## HBSD External Router Classes

**StatisticsManager class:**

This class is an implementation of HBSD network statistics management algorithm described in our paper:

**Amir Krifa**, **Chadi Barakat**, **Thrasyvoulos Spyropoulos**, "**An Optimal Joint Scheduling and Drop Policy for Delay Tolerant Networks**", in proceedings of the WoWMoM Workshop on Autonomic and Opportunistic Communications, Newport Beach (CA), June 2008.

StatisticsManager class maintains network statistics and update it each time a new metadata bundle is received or whenever some local storage related events occur (a bundle is dropped due to congestion, a new bundle is added,…).

StatisticsManager also calculates and returns the utility of a given bundle (given its life time) with respect to the routing metric to optimize (either the network average delivery rate or delivery delay).

**PeerListener Class:**

The PeerListener object exists as a thread dedicated to processing HBSD router-specific bundles received by peer routers. DTN2 specifies that any EID with an endpoint that begins with ext.rtr (e.g., dtn://node.dtn/ext.rtr/HBSD) is to be destined for an external router, and it generates a specific XML event when it receives a bundle containing the ext.rtr endpoint. When the HBSD_Routing class receives the event, it dispatches the PeerListener objects thread to process the bundle. Specifically, the PeerListener thread extracts the data from the bundle, deletes the bundle, and then makes a call into the Policy Manager passing the bundles data. This is the recipient half of the mechanism for exchanging meta data between routers. To send meta data, the policy manager must inject a bundle into DTN2. This is done by the Policy Manager via a method in the Requester class.

**Requester Class:**

This is a utility class responsible for composing and sending all XML messages to the DTN2 daemon. These classes are used throughout HBSD.

To expand on injecting a bundle, since it is fundamental to HBSD exchanging meta data with a peer node: When the Requester injects a bundle for the router it assigns a unique request ID to the bundle. It is up to the policy manager to later associate the injected bundle with the request using the id. The Requester only plays a minor role in injecting bundles: it sends the request to DTN2 after the policy manager creates the data. But it should be noted that injected bundles are handled differently from other bundles by HBSD. HBSD creates a Bundle object for an injected bundle, but it does not retain knowledge of the bundle in the Bundles class.

**GBOF class:**

This is another utility class, and it contains static methods for manipulating the GBOF. This includes formatting the GBOF as XML as required by DTN2. It

also includes methods for creating a hash key from the values that make up the GBOF. This hash key is used extensively and consistently throughout HBSD and the Policy Manager for referencing a bundle.

**HBSD Class:**

The HBSD class contains the main body of the router. This class reads the command line arguments, loads the configuration file (if defined), starts logging, and sets up the SAX (Simple API for XML) handler. Once initialized, it joins the DTN2 multicast group and continuously loops receiving locally broadcast messages from the DTN2 daemon, dtnd. When XML messages are received from DTN2, the SAX handler is responsible for parsing the message and dispatching the appropriate method.

**HBSD_SAX class:**

This class is invoked when HBSD receives an XML message from the DTN2 daemon. It extends the C++ SAX DefaultHandler class. HBSD_SAX parses the XML message and calls the corresponding method in the Handlers class.

**Handlers class:**

This is an abstract class that defines a method for each XML event message that may be received by HBSD_SAX. The HBSD_Routing class is the real implementation of the Handlers class. We use an abstract class that supplies null methods for all XML messages. If the class that extends Handlers, i.e. HBSD_Routing, does not support an event then the empty method in Handlers in invoked.

**XMLTree class:**

This is a utility class. When HBSD_SAX parses an XML message each element is placed in an XMLTree object. XMLTree objects may be linked to each other to represent the hierarchy of elements in an XML message. XMLTree objects have methods for accessing attributes and child elements.

**HBSD_Routing class:**

The HBSD_Routing class is the heart of the router, extending the methods defined by Handlers. It is here that the XML messages sent by the DTN2 daemon, as represented by XMLTree objects, are initially acted upon.

**Logging class:**

This is an interface that defines the logging class used by the HBSD router. HBSD provides one implementation of the Logging class: Console_Logging. By default, Console_Logging is used though you can define which implementation to invoke via the HBSD configuration file.

**Console_Logging class:**

This is a simple implementation of the Logging interface that outputs logging messages to stdout. It is the default logging class

**ConfFile class:**

This is a utility class that reads and parses the HBSD configuration file.

**Bundles class:**

This class manages the set of individual Bundle objects.

**Node class:**

A Node object represents a node, e.g. dtn://node.dtn. HBSD creates a Node object whenever it learns of a node, such as when a link is established to a node, or when a received bundle references a node.

**Nodes class:**

This is the class that manages the set of individual Node objects.

**Link class:**

A Link object represents a DTN2 link and an instance is created whenever DTN2 notifies HBSD that it has created a link. A Link object may become associated with a Node object when the link is opened; a DTN2 link that is not open is not associated with a Node. When a link is open it represents communication with another node. HBSD will associate the Link with the corresponding Node object, unless the Node object is already associated with another Link object. A node will never be associated with more than one link, even if there are multiple links open to the same node.

**Links class:**

The Link class manages the set of individual Link objects.

**Policy class:**

The Policy class defines the interface to be implemented by a Policy Manager. The interface source code describes the individual methods. By default, HBSD_Policy implements this class, but other implementations can be defined via the HBSD configuration file.

**HBSD_Policy class:**

This class is an implementation of the Policy class. It provides the HBSD scheduling and drop algorithm, but it is also generically referred to as the Policy Manager. There are calls into the Policy class sprinkled throughout the router, often mirroring the XML events defined by the /etc/router.xsd schema file. HBSD_Policy largely consists of manipulating shadow data structures dealing with bundles and nodes. The primary function of HBSD_Policy is to prioritize the delivery and replication of bundles in anticipation of the local node coming into contact with another node. The assumption is that HBSD will be able to replicate only a subset of its bundles on each node that it meets, and that some of the bundles will expire before HBSD comes in contact with the actual destination node.

**More Questions:**

---

Please feel free to drop me an email to Amir.Krifa@sophia.inria.fr if you have any question related to any class of the HBSD package.