

PATENT SPECIFICATION

Confidential

March 1999

PANASONIC SINGAPORE LABORATORIES PTE LTD

Ng Kok Leong and Tan Pek Yew

1. TITLE OF INVENTION

Method for allocating and managing unique identification numeral in a network environment.

2. WHAT IS CLAIM IS:

1. A method of allocating and managing whole number identification (ID) number, said method comprising the steps of:
 - (a) Maintaining an array table of flags to indicate whether the ID number is in use.
 - (b) Assigning the smallest available ID numbers as the highest priority to be allocated.
 - (c) Monitoring the number of ID number being released.
 - (d) Reusing the released ID numbers.
2. A method of reducing memory storage for the flags by employing the bits in each byte of the Array Table as flags for indicating whether the identification numbers are being used, as in Claim 1.
3. A method of associating identification number to a bit in an array table according to the position of that bit in the array table.
4. A scalable method of maintaining an array table as in Claim 1, comprising of the steps of:
 - (a) Monitoring the number of bytes being created in the array table.
 - (b) Creating a new byte to indicate that a new allocated identification number is in use if all the existing bytes in the scalable array table are all set.
 - (c) Releasing the last byte in the scalable array table if all the bits in that byte are all unset.
5. A method of accessing flags in the array table for each identification number by using the row and column indices that can be computed from that particular identification number and the size of a byte in the array table.
6. A method of supporting reusability of the ID number by using the table of flags to ensure unique identification number is being allocated.
7. A method of controlling the range of ID number being generated.
8. An apparatus for managing and controlling the range of ID number in a distributed environment, said apparatus comprising of:
 - (a) An Array Table Session Handler to act as an interface for the communication between network nodes and also to provide a communicative service for allocation and de-allocation of numerical range for identification number.

This Sheet Has Been Replaced





- (b) An Array Table Manager to manage and control a set of Array Table for various Protocol layers required at the Network Node.
 - (c) A Protocol stack service manager that maintain the interface and handle request between the Protocol stack layers and the Array Table.
9. An apparatus as mention in part (a) of Claim 8 further comprises of a Session Protocol Handler device. It initiates the exchange of communication primitives, which is used for initialization and management of numerical range of Array Table by communicating with other network nodes using a set of communication primitives.
10. A set of communication primitives handled by the apparatus mention in Claim 9 are used for allocating and de-allocating a specific numerical range. The communication primitives are used to set up an ID reservation session by encapsulating these communication primitives using protocol primitives supported by the host network node.
11. An apparatus as mention in part (a) of Claim 8 further comprises of a contention detection and resolution method. It is used to resolve contention in the numerical range of a similar indexed Array Table of 2 or more separate network nodes by exchanging communication primitives.
12. A method as mention in part (b) of Claim 8 further comprises of an Array Table Management method using Base Array Table to manage the elements of the Array Table within a distributed network environment.
13. A method as mentioned in part (a) of Claim 8 further comprises of method to manage the exchange communication primitives for reservation of numerical range for identification number and management of Array Table elements by using allocation and de-allocation state machines.
14. A method as mentioned in Claim 13 further comprises of method of detecting unused and reusing the numerical range for identification number used the parameters of the communication primitives used in ID reservation sessions.
15. A detection and contention ID range resolution method as mentioned in Claim 11 further comprising the steps of:
- (a) Checking all received broadcast ID reservation session communication primitives for duplications in Array Table element reservations.
 - (b) Sending an ID reservation session communication primitive to reject allocation of the Array Table element based on the current utilization of the Array Table elements if contentions were detected.

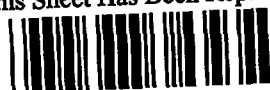
3. DETAIL DESCRIPTION OF THE INVENTION

3.1 Industrial Fields of Application

This invention is applicable to network session application, which requires session ID allocation, server ID allocation, client ID allocation, and transaction ID allocation; communication channel ID allocation and many other applications that require unique whole number for identification. This invention allows the distributed and central control and management of numeric ID on the network environment. The invention enhances scalability and efficiency of flag storage, search and assignment and reduces the probability of ID collision.

3.2 Description of Prior Art

Unique whole number identification numbers are required in many areas for identification. They can be used as session ID and transaction ID in network session controls and communication channel ID in communication controls.



A very simple way of generating the identification numbers is by using a counter only. In this implementation, the counter is incremented and assigned as the identification number whenever an identification number is requested. There is no management of the identification numbers. This implementation does not support reusability of the identification number which can easily run out and is vulnerable to conflicts in identification number assignment if the counter value were to be reused.

