Slot Allocation Using Logical Networks for TDM Virtual-Circuit Configuration for Network-on-Chip

Zhonghai Lu and Axel Jantsch
Department of Electronic, Computer and Software Systems, Royal Institute of Technology, Sweden
Email:{zhonghai,axel}@kth.se

***Abstract*— Configuring Time-Division-Multiplexing (TDM) Virtual Circuits (VCs) for network-on-chip must *guarantee conflict freedom* for overlapping VCs besides *allocating sufficient time slots* to them. These requirements are fulfilled in the slot allocation phase. In the paper, we define the concept of a logical network (LN). Based on this concept, we develop and prove theorems that constitute sufficient and necessary conditions to establish conflict-free VCs. Using these theorems, slot allocation for VCs becomes a procedure of computing LNs and then assigning VCs to different LNs. TDM VC configuration can thus be predictable and correct-by-construction. We have integrated this slot allocation method into our multi-node VC configuration program and applied the program to an industrial application.**

I. Introduction

In Network-on-Chip (NoC), routing packets may bring about unpredictable performance due to contention for shared links and buffers. To overcome the nondeterminism, researchers proposed various *resource reservation* and *priority-based scheduling* mechanisms to achieve Quality of Service (QoS), i.e., to provide guarantees in latency and bandwidth. The Æthereal [1] and Nostrum [2] NoCs establish *Time-Division-Multiplexing (TDM) virtual circuits (VCs)* to offer guaranteed services. The Æthereal VC, which is developed for a network using buffered flow control, is *open-ended*. The Nostrum VC, which is designed for a network employing bufferless flow control, is *closed-loop*. Both networks operate synchronously. The Mango [3] NoC realizes guarantees in an asynchronous (clockless) network by reserving virtual channels for end-to-end connections and using priority-based scheduling in favor of connections in switches. Alternatively, QoS may be achieved through traffic classification in combination with a differentiated service. For example, the QNoC [4] characterizes traffic into four priority classes, and switches make priority-based switching decisions.

VC is a connection-oriented technique in which a deterministic path must be established and associated resources are pre-allocated before packet delivery can start. A TDM VC means that each node along the path configures a time-sliced routing table to reserve time slots for input packets to use output links. This reservation is accomplished in the connection setup phase. In this way, VCs multiplex link bandwidth in a time division fashion. As long as a VC is established, packets sent over it, called *VC packets*, encounter no contention and thus have guarantees in latency and bandwidth. In a network delivering both Best-Effort (BE) and guaranteed-service traffic, BE packets utilize resources that are not reserved by VCs. Configuring VCs involves (1) *path selection*: This has to explore the network path diversity. As a VC has a number of alternative paths, configuring a set of VCs involves an extremely large design space. The space increases exponentially with the number of VCs; (2) *slot allocation*: Since VC packets must not contend with each other, VCs must be configured so that an output link of a switch is allocated to one VC per time slot, i.e., VCs are contention free. In addition, they must be equipped with sufficient slots, thus sufficient bandwidth.

In the paper, we address the TDM VC configuration with focus on the slot allocation problem. Current approaches to this problem ([5], [6], [7], [8]) are somewhat ad hoc. The slot allocation problem has been treated as a purely scheduling problem for which a complicated scheduling method is designed. Such methods locally schedule available slots to a set of sorted VCs one by one. The scheduling method guarantees the exclusive use of slots and sufficient slots. While such approaches are intuitive, they lack formal underpinning on the contention analysis and avoidance. As a result, the scheduling is non-trivial and can be an error-prone process. In contrast, we have furthered the investigations by looking into the fundamental reason of contention. We resort to a formal approach by defining the concept of a *Logical Network (LN)* and developing theorems to guide the construction of conflict-free and bandwidth-satisfied VCs. Based on these theorems, LNs can be formally partitioned and constructed, and slot allocation is a well-controlled process of VC-to-LN assignment, i.e., assigning VCs to different LNs.

The rest of the paper is organized as follows. We outline the related work in Section II. In Section III, we describe the two types of on-chip TDM VCs, namely, *open-ended* and *close-looped* VCs. Using LNs to construct contention-free and bandwidth-satisfied VCs is exemplified in Section IV. Then we present formal underpinning for the LN-based slot allocation in Section V. In Section VI, we detail how to perform slot allocation via VC-to-LN assignment. An industrial case study is reported in Section VII. Finally we conclude the paper in Section VIII.

II. Related Work

As mentioned previously, proposals dealing with the slot allocation problem can be found in [5], [6], [7] and [8]. In [5], the traffic model assumes periodic messages and all message flows have the same period. The scheduling algorithm for slot allocation must guarantee that latency and bandwidth requirements are fulfilled. In case a solution is not found, non-minimal VC paths are explored. This method is integrated into a framework unifying IP-to-node mapping, path selection and slot allocation in [6]. In [8], the scheduling method is strengthened by considering slot sharing and using the estimated knowledge of possible contentions while allocating slots to VCs. Besides, to use flexible routing in a network, messages within a flow are scheduled individually and may use different routes. Consequently, the message scheduling is complicated because it has also to ensure the correct message ordering. In [7], dynamic slot-allocation methods are presented to dynamically perform both routing and allocation of slots at run-time to establish guaranteed connections.

These approaches above only derive *sufficient but not necessary* configurations because they lack formal analysis on the contentions and their avoidance. In our approach, we formally derive and proof the *sufficient and necessary* conditions for conflict analysis and avoidance. Using these theorems, the slot allocation can be conducted predictably and in well-defined steps.

1-4244-1382-6/07/$25.00 ©2007 IEEE

The core concept for our slot allocation method is LN, which generalizes the concepts of *admission class* [10] and *Temporally Disjoint Network (TDN)* [2]. As with LNs, packets belonging to different classes or TDNs do not collide with each other. Admission classes and TDNs are essentially LNs. Comparing with an admission class, a LN takes not only time slots but also the VC path into consideration, thus the VC path-overlapping scenarios can be studied. Comparing with a TDN, a LN is locally defined for a group of overlapping VCs, which can be open-ended or closed-loop. A TDN can be viewed as a special case of a LN in the closed-loop VC when it is globally set up for all overlapping and non-overlapping VCs.

III. TDM-based Virtual Circuits in NoCs

A. Open-ended VCs

The on-chip TDM VCs assume that the network is packet-switched, and nodes share the same notion of time. They have the same clock frequency but not phase [9]. The time unit is slot. Since VC packets encounter no contention, they synchronously advance one step per time slot and never stall, using consecutive slots in consecutive switches. A node must configure a *routing table for VC packets* such that no simultaneous use of shared resources is possible. The routing table, by configuration, knows the *time slot* when a VC packet reaches which inport, and *addressing information* about which outport to use. In effect, the routing function partitions the link bandwidth and avoids contention.

Figure 1 shows two VCs, v_1 and v_2, and the respective routing tables for the switches. The output links of a switch are connected to a buffer or register. A routing table (t, in, out) is equivalent to a *routing* or *slot-allocation* function $\mathcal{R}(t, in) = out$, where t is time slot, *in* an inport, and *out* an outport. v_1 passes switches sw_1 and sw_2 through buffers $\{b_1 \to b_2\}$; v_2 passes switches sw_3 and sw_2 through $\{b_3 \to b_2\}$. The Æthereal NoC [1] proposes this type of VC for QoS. As the path of such a VC is not a loop, we call it open-ended.

