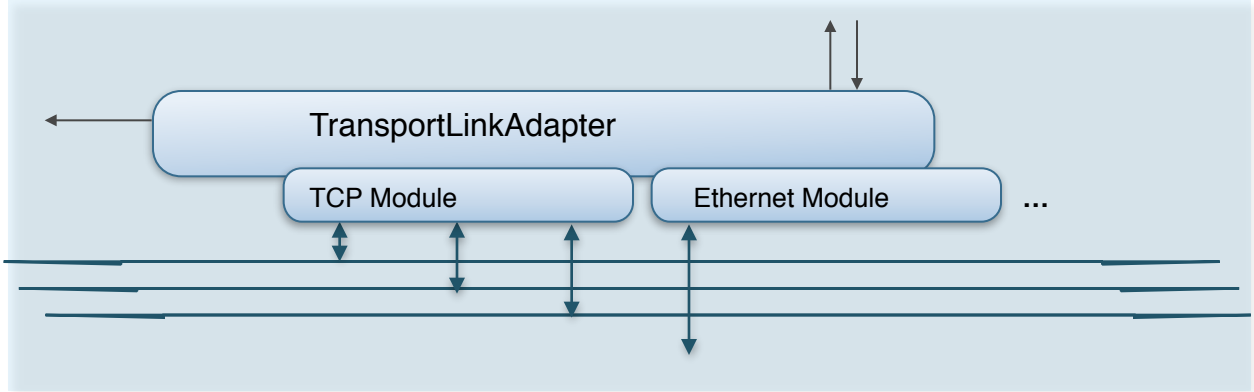
Networking and Distributed Systems
Computing Science Laboratory
Copyright 2015

CCNx Transport Link Adapter

Kevin Fox[1*]

Abstract
The TransportLinkAdapter provides link interface management and is responsible for scheduling message transport for the forwarder by interfacing with link specific transport modules.

[1]*Computing Science Laboratory, PARC*
***Corresponding author**: kevin.fox@parc.com



Introduction

The TransportLinkAdapter is layered between the forwarding engine and external communication channels for managing the transport of messages.

Typically these channels are composed of ethernet or socket based tunnel connections, but may be implemented using shared memory or any other IPC mechanism. From the TransportLinkAdapters point of view messages are sent and received using link specific modules which isolate the implementation details of the link transport mechanisms.

When a new link needs to be created, link specific information which is necessary for creating and establishing the link is passed along to a link specific module, which then establishes the link and passes a structure back to the TransportLinkAdapter providing interfaces for message transmission and the notification and handling of events.

Messages can be queued, ingress or egress, within the TransportLinkAdapter if it's running in a multi-threaded environment. The architecture specified here makes no assumptions about threading or concurrency, although all components in the forwarder are, and should be, thread safe to allow for the case where developers wish to devise and implement scheduling policies and queueing.

1. Components

This section documents the exported interfaces for two components, the TransportLinkAdapter layer, and the interface for it's associated Modules.

1.1 TransportLinkAdapter

1.1.1 athenaTransportLinkAdapter_Create

Create a new TransportLinkAdapter instance. There is typically only one required, but multiple instances may be created if resources are coordinated appropriately.

1.1.2 athenaTransportLinkAdapter_Destroy

Destroy a TransportLinkAdapter instance. This call results in the closing of all registered links and modules.

1.1.3 athenaTransportLinkAdapter_Open

When **athenaTransportLinkAdapter_Open** is called it invokes an open method on the Link Specific Module (see below), the module then registers the new link by up-calling to the TransportLinkAdapter **AddLink** method with the newly create TransportLink instance.

1.1.4 athenaTransportLinkAdapter_Close

Used to close the specified links. This involves coordination with the PIT and FIB. In order to keep the instance list as small as possible so that link vectors that the PIT and FIB need to maintain are minimized, link index slots are reused if possible when new links are added.

The PIT and FIB can be told to no longer accept entries for an index, however, until the PIT and FIB have cleaned their references any new instance using the same index could potentially be sent messages inadvertently.

Once verification has been made that the PIT and FIB have removed all references to the link, the forwarder can accept route commands for a new instance with the same id and the PIT can start accepting interests from the new instance.

The close method is responsible for verifying that references to the link have been removed from the PIT and FIB, which then allows the TransportLinkAdapter reassign the link index identifier.

1.1.5 athenaTransportLinkAdapter_Receive

The TransportLinkAdapter receive method scans its entire list of open instances in a round-robin manner. When a receive event is found the TransportLinkAdapter calls the link specific module receive method and then if a message is returned the TransportLinkAdapter will return that input message along with a result bit vector that indicates the index of interface the message was received on.

If a message was not available nor received on any instance the result vector returned will be empty. A timeout may be provided, which can attempt to ensure the receive returns if no messages become available before the timeout elapses. A timeout parameter may be specified which indicates that the call should block until a message is received.

1.1.6 athenaTransportLinkAdapter_Send

The TransportLinkAdapter Send method attempts to send the provided message on each instance specified in the input vector list. It returns a vector of the links on which the message was successfully sent.

A timeout parameter may be provided which will cause the send to to attempt to return before the specified time has elapsed if an operation would block. A timeout parameter can also indicate that the call should block until all sends have been attempted, but does not imply that failed operations should be retried.

Unrecoverable error events are handled by closing the link, issuing a notification and

marking it pending to close. Subsequently, the PIT and FIB are notified that the link has been removed and the TransportLinkAdapter waits for notification that the PIT and the FIB have eliminated all references to the link from any of their outstanding entries before reusing the link index. Operations to a closed interface should be logged and return an error.

