Curbing Aggregate Member Flow Burstiness to Bound End-to-End Delay in Networks of TDMA Crossbar Real-Time Switches

Qixin Wang*, Yufei Wang*, Rong Zheng†,*, Xue Liu‡

Abstract—**To integrate the nowadays rapidly expanding distributed real-time systems, we need multi-hop real-time switched networks. A (if not "the") widely recognized/adopted real-time switch architecture is the *TDMA crossbar real-time* (TCRT) switch architecture. However, the original TCRT switch architecture assumes per-flow queueing. To support scalability, however, queue sharing (i.e. flow aggregation), must be allowed. With simple flow aggregation, flow burstiness can grow and infect, making schedulability and end-to-end delay bound analysis an open problem. To deal with this, we propose the *real-time aggregate* scheme. The scheme complies with the existing TCRT switch architecture, and deploys spatial-temporal isolation and over-provisioning to curb aggregate member flows' burstiness. This allows us to derive the closed-form end-to-end delay bound, and give the corresponding resource planning and admission control strategies. Simulations are carried out to show the effectiveness of the design.**

I. Introduction

Real-time networks are the venue where distributed real-time systems integrate. As nowadays distributed real-time systems rapidly scale up, real-time networks have to evolve from traditional *Local Area Networks* (LANs) toward multi-hop switched networks [1]. A typical example is avionics. A modern aircraft, such as A380, F-35, or space shuttle, already runs hundreds of processors, and may include hundreds of high-definition real-time video sources [2][3]. Such large number of nodes and traffic cannot be hosted by a single real-time LAN. This forces the launch of several initiatives to develop multi-hop real-time switched networks [4][5]. Similar demands also arise from industrial control, telepresence/tele-robotics, intelligent transportation, and medical device integration etc. [6].

A (if not "the") widely recognized/adopted real-time switch architecture for multi-hop real-time networks is the *Time-Division-Multiple-Access (TDMA) crossbar real-time switch* architecture (simplified as "*TCRT switch*" architecture in the following) [7][8][1][9][10][11]. This architecture is particularly important for mainstream switch manufacturers because it complies with (and even simplifies) a mainstream Internet switch architecture [1]. This lays a smooth evolution path for these manufacturers to build real-time switches.

However, the existing TCRT switch architecture assumes per-flow queueing. It is well-known that per-flow queueing has poor scalability [12][13][14]. In fact, this is the reason why nearly all high performance switches nowadays carry out certain *flow aggregation*: flows have to share queues sometimes. Flows sharing a same queue are referred to as an *aggregate*.

With aggregates, how to provide real-time delay guarantee becomes a non-trivial problem. Simply aggregating flows in TCRT switches allows the member flows' burstiness to grow and infect (an example is in Section III). As these member flows join other aggregates, the burstiness infection can spread further. This seriously complicates the network model, making schedulability and *end-to-end* (E2E) delay bound analysis an open problem.

To deal with the problem, we propose a novel scheme called *real-time aggregates*. This scheme exploits the features of TCRT switch architecture to curb aggregate member flows' burstiness. With real-time aggregates, we can derive closed form E2E delay bound, along with the corresponding resource planning and admission control strategies. Simulation shows the real-time aggregate scheme is efficient in terms of providing short E2E delay bound and high network utilization.

The remainder of the paper is organized as follows. Section II introduces the TCRT switch architecture; Section III proposes a naive aggregation scheme, and shows how it leaves E2E delay bound an open problem; Section IV proposes and analyzes the *real-time aggregate* scheme; Section V evaluates real-time aggregates; Section VI discusses related work; and Section VII concludes the paper.

II. Background

We shall first introduce the existing/basic TCRT switch architecture[1].

Network switches (no matter real-time or not) can be categorized as output queueing or input queueing. In output queueing, input ports (simplified as "*inputs*" in the following) does not buffer packets. Once a packet enters an input, it is immediately routed to its corresponding output port (simplified as "*output*" in the following) and buffered there.

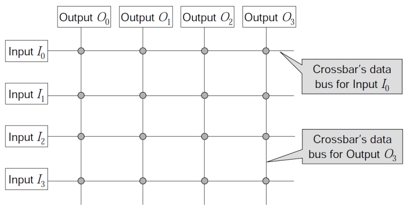
Though output queueing is intuitive, its inherent "N speed-up" problem [15] limits its adoption. Input queueing, instead, becomes the de facto standard among switch vendors[1][16].

In input queueing, a *crossbar* fabric connects inputs with outputs (see Fig. 1(a)). Packets are only buffered at inputs; when a packet enters an input, it is immediately routed into the right queue in the input (see Fig. 1(b)). At scheduled time, an output connects with one input, picks one of its queues, and fetches the queue's header packet. The fetched packet exits the output directly without further buffering (see Fig. 1(c)).

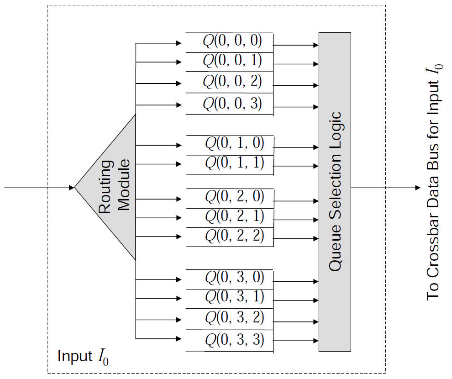
To facilitate scheduling, each packet is typically manipulated as fixed-size fragments called *cells*. The time cost to

* Dept. of Computing, The Hong Kong Polytechnic University, Email: csqwang@comp.polyu.edu.hk
† Dept. of Computer Science, University of Houston
‡ School of Computer Science, McGill University

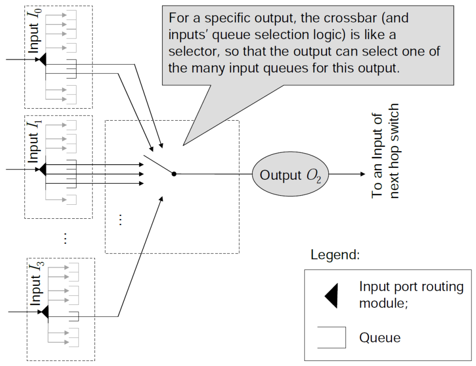
[1]Because both this paper and [8] share the same background, the content of this section is mostly the same as the Background section of [8].



(a) crossbar fabric, which connects inputs with outputs; each input connects to a data bus (the horizontal line segments) that intersects with each output's data bus (the vertical line segments); the intersections (grey dots) can be connected/disconnected in runtime by scheduler(s).



(b) an input port: packet routing and queueing are carried out in it; in input i, the kth queue buffering packets to output j is denoted as $Q(i, j, k)$.


Input I_0
Input I_1
Input I_3
For a specific output, the crossbar (and inputs' queue selection logic) is like a selector, so that the output can select one of the many input queues for this output.
Output O_2
To an Input of next hop switch
Legend:
Input port routing module;
Queue

(c) an output port: at different time slot, the output fetches packets from different input queues according to the switch scheduling scheme.

Fig. 1. Input Queueing Switch Architecture (quoted from [8])

send one cell across the crossbar is called one *cell-time*. To satisfy the crossbar constraint that at any time instance, an input can connect to at the most one output and vice versa, the switch operates periodically, and the period is one cell-time. At the beginning of each cell-time, the switch scheduler decides a one-to-one matching (simplified as "*matching*" in the following) between inputs and outputs and connect/disconnect crossbar intersections (the grey dots in Fig. 1(a)) accordingly. During the cell-time, each output tries to fetch a cell from its matched input for outputing.

Depending on different queueing and cell matching schemes, many input queueing switch architectures exist. TCRT switch architecture is one of them [1][9][10][11]. It runs as follows.

Each input carries out per-flow queueing. Each output maintains a TDMA schedule of M cell-time, a.k.a., the *M-slot frame*. The gth ($g = 0, 1, \ldots, M-1$) slot of the frame

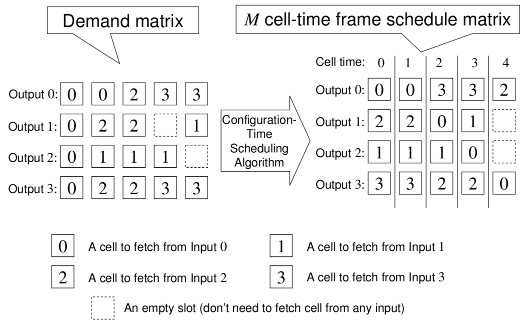

Demand matrix
M cell-time frame schedule matrix
Cell time: 0 1 2 3 4
Output 0:
Output 1:
Output 2:
Output 3:
Configuration-Time Scheduling Algorithm
0 A cell to fetch from Input 0
1 A cell to fetch from Input 1
2 A cell to fetch from Input 2
3 A cell to fetch from Input 3
An empty slot (don't need to fetch cell from any input)

Fig. 2. Conflict free schedule for TCRT switch (quoted from [17]): in this example, the switch has $N = 4$ inputs/outputs, $M = 5$; each row of the "schedule matrix" is a conflict free schedule for its corresponding output.

specifies which per-flow queue in which input to grant (i.e., to send a "grant" signal) at the beginning of the gth cell-time. Here g is a global counter incremented by 1 every cell-time (modulo M). On receiving a grant, the input port per-flow queue sends its header cell to the granting output during the cell-time; or do nothing if the queue is empty.

To ease narration, in the following, we use the term "*M-slot frame*" and "*frame*" interchangeably; and the term "*slot*" and "*cell-time*" interchangeably.

Remember crossbar requires that in any time instance an input can connect to no more than one output and vice versa. That is, the M-slot frame schedules of all N outputs must be a *matching* between the N inputs and N outputs in each cell-time. We call this requirement "*conflict free*" (see Fig. 2).

An important feature for TCRT switch architecture is its schedulability test method [1], quoted here as Theorem 1:

Theorem 1 (Schedulability): For an N inputs N outputs TCRT switch, if in every M-slot frame, each output needs to receive no more than M cells, and each input needs to send no more than M cells, then we can always find a *conflict free* schedule with a time cost of $\mathcal{O}(N^4)$.