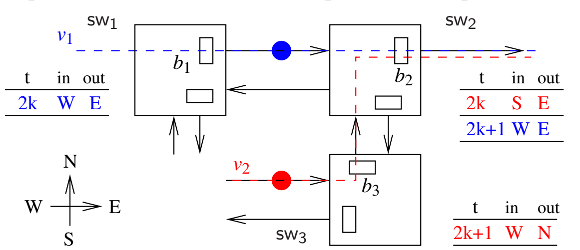


Fig. 1. Open-ended virtual circuits

In open-ended VCs, packets may be partitioned into *target classes* to avoid contention. With respect to a buffer b, a target class is the set of packets that will occupy slot d in a slot window D. This set of packets may come from any network node as long as they will take slot d in a slot window D of buffer b. As a target class owns dedicated slots of buffers, packets of different classes do not collide. A target class has a reference buffer whereas an admission class [10] does not. It can be viewed as a special case of the admission class. The union of all the target classes for all buffers in the network gives the corresponding admission class. By globally orchestrating the packet admission, contention can be avoided for packets belonging to different VCs. As illustrated in Figure 1, v_1 and v_2 only overlap in b_2, denoted $v_1 \cap v_2 = \{b_2\}$. v_1 packets are admitted on even slots of b_1. In sw_1, $(2k, W, E)$ means that sw_1 reserves its E (East) output link at slots $2k$ ($k \in \mathbb{N}$) for its W (West) inport ($\mathcal{R}(2k, W) = E$). As we can also see, v_2 packets are admitted on odd slots $2k+1$ of b_3, and sw_3 configures its odd slots for v_2. Since a v_1 packet reaches sw_2 one slot after reaching sw_1, sw_2 assigns its odd slots to v_1. Similarly, sw_2 allocates its even slots to v_2. As v_1 and v_2 alternately use the shared buffer b_2 and its associated output link, v_1 and v_2 do not conflict.

B. Container-based Closed-loop VCs

The Nostrum NoC [2] also suggests a TDM VC for QoS. However, a Nostrum VC has a cyclic path, i.e., a closed loop. On the loop, at least one *container* is rotated. A container is a *special packet* used to carry data packets, like a vehicle carrying passengers. The reason to have a loop is due to the fact that Nostrum uses deflection routing [10] whereas switches have no buffer queues. An incoming packet is either sunk or has to be switched out occupying one outgoing link. Since all outgoing links of a switch might be occupied by all incoming packets, a looped container ensures that there is an output link available for locally admitting a VC packet into the container, thus the network. VC packets are loaded into the container from a source, and copied (for multicast) or unloaded at the destination, by-passing other switches. Similarly to open-ended VCs, containers as VC packet carriers have higher priority than BE packets and do not contend with each other.

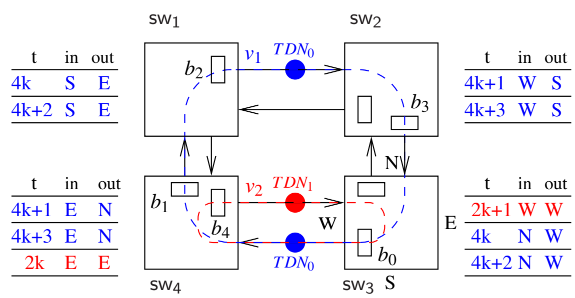


Fig. 2. Closed-loop virtual circuits

The Nostrum VC [2] uses TDNs to ensure conflict freedom. In [2], TDNs are descriptively rather than formally defined. TDNs are independent of VC paths. They are globally set up in a network. The number of TDNs depends on the network topology and the buffer stages in the switches [2]. For example, as shown in Figure 2, in a mesh network with one buffer per outport in the switches, exactly two TDNs exist, TDN_0 and TDN_1. To allow more TDNs, more buffers in the switches must be used. For example, placing two buffers in the switches, one at the inport, the other at the outport, results in four TDNs. In Figure 2, two VCs, v_1 and v_2, are configured. v_1 loops on sw_3, sw_4, sw_1 and sw_2 through $\{b_0 \to b_1 \to b_2 \to b_3 \to b_0\}$; v_2 loops on sw_3 and sw_4 through $\{b_0 \to b_4 \to b_0\}$; and $v_1 \cap v_2 = \{b_0\}$. v_1 and v_2 subscribe to TDN_0 and TDN_1, respectively. Besides, v_1 launches two containers and v_2 one container. The resulting routing tables for switches are also shown in Figure 2. Since TDNs are temporally disjoint, overlapping VCs allocated on different TDNs are free from conflict.

IV. Slot Allocation Using LNs

A. An Overview of Slot Allocation in a VC Configuration Flow

Figure 3 sketches a VC configuration flow. The input to the flow is a VC specification set. A VC allows having multiple sources and destinations (multi-node VC, see examples in Section VII.B). The output is a set of TDM VC implementations, which can be either open-ended or closed-loop. The configuration consists of *path selection* and *slot allocation*. Both problems are interdependent. They are also orthogonal. The VC configuration is likely iterative until a

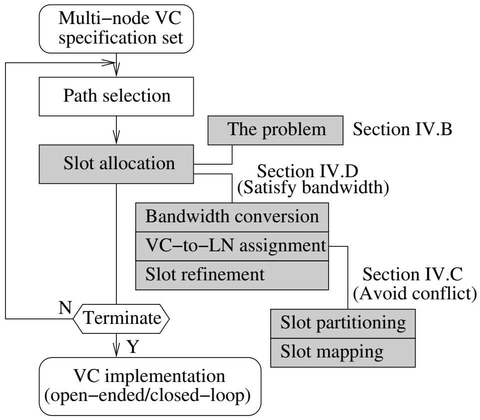


Fig. 3. Slot allocation in a VC configuration flow

termination condition is met. In this paper, we only briefly introduce our path selection method in Section VII-A. Assuming that the path selection is done, we focus on the *slot allocation* problem.

In the following of this section, we first formulate the slot allocation problem in Section IV-B. Then we illustrate the concept of the LN by exemplifying the LN construction for *conflict freedom* in Section IV-C. Then we show how to *satisfy bandwidth* demand using LNs in Section IV-D. We shall see that our method is applicable to both *open-ended* and *closed-loop* VCs.

B. The Slot-Allocation Problem Formulation

We first introduce definitions, and then define the problem.

Definition 1: A network is a directed graph $\mathcal{G} = M \times E$, where each vertex $m_i \in M$ represents a node, and each edge $e_i \in E$ represents a link. All edges are unique.

Definition 2: A *VC specification set after path selection* $\bar{V}$ comprises a set of VCs to be configured on the network $\mathcal{G}$, $\bar{V} = \{\bar{v}_1, \bar{v}_2, \cdots, \bar{v}_n\}$. For each VC $\bar{v}_i \in \bar{V}$, $\bar{v}_i = (m_i, \bar{bw}_i)$, where:

- $m_i \subseteq M$: a subset of nodes in M to be visited by $\bar{v}_i$. The node set is ordered and two consecutive nodes in m_i are adjacent in the network.
- $\bar{bw}_i$: minimum bandwidth requirement (bits/second) of $\bar{v}_i$.

Definition 3: A *VC implementation set* V comprises a set of TDM VC, $V = \{v_1, v_2, \cdots, v_n\}$. Each VC implementation $v_i \in V$ implements $\bar{v}_i$, and $v_i = (bw_i,\ R_{i,j})$, where:

- bw_i: the supported bandwidth (bits/second) of v_i.
- $R_{i,j}$: a partial routing table created for a visiting node n_j by v_i. $\forall r_z \in R_{i,j}$, r_z is an entry $(t, e_{in,x}, e_{out,y})$, specifying that node n_j reserves slot t for a v_i packet from input link $e_{in,x}$ to use output link $e_{out,y}$. R_j is the routing table of n_j, and $R_j = \sum_i R_{i,j}$.

Definition 4: At node n_j, a *slot-allocation* function $\mathcal{R}_j$: $(\mathcal{T}, E_{in,j}) \rightarrow E_{out,j}$ reserves slot $t \in \mathcal{T}$ for a VC packet from input edge $e_{in,j} \in E_{in,j}$ to use output edge $e_{out,j} \in E_{out,j}$.