Another existing method of identification number generation is by using a counter and a list of identification numbers. The counter is used to generate the identification number and the list contains the identification numbers in use. Whenever there is a request for the identification number, the counter is incremented and its value is checked against the list of allocated identification numbers. If the counter value is found in the list, the counter is incremented again and its new value is checked. These steps are repeated until the new counter value is not found in the list so that this value can be assigned and then added to the list. This implementation provides reusability of the identification numbers but it incurs a tremendous amount of time in searching as well as a large storage memory especially when many identification numbers are being allocated. To enhance search in this method, background sorting of the identification numbers in the list is used. However, this enhancement merely transfers the processing from foreground to background, which does not actually lighten the processing load of the processor.

A third known method is by using a random number generator. This method has been employed in the in many communication protocol, for instance, in IETF's RTCP (real-time transport control protocol), the SSRC field of the RTCP communication primitives are generated using the MD5 routines published in RFC 1321[1]. This method also does not ensure that the identification number assigned is unique and there is a possibility of conflicts.

3.3 *Problems that the Invention is to Solve*

Most methods available either require large utilization of resources such as memories and computation for generating unique and reusable IDs or do not ensure that all IDs generated are unique. The present invention supports reusability of unique identification (ID) number, thereby reducing the probability of non-uniqueness of the identification number being allocated and running out of identification number. Besides, it provides an efficient use of memory space for the flags, which are used to indicate whether the identification number is being allocated, by assigning a bit for a flag and maintaining a scalable array table that can expand or contract in size depending on the largest identification number being allocated. In addition, it also provides an easy and efficient method of checking and accessing the flag by means of a look-up table in which the flag that is associated to a particular identification number, can be determined from that identification number. This reduces the processing time necessary for sorting and searching.

Most communication protocol primitives requires a unique numeral ID to mark the singleness of the message or a session the communication primitives seek to obtain for one or more network nodes. Having similar ID in the communication primitives at the same time causes collision. In many communication protocol, for example IETF's RTCP (real-time transport control protocol), a unique ID is used for the SSRC field of the RTCP communication primitives. The SSRC field is used to identifies the synchronization source. Collisions occur if protocol primitives with similar SSRC fields are used and detected by the sending network node. The occurrence of collision of SSRC fields increases as the number of messages in the network increases. To resolve the collisions, each network node capable of generating messages with SSRC fields needs to have collision detection and resolution capabilities. As a result of collision, network resources such as bandwidth and the computation resources of the network node will be wasted. The present invention provides methods to avoid such collision even in a congested network environment for each network node. These methods for collision avoidance can be applied to both distributed and centrally control and management of unique numeral ID.

3.4 *Means of Solving the Problem*

This Sheet Has Been Replaced



To resolve the problems outlined in section 3.3, the invented method provides a scalable array table of bits to indicate the allocated identification numbers; a means to associate the identification number to a bit in the table to act as flag so as to enhance flag checking and access; a means to reuse the released identification number by using the scalable array table of bits to ensure only unassigned identification number is allocated; a means of reserving identification in a distributed network node environment.

3.5 *Operation of the Invention*

This invention involves mapping the identification numbers to an array table of bits based on the bit positions as shown in Figure 1. Each bit acts as a flag to indicate whether the identification number is in use. The mapping function provides a simple and efficient access of the flag by computing the RowIndex and ColumnIndex using the identification number and the size of each byte in the array table. The identification number is allocated with the smallest available value first to ensure that all the lower bytes in the array table are set before new byte is created. This is to facilitate the implementation of a scalable array table for efficient memory usage. The availability of an identification number is either indicated by the unset bit (0) in the array table as shown in Figure 2 or there is no bit being assigned to it yet. An identification number is released by unsetting the bit corresponding to it and it is ready for reuse. To provide identification number collision avoidance to marked a unique session for a set of network nodes, a set of communication protocol primitives is used. This communication protocol primitives are exchange between network nodes for the reservation of numeral ID range such that ID collisions protocol layers are minimized.

3.6 *Embodiments*

The invention will be best understood by reading the following detailed description of an illustrative embodiment with reference to the accompanying drawings listed in Section 4. Figure 1 to Figure 8, in essence, also give an illustration of a central controlled identification number provided the management and control of array table as mentioned in Figure 1 is managed from a logical network node.

