BRIDGING

August 2000

BRIDGING

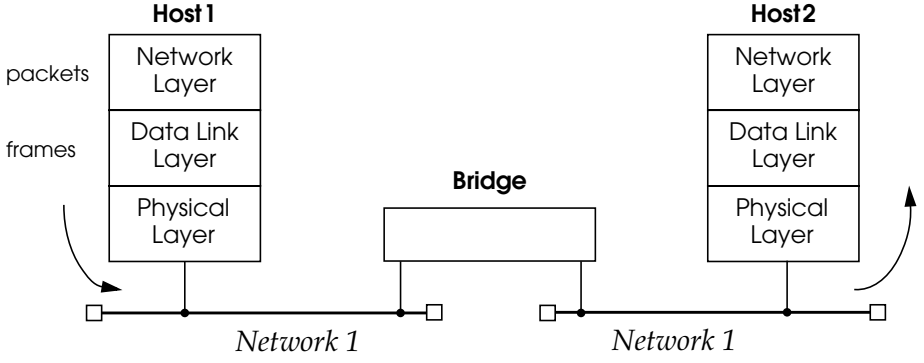
Table of Contents

REFERENCE

1 Configuring the BinTec router as a Bridge

1.1 Background on Bridging

Bridging is one of the easiest ways to connect network segments. A bridge is attached to two or more networks and simply forwards frames between them. The contents of these frames are of no concern to the bridge; frames are forwarded unchanged.



In transparent bridging each bridge makes its own routing decisions and is therefore 'transparent' to the communicating hosts on the end networks. Additionally, a transparent bridge configures itself (in terms of routing information) after coming into service.

Because a bridge forwards complete frames between connected networks many different protocols can coexist on either network, the messages are forwarded unchanged (protocol information is passed as raw data in the ethernet frames). Bridges are used when multiple-protocol packets need to be shared among networks.

1.2 Bridging with the BinTec router

1.2.1 Bridging Features

As with most bridges you simply have to connect the BinTec router to two or more network segments and turn bridging on. However, several bridging features are available on the BinTec router that can be used to overcome some of the limitations inherent in bridging.

Learning Bridges

The BinTec router is a learning bridge. Using the source and destination hardware addresses the BinTec router decides which physical interface to forward each packet it receives. This decision is made by consulting its forwarding database, the ***dot1dTpFdbTable***. Each entry in this table has the following fields:

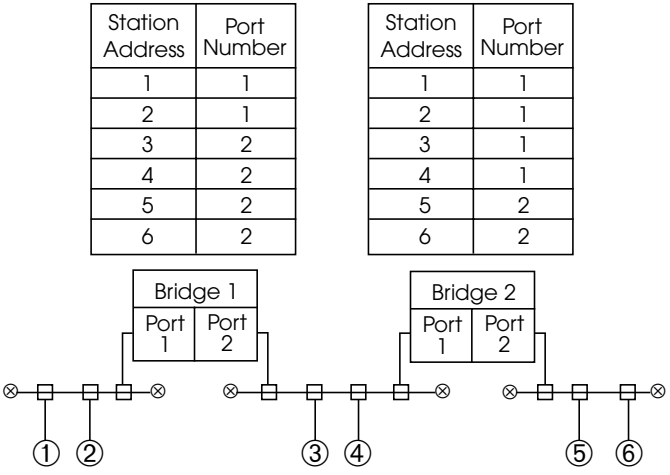
Address	Contains MAC (Medium access control) addresses.
DestPortIfIndex	The respective BinTec router interface number to use when bridging frames for this address.
Status	How this information was learned.
Age	How old this information is (see below).

When the BinTec router first comes into service, its forwarding database is empty. Then, when a frame is received, the source address of the frame and the interface it was received on are entered into the database. Since the destination interface is not yet known, a copy of the frame is broadcasted on each of the BinTec router's interfaces. As frames are propagated, other learning bridges perform the same

procedure. This allows other bridges on the network to rapidly build up their forwarding database.

To ensure that the ***dot1dTpFdbTable*** is current, and doesn't get too large, each entry has an ***Age*** associated with it. Whenever a frame is received from that address this field is reset. If no frame arrives before the Aging Timer expires, the entry is removed from the database. The BinTec router uses the ***dot1dTpAgingTime*** variable which is set to 300 seconds by default.