Using the definitions above, we formulate the problem as follows: *Given* a network $\mathcal{G}$ and a VC specification set $\bar{V}$, *find* a VC implementation set V and *determine* for V a *slot-allocation function* $\mathcal{R}_j()$ for each node n_j, such that

$$\forall e_{in,x} \neq e_{in,y}, \mathcal{R}_j(t, e_{in,x}) \neq \mathcal{R}_j(t, e_{in,y}) \tag{1}$$

$$\bar{bw}_i \leq bw_i \tag{2}$$

$$\forall\ edge\ e_k,\ Bw(e_k) \leq \kappa_{bw}(e_k) \tag{3}$$
$$where\ Bw(e_k) = \textstyle\sum_i bw_i\ \ if\ e_k \in Edge(v_i)$$

Condition (1) says that VC packets can not be switched to the same output link simultaneously, i.e., VCs must be conflict free. Condition (2) expresses that each VC's bandwidth constraint must be satisfied. Condition (3) means that the total normalized (with the link capacity) bandwidth reserved by all VCs on a link cannot exceed the link bandwidth threshold κ_{bw}, which is defined in terms of the link capacity and $0 \leq \kappa_{bw} \leq 1$.

C. Conflict Avoidance with LNs

To convey the basic ideas of a LN before delving into formalism, we describe how conflict can be avoided between overlapping VCs by alternatively scheduling VCs on the use of the shared buffer(s). As we develop further, we shall see that LNs are the natural result of systematically avoiding collision between overlapping VCs. To be specific, when two VCs overlap, the conflict avoidance is assured through two steps: *slot partitioning* and *slot mapping*. These two steps create LNs and complete assigning VCs to LNs. We describe the two steps with a pair of *closed-loop VCs* (v_1, v_2) in Figure 2.

1) *Slot partitioning*: As conflicts might occur in a shared buffer, we partition the slots of the shared buffer into sets with a regular interval. In Figure 2, b_0 is the only shared buffer of v_1 and v_2, $v_1 \cap v_2 = \{b_0\}$. We partition the slots of b_0 (b_0 is called the *reference buffer* for v_1 and v_2, $Ref(v_1, v_2) = b_0$.) into two sets, an even set $s_0^2(b_0)$ for $t = 2k$ and an odd set $s_1^2(b_0)$ for $t = 2k+1$. The notation $s_\tau^T(b_0)$ represents pairs $(\tau + kT, b_0)$, which is the τth slot set of the total T slot sets, $\tau \in [0, T)$ and $T \in \mathbb{N}$. Pair (t, b_0) refers to the slot of b_0 at time instant t.

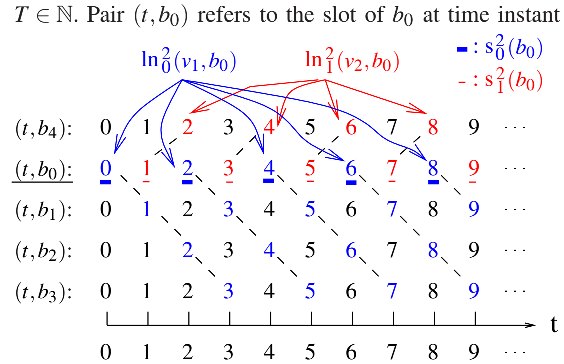


Fig. 4. Creating LNs by mapping slots on VCs

2) *Slot mapping*: The partitioned slot sets can be mapped to slot sets of other buffers on a VC regularly and unambiguously because a VC packet or container advances one step each and every slot. For example, a v_1 packet holding slot t at buffer b_0, i.e., pair (t, b_0), will consecutively take slot $t+1$ at b_1 (pair $(t+1, b_1)$), slot $t+2$ at b_2 (pair $(t+2, b_2)$), and slot $t+3$ at b_3 (pair $(t+3, b_3)$). After mapping the slot set $s_0^2(b_0)$ on v_1 and $s_1^2(b_0)$ on v_2, we obtain two slot sets $\{s_0^2(b_0), s_1^2(b_1), s_0^2(b_2), s_1^2(b_3)\}$ and $\{s_1^2(b_0), s_0^2(b_4)\}$. We refer to the logically networked slot sets in a set of buffers of a VC as a *LN*. We denote the two LNs as $ln_0^2(v_1, b_0)$ and $ln_1^2(v_2, b_0)$, respectively. $ln_0^2(v_1, b_0) = \{s_0^2(b_0), s_1^2(b_1), s_0^2(b_2), s_1^2(b_3)\}$ and $ln_1^2(v_2, b_0) = \{s_1^2(b_0), s_0^2(b_4)\}$. Let T be the number of LNs, the notation $ln_\tau^T(v, b)$ represents the τth LN of the total T LNs on v with respect to b. We illustrate the mapped slot sets for $s_0^2(b_0)$ and $s_1^2(b_0)$ and the resulting LNs in Figure 4. We can also see that LNs are the result of VC assignment to slot sets, specifically, v_1 to $ln_0^2(v_1, b_0)$ and v_2 to $ln_1^2(v_2, b_0)$.

As $ln_0^2(v_1, b_0) \cap ln_1^2(v_2, b_0) = \emptyset$, v_1 and v_2 are conflict free, as we shall show formally in Section V.

D. Bandwidth Satisfaction with LNs

In addition to be contention free, VCs must satisfy their bandwidth requirements. This is achieved in three steps: *bandwidth conversion*, *VC-to-LN assignment* and *slot refinement*. We exemplify the three steps with Figure 5 that shows three *open-ended VCs*, v_1, v_2 and v_3. The buffer set of VCs is listed in Table 5. As can be seen, $v_1 \cap v_2 = \{b_1\}$, $v_1 \cap v_3 = \{b_2, b_3\}$ and $v_2 \cap v_3 = \emptyset$.

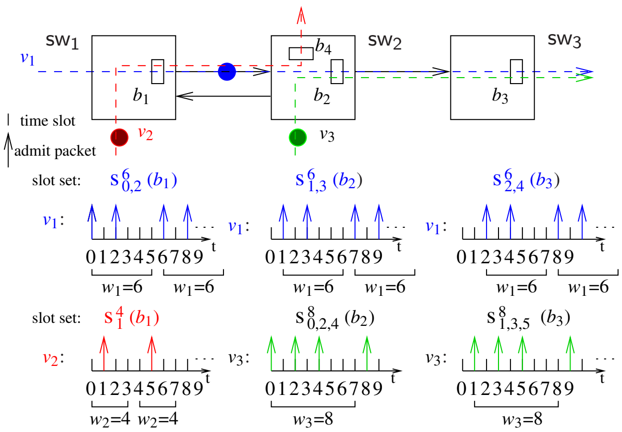


Fig. 5. Packets admitted on slot sets of buffers, i.e., on LNs

VC	*Buf. set*	*bw*	*N*	*W*	LN	Slot set
v_1	b_1, b_2, b_3	1/3	2	6	$ln_0^2(v_1, b_1)$	$\{s_{0,2}^6(b_1),\ s_{1,3}^6(b_2),\ s_{2,4}^6(b_3)\}$
v_2	b_1, b_4	1/4	1	4	$ln_1^2(v_2, b_1)$	$\{s_1^4(b_1),\ s_0^4(b_4)\}$
v_3	b_2, b_3	3/8	3	8	$ln_0^2(v_3, b_2)$	$\{s_{0,2,4}^8(b_2),\ s_{1,3,5}^8(b_3)\}$