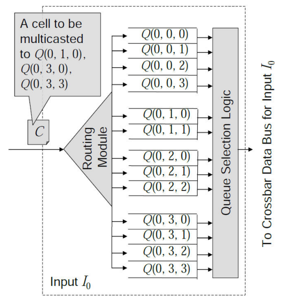
The corresponding $\mathcal{O}(N^4)$ scheduling algorithm is in [1].

So far, we are assuming all flows are unicast. The extension to support multicast is simple [8], as shown in Fig. 3. When a to-be-multicasted cell enters an input (see Fig. 3 (a)), it is duplicated into m copies (see Fig. 3 (b)). Depending on multicast routing, each copy enters its corresponding input port per-flow queue, and the rest is the same as unicast. When the copy enters the next-hop switch, same thing can happen again for further multicast branching.

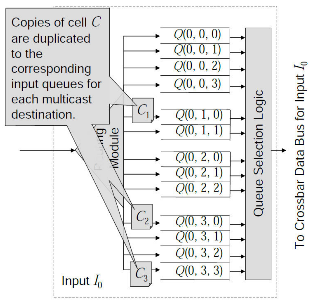
Such extension complies with the common constraint of crossbar that at any time instance, one input can connect to at the most one output and vice versa; hence will benefit smooth design evolution. Later, we will use this multicast scheme to design our flow aggregate mechanisms.

III. Naive Approach: Per-Aggregate Queueing

To address the flow aggregation demand proposed in Section I, in this section, we shall attempt a naive approach: add *per-aggregate queueing* into the aforementioned TCRT switches. Later in this section, we will show due to insufficient flow isolation, this approach makes the analysis of each flow's



(a) Step 1



(b) Step 2

Fig. 3. Multicast in TCRT switch (quoted from [8], note C_2 and C_3 may be fetched/outputted in different cell-times, but this does not matter).

schedulability (i.e., how much resources must be allocated to guarantee the existence of an E2E delay bound?) and E2E delay bounds (i.e., what is the E2E delay bound?) an open problem. A better solution is then proposed in Section IV.

Per-aggregate queueing means flows of a same aggregate share the same queue, as shown by Fig. 4.

Formally, let set F represent an aggregate, where $f \in F$ iff f is a member flow of F ("*iff*" means "*if and only if*"). A queue can be uniquely identified as $Q(I, O, F)$: I is the input where the queue resides; O is the intended output; and F is the aggregate the queue exclusively serves. In addition, let

$$\mathcal{P}_F \overset{def}{=} \{X | \exists \text{ flow } f \in X \text{ and } f \text{ joins aggregate } F \text{ right after leaving aggregate } X\} \tag{1}$$

denote the set of *Predecessor Aggregates* for F; and let

$$\mathcal{S}_F \overset{def}{=} \{X | \exists \text{ flow } f \in X \text{ and } f \text{ joins aggregate } X \text{ right after leaving aggregate } F\} \tag{2}$$

denote the set of *Successor Aggregates* for F.

Topologically, an aggregate F starts from an output O_0 that fetches cells from a set of queues $\{Q(I, O_0, X \cap F) | \forall X \in \mathcal{P}_F\}$ (note O_0 and $X \cap F$ together determines I). We call O_0 the *Aggregator* of F, or equivalently, O_0 *creates* F. F then passes several subsequent switches. Without loss of generality, suppose they are switch $1, 2, \ldots, k-1$ respectively. Suppose in switch i ($i = 1 \sim k-1$), F is queued in $Q(I_i, O_i, F)$ and forwarded to output O_i; and suppose O_{k-1}, the last hop of output for F, wires to input I_k of switch k. Input I_k then is F's *Segregator*, where F segregates into $|\mathcal{S}_F|$ queues ($|\bullet|$ is the cardinal of set $\bullet$). For flow $f \in F$ that joins $X \in \mathcal{S}_F$, the flow enters queue $Q(I_k, O, F \cap X)$, where O is determined by I_k and $F \cap X$.

These concepts on aggregates are more intuitively explained by an analogy to express trains, depicted in Fig. 4's caption.

As shown in Fig. 4, aggregates can share the same physical link(s), but their queues are mutually exclusive. This spatially partitions flows of different aggregates. However, within each aggregate, per-aggregate queueing is unable to isolate the aggregate's member flows. If one member flow is *bursty* (i.e. the flow's data rate changes drastically; e.g., to have no cell arriving in one M-cell-time frame, and then have four cells arriving in the next frame), it may make other flows bursty.

Fig. 5 shows an example on how the burstiness of a flow may emerge and "*grow*" due to clock drift [18]; and the burstiness of flows may "*infect*" each other due to queue sharing.

The example assumes six consecutively connected switches, $S_1 \sim S_6$, along an aggregate F. The events taken place in $S_1 \sim S_6$ are shown in Fig. 5 by six synchronized time axes from top to bottom: the top time axis for switch S_1, the second from top time axis for switch S_2, so on and so forth.

Without loss of generality, we assume TCRT switches always run an M-slot frame with $M = 10$ (note in reality, for giga-bps switches, M is usually in the order of $10^3 \sim 10^6$). Let τ_i (second) denote the duration of a cell-time for switch S_i; and $\tau_1 < \tau_2 < \ldots < \tau_6$, which implies clock drift.

The aggregate F through $S_1 \sim S_6$ consists of two flows: f_a and f_b. f_a's source end maximal traffic load is 3 cell/frame, while f_b's source end maximal traffic load is 1 cell/frame. They enter the per-aggregate queue Q_1 in switch S_1's input port, and is fetched by an output port of S_1, namely output O_1. Based on f_a and f_b's source end maximal traffic load, O_1 shall grant Q_1 for 4 times per M-slot frame. In Fig. 5, O_1 grants Q_1 for the kth time at $g_k^{(1)}$.

Unfortunately, f_a is bursty, either ever since source end, or due to burstiness growth/infection in the network. Therefore, f_a injects no traffic load throughout our observation period. f_b, however, is steady. The kth cell of f_b, denoted as c_k, arrives at Q_1 at time $a_k^{(1)}$. For example, in Fig. 5, c_0 arrives at Q_1 at time $a_0^{(1)}$. However, c_0 missed the grant at $g_3^{(1)}$ due to clock drift. On the other hand, because the bursty flow f_a injects no cells, c_0 is fetched by O_1 at $g_4^{(1)}$ using a slot originally for f_a. Similar things can happen at switch $S_2 \sim S_5$, so that $c_0 \sim c_3$ arrive at switch S_6 within one frame, using 3 additional slots originally for f_a, i.e., f_b grows bursty. This burst of $c_0 \sim c_3$ can infect other flows, as flow f_b later joins other aggregates.

Note we cannot stop the growth of f_b's burstiness by allocating more slots per frame to Q_i ($i = 1 \sim 5$), because bad phasing (like the one where cell arrival time $a_0^{(1)}$ just misses grant time $g_3^{(1)}$ in Fig. 5) can still happen.

Due to the complexity of flow burstiness growth and infection, a tight per-aggregate queueing burstiness upper bound is still an open problem (this concurs with the experience on flow aggregation research for other switch architectures [12][19][20]). That is, in general, we do not know the exact worst case burstiness of a flow in a per-aggregate queueing TCRT switched network. This stops us from deriving efficient schedulability test and tight E2E delay bound for per-aggregate queueing; though a conservative sufficient schedulability test and a loose E2E bound exist by applying classic DiffServ math framework [20][21][22].

IV. Real-Time Aggregate

To fix the shortcomings of per-aggregate queueing, we propose *real-time aggregate*, which carries out spatial-temporal partitioning and over-provisioning to curb the burstiness "growth" and "infection", hence making schedulability and E2E delay bound analysis possible.

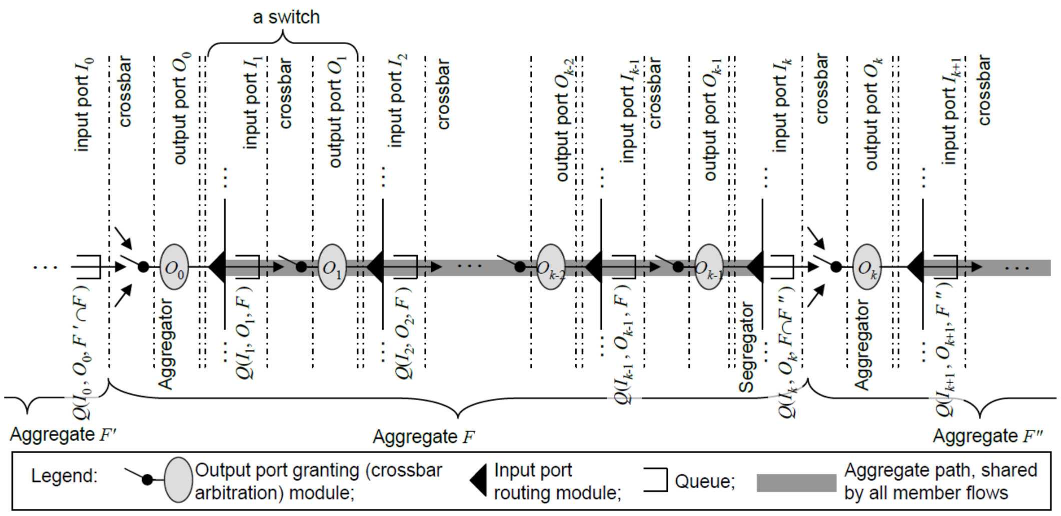


Fig. 4. Aggregate Topological Architecture. We can imagine every switch as a railway station, with each input an arrival platform, and each output a departure platform. An aggregate is an express train that runs non-stop from a unique departure platform (i.e., its aggregator) to a unique arrival platform (i.e., its segregator). A flow is a passenger, who boards/alights-from express trains (aggregates) at their aggregators/segregators. A departure platform (output) can serve as the aggregator for some express trains (aggregates), and as pass-by platform for some other express trains. Same is for an arrival platform (input) as segregator and pass-by.