As shown in Figure 1, the identification number is generated from the RowIndex and ColumnIndex of the unset flag in the array table using the following relationship:

$$ID = RowIndex * ByteSize + ColumnIndex \quad (1)$$

where ID denotes the value of the specified identification number

ByteSize denotes the number of bits in each byte in the array table.

This invention supports the range of identification number from 0 to (ByteSize*NoOfRowAllocated-1). Note that NoOfRowAllocated denotes the number of rows that are allocated dynamically.

Figure 3 depicts the flow chart for the case when an identification number is being requested. This request will first lead to the check, marked by literal 3.1, for any released identification number available. This is done by means of a ReleasedID counter, which monitors the number of identification numbers being released. It is incremented whenever an identification number is released and decremented whenever a released identification number is being reallocated. A positive value of the ReleasedID counter will lead to literal 3.2 in which the array table will be searched byte by byte from the first byte of the array table, until the byte, which contains at least one unset bit (0) is first found. This byte will in turn be searched bit by bit from its first bit for the unset flag, which indicates the first unassigned identification number present. The first found unset flag, which corresponds to the smallest unallocated identification number, will be set to indicate that the identification number is assigned. The value of the identification number assigned is computed using the RowIndex and ColumnIndex of the location of the flag in the array table with the aid of Equation (1). On the other hand, if there is no released identification number available, the last byte in the array table will be checked for any unassigned identification number, marked by literal 3.3. If there are unassigned identification



numbers available, the smallest value determined using Equation (1) will be assigned and its corresponding flag will be set, literal 3.4. The actions in literal 3.5 will be performed if there is no unset bit in the last byte of the array table as depicted in Figure 4a. Note that all the bits in the array table are set (1) showing that there is no released and unassigned identification numbers. This will lead to the creation of a new byte in the array table to serve as the flag for the to-be assigned identification number. The updated array table with the lowest significant bit set is shown in Figure 4b and the value of the newly assigned identification number is determined using Equation (1).

The flow chart for the case when release of a specified identification number is requested is shown in Figure 5. As depicted in literal 5.1, the specified identification number is checked whether it is being assigned. The flag in the array table that indicates whether the specified identification number is assigned can be determined by the following relationships:

$$\text{RowIndex} = \text{ID} / \text{ByteSize} \quad (2)$$

$$\text{ColumnIndex} = \text{ID} \bmod \text{ByteSize} \quad (3)$$

where ID denotes the value of the specified identification number
 / denotes the integer division
 ByteSize denotes the number of bits in each byte in the array table
 mod denotes modulo

This Sheet Has Been Replaced



The flag is thus bit[RowIndex,ColumnIndex]. If this bit were not set or does not exist, the specified identification number not found error will be reported as marked in literal 5.2. If the specified identification number has been assigned, the bit[RowIndex,ColumnIndex] will be unset (0), literal 5.3. Next, the last byte in the array table is checked whether it is zero, as in literal 5.4. A zero value of last byte will be deleted as depicted in literal 5.5 and pictorially shown in Figure 6. The new last byte is again checked and literal 5.4 and literal 5.5 are repeated until a non-zero last byte is encountered.

Figure 7 shows the case when a desired identification number is being requested. Literal 7.1 depicts that the current size of the array table is used to determine whether it can support the desired identification number. This can be performed by using Equation (2) to obtain the RowIndex and comparing it with the number of rows (bytes) created in the array table. Note that the value of n-1 for RowIndex indicates that there are n rows required since the RowIndex begins with 0. Literal 7.2 handles the case when there are sufficient rows in the array table to support the desired identification number and the corresponding bit[RowIndex,ColumnIndex] is set. If the current array table could not support the desired identification number, new rows will be created in the array table with the correspond bit[RowIndex,ColumnIndex] set, as marked in literal 7.3. This is further illustrated in Figure 8, which depicts the situation in which two more bytes are necessary to be added into the array table in order to support the desired identification number. However, limitation in the number of bytes to be added to the array table can be easily imposed in the implementation by exiting the flow once Literal 7.1 fails or the number of bytes required to be added exceed certain value.