TABLE I

VC PARAMETERS AND VC-TO-LN ASSIGNMENT RESULTS FOR FIG. 5

1) *Bandwidth conversion*: We first translate the VC bandwidth requirement in bits/second into packets/slot. As bandwidth is an average measurement, we can further scale it to N packets per W slots. W is the window size. For example, we translate $bw_1 = 1/3$ into $2/6$ (2 packets every 6 slots), i.e., $N_1 = 2$, $W_1 = 6$, as listed in Table I, where the bandwidth bw metric is packets/slot.
2) *VC-to-LN assignment*: In this step, we assign VCs to LNs pair-wise using the two steps for *conflict avoidance* in Section IV-C. Additionally we must check whether their bandwidth demand can be satisfied. This check is conducted after the first step *slot partitioning*. Given a pair of overlapping VCs, the number T of partitioned sets with respect to the reference buffer equals the number of LNs. To satisfy the bandwidth requirement of a VC v, a sufficient number N_{ln} of LNs must be allocated to v. This number can be derived from $N_{ln} = \lceil NT/W \rceil$[1], because we must satisfy $N_{ln}/T \geq N/W$, where N_{ln}/T is the bandwidth supported by the allocated LNs and N/W the requested bandwidth. The bandwidth requirements of the three VCs in Figure 5 are given in column bw of Table I.
We first perform the VC-to-LN assignment with VC pair (v_1, v_2). Since $v_1 \cap v_2 = \{b_1\}$, $Ref(v_1, v_2) = b_1$. Let $T = 2$, we partition b_1's slots into odd and even sets, implying two LNs. Either VC can be allocated to one LN, i.e., $N_{ln,1} = N_{ln,2} = 1$, offering bandwidth $N_{ln,1}/T = N_{ln,2}/T = 1/2$. Since the bandwidth demand of v_1 and v_2 is less then $1/2$, the resulting VC-to-LN assignment will meet the bandwidth constraint. Then we can continue to map the even set on v_1 and the odd set on v_2, obtaining the even LN $ln_0^2(v_1, b_1)$ for v_1 and the odd LN $ln_1^2(v_2, b_1)$ for v_2. Since $ln_0^2(v_1, b_1) \cap ln_1^2(v_2, b_1) = \emptyset$, v_1 and v_2 are conflict free.
Next, we perform the VC-to-LN assignment with (v_1, v_3). Let their reference buffer be b_2, $Ref(v_1, v_3) = b_2$. Since v_1 already holds even slots in b_1, it takes odd slots in b_2, i.e., $s_1^2(b_2)$. We assign the remaining even slots in b_2, i.e., $s_0^2(b_2)$, to v_3. Therefore, $N_{ln,3}/T = 1/2 > 3/8$. We are certain that the supported bandwidth suffices the demand of v_3. We map the slot set $s_0^2(b_2)$ on v_3, obtaining $ln_0^2(v_3, b_2)$. As $ln_0^2(v_1, b_1) \cap ln_0^2(v_3, b_2) = \emptyset$, v_1 and v_3 are also conflict free. The VC-to-LN assignments are shown in column LN of Table I.
3) *Slot refinement*: The success of VC-to-LN assignment for all VCs means that all VCs are conflict free and enough bandwidth can be reserved. But, a VC may demand only a fraction of slot sets from its assigned LNs. For instance, "v_2 on $ln_1^2(v_2, b_1)$" means that v_2 can use one of every two slots. But $N_2 = 1$ and $W_2 = 4$, v_2 actually demands only one out of four slots. This means that we need to further refine the supplied bandwidth. We first find the candidate slot sets of a reference buffer and then only assign N of them within window size W to v. For example, v_3 has four candidate slot sets over b_2, $s_{0,2,4,6}^8(b_2)$. We allocate any three of the four to v_3, for instance, $s_{0,2,4}^8(b_2)$. These slot sets are mapped to $s_{1,3,5}^8(b_3)$, forming the LN $ln_0^2(v_3, b_2)$. The slot sets reserved by the three VCs are illustrated in Figure 5 and listed in column Slot set of Table I. Note that the two columns LN and Slot set of Table I are equivalent.

After the three steps above, the VCs are constructed without conflict and with bandwidth requirements satisfied. In the following, we consider the *Slot refinement* as part of step *VC-to-LN assignment* to make the presentation concise.

E. Requirements for LN-oriented Slot Allocation

We have described so far three techniques: (1) establishing VCs by configuring slot-sliced routing tables; (2) partitioning and mapping slots into LNs; (3) assigning VCs to different LNs. These techniques must promise conflict freedom and provide enough bandwidth. However, there are several key questions that are not yet addressed:

- How many LNs exist when VCs overlap? LN is not global for all VCs. Instead it is local for a group of overlapping VCs. This number is crucial because it defines how to partition and then map slots.
- In the examples, assigning overlapping VCs to different LNs has secured conflict freedom. Is it a sufficient and necessary condition, in general?
- LN is partitioned with respect to a reference buffer, which is a shared buffer. As overlapping VCs may have many shared buffers, how is this reference buffer selected? Are LNs with respect to all shared buffers equivalent?

In the next section, we answer these questions formally.

V. Formal Underpinning on LN-based Slot Allocation

A. Assumptions and Definitions

We consider static VCs, meaning that VCs do not change their paths and characteristics throughout system execution. We also assume that one LN is allocated to only one VC. But one VC may subscribe to multiple LNs.

Definition 5: A VC v comprises an ordered set of buffers $< b_0, b_1, b_2, \cdots, b_{H-1} >$. The size of v, denoted $|v|$, is the number of

[1] $\lceil x \rceil$ is the ceiling function that returns the least integer not less than x.

buffers, H. $d_{\overrightarrow{b_ib_j}}$ is the distance in number of slots[2] from b_i to b_j. On v, $d_{\overrightarrow{b_ib_{i+1}}} = 1$, meaning that the buffers are adjacent.

Definition 6: The *admission pattern* on a VC requires that N packets are admitted in a sequence of D ($D \geq N$) time slots. This gives a bandwidth requirement of N/D packets/slot, but the exact time slots for admitting the N packets are not specified. A *packet flow* is defined by infinitely repeating the admission pattern. We call D the *admission cycle*. With respect to a buffer b and a natural $d < D$, we define a *target class* as an infinite set of packets that arrive at buffer b at slots $d + kD$, $\forall k \in \mathbb{N}$. We call d the *initial distance* of the target class to buffer b.

For an open-ended VC, $D = W$, where W is the window size of a VC packet flow; For a closed-loop VC, $D = H$, since v is a loop and a container revisits the same buffer after H slots. N is the number of containers launched on the VC.

Definition 7: Two VCs v_1 and v_2 *overlap* if they share at least one buffer, i.e., $v_1 \cap v_2 \neq \emptyset$. The two VCs *conflict* in buffer b, denoted $b \in v_1 \wedge v_2$, if and only if it is possible that two packets, one from each VC, visit buffer b at the same time. $v_1 \wedge v_2 = \emptyset$ means that v_1 and v_2 are conflict free.

Definition 8: Given a VC $v = < b_0, b_1, b_2, \cdots, b_{H-1} >$ and its admission cycle D, $b_i \in v$, a natural $1 \leq T \leq D$ and a natural τ, $0 \leq \tau < T$, we define a *LN* $ln^T_\tau(v, b_i)$ as an infinite set of *(time slot, buffer)* pairs as follows:

$$ln^T_\tau(v, b_i) = \{(t, b_j) | t = \tau + d_{\overrightarrow{b_ib_j}} + kT,\ 0 \leq j < H,\ \forall k \in \mathbb{N}\}$$

Hence, a LN is defined for a given VC and one of its buffers. The number of LNs for a VC is always equal to T. The motivation of the LN is to precisely define the flow of packets on the VC and each target class is dedicated to exactly one LN. The time when packets visit buffers of the VC is given by the *(time slot, buffer)* pairs of the LN. On a LN, every T slots a packet visits a particular buffer. Consequently, the bandwidth possessed by a LN is $1/T$ packets/slot.