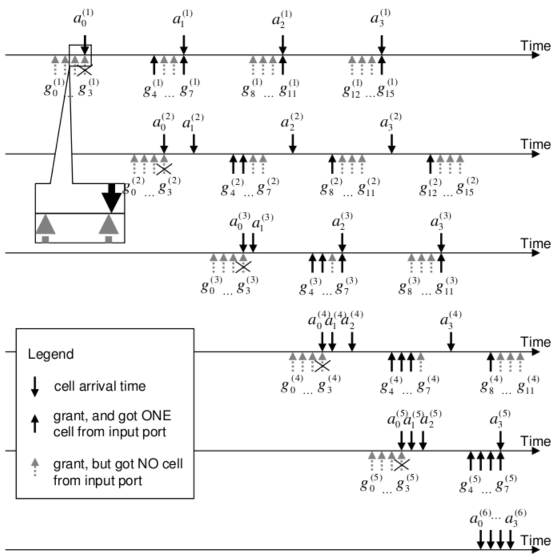

Time
Legend
cell arrival time
grant, and got ONE cell from input port
grant, but got NO cell from input port

Fig. 5. Burstiness Growth and Infection Example. This figure depicts in time domain a series of events happened in six hops of switches along an aggregate.

A. Heuristics

Fig. 5's example shows a main drawback of per-aggregate queueing: cells originally belong to different M-slot frames may pile up together. For example, in Fig. 5, cell $c_0 \sim c_3$ originally belong to 4 different frames respectively when arriving at switch S_1; but they all arrive at switch S_6 in one frame. The forwarding output of switch S_6 thus may forward $c_0 \sim c_3$ within one M-slot frame, propagating burstiness to other parts of the network.

To make an analogy, per-aggregate queueing is like queueing all words of an article without any comma. Such an article is certainly hard to read and error prone. But the solution

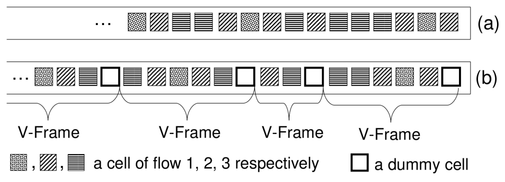


Fig. 6. With/without (dummy cells delineating) temporal isolation

is simple: just add commas between words, then the article becomes readable. These commas provide temporal isolation.

Similarly, we can add temporal isolation to the original per-aggregate queueing. This is illustrated by Fig. 6. Fig. 6(a) shows a queue for per-aggregate queueing: all cells are piled up together. Our plan is to insert *dummy cells* between these cells (see Fig. 6(b)) to separate cells that should not be forwarded within a same M-slot frame. These dummy cells serve the function of "commas". When an output "reads" (i.e., forwards) cells from queue (b), it should "pause" whenever it reaches a "comma" (i.e., dummy cell) until the next M-slot frame starts.

With dummy cells, we add temporal isolation to the spatial isolation of per-aggregate queueing (different aggregates are spatially isolated from each other because they use different queues). This combined spatial-temporal isolation better curbs the burstiness growth and infection. But we still have one more glitch: since we admit clock drift exists between switches, the incoming of cells may be slightly faster than the reading of cells. This may cause queue overflow.

We address this problem with *over-provisioning*. We shift the "comma" to include some more words than the original "sentence" length. Then in each M-slot frame, if the next hop output "reads" until it sees a "comma", it reads more than needed (i.e., over-provisioning). In this way, we speed up the "reading" of cells, so that possible clock-drift is compensated.

To implement this, we let the aggregator insert a dummy

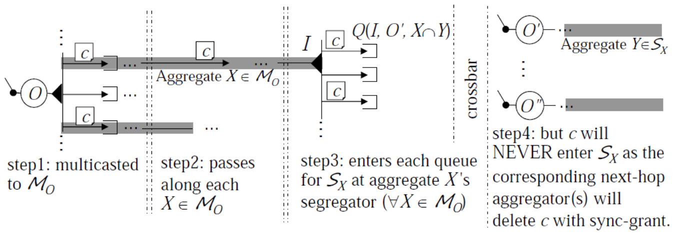

c
Aggregate $X \in \mathcal{M}_O$
I
$Q(I, O', X \cap Y)$
crossbar
O'
Aggregate $Y \in \mathcal{S}_X$
O''
O
step1: multicasted to $\mathcal{M}_O$
step2: passes along each $X \in \mathcal{M}_O$
step3: enters each queue for $\mathcal{S}_X$ at aggregate X's segregator ($\forall X \in \mathcal{M}_O$)
step4: but c will NEVER enter $\mathcal{S}_X$ as the corresponding next-hop aggregator(s) will delete c with sync-grant.

Fig. 7. The Life Cycle of a Dummy Cell c

cell every $(M+1)$ cell time instead of every M cell time. In addition, dummy cells are created and inserted by each aggregate's aggregator, and *deleted* (consumed) by the aggregate's segregator. This ensures dummy cells of each aggregate will NOT enter other aggregates to cause confusion.

The above heuristics leads to our *real-time aggregate* design.

B. Real-Time Aggregate Design Details

To implement the heuristics of Section IV-A, we reuse the per-aggregate queueing topology architecture of Fig. 4, and revise the granting mechanism in the TCRT switch of Section II to achieve the effects shown in Fig. 7.

In Fig. 7, output O plays the role of aggregator for two aggregates. Let $\mathcal{M}_O \overset{def}{=} \{X | O \text{ is the aggregator for aggregate } X\}$. Then every $(M+1)$ cell-time, O multicast a dummy cell c to $\mathcal{M}_O$. Every copy of c respectively passes along each aggregate $X \in \mathcal{M}_O$, marking the temporal border (i.e., beginning) of a new $(M+1)$ cell-time frame, a.k.a. "*virtual-frame*" or "*v-frame*", to be differentiated from the "M cell-time frame", a.k.a. "frame".

When c enters the segregator of $X (\forall X \in \mathcal{M}_O)$ (denoted as input I in the figure), c is duplicated and enqueued *as if* it is going to be further multicasted to $\mathcal{S}_X$, X's successor aggregates. Specifically, for each $Y \in \mathcal{S}_X$, suppose output O' is Y's aggregator, and $Q(I, O', X \cap Y)$ is the queue in segregator I that corresponds to Y (i.e., all cells leaving aggregate X and joining aggregate Y will be queued in $Q(I, O', X \cap Y)$), then a copy of dummy cell c will enter $Q(I, O', X \cap Y)$. When it reaches $Q(I, O', X \cap Y)$'s header, aggregator O' sees the beginning of the next v-frame. O' will hence block until the next M cell-time frame. In the next M cell-time frame, the first thing O' will do is to delete c, and then forward the remaining cells from $Q(I, O', X \cap Y)$ *until* the next dummy cell reaches $Q(I, O', X \cap Y)$'s header.

Since O' deletes c, c would NOT enter Y, although c attempts to. Also, since O' blocks at seeing c till the next frame, O' will not aggregate cells generated in different v-frames in one frame. This temporally curbs burst growth.

The above behavior is formally specified by the pseudo codes in Fig. 8 and 9. The pseudo codes extend the "grant" protocol in the TCRT switch described in Section II. At the beginning of each cell-time, an output executes OnCellTimeStart() (see Fig. 8) to grant input. On getting a grant g from an output, an input executes OnGranted(g) (see Fig. 9). Some pseudo code details are explained in the following:

Each output still maintains an M-slot frame (a.k.a. M cell-time frame) schedule. Each slot corresponds to a *slot descriptor* describing what the output needs to do during that slot. Let S_O denote output O's M-slot frame schedule. Let s_i^O denote the ith ($i = 0 \sim M-1$) slot descriptor of S_O; and s_i^O has three fields:

1) $s_i^O.queue$ is the input queue to grant at the ith slot;
2) $s_i^O.role$ specifies O's role for $s_i^O.queue$. It can be either "aggregator" or "others".
3) $s_i^O.sync$ is a boolean that is only valid when $s_i^O.role$ is "aggregator". It is "true" iff it is the first slot in S_O that grants the queue of $s_i^O.queue$.

$s_i^O.role$ and $s_i^O.sync$ together differentiate three types of grant: *normal-data-grant*, *aggregator-data-grant*, and *sync-grant* (see Fig. 8 line 4, 12, 14; and Fig. 9 line 5, 7, 8). "Normal-data-grant" is the conventional grant that fetches the input queue's header cell if there is one. It is used by *intermediate nodes* (the outputs that the aggregate passes along, but not the aggregator for the aggregate) of an aggregate. In contrast, if an aggregator wants to issue a data grant, it must use "aggregator-data-grant". The difference between "normal-" and "aggregator-" data-grant is that aggregator-data-grant fetches nothing if the input queue header is a dummy cell. This realizes the heuristic of "pause" for temporal isolation. Meanwhile, the dummy cell will block there until it is "deleted" by a "sync-grant" from the aggregator, issued once every M-slot frame. This realizes the heuristic of "pause" *until* the next frame to "resume reading". Note the only purpose of "sync-grant" is to delete blocking dummy cells; "sync-grant" does not fetch cells.

Suppose output O grants Q for v times in each M-slot frame at slot s_{i_0}, s_{i_1}, ..., $s_{i_{v-1}}$. If O is an aggregator for queue Q, then O *sync-grants* Q in the first granting slot s_{i_0}; and *aggregator-data-grants* Q in all other granting slots $s_{i_1} \sim s_{i_{v-1}}$. If O is not an aggregator for Q, then $s_{i_0} \sim s_{i_{v-1}}$ are all for *normal-data-grants*.

Also, for an output O, every $(M+1)$ cell-time, it needs to multicast a dummy cell to $\mathcal{M}_O$ (this is triggered by the combined effects of variable i, $newVFrame$, and $lastVFrameStartsAt$ in Fig. 8 line 2, 7, 9, 10, 17, 18, 19). This means every M-slot frame, at the most one slot will be sacrificed for outputing dummy cell, instead of data cell. This is remedied by allocating one more slot for normal- or aggregator-data-grant during the resource planning and admission control stage (see Section IV-C-"Resource Planning Method"). Note if the sacrificed cell happens to be a sync-grant, there will be no negative effects. Because sync-grant's job is only to delete blocking dummy cells in the *previous* hop; sync-grant does *not* forward cells.

C. Analysis

Without loss of generality, in the following, we use aggregate F in Fig. 4 as an example for our analysis.

