CHAPTER 3

UNIQUE INDEX MANAGEMENT IN ROUTING
3.1 Introduction

Management information is viewed as a collection of managed objects, residing in a virtual information store, termed the Management Information Base (MIB). Collections of related objects are defined in MIB modules. These modules are written using an adapted subset of OSI's Abstract Syntax Notation One, ASN.1 (1988), termed the Structure of Management Information (SMI).

It may be useful to define the acceptable lower-bounds of implementation, along with the actual level of implementation achieved. It is the purpose of this document to define the notation used for these purposes.

This defines a portion of the Management Information Base (MIB) for use with network management protocols in the Internet community. In particular, it describes managed objects for information related to the network's Routing Table for routing within a Fabric[24,26]. Managed objects specific to particular routing protocols, such as the Fabric Shortest Path First (FSPF) protocol [FC-SW-4], are not specified in this MIB module.

Ever though real time recovery from MIB storage and retrieval failure take place but it is essential for maintaining network performance, previous approaches still suffer from critical solution as follows, existing storage retrieval algorithm [18, 17, 30] con provide data lines for network planning but they may required “global” network configuration changes for every retrieval, thus incurring a very high management over head or network description in retrieval. Next a greedy channel assignment algorithm [21] can value the requirement of network storage but one local greedy reconfiguration could cause, QOS degradation or another failure at neighboring nodes, trigger “propagation” of QOS failures[19,20].

To overcome the above limitation, we propose a novel globalized framework for storage retrieval from the router called “Unique index management in routing “(UIMR) that allows the how to store & retrieval the data’s from the data very effective. In its core, UIMR includes the planning algorithm of managing, releasing and searching the indexes and which minimize the searching time with respect to network routers settings. UMR first places in the Router group storage around an a head based on router associations. Then, by improving the current algorithm of indexes, the algorithm reduces the number of updates that each managing reconfiguration has to make.

Department of Computer Science, S.K University, Anantapur.
Unique index management in routing

Next UIMR is also equipped with virtual fabrics switch index, fiber index to accurate line quality information and any configuration plan generated can be applied.

UIMR is implemented and evaluates extensively via experimentation on t1FC router as well as identifies status, description, reference, object group clauses. Our evaluation result show that UIMR outperforms existing methods such as greedy channel assignments and also with the data structure with linked range and bit lists. First, this design is accommodating two times more flexible than before assignment. Next, UIMR avoids the creation retrieval and deletion of the indexes with 215 indexes of accuracy. Third, UIMRs improve network through by a very good results respectively over the rerouting data structure scheme.

The rest of this chapter is organized as follows section 3.2 describes the motivation behind this work, section 3.3 provides the design rationale and algorithm of UIMR, section 3.4 describes the implementation and experimental results of UIMR. Section 3.5 shows the model, methods and performance evaluation. Section 3.6 discusses the remaining issues associated with UIMR and concludes the chapter.

3.2 Motivation

MIB contains one object group, the RouteGroup, which contains objects to allow the displaying and the figuring of routes in the Fibre Channel Routing tables for the locally managed switches.

RouteTable's INDEX

It is normally valuable for a MIB table that contains routes to be ordered such that a management application is able to query the table based on some attribute, without having to read every row in the MIB table. This requires that the rows in the table be ordered according to such attributes, and thus that those attributes be represented by objects included in the table's INDEX clause. Examples of this can be seen in the ipCidrRouteTable and, more recently, the inetCidrRouteTable.

While this useful feature results in an unusually large number (ten) of objects in the RouteTable's INDEX clause, all ten are either integers or strings of 3 (or zero) octet length, so the resulting OIDs are not unusually large. (Specifically, the aggregate number of sub-identifiers to be appended to an OBJECT-TYPE's OID, when naming an instance of an object in the RouteTable, is at most 22 sub-identifiers;
i.e., less than the *minimum* number to be appended for the inetCidrRouteTable table.)

3.2.1 Why is index management necessary?
For a detailed overview of the documents that describe the current Internet-Standard Management Framework. Managed objects are accessed via a virtual information store, termed the Management Information Base or MIB. MIB objects are generally accessed through the Simple Network Management Protocol (SNMP). Objects in the MIB are defined using the mechanisms defined in the Structure of Management Information (SMI).

3.2.1a Overview of fiber channel & UIMR
The Fibre Channel (FC) is logically a bidirectional point-to-point serial data channel, structured for high performance. Fibre Channel provides a general transport vehicle for higher-level protocols, such as Small Computer System Interface (SCSI) command sets, the High-Performance Parallel Interface (HIPPI) data framing, IP (Internet Protocol), IEEE 802.2, and others.

Physically, Fibre Channel is an interconnection of multiple communication points, called N_Ports, interconnected either by a switching network, called a Fabric, or by a point-to-point link. A Fibre Channel "node" consists of one or more N_Ports. A Fabric may consist of multiple Interconnect Elements, some of which are switches. An N_Port connects to the Fabric via a port on a switch called an F_Port. When multiple FC nodes are connected to a single port on a switch via an "Arbitrated Loop" topology, the switch port is called an FL_Port, and the nodes' ports are called NL_Ports. The term Nx_Port is used to refer to either an N_Port or an NL_Port. The term Fx_Port is used to refer to either an F_Port or an FL_Port. A switch port, which is interconnected to another switch port via an Inter-Switch Link (ISL), is called an E_Port. A B_Port connects a bridge device with an E_Port on a switch; a B_Port provides a subset of E_Port functionality. Many Fibre Channel components, including the fabric, each node, and most ports, have globally-unique names. These globally-unique names are typically formatted as World Wide Names (WWNs). More information on WWNs can be found in [FC-FS]. WWNs are expected to be persistent across agent and unit resets.
Fibre Channel frames contain 24-bit address identifiers that identify the frame's source and destination ports. Each FC port has both an address identifier and a WWN. When a fabric is in use, the FC address identifiers are dynamic and are assigned by a switch. Each octet of a 24-bit address represents a level in an address hierarchy, a Domain_ID being the highest level of the hierarchy.