The LNs of a VC have an inherent property: if $\tau_1, \tau_2 \in [0, T-1]$ and $\tau_1 \neq \tau_2$, then packets admitted on different LNs never collide, because $ln^T_{\tau_1}(v, b) \cap ln^T_{\tau_2}(v, b) = \emptyset$.

Definition 9: A LN-cover is a complete set of LNs defined for a VC v with respect to a buffer b, $b \in v$,

$$LN\text{–}cover(v, b, T) = \{ln^T_\tau(v, b) \mid 0 \leq \tau < T\}$$

Definition 10: VC-to-LN assignment/subscription: a VC v is assigned to or subscribes to $ln^T_\tau(v, b)$ if and only if, on v, a target class, which has an initial distance d to buffer b and the admission cycle D, satisfies $mod(d + kD, T) = \tau$, $\forall k \in \mathbb{N}$.

If a VC v does not overlap with any other VCs, the maximum number of LNs on v is D, since v allows for up to D target classes and one class uses exactly one LN.

B. Overlapping VCs

Lemma 1: Let v_1 and v_2 be two overlapping VCs and D_1, D_2 be their admission cycles, respectively. Let c_1 and c_2 be any two target classes on v_1 and v_2 with respect to a shared buffer b, respectively; d_1 and d_2 are the initial distances of c_1 and c_2 to buffer b, respectively. We have $b \in v_1 \wedge v_2$ iff $\exists k_1, k_2 \in \mathbb{N}$ such that $d_1 + k_1D_1 = d_2 + k_2D_2$.

Proof:

(1) Sufficient: We assume that $\exists k_1, k_2 \in \mathbb{N}$ such that $d_1 + k_1D_1 = d_2 + k_2D_2 (= t)$. The left-hand side of the equation implies that c_1 enters buffer b at time slot t, and the right-hand side implies that c_2 enters b the same slot. Hence $b \in v_1 \wedge v_2$.

[2]As one hop takes one slot to travel, we equivalently measure the distance in number of slots.

(2) Necessary: Suppose, after t slots, c_1 and c_2 collide in buffer b, $b \in v_1 \wedge v_2$. For c_1, $t = d_1 + k_1D_1$; for c_2, $t = d_2 + k_2D_2$. Therefore $d_1 + k_1D_1 = d_2 + k_2D_2$. ∎

Theorem 1: Let T be the number of LNs, which two overlapping VCs, v_1 and v_2, can subscribe to without conflict. Then T is a Common Factor (CF) of their admission cycles, D_1 and D_2.

Proof: Suppose that b is the reference buffer.

Let $ln^T_{\tau_1}(v_1, b)$ and $ln^T_{\tau_2}(v_2, b)$ be the LN subscribed by v_1 and v_2, respectively. According to Definition 10, we have $\tau_1 = mod(d_1 + k_1D_1, T)$ and $\tau_2 = mod(d_2 + k_2D_2, T)$.

We start with $\tau_1 = mod(d_1 + k_1D_1, T)$, $\forall k_1 \in \mathbb{N}$. When $k_1 = 0$, $d_1 = k'_1T + \tau_1$; when $k_1 = 1$, $d_1 + D_1 = k''_1T + \tau_1$ and $k''_1 > k'_1$. From the last two equations, we get $D_1 = (k''_1 - k'_1)T$, meaning that T is a factor of D_1.

Similarly, using $\tau_2 = mod(d_2 + k_2D_2, T)$, $\forall k_2 \in \mathbb{N}$, we can derive that T is a factor of D_2.

Therefore T is a CF of D_1 and D_2, i.e., $T \in CF(D_1, D_2)$. ∎

By Theorem 1, the number T of LNs for v_1 and v_2 can be any value in the common factor set $CF(D_1, D_2)$. The least number of LNs is 1. However, if the number of LNs for two VCs is 1, only one of the two VCs can subscribe to it. There is no room for the other VC. Therefore we need at least two LNs. In general, if n VCs overlap in a shared buffer, there must be at least n LNs, one for each VC, to avoid conflict. In order to maximize the number of options and have finer LN bandwidth granularity, we consider the number T of LNs to be the *Greatest Common Divisor (GCD)* throughout the paper. Hence, for the two overlapping VCs, v_1 and v_2, the number T of LNs equals $GCD(D_1, D_2)$.

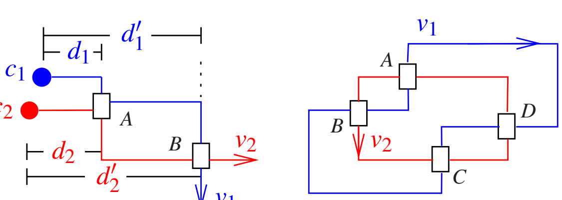
Theorem 2: Assigning v_1 and v_2 to different LNs with respect to any shared buffer is a sufficient and necessary condition to avoid conflict between v_1 and v_2.

Proof: By Theorem 1, the maximum number T of LNs for v_1 and v_2 is $T = GCD(D_1, D_2)$. We can write $D_1 = A_1T$ and $D_2 = A_2T$, where A_1 and A_2 are co-prime.

By Definition 10, v_1 and v_2 subscribe to different LNs $\Leftrightarrow$ $mod(d_1 + k_1D_1, T) \neq mod(d_2 + k_2D_2, T)$. Since $D_1 = A_1T$ and $D_2 = A_2T$, $mod(d_1 + k_1D_1, T) \neq mod(d_2 + k_2D_2, T)$ $\Leftrightarrow$ $mod(d_1, T) \neq mod(d_2, T)$.

(1) Sufficient: $mod(d_1, T) \neq mod(d_2, T)$ $\Rightarrow$ $d_1 + k'_1T \neq d_2 + k'_2T$, $\forall k'_1, k'_2 \in \mathbb{N}$. When $k'_1 = k_1A_1$ and $k'_2 = k_2A_2$, $\forall k_1, k_2 \in \mathbb{N}$ $\Rightarrow$ $d_1 + k_1A_1T \neq d_2 + k_2A_2T$ $\Rightarrow$ $d_1 + k_1D_1 \neq d_2 + k_2D_2$. According to Lemma 1, v_1 and v_2 do not conflict, i.e., $v_1 \wedge v_2 = \emptyset$.

(2) Necessary: Suppose $v_1 \wedge v_2 = \emptyset$ $\Rightarrow$ $d_1 + k_1D_1 \neq d_2 + k_2D_2$, $\forall k_1, k_2 \in \mathbb{N}$. But let us assume $mod(d_1, T) = mod(d_2, T)$. Then we have $d_1 - d_2 \neq k_2D_2 - k_1D_1$ but $d_1 - d_2 = kT$, $k \in \mathbb{Z}$. $\Rightarrow$ $k + k_1A_1 \neq k_2A_2$, $\forall k_1, k_2 \in \mathbb{N}$. However, this inequality is not always true, for example, when $k_1 = A_2$; $k_2 = A_1 + 1$; $k = A_2$. Thus, our assumption cannot be true, and $mod(d_1, T) \neq mod(d_2, T)$. This means that v_1 and v_2 subscribe to different LNs. ∎



(a) (b)

Fig. 6. Two or multiple shared buffers

By Theorem 2, VCs must stay in different LNs referring to *any shared buffer*. However, as overlapping VCs may have multiple shared buffers, LN partitioning might change with a different reference buffer. Figure 6a shows that two open-ended VCs, v_1 and v_2, overlap in buffers A and B. Apparently, no conflict with respect to buffer A does not imply no conflict with respect to another buffer B. We derive the following theorem to check the *reference consistency*.