We first define some notations and conventions:

Definition 1: We call two dummy cells c_0 and c_1 "*consecutive dummy cells*" from output O, iff O does not output any other dummy cells between outputting c_0 and c_1.

```
1. void OnCellTimeStart() {
2.     $i \leftarrow ((i+1) \mod M)$;
3.     //suppose $s_i^O.queue$ refers to $Q(I,O,F)$;
4.     if ($s_i^O.role$ == "aggregator" and $s_i^O.sync$ == true) {
5.         sync-grant $Q(I,O,F)$;
6.     }
7.     if ($newVFrame$) {
8.         create and output a multicast dummy cell $c$ to $\mathcal{M}_O$, and at the
           segregator of each $X \in \mathcal{M}_O$, $c$ will be enqueued as if it will
           further multicast to $\mathcal{S}_X$ ($X$'s successor aggregates);
           //Note1: $\mathcal{M}_O \overset{def}{=} \{X|O$ is the aggregator for aggregate $X\}$.
           //Note2: The above life cycle of $c$ is better explained by Fig. 7.
           //Note although $c$ will enter each queue $X$ segregates to, $c$ will
           //be deleted at the corresponding queue's header, see Fig. 9 line 6.
9.         $newVFrame \leftarrow$ false;
10.        $lastVFrameStartsAt \leftarrow i$;
11.    }else{
12.        if ($s_i^O.role$ == "aggregator" and $s_i^O.sync$ == false ) {
13.            fetch and output $Q(I,O,F)$'s header
               cell (if any) with an aggregator-data-grant;
14.        }else if ($s_i^O.role \neq$ "aggregator"){
15.            fetch and output $Q(I,O,F)$'s header
               cell (if any) with a normal-data-grant;
16.        } //else do nothing (particularly, output nothing)
17.        if ($i$ == $lastVFrameStartsAt$) {
18.            $newVFrame \leftarrow$ true;
19.        }
20.    }
21. }
```

Fig. 8. OnCellTimeStart procedure, called at the beginning of each cell-time by an output port O. Global variables i, $lastVFrameStartsAt$, and $newVFrame$ are initialized to $(M-1)$, $(M-1)$, and **true** respectively.

```
1. void OnGranted($g$) {
2.     //suppose $g.queue$ refers to $Q(I,O,F)$;
3.     if ($Q(I,O,F)$ is empty) return;
4.     Let $c$ refer to $Q(I,O,F)$'s header cell;
5.     if ($g.type$ == sync-grant) {
6.         if ($c$ is a dummy cell) delete $c$; //else do nothing ("pause")
7.     }else if ($g.type$ == aggregator-data-grant){
           if ($c$ is not a dummy cell) send $c$ to $O$; //else do nothing ("pause")
8.     }else /* $g.type$ == normal-data-grant */ send $c$ to $O$;
9. }
```

Fig. 9. OnGranted procedure, called at the beginning of each cell-time by an input iff it receives a grant g: $g.queue$ is the queue granted; $g.type$ is one of sync-grant, aggregator-data-grant, or normal-data-grant.

Special care should be taken at the ends of flows. Without loss of generality, suppose input I_0 in Fig. 4 connects to a source end computer, which enqueues flow f into $Q(I_0, O_0, F' \cap F)$. Then every $(M+1)$ cell-time, the source end computer shall enqueue a dummy cell into $Q(I_0, O_0, F' \cap F)$. To facilitate narration, we define the following:

Definition 2: Suppose between enqueueing two consecutive dummy cells, the source end computer enqueues $\tilde{n}_f^{src}$ cells of f into $Q(I_0, O_0, F' \cap F)$. We denote

$$\tilde{N}_f^{src} \overset{def}{=} \sup\{\tilde{n}_f^{src}\}. \qquad (3)$$

We say that flow f has a *worst case source end virtual traffic load of $\tilde{N}_f^{src}$ cell/v-frame.*

Here we use "$\sim$" to indicate the corresponding parameter is related to certain "virtual" concepts, such as "virtual frame".

The definition also tells us how to measure/specify $\tilde{N}_f^{src}$. For example, if our system's v-frame size is $(M+1) = 1001$ (cell/v-frame), then the source end computer enqueues a dummy cell every 1001 cell-time. If the source end computer NEVER enqueues more than 9 cells of f between enqueueing dummy cells, then $\tilde{N}_f^{src} = 9$ (cell/v-frame).

Also, from now on, unless explicitly noted, let us assume *the default unit for time and data are "second" and "cell" respectively.*

We use $\tau^{(i)}$ to denote the cell-time duration of output O_i ($i = 0, 1, \ldots$) in Fig. 4. Therefore, O_i's M-slot frame duration $P^{(i)} = M\tau^{(i)}$. We admit the existence of clock drift, and denote the minimum and maximum cell-time duration of all switches in the system as τ_{min} and τ_{max} respectively. Correspondingly, the minimum and maximum M-slot frame duration are $P_{min} = M\tau_{min}$ and $P_{max} = M\tau_{max}$.

We use $\tilde{w}(f, Q)$ to denote the maximum number of flow f's cells arriving at queue Q between the arrival of any two consecutive dummy cells. Intuitively, this is the worst case traffic load of f enqueued into Q in each v-frame. Hence $\tilde{w}(f, Q)$'s unit is *cell/v-frame*. Correspondingly, let $v(O, Q)$ (cell/frame) denote the number of slots that output O schedules to grant Q in each frame. Note, if O is an aggregator for Q, then the first slot is for sync-grant, the other $v(O, Q) - 1$ slots are for aggregator-data-grant. If O is not an aggregator for Q, then all $v(O, Q)$ slots are for normal-data-grant.

Knowing the traffic load is the first step to plan resource allocation. Therefore, we need the following lemma:

Lemma 1 (Burstiness Bound): Suppose flow f's worst case source end virtual traffic load (see Definition 2) is $\tilde{N}_f^{src}$ cell/v-frame. Suppose there are M slots per frame (i.e. $M+1$ slots per v-frame). If output O is the $\tilde{h}$th ($\tilde{h} = 1, 2, \ldots$) aggregator that f encounters after leaving its source end computer (excluding the source end computer). Then regardless of τ_{min} and τ_{max}, between any two consecutive dummy cells (i.e., in each v-frame) outputted from O, there are no more than $2\tilde{N}_f^{src}$ cells of f if $\tilde{h} = 1$; and no more than $(2\tilde{h} - 1)\tilde{N}_f^{src}$ cells of f if $\tilde{h} \geq 2$ and $M \geq (2\tilde{h} - 1)$.

Proof: Please see Appendix A. ■

We have five important remarks/observations on Lemma 1:

Firstly, in almost all practical cases, $M >> (2\tilde{h} - 1)$. Hence we are not interested in cases where $\tilde{h} \geq 2$ and $M < (2\tilde{h} - 1)$.

Secondly, Lemma 1 is NOT about overflow or delay; rather, it is only about burstiness: the possible number of f's cells between any consecutive dummy cells enqueued. The fact that Lemma 1 does not involve τ_{min} and τ_{max}, the shortest and longest cell-time duration of all switches, implies our real-time aggregate can bound burstiness growth and infection regardless of clock-drift.

Thirdly, however, clock-drift may still cause queue overflow, which means infinite E2E delay. Therefore, the delay bound analysis (see Theorem 2 and 3) will still involve τ_{min} and τ_{max}. In fact, we will see Theorem 2 and 3 give a sufficient condition involving τ_{min} and τ_{max} that bounds E2E delay, and hence avoids queue overflow.

Fourthly, Lemma 1 also tells that if f passes $\tilde{h}$ hops of real-time aggregates, then f's burstiness is bounded by $\mathcal{O}(\tilde{h})$. In many cases (see Section V), we can configure a network so that $\tilde{h} = \mathcal{O}(\log_2 h)$, where h is the number of physical links that f passes. Therefore, the burstiness of f in such networks is controlled by $\mathcal{O}(\log_2 h)$. Or if we always configure a single real-time aggregate from source to destination end of f, then the burstiness is controlled by $\mathcal{O}(1)$.

Fifthly, Lemma 1 tells us how to calculate

$$\tilde{N}_f^{(0)} \overset{def}{=} \tilde{w}(f, Q(I_0, O_0, F' \cap F)) \text{ (cell/v-frame)} \quad (4)$$

for any $f \in F' \cap F$ (where F is the aggregate of concern, and $F' \in \mathcal{P}_F$). For example, if in Fig. 4, input I_0 connects to source end computer that enqueues flow f into $Q(I_0, O_0, F' \cap F)$ with $\tilde{N}_f^{src} = 9$ (cell/v-frame); then $\tilde{N}_f^{(0)} \overset{def}{=} \tilde{w}(f, Q(I_0, O_0, F' \cap F)) = \tilde{N}_f^{src} = 9$ (cell/v-frame); and each v-frame outputed from O_0 contains no more than $2\tilde{N}_f^{src} = 18$ cells of flow f, which also means $\tilde{w}(f, Q(I_k, O_k, F \cap F'')) = 18$ (cell/v-frame). So on and so forth.

With all $\tilde{N}_f^{(0)} = \tilde{w}(f, Q(I_0, O_0, F' \cap F))$ at O_0 (see Fig. 4) known, we can plan resources, analyze E2E delay bound, test schedulability, and create TDMA schedules as follows.

Resource Planning Method O_0 shall allocate

$$\begin{aligned} & v(O_0, Q(I_0, O_0, F' \cap F)) \\ = \ & 3 + \sum_{\forall f \in F' \cap F} \tilde{N}_f^{(0)} \overset{def}{=} \tilde{N}_{F' \cap F}^{(0)} \text{ (cell/frame)} \end{aligned} \quad (5)$$

to grant $Q(I_0, O_0, F' \cap F)$. Note the first slot is to sync-grant; the other two additional slots are for over-provisioning, whose meaning will become clear during the analysis of Theorem 2 and 3; and $\tilde{N}_{F' \cap F}^{(0)}$ is a notational shortcut.

Eq. (5) means O_0 totally allocates

$$\tilde{N}_F^{(0)} \overset{def}{=} \sum_{\forall F' \in \mathcal{P}_F} \tilde{N}_{F' \cap F}^{(0)} \text{ (cell/frame)} \quad (6)$$

for aggregate F. Subsequently, O_i ($i = 1 \sim k-1$) shall allocate

$$v(O_i, Q(I_i, O_i, F)) = \tilde{N}_F^{(0)} + 2 \overset{def}{=} \tilde{N}_F^{(1)} \text{ (cell/frame)} \quad (7)$$

for aggregate F. Note the two additional slots are for over-provisioning. Also note since O_i is not the aggregator for F, all allocated slots are for normal-data-grant.