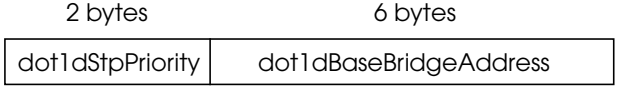
A simple bridge example is shown below.



The Spanning Tree Algorithm

The simple learning procedure explained above is only sufficient for simple networks; i.e. multiple paths between two segments can not exist since the learning process would constantly cause entries to be overwritten. In such cases the BinTec router uses an additional mechanism, known as the Spanning Tree Algorithm that compensates for network topologies with multiple paths between stations.

The spanning tree algorithm defines special frames (messages) known as bridge protocol units (BPDU) which are exchanged among all bridges on a network. Each bridge is uniquely identified by an 8 byte Bridge ID. On the BinTec router this ID is determined using the ***dot1dStpPriority*** and ***dot1dBaseBridgeAddress*** objects as follows:



Also one bridge must be singled out as the root bridge; this is the bridge with the smallest identifier and the highest priority value. After the root bridge has been chosen, each bridge determines which port offers the lowest cost path to the root (its root port). The root bridge's address is stored in the ***dot1dStpRootPort*** object. Configuration BPDUs are transmitted at regular intervals, defined by the Hello Time (***dot1dStpHelloTime***), to ensure that the root bridge information is current.

Bridge Filtering

Bridge Filtering allows you to control the amount of traffic that may be passed over the BinTec router's interfaces. This is an important tool when bridging over ISDN links since every WAN connection costs money. Bridge Filters follow the same concept used with Access Lists in IP Routing consisting of Allow and Deny Tables as follows.

dot1dStaticAllowTable	Defines packets that may be bridged.
dot1dStaticDenyTable	Defines packets that may NOT be bridged.

Each table consists of the following fields:

SrcIfIndex	The BinTec router interface the frame was received on.

DstIfIndex	The BinTec router interface the frame would be forwarded on.
ByteOffset	The number of bytes to skip (starting from the beginning) before making a comparison.
Mask	Which bytes are compared.
Value	The actual value to look for in the packet.
Status	The status of this entry.
Age	The age of this entry in seconds (used with the ***Status*** field).

Using these fields you can filter packets based on one or more criteria:

1. The packet's Source Interface
2. The packet's Destination Interface
3. A specific field in the Ethernet Frame

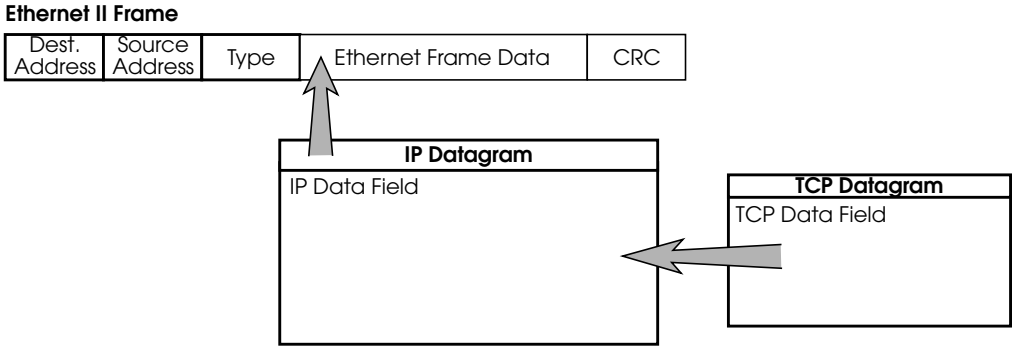
Filter Matching Procedure

Bridge filtering is sometimes referred to as packet-filtering because the decision to allow or deny (filter) a packet is based on the contents of the packet. As frames are received the contents are compared to each defined filter, starting with the Allow Table, then the Deny Table. A packet is said to match a filter if all its conditions are met. A filter condition may be one of the following.