**Routing protocols**: The routing of frames within the Fabric is normally based on the standard routing protocol, called the Fabric Shortest Path First (FSPF) protocol. The operation of FSPF (or of any other routing protocol) allows a switch to generate and maintain its own routing table of how to forward frames it receives; i.e., a table in which to look up the destination address of a received frame in order to determine the best link by which to forward that frame towards its destination.

### 3.2.1b Virtual Fabrics
The latest standard for an interconnecting Fabric containing multiple Fabric Switch elements is [FC-SW-4] (which replaces the previous revision, [FC-SW-3]). [FC-SW-4] carries forward the existing specification for the operation of a single Fabric in a physical infrastructure, augmenting it with the definition of Virtual Fabrics and with the specification of how multiple Virtual Fabrics can operate within one (or more) physical infrastructures. The use of Virtual Fabrics provides for each frame to be tagged in its header to indicate which one of several Virtual Fabrics that frame is being transmitted on. All frames entering a particular "Core Switch" [FC-SW-4] (i.e., a physical switch) on the same Virtual Fabric are processed by the same "Virtual Switch" within that Core switch.

### 3.2.1c Relationship to Other MIBs
The first standardized MIB for Fibre Channel was focused on Fibre Channel switches. It is being replaced by the more generic Fibre Channel Management MIB [FC-MGMT], which defines basic information for Fibre Channel hosts and switches, including extensions to the standard IF-MIB] for Fibre Channel interfaces.

This MIB extends beyond [FC-MGMT] to cover the routing of traffic within a Fabric of a Fibre Channel network. The standard routing protocol for Fiber Channel

---

*Department of Computer Science, S.K University, Anantapur.*
is FSPF [FC-SW-4]. Another MIB specifies management information specific to FSPF.

3.2.2 MIB Model and Assumptions

An MIB model is assumed to consist of routing information that is independent of FSPF. It would still apply even if a routing protocol other than FSPF were in use in the network. The MIB imports some common textual conventions from T11fc MIB, defined.

3.2.2a MIB Overview

This MIB module provides the means for monitoring the operation of, and configuring some parameters of, one or more instances of the FSPF protocol. (Note that there are no definitions in this MIB module of "managed actions" that can be invoked via SNMP.)

3.2.2b Fibre Channel Management Instance

A Fibre Channel management instance is defined in [FC-MGMT] as a separable managed instance of Fibre Channel functionality. Fibre Channel functionality may be grouped into Fibre Channel management instances in whatever way is most convenient for the implementation(s). For example, one such grouping accommodates a single SNMP agent with multiple AgentX [RFC2741] sub-agents, each sub-agent implementing a different Fibre Channel management instance.

The object, fcmlnstancelndex, is IMPORTed from the FC-MGMT-MIB [FC-MGMT] as the index value that uniquely identifies each Fibre Channel management instance within the same SNMP context.

3.2.2c Switch Index

The FC-MGMT-MIB [FC-MGMT] defines the fcmSwitchTable as a table of information about Fibre Channel switches that are managed by Fibre Channel management instances. Each Fibre Channel management instance can manage one or more Fibre Channel switches. The Switch Index, fcmSwitchlndex, is IMPORTed from the FC-MGMT-MIB as the index value that uniquely identifies a Fibre Channel switch among those (one or more) managed by the same Fibre Channel management instance.
3.2.2d Fabric Index

Whether operating on a physical Fabric (i.e., without Virtual Fabrics) or within a Virtual Fabric, the operation of FSPF within a Fabric is identical. Therefore, this MIB defines all Fabric-related information in tables that are INDEX-ed by an arbitrary integer, named a "Fabric Index", the syntax of which is IMPORTed from the T11-TC-MIB. When a device is connected to a single physical Fabric, without use of any virtual Fabrics, the value of this Fabric Index will always be 1. In an environment of multiple virtual and/or physical Fabrics, this index provides a means to distinguish one Fabric from another.

It is quite possible, and may even be likely, that a Fibre Channel switch will have ports connected to multiple virtual and/or physical Fabrics. Thus, in order to simplify a management protocol query concerning all the Fabrics to which a single switch is connected, fcmSwitchIndex will be listed before t11FcRouteFabricIndex when they both appear in the same INDEX clause.

3.2.2e The RouteGroup Group

This MIB contains one object group, the t11FcRouteGroup, which contains objects to allow the displaying and the configuring of routes in the Fibre Channel Routing tables for the locally managed switches.

It is normally valuable for a MIB table that contains routes to be ordered such that a management application is able to query the table based on some attribute, without having to read every row in the MIB table. This requires that the rows in the table be ordered according to such attributes, and thus that those attributes be represented by objects included in the table's INDEX clause. Examples of this can be seen in the ipCidrRouteTable [RFC2096] and, more recently, the inetCidrRouteTable in [RFC4292].

While this useful feature results in an unusually large number (ten) of objects in the t11FcRouteTable's INDEX clause, all ten are either integers or strings of 3 (or zero) octet length, so the resulting OIDs are not unusually large. (Specifically, the aggregate number of sub-identifiers to be appended to an OBJECT-TYPE's OID, when naming an instance of an object in the t11FcRouteTable, is at most 22 sub-identifiers; i.e., less than the *minimum* number to be appended for the inetCidrRouteTable table.)
3.2.3 Limitations of Existing Approaches

One of the application but not limited to, is index generation for MIB tables where a unique index need to be used with a conceptual row for creation/retrieval/destroy operations.