Lemma 1 tells us how to calculate $\tilde{N}_f^{(0)}$ ($\forall f \in F' \cap F$, where F is the aggregate of concern, and $F' \in \mathcal{P}_F$). Given all $\tilde{N}_f^{(0)}$s, Eq. (5) $\sim$ (7), tell us how many slots per frame to grant aggregate F along its path.

Next we will see how to calculate delay bounds.

Real-Time Delay Bounds Still, without loss of generality, we refer to Fig. 4, and study a flow f that joins aggregate F at O_0 from $Q(I_0, O_0, F' \cap F)$, and leaves F at I_k from queue $Q(I_k, O_k, F \cap F'')$.

For a cell c of flow f, let $L_f^{(0)}$, $L_f^{(k-1)}$, and $L_f^{(k)}$ denote the time that c leaves O_0, O_{k-1}, and O_k respectively. We still use $\tau^{(k)}$ to denote the cell-time duration (second) of switch k, $P^{(k)} \overset{def}{=} M\tau^{(k)}$; use τ_{min} and τ_{max} to denote the shortest and longest cell-time duration of all switches, $P_{min} \overset{def}{=} M\tau_{min}$ and $P_{max} \overset{def}{=} M\tau_{max}$. Note $\tau_{max} - \tau_{min}$ implies the extent of clock drift in the switched networks. Then we can claim the following:

Theorem 2 (Inner Aggregate Delay Bound): Suppose we allocate resources per Eq. (5) $\sim$ (7). If $\tilde{N}_F^{(1)} < M$ and $\tau_{max} - \tau_{min} < \frac{\tau_{min}}{M}$, then

$$L_f^{(k-1)} - L_f^{(0)} \le D_F^{(1 \sim k-1)}$$
$$\overset{def}{=} (k-1)(M - \tilde{N}_F^{(0)})\tau_{max} + \frac{\tilde{N}_F^{(0)} P_{max}}{\tilde{N}_F^{(0)} + 1} \le kP_{max}, \quad (8)$$

where $\tilde{N}_F^{(0)}$ and $\tilde{N}_F^{(1)}$ are defined in Eq. (6) and (7) respectively.

Proof: Please see Appendix B. ■

Theorem 3 (Inter Aggregate Delay Bound): Suppose we allocate resources per Eq. (5) $\sim$ (7). If $\tilde{N}_F^{(1)} < M$ and $\tau_{max} - \tau_{min} < \frac{\tau_{min}}{M}$, then

$$\begin{aligned} L_f^{(k)} - L_f^{(0)} &\le D_F^{(1 \sim k-1)} + 3P^{(k)} + 2\tau^{(k)} \quad (9) \\ &\le (k+3)P_{max} + 2\tau_{max}, \quad (10) \end{aligned}$$

where $D_F^{(1 \sim k-1)}$ is given in Eq. (8).

Proof: Please see Appendix C. ■

Note the preconditions in Theorem 2 and 3 define the constraint on all clock drifts between switches: $\tau_{max} - \tau_{min} < \tau_{min}/M$. As long as this constraint is met, we can have solid delay bounds in spite of the existence of clock drifts.

By applying Theorem 3 along flow's path, we can calculate flow's E2E delay bound (note the 0th aggregate is the source end computer and the input port/queue it connects to).

Switch Schedulability and Scheduling Given the flows in the network, their worst case source end virtual traffic loads (i.e., $\tilde{N}_f^{src}$), and their routing plans among real-time aggregates, Lemma 1 and Eq. (5)$\sim$(7) can decide how many slot/frame each output shall grant each input. We can then reuse Theorem 1 to test schedulability; and reuse the corresponding polynomial time scheduling algorithm described in [1] to derive the schedule.

V. Evaluation

We evaluate the efficiency of our real-time aggregate design in networks of TCRT switches.

Specifically, the physical link layout of our evaluated networks takes the form of grid. A grid of edge length E consists of $(E+1) \times (E+1)$ TCRT switches. Switches are deployed in a two-dimensional plane at coordinates (x, y) (where $x = 0, 1, 2, \ldots, E$ and $y = 0, 1, 2, \ldots, E$). For simplicity, we use (x, y) to denote the TCRT switch at coordinate (x, y). Switch (x, y) has a directional physical link to connect it to each of its one hop neighbors (here "one hop" means "geographical distance of one"). Fig. 10 (a) illustrates the physical link layout of a grid of 4×4 (i.e., $E = 4$).

Given an aforementioned grid network of $E \times E$, we then overlay $(1 + \lfloor \log_2 E \rfloor)$ layers of aggregates upon the network. The Lth ($L = 0, 1, \ldots, \lfloor \log_2 E \rfloor$) layer of aggregates also form a grid, which connects those switches whose x, y coordinates are both multiples of 2^L. For example, Fig. 10(b) is the aggregate layout of the 4×4 network of Fig. 10(a).

We evaluate five grid networks, where $E = 1, 2, 4, 8, 16$ respectively. In each network, there are $(E+1) \times (E+1)$ TCRT switches. Each input/output of these switches is of capacity

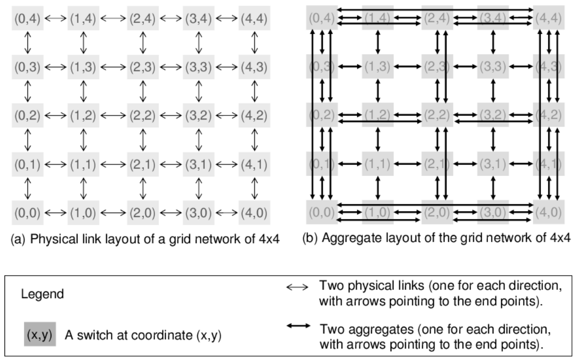

(a) Physical link layout of a grid network of 4x4
(b) Aggregate layout of the grid network of 4x4
Legend
(x,y) A switch at coordinate (x,y)
Two physical links (one for each direction, with arrows pointing to the end points).
Two aggregates (one for each direction, with arrows pointing to the end points).

Fig. 10. Grid Network of TCRT Switches: Physical Link Layout and Aggregate Layout

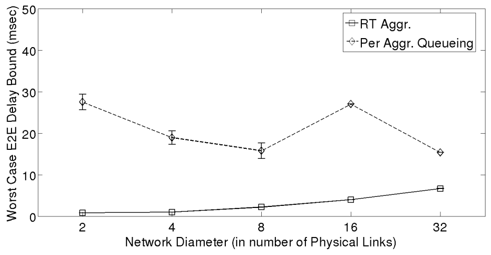


Fig. 11. Worst case E2E delay bound statistics (each dot is the mean of the corresponding 100 trials; error bars show the 95% confidence range).

$10 \sim 10.0040016$Gbps (the range is due to clock drift between different switches, i.e., the clock period of different switches are not exactly the same, hence the time used to transmit one bit are not exactly the same). For convenience, we assume each cell is 500 bit (instead of the de facto standard of 512 bit), $M = 2000$ slot/frame. These result in $\tau_{max} = 50$ns, $\tau_{min} = 49.98$ns, $P_{max} = 0.1$msec, and $P_{min} = 0.09996$msec, which complies with [6]'s suggestions: the frame duration is orders of magnitude less than typical real-time tasks' periods (which are typically $>>$ 1ms [23][6][24][25]).

We compare two flow aggregate methods: real-time aggregate and per-aggregate queueing. For each aggregate method, we run 100 trials. In each trial, we add into the network randomly generated periodical real-time flows: 90% of them are sensing/actuating traffic with worst case source end virtual traffic load of $\tilde{N}_f^{src} = 1$ cell/v-frame (in other words, 1 cell every $(M+1)$ cell-time; based on the aforementioned configuration parameters, this corresponds to a constant data rate of about 5Mbps, enough to encapsulate a typical real-time control flow [23][6][24][25]); and the rest are video flows with $\tilde{N}_f^{src} = 16$ cell/v-frame (in other words, 16 cells every $(M+1)$ cell-time; based on the aforementioned configuration parameters, this corresponds to a constant data rate of about 80Mbps, enough to encapsulate a typical real-time video flow [23][6][24][25]). The source and destination ends are randomly picked from the switches in the network according to uniform distribution (here, we are not associating our simulation with any specific networked real-time applications; without the application specific knowledge, uniform distribution is a natural and generic enough choice, just as most software libraries' default random number generator assumes uniform distribution on $[0, 1)$). Once source/destination ends are picked, the flow is routed via the aggregate layout of the network using Dijkstra's shortest path algorithm (each aggregate is considered to be of length "1" in the Dijkstra path planning).

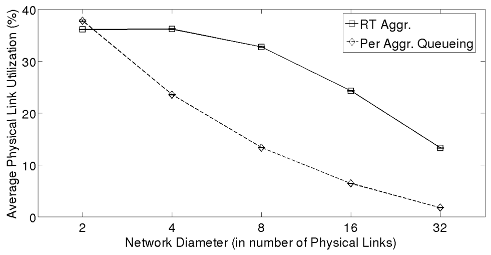


Fig. 12. Average physical link utilization statistics (each dot is the mean of the corresponding 100 trials; error bars show the 95% confidence range).

Under real-time aggregate method, in each switch that the route passes, the corresponding output allocates resource according to Section IV-C-"Resource Planning Method". Then we test the schedulability of such resource allocation according to Section IV-C-"Switch Schedulability and Scheduling"; and calculate the flow's E2E delay bound with Theorem 2 and 3. If every switch on the route can afford the resource allocation and the E2E delay bound is within 50 msec (we choose 50msec because through literature survey, it is a commonly acceptable E2E delay bound for networked real-time applications [23][6][24][25]), the flow is admitted.

Under per-aggregate queueing method, though an efficient schedulability test and tight E2E delay bound are still open problems, we find that we can apply the DiffServ math model [20][21][22] to give an sufficient schedulability test and corresponding E2E delay bound. Specifically, every aggregate queue is regarded as a DiffServ queue; and according to [20][21][22], we allocate each queue a number of TDMA slots so that the service rate is no less than the total source end arrival rates of the queue's member flows. Then we can apply Theorem 2.4.2 of [20] to carry out a sufficient schedulability test and derive the corresponding E2E delay bound.