Theorem 3: Suppose that two overlapping VCs, v_1 and v_2, have two shared buffers A and B. Let the distances from buffer A to B along v_1 and v_2 be $d_{\vec{AB}}(v_1)$ and $d_{\vec{AB}}(v_2)$, respectively. Let the initial distance of c_1 to A be d_1, to B be d'_1; from c_2 to A be d_2, to B be d'_2. Assume that c_1 on v_1 and c_2 on v_2 do not conflict in A, then $d_{\vec{AB}}(v_1) - d_{\vec{AB}}(v_2) = kT$, where $T = GCD(D_1, D_2)$ and $k \in \mathbb{Z}$, is a sufficient and necessary condition for c_1 and c_2 to be conflict-free with respect to B. If so, we say the two shared buffers are *consistent*.

Proof: $d_{\vec{AB}}(v_1) = d'_1 - d_1$ and $d_{\vec{AB}}(v_2) = d'_2 - d_2 \Rightarrow d_{\vec{AB}}(v_1) - d_{\vec{AB}}(v_2) = (d'_1 - d'_2) - (d_1 - d_2)$. Further, $d_{\vec{AB}}(v_1) - d_{\vec{AB}}(v_2) = kT \Leftrightarrow mod(d'_1 - d'_2, T) = mod(d_1 - d_2, T)$. Condition $mod(d_1, T) \neq mod(d_2, T) \Leftrightarrow mod(d'_1, T) \neq mod(d'_2, T)$. Thus c_1 and c_2 are conflict free with respect to B. ■

By Theorem 3, we can further conclude that if two VCs have multiple shared buffers, all shared buffers must be consistent in order to be conflict-free. For instance, as shown in Figure 6b, if the two closed-loop VCs, v_1 and v_2, have no conflict, then all shared buffers $v_1 \cap v_2 = \{A, B, C, D\}$ must be consistent. If the consistency is checked *pair-wise*, the total number of checking times is $C_u^2 = u(u-1)/2$, where u is the number of shared buffers. However, the check can be done efficiently.

Theorem 4: Suppose that v_1 and v_2 have at least three shared buffers $A, B, C \in v_1 \cap v_2$. If A and B, and B and C are consistent, then A and C are consistent.

Proof: As A and B are consistent, $d_{\vec{AB}}(v_1) - d_{\vec{AB}}(v_2) = k_1 T$. As A and C are consistent, $d_{\vec{AC}}(v_1) - d_{\vec{AC}}(v_2) = k_2 T$. By deducting the two equations, we have, $d_{\vec{AB}}(v_1) - d_{\vec{AB}}(v_2) - (d_{\vec{AC}}(v_1) - d_{\vec{AC}}(v_2)) = (k_1 - k_2)T$. Further, we have $d_{\vec{BC}}(v_1) - d_{\vec{BC}}(v_2) = k_3 T$, $k_3 \in \mathbb{Z}$. According to Theorem 3, B and C are consistent. ■

By Theorem 4, reference consistency may be linearly checked. As a result, the total number of checking times is reduced to $u-1$. If all shared buffers are consistent, any shared buffer can be used as a *reference buffer* to conduct LN partitioning and assignment. If they are not consistent, v_1 and v_2 conflict.

In summary, we have formally answered the questions in Section IV-E. The number of LNs of two overlapping VCs, v_1 and v_2, equals $GCD(D_1, D_2)$. Assigning VCs to different LNs is sufficient and necessary to promise conflict freedom. If overlapping VCs have multiple shared buffers, reference consistency must be first checked, and this check can be done linearly. If consistent, anyone of the shared buffers can be used as the reference buffer.

VI. The LN-based Slot Allocation Method

A. The Slot Allocation Algorithm

Algorithm 1 shows the pseudo code of the slot allocation method. The input is a set of n VC specifications, and the output is a set of TDM VC implementations. If the procedure returns true, the implementation set contains a TDM VC implementation for each VC specification, and routing tables in switches. If the procedure fails, the implementation set is empty. As the slot allocation is iterative, the algorithm has a complexity of $O(n^2)$. The slot allocation comprises

- *VC-to-LN assignment*: This step assigns VCs to different LNs. It is conducted pair-wise in a well-defined order and incrementally.

Algorithm 1 The pseudo code of LN-based slot allocation

Input: Q: a set of path-defined VC specification, $\{\bar{v}_1, \bar{v}_2, \cdots, \bar{v}_n\}$.
Output: S: a set of TDM VC implementation, $\{v_1, v_2, \cdots, v_n\}$.

```
Initially, state(v_i)=0; // v_i's LN assignment is not conducted.
Sort Q by a priority criterion;
bool slot_allocation(Q, &S){
  for i=1 to n {
    for j=1 to n {
     if (i!=j)
      if (VC_to_LN(v_i, v_j)==false) // pair-wise VC-to-LN assignment
        return false;
  for i=1 to n
    create_routing_table(v_i);
  return true; }
```

- *Routing table creation*: This step is performed only if the previous step is performed successfully. Using the VC-to-LN assignment for each VC and the VC path, we can accordingly configure routing tables in switches.

Next, we detail the two steps.

B. The VC-to-LN Assignment Procedure

VC-to-LN assignment is the key step for the LN-based slot allocation method. We sketch the VC-to-LN procedure in Algorithm 2. The input to the algorithm is a pair of VCs, (v_i, v_j)[3], and their paths are known. The function returns true if VC-to-LN assignment is done successfully for both VCs, and returns false otherwise. A VC v has two configuration states, either 0 or 1. 'state(v)=0' means that VC-to-LN assignment has not performed for v yet; 'state(v)=1' means that the VC-to-LN assignment for v is done successfully.

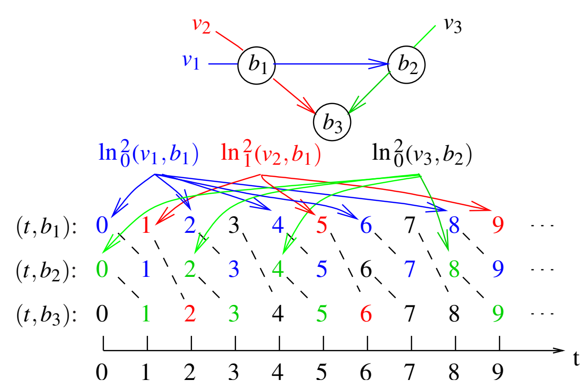


Fig. 7. An example of VC-to-LN assignment

VC	*Buf. set*	bw	N	$W(D)$	LN
v_1	b_1, b_2	1/2	1	2	$ln_0^2(v_1, b_1) = \{s_0^2(b_1), s_1^2(b_2)\}$
v_2	b_1, b_3	1/4	1	4	$ln_1^2(v_2, b_1) = \{s_1^4(b_1), s_2^4(b_3)\}$
v_3	b_2, b_3	3/8	3	8	$ln_0^2(v_3, b_2) = \{s_{0,2,4}^8(b_2), s_{1,3,5}^8(b_3)\}$

TABLE II
VC PARAMETERS AND VC-TO-LN ASSIGNMENT RESULTS FOR FIG.7

We exemplify how this VC-to-LN assignment is conducted. Figure 7, where a bubble represents a buffer, shows three VCs, v_1, v_2 and v_3. Their paths and parameters are listed in Table II. As elaborated below, the VC-to-LN assignments are performed in order (v_1, v_2), (v_1, v_3) and (v_2, v_3).

[3]VC pairs (v_i, v_j) and (v_j, v_i) are equivalent in the paper.