There are several management objects defined in this MIB module with a MAX-ACCESS clause of read-write and/or read-create. Such objects may be considered sensitive or vulnerable in some network environments. The support for SET operations in a non-secure environment without proper protection can have a negative effect on network operations. These objects and their sensitivity/vulnerability are:

- tl1FcRouteDomainId, tl1FcRouteMetric, tl1FcRouteType,
- tl1FcRouteIfDown, tl1FcRouteRowStatus

-- configure new routes and/or modify existing routes.

Such objects may be considered sensitive or vulnerable in some network environments. For example, the ability to change network topology or network speed may afford an attacker the ability to obtain better performance at the expense of other network users. The support for SET operations in a non-secure environment without proper optimization can have a negative effect on network operations.

Some of the readable objects in this MIB module (i.e., objects with a MAX-ACCESS other than not-accessible) may be considered sensitive or vulnerable in some network environments. It is thus important to control even GET and/or NOTIFY access to these objects and possibly to even optimize the values of these objects when sending them over the network via SNMP.

The objects and their sensitivity/vulnerability are: the write-able objects listed above plus one other:

LastChangeTime

-- the time of the last routing table change.

SNMP versions prior to SNMPv3 did not include adequate security. Even if the network itself is secure (for example by using IPSec), even then, there is no control as to who on the secure network is allowed to access and GET/SET (read/change/create/delete) the objects in this MIB module.
It is RECOMMENDED that implementers consider the security features as provided by the SNMPv3 framework including full support for the SNMPv3 cryptographic mechanisms (for authentication and privacy).

Further, deployment of SNMP versions prior to SNMPv3 is NOT RECOMMENDED. Instead, it is RECOMMENDED to deploy SNMPv3 and to enable cryptographic security.

It is then a customer/operator responsibility to ensure that the SNMP entity giving access to an instance of this MIB module is properly configured to give access to the objects only to those principals (users) that have legitimate rights to indeed GET or SET (change/create/delete) them.

3.3 The UIMR design

This section details UIMR. We first present its design rationale and overall algorithm. Then, we detail UIMR's index algorithms. Finally, we discuss the complexity of UIMR.

3.3.1 Overview

This algorithm describes technique to generate and manage unique integer indexes from a specified range of integers. Generated index can be used in any application where a unique integer from a specified range need to be served as key to a particular record and when record is freed index need to be reused.

Efficiency of this algorithm lies in its simplicity and capability to manage large range of index in minimal recourses, such as running time and memory requirement.

This Algorithm is best suited for problems where 0 or 1-based unique indexes (however non 0 or 1 based indexes can also be managed with calculating a fix offset) are required to be managed with frequent operation like checking whether a index is free or not, finding first free index, reserving and freeing indexes with optimal memory usage in average case.

One of the application but not limited to, is index generation for MIB tables where a unique index need to be used with a conceptual row for creation/retrieval/destroy operations.
3.3.2 Top-Down Approach

Algorithm works on bit state (0 or 1), one of the states is used to indicate free or used index in the memory. Thus 1 bit memory is required to represent one integer index, and state of this bit can be used to determine whether index is free or occupied.

For example to generate 512 integer indexes we need 512 bit memory that is 64 Byte. In addition to this we also need few more byte to make searching of the indexes faster and efficient to use.

![Figure 3.1: Empty Chunk](image)

Above picture illustrate core of the algorithm for 8 bit base size. At level 0 has 8 bit memory which can point up to another 8 byte memory of level 1. Again at level 1 each bit can point to another 1 byte at next level. Thus level 1 has 8 byte memory and level 2 has 64 bytes of memory. If level 2 is our final level, then each bit at level 2 represents a unique index.

In above example level 2 has 64 Byte and is capable to manage 64*8 = 512 unique indexes.

3.3.2a Properties

- If all bits are set in a chunk, bit pointing to this chunk at 1 level below must also be set to 1 if exist, else bit pointing to this chunk at 1 level below must be 0.
- Level 0 will have only one chunk.
- All other higher level can have max up to N chunk, where N is no. of total bits in immediate lower level.
3.3.2b Managing Indexes

Managing indexes are some important operations to manage indexes.
Memory array at level MAX_LEVEL is used to generate and manage integer indexes.
Considering bit state 0 for free indexes and 1 for occupied indexes. Here

3.3.2c Occupying an Index

Initially all bits at all levels will be set to 0. This indicates all indexes are free

to be used.

reserveIndex ()

ALGORITHM: An algorithm for occupying an index.

\[
\begin{align*}
& h = \text{MAX\_LEVEL} - 1 \\
& \text{while } h >= 0 \\
& p = \text{index} \mod \text{BASE\_SIZE} \\
& \text{index} = \text{index} / \text{BASE\_SIZE} \\
& \text{idx\_db}[h][\text{index}] \text{ OR } (1 << p) \\
& \text{if } \text{idx\_db}[h][\text{index}] = \text{ALL\_SET} \\
& \text{h} = h - 1 \\
& \text{else} \\
& \text{break}
\end{align*}
\]

Figure 3.2. Reserve index 1
3.3.2d Releasing an Index

Case 1: When an index is released, corresponding bit at level 2 is set to 0 indicating that this index is now available to reuse.

If all bit on a single chunk (of size BASE_SIZE, in above example it is 8) are not occupied at current level, move one level below and set bit pointing to this chunk to 0.