As mentioned before, given the network and aggregate method, we carry out 100 trials. In each trial, we keep adding real-time flows into the network until no more flow can be admitted. Each admitted flow corresponds to an E2E delay bound. Then we calculate the worst case E2E delay bound of all admitted flows, and the average physical link utilization. The average physical link utilization is calculated as follows: a flow with a worst case source end traffic load of $\tilde{N}_f^{src}$ cell per $(M+1)$ cell-time contributes a utilization of $\tilde{N}_f^{src}/(M+1)$ for each physical link it passes.

In summary, in each trial, we can derive a worst case (i.e., maximum) E2E delay bound of all flows admitted, and an average physical link utilization. The statistics of the two metrics are plotted in Fig. 11 and Fig. 12. In the figures, each dot represents the mean of the corresponding 100 trials, while the error bars represent the corresponding mean value's 95% confidence interval. Note some mean values are very accurate, resulting nearly overlapping upper and lower error bars.

According to Fig. 11, with no less than 95% confidence, we can claim all the flows admitted have E2E delay bound below the real-time deadline requirement of 50msec, a commonly acceptable E2E delay bound for networked real-time applications [23][6][24][25]. What is more, real-time aggregate achieves much better worst case E2E delay bound (all mean values are below 10msec) than per-aggregate queueing (whose mean values fluctuate from 15msec to even near 30msec).

According to Fig. 12, with no less than 95% confidence, we can claim the following. First, under both aggregate methods, the schedulable average physical link utilization decreases as the network diameter H increases. This is because as H increases, flows on average travels more hops of aggregates. Their burstiness increases each time it joins a new aggregate, which degrades schedulability. Second, real-time aggregate achieves much higher schedulable average physical link utilization than per-aggregate queueing. When $H = 4$, 8, 16, and 32, the former is respectively 1.5, 2.5, 3.8, and 7.6 times that of the latter. Third, note our real-time aggregate performance analysis already takes into consideration the dummy cell overhead. The results show that even take into consideration of dummy cell overhead, real-time aggregate still achieves much better schedulable average physical link utilization and worst case E2E delay bound than per-aggregate queueing.

VI. Related Work

In the real-time community. There are three sets of highly relevant works.

The first set is Pinwheel scheduling [26]. Though also based on TDMA, Pinwheel scheduling assumes one CPU per node or independent multiprocessors, and mainly focuses on finding the optimal TDMA scheduling period. In contrast, we have N outputs contending for N inputs in parallel within a same switch, and focus on finding a contention free crossbar schedule (matching).

The second set is hierarchical scheduling [27]. However, hierarchical scheduling are about CPUs. Though recently, Santos et al. [28] proposes using hierarchical scheduling for *output queueing* real-time switches, how to migrate the hierarchical CPU task model to the popular *input queueing* TCRT switch architecture without introducing much modifications is still a non-trivial open problem, not to mention supporting aggregates and clock drift.

The third set is non-work-conserving switch scheduling (e.g., Stop-and-Go[29]). But these schemes also assume *output queueing* instead of *input queueing* crossbar switch architecture, which only becomes predominant more recently. In addition, to our best knowledge, the existing non-work-conserving switch scheduling (such as Stop-and-Go) schemes are not about flow aggregation, neither do they cover the burstiness growth and infection problems caused by clock drift between switches.

In the networking community, first, we notice that our multi-hop real-time switched networks bear drastically different design philosophy, traffic features, and network coverage compared to those of Internet. Internet prefers flow aggregation (shared queues) to flow isolation (e.g., per-flow routing) in pursuit of scalability. Unlike Internet, mission critical real-time networks/applications need flow isolation, zero packet loss, and hard E2E delay bound to guarantee dependability.

It is worth noting that in earlier years, the Internet community did propose many per-flow queueing zero packet loss QoS schemes, e.g., WFQ [30]. However, these schemes mostly assume output queueing, and need time-stamp based packet sorting. Due to implementation and runtime complexity, they are not widely implemented by manufacturers.

It is the other branch of efforts on ATM and telephone switches that finally evolves into today's widely adopted input queueing TCRT switch architecture [7][8][1][9][10][11], which this paper is about (see Section II).

Another set of architectures supporting per flow queueing and zero packet loss is the real-time LANs, a.k.a. fieldbuses [31][32]. But their support for real-time mostly assumes shared medium, hence is for LANs instead of multi-hop networks. In fact, to merge fieldbuses to multi-hop switched networks would need the aforementioned real-time switches [11][9][1].

TTEthernet [33] is a fieldbus standard that considers multi-hop real-time support. However, TTEthernet standard assumes the underlying multi-hop switched network already guarantees bounded E2E delay. The standard itself does not specify the detailed design of the switches. Therefore, our real-time switch/aggregate design can complement TTEthernet by providing a detailed design that meets its core assumption.

There are other real-time fieldbus standards that involve support for real-time flow aggregate. IEC 61784 [34] defines a set of *communication profiles* on flow aggregate. However, it does not specify how to realize such profiles. In other words, IEC 61784 is an open standard: our TCRT switch real-time aggregate mechanism provides one way to realize the IEC 61784 real-time flow aggregate communication profile.

There are also other related work on how to realize and analyze real-time flow aggregate.

MPLS [35] is a flow labeling mechanism for aggregation based routing. However, MPLS is a Layer 2.5 mechanism, which is above Layer 2 (the Data Link Layer); while the real-time aggregate design of this paper is a strictly Layer 2 design. In other words, real-time aggregate can serve MPLS.

Sun and Shin proposed *Guaranteed Rate* (GR) server based flow aggregates with bounded E2E delay in [12]. However, GR servers (e.g., WFQ [30]) are not widely implemented as they usually assume output queueing and need packet sorting.

In contrast, serving flow aggregates with FIFO is widely implemented due to its simplicity. This method is also known as DiffServ [22]. As Wang et al. [13] point out, DiffServ's schedulability and E2E delay bound are very susceptible to rogue bursty traffic, mainly due to lack of isolation in FIFO. A generic schedulability test and *tight* E2E delay bound are still open problems. However, there is a well-known *sufficient* schedulability test and corresponding E2E delay bound analysis framework developed by Boudec et al. [20][21]. This framework can be applied to per-aggregate queueing TCRT switched networks (in fact, per-aggregate queueing is the way to implement DiffServ on the TCRT switch architecture). In Section V, we used this DiffServ analysis framework to analyze the per-aggregate queueing performance, and compared it with that of real-time aggregate.

The IEEE 802.1 AVB task group has recently released the IEEE 802.1Qav specifications [36], which also proposes a flow aggregate mechanism for multi-hop switched networks. However, this mechanism is designed for output queueing work-conserving switch architecture with prioritized scheduling. In contrast, this paper's aggregate mechanism is designed for the input queueing non-work-conserving crossbar switch architecture with TDMA scheduling.

Finally, Scharbarg et al. [37] give a probabilistic E2E delay bound for aggregates in AFDX [4] switched networks. In contrast, this paper focuses on providing a deterministic E2E delay bound instead.

VII. Conclusion

In this paper, we proposed a novel flow aggregation (queue sharing) mechanism for the popular TCRT switches, which are widely recognized/adopted to build multi-hop real-time networks for integrating nowadays quickly expanding distributed real-time systems. The mechanism, called "real-time aggregates", exploits the TCRT switch's features and deploys spatial-temporal isolations to curb the burstiness growth and infection of aggregate's member flows. This allows us to derive closed form E2E delay bound and the corresponding resource-planning/admission-control strategies. Simulations show that real-time aggregates can guarantee short E2E delay bound and provide high utilization of the network resources.

VIII. Acknowledgement

The research project related to this paper in Hong Kong Polytechnic University (HKPolyU) is supported in part by Hong Kong RGC General Research Fund (GRF) PolyU 5245/09E, The HKPolyU Internal Competitive Research Grant (DA) A-PJ68, HKPolyU Newly Recruited Junior Academic Staff Grant A-PJ80, HK PolyU Fund for GRF Project Rated 3.5 (DA) grant A-PK46 (RGC No. PolyU 5300/10E), A-PL82 (RGC No. PolyU 5285/11E), and Department of Computing start up fund. Rong Zheng is supported in part by NSF CNS-0832084 and the HKPolyU Department of Computing Sabbatical Visitor Program. Xue Liu is supported by NSERC Discovery Grant 341823-07, FQRNT grant 2010-NC-131844, and CFI Leaders Opportunity Fund 23090. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of sponsors.

The authors thank Mr. Lixiong Chen and anonymous reviewers for their efforts/advice on improving this paper.

References

[1] Q. Wang *et al.*, "Adapting a main-stream internet switch architecture for multi-hop real-time industrial networks," *IEEE TII*, vol. 6, no. 3, 2010.

[2] *How space shuttles work.* science.howstuffworks.com/space-shuttle9.htm.

[3] *Electro-Optical DAS.* http://www.youtube.com/user/F35JSFVideos#p/u/7/CwvnhFgzIKI.

[4] *Aircraft Data Network Part 7: Avionics Full Duplex Switched Ethernet (AFDX) Network.* ARINC 664, Jun. 2005.

[5] M. Inc. and T. Shanley, *InfiniBand Network Architecture.* Addison-Wesley Professional, Nov. 2002.

[6] Q. Wang, "Real-time and embedded systems building blocks for cyber-physical systems," Ph.D. dissertation, CS Dept., UIUC, 2008.

[7] L. Rao *et al.*, "Analysis of TDMA Crossbar Real-Time Switch Design for AFDX Networks," *Proc. of INFOCOM'12*, Mar. 2012.

[8] L. Chen *et al.*, "A real-time multicast routing scheme for multi-hop switched fieldbuses," *Proc. of INFOCOM'11*, pp. 3209–3217, 2011.

[9] F. Dopatka *et al.*, "Design of a realtime industrial ethernet network including hot-pluggable asynchronous devices," *Proc. of IEEE ISIE'07*, 2007.

[10] C.-S. Chang, W.-J. Chen, and H.-Y. Huang, "On service guarantees for input buffered crossbar switches: a capacity decomposition approach by birkhoff and von neumann," *IEEE IWQoS'99*, pp. 79–86, 1999.

[11] Y.-W. Leung and T.-S. Yum, "A TDM-based multibus packet switch," *IEEE Trans. on Communications*, vol. 45, no. 7, pp. 859–866, Jul. 1997.

[12] W. Sun and K. G. Shin, "End-to-end delay bounds for traffic aggregates under guaranteed-rate scheduling algorithms," *IEEE/ACM TON*, vol. 13, no. 5, pp. 1188–1201, Oct. 2005.