Algorithm 2 The VC-to-LN assignment procedure

bool **VC_to_LN**(v_i, v_j){
if ($v_i \cap v_j == \emptyset$) return true;
if (reference_consistency(v_i, v_j)==false) return false;
// v_i and v_j overlap but satisfy reference consistency
take any shared buffer b as the reference buffer $Ref(v_i, v_j) = b$;
compute the shared number T of LNs, $T = GCD(D_i, D_j)$;
if (state(v_i)==0 && state(v_j)==0) {
  // Both states are 0
  for v in $\{v_i, v_j\}$ {
   compute the available LN set for v, $AS_{ln}(v)$;
   compute the required number of LNs $N_{ln}(v) = \lceil NT/D \rceil$;
   if $|AS_{ln}(v)| < N_{ln}(v)$ return false;
   assign LNs from AS_{ln} to v;
   allocate slot sets in the assigned LNs within D to v;
   state(v_i)=1; state(v_j)=1; }
  return true; }
if (state(v_i) != state(v_j) {
  // One state is 0 and the other 1
  // suppose (state(v_i)=0 and state(v_j)=1)
  map v_j's allocated slot sets to the new LN set as the consumed LN set by v_j, $CS_{ln}(v_j)$;
  compute the available LN set for v_i, $AS_{ln}(v_i)$;
  compute the required number of LNs $N_{ln}(v_i) = \lceil N_i T_i / D_i \rceil$;
  if $|AS_{ln}(v_i)| < N_{ln}(v_i)$ return false;
  assign LNs from $AS_{ln}(v_i)$ to v_i;
  allocate slot sets in the assigned LNs within D_i to v_i;
  state(v_i)=1;
  return true; }
if (state(v_i)==1 && state(v_j)==1) {
  // Both states are 1
  map v_i's allocated slot sets to the new LN set as the consumed LN set by v_i, $CS_{ln}(v_i)$;
  map v_j's allocated slot sets to the new LN set as the consumed LN set by v_j, $CS_{ln}(v_j)$;
  if ($CS_{ln}(v_i) \cap CS_{ln}(v_j) == \emptyset$)
   return true;
  else return false;}

1) VC_to_LN(v_1, v_2): $Ref(v_1, v_2) = b_1$. Since $D_1 = 2$ and $D_2 = 4$, $T = GCD(D_1, D_2) = 2$. We can partition b_1's slots into two logical sets. Initially, state(v_1)=0 and state(v_2)=0. The branch of "Both states are 0" is executed. We take v_1 first. The available LN set for v_1 $AS_{ln}(v_1) = \{0, 1\}$, thus $|AS_{ln}(v_1)| = 2$. The required number of LNs $N_{ln}(v_1) = \lceil N_1 T / W_1 \rceil = 1$. As $|AS_{ln}(v_1)| > N_{ln}(v_1)$, there are enough LNs to support v_1 bandwidth. We assign $ln_0^2(v_1, b_1)$ to v_1. The consumed LN set of v_1 $CS_{ln}(v_1) = \{0\}$. We then allocate slot sets $s_0^2(b_1)$ and $s_1^2(b_2)$ to v_1. The two sets constitute LN $ln_0^2(v_1, b_1)$. Next, we take v_2 up. $AS_{ln}(v_2) = \{0, 1\} - CS_{ln}(v_1) = \{1\}$. The required number of LNs of v_2 $N_{ln}(v_2) = \lceil N_2 T / W_2 \rceil = 1$. We assign $ln_1^2(v_2, b_1)$ to v_2. Then we allocate slot sets $s_1^4(b_1)$ and $s_2^4(b_3)$ to v_2. After this assignment, state(v_1)=1 and state(v_2)=1.
2) VC_to_LN(v_1, v_3): $Ref(v_1, v_3) = b_2$. As $D_1 = 2$ and $D_3 = 8$, $T = GCD(D_1, D_3) = 2$. Since state(v_1)=1 and state(v_3)=0, the branch of "One state is 0 and the other 1" is executed. We map $ln_0^2(v_1, b_1)$ with respect to the reference buffer b_2, resulting in an equivalent LN $ln_1^2(v_1, b_2)$. Thus the consumed LN set of v_1 $CS_{ln}(v_1) = \{1\}$. The available LN set of v_3 is $AS_{ln}(v_3) = \{0, 1\} - CS_{ln}(v_1) = \{0\}$. The required number of LNs of v_3 $N_{ln}(v_3) = \lceil N_3 T / W_3 \rceil = 1$. We assign $ln_0^2(v_3, b_2)$ to v_3. Then we allocate slot sets $s_{0,2,4}^8(b_2)$ and $s_{1,3,5}^8(b_3)$ to v_3. After this assignment, state(v_3)=1.
3) VC_to_LN(v_2, v_3): $Ref(v_2, v_3) = b_3$. As $D_2 = 4$ and $D_3 = 8$, $T = GCD(D_2, D_3) = 4$. Since state(v_2)=1 and state(v_3)=1, the branch of "Both states are 1" is executed. In this step, we check whether the allocated slot sets for v_2 and v_3 can stay in different LNs after mapping them to the four LNs with respect to the reference buffer b_3. We map $s_1^4(b_1)$ of v_2 on b_3, obtaining an equivalent LN $ln_2^4(v_2, b_3)$. Then we map $s_{0,2,4}^8(b_2)$ of v_3 on b_3, obtaining LN $ln_{1,3}^4(v_3, b_3)$. Because $ln_2^4(v_2, b_3) \cap ln_{1,3}^4(v_3, b_3) = \emptyset$, v_2 and v_3 are conflict free with their slot assignment.

After the above three steps, the VC-to-LN assignments for the three VCs are successful. The slot sets are allocated accordingly, as shown in Table II. These can be used to create routing tables in switches.

C. Routing Table Creation

When the VC-to-LN assignment is successful for all VCs, a feasible solution or configuration is found. With each VC, a switch's partial routing table is created according to the VC's path and the allocated LNs, more accurately, the allocated slot sets within the admission cycle. The slot sets determine when the VC passes a particular buffer in a switch. For instance, if a VC v with an admission cycle D subscribes to $s_{\tau_1}^D(b)$ and $s_{\tau_2}^D(b)$, then slots $\tau_1 + kD$ and $\tau_2 + kD$ ($k \in \mathbb{N}$) of b are reserved for v. The VC path determines the input link e_{in} and the output link e_{out} of the switch used by v packets at the reserved slots. Thus, two routing table entries, $(\tau_1 + kD, e_{in}, e_{out})$ and $(\tau_2 + kD, e_{in}, e_{out})$, can be created in the switch. By composing the partial routing tables of all visiting VCs in a switch, we obtain a complete routing table for the switch. Optimization is also used to shrink the size of the routing tables. For example, entries $(4k, e_{in}, e_{out})$ and $(4k+2, e_{in}, e_{out})$ can be reduced to one entry $(2k, e_{in}, e_{out})$.

VII. An Industrial Case Study

A. The TDM VC Configuration Program

We have integrated the LN-based slot allocation method into our TDM VC configuration program. To explore the path diversity of VCs, this program runs a back-tracking algorithm. The algorithm is a recursive function performing a depth-first search. The solution space in a tree structure is generated while the search is conducted. At any time during the search, only the route from the start node to the current expansion node is saved. As a result, the memory requirement of the algorithm is $O(n)$, where n is the number of VCs. This is important since the solution space organization needs excessive memory if stored in its entirety. Whenever two VCs overlap, the assignment of VCs to LNs is performed. If they can be assigned to two different LNs with sufficient bandwidth, the assignment is done successfully. Otherwise, the assignment fails, and other path alternatives (back-tracking) have to be considered. This VC-to-LN assignment serves as a bounding function by which, if it fails, the algorithm prunes the current expansion node's subtrees, thus making the search efficient. In general, the more the alternative paths, the longer the run time. The program allows us to set the number of alternative paths to tradeoff between runtime and capability.