SrcIfIndex	= the interface this frame was received on.
DstIfIndex	= the interface this frame would be forwarded on.
Value	= the contents of the frame starting from ***ByteOffset*** bytes

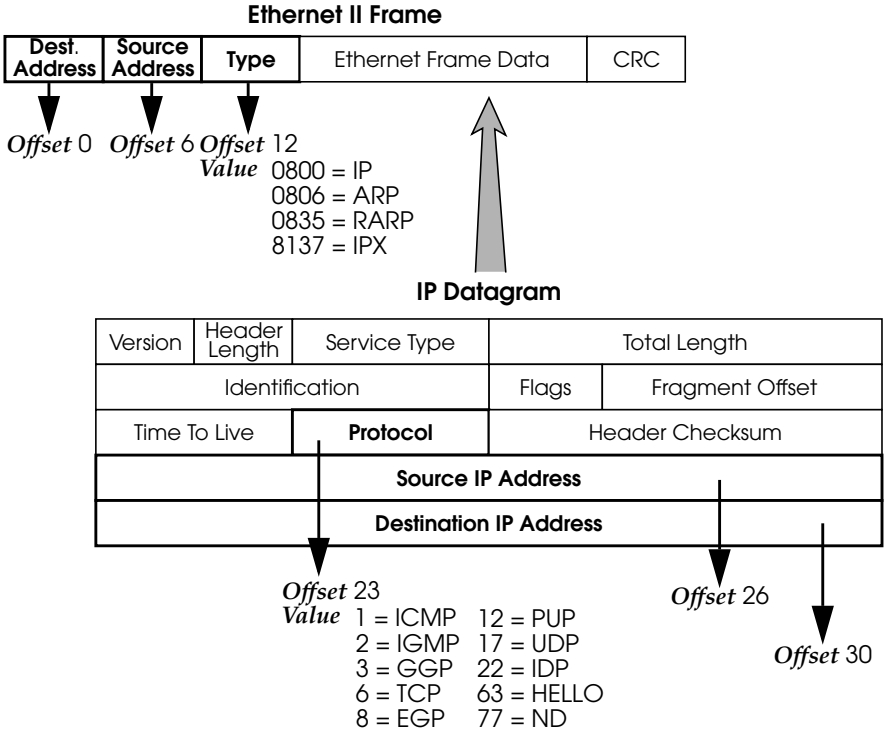
NOTE: A "0" in the ***SrcIfIndex*** and/or ***DstIfIndex*** fields means match any interface.

The common practice of encapsulating datagrams within datagrams (as shown below) poses no problem to the BinTec router. As a bridge it doesn't differentiate the individual fields, it simply sees the complete packet as a sequence of bytes. Using the ***ByteOffset*** and ***Value*** fields you can define filters that are based on the actual contents of the frame.

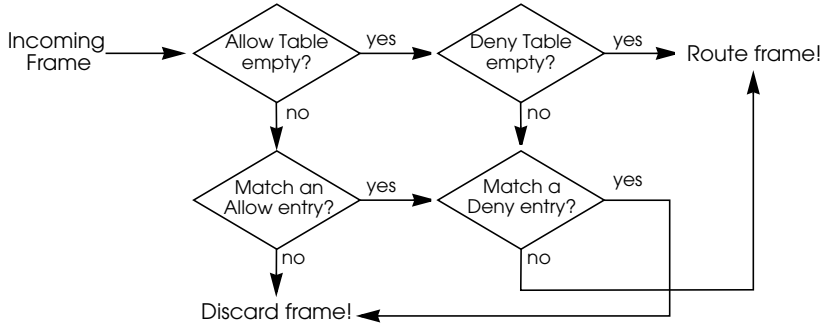


When filtering packets based on contents of the ethernet frame it's important to know the relative locations (offset) and possible values of the frame's fields. Below is an ether-

net frame that highlights some of the more interesting fields (with respect to bridge filtering).



The packet is compared with all Allow and Deny Table entries and a decision is made based on the following algorithm.



1.3 Configuring Bridging on the BinTec router

1.3.1 Enabling Bridging

There are two basic requirements to fulfil before the BinTec router begins bridging.

1. The ***biboAdmBridgeEnable*** object must be set to "enabled".
2. At least two interfaces must be enabled in the ***dot1dStpPortTable.*** Here, two or more interfaces' ***Enable*** field must be set to "enabled".