The ID numeral management and allocation can also be created in a distributed manner which means that each network node manages it's own set of array table individually in a network environment as illustrated in Figure 9. Figure 9 shows an environment where there are N number of network nodes connected to the network, as marked by literal 9.8, where each node manages it's own array table such that a unique numeral can be generated within the maximum range as defined for the array table. The unique numeral generated using the Array Table of the network node are used by the communication primitives of the Protocol Stack if required, as marked by literal 9.2, supported at the network node. This unique number is generated to prevent the same numeral begin generated by 2 or more individual network nodes. For allocation, initialization and deletion of any members of the Array Table, ID reservations communication primitives, as listed in table in Figure 10, issued to other network nodes connected to the network by the Session Protocol Handler, as marked by literal 9.7. Depending on the network types supported by the network node, the messages are exchange by encapsulating it into the appropriate network primitives supported by the protocol stack available on the network node. The selection of the protocol layer to be used for transmitting and receiving the ID reservation messages is determined and managed by the Protocol Stack Service manager. Communication for reservation

of ID is preformed by passing the ID reservation message as a data to the service access point of the Protocol Stack, as marked by literal 9.2. The Protocol Stack, 9.2, based on the service required will encapsulate the data message using the appropriate network layer. For each of the messages received and transmitted, the data parameters of the messages generated and received by the Session Protocol Handler are interpreted by the Array Table Session Handler.

The Array Table Session Handler decides if elements in the Array Table, as marked by literal 9.4, should be added or deleted. Each element in the Array Table represent one numerical space for the numeral ID to be generated. The decision for deletion or addition of Array Table elements is passed from the Array Session Handler to the Array Table Manager where the actual deletion or addition is being performed. The Array Table Manager also manages and updates the entries of the Base Address Table for each Array Tables within the network Node. Each Array Table are link to a Base Address table consisting of an array of numeral or it's representatives. Each of the numeral location in the Base Address table has three parameters, <START_ADDR, SIZE, ARRAY_TABLE_#>. The START_ADDR is used to give the offset of the Array Table to start generating the ID based on the specific location using the equation(4). The SIZE parameter gives the range of ID can be generated starting from the START_ADDR. The ARRAY_TABLE_# specifies the Array Table managed by the Array Table Manager. Figure 11 show how each elements of the Array Table are link to the Base address table. To reduce the memory utilization only the elements that begins with START_ADDR, as marked by literal 11.2, are link to the one elements of the Base Table as marked by literal 11.3. The rest of the elements within the range as indicated by the parameter SIZE are represented by a continuous stream of Array Table elements. The last of the array element of Array table that are associated with the Base Address Table entry are determined by the parameter, SIZE, as marked by literal 11.4.

$$ID = START_ADDR + RowIndex * ByteSize + ColumnIndex \quad (4)$$

Each ID reservation session is created to handle an Array Table as marked by literal 9.4. One ID reservation session only handles deletion or addition of elements in one Array Table only. The life span of each session is controlled by the Array Table Session Handler. Each session is created by requests initiated by the Array Table Manager, as marked by literal 9.3 where requests for items to be added to or deleted from the Array Table are made and maintained. For each Session initiated by the Array Table Manager, this session is managed by the Array Table Session Handler and it will control how the ID reservation communication primitive is send through the Session protocol handler. This allows the range of ID to be generated from the Array Table to be increased or decreased. The source for request to increase the elements in the Array table within the network node comes from either the Array Table Manager or the Protocol Stack Service Manager, as marked by literal 9.5. Protocol Stack Service Manager manages the ID generated from the Array Table and passed to the Protocol Stack. It also serves as an interface between the Array Table and the Protocol Stack. Each Array Table generates unique number for a single layer of protocol stack using the elements in the Array Table. Generated ID from the Array Table for each layer of the protocol stack is managed by the Protocol Stack Service Manager. If the new ID is required by the protocol stack, the Array Table Manager will either search for unused elements in the Array Table for each protocol stack layer to create a new ID or to request for more Array Table elements via Array Table Session Handler.

There are 2 stages where the request of Array Table Manager can allocate Array Table elements:-

- (1) Boot-up stage where the network node just joined the network. The request for elements in the Array Table originates from the Array Table manager.
- (2) Run-time stage when the Protocol Stack request for more ID because all elements of the Array Table are used.

Deletion of elements within the Array Table are initiated on 2 occasions :-

- (1) When an the network node's Array Table Session Handler detected that another node requires more elements in the Array Table. The unused area will be deleted from the Array Table of the protocol stack layer.

This Sheet Has Been Replaced



- (2) When there is contention detected by the protocol stack, the entire range of the Array table will be relocated to another base address.