1.1.7 _athenaTransportLinkAdapter_AddLink

The **_athenaTransportLinkAdapter_AddLink** method is only used internally by link specific modules to register new links with the TransportLinkAdapter. Links may be removed from the module by calling the TransportLinkAdapter close method for the designated instance index.

Each new instance passed onto **_athenaTransportLinkAdapter_AddLink** is inserted into the TransportLinkAdapter list of active instances either by reusing an available closed interface (see below) or appending it to the end of its current list. Modules may add an instance at any time, however instances remain in a pending state until they've been verified or allowed. Until that time, messages may not be sent or received on a link that's marked pending.

Instance names may be specified by the caller and must be unique in the context of the current set of active links. If not specified, a unique instance name is assigned by the TransportLinkAdapter. Attempting to add an instance with a non-unique name will fail.

The index of each instance in the TransportLinkAdapters instance list is used as the interface vector id in all PIT and FIB entries as well as the TransportLinkAdapters send, receive and close methods. Index identifiers may not be exported outside of the Forwarder as the links backing them may change and only the Forwarder is notified of their status; only the names of links may be exchanged with externally.

1.2 TransportLinkAdapter Modules

Link specific modules are responsible for creating, cleaning up and managing events on behalf of all their created instances.

Each transport specific module provides a pair of interfaces to the TransportLinkAdapter, open and poll.

1.2.1 athenaTransportLinkModule_Open

The module open method is called by the TransportLinkAdapter with a set of module specific arguments. After the link is established the module calls back into the TransportLinkAdapter with an TransportLink structure containing the send, receive and close operations specific to the new instance along with a set of event flags and a pointer to any necessary private instance specific data (see **athenaTransportLinkAdapter_AddLink**).

1.2.2 athenaTransportLinkModule_Poll

The module poll method is called by the TransportLinkAdapter to allow the module to refresh events on its instances and update the event flags in each of its exported instance structures appropriately. These event flags are used by the TransportLinkAdapter to schedule message transport.

```
AthenaTransportLinkInstance {
   char *Name;
   int Id;
   int (*Send)(CCNxMetaMessage *message);
   CCNxMetaMessage *(*Receive)();
   void (*Close)();
   int readEvent;
   int writeEvent;
   int errorEvent;
   void *instanceData;
}
```

1.2.3 LinkModule Instance Data

Module specific instances are interfaced by methods provided in by an instance structure.

1.2.4 Send

An instances Send method is responsible for converting the provided CCNxMetaMessage into a format appropriate for its interface and transmitting the message. It returns zero if the transfer completes successfully, and -1 on failure. In the case of failure, errno is set to give an indication as to the cause:

EIO	5	Input/output error
EFAULT	14	Bad address
EINVAL	22	Invalid argument
EFBIG	27	File too large
EAGAIN	35	Resource Temporarily unavailable

[Typically, a message will need to be converted to raw wire format, which may be obtained from the message using ccnxWireFormatFacade_Get.(?)]

1.2.5 Receive

The Receive method is responsible for reading a complete message and wrapping it in a **CCNxMetaMessage** before returning it. A Receive call can succeed even if a message isn't returned. This can be the case when an instance is created to accept connections from a remote party, in which case the module will create the new instance by calling **athenaTransportLinkAdapter_AddLink** to instantiate the new link and will return an empty result Vector with no message.

1.2.6 Close

The Close method is provided so a module can perform any necessary cleanup before being decommissioned.

1.2.7 Events

Read, write and error events are all to be marked appropriately by the Module for each instance when the Link Specific Module Poll method is called. When the TransportLinkAdapter needs to schedule an operation it inspects these flags before invoking any other methods on the associated link.

2.0 Interface Specifications

```
AthenaTransportLinkAdapter *
athenaTransportLinkAdapter_Create(
    AthenaTransportLinkAdapter_RemoveLinkCallback *
        removeLinkCallback)

void
athenaTransportLinkAdapter_Destroy(
    AthenaTransportLinkAdapter **)

PARCBitVector *
AthenaTransportLinkAdapter_Receive(
    AthenaTransportLinkAdapter *athenaTransportLinkAdapter,
    CCNxMetaMessage **ccnxMessage,
    struct timeval *timeout)

PARCBitVector *
athenaTransportLinkAdapter_Send(
    AthenaTransportLinkAdapter *athenaTransportLinkAdapter,
    CCNxMetaMessage *ccnxMessage,
    PARCBitVector *egressLinkVector)

int
athenaTransportLinkAdapter_Open(
    AthenaTransportLinkAdapter *athenaTransportLinkAdapter,
    char *moduleName,
    char *linkName,
    void *linkData)

void
athenaTransportLinkAdapter_Poll(
    AthenaTransportLinkAdapter *athenaTransportLinkAdapter)

PARCBitVector *
athenaTransportLinkAdapter_Close(
    AthenaTransportLinkAdapter *athenaTransportLinkAdapter,
    PARCBitVector *linkVector)

char *
athenaTransportLinkAdapter_LinkIdToName(
    AthenaTransportLinkAdapter *athenaTransportLinkAdapter,
    unsigned linkId)

unsigned
athenaTransportLinkAdapter_LinkNameToId(
    AthenaTransportLinkAdapter *athenaTransportLinkAdapter,
    char *linkName)
```