NOTE: Both steps can be accomplished using Setup Tool. Refer to Chapter 5 of the User's Guide.

1.3.2 Bridge Initialization

Once bridging has been enabled, the system goes through three internal phases before bridging can actually take place.

1. **Listening**—During the listening phase the system transmits configuration BPDUs and evaluates any others it receives. In this phase the spanning tree is computed and the root bridge is determined. The bridge ID of the root bridge is then set in ***DesignatedRoot*** variable of the ***dot1dStpPortTable***.
2. **Learning—**The system then switches to the learning phase where all frames it receives are evaluated.
3. **Forwarding**—After a preset time (defined in IEEE802.1d-1990), the BinTec router switches to the forwarding state.

Once the system reaches the forwarding state, the BinTec router checks the ***dot1dStpPortTable*** to see which interfaces are available for bridging.

1.4 Using the BinTec router as a Bridge

Below are some examples of setting up the BinTec router as a bridge.

- In the first example we'll use the BinTec router to bridge between two local LAN segments (i.e., a BRICK-XM or BRICK-XL with two LAN interfaces is assumed.)
- In the second example we'll use the BinTec router as a bridge to connect a local LAN with a remote LAN over a dialup ISDN link.
- In the last example we'll extend the previous example showing you how to add filters to control bridging traffic and to save money.

1.4.1 Bridging between LANs

Step 1

First make sure the bridging service is enabled on the BinTec router.

The ***biboAdmBridgeEnable*** variable must be set to "**enabled**".

```
mybrick: > biboAdmBridgeEnable=enabled

biboAdmBridgeEnable( rw):       enabled

mybrick : admin>
```

Step 2

Next we need to enable the interfaces we want to bridge between. The ***dot1dStpPortTable*** lists all available interfaces for bridging. For ethernet interfaces you must use the "*-llc" interface. This ensures that the LLC (Link Layer Control) frame format is used.

```
mybrick : admin> dot1dStpPortTable

inx IfIndex(*ro)                Number(ro)              Priority(rw)
    State(ro)                   Enable(rw)              PathCost(rw)
    DesignatedRoot(ro)          DesignatedCost(ro)      DesignatedBridge(ro)
    DesignatedPort(ro)          ForwardTransitions(ro)  BackupForIfIndex(rw)

 00 1001                        0                       0
                                disabled                0
    80:0:0:a0:f9:0:e:91         0                       80:0:0:a0:f9:0:e:91
    0                           0                       0

01  2001                        0                       0
                                disabled                0
    23:a0:21:a3:f5:0:d:88       0                       80:0:0:a0:f9:0:e:91
    0                           0                       0

mybrick : dot1dStpPortTable> Enable:00=enable Enable:01=enable
```

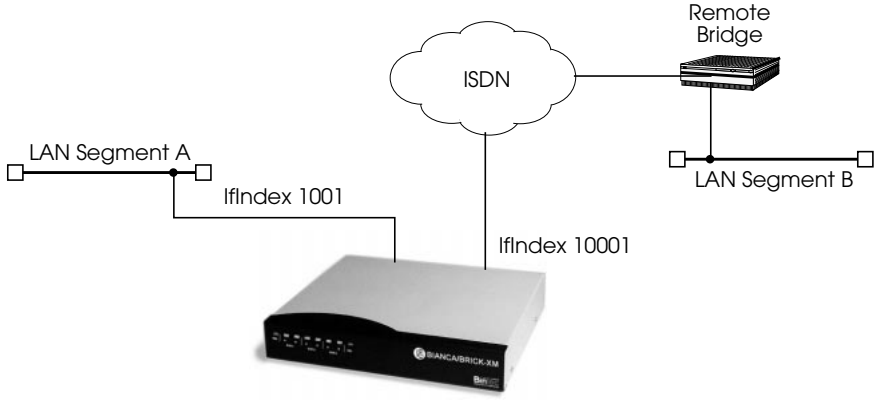
Step 3

That's all that is required for our setup. The BinTec router will now run through it's Bridge Initialization functions. You can optionally see the state of the bridging interfaces by displaying the ***dot1dStpPortTable***. You can also see the list

of learned bridging entries by displaying the ***dot1dTpFdpTable***.

```
mybrick : admin> dot1dTpFdbTable

inx Address(*rw)          DestPortIfIndex(rw)      Status(-rw)
    Age(ro)

    00 0:a0:f9:0:e:91     1001                     self
    0 00:12:47.00

    00 0:a0:f9:0:c:2c     2001                     self
    0 00:12:47.00

    00 0:a0:f9:0:b:12     1001                     learned
    0 00:12:47.00

mybrick : dot1dTpFdbTable >
```

