IceStorm Concepts

This section discusses several concepts that are important for understanding IceStorm's capabilities.

- **Message**
  An IceStorm *message* is strongly typed and is represented by an invocation of a Slice operation: the operation name identifies the type of the message, and the operation parameters define the message contents. A message is published by invoking the operation on an IceStorm proxy in the normal fashion. Similarly, subscribers receive the message as a regular servant upcall. As a result, IceStorm uses the "push" model for message delivery; polling is not supported.

- **IceStorm Topics**
  An application indicates its interest in receiving messages by subscribing to a *topic*. An IceStorm server supports any number of topics, which are created dynamically and distinguished by unique names. Each topic can have multiple publishers and subscribers.

  A topic is essentially equivalent to an application-defined Slice interface: the operations of the interface define the types of messages supported by the topic. A publisher uses a proxy for the topic interface to send its messages, and a subscriber implements the topic interface (or an interface derived from the topic interface) in order to receive the messages. This is no different than if the publisher and subscriber were communicating directly in the traditional client-server style; the interface represents the contract between the client (the publisher) and the server (the subscriber), except IceStorm transparently forwards each message to multiple recipients.

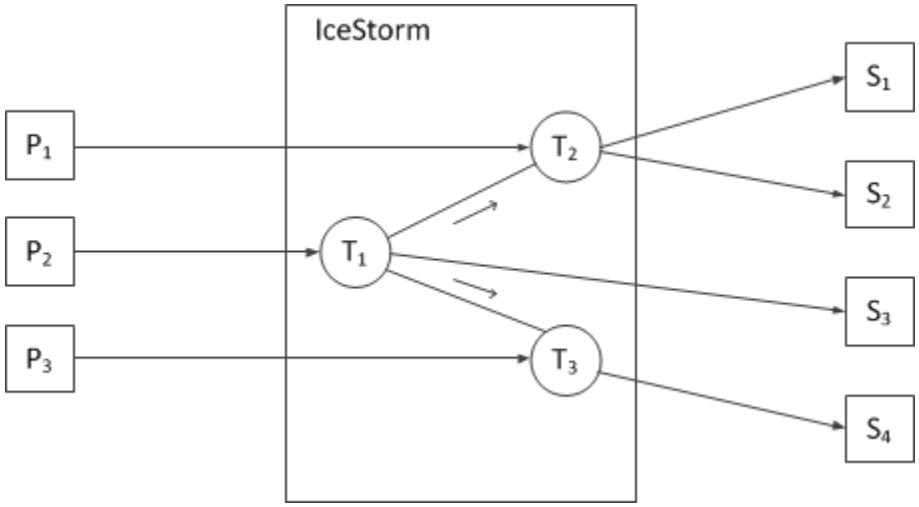
  IceStorm does not verify that publishers and subscribers are using compatible interfaces, therefore applications must ensure that topics are used correctly.

- **Unidirectional Messages**
  IceStorm messages are unidirectional, that is, they must have `void` return type, cannot have out-parameters, and cannot raise user exceptions. It follows that a publisher cannot receive replies from its subscribers. Any of the Ice transports (TCP, SSL, and UDP) can be used to publish and receive messages.

- **Federation**
  IceStorm supports the formation of topic graphs, also known as federation. A topic graph is formed by creating links between topics, where a *link* is a unidirectional association from one topic to another. Each link has a *cost* that may restrict message delivery on that link. A message published on a topic is also published on all of the topic's links for which the message cost does not exceed the link cost.

  Once a message has been published on a link, the receiving topic publishes the message to its subscribers, but does not publish it on any of its links. In other words, IceStorm messages propagate at most one hop from the originating topic in a federation.

  The following figure presents an example of topic federation. Topic T_1 has links to T_2 and T_3, as indicated by the arrows. The subscribers S_1 and S_2 receive all messages published on T_2, as well as those published on T_1. Subscriber S_3 receives messages only from T_1, and S_4 receives messages from both T_3 and T_1.



Topic federation.

  IceStorm makes no attempt to prevent a subscriber from receiving duplicate messages. For example, if a subscriber is subscribed to both T_2 and T_3, then it would receive two requests for each message published on T_1.

- **Quality of Service**
  IceStorm allows each subscriber to specify its own *quality of service* (QoS) parameters that affect the delivery of its messages. Quality of service parameters are represented as a dictionary of name-value pairs.

- **Replication**
  IceStorm supports replication to provide higher availability for publishers and subscribers.

- **Persistent Mode**
  IceStorm's default behavior maintains information about topics, links, and subscribers in a database. However, a message sent via IceStorm is not stored persistently, but rather is discarded as soon as it is delivered to the topic's current set of subscribers. If an error occurs during delivery to a subscriber, IceStorm does not queue messages for that subscriber.

  By default, IceStorm stores its persistent state in a Freeze database. However, you can also configure IceStorm to use a different database, such as PostgreSQL.

- **Transient Mode**
  IceStorm can optionally run in a fully transient mode in which no database is required. Replication is not supported in this mode.

- **Subscriber Errors**
  If IceStorm encounters a failure while attempting to deliver a message to a subscriber, the subscriber is immediately unsubscribed from the topic on which the message was published. This is important if you make changes to a Slice data type or operation signature: if you do, you must ensure that both publishers and subscribers use the same Slice definitions; if you do not, IceStorm is likely to encounter marshaling errors when forwarding an event to a subscriber with a mismatched Slice definition and remove the subscription.

See Also

- Oneway Invocations
- Topic Federation
- IceStorm Quality of Service
- Configuring IceStorm
- Freeze