Case 1a: Repeat above step until, reached to level 0.
ALGORITHM: An algorithm for releasing an index taking unreserved chunks into consideration.

```plaintext
unreserveIndex (index)
    h = MAX_LEVEL - 1
    while h >= 0
        p = index % BASE_SIZE
        index = index / BASE_SIZE
        if idx_db[h][index] = ALL_SET idx_db[h][index] AND ~(1 << p)
            h = h - 1
        else
            idx_db[h][index] AND ~(1 << p)
            break
```

![Diagram](image)

Fig. 3.5: Releasing index 4, Case 1b

3.3.2e Searching First Free Index

Let p be the position of first bit in the chunk which is 0 and i be the chunk no. of the current level which is being searched and h is current level.

This operation finds first free bit in a give chunk of size BASE_SIZE and returns its bit position from right.
**Unique index management in routing**

**ALGORITHM**: Operation flow for searching the first free index in chunk of current level.

```plaintext
findFree (h, i)
   for p -> 0 to BASE_SIZE
   if idx_db[h][i] & (1<<p) = 0
      break
   return p

This operation finds first smallest free index.

getFreeIndex (h, i)
   while h < MAX_LEVEL
      p = findFree(h, i)
      if p >= BASE_SIZE
         return -1
      i = i*BASE_SIZE + p
      return i
```

3.3.2f Checking Index Status

This operation can determines whether a particular index is free or reserved. It has complexity O(1).

**ALGORITHM**: Determination of status of checking the index whether the particular index is free or reused.

```plaintext
isIndexFree (index)
p = index % BASE_SIZE
i = index / BASE_SIZE
if idx_db [MAX_LEVEL-1][i] & (1<<p) = TRUE
   return FALSE
else
   return TRUE
```

3.3.3 Complexity of UIMR

Thanks to distributed and localized design, UIMR incurs reasonable storage and computation overheads, compared to centralized counterpart as follows.
3.3.3a Running Time

This algorithm is extremely fast, number of index to be generated does not have much affect on running time of various operation.

At very broad level following is the no of operation that this algorithm performs for various cases –

3.3.3b occupying an index
Best Case – $O(1)$
Worst Case - $O(\log_b n)$

3.3.3c Releasing an Index
Best Case – $O(1)$
Worst Case - $O(\log_b n)$

3.3.3d Finding an Free Index:
(Using sequential search for free bit in chunk)
Best Case – $O(\log_b n)$
Worst Case - $O(b \cdot \log_b n)$

3.4 System Implementation

We also implemented UIMR in Linux-based systems and evaluated it on our experimental setup. We first give the architectural details of this implementation, and then describe our experimentation setup. Finally, we present the experimental results.

Figure 3.6: UIMR hardware Architecture
Unique index management in routing

Figure 3.7: UIMR software Architecture

UIMR’s implementation and prototype: 3.6 shows UIMR is implemented across network and link layers as a loadable module of Linux 2.6 kernel. 3.7 UIMR software is then installed in T11FF routers and evaluated extensively in our experimental setup.

3.4.1 Implementation Details

We implemented UIMR in Linux-based systems & C datastructures with both Pentium-based devices (e.g., laptops) and StrongARM-based devices (e.g., Stargates and iPAQs) and Lucent IEEE 802.3,NIC, checked for $2^{15}$ indexes.

It is normally valuable for a MIB table that contains routes to be ordered such that a management application is able to query the table based on some attribute, without having to read every row in the MIB table. This requires that the rows in the table be ordered according to such attributes, and thus that those attributes be represented by objects included in the table’s INDEX clause. Examples of this can be seen in the ipCidrRouteTable and, more recently, the inetCidrRouteTable in.

While this useful feature results in an unusually large number of objects in the RouteTable’s INDEX clause, all are either integers or strings of 3 (or zero) octet length, so the resulting OIDs are not unusually large. (Specifically, the aggregate number of sub-identifiers to be appended to an OBJECT-TYPE’s OID, when naming an instance of an object in the RouteTable, is at most 22 sub-identifiers; i.e., less than the minimum number to be appended for the inetCidrRouteTable table.)
ROUTE-MIB DEFINITIONS ::= BEGIN
IMPORTS
MODULE-IDENTITY, OBJECT-TYPE,
Unsigned32, mib-2 FROM SNMPv2-SMI
MODULE-COMPLIANCE, OBJECT-GROUP FROM SNMPv2-CONF
RowStatus, TimeStamp,
StorageType FROM SNMPv2-TC
InterfaceIndex, InterfaceIndexOrZero FROM IF-MIB
fcmInstanceIdOrZero, fcmSwitchIndex,
FcAddressIdOrZero, FcDomainIdOrZero FROM FC-MGMT-MIB
T11FabricIndex FROM T11-TC-MIB

3.4.2 Experimental Setup

"The MIB module for configuring and displaying Route Information and we
have checked the details for t11FF router.

ALGORITHM: Route notification MIB module for configuring and displaying route
information.

Route Notifications OBJECT IDENTIFIER ::= \{ t11FFRouteMIB 0 \}
Route Objects OBJECT IDENTIFIER ::= \{ t11FFRouteMIB 1 \}
Route Conformance OBJECT IDENTIFIER ::= \{ t11FFrouteMIB 2 \}

-- Per-Fabric routing information

RouteFabricTable OBJECT-TYPE
SYNTAX SEQUENCE OF T11FcRouteFabricEntry
MAX-ACCESS not-accessible
STATUS current
DESCRIPTION
"The table containing Fibre Channel Routing information
that is specific to a Fabric."
Unique index management in routing

::= { RouteObjects 1 }

RouteFabricEntry OBJECT-TYPE
SYNTAX T11FcRouteFabricEntry
MAX-ACCESS not-accessible
STATUS current
DESCRIPTION
"Each entry contains routing information specific to a particular Fabric on a particular switch (identified by values of fcmInstanceIndex and fcmSwitchIndex)."
INDEX { fcmInstanceIndex, fcmSwitchIndex,
t11FcRouteFabricIndex }
::= { t11FcRouteFabricTable 1 }