B. The Case Study

We applied our program to a real application provided by Ericsson Radio Systems. As mapped onto a 4×4 mesh in Figure 8, this application consists of 16 IPs. Specifically, n_2, n_3, n_6, n_9, n_{10} and n_{11} are ASICs; n_4, n_7, n_{12}, n_{13}, n_{14} and n_{15} are DSPs; n_5, n_8 and n_{16} are

FPGAs; n_1 is a device processor which loads all nodes with program and parameters at start-up, sets up and controls resources in normal operation. Traffic to/from n_1 is for system initial configuration and no longer used afterwards. There are 26 node-to-node traffic flows that are categorized into nine types of traffic flows {**a**, **b**, **c**, **d**, **e**, **f**, **g**, **h**, **i**}, as marked in the figure. Traffic **a** and **h** are multi-cast traffic, and others are unicast traffic. The traffic flows are associated with a bandwidth requirement. In this case study, we use *closed-loop VCs* to implement all traffic flows, $\kappa_{bw} = 1$.

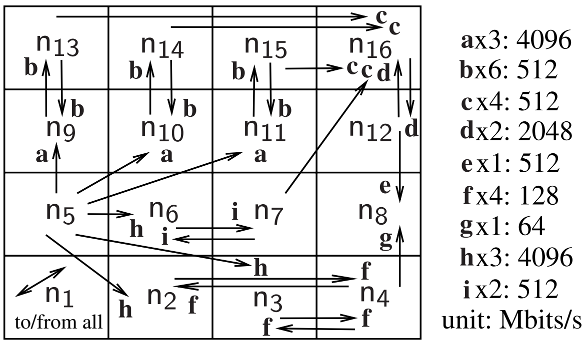


Fig. 8. Traffic flows for a radio system

The case study comprises *VC specification* and *VC configuration*. The first phase involves *determining link capacity*, *normalizing VC bandwidth demand* and *merging traffic flows*. The second phase runs the configuration program, exploring VC's all minimal paths.

We first determine the minimum link capacity bw_{link} by considering a heaviest loaded link. Link $e(n_5, n_9)$ is such a link since the **a**-type traffic passes it and $bw_a = 4096$ Mbits/s. To support bw_a, $bw_{link} \geq 4096$ Mbits/s. We choose 4096 Mbits/s for bw_{link}. This is an initial estimation and subject to optimization later on. Afterwards, we normalize the bandwidth demand into a fraction of bw_{link}. For example, 512 Mbits/s is equivalent to 1/8 link capacity.

Then we *merge traffic flows* in order to construct efficient VCs by taking advantage of multi-node VCs. This can be done for multicast and low-bandwidth traffic. For the two multi-cast traffic **a** and **h**, we build two multi-node VCs as $\dot{v}_a(n_5, n_9, n_{10}, n_{11})$ and $\dot{v}_h(n_5, n_6, n_2, n_3)$. The notation $\dot{v}$ refers to a VC specification before path selection. Traffic **b**, **c** and **f** require low bandwidth. We specify a VC to include as many nodes as a type of traffic flow spreads. For traffic **b**, we define a six-node VC, $\dot{v}_b(n_9, n_{10}, n_{11}, n_{13}, n_{14}, n_{15})$; for **c**, a five-node VC $\dot{v}_c(n_{13}, n_{14}, n_{15}, n_{16}, n_7)$; for **f**, a three-node VC $\dot{v}_f(n_2, n_3, n_4)$. Furthermore, as we use a closed-loop VC, two-simplex traffic flows can be merged into one duplex flow. For instance, for two **i** flows, we specify only one VC $\dot{v}_i(n_6, n_7)$. Note that, while merging traffic flows, the resulting VC must be able to provide enough bandwidth to support the flows. Performing this step results in 9 multi-node VCs.

With the three steps above, we complete defining the VC specification set. While executing the program to configure the VCs, we investigate the impact of VC sorting. Since VC sorting determines the VC levels in the solution tree and the VC-to-LN assignment order, it affects the runtime and the number of solutions. We tried three sorting schemes: *random*, *higher bandwidth first*, *less number of path options first*. In order to compare the potential of the schemes, our algorithm terminates after all solutions are found. We did not do any tweaking or tuning but used the original IP-to-node mapping and IP communication patterns without change. Corresponding to the three sorting schemes, the number of solutions found is 33, 30 and 76; the run time is 6, 6 and 12 seconds. Sorting by the number of path options is best in this example. This means that VCs with fewer alternative paths should be layouted first because they are more constrained. As a result, pruning their subtrees and allocating slots are more effective when they are considered in the upper levels in the tree.

VIII. Conclusion

Slot allocation is a critical problem for TDM VC configuration. Its complexity arises from various path overlapping and bandwidth sharing scenarios. In the paper, based on our concept of LN, we develop and proof sufficient and necessary conditions for the configuration of conflict-free and bandwidth-satisfied VCs. They are applicable to both open-ended and closed-loop VCs in the state-of-the-art NoC proposals. We have also detailed the steps to perform the VC-to-LN assignment, i.e., slot allocation, and integrated the method into our multi-node TDM VC configuration program. Our industrial case study justifies our approach in effectiveness and practicality.

In the paper, we have considered *non-stalled* TDM VCs where VC packets use consecutive slots in consecutive switches. This type of TDM VC couples the latency requirement with the bandwidth requirement. For low-bandwidth low-latency traffic, it leads to overbooking bandwidth in order to satisfy the low latency constraint. In the future, we will extend our framework to cover *stallable* TDM VCs in order to make more efficient use of link bandwidth and to allow asynchronous network communication.

Acknowledgment

The research is partially supported by the EU FP6 project SPRINT under contract 027580.

References

[1] K. Goossens, J. Dielissen, and A. Rădulescu, "The Æthereal network on chip: Concepts, architectures, and implementations," *IEEE Design and Test of Computers*, vol. 22, no. 5, pp. 21–31, Sept-Oct 2005.

[2] M. Millberg, E. Nilsson, R. Thid, and A. Jantsch, "Guaranteed bandwidth using looped containers in temporally disjoint networks within the Nostrum network on chip," in *Proceedings of the Design Automation and Test in Europe Conference*, February 2004.

[3] T. Bjerregaard and J. Sparso, "A router architecture for connection-oriented service guarantees in the MANGO clockless network-on-chip," in *Proceedings of the Design, Automation and Test in Europe Conference*, 2005, pp. 1226–1231.

[4] E. Bolotin, I. Cidon, R. Ginosar, and A. Kolodny, "QNoC: QoS architecture and design process for network on chip," *The Journal of Systems Architecture*, December 2003.

[5] O. P. Gangwal, A. Rădulescu, K. Goossens, S. González Pestana, and E. Rijpkema, "Building predictable systems on chip: An analysis of guaranteed communication in the Æthereal network on chip," in *Dynamic and Robust Streaming In And Between Connected Consumer-Electronics Devices*, ser. Philips Research Book Series, P. van der Stok, Ed. Springer, 2005, vol. 3, ch. 1, pp. 1–36.

[6] A. Hansson, K. Goossens, and A. Rădulescu, "A unified approach to constrained mapping and routing on network-on-chip architectures," in *Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis*, Sept. 2005.

[7] T. Marescaux, B. Bricke, P. Debacker, V. N. Nollet, and H. Corporaal, "Dynamic time-slot allocation for QoS enabled networks on chip," in *Proc. of the IEEE 3rd Workshop on Embedded Systems for Real-Time Multimedia (ESTIMEdia'05)*, September 2005, pp. 47–52.

[8] S. Stuijk, T. Basten, M. Geilen, A. H. Ghamarian, and B. Theelen, "Resource-efficient routing and scheduling of time-constrained network-on-chip communication," in *Proceedings of the 9th Euromicro Conference on Digital System Design*, Aug. 2006.

[9] E. Nilsson and J. Öberg, "Reducing peak power and latency in 2D mesh NoCs using globally pseudochronous locally synchronous clocking," in *Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis*, September 2004.

[10] J. T. Brassil and R. L. Cruz, "Bounds on maximum delay in networks with deflection routing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 6, no. 7, pp. 724–732, July 1995.