inform the user that it is time to start a new round of synchronization. To prevent a faulty publisher (which co-locates with the session server) from indefinitely delaying the synchronization for this user, a participant starts a timer whenever submitting a candidate operation. If the publisher is not faulty, the user should either receive a synchronization notification, or a remote operation before the timer expires. If it does not, it suspects the publisher.

When a user initiates a round of synchronization, it takes a checkpoint of the current version of the shared document, applies a secure hash on it (such as SHA-2), and multicasts the hash value together with the vector clock value (referred to as a sync message) at the synchronization point to all other users.

When a user collects 2f sync messages from other users that are consistent with its own for the same round of synchronization, it is happy with the current publisher. The user then labels its latest checkpoint as stable. It may proceed to removing all logged operations and the previous stable checkpoint if the garbage collection policy allows.

If a user could collect 2f+1 consistent sync messages from other users, but they are *not* consistent with its own for the same round of synchronization, it suspects the primary, but proceeds to requesting a state transfer from other participants. When it receives the version of the shared document, it replaces its own using the received one and moves forward. The reason why it has to move on is that 2f+1 other participants are happy with the current publisher and hence it does not have enough votes to overturn the current publisher.

**Checkpointing and logging**. For each round of synchronization, a stable checkpoint of the shared document will be produced at each user. All incoming and outgoing update operations are logged.

If the editing session is relatively short, all checkpoints and logged operations can be preserved. They would be useful for recovery when threats such as PA1 or PU1 (malicious updates) are detected. Quite often, there may be a significant delay between when a user starts to maliciously update the document and when the fault is detected. The logs can be useful for forensic analysis to determine the starting point of the malicious updates. Once the starting point is detected, the document is reversed back to the most recent stable checkpoint of the document prior to the starting point, and the logged operations *excluding those faulty updates* since that stable checkpoint are used to undo the damages done by the malicious user.