RouteFabricEntry ::= 
SEQUENCE {
RouteFabricIndex T11FabricIndex,
RouteFabricLastChange TimeStamp
}

RouteFabricIndex OBJECT-TYPE
SYNTAX T11FabricIndex
MAX-ACCESS not-accessible
STATUS current
DESCRIPTION
"A unique index value that uniquely identifies a particular Fabric.
In a Fabric conformant to FC-SW-3, only a single Fabric can operate within a physical infrastructure, and thus the value of this Fabric Index will always be 1. In a Fabric conformant to FC-SW-4, multiple Virtual Fabrics can operate within one (or more) physical infrastructures. In such a case, index value is used to uniquely identify a particular Fabric within a physical infrastructure."
::= { RouteFabricEntry 1 }

RouteFabricLastChange OBJECT-TYPE
SYNTAX TimeStamp
Unique index management in routing

MAX-ACCESS read-only
STATUS current
DESCRIPTION
"The value of sysUpTime at the most recent time when any corresponding row in the t11FcRouteTable was created, modified, or deleted. A corresponding row in the t11FcRouteTable is for the same management instance, the same switch, and same Fabric as the row in this table.
If no change has occurred since the last restart of the management system, then value of this object is 0."
 ::= { RouteFabricEntry 2 }
T11FF Routing table
RouteTable OBJECT-TYPE
SYNTAX SEQUENCE OF RouteEntry
MAX-ACCESS not-accessible
STATUS current
DESCRIPTION
"The Routing tables for the locally managed switches. This table lists all the routes that are configured in and/or computed by any local switch for any Fabric.
Such routes are used by a switch to forward frames (of user data) on a Fabric. The conceptual process is based on extracting the Destination Fibre Channel Address Identifier (D_ID) out of a received frame (of user data) and comparing it to each entry of this table that is applicable to the given switch and Fabric. Such comparison consists of first performing a logical-AND of the extracted D_ID with a mask (the value of t11FcRouteDestMask) and second comparing the result of that 'AND' operation to the value of t11FcRouteDestAddrId. A similar comparison is made of the Source Fibre Channel Address Identifier (S_ID) of a frame against the t11FcRouteSrcAddrId and t11FcRouteSrcMask values of an entry. If an entry's value of t11FcRouteInInterface is non-zero, then a further comparison determines if the frame was received on the appropriate interface. If all of these comparisons for a particular entry are successful, then that entry represents a potential route for forwarding the received frame.
For entries configured by a user, t11FcRouteProto has the value 'netmgmt'; only entries of this type can be deleted by the user."
Unique index management in routing

ALGORITHM: Entry configuration by the user, t11FC routerproto with various n/w managements.

::= { t11FcRouteObjects 2 }
RouteEntry OBJECT-TYPE
SYNTAX  T11FcRouteEntry
MAX-ACCESS not-accessible
STATUS current
DESCRIPTION
"Each entry contains a route to a particular destination, possibly from a particular subset of source addresses, on a particular Fabric via a particular output interface and learned in a particular manner."

INDEX { fcmlInstanceIndex, fcmSwitchIndex,
RouteFabricIndex,
RouteDestAddrId, RouteDestMask,
RouteSrcAddrId, RouteSrcMask,
RouteInInterface, RouteProto,
RouteOutInterface }
::= { t11FcRouteTable 1 }
RouteEntry ::= SEQUENCE
{
RouteDestAddrId  FcAddressIdOrZero,
RouteDestMask  FcAddressIdOrZero,
RouteSrcAddrId  FcAddressIdOrZero,
RouteSrcMask  FcAddressIdOrZero,
RouteInInterface  InterfaceIndexOrZero,
RouteProto  INTEGER,
RouteOutInterface  InterfaceIndex,
RouteDomainId  FcDomainIdOrZero,
RouteMetric  Unsigned32,
RouteType  INTEGER,
RouteIfDown  INTEGER,
RouteStorageType  StorageType,
RouteRowStatus  RowStatus

Department of Computer Science, S.K University, Anantapur.
Unique index management in routing

RouteDestAddrld OBJECT-TYPE
SYNTAX FcAddressIdOrZero (SIZE (3))
MAX-ACCESS not-accessible
STATUS current
DESCRIPTION
"The destination Fibre Channel Address Identifier of this route. A zero-length string for this field is not allowed."
::= { t11FcRouteEntry 1 }

RouteDestMask OBJECT-TYPE
SYNTAX FcAddressIdOrZero
MAX-ACCESS not-accessible
STATUS current
DESCRIPTION
"The mask to be logical-ANDed with a destination Fibre Channel Address Identifier before it is compared to the value in the t11FcRouteDestAddrld field. Allowed values are 255.255.255, 255.255.0, or 255.0.0. FSPF's definition generates routes to a Domain_ID, so the mask for all FSPF-generated routes is 255.0.0. The zero-length value has the same meaning as 0.0.0."
::= { t11FcRouteEntry 2 }

RouteSrcAddrld OBJECT-TYPE
SYNTAX FcAddressIdOrZero
MAX-ACCESS not-accessible
STATUS current
DESCRIPTION
"The source Fibre Channel Address Identifier of this route. Note that if this object and the corresponding instance of t11FcRouteSrcMask both have a value of 0.0.0, then this route matches all source addresses. The zero-length value has the same meaning as 0.0.0."
::= { t11FcRouteEntry 3 }

Algorithm: Instance of t11FC router, SRC mask which match all source address.
::= { t11FcRouteEntry 3 }

Department of Computer Science, S.K University, Anantapur.
Unique index management in routing

RouteSrcMask OBJECT-TYPE
SYNTAX FcAddressIdOrZero
MAX-ACCESS not-accessible
STATUS current

DESCRIPTION
"The mask to be logical-ANDed with a source Fibre Channel Address Identifier before it is compared to the value in the t11FcRouteSrcAddrId field. Allowed values are 255.255.255, 255.255.0, 255.0.0, or 0.0.0. The zero-length value has the same meaning as 0.0.0."