1.4.2 Bridging over WAN Links



Step 1

We'll assume the bridging service has been enabled (see previous example). First, create a new PPP interface for the Remote Bridge. Creating PPP interfaces is covered in chapter 7. Here we set the ***Type*** field in the ***biboPPPTable*** to "**`isdn_dialup`**". The BinTec router will assign a unique

numerical value to the ***IfIndex*** field that we'll need in Step 2.

```
mybrick: admin > biboPPPType=isdn_dialup
05: biboPPPType.1.5( rw):      isdn_dialup

mybrick : biboPPPTable > biboPPPTable
inx  IfIndex(ro)              Type(*rw)              Encapsulation(-rw)
     Keepalive(rw)            Timeout(rw)            Compression(rw)
     Authentication(rw)       AuthIdent(rw)          AuthSecret(rw)
     IpAddress(rw)            RetryTime(rw)          BlockTime(rw)
     MaxRetries(rw)           ShortHold(rw)          InitConn(rw)
     MaxConn(rw)              MinConn(rw)            Callback(rw)
     Layer1Protocol(rw)       LoginString(rw)

05   10006                    isdn_dialup            ppp
     off                      3000                   none
     none
     static                   4                      300
     5                        20                     1
     1                        1                      disabled
     data_64k

mybrick : biboPPPTable>
```

Step 2

Now we can add the Remote Bridge's telephone number to the ***biboDial-Table***. We set the ***IfIndex*** and the ***Number*** field in one operation. In the ***IfIndex*** field, we use the number assigned by the BinTec router in the previous step.

```
mybrick: biboPPPTable > biboDialIfIndex=10006 biboDialNumber=555

06: biboDialIfIndex.10006.6( rw):      10006
06: biboDialNumber.10006.6( rw):       "555"

mybrick : biboDialTable >
```

Step 3

Now we can enable the local and the remote interfaces to bridge between. The ***dot1dStpPortTable*** should have an en-

try for our local ethernet segment as well as our new PPP partner interface.

```
mybrick : admin> dot1dStpPortTable

inx IfIndex(*ro)                Number(ro)              Priority(rw)
    State(ro)                   Enable(rw)              PathCost(rw)
    DesignatedRoot(ro)          DesignatedCost(ro)      DesignatedBridge(ro)
    DesignatedPort(ro)          ForwardTransitions(ro)  BackupForIfIndex(rw)

 00 1001                        0                       0
    forwarding                  disabled                0
    80:0:0:a0:f9:0:e:91         0                       80:0:0:a0:f9:0:e:91
    0                           0                       0

05  10001                       0                       0
     broken                     disabled                0
                                0
    0                           0                       0

mybrick : dot1dStpPortTable> Enable:00=enable Enable:05=enable
```

The configuration is complete. We can optionally verify that bridging has started by displaying the ***dot1dStpPortTable*** and the ***dot1dTpFdpTable*** as mentioned in the previous example.

NOTE: For most sites bridging over WAN links is less desirable due to the possibility of increased ISDN costs incurred through dialup connections. With careful consideration and planning however bridge filters can be used to make bridging over WAN links a viable alternative.

Further optional settings are afforded by the following variables, which give you additional influence over your WAN-link bridges:

Delay before Change of State

The first is ***dot1StpBridgePPPForwardDelay*** and is an addition to the ***dot1dStp*** table. The unit of the value of this

variable is 1/100 seconds, the range lies between 100 and 3000 and the default value is 500 (= 5 seconds).