[13] S. Wang *et al.*, "Providing absolute differentiated services for real-time applications in static-priority scheduling networks," *IEEE/ACM TON*, vol. 12, no. 2, 2004.

[14] I. Stoica, S. Shenker, and H. Zhang, "Core-stateless fair queueing: achieving approximation fair bandwidth allocations in high speed networks," *Proc. of ACM SIGCOMM'98*, pp. 118–130, Oct. 1998.

[15] S. Gopalakrishnan *et al.*, "Switch scheduling and network design for real-time systems," in *Proc. of IEEE RTAS 2006*, Apr. 2006.

[16] N. McKeown, "The iSLIP scheduling algorithm for input-queued switches," *IEEE/ACM TON*, vol. 7, no. 2, Apr. 1999.

[17] Q. Wang *et al.*, "A switch design for real-time industrial networks," in *Proc. of IEEE RTAS 2008)*, Apr. 2008, pp. 367–376.

[18] A. D. Kshemkalyani and M. Singhal, *Distributed Computing: Principles, Algorithms, and Systems.* Cambridge Univ. Press, Mar. 2011.

[19] J. A. Cobb, "Preserving quality of service guarantees in spite of flow aggregation," *IEEE/ACM TON*, vol. 10, no. 1, pp. 43–53, Feb. 2002.

[20] J.-Y. L. Boudec and P. Thiran, *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet.* Springer, 2001.

[21] A. Charny *et al.*, "Delay bounds in a network with aggregate scheduling," in *Intnl' Workshop on Quality of Future Internet Services*, 2000.

[22] L. L. Peterson and B. S. Davie, *Computer Networks: A System Approach*, 4th ed. Margan Kaufmann, 2007.

[23] R. J. Trew, Ed., *Proceedings of the IEEE: Special Issue Cyber-Physical Systems.* IEEE, Jan. 2012, vol. 100, no. 1.

[24] B. Fisher *et al.*, "Seeing, hearing, and touching: Putting it all together," *SIGGRAPH'04 Course*, 2004.

[25] M. Glencross *et al.*, "Exploiting perception in high-fidelity virtual environments," *SIGGRAPH'06 Course*, 2006.

[26] R. Holte *et al.*, "The pinwheel: A real-time scheduling problem," *Proc. of Annu. Hawaii Intl' Conf. on Sys. Sci.*, vol. 2, 1989.

[27] Z. Deng and J. W.-S. Liu, "Scheduling real-time applications in an open environment," *Proc. of IEEE RTSS'97*, 1997.

[28] R. Santos *et al.*, "Multi-level hierarchical scheduling in ethernet switches," *Proc. of EMSOFT'11*, pp. 185–194, 2011.

[29] S. Golestani, "A stop-and-go queueing framework for congestion management," *ACM SIGCOMM*, pp. 8–18, Sep. 1990.

[30] A. K. Parekh and R. G. Gallager, "A generalized processor sharing aproach to flow control in integrated services network: the single node case," *IEEE/ACM TON*, vol. 1, pp. 344–357, Jun. 1993.

[31] *Road Vehicles - Exchange of Digital Information - Controller Area etwork (CAN) for High-Speed.* ISO Std. 11898, 1993.

[32] S. Fischmeister *et al.*, "Hardware acceleration for conditional state-based communication scheduling on real-time ethernet," *IEEE Tran. on Industrial Informatics*, vol. 5, no. 3, 2009.

[33] *TTEthernet Specification.* TTTech Computertechnik AG, 2008.

[34] *Industrial communication networks - Profiles - Part 2: Additional fieldbus profiles for real-time networks based on ISO/IEC 8802-3.* IEC 61784-2 Ed. 2.0 b:2010, 2010.

[35] *MPLS Working Group, IETF.* http://www.ietf.org/html.charters/mpls-charter.html.

[36] *IEEE Standard 802.1Qav*, 2009.

[37] J.-L. Scharbarg *et al.*, "A probabilistic analysis of end-to-end delays on an afdx avionic network," *IEEE TII*, vol. 5, no. 1, 2009.

Appendix A
Proof of Lemma 1

Suppose output O's M-slot frame schedule consists of slot $s_0 \sim s_{M-1}$. In this section, we make a convention that an "M-

slot frame" or "frame" refers to the M consecutive cell-time *starting from slot* s_0.

To prove Lemma 1, let us first prove Lemma 2 and 3.

Lemma 2: During each M-slot frame starting from slot s_0, output O_0 in Fig. 4 can at the most fetch one v-frame from the header of $Q(I_0, O_0, F' \cap F)$.

Proof: In $Q(I_0, O_0, F' \cap F)$, every v-frame is delineated by two consecutive dummy cells. The two dummy cells can only be removed from $Q(I_0, O_0, F' \cap F)$ by sync-grant from O_0. Let $s_0 \sim s_v$ denote the slots in each frame that O_0 grants $Q(I_0, O_0, F' \cap F)$. Then according to Section IV-B-Grant, s_0 is the only sync-grant slot.

If at s_0, the header of $Q(I_0, O_0, F' \cap F)$ is a dummy cell, then the dummy cell is removed by the sync-grant; and till the end of the M-slot frame, O_0 can fetch at the most one v-frame from $Q(I_0, O_0, F' \cap F)$. O_0 cannot fetch more than one v-frame in the current frame, because it issues no more sync-grant to remove the next dummy cell in $Q(I_0, O_0, F' \cap F)$.

If at s_0, the header of $Q(I_0, O_0, F' \cap F)$ is not a dummy cell, then till the end of the M-slot frame, O_0 cannot issue any more sync-grant to remove the dummy cell leading the next v-frame. Therefore, in the current frame, O_0 can at the most get one v-frame from $Q(I_0, O_0, F' \cap F)$. ■

Lemma 3: For each k ($k \leq M$) consecutive v-frames outputed from O_0, let $\mathcal{D}$ denote their data cells that originally come from $Q(I_0, O_0, F' \cap F)$. Then the data cells of $\mathcal{D}$ come from at the most $(k+2)$ consecutive v-frames outputed from $Q(I_0, O_0, F' \cap F)$.

Proof: O_0 outputs k consecutive v-frames using $k(M+1)$ consecutive cell-time. Therefore, these k consecutive v-frames are outputed during at the most

$$\begin{aligned} & \left\lceil \frac{k(M+1)}{M} \right\rceil + 1 = \left\lceil k + \frac{k}{M} \right\rceil + 1 \\ = \quad & k + 2 \quad \text{(because } k \leq M\text{)} \end{aligned}$$

consecutive M-slot frames. According to Lemma 2, these $(k+2)$ M-slot frames can fetch at the most $(k+2)$ consecutive v-frames from $Q(I_0, O_0, F' \cap F)$. ■

Now we are ready to prove Lemma 1.

Case 1: $\tilde{h} = 1$. Since O outputs a v-frame every $M+1$ cell-time, the slots belong to at the most 2 consecutive M-slot frames of O. According to Lemma 2, the data contents can come from at the most 2 v-frames from previous hop (i.e., f's source end computer). That is, each v-frame outputed by O can contain at the most $2\tilde{N}_f^{src}$ cells of f.

Case 2: $\tilde{h} = 2$. Using similar analysis as Case 1, for each v-frame outputed from O, the data contents come from at the most 2 v-frames from previous aggregate. Suppose O' is the aggregator for previous aggregate. O' outputs 2 v-frames using $2M+2$ cell-time: $slot_0 \sim slot_{2M+1}$. These slots belong to either 3 or 4 consecutive M-slot frames of O'.

Suppose these slots belong to 4 consecutive frames: $frame_0 \sim frame_3$. Then $slot_{2M+1}$ and only $slot_{2M+1}$ belongs to $frame_3$, and $slot_{2M+1}$ must be $frame_3$'s first slot. Because an aggregator cannot do aggregator-data-grant before doing sync-grant during any frame, $slot_{2M+1}$ must not be an aggregator-data-grant. Therefore, $slot_{2M+1}$ cannot fetch any content of flow f. Therefore, only $frame_0 \sim frame_2$ can fetch contents of flow f.

Therefore, either way, the slots among $slot_0 \sim slot_{2M+1}$ that contain data from flow f come from at most 3 consecutive frames of O'. According to Lemma 2, the data contents come from at the most 3 v-frames from the previous hop (i.e., f's source end computer). That is, there can be at the most $3\tilde{N}_f^{src}$ cells of f.

Case 3: $\tilde{h} \geq 3$. For convenience, denote $O_{\tilde{h}} = O$. Suppose after leaving the source end computer, flow f passes aggregator $O_1, O_2, \ldots, O_{\tilde{h}}$ sequentially. Using similar as Case 2, for each v-frame outputed from $O_{\tilde{h}}$, the cells contain contents of f must come from at the most 3 consecutive v-frames outputed from $O_{\tilde{h}-2}$. Then we can recursively apply Lemma 3 till we reach the conclusion that the cells contain contents of f must come from at the most $3 + 2(\tilde{h} - 2) = 2\tilde{h} - 1$ consecutive v-frames from f's source end computer. That is, there can be at the most $(2\tilde{h} - 1)\tilde{N}_f^{src}$ cells of f.

Combining Case 1 $\sim$ 3, we prove the lemma. ■

Appendix B
Proof of Theorem 2

Our analysis may use two special functions of time t: *affine function* $\Lambda[\sigma, \rho](t)$ and *rate-delay function* $\Gamma[r, \delta](t)$ defined as follows:

$$\Lambda[\sigma, \rho](t) \overset{def}{=} \begin{cases} 0 & (t \leq 0) \\ \sigma + \rho t & (t > 0) \end{cases}; \tag{11}$$

$$\Gamma[r, \delta](t) \overset{def}{=} r[t - \delta]^+, \tag{12}$$

$$\text{where } [x]^+ \overset{def}{=} \begin{cases} 0 & (x \leq 0) \\ x & (x > 0) \end{cases}.$$

We can use network calculus to prove Theorem 2.