ALGORITHM: Fiber channel interface with IfIndex under value object as non zero.
::= {t11FFRouteEntry 4}

RouteInInterface OBJECT-TYPE
SYNTAX InterfaceIndexOrZero
MAX-ACCESS not-accessible
STATUS current

DESCRIPTION
"If the value of this object is non-zero, it is the value of ifIndex that identifies the local Fibre Channel interface through which a frame must have been received in order to match with this entry. If the value of this object is zero, the matching does not require that the frame be received on any specific interface."

ALGORITHM: Algorithm for Route Learning Mechanism.
::= {t11FcRouteEntry 5}

RouteProto OBJECT-TYPE
SYNTAX INTEGER {
  other(1),
  local(2),
  netmgmt(3),
  fspf(4)
}
MAX-ACCESS not-accessible
STATUS current

DESCRIPTION
"The mechanism via which this route was learned:
other(1) - not specified
local(2) - local interface
netmgmt(3) - static route
fspf(4) - Fibre Shortest Path First"

ALGORITHM: Fiber channel interface that identify local value of IfIndex.
::= { t1IIfcRouteEntry 6 }
RouteOutInterface OBJECT-TYPE
SYNTAX InterfaceIndex
MAX-ACCESS not-accessible
STATUS current

DESCRIPTION
"The value of ifIndex that identifies the local
Fibre Channel interface through which the next hop
of this route is to be reached."

ALGORITHM: To identify he domain ID of the HOP.
::= { t1IIfcRouteEntry 7 }
RouteDomainId OBJECT-TYPE
SYNTAX FcDomainIdOrZero
MAX-ACCESS read-create
STATUS current

DESCRIPTION
"The domain ID of next hop switch.
This object can have a value of zero if the value
of t1IIfcRouteProto is 'local'."
Unique index management in routing

ALGORITHM: Routing metrics of the route which depend on t11FC.

::= { t11FcRouteEntry 8 }
RouteMetric OBJECT-TYPE
SYNTAX Unsigned32 (0..65536)
MAX-ACCESS read-create
STATUS current

DESCRIPTION
"The routing metric for this route.
The use of this object is dependent on t11FcRouteProto."

ALGORITHM: Identify the type of Router as local or remote with next fiber channel.

::= { t11FcRouteEntry 9 }
RouteType OBJECT-TYPE
SYNTAX INTEGER {
    local(1),
    remote(2)
}
MAX-ACCESS read-create
STATUS current

DESCRIPTION
"The type of route.
local(1) - a route for which the next Fibre Channel port is the final destination;
remote(2) - a route for which the next Fibre Channel port is not the final destination."

DEFVAL {local}
Unique index management in routing

ALGORITHM: Mapping of O/P interface and tl 1FC router out interface.
::= { t11FcRouteEntry 10 }

RoutelfDown OBJECT-TYPE
SYNTAX INTEGER {
    remove(1),
    retain(2)
}
MAX-ACCESS read-create
STATUS current

DESCRIPTION
"The value of this object indicates what happens to the route when the output interface (given by the corresponding value of t11FcRouteOutInterface) is operationally 'down'. If this object's value is 'retain', the route is to be retained in this table. If this object's value is 'remove', the route is to be removed from this table."
DEFVAL { retain }

ALGORITHM: Route storage type for the conceptual row.
::= { t11FcRouteEntry 11 }

RouteStorageType OBJECT-TYPE
SYNTAX StorageType
MAX-ACCESS read-create
STATUS current

DESCRIPTION
"The storage type for this conceptual row, Conceptual rows having the value 'permanent' need not allow write-access to any columnar objects in the row."
DEFVAL { nonVolatile }
::= { t11FcRouteEntry 12 }

Mapping of the OBJECT-GROUP macro

For conformance purposes, it is useful to define a collection of related managed objects. The OBJECT-GROUP macro is used to define each such collection of related objects. It should be noted that the expansion of the OBJECT-
GROUP macro is something which conceptually happens during implementation and not during run-time.

To "implement" an object, an agent must return a reasonably accurate value for management protocol retrieval operations; similarly, if the object is writable, then in response to a management protocol set operation, an agent must accordingly be able to reasonably influence the underlying managed entity. If an agent cannot implement an object, the management protocol provides for it to return an exception or error, e.g., noSuchObject [4]. Under no circumstances shall an agent return a value for objects which it does not implement - it must always return the appropriate exception or error, as described in the protocol specification [4].

Note that the OBJECT-GROUP macro itself provides no conformance information. Rather, conformance information is specified through the inclusion of defined groups in a MODULE-COMPLIANCE macro.

Mapping of the OBJECTS clause

The OBJECTS clause, which must be present, is used to specify each object contained in the conformance group. Each of the specified objects must be defined in the same information module as the OBJECT-GROUP macro appears, and must have a MAX-ACCESS clause value of "accessible-for-notify", "read-only", "read-write", or "read-create".

It is required that every object defined in an information module with a MAX-ACCESS clause other than "not-accessible" be contained in at least one object group. This avoids the common error of adding a new object to an information module and forgetting to add the new object to a group.

Mapping of the STATUS clause

The STATUS clause, which must be present, indicates whether this definition is current or historic.