This variable defines how long the port of a PPP connection (leased line or dial-up connection) should wait before a change of state may take place. After the establishment of an ISDN connection, the state takes this set period of time to change from listening to learning, then the same time again to change from learning to forwarding. ***dot1StpBridgePPPForwardDelay*** only affects WAN connections like PPP and X.25.

When the default value (500) is set, it takes 5 seconds to change the state from listening to learning and another 5 seconds to change from learning to forwarding. After an ISDN connection has been made, it consequently takes 10 seconds until data is transmitted in the state forwarding.

This period of time is necessary to detect redundant paths.

Backup for a Leased Line

The second variable is ***dot1dStpPortBackupForIfIndex*** and is an addition to the ***dot1dStpPortTable***.

This variable is conceptualised for a situation in which two BinTec routers are bridging two local networks over PPP connections. One of these is a leased line, the other a dialup line.

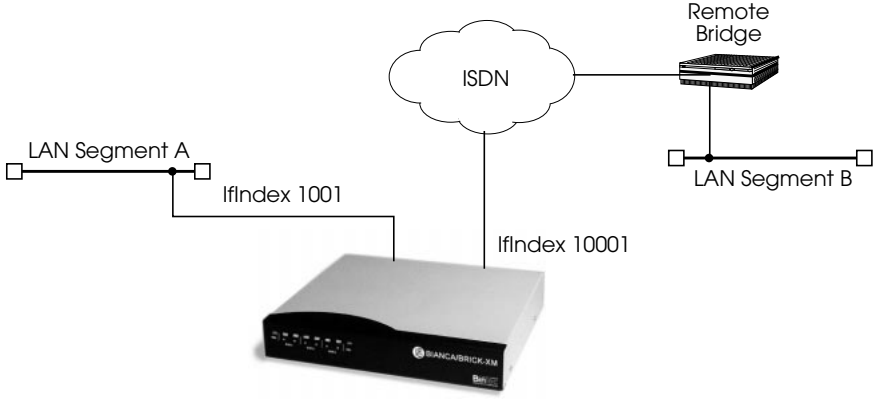
dot1dStpPortBackupForIfIndex is used to configure the dialup connection as a backup connection for the leased line connection.

In the ***dot1dStpPortTable***, set the value of the ***dot1StpPortBackupForIfIndex*** variable for the interface of the dialup line with the same value as in the ***dot1dStpPortIfIndex*** of the leased line port. This effectively makes the port of the dialup line serve as the backup link for the leased line. As long as the leased line functions properly (its state is forwarding), the dialup link is not established. Should the leased line fail, however, the dialup link

(backup) is established and the entries for the port of the leased line in the ***dot1dTpFdbTable*** are deleted.

1.4.3 Controlling Bridging Activity Using Filters

Now we want to show you how to use Bridge Filters to control bridging traffic. Bridge filters are most commonly used when bridging over ISDN WAN links to minimize costs. Remember that bridge filtering is based on the contents of the ethernet frame. An overview of the ethernet frame format was covered here.



Following are three examples that could be used to extend the previous example for bridging over WAN links. We'll assume bridging is already configured so we can focus on the filter entries. The following three examples show how to:

1. **Filtering frames sent from a particular host (by MAC address)**
   Here we want to single out packets from a specific host by filtering the MAC address field of the ethernet frame.

2. **Filtering all IPX packets coming from the local LAN**
   Here we want to filter out all IPX packets, this can be done by filtering out the Type field in the MAC header.
3. **Filtering broadcast packets**
   Here we want to filter out all broadcast packets (destination address field is ff:ff:ff:ff:ff:ff in hexa-decimal).

Filtering frames sent from a particular host (by MAC address)

To filter out all frames sent from a specific host we first need the host's MAC (hardware) address. Then, all we need is one Deny Filter that filters out all frames sent from this host.

