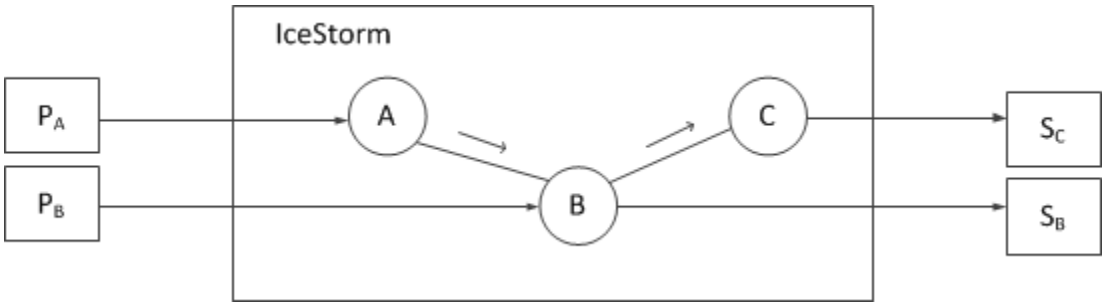
Topic Federation

The ability to link topics together into a federation provides IceStorm applications with a lot of flexibility, while the notion of a "cost" associated with links allows applications to restrict the flow of messages in creative ways. IceStorm applications have complete control of topic federation using the `To picManager` interface described in the online XREF Slice API Reference, allowing links to be created and removed dynamically as necessary. For many applications, however, the topic graph is static and therefore can be configured using the administrative tool .

On this page:

IceStorm Message Propagation

IceStorm messages are never propagated over more than one link. For example, consider the topic graph shown below:



Message propagation.

In this case, messages published on `A` are propagated to `B`, but `B` does not propagate `A`'s messages to `C`. Therefore, subscriber S_B receives messages published on topics `A` and `B`, but subscriber S_C only receives messages published on topics `B` and `C`. If the application needs messages to propagate from `A` to `C`, then a link must be established directly between `A` and `C`.

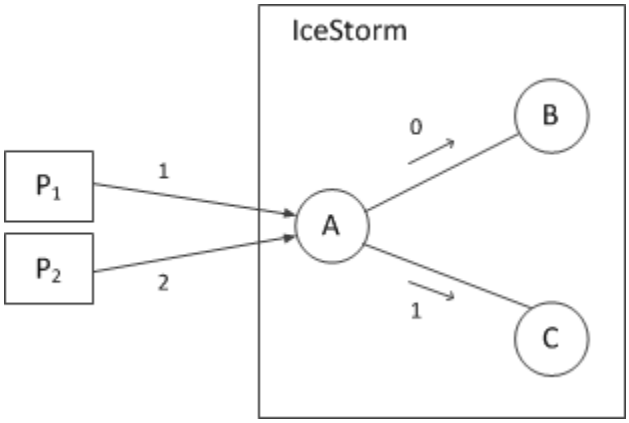
Using Cost to Limit Message Propagation

As described above, IceStorm messages are only propagated on the originating topic's immediate links. In addition, applications can use the notion of cost to further restrict message propagation.

A cost is associated with messages and links. When a message is published on a topic, the topic compares the cost associated with each of its links against the message cost, and only propagates the message on those links whose cost equals or exceeds the message cost. A cost value of zero (`0`) has the following implications:

- messages with a cost value of zero (`0`) are published on all of the topic's links regardless of the link cost;
- links with a cost value of zero (`0`) accept all messages regardless of the message cost.

For example, consider the following topic graph:



Cost semantics.

Publisher P_1 publishes a message on topic `A` with a cost of `1`. This message is propagated on the link to topic `B` because the link has a cost of `0` and therefore accepts all messages. The message is also propagated on the link to topic `C`, because the message cost does not exceed the link cost (`1`). On the other hand, the message published by P_2 with a cost of 2 is only propagated on the link to `B`.

Request Context for Cost

The cost of a message is specified in an Ice request context. Each Ice proxy operation has an implicit argument of type `Ice::Context` representing the request context. This argument is rarely used, but it is the ideal location for specifying the cost of an IceStorm message because an application only needs to supply a request context if it actually uses IceStorm's cost feature. If the request context does not contain a cost value, the message is assigned the default cost value of zero (0).

Publishing a Message with a Cost

The code examples below demonstrate how a collector can publish a measurement with a cost value of 5. First, the C++ version:

C++

```
Measurement m = getMeasurement();
Ice::Context ctx;
ctx["cost"] = "5";
monitor->report(m, ctx);
```

And here is the equivalent version in Java:

Java

```
Measurement m = getMeasurement();
java.util.HashMap ctx = new java.util.HashMap();
ctx.put("cost", "5");
monitor.report(m, ctx);
```

Receiving a Message with a Cost

A subscriber can discover the cost of a message by examining the request context supplied in the `Ice::Current` argument. For example, here is a C++ implementation of `Monitor::report` that displays the cost value if it is present:

C++

```
virtual void report(const Measurement& m, const Ice::Current& curr) {
    Ice::Context::const_iterator p = curr.ctx.find("cost");
    cout << "Measurement report:" << endl
         << "  Tower: " << m.tower << endl
         << "  W Spd: " << m.windSpeed << endl
         << "  W Dir: " << m.windDirection << endl
         << "   Temp: " << m.temperature << endl
         << "   Temp: " << m.temperature << endl;
    if (p != curr.ctx.end())
        cout << "   Cost: " << p->second << endl;
    cout << endl;
}
```

And here is the equivalent Java implementation:

Java

```
public void report(Measurement m, Ice.Current curr) {
    String cost = null;
    if (curr.ctx != null)
        cost = curr.ctx.get("cost");
    System.out.println(
        "Measurement report:\n" +
        "  Tower: " + m.tower + "\n" +
        "  W Spd: " + m.windSpeed + "\n" +
        "  W Dir: " + m.windDirection + "\n" +
        "   Temp: " + m.temperature);
    if (cost != null)
        System.out.println("   Cost: " + cost);
    System.out.println();
}
```

For the sake of efficiency, the Ice for Java run time may supply a null value for the request context in `Ice.Current`, therefore an application is required to check for null before using the request context.

Automating IceStorm Federation

Given the restrictions on message propagation described in the previous sections, creating a complex topic graph can be a tedious endeavor. Of course, creating a topic graph is not typically a common occurrence, since IceStorm keeps a persistent record of the graph. However, there are situations where an automated procedure for creating a topic graph can be valuable, such as during development when the graph might change significantly and often, or when graphs need to be recomputed based on changing costs.

Administration Tool Script

A simple way to automate the creation of a topic graph is to create a text file containing commands to be executed by the IceStorm administration tool. For example, the commands to create the topic graph shown earlier are shown below:

```
create A B C
link A B 0
link A C 1
```

If we store these commands in the file `graph.txt`, we can execute them using the following command:

```
$ icestormadmin --Ice.Config=config < graph.txt
```

We assume that the configuration file `config` contains the definition for the property `IceStormAdmin.TopicManager.Default`.

Proxy Considerations for IceStorm Federation

Note that, if you federate IceStorm servers, you must ensure that the proxies for the linked topics always use the same host and port (or, alternatively, can be indirectly bound via IceGrid), otherwise the federation cannot be re-established if one of the servers in the federation shuts down and is restarted later.

See Also

- IceStorm Administration
- Request Contexts
- The Current Object
- IceStorm Properties
- IceGrid