The value "current" means that the definition is current and valid. The value "obsolete" means the definition is obsolete and the group should no longer be used for defining conformance. While the value
"deprecated" also indicates an obsolete definition, it permits new/continued use of conformance definitions using this group.

Mapping of the DESCRIPTION clause

The DESCRIPTION clause, which must be present, contains a textual definition of that group, along with a description of any relations to other groups. Note that generic compliance requirements should not be stated in this clause. However, implementation relationships between this group and other groups may be defined in this clause.

Mapping of the REFERENCE clause

The REFERENCE clause, which need not be present, contains a textual cross-reference to some other document, either another information module which defines a related assignment, or some other document which provides additional information relevant to this definition.

Mapping of the OBJECT-GROUP value

The value of an invocation of the OBJECT-GROUP macro is the name of the group, which is an OBJECT IDENTIFIER, an administratively assigned name.

3.4.3 Experimental Results

Using the above setup, we first show how effectively UIMR uses data storage and its accessing with its hybrid approach for measuring efficient ways in searching the data in the route. Then, we show that by using the data storage UIMR's unicast-based approach measures link quality more accurately than the existing bit, linked, rage lists approach. Finally, we show that UIMR's uni-directional link quality effectively identifies data, and improves the efficiency of utilizing the channel capacity over BAP's bi-directional link quality and we have also shown Comparison table for managing $2^{15}$ indexes with different techniques.
TABLE 3.1: Comparison Of Bit list with UIMR

<table>
<thead>
<tr>
<th></th>
<th>UIMR(this tech.)</th>
<th>Bit list</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory</td>
<td>4132 Bytes</td>
<td>4096 Bytes</td>
</tr>
<tr>
<td>Finding First Free Index</td>
<td>3 to 96 Comparison + d</td>
<td>1 to 1024 Comparison +d</td>
</tr>
<tr>
<td>Checking arbitrary Index Status</td>
<td>D</td>
<td>D</td>
</tr>
<tr>
<td>Reserving arbitrary Index</td>
<td>d to 3d</td>
<td>D</td>
</tr>
<tr>
<td>Freeing arbitrary Index</td>
<td>d to 3d</td>
<td>D</td>
</tr>
</tbody>
</table>

Table 3.1 and graphs specify parameter memory, finding first free index, checking arbitrary index status, reserving arbitrary and freeing arbitrary index, with respect to memory though bit list taking less with 36 bytes but large difference found in placing or retrieving data from router storage, where D=set/reset/check-bit-status, d= delay to calculate offset and index, x= delay to merge/split range list. (Refer appendix B)

TABLE 3.2: Comparison Of Link list with UIMR

<table>
<thead>
<tr>
<th></th>
<th>UIMR(this tech)</th>
<th>Link list</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory</td>
<td>4132 Bytes</td>
<td>0 – 262144 Bytes</td>
</tr>
<tr>
<td>Finding First Free Index</td>
<td>3 to 96 Comparison + d</td>
<td>1</td>
</tr>
<tr>
<td>Checking arbitrary Index Status</td>
<td>D</td>
<td>1 to $2^{15}$ Traversing and Comparison</td>
</tr>
<tr>
<td>Reserving arbitrary Index</td>
<td>d to 3d</td>
<td>1 (need manage duplicates)</td>
</tr>
<tr>
<td>Freeing arbitrary Index</td>
<td>d to 3d</td>
<td>1 (need manage duplicates)</td>
</tr>
<tr>
<td>Remark</td>
<td></td>
<td>What if we need least free index?</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For this link list need to managed in sorted order</td>
</tr>
</tbody>
</table>
Table 3.2 & the graph plots the UIMR technique with the linked list, with respect to the memory linked list ranges from 0 to 262144 bytes whereas memory requirement with UIMR is only 4132 bytes, which proves as remarkable. Finding, checking & freeing the index also proved to be good. In retrieving & freeing arbitrary index, though linked list takes only one byte but needs to manage duplicates which is purely avoided in UIMR.

<table>
<thead>
<tr>
<th>TABLE 3.3 : Comparison Of Range list with UIMR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory</td>
</tr>
<tr>
<td>Finding First Free Index</td>
</tr>
<tr>
<td>Checking arbitrary Index Status</td>
</tr>
<tr>
<td>Reserving arbitrary Index</td>
</tr>
<tr>
<td>Freeing arbitrary Index</td>
</tr>
<tr>
<td>Remark</td>
</tr>
<tr>
<td>UIMR(this tech)</td>
</tr>
<tr>
<td>4132 Bytes</td>
</tr>
<tr>
<td>3 to 96 Comparison + d</td>
</tr>
<tr>
<td>D</td>
</tr>
<tr>
<td>d to 3d</td>
</tr>
<tr>
<td>12 – 196608 bytes</td>
</tr>
<tr>
<td>Range list</td>
</tr>
<tr>
<td>12 – 196608 bytes</td>
</tr>
<tr>
<td>1 to 2&lt;sup&gt;14&lt;/sup&gt;</td>
</tr>
<tr>
<td>1 to 2&lt;sup&gt;14&lt;/sup&gt; + x + y</td>
</tr>
<tr>
<td>12 – 196608 bytes</td>
</tr>
</tbody>
</table>
Table 3.3 & the graph plots the UIMR technique with the linked list, with respect to the memory linked list ranges from 0 to 262144 bytes, whereas memory requirement with UIMR is only 4132 bytes, which proves as remarkable, & finding, checking & freeing the index also proved to be good. In retrieving & freeing arbitrary index, though linked list takes only one byte but needs to manage duplicates which is purely avoided in UIMR.