Figure 12 illustrates the state machine in allocating elements for a single Array Table element. This state machine can be implemented in the Array Table Session Handler layer. When the state machine received a request in the form of **Alloc_Mem** signal, as marked by literal 12.2, for more Array Table elements to be reserved from the Protocol Stack Service Manager, 9.5, or the Array Table Manager, 9.3, the state machine transit to **REQ_ALLOC** state, as marked by literal 12.3. At this state, the Session Protocol Handler send out the message, **Broad_Alloc**, specifying in the parameters of the communication primitive the amount of elements required. After sending the **Broad_Alloc** message, the Array Table Session Handler waits for acknowledgements that may be received by listening to the service access point created in the protocol stack. At the **WAIT_ACK** state, as marked by literal 12.5, if the first **Free_DeAlloc**, as marked by literal 12.7 is received, the state machine will proceed to the **ALLOC_ARR** state, as marked by literal 12.8, that allows allocation of the specified Array Table to be conducted by the Array Table Manager. The Array Table elements allocation is performed based on the size of elements released indicated in the parameters of the **Free_DeAlloc** message received. If a specific time has elapse while in the **WAIT_ACK** state, which may indicates the network node using the same Array Table element is no longer in the network or no longer requires it, the state machine will transit to the **ALLOC_ARR** state. While still in the **WAIT_ACK** state, more than one **Reject_Alloc** messages can be received and none **Free_DeAlloc** message as marked by literal 12.6. Under this circumstance, the state machine transit back to **REQ_ALLOC** state to allocate another range of memory required using different parameter for the **Broad_Alloc** message. After successful allocation of Array Table elements, the state machine goes into the **IDLE** state as indicated by literal 12.1 from **ALLOC_ARR** state.

Figure 13 illustrates the delocation of the Array Table element at run time and boot time. Upon detection of a **Broad_Alloc**, as marked by literal 13.2, the state machine transit to **CONT_DET** stage, as marked by literal 13.3. In the **CONT_DET** state, the parameters of the **Broad_Alloc**, are checked if the Array Table elements requested contend with current used Array Elements. If Array Table Elements request contention with all the elements used, the **Reject_Alloc** message is sent back to the sender and the state machine transit back to the **IDLE** state, as marked by literal 13.1. If some or all of the requested Array Table elements are unused, it marked no contention is detected and gives the **No_Content** signal, as marked by literal 13.5. The state machine transit to **DE_ALLOC** state, as marked by literal 13.5. In the **DE_ALLOC** state, the Array Table Manager frees the unused Array Table elements. The size of freed Array Table elements are reflected in the parameters of the **Free_DeAlloc** message, as marked by literal 13.7, send back to the Network node that request for the release of the Array Table elements. After sucessful deallocation of the message, the state machine transit back to **IDLE** state from **DE_ALLOC** state.

This Sheet Has Been Replaced



4. DESCRIPTION OF DRAWINGS

Figure 1 Mapping of identification numbers and the array table of bits.

This figure depicts how the identification numbers are associated to the array table of bits. It assumes that the current size of each byte in the table is 8 bits. In fact, it can be used with any byte size greater than zero and the greater the byte size, the greater the memory saving.

Figure 2 Example of a scalable array table with some identification numbers being released.

This figure depicts the scalable array table with five released identification numbers and the biggest three identification numbers not allocated. '0' indicates released or unassigned identification number whereas '1' indicates that the identification number is assigned.

Figure 3 Flow chart for the case when an identification number is being requested.

Figure 4 Creation of a new byte in the array table when an identification number is requested.

Figure 4a depicts the situation before the identification number is allocated whereas Figure 4b depicts the updated array table after the identification number has been assigned.

Figure 5 Flow chart for the case when an identification number is to be released.

Figure 6 Deletion of the last byte in the array table when an identification number is released.

Figure 6a depicts the situation before the identification number is released whereas Figure 6b depicts the situation when the corresponding bit is unset and Figure 6c depicts the final updated array table after the identification number has been released.

Figure 7 Flow chart for the case when a desired identification number is requested.

Figure 8 Creation of new bytes in the array table when an identification number is desired. Figure 8a depicts the situation before the identification number of 35 is allocated whereas Figure 8b depicts the updated array table after the identification number of 35 has been assigned.

Figure 9 shows a schematic view of a distributed management of Array table network node in a network environment.

Figure 10 shows a list of ID reservation communicative primitives used for reservation of numeral range.

Figure 11 shows an Array Table with entires from the Base Address Table. Is is used to generate a numeral ID that by the Array Table of the communication node.

Figure 12 shows the ID reservation session engaged in allocation of elements of a specified Array Table. The diagram is the state transistion diagram for allocating Array Table elements.

Figure 13 shows the ID reservation session engaged in the dellocation of elements of a specified Array Table. The diagram is a state transistion diagram for dellocating Array Table elements.

This Sheet Has Been Replaced