Assuming our source host had a MAC address of 0:a0:f9:0:e:19 and was attached to the BinTec router's first ethernet interface (en1 or 1001) our filter would be created as follows.

```
mybrick: admin > dot1dStaticDenyTable

inx SrcIfIndex(*rw)          DstIfIndex(*rw)          ByteOffset(rw)          Mask(rw)
    Value(rw)                Status(-rw)              Age(rw)

mybrick: dot1dStaticDenyTable >SrcIfIndex=1001 DstIfIndex=0 ByteOffset=6
                               Value=0:a0:f9:0:a0:19

00: dot1dStaticDenySrcIfIndex.1001.0( rw):          1001
00: dot1dStaticDenyDstIfIndex.1001.0( rw):          0
00: dot1dStaticDenyByteOffset.1001.0( rw):          6
00: dot1dStaticDenyValue.1001.0( rw):               0:a0:f9:0:a0:19

mybrick: dot1dStaticDenyTable > dot1dStaticDenyTable

00  1001                     10001                    6
    0:a0:f9:0:a0:19          permanent                0 00:03:24.00

mybrick: dot1dStaticDenyTable >
```

Filtering all IPX packets coming from the local LAN

NOTE: Since the ***SrcIfIndex*** and ***DstIfIndex*** fields are index variables (required for creation of new table entries) we also specify them here. Also rememer that the special value "0" matches all interfaces.

A single Deny filter is all that's needed to filter out all IPX packets originating on a specific LAN segment. IPX packets are identified by the protocol ID of 0x8137 in the Type field of the Ethernet frame. This filter will ensure that no IPX packets coming from the LAN segment are bridged to our ISDN interface at 10001.

As in the previous example we'll assume basic bridging has already been configured. Our filter would be created as follows.

```
mybrick: dot1dStaticDenyTable >

mybrick: dot1dStaticDenyTable > SrcIfIndex=1001 DstIfIndex=10001
                                ByteOffset=12 Value=81:37

01: dot1dStaticDenySrcIfIndex.1001.0.1( rw):          1001
01: dot1dStaticDenyDstIfIndex.1001.0.1( rw):          10001
01: dot1dStaticDenyByteOffset.1001.0.1( rw):          12
01: dot1dStaticDenyValue.1001.0.1( rw):               81:37

mybrick: dot1dStaticDenyTable > dot1dStaticDenyTable

inx SrcIfIndex(*rw)          DstIfIndex(*rw)          ByteOffset(rw)          Mask(rw)
    Value(rw)                Status(-rw)              Age(rw)

01  1001                     10001                    0
    81:37                    permanent                0 00:10:06.00

mybrick: dot1dStaticDenyTable>
```

NOTE: If we had several exceptions to this rule to account for (filter all IPX packets except those from hosts x,y, and z) we would create either a series of Allow entries or Deny entries for individual host MAC addresses depending on which required the least entries.

Filtering broadcast packets

Broadcast packets can also be easily filtered out using the "Destination Address" field in the MAC frame. Broadcast addresses can be identified by the value ff:ff:ff:ff:ff:ff there. This filter will ensure that no new ISDN connections are opened to dialup partner interface 10001 to bridge broadcast frames sent from this LAN segment.

Since the Destination Address is the very first field in the MAC frame no offset value is required. Our filter would be created as follows.

```
mybrick: dot1dStaticDenyTable >

mybrick: dot1dStaticDenyTable >SrcIfIndex=1001 DstIfIndex=10001 Value=ff:ff:ff:ff:ff:ff

02: dot1dStaticDenySrcIfIndex.1001.0.2( rw):        1001
02: dot1dStaticDenyDstIfIndex.1001.0.2( rw):        10001
02: dot1dStaticDenyValue.1001.0.2( rw):             ff:ff:ff:ff:ff:ff

mybrick: dot1dStaticDenyTable > dot1dStaticDenyTable

inx SrcIfIndex(*rw)          DstIfIndex(*rw)          ByteOffset(rw)          Mask(rw)
    Value(rw)                Status(-rw)              Age(rw)

02  1001                     10001                    0
    ff:ff:ff:ff:ff:ff        permanent                0 00:19:06.00

mybrick: dot1dStaticDenyTable>
```

NOTE This filter catches all broadcast packets; hosts on the LAN segment would no longer be capable of establishing IP connections to remote hosts because this fillter also restricts ARP requests. If this is not desired, individual hosts could be configured to operate with static arp entries.

On most systems (Solaris, SunOS and DOS) the command is similar to:

```
arp -s <hostname> <ether_address>
```

Refer to the documentation provided by your OS.