**TABLE3.4 Comparison table for managing $2^{15}$ indexes with different techniques**

<table>
<thead>
<tr>
<th></th>
<th>UIMR (this tech)</th>
<th>Bit list</th>
<th>Link list</th>
<th>Range List</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory</td>
<td>4132 Bytes</td>
<td>4096 Bytes</td>
<td>0 – 262144 Bytes</td>
<td>12 – 196608 bytes</td>
</tr>
<tr>
<td>Finding First Free Index</td>
<td>3 to 96 Comparison + d</td>
<td>1 to 1024 Comparison + d</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Checking arbitrary Index Status</td>
<td>D</td>
<td>D</td>
<td>1 to $2^{15}$ Traversing and Comparison</td>
<td>1 to $2^{14}$ Traversing and Comparison</td>
</tr>
<tr>
<td>Reserving arbitrary Index</td>
<td>d to 3d</td>
<td>D</td>
<td>1 (need manage duplicates)</td>
<td>$2^{14} + x + y$</td>
</tr>
</tbody>
</table>
Unique index management in routing

<table>
<thead>
<tr>
<th>Freeing arbitrary Index</th>
<th>d to 3d</th>
<th>D</th>
<th>1 (need manage duplicates)</th>
<th>1 to $2^D + x + y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Remark</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

What if we need least free index? For this link list need to managed in sorted order

d = delay to calculate offset and index + set/reset/check bit status
x = delay to merge/split range list.
Y = delay to allocate and insert a node

Department of Computer Science, S.K University, Anantapur.
Table 3.4 & the graph plots the UIMR technique with the linked list, with respect to the memory linked list ranges from 0 to 262144 bytes where as memory requirement with UIMR is only 4132 bytes, which proves as remarkable, & finding, checking & freeing the index also proved to be good. In retrieving & freeing arbitrary index, though linked list takes only one byte but needs to manage duplicates which is purely avoided in UIMR.

3.5 Performance Evaluation

We now present the evaluation results of the current implementation of UIMR. We first describe the effectiveness of UIMR planning & the present key experimental evaluation results. Finally we discuss some of the remaining issues associated with UIMR.

3.5.1. Effectiveness of the UIMR planning

We measured the effectiveness of UIMR in meeting varying requirements like bit, linked, range lists, especially for a large-scale indexes in routing network. We use a 11FF router and 50 nodes that initially is assigned to symmetric link capacity as shown in. While searching the information with the constraints in the routing table at different times (i.e., T1, ..., T5), we evaluate the improvement in memory requirement UIMR can make.

As shown in the tables, UIMR reconfigures a network router to meet different requirements. Before each route data to be stored or searched in the MIB, this algorithm describes technique to generate and manage unique integer indexes from a specified range of integers. Generated index can be used where a unique integer from a specified range need to be served as key to a particular record and when record is freed index need to be reused.

Efficiency of this algorithm lies in its simplicity and capability to manage large range of index in minimal recourses, such as running time and memory requirement.

This Algorithm is best suited for problems where 0 or 1-based unique indexes (however non 0 or 1 based indexes can also be managed with calculating a fix offset) are required to be managed with frequent operation like checking whether a index is free or not, finding first free index, reserving and freeing indexes with optimal memory usage in average case.
3.5.2. Evaluation Results

We evaluated the impact of the reconfiguration range. We use the same experiment settings as the previous one and focus on reconfiguration requests at T1. As we increase the index from each data, we measure the memory capacity improvement achieved by the reconfiguration plans. In addition, we calculate the improvement per change as the Cost-effectiveness of reconfiguration planning with different indexes.

The above tables plot the available memory capacity after reconfiguration over different indexes.

As shown in the figure, UIMR can improve the available bandwidth by increasing the reconfiguration range. However, its improvement becomes marginal as the range increases.

Because of the limited number of links. Furthermore, because reconfiguration plans with a larger range are required to incur more changes in network settings, the checking and reserving arbitrary index gain per change degrades significantly.

We evaluated how efficiently and accurately a monitoring method in UIMR measures network state information. We measure the active probing overhead of UIMR. At the same time, we measure memory requirement, finding first free index, checking arbitrary, reserving arbitrary, freeing indexes and compare it with bit, linked, range lists throughput to evaluate its probing accuracy. As shown in Figure, UIMR efficiently monitors the network state, while providing highly accurate link-quality information. The monitoring overhead of each node ranges from 0 to 215 indexes.

By contrast, UIMR can synchronize multiple indexes with a reasonable delay (0.8%) improving end-to-end throughput. (the upper figure).

3.6 Conclusion

We first discuss some of the remaining issues associated with UIMR and hardware forwarding resources in a router domain. and then make concluding remarks.
3.6.1 Remaining Issues

Joint optimization with farwarding resources and routing: UIMR decouples network reconfiguration from farwarding resources and routing. Farwarding in a right way might be able to achieve better performance if two problems are jointly considered. Even though there have been a couple of proposals to solve this problem [8, 89], they only provide theoretical bounds without considering practical system issues. Even though its design goal is to recover from network failures as a best-effort service, UIMR is the first step to solve this optimization problem, which we will address in future work.

Use of UIMR in IEEE 802.3: UIMR is mainly evaluated in IEEE 802.3 networks, where are available. However, UIMR can also be effective in a network with a large number of index. Because UIMR includes a link-association primitive, it can learn available channel indexes by associating with idle interfaces of neighboring nodes, and it further limits the range of a group.

3.6.2 Concluding Remarks

When compared to the bit list, link list and range list this is efficient & lies in its simplicity and capability to manage large range of index in minimal recourses, such as running time and memory requirement.

This Algorithm is best suited for problems where 0 or 1-based unique indexes (however non 0 or 1 based indexes can also be managed with calculating a fix offset) are required to be managed with frequent operation like checking whether a index is free or not, finding first free index, reserving and freeing indexes with optimal memory usage in average case.