According to Eq. (6), O_0 allocates $\tilde{N}_F^{(0)}$ slots in each of its M-slot frame to serve aggregate F. Due to TDMA, these $\tilde{N}_F^{(0)}$ slots have fixed locations (i.e., indices) in the M-slot frame. Hence the arrival of aggregate F at $Q(I_1, O_1, F)$ conforms to arrival curve $\alpha_F^{(1)}(t)$:

$$\alpha_F^{(1)}(t) = \Lambda[\sigma_F^{(1)}, \rho_F^{(1)}](t), \tag{13}$$

$$\text{where} \quad \sigma_F^{(1)} = \tilde{N}_F^{(0)}, \tag{14}$$

$$\rho_F^{(1)} = \frac{\tilde{N}_F^{(0)}}{P_{min}}. \tag{15}$$

According to Theorem 1.4.6 (and its corollary) in [20], we can regard $O_1 \sim O_{k-1}$ as one single server in a black box, which serves $Q(I_1, O_1, F)$ with a service curve of

$$\beta_F^{(1 \sim k-1)}(t) = \Gamma[r_F^{(1 \sim k-1)}, \delta_F^{(1 \sim k-1)}](t), \tag{16}$$

where

$$r_F^{(1 \sim k-1)} = \frac{\tilde{N}_F^{(1)} - 1}{P_{max}}, \tag{17}$$

$$\delta_F^{(1 \sim k-1)} = (k-1)(M - (\tilde{N}_F^{(1)} - 1) + 1)\tau_{max}. \tag{18}$$

Note at the most 1 slot might be overtaken for sending dummy cells in an M-slot frame. Hence in Eq. (17) and (18) we use $(\tilde{N}_F^{(1)} - 1)$ instead of $\tilde{N}_F^{(1)}$.

Since

$$\begin{aligned}
& \tau_{max} - \tau_{min} < \frac{\tau_{min}}{M} \\
\Rightarrow \quad & MM(\tau_{max} - \tau_{min}) < M\tau_{min} \\
\Rightarrow \quad & M(P_{max} - P_{min}) < P_{min} \\
\Rightarrow \quad & \tilde{N}_F^{(0)}(P_{max} - P_{min}) < P_{min} \\
\Rightarrow \quad & \frac{\tilde{N}^{(1)} - 1}{P_{max}} > \frac{\tilde{N}_F^{(0)}}{P_{min}} \Rightarrow r_F^{(1\sim k-1)} > \rho_F^{(1)},
\end{aligned}$$

we can apply basic network calculus to get:

$$\begin{aligned}
& L_f^{(k-1)} - L_f^{(0)} \le D_F^{(1\sim k-1)} = \delta_F^{(1\sim k-1)} + \frac{\sigma_F^{(1)}}{r_F^{(1\sim k-1)}} \\
= \quad & (k-1)(M - \tilde{N}_F^{(1)} + 2)\tau_{max} + \frac{\tilde{N}_F^{(0)}}{\tilde{N}_F^{(1)} - 1} P_{max} \\
= \quad & (k-1)(M - \tilde{N}_F^{(0)})\tau_{max} + \frac{\tilde{N}_F^{(0)}}{\tilde{N}_F^{(0)} + 1} P_{max} \\
\le \quad & (k-1)M\tau_{max} + P_{max} \\
= \quad & (k-1)P_{max} + P_{max} = kP_{max}. \qquad \blacksquare
\end{aligned}$$

APPENDIX C
PROOF OF THEOREM 3

Let $\check{c}_i$ denote the ith ($i = 0, 1, \ldots$) dummy cell outputed from O_0 since system starts. Let $\check{L}_i^{(0)}$, $\check{L}_i^{(k-1)}$, and $\check{L}_i^{(k)}$ denote the time that $\check{c}_i$ leaves O_0, O_{k-1}, and O_k respectively. Then according to Theorem 2, we have

$$\check{L}_i^{(k-1)} - \check{L}_i^{(0)} \le D_F^{(1\sim k-1)}, \tag{19}$$

where $D_F^{(1\sim k-1)}$ is defined in Eq. (8).

We further have the following lemma:

Lemma 4:

$$\check{L}_i^{(k)} - \check{L}_i^{(0)} \le D_F^{(1\sim k-1)} + 2P^{(k)} + \tau^{(k)}, \tag{20}$$

where $P^{(k)}$ and $\tau^{(k)}$ are the M-slot frame and cell-time duration (in the unit of "second") of O_k.

Proof: We can prove by induction.

Eq. (20) holds for $i = 0$. Suppose Eq. (20) holds for some $i \ge 0$, let us prove it also holds for $i + 1$.

Case 1: $\check{L}_i^{(k)} \ge \check{L}_{i+1}^{(k-1)}$

In this case, at $\check{L}_i^{(k)}$, $\check{c}_{i+1}$ already arrives at $Q(I_k, O_k, F \cap F'')$, which also means all the v-frame contents between $\check{c}_i$ and $\check{c}_{i+1}$ are already backlogged before $\check{c}_{i+1}$ in $Q(I_k, O_k, F \cap F'')$.

Note O_k goes through a full M-slot frame from $\check{L}_i^{(k)} - \tau^{(k)}$ to $\check{L}_i^{(k)} - \tau^{(k)} + P^{(k)}$, issuing enough data-grant to clear the v-frame data cells backlogged in front of $\check{c}_{i+1}$.

Therefore when O_k issues sync-grant at $\check{L}_i^{(k)} - \tau^{(k)} + P^{(k)}$, $\check{c}_{i+1}$ is cleared in that cell-time, which means

$$\begin{aligned}
& \check{L}_{i+1}^{(k)} = \check{L}_i^{(k)} - \tau^{(k)} + P^{(k)} + \tau^{(k)} = \check{L}_i^{(k)} + P^{(k)} \\
\le \quad & \check{L}_i^{(0)} + D_F^{(1\sim k-1)} + 2P^{(k)} + \tau^{(k)} + P^{(k)} \text{ (see Eq. (20))} \\
\le \quad & \check{L}_{i+1}^{(0)} - P^{(k)} + D_F^{(1\sim k-1)} + 3P^{(k)} + \tau^{(k)} \qquad (21) \\
= \quad & \check{L}_{i+1}^{(0)} + D_F^{(1\sim k-1)} + 2P^{(k)} + \tau^{(k)},
\end{aligned}$$

where Ineq. (21) is because

$$\begin{aligned}
& \tau_{max} - \tau_{min} < \frac{\tau_{min}}{M} \Rightarrow P_{min} + \tau_{min} > P_{max} \\
\Rightarrow \quad & P^{(0)} + \tau^{(0)} > P^{(k)} \\
\Rightarrow \quad & \check{L}_{i+1}^{(0)} - (P^{(0)} + \tau^{(0)}) \le \check{L}_{i+1}^{(0)} - P^{(k)} \\
\Rightarrow \quad & \check{L}_i^{(0)} = \check{L}_{i+1}^{(0)} - (P^{(0)} + \tau^{(0)}) \le \check{L}_{i+1}^{(0)} - P^{(k)}.
\end{aligned}$$

Case 2: $\check{L}_i^{(k)} < \check{L}_{i+1}^{(k-1)}$

In this case, at $\check{L}_{i+1}^{(k-1)}$, $\check{c}_{i+1}$ arrives at $Q(I_k, O_k, F \cap F'')$; and since $\check{c}_i$ has already left, at the most one v-frame of data cells are backlogged before $\check{c}_{i+1}$ in $Q(I_k, O_k, F \cap F'')$.

Suppose t is the first time O_k sync-grants $Q(I_k, O_k, F \cap F'')$ after $\check{L}_{i+1}^{(k-1)}$, then

$$t \le \check{L}_{i+1}^{(k-1)} + P^{(k)}. \tag{22}$$

By $t + P^{(k)}$, all the v-frame data cell backlog in front of $\check{c}_{i+1}$ are cleared. Therefore, $\check{c}_{i+1}$ leaves O_k by $t + P^{(k)} + \tau^{(k)}$ at the latest. That is

$$\begin{aligned}
& \check{L}_{i+1}^{(k)} \le t + P^{(k)} + \tau^{(k)} \\
\le \quad & \check{L}_{i+1}^{(k-1)} + P^{(k)} + P^{(k)} + \tau^{(k)} \text{ (see Eq. (22))} \\
\le \quad & \check{L}_{i+1}^{(0)} + D_F^{(1\sim k-1)} + 2P^{(k)} + \tau^{(k)} \text{ (see Eq. (19))}.
\end{aligned}$$

Combining Case 1 and 2, Eq. (20) also holds for $i + 1$. $\blacksquare$

Proof of Theorem 3: Without loss of generality, suppose cell c of flow f is sandwiched between dummy cell $\check{c}_i$ and $\check{c}_{i+1}$, then

$$\check{L}_i^{(0)} \le L_f^{(0)} \le \check{L}_{i+1}^{(0)}. \tag{23}$$

Due to Theorem 2,

$$\begin{aligned}
& L_f^{(k-1)} \le L_f^{(0)} + D_F^{(1\sim k-1)} \\
\le \quad & L_f^{(0)} + D_F^{(1\sim k-1)} + 2P^{(k)} + \tau^{(k)} \\
\overset{def}{=} \quad & t \quad \text{(denoted as } t \text{ for convenience)}.
\end{aligned}$$

That is, c arrives at $Q(I_k, O_k, F \cap F'')$ by t at the latest. On the other hand,

$$\begin{aligned}
& \check{L}_i^{(k)} \le \check{L}_i^{(0)} + D_F^{(1\sim k-1)} + 2P^{(k)} + \tau^{(k)} \text{ (see Lemma 4)} \\
& \le L_f^{(0)} + D_F^{(1\sim k-1)} + 2P^{(k)} + \tau^{(k)} = t. \text{ (see Ineq. (23))}
\end{aligned}$$

That is, dummy cell $\check{c}_i$ leaves $Q(I_k, O_k, F \cap F'')$ by t. Therefore, by t, even if c is still in $Q(I_k, O_k, F \cap F'')$, there is no dummy cell backlogged before c; in addition, there is at the most one v-frame of data cells backlogged before c. Therefore

$$\begin{aligned}
& L_f^{(k)} \le t + P^{(k)} + \tau^{(k)} \\
= \quad & L_f^{(0)} + D_F^{(1\sim k-1)} + 3P^{(k)} + 2\tau^{(k)}. \qquad (24)
\end{aligned}$$

Eq. (24) means c is backlogged in $Q(I_k, O_k, F \cap F'')$ for at the most $D_F^{(1\sim k-1)} + 3P^{(k)} + 2\tau^{(k)} \le (k+3)P_{max} + 2\tau_{max}$ seconds. $\blacksquare$